MMDetection

Release 2.19.0

MMDetection Authors

Nov 30, 2021

GET STARTED

CHAPTER

ONE

PREREQUISITES

- Linux or macOS (Windows is in experimental support)
- Python 3.6+
- PyTorch 1.3+
- CUDA 9.2+ (If you build PyTorch from source, CUDA 9.0 is also compatible)
- GCC 5+
- MMCV

Compatible MMDetection and MMCV versions are shown as below. Please install the correct version of MMCV to avoid installation issues.

Note: You need to run `pip uninstall mmcv` first if you have mmcv installed. If mmcv and mmcv-full are both installed, there will be `ModuleNotFoundError`.

CHAPTER

TWO

INSTALLATION

2.1 Prepare environment

1. Create a conda virtual environment and activate it.

```
conda create -n openmmlab python=3.7 -y
conda activate openmmlab
```

2. Install PyTorch and torchvision following the official instructions, e.g.,

```
conda install pytorch torchvision -c pytorch
```

Note: Make sure that your compilation CUDA version and runtime CUDA version match. You can check the supported CUDA version for precompiled packages on the PyTorch website.

`E.g.1` If you have CUDA 10.1 installed under `/usr/local/cuda` and would like to install PyTorch 1.5, you need to install the prebuilt PyTorch with CUDA 10.1.

```
conda install pytorch cudatoolkit=10.1 torchvision -c pytorch
```

`E.g. 2` If you have CUDA 9.2 installed under `/usr/local/cuda` and would like to install PyTorch 1.3.1., you need to install the prebuilt PyTorch with CUDA 9.2.

```
conda install pytorch=1.3.1 cudatoolkit=9.2 torchvision=0.4.2 -c pytorch
```

If you build PyTorch from source instead of installing the prebuilt package, you can use more CUDA versions such as 9.0.

2.2 Install MMDetection

It is recommended to install MMDetection with MIM, which automatically handle the dependencies of OpenMMLab projects, including mmcv and other python packages.

```
pip install openmim
mim install mmdet
```

Or you can still install MMDetection manually:

1. Install mmcv-full.

```
pip install mmcv-full -f https://download.openmmlab.com/mmcv/dist/{cu_version}/
↪{torch_version}/index.html
```

(continues on next page)

(continued from previous page)

Please replace `{cu_version}` and `{torch_version}` in the url to your desired one. For example, to install the latest `mmcv-full` with `CUDA 11.0` and `PyTorch 1.7.0`, use the following command:

```
pip install mmcv-full -f https://download.openmmlab.com/mmcv/dist/cu110/torch1.7.0/
↪index.html
```

See here for different versions of MMCV compatible to different PyTorch and CUDA versions.

Optionally you can compile mmcv from source if you need to develop both mmcv and mmdet. Refer to the guide for details.

2. Install MMDetection.

   You can simply install mmdetection with the following command:

```
pip install mmdet
```

   or clone the repository and then install it:

```
git clone https://github.com/open-mmlab/mmdetection.git
cd mmdetection
pip install -r requirements/build.txt
pip install -v -e .  # or "python setup.py develop"
```

3. Install extra dependencies for Instaboost, Panoptic Segmentation, LVIS dataset, or Albumentations.

```
# for instaboost
pip install instaboostfast
# for panoptic segmentation
pip install git+https://github.com/cocodataset/panopticapi.git
# for LVIS dataset
pip install git+https://github.com/lvis-dataset/lvis-api.git
# for albumentations
pip install albumentations>=0.3.2 --no-binary imgaug,albumentations
```

Note:

a. When specifying `-e` or `develop`, MMDetection is installed on dev mode , any local modifications made to the code will take effect without reinstallation.

b. If you would like to use `opencv-python-headless` instead of `opencv-python`, you can install it before installing MMCV.

c. Some dependencies are optional. Simply running `pip install -v -e .` will only install the minimum runtime requirements. To use optional dependencies like `albumentations` and `imagecorruptions` either install them manually with `pip install -r requirements/optional.txt` or specify desired extras when calling `pip` (e.g. `pip install -v -e .[optional]`). Valid keys for the extras field are: `all`, `tests`, `build`, and `optional`.

d. If you would like to use `albumentations`, we suggest using `pip install albumentations>=0.3.2 --no-binary imgaug,albumentations`. If you simply use `pip install albumentations>=0.3.2`, it will install `opencv-python-headless` simultaneously (even though you have already installed `opencv-python`). We should not allow `opencv-python` and `opencv-python-headless` installed at the same time, because it might cause unexpected issues. Please refer to official documentation for more details.

2.3 Install without GPU support

MMDetection can be built for CPU only environment (where CUDA isn't available).

In CPU mode you can run the demo/webcam_demo.py for example. However some functionality is gone in this mode:

- Deformable Convolution
- Modulated Deformable Convolution
- ROI pooling
- Deformable ROI pooling
- CARAFE: Content-Aware ReAssembly of FEatures
- SyncBatchNorm
- CrissCrossAttention: Criss-Cross Attention
- MaskedConv2d
- Temporal Interlace Shift
- nms_cuda
- sigmoid_focal_loss_cuda
- bbox_overlaps

If you try to run inference with a model containing above ops, an error will be raised. The following table lists affected algorithms.

Notice: MMDetection does not support training with CPU for now.

2.4 Another option: Docker Image

We provide a Dockerfile to build an image. Ensure that you are using docker version >=19.03.

```
# build an image with PyTorch 1.6, CUDA 10.1
docker build -t mmdetection docker/
```

Run it with

```
docker run --gpus all --shm-size=8g -it -v {DATA_DIR}:/mmdetection/data mmdetection
```

2.5 A from-scratch setup script

Assuming that you already have CUDA 10.1 installed, here is a full script for setting up MMDetection with conda.

```
conda create -n openmmlab python=3.7 -y
conda activate openmmlab

conda install pytorch==1.6.0 torchvision==0.7.0 cudatoolkit=10.1 -c pytorch -y

# install the latest mmcv
pip install mmcv-full -f https://download.openmmlab.com/mmcv/dist/cu101/torch1.6.0/index.
↪html
```

(continues on next page)

(continued from previous page)

```
# install mmdetection
git clone https://github.com/open-mmlab/mmdetection.git
cd mmdetection
pip install -r requirements/build.txt
pip install -v -e .
```

2.6 Developing with multiple MMDetection versions

The train and test scripts already modify the PYTHONPATH to ensure the script use the MMDetection in the current directory.

To use the default MMDetection installed in the environment rather than that you are working with, you can remove the following line in those scripts

```
PYTHONPATH="$(dirname $0)/..":$PYTHONPATH
```

CHAPTER

THREE

VERIFICATION

To verify whether MMDetection is installed correctly, we can run the following sample code to initialize a detector and inference a demo image.

```
from mmdet.apis import init_detector, inference_detector

config_file = 'configs/faster_rcnn/faster_rcnn_r50_fpn_1x_coco.py'
# download the checkpoint from model zoo and put it in `checkpoints/`
# url: https://download.openmmlab.com/mmdetection/v2.0/faster_rcnn/faster_rcnn_r50_fpn_
↪1x_coco/faster_rcnn_r50_fpn_1x_coco_20200130-047c8118.pth
checkpoint_file = 'checkpoints/faster_rcnn_r50_fpn_1x_coco_20200130-047c8118.pth'
device = 'cuda:0'
# init a detector
model = init_detector(config_file, checkpoint_file, device=device)
# inference the demo image
inference_detector(model, 'demo/demo.jpg')
```

The above code is supposed to run successfully upon you finish the installation.

CHAPTER

FOUR

BENCHMARK AND MODEL ZOO

4.1 Mirror sites

We only use aliyun to maintain the model zoo since MMDetection V2.0. The model zoo of V1.x has been deprecated.

4.2 Common settings

- All models were trained on `coco_2017_train`, and tested on the `coco_2017_val`.
- We use distributed training.
- All pytorch-style pretrained backbones on ImageNet are from PyTorch model zoo, caffe-style pretrained backbones are converted from the newly released model from detectron2.
- For fair comparison with other codebases, we report the GPU memory as the maximum value of `torch.cuda.max_memory_allocated()` for all 8 GPUs. Note that this value is usually less than what `nvidia-smi` shows.
- We report the inference time as the total time of network forwarding and post-processing, excluding the data loading time. Results are obtained with the script benchmark.py which computes the average time on 2000 images.

4.3 ImageNet Pretrained Models

It is common to initialize from backbone models pre-trained on ImageNet classification task. All pre-trained model links can be found at open_mmlab. According to `img_norm_cfg` and source of weight, we can divide all the ImageNet pre-trained model weights into some cases:

- TorchVision: Corresponding to torchvision weight, including ResNet50, ResNet101. The `img_norm_cfg` is `dict(mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)`.
- Pycls: Corresponding to pycls weight, including RegNetX. The `img_norm_cfg` is `dict( mean=[103.530, 116.280, 123.675], std=[57.375, 57.12, 58.395], to_rgb=False)`.
- MSRA styles: Corresponding to MSRA weights, including ResNet50_Caffe and ResNet101_Caffe. The `img_norm_cfg` is `dict( mean=[103.530, 116.280, 123.675], std=[1.0, 1.0, 1.0], to_rgb=False)`.
- Caffe2 styles: Currently only contains ResNext101_32x8d. The `img_norm_cfg` is `dict(mean=[103.530, 116.280, 123.675], std=[57.375, 57.120, 58.395], to_rgb=False)`.

- Other styles: E.g SSD which corresponds to `img_norm_cfg` is `dict(mean=[123.675, 116.28, 103.53], std=[1, 1, 1], to_rgb=True)` and YOLOv3 which corresponds to `img_norm_cfg` is `dict(mean=[0, 0, 0], std=[255., 255., 255.], to_rgb=True)`.

The detailed table of the commonly used backbone models in MMDetection is listed below :

4.4 Baselines

4.4.1 RPN

Please refer to RPN for details.

4.4.2 Faster R-CNN

Please refer to Faster R-CNN for details.

4.4.3 Mask R-CNN

Please refer to Mask R-CNN for details.

4.4.4 Fast R-CNN (with pre-computed proposals)

Please refer to Fast R-CNN for details.

4.4.5 RetinaNet

Please refer to RetinaNet for details.

4.4.6 Cascade R-CNN and Cascade Mask R-CNN

Please refer to Cascade R-CNN for details.

4.4.7 Hybrid Task Cascade (HTC)

Please refer to HTC for details.

4.4.8 SSD

Please refer to SSD for details.

4.4.9 Group Normalization (GN)

Please refer to Group Normalization for details.

4.4.10 Weight Standardization

Please refer to Weight Standardization for details.

4.4.11 Deformable Convolution v2

Please refer to Deformable Convolutional Networks for details.

4.4.12 CARAFE: Content-Aware ReAssembly of FEatures

Please refer to CARAFE for details.

4.4.13 Instaboost

Please refer to Instaboost for details.

4.4.14 Libra R-CNN

Please refer to Libra R-CNN for details.

4.4.15 Guided Anchoring

Please refer to Guided Anchoring for details.

4.4.16 FCOS

Please refer to FCOS for details.

4.4.17 FoveaBox

Please refer to FoveaBox for details.

4.4.18 RepPoints

Please refer to RepPoints for details.

4.4.19 FreeAnchor

Please refer to FreeAnchor for details.

4.4.20 Grid R-CNN (plus)

Please refer to Grid R-CNN for details.

4.4.21 GHM

Please refer to GHM for details.

4.4.22 GCNet

Please refer to GCNet for details.

4.4.23 HRNet

Please refer to HRNet for details.

4.4.24 Mask Scoring R-CNN

Please refer to Mask Scoring R-CNN for details.

4.4.25 Train from Scratch

Please refer to Rethinking ImageNet Pre-training for details.

4.4.26 NAS-FPN

Please refer to NAS-FPN for details.

4.4.27 ATSS

Please refer to ATSS for details.

4.4.28 FSAF

Please refer to FSAF for details.

4.4.29 RegNetX

Please refer to RegNet for details.

4.4.30 Res2Net

Please refer to Res2Net for details.

4.4.31 GRoIE

Please refer to GRoIE for details.

4.4.32 Dynamic R-CNN

Please refer to Dynamic R-CNN for details.

4.4.33 PointRend

Please refer to PointRend for details.

4.4.34 DetectoRS

Please refer to DetectoRS for details.

4.4.35 Generalized Focal Loss

Please refer to Generalized Focal Loss for details.

4.4.36 CornerNet

Please refer to CornerNet for details.

4.4.37 YOLOv3

Please refer to YOLOv3 for details.

4.4.38 PAA

Please refer to PAA for details.

4.4.39 SABL

Please refer to SABL for details.

4.4.40 CentripetalNet

Please refer to CentripetalNet for details.

4.4.41 ResNeSt

Please refer to ResNeSt for details.

4.4.42 DETR

Please refer to DETR for details.

4.4.43 Deformable DETR

Please refer to Deformable DETR for details.

4.4.44 AutoAssign

Please refer to AutoAssign for details.

4.4.45 YOLOF

Please refer to YOLOF for details.

4.4.46 Seesaw Loss

Please refer to Seesaw Loss for details.

4.4.47 CenterNet

Please refer to CenterNet for details.

4.4.48 YOLOX

Please refer to YOLOX for details.

4.4.49 PVT

Please refer to PVT for details.

4.4.50 SOLO

Please refer to SOLO for details.

4.4.51 QueryInst

Please refer to QueryInst for details.

4.4.52 Other datasets

We also benchmark some methods on PASCAL VOC, Cityscapes and WIDER FACE.

4.4.53 Pre-trained Models

We also train Faster R-CNN and Mask R-CNN using ResNet-50 and RegNetX-3.2G with multi-scale training and longer schedules. These models serve as strong pre-trained models for downstream tasks for convenience.

4.5 Speed benchmark

4.5.1 Training Speed benchmark

We provide analyze_logs.py to get average time of iteration in training. You can find examples in Log Analysis.

We compare the training speed of Mask R-CNN with some other popular frameworks (The data is copied from detectron2). For mmdetection, we benchmark with mask_rcnn_r50_caffe_fpn_poly_1x_coco_v1.py, which should have the same setting with mask_rcnn_R_50_FPN_noaug_1x.yaml of detectron2. We also provide the checkpoint and training log for reference. The throughput is computed as the average throughput in iterations 100-500 to skip GPU warmup time.

4.5.2 Inference Speed Benchmark

We provide benchmark.py to benchmark the inference latency. The script benchmarkes the model with 2000 images and calculates the average time ignoring first 5 times. You can change the output log interval (defaults: 50) by setting `LOG-INTERVAL`.

```
python toools/benchmark.py ${CONFIG} ${CHECKPOINT} [--log-interval $[LOG-INTERVAL]] [--
↪fuse-conv-bn]
```

The latency of all models in our model zoo is benchmarked without setting `fuse-conv-bn`, you can get a lower latency by setting it.

4.6 Comparison with Detectron2

We compare mmdetection with Detectron2 in terms of speed and performance. We use the commit id 185c27e(30/4/2020) of detectron. For fair comparison, we install and run both frameworks on the same machine.

4.6.1 Hardware

- 8 NVIDIA Tesla V100 (32G) GPUs
- Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz

4.6.2 Software environment

- Python 3.7
- PyTorch 1.4
- CUDA 10.1
- CUDNN 7.6.03
- NCCL 2.4.08

4.6.3 Performance

4.6.4 Training Speed

The training speed is measure with s/iter. The lower, the better.

4.6.5 Inference Speed

The inference speed is measured with fps (img/s) on a single GPU, the higher, the better. To be consistent with Detectron2, we report the pure inference speed (without the time of data loading). For Mask R-CNN, we exclude the time of RLE encoding in post-processing. We also include the officially reported speed in the parentheses, which is slightly higher than the results tested on our server due to differences of hardwares.

4.6.6 Training memory

CHAPTER

FIVE

1: INFERENCE AND TRAIN WITH EXISTING MODELS AND STANDARD DATASETS

MMDetection provides hundreds of existing and existing detection models in Model Zoo), and supports multiple standard datasets, including Pascal VOC, COCO, CityScapes, LVIS, etc. This note will show how to perform common tasks on these existing models and standard datasets, including:

- Use existing models to inference on given images.
- Test existing models on standard datasets.
- Train predefined models on standard datasets.

5.1 Inference with existing models

By inference, we mean using trained models to detect objects on images. In MMDetection, a model is defined by a configuration file and existing model parameters are save in a checkpoint file.

To start with, we recommend Faster RCNN with this configuration file and this checkpoint file. It is recommended to download the checkpoint file to `checkpoints` directory.

5.1.1 High-level APIs for inference

MMDetection provide high-level Python APIs for inference on images. Here is an example of building the model and inference on given images or videos.

```
from mmdet.apis import init_detector, inference_detector
import mmcv

# Specify the path to model config and checkpoint file
config_file = 'configs/faster_rcnn/faster_rcnn_r50_fpn_1x_coco.py'
checkpoint_file = 'checkpoints/faster_rcnn_r50_fpn_1x_coco_20200130-047c8118.pth'

# build the model from a config file and a checkpoint file
model = init_detector(config_file, checkpoint_file, device='cuda:0')

# test a single image and show the results
img = 'test.jpg'  # or img = mmcv.imread(img), which will only load it once
result = inference_detector(model, img)
# visualize the results in a new window
model.show_result(img, result)
```

(continues on next page)

(continued from previous page)

```
# or save the visualization results to image files
model.show_result(img, result, out_file='result.jpg')

# test a video and show the results
video = mmcv.VideoReader('video.mp4')
for frame in video:
    result = inference_detector(model, frame)
    model.show_result(frame, result, wait_time=1)
```

A notebook demo can be found in demo/inference_demo.ipynb.

Note: `inference_detector` only supports single-image inference for now.

5.1.2 Asynchronous interface - supported for Python 3.7+

For Python 3.7+, MMDetection also supports async interfaces. By utilizing CUDA streams, it allows not to block CPU on GPU bound inference code and enables better CPU/GPU utilization for single-threaded application. Inference can be done concurrently either between different input data samples or between different models of some inference pipeline.

See `tests/async_benchmark.py` to compare the speed of synchronous and asynchronous interfaces.

```
import asyncio
import torch
from mmdet.apis import init_detector, async_inference_detector
from mmdet.utils.contextmanagers import concurrent

async def main():
    config_file = 'configs/faster_rcnn/faster_rcnn_r50_fpn_1x_coco.py'
    checkpoint_file = 'checkpoints/faster_rcnn_r50_fpn_1x_coco_20200130-047c8118.pth'
    device = 'cuda:0'
    model = init_detector(config_file, checkpoint=checkpoint_file, device=device)

    # queue is used for concurrent inference of multiple images
    streamqueue = asyncio.Queue()
    # queue size defines concurrency level
    streamqueue_size = 3

    for _ in range(streamqueue_size):
        streamqueue.put_nowait(torch.cuda.Stream(device=device))

    # test a single image and show the results
    img = 'test.jpg'  # or img = mmcv.imread(img), which will only load it once

    async with concurrent(streamqueue):
        result = await async_inference_detector(model, img)

    # visualize the results in a new window
    model.show_result(img, result)
    # or save the visualization results to image files
    model.show_result(img, result, out_file='result.jpg')
```

(continues on next page)

(continued from previous page)

```
asyncio.run(main())
```

5.1.3 Demos

We also provide three demo scripts, implemented with high-level APIs and supporting functionality codes. Source codes are available here.

Image demo

This script performs inference on a single image.

```
python demo/image_demo.py \
    ${IMAGE_FILE} \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    [--device ${GPU_ID}] \
    [--score-thr ${SCORE_THR}]
```

Examples:

```
python demo/image_demo.py demo/demo.jpg \
    configs/faster_rcnn/faster_rcnn_r50_fpn_1x_coco.py \
    checkpoints/faster_rcnn_r50_fpn_1x_coco_20200130-047c8118.pth \
    --device cpu
```

Webcam demo

This is a live demo from a webcam.

```
python demo/webcam_demo.py \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    [--device ${GPU_ID}] \
    [--camera-id ${CAMERA-ID}] \
    [--score-thr ${SCORE_THR}]
```

Examples:

```
python demo/webcam_demo.py \
    configs/faster_rcnn/faster_rcnn_r50_fpn_1x_coco.py \
    checkpoints/faster_rcnn_r50_fpn_1x_coco_20200130-047c8118.pth
```

Video demo

This script performs inference on a video.

```
python demo/video_demo.py \
    ${VIDEO_FILE} \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    [--device ${GPU_ID}] \
    [--score-thr ${SCORE_THR}] \
    [--out ${OUT_FILE}] \
    [--show] \
    [--wait-time ${WAIT_TIME}]
```

Examples:

```
python demo/video_demo.py demo/demo.mp4 \
    configs/faster_rcnn/faster_rcnn_r50_fpn_1x_coco.py \
    checkpoints/faster_rcnn_r50_fpn_1x_coco_20200130-047c8118.pth \
    --out result.mp4
```

5.2 Test existing models on standard datasets

To evaluate a model's accuracy, one usually tests the model on some standard datasets. MMDetection supports multiple public datasets including COCO, Pascal VOC, CityScapes, and more. This section will show how to test existing models on supported datasets.

5.2.1 Prepare datasets

Public datasets like Pascal VOC or mirror and COCO are available from official websites or mirrors. Note: In the detection task, Pascal VOC 2012 is an extension of Pascal VOC 2007 without overlap, and we usually use them together. It is recommended to download and extract the dataset somewhere outside the project directory and symlink the dataset root to `$MMDETECTION/data` as below. If your folder structure is different, you may need to change the corresponding paths in config files.

```
mmdetection
├── mmdet
├── tools
├── configs
├── data
│   ├── coco
│   │   ├── annotations
│   │   ├── train2017
│   │   ├── val2017
│   │   ├── test2017
│   ├── cityscapes
│   │   ├── annotations
│   │   ├── leftImg8bit
│   │   │   ├── train
│   │   │   ├── val
│   │   ├── gtFine
```

(continues on next page)

(continued from previous page)

```
│   │   │   ├── train
│   │   │   ├── val
│   │   ├── VOCdevkit
│   │   │   ├── VOC2007
│   │   │   ├── VOC2012
```

Some models require additional COCO-stuff datasets, such as HTC, DetectoRS and SCNet, you can download and unzip then move to the coco folder. The directory should be like this.

```
mmdetection
├── data
│   ├── coco
│   │   ├── annotations
│   │   ├── train2017
│   │   ├── val2017
│   │   ├── test2017
│   │   ├── stuffthingmaps
```

Panoptic segmentation models like PanopticFPN require additional COCO Panoptic datasets, you can download and unzip then move to the coco annotation folder. The directory should be like this.

```
mmdetection
├── data
│   ├── coco
│   │   ├── annotations
│   │   │   ├── panoptic_train2017.json
│   │   │   ├── panoptic_train2017
│   │   │   ├── panoptic_val2017.json
│   │   │   ├── panoptic_val2017
│   │   ├── train2017
│   │   ├── val2017
│   │   ├── test2017
```

The cityscapes annotations need to be converted into the coco format using `tools/dataset_converters/cityscapes.py`:

```
pip install cityscapesscripts

python tools/dataset_converters/cityscapes.py \
    ./data/cityscapes \
    --nproc 8 \
    --out-dir ./data/cityscapes/annotations
```

TODO: CHANGE TO THE NEW PATH

5.2.2 Test existing models

We provide testing scripts for evaluating an existing model on the whole dataset (COCO, PASCAL VOC, Cityscapes, etc.). The following testing environments are supported:

- single GPU
- single node multiple GPUs
- multiple nodes

Choose the proper script to perform testing depending on the testing environment.

```
# single-gpu testing
python tools/test.py \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    [--out ${RESULT_FILE}] \
    [--eval ${EVAL_METRICS}] \
    [--show]

# multi-gpu testing
bash tools/dist_test.sh \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    ${GPU_NUM} \
    [--out ${RESULT_FILE}] \
    [--eval ${EVAL_METRICS}]
```

`tools/dist_test.sh` also supports multi-node testing, but relies on PyTorch's launch utility.

Optional arguments:

- `RESULT_FILE`: Filename of the output results in pickle format. If not specified, the results will not be saved to a file.
- `EVAL_METRICS`: Items to be evaluated on the results. Allowed values depend on the dataset, e.g., `proposal_fast`, `proposal`, `bbox`, `segm` are available for COCO, `mAP`, `recall` for PASCAL VOC. Cityscapes could be evaluated by `cityscapes` as well as all COCO metrics.
- `--show`: If specified, detection results will be plotted on the images and shown in a new window. It is only applicable to single GPU testing and used for debugging and visualization. Please make sure that GUI is available in your environment. Otherwise, you may encounter an error like `cannot connect to X server`.
- `--show-dir`: If specified, detection results will be plotted on the images and saved to the specified directory. It is only applicable to single GPU testing and used for debugging and visualization. You do NOT need a GUI available in your environment for using this option.
- `--show-score-thr`: If specified, detections with scores below this threshold will be removed.
- `--cfg-options`: if specified, the key-value pair optional cfg will be merged into config file
- `--eval-options`: if specified, the key-value pair optional eval cfg will be kwargs for dataset.evaluate() function, it's only for evaluation

5.2.3 Examples

Assuming that you have already downloaded the checkpoints to the directory `checkpoints/`.

1. Test Faster R-CNN and visualize the results. Press any key for the next image. Config and checkpoint files are available here.

```
python tools/test.py \
    configs/faster_rcnn/faster_rcnn_r50_fpn_1x_coco.py \
    checkpoints/faster_rcnn_r50_fpn_1x_coco_20200130-047c8118.pth \
    --show
```

2. Test Faster R-CNN and save the painted images for future visualization. Config and checkpoint files are available here.

```
python tools/test.py \
    configs/faster_rcnn/faster_rcnn_r50_fpn_1x_coco.py \
    checkpoints/faster_rcnn_r50_fpn_1x_coco_20200130-047c8118.pth \
    --show-dir faster_rcnn_r50_fpn_1x_results
```

3. Test Faster R-CNN on PASCAL VOC (without saving the test results) and evaluate the mAP. Config and checkpoint files are available here.

```
python tools/test.py \
    configs/pascal_voc/faster_rcnn_r50_fpn_1x_voc.py \
    checkpoints/faster_rcnn_r50_fpn_1x_voc0712_20200624-c9895d40.pth \
    --eval mAP
```

4. Test Mask R-CNN with 8 GPUs, and evaluate the bbox and mask AP. Config and checkpoint files are available here.

```
./tools/dist_test.sh \
    configs/mask_rcnn_r50_fpn_1x_coco.py \
    checkpoints/mask_rcnn_r50_fpn_1x_coco_20200205-d4b0c5d6.pth \
    8 \
    --out results.pkl \
    --eval bbox segm
```

5. Test Mask R-CNN with 8 GPUs, and evaluate the **classwise** bbox and mask AP. Config and checkpoint files are available here.

```
./tools/dist_test.sh \
    configs/mask_rcnn/mask_rcnn_r50_fpn_1x_coco.py \
    checkpoints/mask_rcnn_r50_fpn_1x_coco_20200205-d4b0c5d6.pth \
    8 \
    --out results.pkl \
    --eval bbox segm \
    --options "classwise=True"
```

6. Test Mask R-CNN on COCO test-dev with 8 GPUs, and generate JSON files for submitting to the official evaluation server. Config and checkpoint files are available here.

```
./tools/dist_test.sh \
    configs/mask_rcnn/mask_rcnn_r50_fpn_1x_coco.py \
    checkpoints/mask_rcnn_r50_fpn_1x_coco_20200205-d4b0c5d6.pth \
```

(continues on next page)

(continued from previous page)

```
    8 \
    --format-only \
    --options "jsonfile_prefix=./mask_rcnn_test-dev_results"
```

This command generates two JSON files `mask_rcnn_test-dev_results.bbox.json` and `mask_rcnn_test-dev_results.segm.json`.

7. Test Mask R-CNN on Cityscapes test with 8 GPUs, and generate txt and png files for submitting to the official evaluation server. Config and checkpoint files are available here.

```
./tools/dist_test.sh \
    configs/cityscapes/mask_rcnn_r50_fpn_1x_cityscapes.py \
    checkpoints/mask_rcnn_r50_fpn_1x_cityscapes_20200227-afe51d5a.pth \
    8 \
    --format-only \
    --options "txtfile_prefix=./mask_rcnn_cityscapes_test_results"
```

The generated png and txt would be under `./mask_rcnn_cityscapes_test_results` directory.

5.2.4 Test without Ground Truth Annotations

MMDetection supports to test models without ground-truth annotations using `CocoDataset`. If your dataset format is not in COCO format, please convert them to COCO format. For example, if your dataset format is VOC, you can directly convert it to COCO format by the script in tools. If your dataset format is Cityscapes, you can directly convert it to COCO format by the script in tools. The rest of the formats can be converted using this script.

```
python tools/dataset_converters/images2coco.py \
    ${IMG_PATH} \
    ${CLASSES} \
    ${OUT} \
    [--exclude-extensions]
```

arguments

- `IMG_PATH`: The root path of images.
- `CLASSES`: The text file with a list of categories.
- `OUT`: The output annotation json file name. The save dir is in the same directory as `IMG_PATH`.
- `exclude-extensions`: The suffix of images to be excluded, such as 'png' and 'bmp'.

After the conversion is complete, you can use the following command to test

```
# single-gpu testing
python tools/test.py \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    --format-only \
    --options ${JSONFILE_PREFIX} \
    [--show]

# multi-gpu testing
bash tools/dist_test.sh \
    ${CONFIG_FILE} \
```

(continues on next page)

(continued from previous page)

```
    ${CHECKPOINT_FILE} \
    ${GPU_NUM} \
    --format-only \
    --options ${JSONFILE_PREFIX} \
    [--show]
```

Assuming that the checkpoints in the model zoo have been downloaded to the directory `checkpoints/`, we can test Mask R-CNN on COCO test-dev with 8 GPUs, and generate JSON files using the following command.

```
./tools/dist_test.sh \
    configs/mask_rcnn/mask_rcnn_r50_fpn_1x_coco.py \
    checkpoints/mask_rcnn_r50_fpn_1x_coco_20200205-d4b0c5d6.pth \
    8 \
    -format-only \
    --options "jsonfile_prefix=./mask_rcnn_test-dev_results"
```

This command generates two JSON files `mask_rcnn_test-dev_results.bbox.json` and `mask_rcnn_test-dev_results.segm.json`.

5.2.5 Batch Inference

MMDetection supports inference with a single image or batched images in test mode. By default, we use single-image inference and you can use batch inference by modifying `samples_per_gpu` in the config of test data. You can do that either by modifying the config as below.

```
data = dict(train=dict(...), val=dict(...), test=dict(samples_per_gpu=2, ...))
```

Or you can set it through `--cfg-options` as `--cfg-options data.test.samples_per_gpu=2`

5.2.6 Deprecated ImageToTensor

In test mode, `ImageToTensor` pipeline is deprecated, it's replaced by `DefaultFormatBundle` that recommended to manually replace it in the test data pipeline in your config file. examples:

```
# use ImageToTensor (deprecated)
pipelines = [
   dict(type='LoadImageFromFile'),
   dict(
       type='MultiScaleFlipAug',
       img_scale=(1333, 800),
       flip=False,
       transforms=[
           dict(type='Resize', keep_ratio=True),
           dict(type='RandomFlip'),
           dict(type='Normalize', mean=[0, 0, 0], std=[1, 1, 1]),
           dict(type='Pad', size_divisor=32),
           dict(type='ImageToTensor', keys=['img']),
           dict(type='Collect', keys=['img']),
       ])
   ]
```

(continues on next page)

(continued from previous page)

```
# manually replace ImageToTensor to DefaultFormatBundle (recommended)
pipelines = [
   dict(type='LoadImageFromFile'),
   dict(
       type='MultiScaleFlipAug',
       img_scale=(1333, 800),
       flip=False,
       transforms=[
           dict(type='Resize', keep_ratio=True),
           dict(type='RandomFlip'),
           dict(type='Normalize', mean=[0, 0, 0], std=[1, 1, 1]),
           dict(type='Pad', size_divisor=32),
           dict(type='DefaultFormatBundle'),
           dict(type='Collect', keys=['img']),
       ])
   ]
```

5.3 Train predefined models on standard datasets

MMDetection also provides out-of-the-box tools for training detection models. This section will show how to train *predefined* models (under configs) on standard datasets i.e. COCO.

Important: The default learning rate in config files is for 8 GPUs and 2 img/gpu (batch size = 8*2 = 16). According to the linear scaling rule, you need to set the learning rate proportional to the batch size if you use different GPUs or images per GPU, e.g., `lr=0.01` for 4 GPUs * 2 imgs/gpu and `lr=0.08` for 16 GPUs * 4 imgs/gpu.

5.3.1 Prepare datasets

Training requires preparing datasets too. See section *Prepare datasets* above for details.

Note: Currently, the config files under `configs/cityscapes` use COCO pretrained weights to initialize. You could download the existing models in advance if the network connection is unavailable or slow. Otherwise, it would cause errors at the beginning of training.

5.3.2 Training on a single GPU

We provide `tools/train.py` to launch training jobs on a single GPU. The basic usage is as follows.

```
python tools/train.py \
    ${CONFIG_FILE} \
    [optional arguments]
```

During training, log files and checkpoints will be saved to the working directory, which is specified by `work_dir` in the config file or via CLI argument `--work-dir`.

By default, the model is evaluated on the validation set every epoch, the evaluation interval can be specified in the config file as shown below.

```
# evaluate the model every 12 epoch.
evaluation = dict(interval=12)
```

This tool accepts several optional arguments, including:

- `--no-validate` (**not suggested**): Disable evaluation during training.
- `--work-dir ${WORK_DIR}`: Override the working directory.
- `--resume-from ${CHECKPOINT_FILE}`: Resume from a previous checkpoint file.
- `--options 'Key=value'`: Overrides other settings in the used config.

Note:

Difference between `resume-from` and `load-from`:

`resume-from` loads both the model weights and optimizer status, and the epoch is also inherited from the specified checkpoint. It is usually used for resuming the training process that is interrupted accidentally. `load-from` only loads the model weights and the training epoch starts from 0. It is usually used for finetuning.

5.3.3 Training on multiple GPUs

We provide `tools/dist_train.sh` to launch training on multiple GPUs. The basic usage is as follows.

```
bash ./tools/dist_train.sh \
    ${CONFIG_FILE} \
    ${GPU_NUM} \
    [optional arguments]
```

Optional arguments remain the same as stated *above*.

Launch multiple jobs simultaneously

If you would like to launch multiple jobs on a single machine, e.g., 2 jobs of 4-GPU training on a machine with 8 GPUs, you need to specify different ports (29500 by default) for each job to avoid communication conflict.

If you use `dist_train.sh` to launch training jobs, you can set the port in commands.

```
CUDA_VISIBLE_DEVICES=0,1,2,3 PORT=29500 ./tools/dist_train.sh ${CONFIG_FILE} 4
CUDA_VISIBLE_DEVICES=4,5,6,7 PORT=29501 ./tools/dist_train.sh ${CONFIG_FILE} 4
```

5.3.4 Training on multiple nodes

MMDetection relies on `torch.distributed` package for distributed training. Thus, as a basic usage, one can launch distributed training via PyTorch's launch utility.

5.3.5 Manage jobs with Slurm

Slurm is a good job scheduling system for computing clusters. On a cluster managed by Slurm, you can use `slurm_train.sh` to spawn training jobs. It supports both single-node and multi-node training.

The basic usage is as follows.

```
[GPUS=${GPUS}] ./tools/slurm_train.sh ${PARTITION} ${JOB_NAME} ${CONFIG_FILE} ${WORK_DIR}
```

Below is an example of using 16 GPUs to train Mask R-CNN on a Slurm partition named *dev*, and set the work-dir to some shared file systems.

```
GPUS=16 ./tools/slurm_train.sh dev mask_r50_1x configs/mask_rcnn_r50_fpn_1x_coco.py /nfs/
↪xxxx/mask_rcnn_r50_fpn_1x
```

You can check the source code to review full arguments and environment variables.

When using Slurm, the port option need to be set in one of the following ways:

1. Set the port through `--options`. This is more recommended since it does not change the original configs.

   ```
   CUDA_VISIBLE_DEVICES=0,1,2,3 GPUS=4 ./tools/slurm_train.sh ${PARTITION} ${JOB_NAME}␣
   ↪config1.py ${WORK_DIR} --options 'dist_params.port=29500'
   CUDA_VISIBLE_DEVICES=4,5,6,7 GPUS=4 ./tools/slurm_train.sh ${PARTITION} ${JOB_NAME}␣
   ↪config2.py ${WORK_DIR} --options 'dist_params.port=29501'
   ```

2. Modify the config files to set different communication ports.

   In `config1.py`, set

   ```
   dist_params = dict(backend='nccl', port=29500)
   ```

   In `config2.py`, set

   ```
   dist_params = dict(backend='nccl', port=29501)
   ```

   Then you can launch two jobs with `config1.py` and `config2.py`.

   ```
   CUDA_VISIBLE_DEVICES=0,1,2,3 GPUS=4 ./tools/slurm_train.sh ${PARTITION} ${JOB_NAME}␣
   ↪config1.py ${WORK_DIR}
   CUDA_VISIBLE_DEVICES=4,5,6,7 GPUS=4 ./tools/slurm_train.sh ${PARTITION} ${JOB_NAME}␣
   ↪config2.py ${WORK_DIR}
   ```

CHAPTER

SIX

2: TRAIN WITH CUSTOMIZED DATASETS

In this note, you will know how to inference, test, and train predefined models with customized datasets. We use the balloon dataset as an example to describe the whole process.

The basic steps are as below:

1. Prepare the customized dataset
2. Prepare a config
3. Train, test, inference models on the customized dataset.

6.1 Prepare the customized dataset

There are three ways to support a new dataset in MMDetection:

1. reorganize the dataset into COCO format.
2. reorganize the dataset into a middle format.
3. implement a new dataset.

Usually we recommend to use the first two methods which are usually easier than the third.

In this note, we give an example for converting the data into COCO format.

Note: MMDetection only supports evaluating mask AP of dataset in COCO format for now. So for instance segmentation task users should convert the data into coco format.

6.1.1 COCO annotation format

The necessary keys of COCO format for instance segmentation is as below, for the complete details, please refer here.

```
{
    "images": [image],
    "annotations": [annotation],
    "categories": [category]
}


image = {
    "id": int,
    "width": int,
    "height": int,
```

(continues on next page)

(continued from previous page)

```
    "file_name": str,
}

annotation = {
    "id": int,
    "image_id": int,
    "category_id": int,
    "segmentation": RLE or [polygon],
    "area": float,
    "bbox": [x,y,width,height],
    "iscrowd": 0 or 1,
}

categories = [{
    "id": int,
    "name": str,
    "supercategory": str,
}]
```

Assume we use the balloon dataset. After downloading the data, we need to implement a function to convert the annotation format into the COCO format. Then we can use implemented COCODataset to load the data and perform training and evaluation.

If you take a look at the dataset, you will find the dataset format is as below:

```
{'base64_img_data': '',
 'file_attributes': {},
 'filename': '34020010494_e5cb88e1c4_k.jpg',
 'fileref': '',
 'regions': {'0': {'region_attributes': {},
   'shape_attributes': {'all_points_x': [1020,
     1000,
     994,
     1003,
     1023,
     1050,
     1089,
     1134,
     1190,
     1265,
     1321,
     1361,
     1403,
     1428,
     1442,
     1445,
     1441,
     1427,
     1400,
     1361,
     1316,
     1269,
     1228,
```

(continues on next page)

(continued from previous page)

```
    1198,
    1207,
    1210,
    1190,
    1177,
    1172,
    1174,
    1170,
    1153,
    1127,
    1104,
    1061,
    1032,
    1020],
   'all_points_y': [963,
    899,
    841,
    787,
    738,
    700,
    663,
    638,
    621,
    619,
    643,
    672,
    720,
    765,
    800,
    860,
    896,
    942,
    990,
    1035,
    1079,
    1112,
    1129,
    1134,
    1144,
    1153,
    1166,
    1166,
    1150,
    1136,
    1129,
    1122,
    1112,
    1084,
    1037,
    989,
    963],
   'name': 'polygon'}}},
```

(continues on next page)

(continued from previous page)

```
 'size': 1115004}
```

The annotation is a JSON file where each key indicates an image's all annotations. The code to convert the balloon dataset into coco format is as below.

```
import os.path as osp

def convert_balloon_to_coco(ann_file, out_file, image_prefix):
    data_infos = mmcv.load(ann_file)

    annotations = []
    images = []
    obj_count = 0
    for idx, v in enumerate(mmcv.track_iter_progress(data_infos.values())):
        filename = v['filename']
        img_path = osp.join(image_prefix, filename)
        height, width = mmcv.imread(img_path).shape[:2]

        images.append(dict(
            id=idx,
            file_name=filename,
            height=height,
            width=width))

        bboxes = []
        labels = []
        masks = []
        for _, obj in v['regions'].items():
            assert not obj['region_attributes']
            obj = obj['shape_attributes']
            px = obj['all_points_x']
            py = obj['all_points_y']
            poly = [(x + 0.5, y + 0.5) for x, y in zip(px, py)]
            poly = [p for x in poly for p in x]

            x_min, y_min, x_max, y_max = (
                min(px), min(py), max(px), max(py))


            data_anno = dict(
                image_id=idx,
                id=obj_count,
                category_id=0,
                bbox=[x_min, y_min, x_max - x_min, y_max - y_min],
                area=(x_max - x_min) * (y_max - y_min),
                segmentation=[poly],
                iscrowd=0)
            annotations.append(data_anno)
            obj_count += 1

    coco_format_json = dict(
        images=images,
```

(continues on next page)

(continued from previous page)

```
        annotations=annotations,
        categories=[{'id':0, 'name': 'balloon'}])
    mmcv.dump(coco_format_json, out_file)
```

Using the function above, users can successfully convert the annotation file into json format, then we can use `CocoDataset` to train and evaluate the model.

6.2 Prepare a config

The second step is to prepare a config thus the dataset could be successfully loaded. Assume that we want to use Mask R-CNN with FPN, the config to train the detector on balloon dataset is as below. Assume the config is under directory `configs/balloon/` and named as `mask_rcnn_r50_caffe_fpn_mstrain-poly_1x_balloon.py`, the config is as below.

```
# The new config inherits a base config to highlight the necessary modification
_base_ = 'mask_rcnn/mask_rcnn_r50_caffe_fpn_mstrain-poly_1x_coco.py'

# We also need to change the num_classes in head to match the dataset's annotation
model = dict(
    roi_head=dict(
        bbox_head=dict(num_classes=1),
        mask_head=dict(num_classes=1)))

# Modify dataset related settings
dataset_type = 'COCODataset'
classes = ('balloon',)
data = dict(
    train=dict(
        img_prefix='balloon/train/',
        classes=classes,
        ann_file='balloon/train/annotation_coco.json'),
    val=dict(
        img_prefix='balloon/val/',
        classes=classes,
        ann_file='balloon/val/annotation_coco.json'),
    test=dict(
        img_prefix='balloon/val/',
        classes=classes,
        ann_file='balloon/val/annotation_coco.json'))

# We can use the pre-trained Mask RCNN model to obtain higher performance
load_from = 'checkpoints/mask_rcnn_r50_caffe_fpn_mstrain-poly_3x_coco_bbox_mAP-0.408__
↪segm_mAP-0.37_20200504_163245-42aa3d00.pth'
```

6.3 Train a new model

To train a model with the new config, you can simply run

```
python tools/train.py configs/balloon/mask_rcnn_r50_caffe_fpn_mstrain-poly_1x_balloon.py
```

For more detailed usages, please refer to the *Case 1*.

6.4 Test and inference

To test the trained model, you can simply run

```
python tools/test.py configs/balloon/mask_rcnn_r50_caffe_fpn_mstrain-poly_1x_balloon.py␣
↪work_dirs/mask_rcnn_r50_caffe_fpn_mstrain-poly_1x_balloon/latest.pth --eval bbox segm
```

For more detailed usages, please refer to the *Case 1*.

CHAPTER

SEVEN

3: TRAIN WITH CUSTOMIZED MODELS AND STANDARD DATASETS

In this note, you will know how to train, test and inference your own customized models under standard datasets. We use the cityscapes dataset to train a customized Cascade Mask R-CNN R50 model as an example to demonstrate the whole process, which using `AugFPN` to replace the default `FPN` as neck, and add `Rotate` or `Translate` as training-time auto augmentation.

The basic steps are as below:

1. Prepare the standard dataset
2. Prepare your own customized model
3. Prepare a config
4. Train, test, and inference models on the standard dataset.

7.1 Prepare the standard dataset

In this note, as we use the standard cityscapes dataset as an example.

It is recommended to symlink the dataset root to `$MMDETECTION/data`. If your folder structure is different, you may need to change the corresponding paths in config files.

```
mmdetection
├── mmdet
├── tools
├── configs
├── data
│   ├── coco
│   │   ├── annotations
│   │   ├── train2017
│   │   ├── val2017
│   │   ├── test2017
│   ├── cityscapes
│   │   ├── annotations
│   │   ├── leftImg8bit
│   │   │   ├── train
│   │   │   ├── val
│   │   ├── gtFine
│   │   │   ├── train
│   │   │   ├── val
│   ├── VOCdevkit
```

(continues on next page)

(continued from previous page)

```
│   │   ├── VOC2007
│   │   ├── VOC2012
```

The cityscapes annotations have to be converted into the coco format using `tools/dataset_converters/cityscapes.py`:

```
pip install cityscapesscripts
python tools/dataset_converters/cityscapes.py ./data/cityscapes --nproc 8 --out-dir ./
↪data/cityscapes/annotations
```

Currently the config files in `cityscapes` use COCO pre-trained weights to initialize. You could download the pre-trained models in advance if network is unavailable or slow, otherwise it would cause errors at the beginning of training.

7.2 Prepare your own customized model

The second step is to use your own module or training setting. Assume that we want to implement a new neck called `AugFPN` to replace with the default `FPN` under the existing detector Cascade Mask R-CNN R50. The following implements`AugFPN` under MMDetection.

7.2.1 1. Define a new neck (e.g. AugFPN)

Firstly create a new file `mmdet/models/necks/augfpn.py`.

```
from ..builder import NECKS

@NECKS.register_module()
class AugFPN(nn.Module):

    def __init__(self,
                 in_channels,
                 out_channels,
                 num_outs,
                 start_level=0,
                 end_level=-1,
                 add_extra_convs=False):
        pass

    def forward(self, inputs):
        # implementation is ignored
        pass
```

7.2.2 2. Import the module

You can either add the following line to `mmdet/models/necks/__init__.py`,

```
from .augfpn import AugFPN
```

or alternatively add

```
custom_imports = dict(
    imports=['mmdet.models.necks.augfpn.py'],
    allow_failed_imports=False)
```

to the config file and avoid modifying the original code.

7.2.3 3. Modify the config file

```
neck=dict(
    type='AugFPN',
    in_channels=[256, 512, 1024, 2048],
    out_channels=256,
    num_outs=5)
```

For more detailed usages about customize your own models (e.g. implement a new backbone, head, loss, etc) and runtime training settings (e.g. define a new optimizer, use gradient clip, customize training schedules and hooks, etc), please refer to the guideline *Customize Models* and *Customize Runtime Settings* respectively.

7.3 Prepare a config

The third step is to prepare a config for your own training setting. Assume that we want to add `AugFPN` and `Rotate` or `Translate` augmentation to existing Cascade Mask R-CNN R50 to train the cityscapes dataset, and assume the config is under directory `configs/cityscapes/` and named as `cascade_mask_rcnn_r50_augfpn_autoaug_10e_cityscapes.py`, the config is as below.

```
# The new config inherits the base configs to highlight the necessary modification
_base_ = [
    '../_base_/models/cascade_mask_rcnn_r50_fpn.py',
    '../_base_/datasets/cityscapes_instance.py', '../_base_/default_runtime.py'
]

model = dict(
    # set None to avoid loading ImageNet pretrained backbone,
    # instead here we set `load_from` to load from COCO pretrained detectors.
    backbone=dict(init_cfg=None),
    # replace neck from defaultly `FPN` to our new implemented module `AugFPN`
    neck=dict(
        type='AugFPN',
        in_channels=[256, 512, 1024, 2048],
        out_channels=256,
        num_outs=5),
    # We also need to change the num_classes in head from 80 to 8, to match the
    # cityscapes dataset's annotation. This modification involves `bbox_head` and `mask_
↪head`.
```

(continues on next page)

(continued from previous page)

```
    roi_head=dict(
        bbox_head=[
            dict(
                type='Shared2FCBBoxHead',
                in_channels=256,
                fc_out_channels=1024,
                roi_feat_size=7,
                # change the number of classes from defaultly COCO to cityscapes
                num_classes=8,
                bbox_coder=dict(
                    type='DeltaXYWHBBoxCoder',
                    target_means=[0., 0., 0., 0.],
                    target_stds=[0.1, 0.1, 0.2, 0.2]),
                reg_class_agnostic=True,
                loss_cls=dict(
                    type='CrossEntropyLoss',
                    use_sigmoid=False,
                    loss_weight=1.0),
                loss_bbox=dict(type='SmoothL1Loss', beta=1.0,
                               loss_weight=1.0)),
            dict(
                type='Shared2FCBBoxHead',
                in_channels=256,
                fc_out_channels=1024,
                roi_feat_size=7,
                # change the number of classes from defaultly COCO to cityscapes
                num_classes=8,
                bbox_coder=dict(
                    type='DeltaXYWHBBoxCoder',
                    target_means=[0., 0., 0., 0.],
                    target_stds=[0.05, 0.05, 0.1, 0.1]),
                reg_class_agnostic=True,
                loss_cls=dict(
                    type='CrossEntropyLoss',
                    use_sigmoid=False,
                    loss_weight=1.0),
                loss_bbox=dict(type='SmoothL1Loss', beta=1.0,
                               loss_weight=1.0)),
            dict(
                type='Shared2FCBBoxHead',
                in_channels=256,
                fc_out_channels=1024,
                roi_feat_size=7,
                # change the number of classes from defaultly COCO to cityscapes
                num_classes=8,
                bbox_coder=dict(
                    type='DeltaXYWHBBoxCoder',
                    target_means=[0., 0., 0., 0.],
                    target_stds=[0.033, 0.033, 0.067, 0.067]),
                reg_class_agnostic=True,
                loss_cls=dict(
                    type='CrossEntropyLoss',
```

(continues on next page)

(continued from previous page)

```
                    use_sigmoid=False,
                    loss_weight=1.0),
                loss_bbox=dict(type='SmoothL1Loss', beta=1.0, loss_weight=1.0))
        ],
        mask_head=dict(
            type='FCNMaskHead',
            num_convs=4,
            in_channels=256,
            conv_out_channels=256,
            # change the number of classes from defaultly COCO to cityscapes
            num_classes=8,
            loss_mask=dict(
                type='CrossEntropyLoss', use_mask=True, loss_weight=1.0))))

# over-write `train_pipeline` for new added `AutoAugment` training setting
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='LoadAnnotations', with_bbox=True, with_mask=True),
    dict(
        type='AutoAugment',
        policies=[
            [dict(
                 type='Rotate',
                 level=5,
                 img_fill_val=(124, 116, 104),
                 prob=0.5,
                 scale=1)
            ],
            [dict(type='Rotate', level=7, img_fill_val=(124, 116, 104)),
             dict(
                 type='Translate',
                 level=5,
                 prob=0.5,
                 img_fill_val=(124, 116, 104))
            ],
        ]),
    dict(
        type='Resize', img_scale=[(2048, 800), (2048, 1024)], keep_ratio=True),
    dict(type='RandomFlip', flip_ratio=0.5),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='Pad', size_divisor=32),
    dict(type='DefaultFormatBundle'),
    dict(type='Collect', keys=['img', 'gt_bboxes', 'gt_labels', 'gt_masks']),
]

# set batch_size per gpu, and set new training pipeline
data = dict(
    samples_per_gpu=1,
    workers_per_gpu=3,
    # over-write `pipeline` with new training pipeline setting
```

(continues on next page)

(continued from previous page)

```
    train=dict(dataset=dict(pipeline=train_pipeline)))

# Set optimizer
optimizer = dict(type='SGD', lr=0.01, momentum=0.9, weight_decay=0.0001)
optimizer_config = dict(grad_clip=None)
# Set customized learning policy
lr_config = dict(
    policy='step',
    warmup='linear',
    warmup_iters=500,
    warmup_ratio=0.001,
    step=[8])
runner = dict(type='EpochBasedRunner', max_epochs=10)

# We can use the COCO pretrained Cascade Mask R-CNN R50 model for more stable␣
↪performance initialization
load_from = 'https://download.openmmlab.com/mmdetection/v2.0/cascade_rcnn/cascade_mask_
↪rcnn_r50_fpn_1x_coco/cascade_mask_rcnn_r50_fpn_1x_coco_20200203-9d4dcb24.pth'
```

7.4 Train a new model

To train a model with the new config, you can simply run

```
python tools/train.py configs/cityscapes/cascade_mask_rcnn_r50_augfpn_autoaug_10e_
↪cityscapes.py
```

For more detailed usages, please refer to the *Case 1*.

7.5 Test and inference

To test the trained model, you can simply run

```
python tools/test.py configs/cityscapes/cascade_mask_rcnn_r50_augfpn_autoaug_10e_
↪cityscapes.py work_dirs/cascade_mask_rcnn_r50_augfpn_autoaug_10e_cityscapes.py/latest.
↪pth --eval bbox segm
```

For more detailed usages, please refer to the *Case 1*.

CHAPTER

EIGHT

TUTORIAL 1: LEARN ABOUT CONFIGS

We incorporate modular and inheritance design into our config system, which is convenient to conduct various experiments. If you wish to inspect the config file, you may run `python tools/misc/print_config.py /PATH/TO/CONFIG` to see the complete config.

8.1 Modify config through script arguments

When submitting jobs using "tools/train.py" or "tools/test.py", you may specify `--cfg-options` to in-place modify the config.

- Update config keys of dict chains.

  The config options can be specified following the order of the dict keys in the original config. For example, `--cfg-options model.backbone.norm_eval=False` changes the all BN modules in model backbones to `train` mode.

- Update keys inside a list of configs.

  Some config dicts are composed as a list in your config. For example, the training pipeline `data.train.pipeline` is normally a list e.g. `[dict(type='LoadImageFromFile'), ...]`. If you want to change `'LoadImageFromFile'` to `'LoadImageFromWebcam'` in the pipeline, you may specify `--cfg-options data.train.pipeline.0.type=LoadImageFromWebcam`.

- Update values of list/tuples.

  If the value to be updated is a list or a tuple. For example, the config file normally sets `workflow=[('train', 1)]`. If you want to change this key, you may specify `--cfg-options workflow="[(train,1),(val,1)]"`. Note that the quotation mark " is necessary to support list/tuple data types, and that **NO** white space is allowed inside the quotation marks in the specified value.

8.2 Config File Structure

There are 4 basic component types under `config/_base_`, dataset, model, schedule, default_runtime. Many methods could be easily constructed with one of each like Faster R-CNN, Mask R-CNN, Cascade R-CNN, RPN, SSD. The configs that are composed by components from `_base_` are called *primitive*.

For all configs under the same folder, it is recommended to have only **one** *primitive* config. All other configs should inherit from the *primitive* config. In this way, the maximum of inheritance level is 3.

For easy understanding, we recommend contributors to inherit from existing methods. For example, if some modification is made base on Faster R-CNN, user may first inherit the basic Faster R-CNN structure by specifying `_base_ = ../faster_rcnn/faster_rcnn_r50_fpn_1x_coco.py`, then modify the necessary fields in the config files.

If you are building an entirely new method that does not share the structure with any of the existing methods, you may create a folder `xxx_rcnn` under `configs`,

Please refer to mmcv for detailed documentation.

8.3 Config Name Style

We follow the below style to name config files. Contributors are advised to follow the same style.

```
{model}_[model setting]_{backbone}_{neck}_[norm setting]_[misc]_[gpu x batch_per_gpu]_
↪{schedule}_{dataset}
```

`{xxx}` is required field and `[yyy]` is optional.

- `{model}`: model type like `faster_rcnn`, `mask_rcnn`, etc.
- `[model setting]`: specific setting for some model, like `without_semantic` for `htc`, `moment` for `reppoints`, etc.
- `{backbone}`: backbone type like `r50` (ResNet-50), `x101` (ResNeXt-101).
- `{neck}`: neck type like `fpn`, `pafpn`, `nasfpn`, `c4`.
- `[norm_setting]`: `bn` (Batch Normalization) is used unless specified, other norm layer type could be `gn` (Group Normalization), `syncbn` (Synchronized Batch Normalization). `gn-head`/`gn-neck` indicates GN is applied in head/neck only, while `gn-all` means GN is applied in the entire model, e.g. backbone, neck, head.
- `[misc]`: miscellaneous setting/plugins of model, e.g. `dconv`, `gcb`, `attention`, `albu`, `mstrain`.
- `[gpu x batch_per_gpu]`: GPUs and samples per GPU, `8x2` is used by default.
- `{schedule}`: training schedule, options are `1x`, `2x`, `20e`, etc. `1x` and `2x` means 12 epochs and 24 epochs respectively. `20e` is adopted in cascade models, which denotes 20 epochs. For `1x`/`2x`, initial learning rate decays by a factor of 10 at the 8/16th and 11/22th epochs. For `20e`, initial learning rate decays by a factor of 10 at the 16th and 19th epochs.
- `{dataset}`: dataset like `coco`, `cityscapes`, `voc_0712`, `wider_face`.

8.4 Deprecated train_cfg/test_cfg

The `train_cfg` and `test_cfg` are deprecated in config file, please specify them in the model config. The original config structure is as below.

```
# deprecated
model = dict(
    type=...,
    ...
)
train_cfg=dict(...)
test_cfg=dict(...)
```

The migration example is as below.

```
# recommended
model = dict(
   type=...,
   ...
   train_cfg=dict(...),
   test_cfg=dict(...),
)
```

8.5 An Example of Mask R-CNN

To help the users have a basic idea of a complete config and the modules in a modern detection system, we make brief comments on the config of Mask R-CNN using ResNet50 and FPN as the following. For more detailed usage and the corresponding alternative for each modules, please refer to the API documentation.

```
model = dict(
    type='MaskRCNN',  # The name of detector
    backbone=dict(  # The config of backbone
        type='ResNet',  # The type of the backbone, refer to https://github.com/open-
↪mmlab/mmdetection/blob/master/mmdet/models/backbones/resnet.py#L308 for more details.
        depth=50,  # The depth of backbone, usually it is 50 or 101 for ResNet and␣
↪ResNext backbones.
        num_stages=4,  # Number of stages of the backbone.
        out_indices=(0, 1, 2, 3),  # The index of output feature maps produced in each␣
↪stages
        frozen_stages=1,  # The weights in the first 1 stage are fronzen
        norm_cfg=dict(  # The config of normalization layers.
            type='BN',  # Type of norm layer, usually it is BN or GN
            requires_grad=True),  # Whether to train the gamma and beta in BN
        norm_eval=True,  # Whether to freeze the statistics in BN
        style='pytorch' # The style of backbone, 'pytorch' means that stride 2 layers␣
↪are in 3x3 conv, 'caffe' means stride 2 layers are in 1x1 convs.
            init_cfg=dict(type='Pretrained', checkpoint='torchvision://resnet50')),  #␣
↪The ImageNet pretrained backbone to be loaded
    neck=dict(
        type='FPN',  # The neck of detector is FPN. We also support 'NASFPN', 'PAFPN',␣
↪etc. Refer to https://github.com/open-mmlab/mmdetection/blob/master/mmdet/models/necks/
↪fpn.py#L10 for more details.
        in_channels=[256, 512, 1024, 2048],  # The input channels, this is consistent␣
↪with the output channels of backbone
        out_channels=256,  # The output channels of each level of the pyramid feature map
        num_outs=5),  # The number of output scales
    rpn_head=dict(
        type='RPNHead',  # The type of RPN head is 'RPNHead', we also support 'GARPNHead
↪', etc. Refer to https://github.com/open-mmlab/mmdetection/blob/master/mmdet/models/
↪dense_heads/rpn_head.py#L12 for more details.
        in_channels=256,  # The input channels of each input feature map, this is␣
↪consistent with the output channels of neck
        feat_channels=256,  # Feature channels of convolutional layers in the head.
        anchor_generator=dict(  # The config of anchor generator
            type='AnchorGenerator',  # Most of methods use AnchorGenerator, SSD␣
↪Detectors uses `SSDAnchorGenerator`. Refer to https://github.com/open-mmlab/
↪mmdetection/blob/master/mmdet/core/anchor/anchor_generator.py#L10 for more details
```

(continues on next page)

(continued from previous page)

```
            scales=[8],  # Basic scale of the anchor, the area of the anchor in one␣
↪position of a feature map will be scale * base_sizes
            ratios=[0.5, 1.0, 2.0],  # The ratio between height and width.
            strides=[4, 8, 16, 32, 64]),  # The strides of the anchor generator. This is␣
↪consistent with the FPN feature strides. The strides will be taken as base_sizes if␣
↪base_sizes is not set.
        bbox_coder=dict(  # Config of box coder to encode and decode the boxes during␣
↪training and testing
            type='DeltaXYWHBBoxCoder',  # Type of box coder. 'DeltaXYWHBBoxCoder' is␣
↪applied for most of methods. Refer to https://github.com/open-mmlab/mmdetection/blob/
↪master/mmdet/core/bbox/coder/delta_xywh_bbox_coder.py#L9 for more details.
            target_means=[0.0, 0.0, 0.0, 0.0],  # The target means used to encode and␣
↪decode boxes
            target_stds=[1.0, 1.0, 1.0, 1.0]),  # The standard variance used to encode␣
↪and decode boxes
        loss_cls=dict(  # Config of loss function for the classification branch
            type='CrossEntropyLoss',  # Type of loss for classification branch, we also␣
↪support FocalLoss etc.
            use_sigmoid=True,  # RPN usually perform two-class classification, so it␣
↪usually uses sigmoid function.
            loss_weight=1.0),  # Loss weight of the classification branch.
        loss_bbox=dict(  # Config of loss function for the regression branch.
            type='L1Loss',  # Type of loss, we also support many IoU Losses and smooth␣
↪L1-loss, etc. Refer to https://github.com/open-mmlab/mmdetection/blob/master/mmdet/
↪models/losses/smooth_l1_loss.py#L56 for implementation.
            loss_weight=1.0)),  # Loss weight of the regression branch.
    roi_head=dict(  # RoIHead encapsulates the second stage of two-stage/cascade␣
↪detectors.
        type='StandardRoIHead',  # Type of the RoI head. Refer to https://github.com/
↪open-mmlab/mmdetection/blob/master/mmdet/models/roi_heads/standard_roi_head.py#L10 for␣
↪implementation.
        bbox_roi_extractor=dict(  # RoI feature extractor for bbox regression.
            type='SingleRoIExtractor',  # Type of the RoI feature extractor, most of␣
↪methods uses SingleRoIExtractor. Refer to https://github.com/open-mmlab/mmdetection/
↪blob/master/mmdet/models/roi_heads/roi_extractors/single_level.py#L10 for details.
            roi_layer=dict(  # Config of RoI Layer
                type='RoIAlign',  # Type of RoI Layer, DeformRoIPoolingPack and␣
↪ModulatedDeformRoIPoolingPack are also supported. Refer to https://github.com/open-
↪mmlab/mmdetection/blob/master/mmdet/ops/roi_align/roi_align.py#L79 for details.
                output_size=7,  # The output size of feature maps.
                sampling_ratio=0),  # Sampling ratio when extracting the RoI features. 0␣
↪means adaptive ratio.
            out_channels=256,  # output channels of the extracted feature.
            featmap_strides=[4, 8, 16, 32]),  # Strides of multi-scale feature maps. It␣
↪should be consistent to the architecture of the backbone.
        bbox_head=dict(  # Config of box head in the RoIHead.
            type='Shared2FCBBoxHead',  # Type of the bbox head, Refer to https://github.
↪com/open-mmlab/mmdetection/blob/master/mmdet/models/roi_heads/bbox_heads/convfc_bbox_
↪head.py#L177 for implementation details.
            in_channels=256,  # Input channels for bbox head. This is consistent with␣
↪the out_channels in roi_extractor
            fc_out_channels=1024,  # Output feature channels of FC layers.
```

(continues on next page)

(continued from previous page)

```
            roi_feat_size=7,  # Size of RoI features
            num_classes=80,  # Number of classes for classification
            bbox_coder=dict(  # Box coder used in the second stage.
                type='DeltaXYWHBBoxCoder',  # Type of box coder. 'DeltaXYWHBBoxCoder' is␣
↪applied for most of methods.
                target_means=[0.0, 0.0, 0.0, 0.0],  # Means used to encode and decode box
                target_stds=[0.1, 0.1, 0.2, 0.2]),  # Standard variance for encoding and␣
↪decoding. It is smaller since the boxes are more accurate. [0.1, 0.1, 0.2, 0.2] is a␣
↪conventional setting.
            reg_class_agnostic=False,  # Whether the regression is class agnostic.
            loss_cls=dict(  # Config of loss function for the classification branch
                type='CrossEntropyLoss',  # Type of loss for classification branch, we␣
↪also support FocalLoss etc.
                use_sigmoid=False,  # Whether to use sigmoid.
                loss_weight=1.0),  # Loss weight of the classification branch.
            loss_bbox=dict(  # Config of loss function for the regression branch.
                type='L1Loss',  # Type of loss, we also support many IoU Losses and␣
↪smooth L1-loss, etc.
                loss_weight=1.0)),  # Loss weight of the regression branch.
        mask_roi_extractor=dict(  # RoI feature extractor for mask generation.
            type='SingleRoIExtractor',  # Type of the RoI feature extractor, most of␣
↪methods uses SingleRoIExtractor.
            roi_layer=dict(  # Config of RoI Layer that extracts features for instance␣
↪segmentation
                type='RoIAlign',  # Type of RoI Layer, DeformRoIPoolingPack and␣
↪ModulatedDeformRoIPoolingPack are also supported
                output_size=14,  # The output size of feature maps.
                sampling_ratio=0),  # Sampling ratio when extracting the RoI features.
            out_channels=256,  # Output channels of the extracted feature.
            featmap_strides=[4, 8, 16, 32]),  # Strides of multi-scale feature maps.
        mask_head=dict(  # Mask prediction head
            type='FCNMaskHead',  # Type of mask head, refer to https://github.com/open-
↪mmlab/mmdetection/blob/master/mmdet/models/roi_heads/mask_heads/fcn_mask_head.py#L21␣
↪for implementation details.
            num_convs=4,  # Number of convolutional layers in mask head.
            in_channels=256,  # Input channels, should be consistent with the output␣
↪channels of mask roi extractor.
            conv_out_channels=256,  # Output channels of the convolutional layer.
            num_classes=80,  # Number of class to be segmented.
            loss_mask=dict(  # Config of loss function for the mask branch.
                type='CrossEntropyLoss',  # Type of loss used for segmentation
                use_mask=True,  # Whether to only train the mask in the correct class.
                loss_weight=1.0))))  # Loss weight of mask branch.
    train_cfg = dict(  # Config of training hyperparameters for rpn and rcnn
        rpn=dict(  # Training config of rpn
            assigner=dict(  # Config of assigner
                type='MaxIoUAssigner',  # Type of assigner, MaxIoUAssigner is used for␣
↪many common detectors. Refer to https://github.com/open-mmlab/mmdetection/blob/master/
↪mmdet/core/bbox/assigners/max_iou_assigner.py#L10 for more details.
                pos_iou_thr=0.7,  # IoU >= threshold 0.7 will be taken as positive␣
↪samples
                neg_iou_thr=0.3,  # IoU < threshold 0.3 will be taken as negative samples
```

(continues on next page)

(continued from previous page)

```
                min_pos_iou=0.3,  # The minimal IoU threshold to take boxes as positive␣
↪samples
                match_low_quality=True,  # Whether to match the boxes under low quality␣
↪(see API doc for more details).
                ignore_iof_thr=-1),  # IoF threshold for ignoring bboxes
            sampler=dict(  # Config of positive/negative sampler
                type='RandomSampler',  # Type of sampler, PseudoSampler and other␣
↪samplers are also supported. Refer to https://github.com/open-mmlab/mmdetection/blob/
↪master/mmdet/core/bbox/samplers/random_sampler.py#L8 for implementation details.
                num=256,  # Number of samples
                pos_fraction=0.5,  # The ratio of positive samples in the total samples.
                neg_pos_ub=-1,  # The upper bound of negative samples based on the␣
↪number of positive samples.
                add_gt_as_proposals=False),  # Whether add GT as proposals after␣
↪sampling.
            allowed_border=-1,  # The border allowed after padding for valid anchors.
            pos_weight=-1,  # The weight of positive samples during training.
            debug=False),  # Whether to set the debug mode
        rpn_proposal=dict(  # The config to generate proposals during training
            nms_across_levels=False,  # Whether to do NMS for boxes across levels. Only␣
↪work in `GARPNHead`, naive rpn does not support do nms cross levels.
            nms_pre=2000,  # The number of boxes before NMS
            nms_post=1000,  # The number of boxes to be kept by NMS, Only work in␣
↪`GARPNHead`.
            max_per_img=1000,  # The number of boxes to be kept after NMS.
            nms=dict( # Config of NMS
                type='nms',  # Type of NMS
                iou_threshold=0.7 # NMS threshold
                ),
            min_bbox_size=0),  # The allowed minimal box size
        rcnn=dict(  # The config for the roi heads.
            assigner=dict(  # Config of assigner for second stage, this is different for␣
↪that in rpn
                type='MaxIoUAssigner',  # Type of assigner, MaxIoUAssigner is used for␣
↪all roi_heads for now. Refer to https://github.com/open-mmlab/mmdetection/blob/master/
↪mmdet/core/bbox/assigners/max_iou_assigner.py#L10 for more details.
                pos_iou_thr=0.5,  # IoU >= threshold 0.5 will be taken as positive␣
↪samples
                neg_iou_thr=0.5,  # IoU < threshold 0.5 will be taken as negative samples
                min_pos_iou=0.5,  # The minimal IoU threshold to take boxes as positive␣
↪samples
                match_low_quality=False,  # Whether to match the boxes under low quality␣
↪(see API doc for more details).
                ignore_iof_thr=-1),  # IoF threshold for ignoring bboxes
            sampler=dict(
                type='RandomSampler',  # Type of sampler, PseudoSampler and other␣
↪samplers are also supported. Refer to https://github.com/open-mmlab/mmdetection/blob/
↪master/mmdet/core/bbox/samplers/random_sampler.py#L8 for implementation details.
                num=512,  # Number of samples
                pos_fraction=0.25,  # The ratio of positive samples in the total samples.
                neg_pos_ub=-1,  # The upper bound of negative samples based on the␣
↪number of positive samples.
```

(continues on next page)

(continued from previous page)

```
                add_gt_as_proposals=True
            ),  # Whether add GT as proposals after sampling.
            mask_size=28,  # Size of mask
            pos_weight=-1,  # The weight of positive samples during training.
            debug=False))  # Whether to set the debug mode
    test_cfg = dict(  # Config for testing hyperparameters for rpn and rcnn
        rpn=dict(  # The config to generate proposals during testing
            nms_across_levels=False,  # Whether to do NMS for boxes across levels. Only␣
↪work in `GARPNHead`, naive rpn does not support do nms cross levels.
            nms_pre=1000,  # The number of boxes before NMS
            nms_post=1000,  # The number of boxes to be kept by NMS, Only work in␣
↪`GARPNHead`.
            max_per_img=1000,  # The number of boxes to be kept after NMS.
            nms=dict( # Config of NMS
                type='nms',  #Type of NMS
                iou_threshold=0.7 # NMS threshold
                ),
            min_bbox_size=0),  # The allowed minimal box size
        rcnn=dict(  # The config for the roi heads.
            score_thr=0.05,  # Threshold to filter out boxes
            nms=dict(  # Config of NMS in the second stage
                type='nms',  # Type of NMS
                iou_thr=0.5),  # NMS threshold
            max_per_img=100,  # Max number of detections of each image
            mask_thr_binary=0.5))  # Threshold of mask prediction
dataset_type = 'CocoDataset'  # Dataset type, this will be used to define the dataset
data_root = 'data/coco/'  # Root path of data
img_norm_cfg = dict(  # Image normalization config to normalize the input images
    mean=[123.675, 116.28, 103.53],  # Mean values used to pre-training the pre-trained␣
↪backbone models
    std=[58.395, 57.12, 57.375],  # Standard variance used to pre-training the pre-
↪trained backbone models
    to_rgb=True
)  # The channel orders of image used to pre-training the pre-trained backbone models
train_pipeline = [  # Training pipeline
    dict(type='LoadImageFromFile'),  # First pipeline to load images from file path
    dict(
        type='LoadAnnotations',  # Second pipeline to load annotations for current image
        with_bbox=True,  # Whether to use bounding box, True for detection
        with_mask=True,  # Whether to use instance mask, True for instance segmentation
        poly2mask=False),  # Whether to convert the polygon mask to instance mask, set␣
↪False for acceleration and to save memory
    dict(
        type='Resize',  # Augmentation pipeline that resize the images and their␣
↪annotations
        img_scale=(1333, 800),  # The largest scale of image
        keep_ratio=True
    ),  # whether to keep the ratio between height and width.
    dict(
        type='RandomFlip',  # Augmentation pipeline that flip the images and their␣
↪annotations
        flip_ratio=0.5),  # The ratio or probability to flip
```

(continues on next page)

(continued from previous page)

```
    dict(
        type='Normalize',  # Augmentation pipeline that normalize the input images
        mean=[123.675, 116.28, 103.53],  # These keys are the same of img_norm_cfg since␣
↪the
        std=[58.395, 57.12, 57.375],  # keys of img_norm_cfg are used here as arguments
        to_rgb=True),
    dict(
        type='Pad',  # Padding config
        size_divisor=32),  # The number the padded images should be divisible
    dict(type='DefaultFormatBundle'),  # Default format bundle to gather data in the␣
↪pipeline
    dict(
        type='Collect',  # Pipeline that decides which keys in the data should be passed␣
↪to the detector
        keys=['img', 'gt_bboxes', 'gt_labels', 'gt_masks'])
]
test_pipeline = [
    dict(type='LoadImageFromFile'),  # First pipeline to load images from file path
    dict(
        type='MultiScaleFlipAug',  # An encapsulation that encapsulates the testing␣
↪augmentations
        img_scale=(1333, 800),  # Decides the largest scale for testing, used for the␣
↪Resize pipeline
        flip=False,  # Whether to flip images during testing
        transforms=[
            dict(type='Resize',  # Use resize augmentation
                 keep_ratio=True),  # Whether to keep the ratio between height and width,
↪ the img_scale set here will be suppressed by the img_scale set above.
            dict(type='RandomFlip'),  # Thought RandomFlip is added in pipeline, it is␣
↪not used because flip=False
            dict(
                type='Normalize',  # Normalization config, the values are from img_norm_
↪cfg
                mean=[123.675, 116.28, 103.53],
                std=[58.395, 57.12, 57.375],
                to_rgb=True),
            dict(
                type='Pad',  # Padding config to pad images divisible by 32.
                size_divisor=32),
            dict(
                type='ImageToTensor',  # convert image to tensor
                keys=['img']),
            dict(
                type='Collect',  # Collect pipeline that collect necessary keys for␣
↪testing.
                keys=['img'])
        ])
]
data = dict(
    samples_per_gpu=2,  # Batch size of a single GPU
    workers_per_gpu=2,  # Worker to pre-fetch data for each single GPU
    train=dict(  # Train dataset config
```

(continues on next page)

(continued from previous page)

```
        type='CocoDataset',  # Type of dataset, refer to https://github.com/open-mmlab/
↪mmdetection/blob/master/mmdet/datasets/coco.py#L19 for details.
        ann_file='data/coco/annotations/instances_train2017.json',  # Path of annotation␣
↪file
        img_prefix='data/coco/train2017/',  # Prefix of image path
        pipeline=[  # pipeline, this is passed by the train_pipeline created before.
            dict(type='LoadImageFromFile'),
            dict(
                type='LoadAnnotations',
                with_bbox=True,
                with_mask=True,
                poly2mask=False),
            dict(type='Resize', img_scale=(1333, 800), keep_ratio=True),
            dict(type='RandomFlip', flip_ratio=0.5),
            dict(
                type='Normalize',
                mean=[123.675, 116.28, 103.53],
                std=[58.395, 57.12, 57.375],
                to_rgb=True),
            dict(type='Pad', size_divisor=32),
            dict(type='DefaultFormatBundle'),
            dict(
                type='Collect',
                keys=['img', 'gt_bboxes', 'gt_labels', 'gt_masks'])
        ]),
    val=dict(  # Validation dataset config
        type='CocoDataset',
        ann_file='data/coco/annotations/instances_val2017.json',
        img_prefix='data/coco/val2017/',
        pipeline=[  # Pipeline is passed by test_pipeline created before
            dict(type='LoadImageFromFile'),
            dict(
                type='MultiScaleFlipAug',
                img_scale=(1333, 800),
                flip=False,
                transforms=[
                    dict(type='Resize', keep_ratio=True),
                    dict(type='RandomFlip'),
                    dict(
                        type='Normalize',
                        mean=[123.675, 116.28, 103.53],
                        std=[58.395, 57.12, 57.375],
                        to_rgb=True),
                    dict(type='Pad', size_divisor=32),
                    dict(type='ImageToTensor', keys=['img']),
                    dict(type='Collect', keys=['img'])
                ])
        ]),
    test=dict(  # Test dataset config, modify the ann_file for test-dev/test submission
        type='CocoDataset',
        ann_file='data/coco/annotations/instances_val2017.json',
        img_prefix='data/coco/val2017/',
```

(continues on next page)

(continued from previous page)

```
        pipeline=[  # Pipeline is passed by test_pipeline created before
            dict(type='LoadImageFromFile'),
            dict(
                type='MultiScaleFlipAug',
                img_scale=(1333, 800),
                flip=False,
                transforms=[
                    dict(type='Resize', keep_ratio=True),
                    dict(type='RandomFlip'),
                    dict(
                        type='Normalize',
                        mean=[123.675, 116.28, 103.53],
                        std=[58.395, 57.12, 57.375],
                        to_rgb=True),
                    dict(type='Pad', size_divisor=32),
                    dict(type='ImageToTensor', keys=['img']),
                    dict(type='Collect', keys=['img'])
                ])
        ],
        samples_per_gpu=2  # Batch size of a single GPU used in testing
        ))
evaluation = dict(  # The config to build the evaluation hook, refer to https://github.
↪com/open-mmlab/mmdetection/blob/master/mmdet/core/evaluation/eval_hooks.py#L7 for more␣
↪details.
    interval=1,  # Evaluation interval
    metric=['bbox', 'segm'])  # Metrics used during evaluation
optimizer = dict(  # Config used to build optimizer, support all the optimizers in␣
↪PyTorch whose arguments are also the same as those in PyTorch
    type='SGD',  # Type of optimizers, refer to https://github.com/open-mmlab/
↪mmdetection/blob/master/mmdet/core/optimizer/default_constructor.py#L13 for more␣
↪details
    lr=0.02,  # Learning rate of optimizers, see detail usages of the parameters in the␣
↪documentation of PyTorch
    momentum=0.9,  # Momentum
    weight_decay=0.0001)  # Weight decay of SGD
optimizer_config = dict(  # Config used to build the optimizer hook, refer to https://
↪github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/optimizer.py#L8 for␣
↪implementation details.
    grad_clip=None)  # Most of the methods do not use gradient clip
lr_config = dict(  # Learning rate scheduler config used to register LrUpdater hook
    policy='step',  # The policy of scheduler, also support CosineAnnealing, Cyclic, etc.
↪ Refer to details of supported LrUpdater from https://github.com/open-mmlab/mmcv/blob/
↪master/mmcv/runner/hooks/lr_updater.py#L9.
    warmup='linear',  # The warmup policy, also support `exp` and `constant`.
    warmup_iters=500,  # The number of iterations for warmup
    warmup_ratio=
    0.001,  # The ratio of the starting learning rate used for warmup
    step=[8, 11])  # Steps to decay the learning rate
runner = dict(
    type='EpochBasedRunner', # Type of runner to use (i.e. IterBasedRunner or␣
↪EpochBasedRunner)
    max_epochs=12) # Runner that runs the workflow in total max_epochs. For␣
↪IterBasedRunner use `max_iters`
```

(continues on next page)

(continued from previous page)

```
checkpoint_config = dict(  # Config to set the checkpoint hook, Refer to https://github.
↪com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/checkpoint.py for implementation.
    interval=1)  # The save interval is 1
log_config = dict(  # config to register logger hook
    interval=50,  # Interval to print the log
    hooks=[
        # dict(type='TensorboardLoggerHook')  # The Tensorboard logger is also supported
        dict(type='TextLoggerHook')
    ])  # The logger used to record the training process.
dist_params = dict(backend='nccl')  # Parameters to setup distributed training, the port␣
↪can also be set.
log_level = 'INFO'  # The level of logging.
load_from = None  # load models as a pre-trained model from a given path. This will not␣
↪resume training.
resume_from = None  # Resume checkpoints from a given path, the training will be resumed␣
↪from the epoch when the checkpoint's is saved.
workflow = [('train', 1)]  # Workflow for runner. [('train', 1)] means there is only one␣
↪workflow and the workflow named 'train' is executed once. The workflow trains the␣
↪model by 12 epochs according to the total_epochs.
work_dir = 'work_dir'  # Directory to save the model checkpoints and logs for the␣
↪current experiments.
```

8.6 FAQ

8.6.1 Ignore some fields in the base configs

Sometimes, you may set `_delete_=True` to ignore some of fields in base configs. You may refer to mmcv for simple illustration.

In MMDetection, for example, to change the backbone of Mask R-CNN with the following config.

```
model = dict(
    type='MaskRCNN',
    pretrained='torchvision://resnet50',
    backbone=dict(
        type='ResNet',
        depth=50,
        num_stages=4,
        out_indices=(0, 1, 2, 3),
        frozen_stages=1,
        norm_cfg=dict(type='BN', requires_grad=True),
        norm_eval=True,
        style='pytorch'),
    neck=dict(...),
    rpn_head=dict(...),
    roi_head=dict(...))
```

`ResNet` and `HRNet` use different keywords to construct.

```
_base_ = '../mask_rcnn/mask_rcnn_r50_fpn_1x_coco.py'
model = dict(
    pretrained='open-mmlab://msra/hrnetv2_w32',
    backbone=dict(
        _delete_=True,
        type='HRNet',
        extra=dict(
            stage1=dict(
                num_modules=1,
                num_branches=1,
                block='BOTTLENECK',
                num_blocks=(4, ),
                num_channels=(64, )),
            stage2=dict(
                num_modules=1,
                num_branches=2,
                block='BASIC',
                num_blocks=(4, 4),
                num_channels=(32, 64)),
            stage3=dict(
                num_modules=4,
                num_branches=3,
                block='BASIC',
                num_blocks=(4, 4, 4),
                num_channels=(32, 64, 128)),
            stage4=dict(
                num_modules=3,
                num_branches=4,
                block='BASIC',
                num_blocks=(4, 4, 4, 4),
                num_channels=(32, 64, 128, 256)))),
    neck=dict(...))
```

The `_delete_=True` would replace all old keys in `backbone` field with new keys.

8.6.2 Use intermediate variables in configs

Some intermediate variables are used in the configs files, like `train_pipeline/test_pipeline` in datasets. It's worth noting that when modifying intermediate variables in the children configs, user need to pass the intermediate variables into corresponding fields again. For example, we would like to use multi scale strategy to train a Mask R-CNN. `train_pipeline/test_pipeline` are intermediate variable we would like modify.

```
_base_ = './mask_rcnn_r50_fpn_1x_coco.py'
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='LoadAnnotations', with_bbox=True, with_mask=True),
    dict(
        type='Resize',
        img_scale=[(1333, 640), (1333, 672), (1333, 704), (1333, 736),
                   (1333, 768), (1333, 800)],
```

(continues on next page)

(continued from previous page)

```
        multiscale_mode="value",
        keep_ratio=True),
    dict(type='RandomFlip', flip_ratio=0.5),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='Pad', size_divisor=32),
    dict(type='DefaultFormatBundle'),
    dict(type='Collect', keys=['img', 'gt_bboxes', 'gt_labels', 'gt_masks']),
]
test_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(
        type='MultiScaleFlipAug',
        img_scale=(1333, 800),
        flip=False,
        transforms=[
            dict(type='Resize', keep_ratio=True),
            dict(type='RandomFlip'),
            dict(type='Normalize', **img_norm_cfg),
            dict(type='Pad', size_divisor=32),
            dict(type='ImageToTensor', keys=['img']),
            dict(type='Collect', keys=['img']),
        ])
]
data = dict(
    train=dict(pipeline=train_pipeline),
    val=dict(pipeline=test_pipeline),
    test=dict(pipeline=test_pipeline))
```

We first define the new `train_pipeline/test_pipeline` and pass them into `data`.

Similarly, if we would like to switch from `SyncBN` to `BN` or `MMSyncBN`, we need to substitute every `norm_cfg` in the config.

```
_base_ = './mask_rcnn_r50_fpn_1x_coco.py'
norm_cfg = dict(type='BN', requires_grad=True)
model = dict(
    backbone=dict(norm_cfg=norm_cfg),
    neck=dict(norm_cfg=norm_cfg),
    ...)
```

CHAPTER

NINE

TUTORIAL 2: CUSTOMIZE DATASETS

9.1 Support new data format

To support a new data format, you can either convert them to existing formats (COCO format or PASCAL format) or directly convert them to the middle format. You could also choose to convert them offline (before training by a script) or online (implement a new dataset and do the conversion at training). In MMDetection, we recommend to convert the data into COCO formats and do the conversion offline, thus you only need to modify the config's data annotation paths and classes after the conversion of your data.

9.1.1 Reorganize new data formats to existing format

The simplest way is to convert your dataset to existing dataset formats (COCO or PASCAL VOC).

The annotation json files in COCO format has the following necessary keys:

```
'images': [
    {
        'file_name': 'COCO_val2014_000000001268.jpg',
        'height': 427,
        'width': 640,
        'id': 1268
    },
    ...
],

'annotations': [
    {
        'segmentation': [[192.81,
            247.09,
            ...
            219.03,
            249.06]],  # if you have mask labels
        'area': 1035.749,
        'iscrowd': 0,
        'image_id': 1268,
        'bbox': [192.81, 224.8, 74.73, 33.43],
        'category_id': 16,
        'id': 42986
    },
    ...
```

(continues on next page)

(continued from previous page)

```
],

'categories': [
    {'id': 0, 'name': 'car'},
 ]
```

There are three necessary keys in the json file:

- `images`: contains a list of images with their information like `file_name`, `height`, `width`, and `id`.
- `annotations`: contains the list of instance annotations.
- `categories`: contains the list of categories names and their ID.

After the data pre-processing, there are two steps for users to train the customized new dataset with existing format (e.g. COCO format):

1. Modify the config file for using the customized dataset.
2. Check the annotations of the customized dataset.

Here we give an example to show the above two steps, which uses a customized dataset of 5 classes with COCO format to train an existing Cascade Mask R-CNN R50-FPN detector.

1. Modify the config file for using the customized dataset

There are two aspects involved in the modification of config file:

1. The `data` field. Specifically, you need to explicitly add the `classes` fields in `data.train`, `data.val` and `data.test`.
2. The `num_classes` field in the `model` part. Explicitly over-write all the `num_classes` from default value (e.g. 80 in COCO) to your classes number.

In `configs/my_custom_config.py`:

```
# the new config inherits the base configs to highlight the necessary modification
_base_ = './cascade_mask_rcnn_r50_fpn_1x_coco.py'

# 1. dataset settings
dataset_type = 'CocoDataset'
classes = ('a', 'b', 'c', 'd', 'e')
data = dict(
    samples_per_gpu=2,
    workers_per_gpu=2,
    train=dict(
        type=dataset_type,
        # explicitly add your class names to the field `classes`
        classes=classes,
        ann_file='path/to/your/train/annotation_data',
        img_prefix='path/to/your/train/image_data'),
    val=dict(
        type=dataset_type,
        # explicitly add your class names to the field `classes`
        classes=classes,
```

(continues on next page)

(continued from previous page)

```
        ann_file='path/to/your/val/annotation_data',
        img_prefix='path/to/your/val/image_data'),
    test=dict(
        type=dataset_type,
        # explicitly add your class names to the field `classes`
        classes=classes,
        ann_file='path/to/your/test/annotation_data',
        img_prefix='path/to/your/test/image_data'))

# 2. model settings

# explicitly over-write all the `num_classes` field from default 80 to 5.
model = dict(
    roi_head=dict(
        bbox_head=[
            dict(
                type='Shared2FCBBoxHead',
                # explicitly over-write all the `num_classes` field from default 80 to 5.
                num_classes=5),
            dict(
                type='Shared2FCBBoxHead',
                # explicitly over-write all the `num_classes` field from default 80 to 5.
                num_classes=5),
            dict(
                type='Shared2FCBBoxHead',
                # explicitly over-write all the `num_classes` field from default 80 to 5.
                num_classes=5)],
    # explicitly over-write all the `num_classes` field from default 80 to 5.
    mask_head=dict(num_classes=5)))
```

2. Check the annotations of the customized dataset

Assuming your customized dataset is COCO format, make sure you have the correct annotations in the customized dataset:

1. The length for `categories` field in annotations should exactly equal the tuple length of `classes` fields in your config, meaning the number of classes (e.g. 5 in this example).
2. The `classes` fields in your config file should have exactly the same elements and the same order with the `name` in `categories` of annotations. MMDetection automatically maps the uncontinuous `id` in `categories` to the continuous label indices, so the string order of `name` in `categories` field affects the order of label indices. Meanwhile, the string order of `classes` in config affects the label text during visualization of predicted bounding boxes.
3. The `category_id` in `annotations` field should be valid, i.e., all values in `category_id` should belong to `id` in `categories`.

Here is a valid example of annotations:

```
'annotations': [
    {
        'segmentation': [[192.81,
```

(continues on next page)

(continued from previous page)

```
            247.09,
            ...
            219.03,
            249.06]],  # if you have mask labels
        'area': 1035.749,
        'iscrowd': 0,
        'image_id': 1268,
        'bbox': [192.81, 224.8, 74.73, 33.43],
        'category_id': 16,
        'id': 42986
    },
    ...
],

# MMDetection automatically maps the uncontinuous `id` to the continuous label indices.
'categories': [
    {'id': 1, 'name': 'a'}, {'id': 3, 'name': 'b'}, {'id': 4, 'name': 'c'}, {'id': 16,
↪'name': 'd'}, {'id': 17, 'name': 'e'},
 ]
```

We use this way to support CityScapes dataset. The script is in cityscapes.py and we also provide the finetuning configs.

Note

1. For instance segmentation datasets, **MMDetection only supports evaluating mask AP of dataset in COCO format for now**.
2. It is recommended to convert the data offline before training, thus you can still use `CocoDataset` and only need to modify the path of annotations and the training classes.

9.1.2 Reorganize new data format to middle format

It is also fine if you do not want to convert the annotation format to COCO or PASCAL format. Actually, we define a simple annotation format and all existing datasets are processed to be compatible with it, either online or offline.

The annotation of a dataset is a list of dict, each dict corresponds to an image. There are 3 field `filename` (relative path), `width`, `height` for testing, and an additional field `ann` for training. `ann` is also a dict containing at least 2 fields: `bboxes` and `labels`, both of which are numpy arrays. Some datasets may provide annotations like crowd/difficult/ignored bboxes, we use `bboxes_ignore` and `labels_ignore` to cover them.

Here is an example.

```
[
    {
        'filename': 'a.jpg',
        'width': 1280,
        'height': 720,
        'ann': {
            'bboxes': <np.ndarray, float32> (n, 4),
            'labels': <np.ndarray, int64> (n, ),
            'bboxes_ignore': <np.ndarray, float32> (k, 4),
            'labels_ignore': <np.ndarray, int64> (k, ) (optional field)
        }
```

(continues on next page)

(continued from previous page)

```
    },
    ...
]
```

There are two ways to work with custom datasets.

- online conversion

  You can write a new Dataset class inherited from `CustomDataset`, and overwrite two methods `load_annotations(self, ann_file)` and `get_ann_info(self, idx)`, like CocoDataset and VOCDataset.

- offline conversion

  You can convert the annotation format to the expected format above and save it to a pickle or json file, like pascal_voc.py. Then you can simply use `CustomDataset`.

9.1.3 An example of customized dataset

Assume the annotation is in a new format in text files. The bounding boxes annotations are stored in text file `annotation.txt` as the following

```
#
000001.jpg
1280 720
2
10 20 40 60 1
20 40 50 60 2
#
000002.jpg
1280 720
3
50 20 40 60 2
20 40 30 45 2
30 40 50 60 3
```

We can create a new dataset in `mmdet/datasets/my_dataset.py` to load the data.

```
import mmcv
import numpy as np

from .builder import DATASETS
from .custom import CustomDataset


@DATASETS.register_module()
class MyDataset(CustomDataset):

    CLASSES = ('person', 'bicycle', 'car', 'motorcycle')

    def load_annotations(self, ann_file):
        ann_list = mmcv.list_from_file(ann_file)
```

(continues on next page)

(continued from previous page)

```
        data_infos = []
        for i, ann_line in enumerate(ann_list):
            if ann_line != '#':
                continue

            img_shape = ann_list[i + 2].split(' ')
            width = int(img_shape[0])
            height = int(img_shape[1])
            bbox_number = int(ann_list[i + 3])

            anns = ann_line.split(' ')
            bboxes = []
            labels = []
            for anns in ann_list[i + 4:i + 4 + bbox_number]:
                bboxes.append([float(ann) for ann in anns[:4]])
                labels.append(int(anns[4]))

            data_infos.append(
                dict(
                    filename=ann_list[i + 1],
                    width=width,
                    height=height,
                    ann=dict(
                        bboxes=np.array(bboxes).astype(np.float32),
                        labels=np.array(labels).astype(np.int64))
                ))

        return data_infos

    def get_ann_info(self, idx):
        return self.data_infos[idx]['ann']
```

Then in the config, to use `MyDataset` you can modify the config as the following

```
dataset_A_train = dict(
    type='MyDataset',
    ann_file = 'image_list.txt',
    pipeline=train_pipeline
)
```

9.2 Customize datasets by dataset wrappers

MMDetection also supports many dataset wrappers to mix the dataset or modify the dataset distribution for training. Currently it supports to three dataset wrappers as below:

- `RepeatDataset`: simply repeat the whole dataset.
- `ClassBalancedDataset`: repeat dataset in a class balanced manner.
- `ConcatDataset`: concat datasets.

9.2.1 Repeat dataset

We use `RepeatDataset` as wrapper to repeat the dataset. For example, suppose the original dataset is `Dataset_A`, to repeat it, the config looks like the following

```
dataset_A_train = dict(
        type='RepeatDataset',
        times=N,
        dataset=dict(  # This is the original config of Dataset_A
            type='Dataset_A',
            ...
            pipeline=train_pipeline
        )
    )
```

9.2.2 Class balanced dataset

We use `ClassBalancedDataset` as wrapper to repeat the dataset based on category frequency. The dataset to repeat needs to instantiate function `self.get_cat_ids(idx)` to support `ClassBalancedDataset`. For example, to repeat `Dataset_A` with `oversample_thr=1e-3`, the config looks like the following

```
dataset_A_train = dict(
        type='ClassBalancedDataset',
        oversample_thr=1e-3,
        dataset=dict(  # This is the original config of Dataset_A
            type='Dataset_A',
            ...
            pipeline=train_pipeline
        )
    )
```

You may refer to source code for details.

9.2.3 Concatenate dataset

There are three ways to concatenate the dataset.

1. If the datasets you want to concatenate are in the same type with different annotation files, you can concatenate the dataset configs like the following.

   ```
   dataset_A_train = dict(
       type='Dataset_A',
       ann_file = ['anno_file_1', 'anno_file_2'],
       pipeline=train_pipeline
   )
   ```

   If the concatenated dataset is used for test or evaluation, this manner supports to evaluate each dataset separately. To test the concatenated datasets as a whole, you can set `separate_eval=False` as below.

   ```
   dataset_A_train = dict(
       type='Dataset_A',
       ann_file = ['anno_file_1', 'anno_file_2'],
   ```

(continues on next page)

(continued from previous page)

```
    separate_eval=False,
    pipeline=train_pipeline
)
```

2. In case the dataset you want to concatenate is different, you can concatenate the dataset configs like the following.

```
dataset_A_train = dict()
dataset_B_train = dict()

data = dict(
    imgs_per_gpu=2,
    workers_per_gpu=2,
    train = [
        dataset_A_train,
        dataset_B_train
    ],
    val = dataset_A_val,
    test = dataset_A_test
    )
```

   If the concatenated dataset is used for test or evaluation, this manner also supports to evaluate each dataset separately.

3. We also support to define `ConcatDataset` explicitly as the following.

```
dataset_A_val = dict()
dataset_B_val = dict()

data = dict(
    imgs_per_gpu=2,
    workers_per_gpu=2,
    train=dataset_A_train,
    val=dict(
        type='ConcatDataset',
        datasets=[dataset_A_val, dataset_B_val],
        separate_eval=False))
```

   This manner allows users to evaluate all the datasets as a single one by setting `separate_eval=False`.

Note:

1. The option `separate_eval=False` assumes the datasets use `self.data_infos` during evaluation. Therefore, COCO datasets do not support this behavior since COCO datasets do not fully rely on `self.data_infos` for evaluation. Combining different types of datasets and evaluating them as a whole is not tested thus is not suggested.
2. Evaluating `ClassBalancedDataset` and `RepeatDataset` is not supported thus evaluating concatenated datasets of these types is also not supported.

A more complex example that repeats `Dataset_A` and `Dataset_B` by N and M times, respectively, and then concatenates the repeated datasets is as the following.

```
dataset_A_train = dict(
    type='RepeatDataset',
    times=N,
```

(continues on next page)

(continued from previous page)

```
    dataset=dict(
        type='Dataset_A',
        ...
        pipeline=train_pipeline
    )
)
dataset_A_val = dict(
    ...
    pipeline=test_pipeline
)
dataset_A_test = dict(
    ...
    pipeline=test_pipeline
)
dataset_B_train = dict(
    type='RepeatDataset',
    times=M,
    dataset=dict(
        type='Dataset_B',
        ...
        pipeline=train_pipeline
    )
)
data = dict(
    imgs_per_gpu=2,
    workers_per_gpu=2,
    train = [
        dataset_A_train,
        dataset_B_train
    ],
    val = dataset_A_val,
    test = dataset_A_test
)
```

9.3 Modify Dataset Classes

With existing dataset types, we can modify the class names of them to train subset of the annotations. For example, if you want to train only three classes of the current dataset, you can modify the classes of dataset. The dataset will filter out the ground truth boxes of other classes automatically.

```
classes = ('person', 'bicycle', 'car')
data = dict(
    train=dict(classes=classes),
    val=dict(classes=classes),
    test=dict(classes=classes))
```

MMDetection V2.0 also supports to read the classes from a file, which is common in real applications. For example, assume the `classes.txt` contains the name of classes as the following.

```
person
bicycle
car
```

Users can set the classes as a file path, the dataset will load it and convert it to a list automatically.

```
classes = 'path/to/classes.txt'
data = dict(
    train=dict(classes=classes),
    val=dict(classes=classes),
    test=dict(classes=classes))
```

Note:

- Before MMDetection v2.5.0, the dataset will filter out the empty GT images automatically if the classes are set and there is no way to disable that through config. This is an undesirable behavior and introduces confusion because if the classes are not set, the dataset only filter the empty GT images when `filter_empty_gt=True` and `test_mode=False`. After MMDetection v2.5.0, we decouple the image filtering process and the classes modification, i.e., the dataset will only filter empty GT images when `filter_empty_gt=True` and `test_mode=False`, no matter whether the classes are set. Thus, setting the classes only influences the annotations of classes used for training and users could decide whether to filter empty GT images by themselves.
- Since the middle format only has box labels and does not contain the class names, when using `CustomDataset`, users cannot filter out the empty GT images through configs but only do this offline.
- Please remember to modify the `num_classes` in the head when specifying `classes` in dataset. We implemented NumClassCheckHook to check whether the numbers are consistent since v2.9.0(after PR#4508).
- The features for setting dataset classes and dataset filtering will be refactored to be more user-friendly in the future (depends on the progress).

9.4 COCO Panoptic Dataset

Now we support COCO Panoptic Dataset, the format of panoptic annotations is different from COCO format. Both the foreground and the background will exist in the annotation file. The annotation json files in COCO Panoptic format has the following necessary keys:

```
'images': [
    {
        'file_name': '000000001268.jpg',
        'height': 427,
        'width': 640,
        'id': 1268
    },
    ...
]

'annotations': [
    {
        'filename': '000000001268.jpg',
        'image_id': 1268,
        'segments_info': [
            {
```

(continues on next page)

(continued from previous page)

```
                'id':8345037,  # One-to-one correspondence with the id in the annotation␣
↪map.
                'category_id': 51,
                'iscrowd': 0,
                'bbox': (x1, y1, w, h),  # The bbox of the background is the outer␣
↪rectangle of its mask.
                'area': 24315
            },
            ...
        ]
    },
    ...
]

'categories': [  # including both foreground categories and background categories
    {'id': 0, 'name': 'person'},
    ...
 ]
```

Moreover, the `seg_prefix` must be set to the path of the panoptic annotation images.

```
data = dict(
    type='CocoPanopticDataset',
    train=dict(
        seg_prefix = 'path/to/your/train/panoptic/image_annotation_data'
    ),
    val=dict(
        seg_prefix = 'path/to/your/train/panoptic/image_annotation_data'
    )
)
```

CHAPTER

TEN

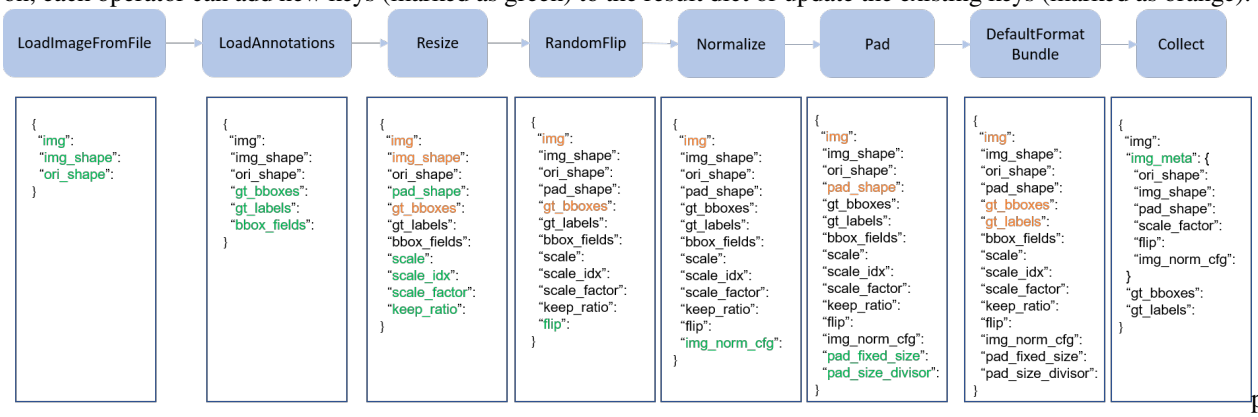
TUTORIAL 3: CUSTOMIZE DATA PIPELINES

10.1 Design of Data pipelines

Following typical conventions, we use `Dataset` and `DataLoader` for data loading with multiple workers. `Dataset` returns a dict of data items corresponding the arguments of models' forward method. Since the data in object detection may not be the same size (image size, gt bbox size, etc.), we introduce a new `DataContainer` type in MMCV to help collect and distribute data of different size. See here for more details.

The data preparation pipeline and the dataset is decomposed. Usually a dataset defines how to process the annotations and a data pipeline defines all the steps to prepare a data dict. A pipeline consists of a sequence of operations. Each operation takes a dict as input and also output a dict for the next transform.

We present a classical pipeline in the following figure. The blue blocks are pipeline operations. With the pipeline going on, each operator can add new keys (marked as green) to the result dict or update the existing keys (marked as orange).



pipeline figure

The operations are categorized into data loading, pre-processing, formatting and test-time augmentation.

Here is a pipeline example for Faster R-CNN.

```
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='LoadAnnotations', with_bbox=True),
    dict(type='Resize', img_scale=(1333, 800), keep_ratio=True),
    dict(type='RandomFlip', flip_ratio=0.5),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='Pad', size_divisor=32),
    dict(type='DefaultFormatBundle'),
```

(continues on next page)

(continued from previous page)

```
    dict(type='Collect', keys=['img', 'gt_bboxes', 'gt_labels']),
]
test_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(
        type='MultiScaleFlipAug',
        img_scale=(1333, 800),
        flip=False,
        transforms=[
            dict(type='Resize', keep_ratio=True),
            dict(type='RandomFlip'),
            dict(type='Normalize', **img_norm_cfg),
            dict(type='Pad', size_divisor=32),
            dict(type='ImageToTensor', keys=['img']),
            dict(type='Collect', keys=['img']),
        ])
]
```

For each operation, we list the related dict fields that are added/updated/removed.

10.1.1 Data loading

`LoadImageFromFile`

- add: img, img_shape, ori_shape

`LoadAnnotations`

- add: gt_bboxes, gt_bboxes_ignore, gt_labels, gt_masks, gt_semantic_seg, bbox_fields, mask_fields

`LoadProposals`

- add: proposals

10.1.2 Pre-processing

`Resize`

- add: scale, scale_idx, pad_shape, scale_factor, keep_ratio
- update: img, img_shape, *bbox_fields, *mask_fields, *seg_fields

`RandomFlip`

- add: flip
- update: img, *bbox_fields, *mask_fields, *seg_fields

`Pad`

- add: pad_fixed_size, pad_size_divisor
- update: img, pad_shape, *mask_fields, *seg_fields

`RandomCrop`

- update: img, pad_shape, gt_bboxes, gt_labels, gt_masks, *bbox_fields

`Normalize`

- add: img_norm_cfg
- update: img

`SegRescale`

- update: gt_semantic_seg

`PhotoMetricDistortion`

- update: img

`Expand`

- update: img, gt_bboxes

`MinIoURandomCrop`

- update: img, gt_bboxes, gt_labels

`Corrupt`

- update: img

10.1.3 Formatting

`ToTensor`

- update: specified by `keys`.

`ImageToTensor`

- update: specified by `keys`.

`Transpose`

- update: specified by `keys`.

`ToDataContainer`

- update: specified by `fields`.

`DefaultFormatBundle`

- update: img, proposals, gt_bboxes, gt_bboxes_ignore, gt_labels, gt_masks, gt_semantic_seg

`Collect`

- add: img_meta (the keys of img_meta is specified by `meta_keys`)
- remove: all other keys except for those specified by `keys`

10.1.4 Test time augmentation

`MultiScaleFlipAug`

10.2 Extend and use custom pipelines

1. Write a new pipeline in a file, e.g., in `my_pipeline.py`. It takes a dict as input and returns a dict.

```
import random
from mmdet.datasets import PIPELINES


@PIPELINES.register_module()
class MyTransform:
    """Add your transform

    Args:
        p (float): Probability of shifts. Default 0.5.
    """

    def __init__(self, p=0.5):
        self.p = p

    def __call__(self, results):
        if random.random() > self.p:
            results['dummy'] = True
        return results
```

2. Import and use the pipeline in your config file. Make sure the import is relative to where your train script is located.

```
custom_imports = dict(imports=['path.to.my_pipeline'], allow_failed_imports=False)

img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='LoadAnnotations', with_bbox=True),
    dict(type='Resize', img_scale=(1333, 800), keep_ratio=True),
    dict(type='RandomFlip', flip_ratio=0.5),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='Pad', size_divisor=32),
    dict(type='MyTransform', p=0.2),
    dict(type='DefaultFormatBundle'),
    dict(type='Collect', keys=['img', 'gt_bboxes', 'gt_labels']),
]
```

3. Visualize the output of your augmentation pipeline

   To visualize the output of your agmentation pipeline, `tools/misc/browse_dataset.py` can help the user to browse a detection dataset (both images and bounding box annotations) visually, or save the image to a designated directory. More detials can refer to *useful_tools*

CHAPTER

ELEVEN

TUTORIAL 4: CUSTOMIZE MODELS

We basically categorize model components into 5 types.

- backbone: usually an FCN network to extract feature maps, e.g., ResNet, MobileNet.
- neck: the component between backbones and heads, e.g., FPN, PAFPN.
- head: the component for specific tasks, e.g., bbox prediction and mask prediction.
- roi extractor: the part for extracting RoI features from feature maps, e.g., RoI Align.
- loss: the component in head for calculating losses, e.g., FocalLoss, L1Loss, and GHMLoss.

11.1 Develop new components

11.1.1 Add a new backbone

Here we show how to develop new components with an example of MobileNet.

1. Define a new backbone (e.g. MobileNet)

Create a new file `mmdet/models/backbones/mobilenet.py`.

```
import torch.nn as nn

from ..builder import BACKBONES


@BACKBONES.register_module()
class MobileNet(nn.Module):

    def __init__(self, arg1, arg2):
        pass

    def forward(self, x):  # should return a tuple
        pass
```

2. Import the module

You can either add the following line to `mmdet/models/backbones/__init__.py`

```
from .mobilenet import MobileNet
```

or alternatively add

```
custom_imports = dict(
    imports=['mmdet.models.backbones.mobilenet'],
    allow_failed_imports=False)
```

to the config file to avoid modifying the original code.

3. Use the backbone in your config file

```
model = dict(
    ...
    backbone=dict(
        type='MobileNet',
        arg1=xxx,
        arg2=xxx),
    ...
```

11.1.2 Add new necks

1. Define a neck (e.g. PAFPN)

Create a new file `mmdet/models/necks/pafpn.py`.

```
from ..builder import NECKS

@NECKS.register_module()
class PAFPN(nn.Module):

    def __init__(self,
                 in_channels,
                 out_channels,
                 num_outs,
                 start_level=0,
                 end_level=-1,
                 add_extra_convs=False):
        pass

    def forward(self, inputs):
        # implementation is ignored
        pass
```

2. Import the module

You can either add the following line to `mmdet/models/necks/__init__.py`,

```
from .pafpn import PAFPN
```

or alternatively add

```
custom_imports = dict(
    imports=['mmdet.models.necks.pafpn.py'],
    allow_failed_imports=False)
```

to the config file and avoid modifying the original code.

3. Modify the config file

```
neck=dict(
    type='PAFPN',
    in_channels=[256, 512, 1024, 2048],
    out_channels=256,
    num_outs=5)
```

11.1.3 Add new heads

Here we show how to develop a new head with the example of Double Head R-CNN as the following.

First, add a new bbox head in `mmdet/models/roi_heads/bbox_heads/double_bbox_head.py`. Double Head R-CNN implements a new bbox head for object detection. To implement a bbox head, basically we need to implement three functions of the new module as the following.

```
from mmdet.models.builder import HEADS
from .bbox_head import BBoxHead

@HEADS.register_module()
class DoubleConvFCBBoxHead(BBoxHead):
    r"""Bbox head used in Double-Head R-CNN

                                    /-> cls
                /-> shared convs ->
                                    \-> reg
    roi features
                                    /-> cls
                \-> shared fc    ->
                                    \-> reg
    """  # noqa: W605

    def __init__(self,
                 num_convs=0,
                 num_fcs=0,
                 conv_out_channels=1024,
                 fc_out_channels=1024,
                 conv_cfg=None,
```

(continues on next page)

(continued from previous page)

```
                 norm_cfg=dict(type='BN'),
                 **kwargs):
        kwargs.setdefault('with_avg_pool', True)
        super(DoubleConvFCBBoxHead, self).__init__(**kwargs)


    def forward(self, x_cls, x_reg):
```

Second, implement a new RoI Head if it is necessary. We plan to inherit the new `DoubleHeadRoIHead` from `StandardRoIHead`. We can find that a `StandardRoIHead` already implements the following functions.

```
import torch

from mmdet.core import bbox2result, bbox2roi, build_assigner, build_sampler
from ..builder import HEADS, build_head, build_roi_extractor
from .base_roi_head import BaseRoIHead
from .test_mixins import BBoxTestMixin, MaskTestMixin


@HEADS.register_module()
class StandardRoIHead(BaseRoIHead, BBoxTestMixin, MaskTestMixin):
    """Simplest base roi head including one bbox head and one mask head.
    """

    def init_assigner_sampler(self):

    def init_bbox_head(self, bbox_roi_extractor, bbox_head):

    def init_mask_head(self, mask_roi_extractor, mask_head):


    def forward_dummy(self, x, proposals):


    def forward_train(self,
                      x,
                      img_metas,
                      proposal_list,
                      gt_bboxes,
                      gt_labels,
                      gt_bboxes_ignore=None,
                      gt_masks=None):

    def _bbox_forward(self, x, rois):

    def _bbox_forward_train(self, x, sampling_results, gt_bboxes, gt_labels,
                            img_metas):

    def _mask_forward_train(self, x, sampling_results, bbox_feats, gt_masks,
                            img_metas):

    def _mask_forward(self, x, rois=None, pos_inds=None, bbox_feats=None):
```

(continues on next page)

(continued from previous page)

```
    def simple_test(self,
                    x,
                    proposal_list,
                    img_metas,
                    proposals=None,
                    rescale=False):
        """Test without augmentation."""
```

Double Head's modification is mainly in the bbox_forward logic, and it inherits other logics from the StandardRoIHead. In the mmdet/models/roi_heads/double_roi_head.py, we implement the new RoI Head as the following:

```
from ..builder import HEADS
from .standard_roi_head import StandardRoIHead


@HEADS.register_module()
class DoubleHeadRoIHead(StandardRoIHead):
    """RoI head for Double Head RCNN

    https://arxiv.org/abs/1904.06493
    """

    def __init__(self, reg_roi_scale_factor, **kwargs):
        super(DoubleHeadRoIHead, self).__init__(**kwargs)
        self.reg_roi_scale_factor = reg_roi_scale_factor

    def _bbox_forward(self, x, rois):
        bbox_cls_feats = self.bbox_roi_extractor(
            x[:self.bbox_roi_extractor.num_inputs], rois)
        bbox_reg_feats = self.bbox_roi_extractor(
            x[:self.bbox_roi_extractor.num_inputs],
            rois,
            roi_scale_factor=self.reg_roi_scale_factor)
        if self.with_shared_head:
            bbox_cls_feats = self.shared_head(bbox_cls_feats)
            bbox_reg_feats = self.shared_head(bbox_reg_feats)
        cls_score, bbox_pred = self.bbox_head(bbox_cls_feats, bbox_reg_feats)

        bbox_results = dict(
            cls_score=cls_score,
            bbox_pred=bbox_pred,
            bbox_feats=bbox_cls_feats)
        return bbox_results
```

Last, the users need to add the module in mmdet/models/bbox_heads/__init__.py and mmdet/models/roi_heads/__init__.py thus the corresponding registry could find and load them.

Alternatively, the users can add

```
custom_imports=dict(
    imports=['mmdet.models.roi_heads.double_roi_head', 'mmdet.models.bbox_heads.double_
↪bbox_head'])
```

to the config file and achieve the same goal.

The config file of Double Head R-CNN is as the following

```
_base_ = '../faster_rcnn/faster_rcnn_r50_fpn_1x_coco.py'
model = dict(
    roi_head=dict(
        type='DoubleHeadRoIHead',
        reg_roi_scale_factor=1.3,
        bbox_head=dict(
            _delete_=True,
            type='DoubleConvFCBBoxHead',
            num_convs=4,
            num_fcs=2,
            in_channels=256,
            conv_out_channels=1024,
            fc_out_channels=1024,
            roi_feat_size=7,
            num_classes=80,
            bbox_coder=dict(
                type='DeltaXYWHBBoxCoder',
                target_means=[0., 0., 0., 0.],
                target_stds=[0.1, 0.1, 0.2, 0.2]),
            reg_class_agnostic=False,
            loss_cls=dict(
                type='CrossEntropyLoss', use_sigmoid=False, loss_weight=2.0),
            loss_bbox=dict(type='SmoothL1Loss', beta=1.0, loss_weight=2.0))))
```

Since MMDetection 2.0, the config system supports to inherit configs such that the users can focus on the modification. The Double Head R-CNN mainly uses a new DoubleHeadRoIHead and a new `DoubleConvFCBBoxHead`, the arguments are set according to the `__init__` function of each module.

11.1.4 Add new loss

Assume you want to add a new loss as `MyLoss`, for bounding box regression. To add a new loss function, the users need implement it in `mmdet/models/losses/my_loss.py`. The decorator `weighted_loss` enable the loss to be weighted for each element.

```
import torch
import torch.nn as nn

from ..builder import LOSSES
from .utils import weighted_loss

@weighted_loss
def my_loss(pred, target):
    assert pred.size() == target.size() and target.numel() > 0
    loss = torch.abs(pred - target)
    return loss
```

(continues on next page)

(continued from previous page)

```
@LOSSES.register_module()
class MyLoss(nn.Module):

    def __init__(self, reduction='mean', loss_weight=1.0):
        super(MyLoss, self).__init__()
        self.reduction = reduction
        self.loss_weight = loss_weight

    def forward(self,
                pred,
                target,
                weight=None,
                avg_factor=None,
                reduction_override=None):
        assert reduction_override in (None, 'none', 'mean', 'sum')
        reduction = (
            reduction_override if reduction_override else self.reduction)
        loss_bbox = self.loss_weight * my_loss(
            pred, target, weight, reduction=reduction, avg_factor=avg_factor)
        return loss_bbox
```

Then the users need to add it in the `mmdet/models/losses/__init__.py`.

```
from .my_loss import MyLoss, my_loss
```

Alternatively, you can add

```
custom_imports=dict(
    imports=['mmdet.models.losses.my_loss'])
```

to the config file and achieve the same goal.

To use it, modify the `loss_xxx` field. Since MyLoss is for regression, you need to modify the `loss_bbox` field in the head.

```
loss_bbox=dict(type='MyLoss', loss_weight=1.0))
```

CHAPTER

TWELVE

TUTORIAL 5: CUSTOMIZE RUNTIME SETTINGS

12.1 Customize optimization settings

12.1.1 Customize optimizer supported by Pytorch

We already support to use all the optimizers implemented by PyTorch, and the only modification is to change the `optimizer` field of config files. For example, if you want to use `ADAM` (note that the performance could drop a lot), the modification could be as the following.

```
optimizer = dict(type='Adam', lr=0.0003, weight_decay=0.0001)
```

To modify the learning rate of the model, the users only need to modify the `lr` in the config of optimizer. The users can directly set arguments following the API doc of PyTorch.

12.1.2 Customize self-implemented optimizer

1. Define a new optimizer

A customized optimizer could be defined as following.

Assume you want to add a optimizer named `MyOptimizer`, which has arguments `a`, `b`, and `c`. You need to create a new directory named `mmdet/core/optimizer`. And then implement the new optimizer in a file, e.g., in `mmdet/core/optimizer/my_optimizer.py`:

```
from .registry import OPTIMIZERS
from torch.optim import Optimizer


@OPTIMIZERS.register_module()
class MyOptimizer(Optimizer):

    def __init__(self, a, b, c)
```

2. Add the optimizer to registry

To find the above module defined above, this module should be imported into the main namespace at first. There are two options to achieve it.

- Modify `mmdet/core/optimizer/__init__.py` to import it.

  The newly defined module should be imported in `mmdet/core/optimizer/__init__.py` so that the registry will find the new module and add it:

```
from .my_optimizer import MyOptimizer
```

- Use `custom_imports` in the config to manually import it

```
custom_imports = dict(imports=['mmdet.core.optimizer.my_optimizer'], allow_failed_
↪imports=False)
```

The module `mmdet.core.optimizer.my_optimizer` will be imported at the beginning of the program and the class `MyOptimizer` is then automatically registered. Note that only the package containing the class `MyOptimizer` should be imported. `mmdet.core.optimizer.my_optimizer.MyOptimizer` **cannot** be imported directly.

Actually users can use a totally different file directory structure using this importing method, as long as the module root can be located in `PYTHONPATH`.

3. Specify the optimizer in the config file

Then you can use `MyOptimizer` in `optimizer` field of config files. In the configs, the optimizers are defined by the field `optimizer` like the following:

```
optimizer = dict(type='SGD', lr=0.02, momentum=0.9, weight_decay=0.0001)
```

To use your own optimizer, the field can be changed to

```
optimizer = dict(type='MyOptimizer', a=a_value, b=b_value, c=c_value)
```

12.1.3 Customize optimizer constructor

Some models may have some parameter-specific settings for optimization, e.g. weight decay for BatchNorm layers. The users can do those fine-grained parameter tuning through customizing optimizer constructor.

```
from mmcv.utils import build_from_cfg

from mmcv.runner.optimizer import OPTIMIZER_BUILDERS, OPTIMIZERS
from mmdet.utils import get_root_logger
from .my_optimizer import MyOptimizer


@OPTIMIZER_BUILDERS.register_module()
class MyOptimizerConstructor(object):

    def __init__(self, optimizer_cfg, paramwise_cfg=None):

    def __call__(self, model):
```

(continues on next page)

(continued from previous page)

```
        return my_optimizer
```

The default optimizer constructor is implemented here, which could also serve as a template for new optimizer constructor.

12.1.4 Additional settings

Tricks not implemented by the optimizer should be implemented through optimizer constructor (e.g., set parameter-wise learning rates) or hooks. We list some common settings that could stabilize the training or accelerate the training. Feel free to create PR, issue for more settings.

- **Use gradient clip to stabilize training**: Some models need gradient clip to clip the gradients to stabilize the training process. An example is as below:

```
optimizer_config = dict(
    _delete_=True, grad_clip=dict(max_norm=35, norm_type=2))
```

  If your config inherits the base config which already sets the `optimizer_config`, you might need `_delete_=True` to override the unnecessary settings. See the config documentation for more details.

- **Use momentum schedule to accelerate model convergence**: We support momentum scheduler to modify model's momentum according to learning rate, which could make the model converge in a faster way. Momentum scheduler is usually used with LR scheduler, for example, the following config is used in 3D detection to accelerate convergence. For more details, please refer to the implementation of CyclicLrUpdater and CyclicMomentumUpdater.

```
lr_config = dict(
    policy='cyclic',
    target_ratio=(10, 1e-4),
    cyclic_times=1,
    step_ratio_up=0.4,
)
momentum_config = dict(
    policy='cyclic',
    target_ratio=(0.85 / 0.95, 1),
    cyclic_times=1,
    step_ratio_up=0.4,
)
```

12.2 Customize training schedules

By default we use step learning rate with 1x schedule, this calls `StepLRHook` in MMCV. We support many other learning rate schedule here, such as `CosineAnnealing` and `Poly` schedule. Here are some examples

- Poly schedule:

```
lr_config = dict(policy='poly', power=0.9, min_lr=1e-4, by_epoch=False)
```

- ConsineAnnealing schedule:

```
lr_config = dict(
    policy='CosineAnnealing',
    warmup='linear',
    warmup_iters=1000,
    warmup_ratio=1.0 / 10,
    min_lr_ratio=1e-5)
```

12.3 Customize workflow

Workflow is a list of (phase, epochs) to specify the running order and epochs. By default it is set to be

```
workflow = [('train', 1)]
```

which means running 1 epoch for training. Sometimes user may want to check some metrics (e.g. loss, accuracy) about the model on the validate set. In such case, we can set the workflow as

```
[('train', 1), ('val', 1)]
```

so that 1 epoch for training and 1 epoch for validation will be run iteratively.

Note:

1. The parameters of model will not be updated during val epoch.
2. Keyword `total_epochs` in the config only controls the number of training epochs and will not affect the validation workflow.
3. Workflows `[('train', 1), ('val', 1)]` and `[('train', 1)]` will not change the behavior of `EvalHook` because `EvalHook` is called by `after_train_epoch` and validation workflow only affect hooks that are called through `after_val_epoch`. Therefore, the only difference between `[('train', 1), ('val', 1)]` and `[('train', 1)]` is that the runner will calculate losses on validation set after each training epoch.

12.4 Customize hooks

12.4.1 Customize self-implemented hooks

1. Implement a new hook

There are some occasions when the users might need to implement a new hook. MMDetection supports customized hooks in training (#3395) since v2.3.0. Thus the users could implement a hook directly in mmdet or their mmdet-based codebases and use the hook by only modifying the config in training. Before v2.3.0, the users need to modify the code to get the hook registered before training starts. Here we give an example of creating a new hook in mmdet and using it in training.

```
from mmcv.runner import HOOKS, Hook


@HOOKS.register_module()
class MyHook(Hook):

    def __init__(self, a, b):
```

(continues on next page)

(continued from previous page)

```
        pass

    def before_run(self, runner):
        pass

    def after_run(self, runner):
        pass

    def before_epoch(self, runner):
        pass

    def after_epoch(self, runner):
        pass

    def before_iter(self, runner):
        pass

    def after_iter(self, runner):
        pass
```

Depending on the functionality of the hook, the users need to specify what the hook will do at each stage of the training in `before_run`, `after_run`, `before_epoch`, `after_epoch`, `before_iter`, and `after_iter`.

2. Register the new hook

Then we need to make `MyHook` imported. Assuming the file is in `mmdet/core/utils/my_hook.py` there are two ways to do that:

- Modify `mmdet/core/utils/__init__.py` to import it.

  The newly defined module should be imported in `mmdet/core/utils/__init__.py` so that the registry will find the new module and add it:

```
from .my_hook import MyHook
```

- Use `custom_imports` in the config to manually import it

```
custom_imports = dict(imports=['mmdet.core.utils.my_hook'], allow_failed_imports=False)
```

3. Modify the config

```
custom_hooks = [
    dict(type='MyHook', a=a_value, b=b_value)
]
```

You can also set the priority of the hook by adding key `priority` to `'NORMAL'` or `'HIGHEST'` as below

```
custom_hooks = [
    dict(type='MyHook', a=a_value, b=b_value, priority='NORMAL')
]
```

By default the hook's priority is set as `NORMAL` during registration.

12.4.2 Use hooks implemented in MMCV

If the hook is already implemented in MMCV, you can directly modify the config to use the hook as below

4. Example: `NumClassCheckHook`

We implement a customized hook named NumClassCheckHook to check whether the `num_classes` in head matches the length of `CLASSSES` in `dataset`.

We set it in default_runtime.py.

```
custom_hooks = [dict(type='NumClassCheckHook')]
```

12.4.3 Modify default runtime hooks

There are some common hooks that are not registered through `custom_hooks`, they are

- log_config
- checkpoint_config
- evaluation
- lr_config
- optimizer_config
- momentum_config

In those hooks, only the logger hook has the `VERY_LOW` priority, others' priority are `NORMAL`. The above-mentioned tutorials already covers how to modify `optimizer_config`, `momentum_config`, and `lr_config`. Here we reveals how what we can do with `log_config`, `checkpoint_config`, and `evaluation`.

Checkpoint config

The MMCV runner will use `checkpoint_config` to initialize `CheckpointHook`.

```
checkpoint_config = dict(interval=1)
```

The users could set `max_keep_ckpts` to only save only small number of checkpoints or decide whether to store state dict of optimizer by `save_optimizer`. More details of the arguments are here

Log config

The `log_config` wraps multiple logger hooks and enables to set intervals. Now MMCV supports `WandbLoggerHook`, `MlflowLoggerHook`, and `TensorboardLoggerHook`. The detail usages can be found in the doc.

```
log_config = dict(
    interval=50,
    hooks=[
        dict(type='TextLoggerHook'),
        dict(type='TensorboardLoggerHook')
    ])
```

Evaluation config

The config of `evaluation` will be used to initialize the `EvalHook`. Except the key `interval`, other arguments such as `metric` will be passed to the `dataset.evaluate()`

```
evaluation = dict(interval=1, metric='bbox')
```

CHAPTER

THIRTEEN

TUTORIAL 6: CUSTOMIZE LOSSES

MMDetection provides users with different loss functions. But the default configuration may be not applicable for different datasets or models, so users may want to modify a specific loss to adapt the new situation.

This tutorial first elaborate the computation pipeline of losses, then give some instructions about how to modify each step. The modification can be categorized as tweaking and weighting.

13.1 Computation pipeline of a loss

Given the input prediction and target, as well as the weights, a loss function maps the input tensor to the final loss scalar. The mapping can be divided into four steps:

1. Set the sampling method to sample positive and negative samples.
2. Get **element-wise** or **sample-wise** loss by the loss kernel function.
3. Weighting the loss with a weight tensor **element-wisely**.
4. Reduce the loss tensor to a **scalar**.
5. Weighting the loss with a **scalar**.

13.2 Set sampling method (step 1)

For some loss functions, sampling strategies are needed to avoid imbalance between positive and negative samples.

For example, when using `CrossEntropyLoss` in RPN head, we need to set `RandomSampler` in `train_cfg`

```
train_cfg=dict(
    rpn=dict(
        sampler=dict(
            type='RandomSampler',
            num=256,
            pos_fraction=0.5,
            neg_pos_ub=-1,
            add_gt_as_proposals=False))
```

For some other losses which have positive and negative sample balance mechanism such as Focal Loss, GHMC, and QualityFocalLoss, the sampler is no more necessary.

13.3 Tweaking loss

Tweaking a loss is more related with step 2, 4, 5, and most modifications can be specified in the config. Here we take Focal Loss (FL) as an example. The following code sniper are the construction method and config of FL respectively, they are actually one to one correspondence.

```
@LOSSES.register_module()
class FocalLoss(nn.Module):

    def __init__(self,
                 use_sigmoid=True,
                 gamma=2.0,
                 alpha=0.25,
                 reduction='mean',
                 loss_weight=1.0):
```

```
loss_cls=dict(
    type='FocalLoss',
    use_sigmoid=True,
    gamma=2.0,
    alpha=0.25,
    loss_weight=1.0)
```

13.3.1 Tweaking hyper-parameters (step 2)

`gamma` and `beta` are two hyper-parameters in the Focal Loss. Say if we want to change the value of `gamma` to be 1.5 and `alpha` to be 0.5, then we can specify them in the config as follows:

```
loss_cls=dict(
    type='FocalLoss',
    use_sigmoid=True,
    gamma=1.5,
    alpha=0.5,
    loss_weight=1.0)
```

13.3.2 Tweaking the way of reduction (step 3)

The default way of reduction is `mean` for FL. Say if we want to change the reduction from `mean` to `sum`, we can specify it in the config as follows:

```
loss_cls=dict(
    type='FocalLoss',
    use_sigmoid=True,
    gamma=2.0,
    alpha=0.25,
    loss_weight=1.0,
    reduction='sum')
```

13.3.3 Tweaking loss weight (step 5)

The loss weight here is a scalar which controls the weight of different losses in multi-task learning, e.g. classification loss and regression loss. Say if we want to change to loss weight of classification loss to be 0.5, we can specify it in the config as follows:

```
loss_cls=dict(
    type='FocalLoss',
    use_sigmoid=True,
    gamma=2.0,
    alpha=0.25,
    loss_weight=0.5)
```

13.4 Weighting loss (step 3)

Weighting loss means we re-weight the loss element-wisely. To be more specific, we multiply the loss tensor with a weight tensor which has the same shape. As a result, different entries of the loss can be scaled differently, and so called element-wisely. The loss weight varies across different models and highly context related, but overall there are two kinds of loss weights, `label_weights` for classification loss and `bbox_weights` for bbox regression loss. You can find them in the `get_target` method of the corresponding head. Here we take ATSSHead as an example, which inherit AnchorHead but overwrite its `get_targets` method which yields different `label_weights` and `bbox_weights`.

```
class ATSSHead(AnchorHead):

    ...

    def get_targets(self,
                    anchor_list,
                    valid_flag_list,
                    gt_bboxes_list,
                    img_metas,
                    gt_bboxes_ignore_list=None,
                    gt_labels_list=None,
                    label_channels=1,
                    unmap_outputs=True):
```

CHAPTER

FOURTEEN

TUTORIAL 7: FINETUNING MODELS

Detectors pre-trained on the COCO dataset can serve as a good pre-trained model for other datasets, e.g., CityScapes and KITTI Dataset. This tutorial provides instruction for users to use the models provided in the *Model Zoo* for other datasets to obtain better performance.

There are two steps to finetune a model on a new dataset.

- Add support for the new dataset following Tutorial 2: Customize Datasets.
- Modify the configs as will be discussed in this tutorial.

Take the finetuning process on Cityscapes Dataset as an example, the users need to modify five parts in the config.

14.1 Inherit base configs

To release the burden and reduce bugs in writing the whole configs, MMDetection V2.0 support inheriting configs from multiple existing configs. To finetune a Mask RCNN model, the new config needs to inherit `_base_/models/mask_rcnn_r50_fpn.py` to build the basic structure of the model. To use the Cityscapes Dataset, the new config can also simply inherit `_base_/datasets/cityscapes_instance.py`. For runtime settings such as training schedules, the new config needs to inherit `_base_/default_runtime.py`. This configs are in the `configs` directory and the users can also choose to write the whole contents rather than use inheritance.

```
_base_ = [
    '../_base_/models/mask_rcnn_r50_fpn.py',
    '../_base_/datasets/cityscapes_instance.py', '../_base_/default_runtime.py'
]
```

14.2 Modify head

Then the new config needs to modify the head according to the class numbers of the new datasets. By only changing `num_classes` in the roi_head, the weights of the pre-trained models are mostly reused except the final prediction head.

```
model = dict(
    pretrained=None,
    roi_head=dict(
        bbox_head=dict(
            type='Shared2FCBBoxHead',
            in_channels=256,
            fc_out_channels=1024,
            roi_feat_size=7,
```

(continues on next page)

(continued from previous page)

```
            num_classes=8,
            bbox_coder=dict(
                type='DeltaXYWHBBoxCoder',
                target_means=[0., 0., 0., 0.],
                target_stds=[0.1, 0.1, 0.2, 0.2]),
            reg_class_agnostic=False,
            loss_cls=dict(
                type='CrossEntropyLoss', use_sigmoid=False, loss_weight=1.0),
            loss_bbox=dict(type='SmoothL1Loss', beta=1.0, loss_weight=1.0)),
        mask_head=dict(
            type='FCNMaskHead',
            num_convs=4,
            in_channels=256,
            conv_out_channels=256,
            num_classes=8,
            loss_mask=dict(
                type='CrossEntropyLoss', use_mask=True, loss_weight=1.0))))
```

14.3 Modify dataset

The users may also need to prepare the dataset and write the configs about dataset. MMDetection V2.0 already support VOC, WIDER FACE, COCO and Cityscapes Dataset.

14.4 Modify training schedule

The finetuning hyperparameters vary from the default schedule. It usually requires smaller learning rate and less training epochs

```
# optimizer
# lr is set for a batch size of 8
optimizer = dict(type='SGD', lr=0.01, momentum=0.9, weight_decay=0.0001)
optimizer_config = dict(grad_clip=None)
# learning policy
lr_config = dict(
    policy='step',
    warmup='linear',
    warmup_iters=500,
    warmup_ratio=0.001,
    step=[7])
# the max_epochs and step in lr_config need specifically tuned for the customized dataset
runner = dict(max_epochs=8)
log_config = dict(interval=100)
```

14.5 Use pre-trained model

To use the pre-trained model, the new config add the link of pre-trained models in the `load_from`. The users might need to download the model weights before training to avoid the download time during training.

```
load_from = 'https://download.openmmlab.com/mmdetection/v2.0/mask_rcnn/mask_rcnn_r50_
↪caffe_fpn_mstrain-poly_3x_coco/mask_rcnn_r50_caffe_fpn_mstrain-poly_3x_coco_bbox_mAP-0.
↪408__segm_mAP-0.37_20200504_163245-42aa3d00.pth'  # noqa
```

CHAPTER

FIFTEEN

TUTORIAL 8: PYTORCH TO ONNX (EXPERIMENTAL)

15.1 How to convert models from Pytorch to ONNX

15.1.1 Prerequisite

1. Install the prerequisites following *get_started.md/Prepare environment*.
2. Build custom operators for ONNX Runtime and install MMCV manually following How to build custom operators for ONNX Runtime
3. Install MMdetection manually following steps 2-3 in *get_started.md/Install MMdetection*.

15.1.2 Usage

```
python tools/deployment/pytorch2onnx.py \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    --output-file ${OUTPUT_FILE} \
    --input-img ${INPUT_IMAGE_PATH} \
    --shape ${IMAGE_SHAPE} \
    --test-img ${TEST_IMAGE_PATH} \
    --opset-version ${OPSET_VERSION} \
    --cfg-options ${CFG_OPTIONS}
    --dynamic-export \
    --show \
    --verify \
    --simplify \
```

15.1.3 Description of all arguments

- `config` : The path of a model config file.
- `checkpoint` : The path of a model checkpoint file.
- `--output-file`: The path of output ONNX model. If not specified, it will be set to `tmp.onnx`.
- `--input-img`: The path of an input image for tracing and conversion. By default, it will be set to `tests/data/color.jpg`.
- `--shape`: The height and width of input tensor to the model. If not specified, it will be set to `800 1216`.
- `--test-img` : The path of an image to verify the exported ONNX model. By default, it will be set to `None`, meaning it will use `--input-img` for verification.
- `--opset-version` : The opset version of ONNX. If not specified, it will be set to `11`.
- `--dynamic-export`: Determines whether to export ONNX model with dynamic input and output shapes. If not specified, it will be set to `False`.
- `--show`: Determines whether to print the architecture of the exported model and whether to show detection outputs when `--verify` is set to `True`. If not specified, it will be set to `False`.
- `--verify`: Determines whether to verify the correctness of an exported model. If not specified, it will be set to `False`.
- `--simplify`: Determines whether to simplify the exported ONNX model. If not specified, it will be set to `False`.
- `--cfg-options`: Override some settings in the used config file, the key-value pair in `xxx=yyy` format will be merged into config file.
- `--skip-postprocess`: Determines whether export model without post process. If not specified, it will be set to `False`. Notice: This is an experimental option. Only work for some single stage models. Users need to implement the post-process by themselves. We do not guarantee the correctness of the exported model.

Example:

```
python tools/deployment/pytorch2onnx.py \
    configs/yolo/yolov3_d53_mstrain-608_273e_coco.py \
    checkpoints/yolo/yolov3_d53_mstrain-608_273e_coco.pth \
```

(continues on next page)

(continued from previous page)

```
    --output-file checkpoints/yolo/yolov3_d53_mstrain-608_273e_coco.onnx \
    --input-img demo/demo.jpg \
    --test-img tests/data/color.jpg \
    --shape 608 608 \
    --show \
    --verify \
    --dynamic-export \
    --cfg-options \
      model.test_cfg.deploy_nms_pre=-1 \
```

15.2 How to evaluate the exported models

We prepare a tool `tools/deplopyment/test.py` to evaluate ONNX models with ONNXRuntime and TensorRT.

15.2.1 Prerequisite

- Install onnx and onnxruntime (CPU version)

```
pip install onnx onnxruntime==1.5.1
```

- If you want to run the model on GPU, please remove the CPU version before using the GPU version.

```
pip uninstall onnxruntime
pip install onnxruntime-gpu
```

  Note: onnxruntime-gpu is version-dependent on CUDA and CUDNN, please ensure that your environment meets the requirements.

- Build custom operators for ONNX Runtime following How to build custom operators for ONNX Runtime
- Install TensorRT by referring to How to build TensorRT plugins in MMCV (optional)

15.2.2 Usage

```
python tools/deployment/test.py \
    ${CONFIG_FILE} \
    ${MODEL_FILE} \
    --out ${OUTPUT_FILE} \
    --backend ${BACKEND} \
    --format-only ${FORMAT_ONLY} \
    --eval ${EVALUATION_METRICS} \
    --show-dir ${SHOW_DIRECTORY} \
    ----show-score-thr ${SHOW_SCORE_THRESHOLD} \
    ----cfg-options ${CFG_OPTIONS} \
    ----eval-options ${EVALUATION_OPTIONS} \
```

15.2.3 Description of all arguments

- `config`: The path of a model config file.
- `model`: The path of an input model file.
- `--out`: The path of output result file in pickle format.
- `--backend`: Backend for input model to run and should be `onnxruntime` or `tensorrt`.
- `--format-only` : Format the output results without perform evaluation. It is useful when you want to format the result to a specific format and submit it to the test server. If not specified, it will be set to `False`.
- `--eval`: Evaluation metrics, which depends on the dataset, e.g., "bbox", "segm", "proposal" for COCO, and "mAP", "recall" for PASCAL VOC.
- `--show-dir`: Directory where painted images will be saved
- `--show-score-thr`: Score threshold. Default is set to `0.3`.
- `--cfg-options`: Override some settings in the used config file, the key-value pair in `xxx=yyy` format will be merged into config file.
- `--eval-options`: Custom options for evaluation, the key-value pair in `xxx=yyy` format will be kwargs for `dataset.evaluate()` function

Notes:

- If the deployed backend platform is TensorRT, please add environment variables before running the file:

```
export ONNX_BACKEND=MMCVTensorRT
```

- If you want to use the `--dynamic-export` parameter in the TensorRT backend to export ONNX, please remove the `--simplify` parameter, and vice versa.

15.2.4 Results and Models

Notes:

- All ONNX models are evaluated with dynamic shape on coco dataset and images are preprocessed according to the original config file. Note that CornerNet is evaluated without test-time flip, since currently only single-scale evaluation is supported with ONNX Runtime.
- Mask AP of Mask R-CNN drops by 1% for ONNXRuntime. The main reason is that the predicted masks are directly interpolated to original image in PyTorch, while they are at first interpolated to the preprocessed input image of the model and then to original image in other backend.

15.3 List of supported models exportable to ONNX

The table below lists the models that are guaranteed to be exportable to ONNX and runnable in ONNX Runtime.

Notes:

- Minimum required version of MMCV is `1.3.5`
- *All models above are tested with Pytorch==1.6.0 and onnxruntime==1.5.1*, except for CornerNet. For more details about the torch version when exporting CornerNet to ONNX, which involves `mmcv::cummax`, please refer to the Known Issues in mmcv.

- Though supported, it is *not recommended* to use batch inference in onnxruntime for `DETR`, because there is huge performance gap between ONNX and torch model (e.g. 33.5 vs 39.9 mAP on COCO for onnxruntime and torch respectively, with a batch size 2). The main reason for the gap is that these is non-negligible effect on the predicted regressions during batch inference for ONNX, since the predicted coordinates is normalized by `img_shape` (without padding) and should be converted to absolute format, but `img_shape` is not dynamically traceable thus the padded `img_shape_for_onnx` is used.
- Currently only single-scale evaluation is supported with ONNX Runtime, also `mmcv::SoftNonMaxSuppression` is only supported for single image by now.

15.4 The Parameters of Non-Maximum Suppression in ONNX Export

In the process of exporting the ONNX model, we set some parameters for the NMS op to control the number of output bounding boxes. The following will introduce the parameter setting of the NMS op in the supported models. You can set these parameters through `--cfg-options`.

- `nms_pre`: The number of boxes before NMS. The default setting is `1000`.
- `deploy_nms_pre`: The number of boxes before NMS when exporting to ONNX model. The default setting is `0`.
- `max_per_img`: The number of boxes to be kept after NMS. The default setting is `100`.
- `max_output_boxes_per_class`: Maximum number of output boxes per class of NMS. The default setting is `200`.

15.5 Reminders

- When the input model has custom op such as `RoIAlign` and if you want to verify the exported ONNX model, you may have to build `mmcv` with ONNXRuntime from source.
- `mmcv.onnx.simplify` feature is based on onnx-simplifier. If you want to try it, please refer to onnx in `mmcv` and onnxruntime op in `mmcv` for more information.
- If you meet any problem with the listed models above, please create an issue and it would be taken care of soon. For models not included in the list, please try to dig a little deeper and debug a little bit more and hopefully solve them by yourself.
- Because this feature is experimental and may change fast, please always try with the latest `mmcv` and `mmdetecion`.

15.6 FAQs

- None

CHAPTER

SIXTEEN

TUTORIAL 9: ONNX TO TENSORRT (EXPERIMENTAL)

16.1 How to convert models from ONNX to TensorRT

16.1.1 Prerequisite

1. Please refer to get_started.md for installation of MMCV and MMDetection from source.
2. Please refer to ONNXRuntime in mmcv and TensorRT plugin in mmcv to install `mmcv-full` with ONNXRuntime custom ops and TensorRT plugins.
3. Use our tool pytorch2onnx to convert the model from PyTorch to ONNX.

16.1.2 Usage

```
python tools/deployment/onnx2tensorrt.py \
    ${CONFIG} \
    ${MODEL} \
    --trt-file ${TRT_FILE} \
    --input-img ${INPUT_IMAGE_PATH} \
    --shape ${INPUT_IMAGE_SHAPE} \
    --min-shape ${MIN_IMAGE_SHAPE} \
    --max-shape ${MAX_IMAGE_SHAPE} \
    --workspace-size {WORKSPACE_SIZE} \
    --show \
    --verify \
```

Description of all arguments:

- `config` : The path of a model config file.
- `model` : The path of an ONNX model file.
- `--trt-file`: The Path of output TensorRT engine file. If not specified, it will be set to `tmp.trt`.
- `--input-img` : The path of an input image for tracing and conversion. By default, it will be set to `demo/demo.jpg`.
- `--shape`: The height and width of model input. If not specified, it will be set to `400 600`.
- `--min-shape`: The minimum height and width of model input. If not specified, it will be set to the same as `--shape`.
- `--max-shape`: The maximum height and width of model input. If not specified, it will be set to the same as `--shape`.
- `--workspace-size` : The required GPU workspace size in GiB to build TensorRT engine. If not specified, it will be set to 1 GiB.
- `--show`: Determines whether to show the outputs of the model. If not specified, it will be set to `False`.
- `--verify`: Determines whether to verify the correctness of models between ONNXRuntime and TensorRT. If not specified, it will be set to `False`.
- `--verbose`: Determines whether to print logging messages. It's useful for debugging. If not specified, it will be set to `False`.

Example:

```
python tools/deployment/onnx2tensorrt.py \
    configs/retinanet/retinanet_r50_fpn_1x_coco.py \
    checkpoints/retinanet_r50_fpn_1x_coco.onnx \
    --trt-file checkpoints/retinanet_r50_fpn_1x_coco.trt \
    --input-img demo/demo.jpg \
    --shape 400 600 \
    --show \
    --verify \
```

16.2 How to evaluate the exported models

We prepare a tool `tools/deplopyment/test.py` to evaluate TensorRT models.

Please refer to following links for more information.

- how-to-evaluate-the-exported-models
- results-and-models

16.3 List of supported models convertible to TensorRT

The table below lists the models that are guaranteed to be convertible to TensorRT.

Notes:

- *All models above are tested with Pytorch==1.6.0, onnx==1.7.0 and TensorRT-7.2.1.6.Ubuntu-16.04.x86_64-gnu.cuda-10.2.cudnn8.0*

16.4 Reminders

- If you meet any problem with the listed models above, please create an issue and it would be taken care of soon. For models not included in the list, we may not provide much help here due to the limited resources. Please try to dig a little deeper and debug by yourself.
- Because this feature is experimental and may change fast, please always try with the latest `mmcv` and `mmdetecion`.

16.5 FAQs

- None

CHAPTER

SEVENTEEN

TUTORIAL 10: WEIGHT INITIALIZATION

During training, a proper initialization strategy is beneficial to speeding up the training or obtaining a higher performance. MMCV provide some commonly used methods for initializing modules like `nn.Conv2d`. Model initialization in MMdetection mainly uses `init_cfg`. Users can initialize models with following two steps:

1. Define `init_cfg` for a model or its components in `model_cfg`, but `init_cfg` of children components have higher priority and will override `init_cfg` of parents modules.
2. Build model as usual, but call `model.init_weights()` method explicitly, and model parameters will be initialized as configuration.

The high-level workflow of initialization in MMdetection is :

model_cfg(init_cfg) -> build_from_cfg -> model -> init_weight() -> initialize(self, self.init_cfg) -> children's init_weight()

17.1 Description

It is dict or list[dict], and contains the following keys and values:

- `type` (str), containing the initializer name in `INTIALIZERS`, and followed by arguments of the initializer.
- `layer` (str or list[str]), containing the names of basiclayers in Pytorch or MMCV with learnable parameters that will be initialized, e.g. `'Conv2d'`,`'DeformConv2d'`.
- `override` (dict or list[dict]), containing the sub-modules that not inherit from BaseModule and whose initialization configuration is different from other layers' which are in `'layer'` key. Initializer defined in `type` will work for all layers defined in `layer`, so if sub-modules are not derived Classes of `BaseModule` but can be initialized as same ways of layers in `layer`, it does not need to use `override`. `override` contains:
  - `type` followed by arguments of initializer;
  - `name` to indicate sub-module which will be initialized.

17.2 Initialize parameters

Inherit a new model from `mmcv.runner.BaseModule` or `mmdet.models` Here we show an example of FooModel.

```
import torch.nn as nn
from mmcv.runner import BaseModule

class FooModel(BaseModule)
        def __init__(self,
```

(continues on next page)

(continued from previous page)

```
                arg1,
                arg2,
                init_cfg=None):
            super(FooModel, self).__init__(init_cfg)
                ...
```

- Initialize model by using `init_cfg` directly in code

```
import torch.nn as nn
from mmcv.runner import BaseModule
# or directly inherit mmdet models

class FooModel(BaseModule)
        def __init__(self,
                arg1,
                arg2,
                init_cfg=XXX):
                super(FooModel, self).__init__(init_cfg)
            ...
```

- Initialize model by using `init_cfg` directly in `mmcv.Sequential` or `mmcv.ModuleList` code

```
from mmcv.runner import BaseModule, ModuleList

class FooModel(BaseModule)
        def __init__(self,
                    arg1,
                    arg2,
                    init_cfg=None):
                super(FooModel, self).__init__(init_cfg)
            ...
            self.conv1 = ModuleList(init_cfg=XXX)
```

- Initialize model by using `init_cfg` in config file

```
model = dict(
        ...
        model = dict(
            type='FooModel',
            arg1=XXX,
            arg2=XXX,
            init_cfg=XXX),
        ...
```

17.3 Usage of init_cfg

1. Initialize model by `layer` key

   If we only define `layer`, it just initialize the layer in `layer` key.

   NOTE: Value of `layer` key is the class name with attributes weights and bias of Pytorch, (so such as `MultiheadAttention layer` is not supported).

- Define `layer` key for initializing module with same configuration.

```
init_cfg = dict(type='Constant', layer=['Conv1d', 'Conv2d', 'Linear'], val=1)
# initialize whole module with same configuration
```

- Define `layer` key for initializing layer with different configurations.

```
init_cfg = [dict(type='Constant', layer='Conv1d', val=1),
            dict(type='Constant', layer='Conv2d', val=2),
            dict(type='Constant', layer='Linear', val=3)]
# nn.Conv1d will be initialized with dict(type='Constant', val=1)
# nn.Conv2d will be initialized with dict(type='Constant', val=2)
# nn.Linear will be initialized with dict(type='Constant', val=3)
```

1. Initialize model by `override` key

- When initializing some specific part with its attribute name, we can use `override` key, and the value in `override` will ignore the value in init_cfg.

```
# layers
# self.feat = nn.Conv1d(3, 1, 3)
# self.reg = nn.Conv2d(3, 3, 3)
# self.cls = nn.Linear(1,2)

init_cfg = dict(type='Constant',
                layer=['Conv1d','Conv2d'], val=1, bias=2,
                override=dict(type='Constant', name='reg', val=3, bias=4))
# self.feat and self.cls will be initialized with         dict(type='Constant', val=1,
↪ bias=2)
# The module called 'reg' will be initialized with dict(type='Constant', val=3, bias=4)
```

- If `layer` is None in init_cfg, only sub-module with the name in override will be initialized, and type and other args in override can be omitted.

```
# layers
# self.feat = nn.Conv1d(3, 1, 3)
# self.reg = nn.Conv2d(3, 3, 3)
# self.cls = nn.Linear(1,2)

init_cfg = dict(type='Constant', val=1, bias=2,         override=dict(name='reg'))

# self.feat and self.cls will be initialized by Pytorch
# The module called 'reg' will be initialized with dict(type='Constant', val=1, bias=2)
```

- If we don't define `layer` key or `override` key, it will not initialize anything.
- Invalid usage

```
# It is invalid that override don't have name key
init_cfg = dict(type='Constant', layer ['Conv1d','Conv2d'], val=1, bias=2,
                    override=dict(type='Constant', val=3, bias=4))

# It is also invalid that override has name and other args except type
init_cfg = dict(type='Constant', layer ['Conv1d','Conv2d'], val=1, bias=2,
                override=dict(name='reg', val=3, bias=4))
```

1. Initialize model with the pretrained model

```
init_cfg = dict(type='Pretrained',
            checkpoint='torchvision://resnet50')
```

More details can refer to the documentation in MMCV and MMCV PR #780

Apart from training/testing scripts, We provide lots of useful tools under the `tools/` directory.

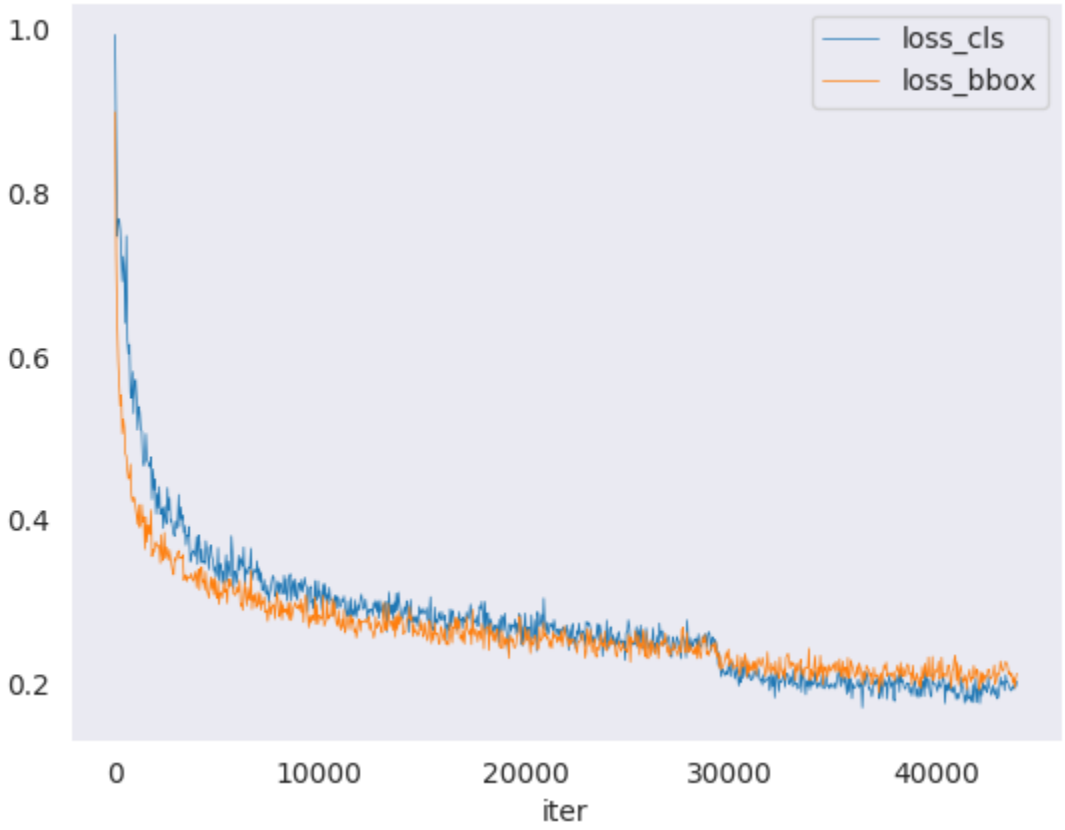
CHAPTER

EIGHTEEN

LOG ANALYSIS

`tools/analysis_tools/analyze_logs.py` plots loss/mAP curves given a training log file. Run `pip install seaborn` first to install the dependency.

```
python tools/analysis_tools/analyze_logs.py plot_curve [--keys ${KEYS}] [--title ${TITLE}
↪] [--legend ${LEGEND}] [--backend ${BACKEND}] [--style ${STYLE}] [--out ${OUT_FILE}]
```



loss curve image

Examples:

- Plot the classification loss of some run.

```
python tools/analysis_tools/analyze_logs.py plot_curve log.json --keys loss_cls --
↪legend loss_cls
```

- Plot the classification and regression loss of some run, and save the figure to a pdf.

```
python tools/analysis_tools/analyze_logs.py plot_curve log.json --keys loss_cls␣
↪loss_bbox --out losses.pdf
```

- Compare the bbox mAP of two runs in the same figure.

```
python tools/analysis_tools/analyze_logs.py plot_curve log1.json log2.json --keys␣
↪bbox_mAP --legend run1 run2
```

- Compute the average training speed.

```
python tools/analysis_tools/analyze_logs.py cal_train_time log.json [--include-
↪outliers]
```

The output is expected to be like the following.

```
-----Analyze train time of work_dirs/some_exp/20190611_192040.log.json-----
slowest epoch 11, average time is 1.2024
fastest epoch 1, average time is 1.1909
time std over epochs is 0.0028
average iter time: 1.1959 s/iter
```

CHAPTER

NINETEEN

RESULT ANALYSIS

`tools/analysis_tools/analyze_results.py` calculates single image mAP and saves or shows the topk images with the highest and lowest scores based on prediction results.

Usage

```
python tools/analysis_tools/analyze_results.py \
      ${CONFIG} \
      ${PREDICTION_PATH} \
      ${SHOW_DIR} \
      [--show] \
      [--wait-time ${WAIT_TIME}] \
      [--topk ${TOPK}] \
      [--show-score-thr ${SHOW_SCORE_THR}] \
      [--cfg-options ${CFG_OPTIONS}]
```

Description of all arguments:

- `config` : The path of a model config file.
- `prediction_path`: Output result file in pickle format from `tools/test.py`
- `show_dir`: Directory where painted GT and detection images will be saved
- `--show`Determines whether to show painted images, If not specified, it will be set to `False`
- `--wait-time`: The interval of show (s), 0 is block
- `--topk`: The number of saved images that have the highest and lowest `topk` scores after sorting. If not specified, it will be set to `20`.
- `--show-score-thr`: Show score threshold. If not specified, it will be set to `0`.
- `--cfg-options`: If specified, the key-value pair optional cfg will be merged into config file

Examples:

Assume that you have got result file in pickle format from `tools/test.py` in the path './result.pkl'.

1. Test Faster R-CNN and visualize the results, save images to the directory `results/`

```
python tools/analysis_tools/analyze_results.py \
       configs/faster_rcnn/faster_rcnn_r50_fpn_1x_coco.py \
       result.pkl \
       results \
       --show
```

1. Test Faster R-CNN and specified topk to 50, save images to the directory `results/`

```
python tools/analysis_tools/analyze_results.py \
       configs/faster_rcnn/faster_rcnn_r50_fpn_1x_coco.py \
       result.pkl \
       results \
       --topk 50
```

1. If you want to filter the low score prediction results, you can specify the `show-score-thr` parameter

```
python tools/analysis_tools/analyze_results.py \
       configs/faster_rcnn/faster_rcnn_r50_fpn_1x_coco.py \
       result.pkl \
       results \
       --show-score-thr 0.3
```

CHAPTER

TWENTY

VISUALIZATION

20.1 Visualize Datasets

`tools/misc/browse_dataset.py` helps the user to browse a detection dataset (both images and bounding box annotations) visually, or save the image to a designated directory.

```
python tools/misc/browse_dataset.py ${CONFIG} [-h] [--skip-type ${SKIP_TYPE[SKIP_TYPE...
↪]}] [--output-dir ${OUTPUT_DIR}] [--not-show] [--show-interval ${SHOW_INTERVAL}]
```

20.2 Visualize Models

First, convert the model to ONNX as described *here*. Note that currently only RetinaNet is supported, support for other models will be coming in later versions. The converted model could be visualized by tools like Netron.

20.3 Visualize Predictions

If you need a lightweight GUI for visualizing the detection results, you can refer DetVisGUI project.

CHAPTER

TWENTYONE

ERROR ANALYSIS

`tools/analysis_tools/coco_error_analysis.py` analyzes COCO results per category and by different criterion. It can also make a plot to provide useful information.

```
python tools/analysis_tools/coco_error_analysis.py ${RESULT} ${OUT_DIR} [-h] [--ann ${ANN}] [--types ${TYPES[TYPES...]}]
```

Example:

Assume that you have got Mask R-CNN checkpoint file in the path 'checkpoint'. For other checkpoints, please refer to our *model zoo*. You can use the following command to get the results bbox and segmentation json file.

```
# out: results.bbox.json and results.segm.json
python tools/test.py \
       configs/mask_rcnn/mask_rcnn_r50_fpn_1x_coco.py \
       checkpoint/mask_rcnn_r50_fpn_1x_coco_20200205-d4b0c5d6.pth \
       --format-only \
       --options "jsonfile_prefix=./results"
```

1. Get COCO bbox error results per category , save analyze result images to the directory `results/`

```
python tools/analysis_tools/coco_error_analysis.py \
       results.bbox.json \
       results \
       --ann=data/coco/annotations/instances_val2017.json \
```

1. Get COCO segmentation error results per category , save analyze result images to the directory `results/`

```
python tools/analysis_tools/coco_error_analysis.py \
       results.segm.json \
       results \
       --ann=data/coco/annotations/instances_val2017.json \
       --types='segm'
```

CHAPTER

TWENTYTWO

MODEL SERVING

In order to serve an `MMDetection` model with `TorchServe`, you can follow the steps:

22.1 1. Convert model from MMDetection to TorchServe

```
python tools/deployment/mmdet2torchserve.py ${CONFIG_FILE} ${CHECKPOINT_FILE} \
--output-folder ${MODEL_STORE} \
--model-name ${MODEL_NAME}
```

Note: ${MODEL_STORE} needs to be an absolute path to a folder.

22.2 2. Build `mmdet-serve` docker image

```
docker build -t mmdet-serve:latest docker/serve/
```

22.3 3. Run `mmdet-serve`

Check the official docs for running TorchServe with docker.

In order to run in GPU, you need to install nvidia-docker. You can omit the `--gpus` argument in order to run in CPU.

Example:

```
docker run --rm \
--cpus 8 \
--gpus device=0 \
-p8080:8080 -p8081:8081 -p8082:8082 \
--mount type=bind,source=$MODEL_STORE,target=/home/model-server/model-store \
mmdet-serve:latest
```

Read the docs about the Inference (8080), Management (8081) and Metrics (8082) APis

22.4 4. Test deployment

```
curl -O curl -O https://raw.githubusercontent.com/pytorch/serve/master/docs/images/3dogs.
↪jpg
curl http://127.0.0.1:8080/predictions/${MODEL_NAME} -T 3dogs.jpg
```

You should obtain a response similar to:

```
[
  {
    "class_name": "dog",
    "bbox": [
      294.63409423828125,
      203.99111938476562,
      417.048583984375,
      281.62744140625
    ],
    "score": 0.9987992644309998
  },
  {
    "class_name": "dog",
    "bbox": [
      404.26019287109375,
      126.0080795288086,
      574.5091552734375,
      293.6662292480469
    ],
    "score": 0.9979367256164551
  },
  {
    "class_name": "dog",
    "bbox": [
      197.2144775390625,
      93.3067855834961,
      307.8505554199219,
      276.7560119628906
    ],
    "score": 0.993338406085968
  }
]
```

And you can use `test_torchserver.py` to compare result of torchserver and pytorch, and visualize them.

```
python tools/deployment/test_torchserver.py ${IMAGE_FILE} ${CONFIG_FILE} ${CHECKPOINT_
↪FILE} ${MODEL_NAME}
[--inference-addr ${INFERENCE_ADDR}] [--device ${DEVICE}] [--score-thr ${SCORE_THR}]
```

Example:

```
python tools/deployment/test_torchserver.py \
demo/demo.jpg \
configs/yolo/yolov3_d53_320_273e_coco.py \
checkpoint/yolov3_d53_320_273e_coco-421362b6.pth \
```

(continues on next page)

(continued from previous page)

```
yolov3
```

CHAPTER

TWENTYTHREE

MODEL COMPLEXITY

`tools/analysis_tools/get_flops.py` is a script adapted from flops-counter.pytorch to compute the FLOPs and params of a given model.

```
python tools/analysis_tools/get_flops.py ${CONFIG_FILE} [--shape ${INPUT_SHAPE}]
```

You will get the results like this.

```
==============================
Input shape: (3, 1280, 800)
Flops: 239.32 GFLOPs
Params: 37.74 M
==============================
```

Note: This tool is still experimental and we do not guarantee that the number is absolutely correct. You may well use the result for simple comparisons, but double check it before you adopt it in technical reports or papers.

1. FLOPs are related to the input shape while parameters are not. The default input shape is (1, 3, 1280, 800).
2. Some operators are not counted into FLOPs like GN and custom operators. Refer to `mmcv.cnn.get_model_complexity_info()` for details.
3. The FLOPs of two-stage detectors is dependent on the number of proposals.

CHAPTER

TWENTYFOUR

MODEL CONVERSION

24.1 MMDetection model to ONNX (experimental)

We provide a script to convert model to ONNX format. We also support comparing the output results between Pytorch and ONNX model for verification.

```
python tools/deployment/pytorch2onnx.py ${CONFIG_FILE} ${CHECKPOINT_FILE} --output-file $
↪{ONNX_FILE} [--shape ${INPUT_SHAPE} --verify]
```

Note: This tool is still experimental. Some customized operators are not supported for now. For a detailed description of the usage and the list of supported models, please refer to *pytorch2onnx*.

24.2 MMDetection 1.x model to MMDetection 2.x

`tools/model_converters/upgrade_model_version.py` upgrades a previous MMDetection checkpoint to the new version. Note that this script is not guaranteed to work as some breaking changes are introduced in the new version. It is recommended to directly use the new checkpoints.

```
python tools/model_converters/upgrade_model_version.py ${IN_FILE} ${OUT_FILE} [-h] [--
↪num-classes NUM_CLASSES]
```

24.3 RegNet model to MMDetection

`tools/model_converters/regnet2mmdet.py` convert keys in pycls pretrained RegNet models to MMDetection style.

```
python tools/model_converters/regnet2mmdet.py ${SRC} ${DST} [-h]
```

24.4 Detectron ResNet to Pytorch

`tools/model_converters/detectron2pytorch.py` converts keys in the original detectron pretrained ResNet models to PyTorch style.

```
python tools/model_converters/detectron2pytorch.py ${SRC} ${DST} ${DEPTH} [-h]
```

24.5 Prepare a model for publishing

`tools/model_converters/publish_model.py` helps users to prepare their model for publishing.

Before you upload a model to AWS, you may want to

1. convert model weights to CPU tensors
2. delete the optimizer states and
3. compute the hash of the checkpoint file and append the hash id to the filename.

```
python tools/model_converters/publish_model.py ${INPUT_FILENAME} ${OUTPUT_FILENAME}
```

E.g.,

```
python tools/model_converters/publish_model.py work_dirs/faster_rcnn/latest.pth faster_
↪rcnn_r50_fpn_1x_20190801.pth
```

The final output filename will be `faster_rcnn_r50_fpn_1x_20190801-{hash id}.pth`.

CHAPTER

TWENTYFIVE

DATASET CONVERSION

`tools/data_converters/` contains tools to convert the Cityscapes dataset and Pascal VOC dataset to the COCO format.

```
python tools/dataset_converters/cityscapes.py ${CITYSCAPES_PATH} [-h] [--img-dir ${IMG_
↪DIR}] [--gt-dir ${GT_DIR}] [-o ${OUT_DIR}] [--nproc ${NPROC}]
python tools/dataset_converters/pascal_voc.py ${DEVKIT_PATH} [-h] [-o ${OUT_DIR}]
```

CHAPTER

TWENTYSIX

BENCHMARK

26.1 Robust Detection Benchmark

`tools/analysis_tools/test_robustness.py` and`tools/analysis_tools/robustness_eval.py` helps users to evaluate model robustness. The core idea comes from Benchmarking Robustness in Object Detection: Autonomous Driving when Winter is Coming. For more information how to evaluate models on corrupted images and results for a set of standard models please refer to robustness_benchmarking.md.

26.2 FPS Benchmark

`tools/analysis_tools/benchmark.py` helps users to calculate FPS. The FPS value includes model forward and post-processing. In order to get a more accurate value, currently only supports single GPU distributed startup mode.

```
python -m torch.distributed.launch --nproc_per_node=1 --master_port=${PORT} tools/
↪analysis_tools/benchmark.py \
    ${CONFIG} \
    ${CHECKPOINT} \
    [--repeat-num ${REPEAT_NUM}] \
    [--max-iter ${MAX_ITER}] \
    [--log-interval ${LOG_INTERVAL}] \
    --launcher pytorch
```

Examples: Assuming that you have already downloaded the `Faster R-CNN` model checkpoint to the directory `checkpoints/`.

```
python -m torch.distributed.launch --nproc_per_node=1 --master_port=29500 tools/analysis_
↪tools/benchmark.py \
       configs/faster_rcnn/faster_rcnn_r50_fpn_1x_coco.py \
       checkpoints/faster_rcnn_r50_fpn_1x_coco_20200130-047c8118.pth \
       --launcher pytorch
```

CHAPTER

TWENTYSEVEN

MISCELLANEOUS

27.1 Evaluating a metric

`tools/analysis_tools/eval_metric.py` evaluates certain metrics of a pkl result file according to a config file.

```
python tools/analysis_tools/eval_metric.py ${CONFIG} ${PKL_RESULTS} [-h] [--format-only]␣
↪[--eval ${EVAL[EVAL ...]}]
                      [--cfg-options ${CFG_OPTIONS [CFG_OPTIONS ...]}]
                      [--eval-options ${EVAL_OPTIONS [EVAL_OPTIONS ...]}]
```

27.2 Print the entire config

`tools/misc/print_config.py` prints the whole config verbatim, expanding all its imports.

```
python tools/misc/print_config.py ${CONFIG} [-h] [--options ${OPTIONS [OPTIONS...]}]
```

CHAPTER

TWENTYEIGHT

HYPER-PARAMETER OPTIMIZATION

28.1 YOLO Anchor Optimization

`tools/analysis_tools/optimize_anchors.py` provides two method to optimize YOLO anchors.

One is k-means anchor cluster which refers from darknet.

```
python tools/analysis_tools/optimize_anchors.py ${CONFIG} --algorithm k-means --input-
↪shape ${INPUT_SHAPE [WIDTH HEIGHT]} --output-dir ${OUTPUT_DIR}
```

Another is using differential evolution to optimize anchors.

```
python tools/analysis_tools/optimize_anchors.py ${CONFIG} --algorithm differential_
↪evolution --input-shape ${INPUT_SHAPE [WIDTH HEIGHT]} --output-dir ${OUTPUT_DIR}
```

E.g.,

```
python tools/analysis_tools/optimize_anchors.py configs/yolo/yolov3_d53_320_273e_coco.py␣
↪--algorithm differential_evolution --input-shape 608 608 --device cuda --output-dir␣
↪work_dirs
```

You will get:

```
loading annotations into memory...
Done (t=9.70s)
creating index...
index created!
2021-07-19 19:37:20,951 - mmdet - INFO - Collecting bboxes from annotation...
[>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] 117266/117266, 15874.5 task/s,␣
↪elapsed: 7s, ETA:     0s

2021-07-19 19:37:28,753 - mmdet - INFO - Collected 849902 bboxes.
differential_evolution step 1: f(x)= 0.506055
differential_evolution step 2: f(x)= 0.506055
......

differential_evolution step 489: f(x)= 0.386625
2021-07-19 19:46:40,775 - mmdet - INFO Anchor evolution finish. Average IOU: 0.
↪6133754253387451
2021-07-19 19:46:40,776 - mmdet - INFO Anchor differential evolution result:[[10, 12],␣
↪[15, 30], [32, 22], [29, 59], [61, 46], [57, 116], [112, 89], [154, 198], [349, 336]]
2021-07-19 19:46:40,798 - mmdet - INFO Result saved in work_dirs/anchor_optimize_result.
↪json
```

(continues on next page)

(continued from previous page)

CHAPTER

TWENTYNINE

CONFUTION MATRIX

A confusion matrix is a summary of prediction results.

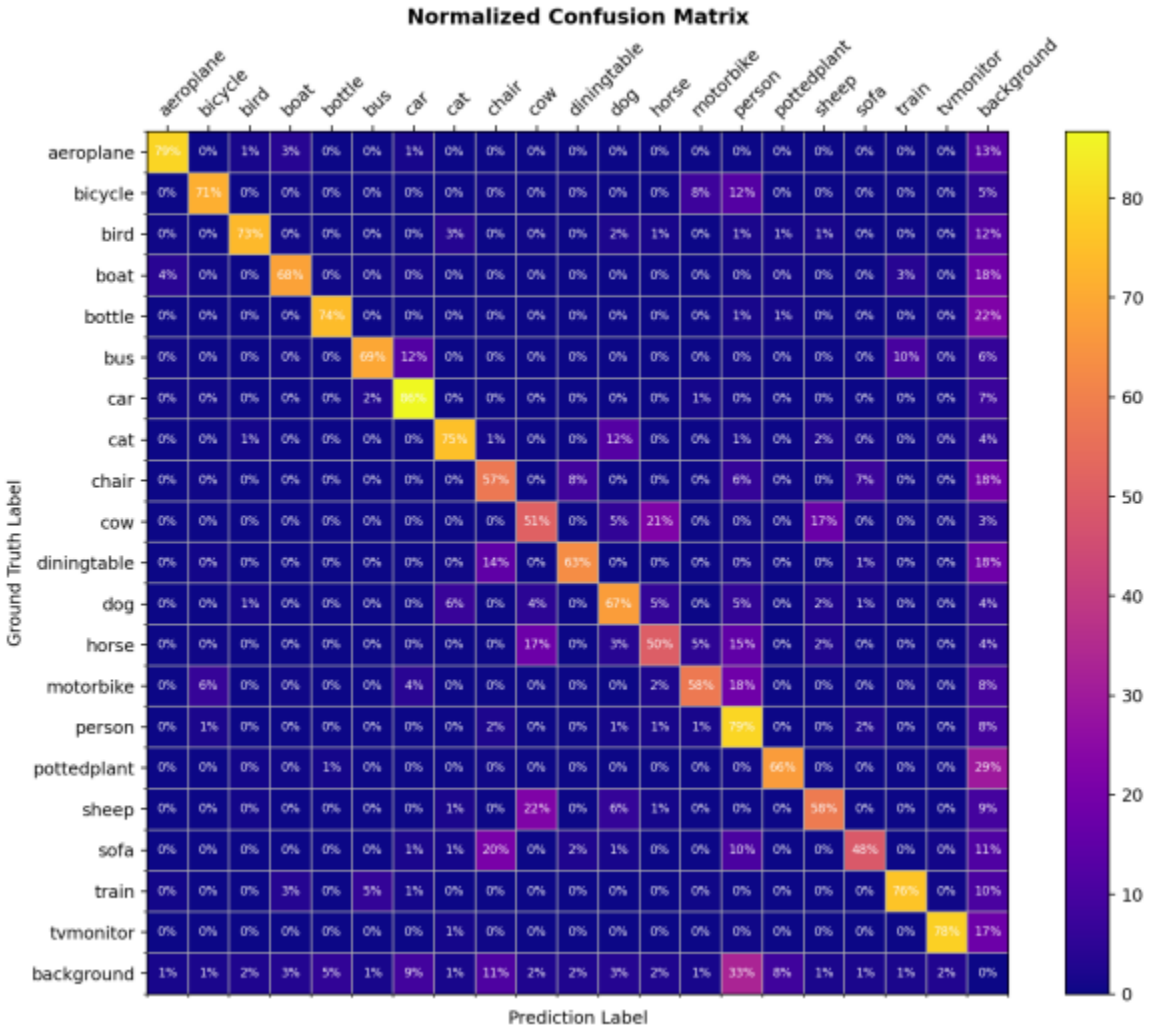
`tools/analysis_tools/confusion_matrix.py` can analyze the prediction results and plot a confution matrix table.

First, run `tools/test.py` to save the `.pkl` detection results.

Then, run

```
python tools/analysis_tools/confusion_matrix.py ${CONFIG}  ${DETECTION_RESULTS}  ${SAVE_
↪DIR} --show
```

And you will get a confution matrix like this:



confution_matrix_example

CHAPTER

THIRTY

CONVENTIONS

Please check the following conventions if you would like to modify MMDetection as your own project.

30.1 Loss

In MMDetection, a `dict` containing losses and metrics will be returned by `model(**data)`.

For example, in bbox head,

```
class BBoxHead(nn.Module):
    ...
    def loss(self, ...):
        losses = dict()
        # classification loss
        losses['loss_cls'] = self.loss_cls(...)
        # classification accuracy
        losses['acc'] = accuracy(...)
        # bbox regression loss
        losses['loss_bbox'] = self.loss_bbox(...)
        return losses
```

`bbox_head.loss()` will be called during model forward. The returned dict contains `'loss_bbox'`, `'loss_cls'`, `'acc'` . Only `'loss_bbox'`, `'loss_cls'` will be used during back propagation, `'acc'` will only be used as a metric to monitor training process.

By default, only values whose keys contain `'loss'` will be back propagated. This behavior could be changed by modifying `BaseDetector.train_step()`.

30.2 Empty Proposals

In MMDetection, We have added special handling and unit test for empty proposals of two-stage. We need to deal with the empty proposals of the entire batch and single image at the same time. For example, in CascadeRoIHead,

```
# simple_test method
...
# There is no proposal in the whole batch
if rois.shape[0] == 0:
    bbox_results = [[
        np.zeros((0, 5), dtype=np.float32)
```

(continues on next page)

(continued from previous page)

```
        for _ in range(self.bbox_head[-1].num_classes)
    ]] * num_imgs
    if self.with_mask:
        mask_classes = self.mask_head[-1].num_classes
        segm_results = [[[] for _ in range(mask_classes)]
                        for _ in range(num_imgs)]
        results = list(zip(bbox_results, segm_results))
    else:
        results = bbox_results
    return results
...

# There is no proposal in the single image
for i in range(self.num_stages):
    ...
    if i < self.num_stages - 1:
          for j in range(num_imgs):
                  # Handle empty proposal
                  if rois[j].shape[0] > 0:
                      bbox_label = cls_score[j][:, :-1].argmax(dim=1)
                      refine_roi = self.bbox_head[i].regress_by_class(
                           rois[j], bbox_label, bbox_pred[j], img_metas[j])
                      refine_roi_list.append(refine_roi)
```

If you have customized `RoIHead`, you can refer to the above method to deal with empty proposals.

30.3 Coco Panoptic Dataset

In MMDetection, we have supported COCO Panoptic dataset. We clarify a few conventions about the implementation of `CocoPanopticDataset` here.

1. For mmdet<=2.16.0, the range of foreground and background labels in semantic segmentation are different from the default setting of MMDetection. The label `0` stands for `VOID` label and the category labels start from `1`. Since mmdet=2.17.0, the category labels of semantic segmentation start from `0` and label `255` stands for `VOID` for consistency with labels of bounding boxes. To achieve that, the `Pad` pipeline supports setting the padding value for `seg`.
2. In the evaluation, the panoptic result is a map with the same shape as the original image. Each value in the result map has the format of `instance_id * INSTANCE_OFFSET + category_id`.

CHAPTER

THIRTYONE

COMPATIBILITY OF MMDETECTION 2.X

31.1 MMDetection 2.18.1

31.1.1 MMCV compatibility

In order to fix the wrong weight reference bug in BaseTransformerLayer, the logic in batch first mode of MultiheadAttention has been changed. We recommend to use MMCV v1.3.17 or higher. For more details, please refer to MMCV PR #1418.

31.2 MMDetection 2.18.0

31.2.1 DIIHead compatibility

In order to support QueryInst, attn_feats is added into the returned tuple of DIIHead.

31.3 MMDetection 2.14.0

31.3.1 MMCV Version

In order to fix the problem that the priority of EvalHook is too low, all hook priorities have been re-adjusted in 1.3.8, so MMDetection 2.14.0 needs to rely on the latest MMCV 1.3.8 version. For related information, please refer to #1120, for related issues, please refer to #5343.

31.3.2 SSD compatibility

In v2.14.0, to make SSD more flexible to use, PR5291 refactored its backbone, neck and head. The users can use the script `tools/model_converters/upgrade_ssd_version.py` to convert their models.

```
python tools/model_converters/upgrade_ssd_version.py ${OLD_MODEL_PATH} ${NEW_MODEL_PATH}
```

- OLD_MODEL_PATH: the path to load the old version SSD model.
- NEW_MODEL_PATH: the path to save the converted model weights.

31.4 MMDetection 2.12.0

MMDetection is going through big refactoring for more general and convenient usages during the releases from v2.12.0 to v2.18.0 (maybe longer). In v2.12.0 MMDetection inevitably brings some BC-breakings, including the MMCV dependency, model initialization, model registry, and mask AP evaluation.

31.4.1 MMCV Version

MMDetection v2.12.0 relies on the newest features in MMCV 1.3.3, including `BaseModule` for unified parameter initialization, model registry, and the CUDA operator `MultiScaleDeformableAttn` for Deformable DETR. Note that MMCV 1.3.2 already contains all the features used by MMDet but has known issues. Therefore, we recommend users to skip MMCV v1.3.2 and use v1.3.2, though v1.3.2 might work for most of the cases.

31.4.2 Unified model initialization

To unify the parameter initialization in OpenMMLab projects, MMCV supports `BaseModule` that accepts `init_cfg` to allow the modules' parameters initialized in a flexible and unified manner. Now the users need to explicitly call `model.init_weights()` in the training script to initialize the model (as in here, previously this was handled by the detector. **The downstream projects must update their model initialization accordingly to use MMDetection v2.12.0**. Please refer to PR #4750 for details.

31.4.3 Unified model registry

To easily use backbones implemented in other OpenMMLab projects, MMDetection v2.12.0 inherits the model registry created in MMCV (#760). In this way, as long as the backbone is supported in an OpenMMLab project and that project also uses the registry in MMCV, users can use that backbone in MMDetection by simply modifying the config without copying the code of that backbone into MMDetection. Please refer to PR #5059 for more details.

31.4.4 Mask AP evaluation

Before PR 4898 and V2.12.0, the mask AP of small, medium, and large instances is calculated based on the bounding box area rather than the real mask area. This leads to higher `APs` and `APm` but lower `APl` but will not affect the overall mask AP. PR 4898 change it to use mask areas by deleting `bbox` in mask AP calculation. The new calculation does not affect the overall mask AP evaluation and is consistent with Detectron2.

31.5 Compatibility with MMDetection 1.x

MMDetection 2.0 goes through a big refactoring and addresses many legacy issues. It is not compatible with the 1.x version, i.e., running inference with the same model weights in these two versions will produce different results. Thus, MMDetection 2.0 re-benchmarks all the models and provides their links and logs in the model zoo.

The major differences are in four folds: coordinate system, codebase conventions, training hyperparameters, and modular design.

31.5.1 Coordinate System

The new coordinate system is consistent with Detectron2 and treats the center of the most left-top pixel as (0, 0) rather than the left-top corner of that pixel. Accordingly, the system interprets the coordinates in COCO bounding box and segmentation annotations as coordinates in range `[0, width]` or `[0, height]`. This modification affects all the computation related to the bbox and pixel selection, which is more natural and accurate.

- The height and width of a box with corners (x1, y1) and (x2, y2) in the new coordinate system is computed as `width = x2 - x1` and `height = y2 - y1`. In MMDetection 1.x and previous version, a "+ 1" was added both height and width. This modification are in three folds:

  1. Box transformation and encoding/decoding in regression.
  2. IoU calculation. This affects the matching process between ground truth and bounding box and the NMS process. The effect to compatibility is very negligible, though.
  3. The corners of bounding box is in float type and no longer quantized. This should provide more accurate bounding box results. This also makes the bounding box and RoIs not required to have minimum size of 1, whose effect is small, though.

- The anchors are center-aligned to feature grid points and in float type. In MMDetection 1.x and previous version, the anchors are in `int` type and not center-aligned. This affects the anchor generation in RPN and all the anchor-based methods.

- ROIAlign is better aligned with the image coordinate system. The new implementation is adopted from Detectron2. The RoIs are shifted by half a pixel by default when they are used to cropping RoI features, compared to MMDetection 1.x. The old behavior is still available by setting `aligned=False` instead of `aligned=True`.

- Mask cropping and pasting are more accurate.

  1. We use the new RoIAlign to crop mask targets. In MMDetection 1.x, the bounding box is quantized before it is used to crop mask target, and the crop process is implemented by numpy. In new implementation, the bounding box for crop is not quantized and sent to RoIAlign. This implementation accelerates the training speed by a large margin (~0.1s per iter, ~2 hour when training Mask R50 for 1x schedule) and should be more accurate.
  2. In MMDetection 2.0, the "`paste_mask()`" function is different and should be more accurate than those in previous versions. This change follows the modification in Detectron2 and can improve mask AP on COCO by ~0.5% absolute.

31.5.2 Codebase Conventions

- MMDetection 2.0 changes the order of class labels to reduce unused parameters in regression and mask branch more naturally (without +1 and -1). This effect all the classification layers of the model to have a different ordering of class labels. The final layers of regression branch and mask head no longer keep K+1 channels for K categories, and their class orders are consistent with the classification branch.

  - In MMDetection 2.0, label "K" means background, and labels [0, K-1] correspond to the K = num_categories object categories.
  - In MMDetection 1.x and previous version, label "0" means background, and labels [1, K] correspond to the K categories.
  - **Note**: The class order of softmax RPN is still the same as that in 1.x in versions<=2.4.0 while sigmoid RPN is not affected. The class orders in all heads are unified since MMDetection v2.5.0.

- Low quality matching in R-CNN is not used. In MMDetection 1.x and previous versions, the `max_iou_assigner` will match low quality boxes for each ground truth box in both RPN and R-CNN training. We observe this sometimes does not assign the most perfect GT box to some bounding boxes, thus MMDetection

2.0 do not allow low quality matching by default in R-CNN training in the new system. This sometimes may slightly improve the box AP (~0.1% absolute).

- Separate scale factors for width and height. In MMDetection 1.x and previous versions, the scale factor is a single float in mode `keep_ratio=True`. This is slightly inaccurate because the scale factors for width and height have slight difference. MMDetection 2.0 adopts separate scale factors for width and height, the improvement on AP ~0.1% absolute.
- Configs name conventions are changed. MMDetection V2.0 adopts the new name convention to maintain the gradually growing model zoo as the following:

```
[model]_(model setting)_[backbone]_[neck]_(norm setting)_(misc)_(gpu x batch)_
↪[schedule]_[dataset].py,
```

  where the (`misc`) includes DCN and GCBlock, etc. More details are illustrated in the *documentation for config*
- MMDetection V2.0 uses new ResNet Caffe backbones to reduce warnings when loading pre-trained models. Most of the new backbones' weights are the same as the former ones but do not have `conv.bias`, except that they use a different `img_norm_cfg`. Thus, the new backbone will not cause warning of unexpected keys.

31.5.3 Training Hyperparameters

The change in training hyperparameters does not affect model-level compatibility but slightly improves the performance. The major ones are:

- The number of proposals after nms is changed from 2000 to 1000 by setting `nms_post=1000` and `max_num=1000`. This slightly improves both mask AP and bbox AP by ~0.2% absolute.
- The default box regression losses for Mask R-CNN, Faster R-CNN and RetinaNet are changed from smooth L1 Loss to L1 loss. This leads to an overall improvement in box AP (~0.6% absolute). However, using L1-loss for other methods such as Cascade R-CNN and HTC does not improve the performance, so we keep the original settings for these methods.
- The sample num of RoIAlign layer is set to be 0 for simplicity. This leads to slightly improvement on mask AP (~0.2% absolute).
- The default setting does not use gradient clipping anymore during training for faster training speed. This does not degrade performance of the most of models. For some models such as RepPoints we keep using gradient clipping to stabilize the training process and to obtain better performance.
- The default warmup ratio is changed from 1/3 to 0.001 for a more smooth warming up process since the gradient clipping is usually not used. The effect is found negligible during our re-benchmarking, though.

31.5.4 Upgrade Models from 1.x to 2.0

To convert the models trained by MMDetection V1.x to MMDetection V2.0, the users can use the script `tools/model_converters/upgrade_model_version.py` to convert their models. The converted models can be run in MMDetection V2.0 with slightly dropped performance (less than 1% AP absolute). Details can be found in `configs/legacy`.

31.6 pycocotools compatibility

`mmpycocotools` is the OpenMMlab's folk of official `pycocotools`, which works for both MMDetection and Detectron2. Before PR 4939, since `pycocotools` and `mmpycocotool` have the same package name, if users already installed `pyccocotools` (installed Detectron2 first under the same environment), then the setup of MMDetection will skip installing `mmpycocotool`. Thus MMDetection fails due to the missing `mmpycocotools`. If MMDetection is installed before Detectron2, they could work under the same environment. PR 4939 deprecates mmpycocotools in favor of official pycocotools. Users may install MMDetection and Detectron2 under the same environment after PR 4939, no matter what the installation order is.

CHAPTER

THIRTYTWO

PROJECTS BASED ON MMDETECTION

There are many projects built upon MMDetection. We list some of them as examples of how to extend MMDetection for your own projects. As the page might not be completed, please feel free to create a PR to update this page.

32.1 Projects as an extension

Some projects extend the boundary of MMDetection for deployment or other research fields. They reveal the potential of what MMDetection can do. We list several of them as below.

- OTEDetection: OpenVINO training extensions for object detection.
- MMDetection3d: OpenMMLab's next-generation platform for general 3D object detection.

32.2 Projects of papers

There are also projects released with papers. Some of the papers are published in top-tier conferences (CVPR, ICCV, and ECCV), the others are also highly influential. To make this list also a reference for the community to develop and compare new object detection algorithms, we list them following the time order of top-tier conferences. Methods already supported and maintained by MMDetection are not listed.

- Involution: Inverting the Inherence of Convolution for Visual Recognition, CVPR21. [paper][github]
- Multiple Instance Active Learning for Object Detection, CVPR 2021. [paper][github]
- Adaptive Class Suppression Loss for Long-Tail Object Detection, CVPR 2021. [paper][github]
- Generalizable Pedestrian Detection: The Elephant In The Room, CVPR2021. [paper][github]
- Group Fisher Pruning for Practical Network Compression, ICML2021. [paper][github]
- Overcoming Classifier Imbalance for Long-tail Object Detection with Balanced Group Softmax, CVPR2020. [paper][github]
- Coherent Reconstruction of Multiple Humans from a Single Image, CVPR2020. [paper][github]
- Look-into-Object: Self-supervised Structure Modeling for Object Recognition, CVPR 2020. [paper][github]
- Video Panoptic Segmentation, CVPR2020. [paper][github]
- D2Det: Towards High Quality Object Detection and Instance Segmentation, CVPR2020. [paper][github]
- CentripetalNet: Pursuing High-quality Keypoint Pairs for Object Detection, CVPR2020. [paper][github]
- Learning a Unified Sample Weighting Network for Object Detection, CVPR 2020. [paper][github]
- Scale-equalizing Pyramid Convolution for Object Detection, CVPR2020. [paper] [github]

- Revisiting the Sibling Head in Object Detector, CVPR2020. [paper][github]
- PolarMask: Single Shot Instance Segmentation with Polar Representation, CVPR2020. [paper][github]
- Hit-Detector: Hierarchical Trinity Architecture Search for Object Detection, CVPR2020. [paper][github]
- ZeroQ: A Novel Zero Shot Quantization Framework, CVPR2020. [paper][github]
- CBNet: A Novel Composite Backbone Network Architecture for Object Detection, AAAI2020. [paper][github]
- RDSNet: A New Deep Architecture for Reciprocal Object Detection and Instance Segmentation, AAAI2020. [paper][github]
- Training-Time-Friendly Network for Real-Time Object Detection, AAAI2020. [paper][github]
- Cascade RPN: Delving into High-Quality Region Proposal Network with Adaptive Convolution, NeurIPS 2019. [paper][github]
- Reasoning R-CNN: Unifying Adaptive Global Reasoning into Large-scale Object Detection, CVPR2019. [paper][github]
- Learning RoI Transformer for Oriented Object Detection in Aerial Images, CVPR2019. [paper][github]
- SOLO: Segmenting Objects by Locations. [paper][github]
- SOLOv2: Dynamic, Faster and Stronger. [paper][github]
- Dense Peppoints: Representing Visual Objects with Dense Point Sets. [paper][github]
- IterDet: Iterative Scheme for Object Detection in Crowded Environments. [paper][github]
- Cross-Iteration Batch Normalization. [paper][github]
- A Ranking-based, Balanced Loss Function Unifying Classification and Localisation in Object Detection, NeurIPS2020 [paper][github]
- RelationNet++: Bridging Visual Representations for Object Detection via Transformer Decoder, NeurIPS2020 [paper][github]
- Generalized Focal Loss V2: Learning Reliable Localization Quality Estimation for Dense Object Detection, CVPR2021[paper][github]
- Instances as Queries, ICCV2021[paper][github]
- Swin Transformer: Hierarchical Vision Transformer using Shifted Windows, ICCV2021[paper][github]
- Focal Transformer: Focal Self-attention for Local-Global Interactions in Vision Transformers, NeurIPS2021[paper][github]
- End-to-End Semi-Supervised Object Detection with Soft Teacher, ICCV2021[paper][github]
- CBNetV2: A Novel Composite Backbone Network Architecture for Object Detection [paper][github]
- Instances as Queries, ICCV2021 [paper][github]

CHAPTER

THIRTYTHREE

CHANGELOG

33.1 v2.19.0 (29/11/2021)

33.1.1 Highlights

- Support Label Assignment Distillation
- Support `persistent_workers` for Pytorch >= 1.7
- Align accuracy to the updated official YOLOX

33.1.2 New Features

- Support Label Assignment Distillation (#6342)
- Support `persistent_workers` for Pytorch >= 1.7 (#6435)

33.1.3 Bug Fixes

- Fix repeatedly output warning message (#6584)
- Avoid infinite GPU waiting in dist training (#6501)
- Fix SSD512 config error (#6574)
- Fix MMDetection model to ONNX command (#6558)

33.1.4 Improvements

- Refactor configs of FP16 models (#6592)
- Align accuracy to the updated official YOLOX (#6443)
- Speed up training and reduce memory cost when using PhotoMetricDistortion. (#6442)
- Make OHEM work with seesaw loss (#6514)

33.1.5 Documents

- Update README.md (#6567)

33.1.6 Contributors

A total of 11 developers contributed to this release. Thanks @FloydHsiu, @RangiLyu, @ZwwWayne, @AndreaPi, @st9007a, @hachreak, @BIGWangYuDong, @hhaAndroid, @AronLin, @chhluo, @vealocia, @HarborYuan, @st9007a, @jshilong

33.2 v2.18.1 (15/11/2021)

33.2.1 Highlights

- Release QueryInst pre-trained weights (#6460)
- Support plot confusion matrix (#6344)

33.2.2 New Features

- Release QueryInst pre-trained weights (#6460)
- Support plot confusion matrix (#6344)

33.2.3 Bug Fixes

- Fix aug test error when the number of prediction bboxes is 0 (#6398)
- Fix SpatialReductionAttention in PVT (#6488)
- Fix wrong use of `trunc_normal_init` in PVT and Swin-Transformer (#6432)

33.2.4 Improvements

- Save the printed AP information of COCO API to logger (#6505)
- Always map location to cpu when load checkpoint (#6405)
- Set a random seed when the user does not set a seed (#6457)

33.2.5 Documents

- Chinese version of Corruption Benchmarking (#6375)
- Fix config path in docs (#6396)
- Update GRoIE readme (#6401)

33.2.6 Contributors

A total of 11 developers contributed to this release. Thanks @st9007a, @hachreak, @HarborYuan, @vealocia, @chhluo, @AndreaPi, @AronLin, @BIGWangYuDong, @hhaAndroid, @RangiLyu, @ZwwWayne

33.3 v2.18.0 (27/10/2021)

33.3.1 Highlights

- Support QueryInst (#6050)
- Refactor dense heads to decouple onnx export logics from `get_bboxes` and speed up inference (#5317, #6003, #6369, #6268, #6315)

33.3.2 New Features

- Support QueryInst (#6050)
- Support infinite sampler (#5996)

33.3.3 Bug Fixes

- Fix init_weight in fcn_mask_head (#6378)
- Fix type error in imshow_bboxes of RPN (#6386)
- Fix broken colab link in MMDetection Tutorial (#6382)
- Make sure the device and dtype of scale_factor are the same as bboxes (#6374)
- Remove sampling hardcode (#6317)
- Fix RandomAffine bbox coordinate recorrection (#6293)
- Fix init bug of final cls/reg layer in convfc head (#6279)
- Fix img_shape broken in auto_augment (#6259)
- Fix kwargs parameter missing error in two_stage (#6256)

33.3.4 Improvements

- Unify the interface of stuff head and panoptic head (#6308)
- Polish readme (#6243)
- Add code-spell pre-commit hook and fix a typo (#6306)
- Fix typo (#6245, #6190)
- Fix sampler unit test (#6284)
- Fix `forward_dummy` of YOLACT to enable `get_flops` (#6079)
- Fix link error in the config documentation (#6252)
- Adjust the order to beautify the document (#6195)

33.3.5 Refactors

- Refactor one-stage get_bboxes logic (#5317)
- Refactor ONNX export of One-Stage models (#6003, #6369)
- Refactor dense_head and speedup (#6268)
- Migrate to use prior_generator in training of dense heads (#6315)

33.3.6 Contributors

A total of 18 developers contributed to this release. Thanks @Boyden, @onnkeat, @st9007a, @vealocia, @yhcao6, @DapangpangX, @yellowdolphin, @cclauss, @kennymckormick, @pingguokiller, @collinzrj, @AndreaPi, @AronLin, @BIGWangYuDong, @hhaAndroid, @jshilong, @RangiLyu, @ZwwWayne

33.4 v2.17.0 (28/9/2021)

33.4.1 Highlights

- Support PVT and PVTv2
- Support SOLO
- Support large scale jittering and New Mask R-CNN baselines
- Speed up `YOLOv3` inference

33.4.2 New Features

- Support PVT and PVTv2 (#5780)
- Support SOLO (#5832)
- Support large scale jittering and New Mask R-CNN baselines (#6132)
- Add a general data structrue for the results of models (#5508)
- Added a base class for one-stage instance segmentation (#5904)
- Speed up `YOLOv3` inference (#5991)
- Release Swin Transformer pre-trained models (#6100)
- Support mixed precision training in `YOLOX` (#5983)
- Support `val` workflow in `YOLACT` (#5986)
- Add script to test `torchserve` (#5936)
- Support `onnxsim` with dynamic input shape (#6117)

33.4.3 Bug Fixes

- Fix the function naming errors in `model_wrappers` (#5975)
- Fix regression loss bug when the input is an empty tensor (#5976)
- Fix scores not contiguous error in `centernet_head` (#6016)
- Fix missing parameters bug in `imshow_bboxes` (#6034)
- Fix bug in `aug_test` of `HTC` when the length of `det_bboxes` is 0 (#6088)
- Fix empty proposal errors in the training of some two-stage models (#5941)
- Fix `dynamic_axes` parameter error in `ONNX` dynamic shape export (#6104)
- Fix `dynamic_shape` bug of `SyncRandomSizeHook` (#6144)
- Fix the Swin Transformer config link error in the configuration (#6172)

33.4.4 Improvements

- Add filter rules in `Mosaic` transform (#5897)
- Add size divisor in get flops to avoid some potential bugs (#6076)
- Add Chinese translation of `docs_zh-CN/tutorials/customize_dataset.md` (#5915)
- Add Chinese translation of `conventions.md` (#5825)
- Add description of the output of data pipeline (#5886)
- Add dataset information in the README file for `PanopticFPN` (#5996)
- Add `extra_repr` for `DropBlock` layer to get details in the model printing (#6140)
- Fix CI out of memory and add PyTorch1.9 Python3.9 unit tests (#5862)
- Fix download links error of some model (#6069)
- Improve the generalization of XML dataset (#5943)
- Polish assertion error messages (#6017)
- Remove `opencv-python-headless` dependency by `albumentations` (#5868)
- Check dtype in transform unit tests (#5969)
- Replace the default theme of documentation with PyTorch Sphinx Theme (#6146)
- Update the paper and code fields in the metafile (#6043)
- Support to customize padding value of segmentation map (#6152)
- Support to resize multiple segmentation maps (#5747)

33.4.5 Contributors

A total of 24 developers contributed to this release. Thanks @morkovka1337, @HarborYuan, @guillaumefrd, @guigarfr, @www516717402, @gaotongxiao, @ypwhs, @MartaYang, @shinya7y, @justiceeem, @zhaojinjian0000, @VVsssssk, @aravind-anantha, @wangbo-zhao, @czczup, @whai362, @czczup, @marijnl, @AronLin, @BIG-WangYuDong, @hhaAndroid, @jshilong, @RangiLyu, @ZwwWayne

33.5 v2.16.0 (30/8/2021)

33.5.1 Highlights

- Support Panoptic FPN and Swin Transformer

33.5.2 New Features

- Support Panoptic FPN and release models (#5577, #5902)
- Support Swin Transformer backbone (#5748)
- Release RetinaNet models pre-trained with multi-scale 3x schedule (#5636)
- Add script to convert unlabeled image list to coco format (#5643)
- Add hook to check whether the loss value is valid (#5674)
- Add YOLO anchor optimizing tool (#5644)
- Support export onnx models without post process. (#5851)
- Support classwise evaluation in CocoPanopticDataset (#5896)
- Adapt browse_dataset for concatenated datasets. (#5935)
- Add `PatchEmbed` and `PatchMerging` with `AdaptivePadding` (#5952)

33.5.3 Bug Fixes

- Fix unit tests of YOLOX (#5859)
- Fix lose randomness in `imshow_det_bboxes` (#5845)
- Make output result of `ImageToTensor` contiguous (#5756)
- Fix inference bug when calling `regress_by_class` in RoIHead in some cases (#5884)
- Fix bug in CIoU loss where alpha should not have gradient. (#5835)
- Fix the bug that `multiscale_output` is defined but not used in HRNet (#5887)
- Set the priority of EvalHook to LOW. (#5882)
- Fix a YOLOX bug when applying bbox rescaling in test mode (#5899)
- Fix mosaic coordinate error (#5947)
- Fix dtype of bbox in RandomAffine. (#5930)

33.5.4 Improvements

- Add Chinese version of `data_pipeline` and (#5662)
- Support to remove state dicts of EMA when publishing models. (#5858)
- Refactor the loss function in HTC and SCNet (#5881)
- Use warnings instead of logger.warning (#5540)
- Use legacy coordinate in metric of VOC (#5627)
- Add Chinese version of customize_losses (#5826)
- Add Chinese version of model_zoo (#5827)

33.5.5 Contributors

A total of 19 developers contributed to this release. Thanks @ypwhs, @zywvvd, @collinzrj, @OceanPang, @ddonatien, @@haotian-liu, @viibridges, @Muyun99, @guigarfr, @zhaojinjian0000, @jbwang1997,@wangbo-zhao, @xvjiarui, @RangiLyu, @jshilong, @AronLin, @BIGWangYuDong, @hhaAndroid, @ZwwWayne

33.6 v2.15.1 (11/8/2021)

33.6.1 Highlights

- Support YOLOX

33.6.2 New Features

- Support YOLOX(#5756, #5758, #5760, #5767, #5770, #5774, #5777, #5808, #5828, #5848)

33.6.3 Bug Fixes

- Update correct SSD models. (#5789)
- Fix casting error in mask structure (#5820)
- Fix MMCV deployment documentation links. (#5790)

33.6.4 Improvements

- Use dynamic MMCV download link in TorchServe dockerfile (#5779)
- Rename the function `upsample_like` to `interpolate_as` for more general usage (#5788)

33.6.5 Contributors

A total of 14 developers contributed to this release. Thanks @HAOCHENYE, @xiaohu2015, @HsLOL, @zhiqwang, @Adamdad, @shinya7y, @Johnson-Wang, @RangiLyu, @jshilong, @mmeendez8, @AronLin, @BIGWangYuDong, @hhaAndroid, @ZwwWayne

33.7 v2.15.0 (02/8/2021)

33.7.1 Highlights

- Support adding MIM dependencies during pip installation
- Support MobileNetV2 for SSD-Lite and YOLOv3
- Support Chinese Documentation

33.7.2 New Features

- Add function `upsample_like` (#5732)
- Support to output pdf and epub format documentation (#5738)
- Support and release Cascade Mask R-CNN 3x pre-trained models (#5645)
- Add `ignore_index` to CrossEntropyLoss (#5646)
- Support adding MIM dependencies during pip installation (#5676)
- Add MobileNetV2 config and models for YOLOv3 (#5510)
- Support COCO Panoptic Dataset (#5231)
- Support ONNX export of cascade models (#5486)
- Support DropBlock with RetinaNet (#5544)
- Support MobileNetV2 SSD-Lite (#5526)

33.7.3 Bug Fixes

- Fix the device of label in multiclass_nms (#5673)
- Fix error of backbone initialization from pre-trained checkpoint in config file (#5603, #5550)
- Fix download links of RegNet pretrained weights (#5655)
- Fix two-stage runtime error given empty proposal (#5559)
- Fix flops count error in DETR (#5654)
- Fix unittest for `NumClassCheckHook` when it is not used. (#5626)
- Fix description bug of using custom dataset (#5546)
- Fix bug of `multiclass_nms` that returns the global indices (#5592)
- Fix `valid_mask` logic error in RPNHead (#5562)
- Fix unit test error of pretrained configs (#5561)
- Fix typo error in anchor_head.py (#5555)

- Fix bug when using dataset wrappers (#5552)
- Fix a typo error in demo/MMDet_Tutorial.ipynb (#5511)
- Fixing crash in `get_root_logger` when `cfg.log_level` is not None (#5521)
- Fix docker version (#5502)
- Fix optimizer parameter error when using `IterBasedRunner` (#5490)

33.7.4 Improvements

- Add unit tests for MMTracking (#5620)
- Add Chinese translation of documentation (#5718, #5618, #5558, #5423, #5593, #5421, #5408. #5369, #5419, #5530, #5531)
- Update resource limit (#5697)
- Update docstring for InstaBoost (#5640)
- Support key `reduction_override` in all loss functions (#5515)
- Use repeatdataset to accelerate CenterNet training (#5509)
- Remove unnecessary code in autoassign (#5519)
- Add documentation about `init_cfg` (#5273)

33.7.5 Contributors

A total of 18 developers contributed to this release. Thanks @OceanPang, @AronLin, @hellock, @Outsider565, @RangiLyu, @ElectronicElephant, @likyoo, @BIGWangYuDong, @hhaAndroid, @noobying, @yyz561, @likyoo, @zeakey, @ZwwWayne, @ChenyangLiu, @johnson-magic, @qingswu, @BuxianChen

33.8 v2.14.0 (29/6/2021)

33.8.1 Highlights

- Add `simple_test` to dense heads to improve the consistency of single-stage and two-stage detectors
- Revert the `test_mixins` to single image test to improve efficiency and readability
- Add Faster R-CNN and Mask R-CNN config using multi-scale training with 3x schedule

33.8.2 New Features

- Support pretrained models from MoCo v2 and SwAV (#5286)
- Add Faster R-CNN and Mask R-CNN config using multi-scale training with 3x schedule (#5179, #5233)
- Add `reduction_override` in MSELoss (#5437)
- Stable support of exporting DETR to ONNX with dynamic shapes and batch inference (#5168)
- Stable support of exporting PointRend to ONNX with dynamic shapes and batch inference (#5440)

33.8.3 Bug Fixes

- Fix size mismatch bug in `multiclass_nms` (#4980)
- Fix the import path of `MultiScaleDeformableAttention` (#5338)
- Fix errors in config of GCNet ResNext101 models (#5360)
- Fix Grid-RCNN error when there is no bbox result (#5357)
- Fix errors in `onnx_export` of bbox_head when setting reg_class_agnostic (#5468)
- Fix type error of AutoAssign in the document (#5478)
- Fix web links ending with `.md` (#5315)

33.8.4 Improvements

- Add `simple_test` to dense heads to improve the consistency of single-stage and two-stage detectors (#5264)
- Add support for mask diagonal flip in TTA (#5403)
- Revert the `test_mixins` to single image test to improve efficiency and readability (#5249)
- Make YOLOv3 Neck more flexible (#5218)
- Refactor SSD to make it more general (#5291)
- Refactor `anchor_generator` and `point_generator` (#5349)
- Allow to configure out the `mask_head` of the HTC algorithm (#5389)
- Delete deprecated warning in FPN (#5311)
- Move `model.pretrained` to `model.backbone.init_cfg` (#5370)
- Make deployment tools more friendly to use (#5280)
- Clarify installation documentation (#5316)
- Add ImageNet Pretrained Models docs (#5268)
- Add FAQ about training loss=nan solution and COCO AP or AR =-1 (# 5312, #5313)
- Change all weight links of http to https (#5328)

33.9 v2.13.0 (01/6/2021)

33.9.1 Highlights

- Support new methods: CenterNet, Seesaw Loss, MobileNetV2

33.9.2 New Features

- Support paper Objects as Points (#4602)
- Support paper Seesaw Loss for Long-Tailed Instance Segmentation (CVPR 2021) (#5128)
- Support MobileNetV2 backbone and inverted residual block (#5122)
- Support MIM (#5143)
- ONNX exportation with dynamic shapes of CornerNet (#5136)
- Add `mask_soft` config option to allow non-binary masks (#4615)
- Add PWC metafile (#5135)

33.9.3 Bug Fixes

- Fix YOLOv3 FP16 training error (#5172)
- Fix Cacscade R-CNN TTA test error when `det_bboxes` length is 0 (#5221)
- Fix `iou_thr` variable naming errors in VOC recall calculation function (#5195)
- Fix Faster R-CNN performance dropped in ONNX Runtime (#5197)
- Fix DETR dict changed error when using python 3.8 during iteration (#5226)

33.9.4 Improvements

- Refactor ONNX export of two stage detector (#5205)
- Replace MMDetection's EvalHook with MMCV's EvalHook for consistency (#4806)
- Update RoI extractor for ONNX (#5194)
- Use better parameter initialization in YOLOv3 head for higher performance (#5181)
- Release new DCN models of Mask R-CNN by mixed-precision training (#5201)
- Update YOLOv3 model weights (#5229)
- Add DetectoRS ResNet-101 model weights (#4960)
- Discard bboxes with sizes equals to `min_bbox_size` (#5011)
- Remove duplicated code in DETR head (#5129)
- Remove unnecessary object in class definition (#5180)
- Fix doc link (#5192)

33.10 v2.12.0 (01/5/2021)

33.10.1 Highlights

- Support new methods: AutoAssign, YOLOF, and Deformable DETR
- Stable support of exporting models to ONNX with batched images and dynamic shape (#5039)

33.10.2 Backwards Incompatible Changes

MMDetection is going through big refactoring for more general and convenient usages during the releases from v2.12.0 to v2.15.0 (maybe longer). In v2.12.0 MMDetection inevitably brings some BC-breakings, including the MMCV dependency, model initialization, model registry, and mask AP evaluation.

- MMCV version. MMDetection v2.12.0 relies on the newest features in MMCV 1.3.3, including `BaseModule` for unified parameter initialization, model registry, and the CUDA operator `MultiScaleDeformableAttn` for Deformable DETR. Note that MMCV 1.3.2 already contains all the features used by MMDet but has known issues. Therefore, we recommend users skip MMCV v1.3.2 and use v1.3.3, though v1.3.2 might work for most cases.
- Unified model initialization (#4750). To unify the parameter initialization in OpenMMLab projects, MMCV supports `BaseModule` that accepts `init_cfg` to allow the modules' parameters initialized in a flexible and unified manner. Now the users need to explicitly call `model.init_weights()` in the training script to initialize the model (as in here, previously this was handled by the detector. The models in MMDetection have been re-benchmarked to ensure accuracy based on PR #4750. **The downstream projects should update their code accordingly to use MMDetection v2.12.0**.
- Unified model registry (#5059). To easily use backbones implemented in other OpenMMLab projects, MMDetection migrates to inherit the model registry created in MMCV (#760). In this way, as long as the backbone is supported in an OpenMMLab project and that project also uses the registry in MMCV, users can use that backbone in MMDetection by simply modifying the config without copying the code of that backbone into MMDetection.
- Mask AP evaluation (#4898). Previous versions calculate the areas of masks through the bounding boxes when calculating the mask AP of small, medium, and large instances. To indeed use the areas of masks, we pop the key `bbox` during mask AP calculation. This change does not affect the overall mask AP evaluation and aligns the mask AP of similar models in other projects like Detectron2.

33.10.3 New Features

- Support paper AutoAssign: Differentiable Label Assignment for Dense Object Detection (#4295)
- Support paper You Only Look One-level Feature (#4295)
- Support paper Deformable DETR: Deformable Transformers for End-to-End Object Detection (#4778)
- Support calculating IoU with FP16 tensor in `bbox_overlaps` to save memory and keep speed (#4889)
- Add `__repr__` in custom dataset to count the number of instances (#4756)
- Add windows support by updating requirements.txt (#5052)
- Stable support of exporting models to ONNX with batched images and dynamic shape, including SSD, FSAF,FCOS, YOLOv3, RetinaNet, Faster R-CNN, and Mask R-CNN (#5039)

33.10.4 Improvements

- Use MMCV `MODEL_REGISTRY` (#5059)
- Unified parameter initialization for more flexible usage (#4750)
- Rename variable names and fix docstring in anchor head (#4883)
- Support training with empty GT in Cascade RPN (#4928)
- Add more details of usage of `test_robustness` in documentation (#4917)
- Changing to use `pycocotools` instead of `mmpycocotools` to fully support Detectron2 and MMDetection in one environment (#4939)
- Update torch serve dockerfile to support dockers of more versions (#4954)
- Add check for training with single class dataset (#4973)
- Refactor transformer and DETR Head (#4763)
- Update FPG model zoo (#5079)
- More accurate mask AP of small/medium/large instances (#4898)

33.10.5 Bug Fixes

- Fix bug in mean_ap.py when calculating mAP by 11 points (#4875)
- Fix error when key `meta` is not in old checkpoints (#4936)
- Fix hanging bug when training with empty GT in VFNet, GFL, and FCOS by changing the place of `reduce_mean` (#4923, #4978, #5058)
- Fix asyncronized inference error and provide related demo (#4941)
- Fix IoU losses dimensionality unmatch error (#4982)
- Fix torch.randperm whtn using PyTorch 1.8 (#5014)
- Fix empty bbox error in `mask_head` when using CARAFE (#5062)
- Fix `supplement_mask` bug when there are zero-size RoIs (#5065)
- Fix testing with empty rois in RoI Heads (#5081)

33.11 v2.11.0 (01/4/2021)

Highlights

- Support new method: Localization Distillation for Object Detection
- Support Pytorch2ONNX with batch inference and dynamic shape

New Features

- Support Localization Distillation for Object Detection (#4758)
- Support Pytorch2ONNX with batch inference and dynamic shape for Faster-RCNN and mainstream one-stage detectors (#4796)

Improvements

- Support batch inference in head of RetinaNet (#4699)

- Add batch dimension in second stage of Faster-RCNN (#4785)
- Support batch inference in bbox coder (#4721)
- Add check for `ann_ids` in `COCODataset` to ensure it is unique (#4789)
- support for showing the FPN results (#4716)
- support dynamic shape for grid_anchor (#4684)
- Move pycocotools version check to when it is used (#4880)

Bug Fixes

- Fix a bug of TridentNet when doing the batch inference (#4717)
- Fix a bug of Pytorch2ONNX in FASF (#4735)
- Fix a bug when show the image with float type (#4732)

33.12 v2.10.0 (01/03/2021)

33.12.1 Highlights

- Support new methods: FPG
- Support ONNX2TensorRT for SSD, FSAF, FCOS, YOLOv3, and Faster R-CNN.

33.12.2 New Features

- Support ONNX2TensorRT for SSD, FSAF, FCOS, YOLOv3, and Faster R-CNN (#4569)
- Support Feature Pyramid Grids (FPG) (#4645)
- Support video demo (#4420)
- Add seed option for sampler (#4665)
- Support to customize type of runner (#4570, #4669)
- Support synchronizing BN buffer in `EvalHook` (#4582)
- Add script for GIF demo (#4573)

33.12.3 Bug Fixes

- Fix ConfigDict AttributeError and add Colab link (#4643)
- Avoid crash in empty gt training of GFL head (#4631)
- Fix `iou_thrs` bug in RPN evaluation (#4581)
- Fix syntax error of config when upgrading model version (#4584)

33.12.4 Improvements

- Refactor unit test file structures (#4600)
- Refactor nms config (#4636)
- Get loading pipeline by checking the class directly rather than through config strings (#4619)
- Add doctests for mask target generation and mask structures (#4614)
- Use deep copy when copying pipeline arguments (#4621)
- Update documentations (#4642, #4650, #4620, #4630)
- Remove redundant code calling `import_modules_from_strings` (#4601)
- Clean deprecated FP16 API (#4571)
- Check whether `CLASSES` is correctly initialized in the initialization of `XMLDataset` (#4555)
- Support batch inference in the inference API (#4462, #4526)
- Clean deprecated warning and fix 'meta' error (#4695)

33.13 v2.9.0 (01/02/2021)

33.13.1 Highlights

- Support new methods: SCNet, Sparse R-CNN
- Move `train_cfg` and `test_cfg` into model in configs
- Support to visualize results based on prediction quality

33.13.2 New Features

- Support SCNet (#4356)
- Support Sparse R-CNN (#4219)
- Support evaluate mAP by multiple IoUs (#4398)
- Support concatenate dataset for testing (#4452)
- Support to visualize results based on prediction quality (#4441)
- Add ONNX simplify option to Pytorch2ONNX script (#4468)
- Add hook for checking compatibility of class numbers in heads and datasets (#4508)

33.13.3 Bug Fixes

- Fix CPU inference bug of Cascade RPN (#4410)
- Fix NMS error of CornerNet when there is no prediction box (#4409)
- Fix TypeError in CornerNet inference (#4411)
- Fix bug of PAA when training with background images (#4391)
- Fix the error that the window data is not destroyed when `out_file is not None` and `show==False` (#4442)
- Fix order of NMS `score_factor` that will decrease the performance of YOLOv3 (#4473)
- Fix bug in HTC TTA when the number of detection boxes is 0 (#4516)
- Fix resize error in mask data structures (#4520)

33.13.4 Improvements

- Allow to customize classes in LVIS dataset (#4382)
- Add tutorials for building new models with existing datasets (#4396)
- Add CPU compatibility information in documentation (#4405)
- Add documentation of deprecated `ImageToTensor` for batch inference (#4408)
- Add more details in documentation for customizing dataset (#4430)
- Switch `imshow_det_bboxes` visualization backend from OpenCV to Matplotlib (#4389)
- Deprecate `ImageToTensor` in `image_demo.py` (#4400)
- Move train_cfg/test_cfg into model (#4347, #4489)
- Update docstring for `reg_decoded_bbox` option in bbox heads (#4467)
- Update dataset information in documentation (#4525)
- Release pre-trained R50 and R101 PAA detectors with multi-scale 3x training schedules (#4495)
- Add guidance for speed benchmark (#4537)

33.14 v2.8.0 (04/01/2021)

33.14.1 Highlights

- Support new methods: Cascade RPN, TridentNet

33.14.2 New Features

- Support Cascade RPN (#1900)
- Support TridentNet (#3313)

33.14.3 Bug Fixes

- Fix bug of show result in async_benchmark (#4367)
- Fix scale factor in MaskTestMixin (#4366)
- Fix but when returning indices in `multiclass_nms` (#4362)
- Fix bug of empirical attention in resnext backbone error (#4300)
- Fix bug of `img_norm_cfg` in FCOS-HRNet models with updated performance and models (#4250)
- Fix invalid checkpoint and log in Mask R-CNN models on Cityscapes dataset (#4287)
- Fix bug in distributed sampler when dataset is too small (#4257)
- Fix bug of 'PAFPN has no attribute extra_convs_on_inputs' (#4235)

33.14.4 Improvements

- Update model url from aws to aliyun (#4349)
- Update ATSS for PyTorch 1.6+ (#4359)
- Update script to install ruby in pre-commit installation (#4360)
- Delete deprecated `mmdet.ops` (#4325)
- Refactor hungarian assigner for more general usage in Sparse R-CNN (#4259)
- Handle scipy import in DETR to reduce package dependencies (#4339)
- Update documentation of usages for config options after MMCV (1.2.3) supports overriding list in config (#4326)
- Update pre-train models of faster rcnn trained on COCO subsets (#4307)
- Avoid zero or too small value for beta in Dynamic R-CNN (#4303)
- Add doccumentation for Pytorch2ONNX (#4271)
- Add deprecated warning FPN arguments (#4264)
- Support returning indices of kept bboxes when using nms (#4251)
- Update type and device requirements when creating tensors `GFLHead` (#4210)
- Update device requirements when creating tensors in `CrossEntropyLoss` (#4224)

33.15 v2.7.0 (30/11/2020)

- Support new method: DETR, ResNest, Faster R-CNN DC5.
- Support YOLO, Mask R-CNN, and Cascade R-CNN models exportable to ONNX.

33.15.1 New Features

- Support DETR (#4201, #4206)
- Support to link the best checkpoint in training (#3773)
- Support to override config through options in inference.py (#4175)
- Support YOLO, Mask R-CNN, and Cascade R-CNN models exportable to ONNX (#4087, #4083)
- Support ResNeSt backbone (#2959)
- Support unclip border bbox regression (#4076)
- Add tpfp func in evaluating AP (#4069)
- Support mixed precision training of SSD detector with other backbones (#4081)
- Add Faster R-CNN DC5 models (#4043)

33.15.2 Bug Fixes

- Fix bug of `gpu_id` in distributed training mode (#4163)
- Support Albumentations with version higher than 0.5 (#4032)
- Fix num_classes bug in faster rcnn config (#4088)
- Update code in docs/2_new_data_model.md (#4041)

33.15.3 Improvements

- Ensure DCN offset to have similar type as features in VFNet (#4198)
- Add config links in README files of models (#4190)
- Add tutorials for loss conventions (#3818)
- Add solution to installation issues in 30-series GPUs (#4176)
- Update docker version in get_started.md (#4145)
- Add model statistics and polish some titles in configs README (#4140)
- Clamp neg probability in FreeAnchor (#4082)
- Speed up expanding large images (#4089)
- Fix Pytorch 1.7 incompatibility issues (#4103)
- Update trouble shooting page to resolve segmentation fault (#4055)
- Update aLRP-Loss in project page (#4078)
- Clean duplicated `reduce_mean` function (#4056)
- Refactor Q&A (#4045)

33.16 v2.6.0 (1/11/2020)

- Support new method: VarifocalNet.
- Refactored documentation with more tutorials.

33.16.1 New Features

- Support GIoU calculation in `BboxOverlaps2D`, and re-implement `giou_loss` using `bbox_overlaps` (#3936)
- Support random sampling in CPU mode (#3948)
- Support VarifocalNet (#3666, #4024)

33.16.2 Bug Fixes

- Fix SABL validating bug in Cascade R-CNN (#3913)
- Avoid division by zero in PAA head when num_pos=0 (#3938)
- Fix temporary directory bug of multi-node testing error (#4034, #4017)
- Fix `--show-dir` option in test script (#4025)
- Fix GA-RetinaNet r50 model url (#3983)
- Update code in docs and fix broken urls (#3947)

33.16.3 Improvements

- Refactor pytorch2onnx API into `mmdet.core.export` and use `generate_inputs_and_wrap_model` for pytorch2onnx (#3857, #3912)
- Update RPN upgrade scripts for v2.5.0 compatibility (#3986)
- Use mmcv `tensor2imgs` (#4010)
- Update test robustness (#4000)
- Update trouble shooting page (#3994)
- Accelerate PAA training speed (#3985)
- Support batch_size > 1 in validation (#3966)
- Use RoIAlign implemented in MMCV for inference in CPU mode (#3930)
- Documentation refactoring (#4031)

33.17 v2.5.0 (5/10/2020)

33.17.1 Highlights

- Support new methods: YOLACT, CentripetalNet.
- Add more documentations for easier and more clear usage.

33.17.2 Backwards Incompatible Changes

FP16 related methods are imported from mmcv instead of mmdet. (#3766, #3822) Mixed precision training utils in `mmdet.core.fp16` are moved to `mmcv.runner`, including `force_fp32`, `auto_fp16`, `wrap_fp16_model`, and `Fp16OptimizerHook`. A deprecation warning will be raised if users attempt to import those methods from `mmdet.core.fp16`, and will be finally removed in V2.10.0.

[0, N-1] represents foreground classes and N indicates background classes for all models. (#3221) Before v2.5.0, the background label for RPN is 0, and N for other heads. Now the behavior is consistent for all models. Thus `self.background_labels` in `dense_heads` is removed and all heads use `self.num_classes` to indicate the class index of background labels. This change has no effect on the pre-trained models in the v2.x model zoo, but will affect the training of all models with RPN heads. Two-stage detectors whose RPN head uses softmax will be affected because the order of categories is changed.

Only call `get_subset_by_classes` when `test_mode=True` and `self.filter_empty_gt=True` (#3695) Function `get_subset_by_classes` in dataset is refactored and only filters out images when `test_mode=True` and `self.filter_empty_gt=True`. In the original implementation, `get_subset_by_classes` is not related to the flag `self.filter_empty_gt` and will only be called when the classes is set during initialization no matter `test_mode` is `True` or `False`. This brings ambiguous behavior and potential bugs in many cases. After v2.5.0, if `filter_empty_gt=False`, no matter whether the classes are specified in a dataset, the dataset will use all the images in the annotations. If `filter_empty_gt=True` and `test_mode=True`, no matter whether the classes are specified, the dataset will call ``get_subset_by_classes` to check the images and filter out images containing no GT boxes. Therefore, the users should be responsible for the data filtering/cleaning process for the test dataset.

33.17.3 New Features

- Test time augmentation for single stage detectors (#3844, #3638)
- Support to show the name of experiments during training (#3764)
- Add `Shear`, `Rotate`, `Translate` Augmentation (#3656, #3619, #3687)
- Add image-only transformations including `Constrast`, `Equalize`, `Color`, and `Brightness`. (#3643)
- Support YOLACT (#3456)
- Support CentripetalNet (#3390)
- Support PyTorch 1.6 in docker (#3905)

33.17.4 Bug Fixes

- Fix the bug of training ATSS when there is no ground truth boxes (#3702)
- Fix the bug of using Focal Loss when there is `num_pos` is 0 (#3702)
- Fix the label index mapping in dataset browser (#3708)
- Fix Mask R-CNN training stuck problem when their is no positive rois (#3713)
- Fix the bug of `self.rpn_head.test_cfg` in `RPNTestMixin` by using `self.rpn_head` in rpn head (#3808)
- Fix deprecated `Conv2d` from mmcv.ops (#3791)
- Fix device bug in RepPoints (#3836)
- Fix SABL validating bug (#3849)
- Use `https://download.openmmlab.com/mmcv/dist/index.html` for installing MMCV (#3840)
- Fix nonzero in NMS for PyTorch 1.6.0 (#3867)
- Fix the API change bug of PAA (#3883)
- Fix typo in bbox_flip (#3886)
- Fix cv2 import error of ligGL.so.1 in Dockerfile (#3891)

33.17.5 Improvements

- Change to use `mmcv.utils.collect_env` for collecting environment information to avoid duplicate codes (#3779)
- Update checkpoint file names to v2.0 models in documentation (#3795)
- Update tutorials for changing runtime settings (#3778), modifying loss (#3777)
- Improve the function of `simple_test_bboxes` in SABL (#3853)
- Convert mask to bool before using it as img's index for robustness and speedup (#3870)
- Improve documentation of modules and dataset customization (#3821)

33.18 v2.4.0 (5/9/2020)

Highlights

- Fix lots of issues/bugs and reorganize the trouble shooting page
- Support new methods SABL, YOLOv3, and PAA Assign
- Support Batch Inference
- Start to publish `mmdet` package to PyPI since v2.3.0
- Switch model zoo to download.openmmlab.com

Backwards Incompatible Changes

- Support Batch Inference (#3564, #3686, #3705): Since v2.4.0, MMDetection could inference model with multiple images in a single GPU. This change influences all the test APIs in MMDetection and downstream codebases. To help the users migrate their code, we use `replace_ImageToTensor` (#3686) to convert legacy test data pipelines during dataset initialization.

- Support RandomFlip with horizontal/vertical/diagonal direction (#3608): Since v2.4.0, MMDetection supports horizontal/vertical/diagonal flip in the data augmentation. This influences bounding box, mask, and image transformations in data augmentation process and the process that will map those data back to the original format.
- Migrate to use `mmlvis` and `mmpycocotools` for COCO and LVIS dataset (#3727). The APIs are fully compatible with the original `lvis` and `pycocotools`. Users need to uninstall the existing pycocotools and lvis packages in their environment first and install `mmlvis` & `mmpycocotools`.

Bug Fixes

- Fix default mean/std for onnx (#3491)
- Fix coco evaluation and add metric items (#3497)
- Fix typo for install.md (#3516)
- Fix atss when sampler per gpu is 1 (#3528)
- Fix import of fuse_conv_bn (#3529)
- Fix bug of gaussian_target, update unittest of heatmap (#3543)
- Fixed VOC2012 evaluate (#3553)
- Fix scale factor bug of rescale (#3566)
- Fix with_xxx_attributes in base detector (#3567)
- Fix boxes scaling when number is 0 (#3575)
- Fix rfp check when neck config is a list (#3591)
- Fix import of fuse conv bn in benchmark.py (#3606)
- Fix webcam demo (#3634)
- Fix typo and itemize issues in tutorial (#3658)
- Fix error in distributed training when some levels of FPN are not assigned with bounding boxes (#3670)
- Fix the width and height orders of stride in valid flag generation (#3685)
- Fix weight initialization bug in Res2Net DCN (#3714)
- Fix bug in OHEMSampler (#3677)

New Features

- Support Cutout augmentation (#3521)
- Support evaluation on multiple datasets through ConcatDataset (#3522)
- Support PAA assign #(3547)
- Support eval metric with pickle results (#3607)
- Support YOLOv3 (#3083)
- Support SABL (#3603)
- Support to publish to Pypi in github-action (#3510)
- Support custom imports (#3641)

Improvements

- Refactor common issues in documentation (#3530)
- Add pytorch 1.6 to CI config (#3532)

- Add config to runner meta (#3534)
- Add eval-option flag for testing (#3537)
- Add init_eval to evaluation hook (#3550)
- Add include_bkg in ClassBalancedDataset (#3577)
- Using config's loading in inference_detector (#3611)
- Add ATSS ResNet-101 models in model zoo (#3639)
- Update urls to download.openmmlab.com (#3665)
- Support non-mask training for CocoDataset (#3711)

33.19 v2.3.0 (5/8/2020)

Highlights

- The CUDA/C++ operators have been moved to `mmcv.ops`. For backward compatibility `mmdet.ops` is kept as warppers of `mmcv.ops`.
- Support new methods CornerNet, DIOU/CIOU loss, and new dataset: LVIS V1
- Provide more detailed colab training tutorials and more complete documentation.
- Support to convert RetinaNet from Pytorch to ONNX.

Bug Fixes

- Fix the model initialization bug of DetectoRS (#3187)
- Fix the bug of module names in NASFCOSHead (#3205)
- Fix the filename bug in publish_model.py (#3237)
- Fix the dimensionality bug when `inside_flags.any()` is `False` in dense heads (#3242)
- Fix the bug of forgetting to pass flip directions in `MultiScaleFlipAug` (#3262)
- Fixed the bug caused by default value of `stem_channels` (#3333)
- Fix the bug of model checkpoint loading for CPU inference (#3318, #3316)
- Fix topk bug when box number is smaller than the expected topk number in ATSSAssigner (#3361)
- Fix the gt priority bug in center_region_assigner.py (#3208)
- Fix NaN issue of iou calculation in iou_loss.py (#3394)
- Fix the bug that `iou_thrs` is not actually used during evaluation in coco.py (#3407)
- Fix test-time augmentation of RepPoints (#3435)
- Fix runtimeError caused by incontiguous tensor in Res2Net+DCN (#3412)

New Features

- Support CornerNet (#3036)
- Support DIOU/CIOU loss (#3151)
- Support LVIS V1 dataset (#)
- Support customized hooks in training (#3395)
- Support fp16 training of generalized focal loss (#3410)

- Support to convert RetinaNet from Pytorch to ONNX (#3075)

Improvements

- Support to process ignore boxes in ATSS assigner (#3082)
- Allow to crop images without ground truth in `RandomCrop` (#3153)
- Enable the the `Accuracy` module to set threshold (#3155)
- Refactoring unit tests (#3206)
- Unify the training settings of `to_float32` and `norm_cfg` in RegNets configs (#3210)
- Add colab training tutorials for beginners (#3213, #3273)
- Move CUDA/C++ operators into `mmcv.ops` and keep `mmdet.ops` as warppers for backward compatibility (#3232)(#3457)
- Update installation scripts in documentation (#3290) and dockerfile (#3320)
- Support to set image resize backend (#3392)
- Remove git hash in version file (#3466)
- Check mmcv version to force version compatibility (#3460)

33.20 v2.2.0 (1/7/2020)

Highlights

- Support new methods: DetectoRS, PointRend, Generalized Focal Loss, Dynamic R-CNN

Bug Fixes

- Fix FreeAnchor when no gt in image (#3176)
- Clean up deprecated usage of `register_module()` (#3092, #3161)
- Fix pretrain bug in NAS FCOS (#3145)
- Fix `num_classes` in SSD (#3142)
- Fix FCOS warmup (#3119)
- Fix `rstrip` in `tools/publish_model.py`
- Fix `flip_ratio` default value in RandomFLip pipeline (#3106)
- Fix cityscapes eval with ms_rcnn (#3112)
- Fix RPN softmax (#3056)
- Fix filename of LVIS@v0.5 (#2998)
- Fix nan loss by filtering out-of-frame gt_bboxes in COCO (#2999)
- Fix bug in FSAF (#3018)
- Add FocalLoss `num_classes` check (#2964)
- Fix PISA Loss when there are no gts (#2992)
- Avoid nan in `iou_calculator` (#2975)
- Prevent possible bugs in loading and transforms caused by shallow copy (#2967)

New Features

- Add DetectoRS (#3064)
- Support Generalize Focal Loss (#3097)
- Support PointRend (#2752)
- Support Dynamic R-CNN (#3040)
- Add DeepFashion dataset (#2968)
- Implement FCOS training tricks (#2935)
- Use BaseDenseHead as base class for anchor-base heads (#2963)
- Add `with_cp` for BasicBlock (#2891)
- Add `stem_channels` argument for ResNet (#2954)

Improvements

- Add anchor free base head (#2867)
- Migrate to github action (#3137)
- Add docstring for datasets, pipelines, core modules and methods (#3130, #3125, #3120)
- Add VOC benchmark (#3060)
- Add `concat` mode in GRoI (#3098)
- Remove cmd arg `autorescale-lr` (#3080)
- Use `len(data['img_metas'])` to indicate `num_samples` (#3073, #3053)
- Switch to EpochBasedRunner (#2976)

33.21 v2.1.0 (8/6/2020)

Highlights

- Support new backbones: RegNetX, Res2Net
- Support new methods: NASFCOS, PISA, GRoIE
- Support new dataset: LVIS

Bug Fixes

- Change the CLI argument `--validate` to `--no-validate` to enable validation after training epochs by default. (#2651)
- Add missing cython to docker file (#2713)
- Fix bug in nms cpu implementation (#2754)
- Fix bug when showing mask results (#2763)
- Fix gcc requirement (#2806)
- Fix bug in async test (#2820)
- Fix mask encoding-decoding bugs in test API (#2824)
- Fix bug in test time augmentation (#2858, #2921, #2944)
- Fix a typo in comment of apis/train (#2877)

- Fix the bug of returning None when no gt bboxes are in the original image in `RandomCrop`. Fix the bug that misses to handle `gt_bboxes_ignore`, `gt_label_ignore`, and `gt_masks_ignore` in `RandomCrop`, `MinIoURandomCrop` and `Expand` modules. (#2810)
- Fix bug of `base_channels` of regnet (#2917)
- Fix the bug of logger when loading pre-trained weights in base detector (#2936)

New Features

- Add IoU models (#2666)
- Add colab demo for inference
- Support class agnostic nms (#2553)
- Add benchmark gathering scripts for development only (#2676)
- Add mmdet-based project links (#2736, #2767, #2895)
- Add config dump in training (#2779)
- Add ClassBalancedDataset (#2721)
- Add res2net backbone (#2237)
- Support RegNetX models (#2710)
- Use `mmcv.FileClient` to support different storage backends (#2712)
- Add ClassBalancedDataset (#2721)
- Code Release: Prime Sample Attention in Object Detection (CVPR 2020) (#2626)
- Implement NASFCOS (#2682)
- Add class weight in CrossEntropyLoss (#2797)
- Support LVIS dataset (#2088)
- Support GRoIE (#2584)

Improvements

- Allow different x and y strides in anchor heads. (#2629)
- Make FSAF loss more robust to no gt (#2680)
- Compute pure inference time instead (#2657) and update inference speed (#2730)
- Avoided the possibility that a patch with 0 area is cropped. (#2704)
- Add warnings when deprecated `imgs_per_gpu` is used. (#2700)
- Add a mask rcnn example for config (#2645)
- Update model zoo (#2762, #2866, #2876, #2879, #2831)
- Add `ori_filename` to img_metas and use it in test show-dir (#2612)
- Use `img_fields` to handle multiple images during image transform (#2800)
- Add upsample_cfg support in FPN (#2787)
- Add `['img']` as default `img_fields` for back compatibility (#2809)
- Rename the pretrained model from `open-mmlab://resnet50_caffe` and `open-mmlab://resnet50_caffe_bgr` to `open-mmlab://detectron/resnet50_caffe` and `open-mmlab://detectron2/resnet50_caffe`. (#2832)

- Added sleep(2) in test.py to reduce hanging problem (#2847)
- Support `c10::half` in CARAFE (#2890)
- Improve documentations (#2918, #2714)
- Use optimizer constructor in mmcv and clean the original implementation in `mmdet.core.optimizer` (#2947)

33.22 v2.0.0 (6/5/2020)

In this release, we made lots of major refactoring and modifications.

1. **Faster speed**. We optimize the training and inference speed for common models, achieving up to 30% speedup for training and 25% for inference. Please refer to model zoo for details.
2. **Higher performance**. We change some default hyperparameters with no additional cost, which leads to a gain of performance for most models. Please refer to compatibility for details.
3. **More documentation and tutorials**. We add a bunch of documentation and tutorials to help users get started more smoothly. Read it here.
4. **Support PyTorch 1.5**. The support for 1.1 and 1.2 is dropped, and we switch to some new APIs.
5. **Better configuration system**. Inheritance is supported to reduce the redundancy of configs.
6. **Better modular design**. Towards the goal of simplicity and flexibility, we simplify some encapsulation while add more other configurable modules like BBoxCoder, IoUCalculator, OptimizerConstructor, RoIHead. Target computation is also included in heads and the call hierarchy is simpler.
7. Support new methods: FSAF and PAFPN (part of PAFPN).

Breaking Changes Models training with MMDetection 1.x are not fully compatible with 2.0, please refer to the *compatibility doc* for the details and how to migrate to the new version.

Improvements

- Unify cuda and cpp API for custom ops. (#2277)
- New config files with inheritance. (#2216)
- Encapsulate the second stage into RoI heads. (#1999)
- Refactor GCNet/EmpericalAttention into plugins. (#2345)
- Set low quality match as an option in IoU-based bbox assigners. (#2375)
- Change the codebase's coordinate system. (#2380)
- Refactor the category order in heads. 0 means the first positive class instead of background now. (#2374)
- Add bbox sampler and assigner registry. (#2419)
- Speed up the inference of RPN. (#2420)
- Add `train_cfg` and `test_cfg` as class members in all anchor heads. (#2422)
- Merge target computation methods into heads. (#2429)
- Add bbox coder to support different bbox encoding and losses. (#2480)
- Unify the API for regression loss. (#2156)
- Refactor Anchor Generator. (#2474)
- Make `lr` an optional argument for optimizers. (#2509)

- Migrate to modules and methods in MMCV. (#2502, #2511, #2569, #2572)
- Support PyTorch 1.5. (#2524)
- Drop the support for Python 3.5 and use F-string in the codebase. (#2531)

Bug Fixes

- Fix the scale factors for resized images without keep the aspect ratio. (#2039)
- Check if max_num > 0 before slicing in NMS. (#2486)
- Fix Deformable RoIPool when there is no instance. (#2490)
- Fix the default value of assigned labels. (#2536)
- Fix the evaluation of Cityscapes. (#2578)

New Features

- Add deep_stem and avg_down option to ResNet, i.e., support ResNetV1d. (#2252)
- Add L1 loss. (#2376)
- Support both polygon and bitmap for instance masks. (#2353, #2540)
- Support CPU mode for inference. (#2385)
- Add optimizer constructor for complicated configuration of optimizers. (#2397, #2488)
- Implement PAFPN. (#2392)
- Support empty tensor input for some modules. (#2280)
- Support for custom dataset classes without overriding it. (#2408, #2443)
- Support to train subsets of coco dataset. (#2340)
- Add iou_calculator to potentially support more IoU calculation methods. (2405)
- Support class wise mean AP (was removed in the last version). (#2459)
- Add option to save the testing result images. (#2414)
- Support MomentumUpdaterHook. (#2571)
- Add a demo to inference a single image. (#2605)

33.23 v1.1.0 (24/2/2020)

Highlights

- Dataset evaluation is rewritten with a unified api, which is used by both evaluation hooks and test scripts.
- Support new methods: CARAFE.

Breaking Changes

- The new MMDDP inherits from the official DDP, thus the `__init__` api is changed to be the same as official DDP.
- The `mask_head` field in HTC config files is modified.
- The evaluation and testing script is updated.
- In all transforms, instance masks are stored as a numpy array shaped (n, h, w) instead of a list of (h, w) arrays, where n is the number of instances.

Bug Fixes

- Fix IOU assigners when ignore_iof_thr > 0 and there is no pred boxes. (#2135)
- Fix mAP evaluation when there are no ignored boxes. (#2116)
- Fix the empty RoI input for Deformable RoI Pooling. (#2099)
- Fix the dataset settings for multiple workflows. (#2103)
- Fix the warning related to `torch.uint8` in PyTorch 1.4. (#2105)
- Fix the inference demo on devices other than gpu:0. (#2098)
- Fix Dockerfile. (#2097)
- Fix the bug that `pad_val` is unused in Pad transform. (#2093)
- Fix the albumentation transform when there is no ground truth bbox. (#2032)

Improvements

- Use torch instead of numpy for random sampling. (#2094)
- Migrate to the new MMDDP implementation in MMCV v0.3. (#2090)
- Add meta information in logs. (#2086)
- Rewrite Soft NMS with pytorch extension and remove cython as a dependency. (#2056)
- Rewrite dataset evaluation. (#2042, #2087, #2114, #2128)
- Use numpy array for masks in transforms. (#2030)

New Features

- Implement "CARAFE: Content-Aware ReAssembly of FEatures". (#1583)
- Add `worker_init_fn()` in data_loader when seed is set. (#2066, #2111)
- Add logging utils. (#2035)

33.24 v1.0.0 (30/1/2020)

This release mainly improves the code quality and add more docstrings.

Highlights

- Documentation is online now: https://mmdetection.readthedocs.io.
- Support new models: ATSS.
- DCN is now available with the api `build_conv_layer` and `ConvModule` like the normal conv layer.
- A tool to collect environment information is available for trouble shooting.

Bug Fixes

- Fix the incompatibility of the latest numpy and pycocotools. (#2024)
- Fix the case when distributed package is unavailable, e.g., on Windows. (#1985)
- Fix the dimension issue for `refine_bboxes()`. (#1962)
- Fix the typo when `seg_prefix` is a list. (#1906)
- Add segmentation map cropping to RandomCrop. (#1880)

- Fix the return value of `ga_shape_target_single()`. (#1853)
- Fix the loaded shape of empty proposals. (#1819)
- Fix the mask data type when using albumentation. (#1818)

Improvements

- Enhance AssignResult and SamplingResult. (#1995)
- Add ability to overwrite existing module in Registry. (#1982)
- Reorganize requirements and make albumentations and imagecorruptions optional. (#1969)
- Check NaN in `SSDHead`. (#1935)
- Encapsulate the DCN in ResNe(X)t into a ConvModule & Conv_layers. (#1894)
- Refactoring for mAP evaluation and support multiprocessing and logging. (#1889)
- Init the root logger before constructing Runner to log more information. (#1865)
- Split `SegResizeFlipPadRescale` into different existing transforms. (#1852)
- Move `init_dist()` to MMCV. (#1851)
- Documentation and docstring improvements. (#1971, #1938, #1869, #1838)
- Fix the color of the same class for mask visualization. (#1834)
- Remove the option `keep_all_stages` in HTC and Cascade R-CNN. (#1806)

New Features

- Add two test-time options `crop_mask` and `rle_mask_encode` for mask heads. (#2013)
- Support loading grayscale images as single channel. (#1975)
- Implement "Bridging the Gap Between Anchor-based and Anchor-free Detection via Adaptive Training Sample Selection". (#1872)
- Add sphinx generated docs. (#1859, #1864)
- Add GN support for flops computation. (#1850)
- Collect env info for trouble shooting. (#1812)

33.25 v1.0rc1 (13/12/2019)

The RC1 release mainly focuses on improving the user experience, and fixing bugs.

Highlights

- Support new models: FoveaBox, RepPoints and FreeAnchor.
- Add a Dockerfile.
- Add a jupyter notebook demo and a webcam demo.
- Setup the code style and CI.
- Add lots of docstrings and unit tests.
- Fix lots of bugs.

Breaking Changes

- There was a bug for computing COCO-style mAP w.r.t different scales (AP_s, AP_m, AP_l), introduced by #621. (#1679)

Bug Fixes

- Fix a sampling interval bug in Libra R-CNN. (#1800)
- Fix the learning rate in SSD300 WIDER FACE. (#1781)
- Fix the scaling issue when `keep_ratio=False`. (#1730)
- Fix typos. (#1721, #1492, #1242, #1108, #1107)
- Fix the shuffle argument in `build_dataloader`. (#1693)
- Clip the proposal when computing mask targets. (#1688)
- Fix the "index out of range" bug for samplers in some corner cases. (#1610, #1404)
- Fix the NMS issue on devices other than GPU:0. (#1603)
- Fix SSD Head and GHM Loss on CPU. (#1578)
- Fix the OOM error when there are too many gt bboxes. (#1575)
- Fix the wrong keyword argument `nms_cfg` in HTC. (#1573)
- Process masks and semantic segmentation in Expand and MinIoUCrop transforms. (#1550, #1361)
- Fix a scale bug in the Non Local op. (#1528)
- Fix a bug in transforms when `gt_bboxes_ignore` is None. (#1498)
- Fix a bug when `img_prefix` is None. (#1497)
- Pass the device argument to `grid_anchors` and `valid_flags`. (#1478)
- Fix the data pipeline for test_robustness. (#1476)
- Fix the argument type of deformable pooling. (#1390)
- Fix the coco_eval when there are only two classes. (#1376)
- Fix a bug in Modulated DeformableConv when deformable_group>1. (#1359)
- Fix the mask cropping in RandomCrop. (#1333)
- Fix zero outputs in DeformConv when not running on cuda:0. (#1326)
- Fix the type issue in Expand. (#1288)
- Fix the inference API. (#1255)
- Fix the inplace operation in Expand. (#1249)
- Fix the from-scratch training config. (#1196)
- Fix inplace add in RoIExtractor which cause an error in PyTorch 1.2. (#1160)
- Fix FCOS when input images has no positive sample. (#1136)
- Fix recursive imports. (#1099)

Improvements

- Print the config file and mmdet version in the log. (#1721)
- Lint the code before compiling in travis CI. (#1715)
- Add a probability argument for the `Expand` transform. (#1651)

- Update the PyTorch and CUDA version in the docker file. (#1615)
- Raise a warning when specifying `--validate` in non-distributed training. (#1624, #1651)
- Beautify the mAP printing. (#1614)
- Add pre-commit hook. (#1536)
- Add the argument `in_channels` to backbones. (#1475)
- Add lots of docstrings and unit tests, thanks to @Erotemic. (#1603, #1517, #1506, #1505, #1491, #1479, #1477, #1475, #1474)
- Add support for multi-node distributed test when there is no shared storage. (#1399)
- Optimize Dockerfile to reduce the image size. (#1306)
- Update new results of HRNet. (#1284, #1182)
- Add an argument `no_norm_on_lateral` in FPN. (#1240)
- Test the compiling in CI. (#1235)
- Move docs to a separate folder. (#1233)
- Add a jupyter notebook demo. (#1158)
- Support different type of dataset for training. (#1133)
- Use int64_t instead of long in cuda kernels. (#1131)
- Support unsquare RoIs for bbox and mask heads. (#1128)
- Manually add type promotion to make compatible to PyTorch 1.2. (#1114)
- Allowing validation dataset for computing validation loss. (#1093)
- Use `.scalar_type()` instead of `.type()` to suppress some warnings. (#1070)

New Features

- Add an option `--with_ap` to compute the AP for each class. (#1549)
- Implement "FreeAnchor: Learning to Match Anchors for Visual Object Detection". (#1391)
- Support Albumentations for augmentations in the data pipeline. (#1354)
- Implement "FoveaBox: Beyond Anchor-based Object Detector". (#1339)
- Support horizontal and vertical flipping. (#1273, #1115)
- Implement "RepPoints: Point Set Representation for Object Detection". (#1265)
- Add test-time augmentation to HTC and Cascade R-CNN. (#1251)
- Add a COCO result analysis tool. (#1228)
- Add Dockerfile. (#1168)
- Add a webcam demo. (#1155, #1150)
- Add FLOPs counter. (#1127)
- Allow arbitrary layer order for ConvModule. (#1078)

33.26 v1.0rc0 (27/07/2019)

- Implement lots of new methods and components (Mixed Precision Training, HTC, Libra R-CNN, Guided Anchoring, Empirical Attention, Mask Scoring R-CNN, Grid R-CNN (Plus), GHM, GCNet, FCOS, HRNet, Weight Standardization, etc.). Thank all collaborators!
- Support two additional datasets: WIDER FACE and Cityscapes.
- Refactoring for loss APIs and make it more flexible to adopt different losses and related hyper-parameters.
- Speed up multi-gpu testing.
- Integrate all compiling and installing in a single script.

33.27 v0.6.0 (14/04/2019)

- Up to 30% speedup compared to the model zoo.
- Support both PyTorch stable and nightly version.
- Replace NMS and SigmoidFocalLoss with Pytorch CUDA extensions.

33.28 v0.6rc0(06/02/2019)

- Migrate to PyTorch 1.0.

33.29 v0.5.7 (06/02/2019)

- Add support for Deformable ConvNet v2. (Many thanks to the authors and @chengdazhi)
- This is the last release based on PyTorch 0.4.1.

33.30 v0.5.6 (17/01/2019)

- Add support for Group Normalization.
- Unify RPNHead and single stage heads (RetinaHead, SSDHead) with AnchorHead.

33.31 v0.5.5 (22/12/2018)

- Add SSD for COCO and PASCAL VOC.
- Add ResNeXt backbones and detection models.
- Refactoring for Samplers/Assigners and add OHEM.
- Add VOC dataset and evaluation scripts.

33.32 v0.5.4 (27/11/2018)

- Add SingleStageDetector and RetinaNet.

33.33 v0.5.3 (26/11/2018)

- Add Cascade R-CNN and Cascade Mask R-CNN.
- Add support for Soft-NMS in config files.

33.34 v0.5.2 (21/10/2018)

- Add support for custom datasets.
- Add a script to convert PASCAL VOC annotations to the expected format.

33.35 v0.5.1 (20/10/2018)

- Add BBoxAssigner and BBoxSampler, the `train_cfg` field in config files are restructured.
- `ConvFCRoIHead` / `SharedFCRoIHead` are renamed to `ConvFCBBoxHead` / `SharedFCBBoxHead` for consistency.

CHAPTER

THIRTYFOUR

FREQUENTLY ASKED QUESTIONS

We list some common troubles faced by many users and their corresponding solutions here. Feel free to enrich the list if you find any frequent issues and have ways to help others to solve them. If the contents here do not cover your issue, please create an issue using the provided templates and make sure you fill in all required information in the template.

34.1 MMCV Installation

- Compatibility issue between MMCV and MMDetection; "ConvWS is already registered in conv layer"; "AssertionError: MMCV==xxx is used but incompatible. Please install mmcv>=xxx, <=xxx."

  Please install the correct version of MMCV for the version of your MMDetection following the installation instruction.

- "No module named 'mmcv.ops'"; "No module named 'mmcv._ext'".

  1. Uninstall existing mmcv in the environment using `pip uninstall mmcv`.
  2. Install mmcv-full following the installation instruction.

34.2 PyTorch/CUDA Environment

- "RTX 30 series card fails when building MMCV or MMDet"

  1. Temporary work-around: do `MMCV_WITH_OPS=1 MMCV_CUDA_ARGS='-gencode=arch=compute_80,code=sm_80' pip install -e .`. The common issue is `nvcc fatal : Unsupported gpu architecture 'compute_86'`. This means that the compiler should optimize for sm_86, i.e., nvidia 30 series card, but such optimizations have not been supported by CUDA toolkit 11.0. This work-around modifies the compile flag by adding `MMCV_CUDA_ARGS='-gencode=arch=compute_80,code=sm_80'`, which tells `nvcc` to optimize for **sm_80**, i.e., Nvidia A100. Although A100 is different from the 30 series card, they use similar ampere architecture. This may hurt the performance but it works.
  2. PyTorch developers have updated that the default compiler flags should be fixed by pytorch/pytorch#47585. So using PyTorch-nightly may also be able to solve the problem, though we have not tested it yet.

- "invalid device function" or "no kernel image is available for execution".

  1. Check if your cuda runtime version (under `/usr/local/`), `nvcc --version` and `conda list cudatoolkit` version match.
  2. Run `python mmdet/utils/collect_env.py` to check whether PyTorch, torchvision, and MMCV are built for the correct GPU architecture. You may need to set `TORCH_CUDA_ARCH_LIST` to reinstall MMCV. The GPU arch table could be found here, i.e. run `TORCH_CUDA_ARCH_LIST=7.0 pip install`

`mmcv-full` to build MMCV for Volta GPUs. The compatibility issue could happen when using old GPUS, e.g., Tesla K80 (3.7) on colab.

3. Check whether the running environment is the same as that when mmcv/mmdet has compiled. For example, you may compile mmcv using CUDA 10.0 but run it on CUDA 9.0 environments.

- "undefined symbol" or "cannot open xxx.so".

1. If those symbols are CUDA/C++ symbols (e.g., libcudart.so or GLIBCXX), check whether the CUDA/GCC runtimes are the same as those used for compiling mmcv, i.e. run `python mmdet/utils/collect_env.py` to see if `"MMCV Compiler"/"MMCV CUDA Compiler"` is the same as `"GCC"/"CUDA_HOME"`.
2. If those symbols are PyTorch symbols (e.g., symbols containing caffe, aten, and TH), check whether the PyTorch version is the same as that used for compiling mmcv.
3. Run `python mmdet/utils/collect_env.py` to check whether PyTorch, torchvision, and MMCV are built by and running on the same environment.

- setuptools.sandbox.UnpickleableException: DistutilsSetupError("each element of 'ext_modules' option must be an Extension instance or 2-tuple")

1. If you are using miniconda rather than anaconda, check whether Cython is installed as indicated in #3379. You need to manually install Cython first and then run command `pip install -r requirements.txt`.
2. You may also need to check the compatibility between the `setuptools`, `Cython`, and `PyTorch` in your environment.

- "Segmentation fault".

1. Check you GCC version and use GCC 5.4. This usually caused by the incompatibility between PyTorch and the environment (e.g., GCC < 4.9 for PyTorch). We also recommend the users to avoid using GCC 5.5 because many feedbacks report that GCC 5.5 will cause "segmentation fault" and simply changing it to GCC 5.4 could solve the problem.
2. Check whether PyTorch is correctly installed and could use CUDA op, e.g. type the following command in your terminal.

```
python -c 'import torch; print(torch.cuda.is_available())'
```

And see whether they could correctly output results.

3. If Pytorch is correctly installed, check whether MMCV is correctly installed.

```
python -c 'import mmcv; import mmcv.ops'
```

If MMCV is correctly installed, then there will be no issue of the above two commands.

4. If MMCV and Pytorch is correctly installed, you man use `ipdb`, `pdb` to set breakpoints or directly add 'print' in mmdetection code and see which part leads the segmentation fault.

34.3 Training

- "Loss goes Nan"

1. Check if the dataset annotations are valid: zero-size bounding boxes will cause the regression loss to be Nan due to the commonly used transformation for box regression. Some small size (width or height are smaller than 1) boxes will also cause this problem after data augmentation (e.g., instaboost). So check the data and try to filter out those zero-size boxes and skip some risky augmentations on the small-size boxes when you face the problem.

2. Reduce the learning rate: the learning rate might be too large due to some reasons, e.g., change of batch size. You can rescale them to the value that could stably train the model.
3. Extend the warmup iterations: some models are sensitive to the learning rate at the start of the training. You can extend the warmup iterations, e.g., change the `warmup_iters` from 500 to 1000 or 2000.
4. Add gradient clipping: some models requires gradient clipping to stabilize the training process. The default of `grad_clip` is `None`, you can add gradient clippint to avoid gradients that are too large, i.e., set `optimizer_config=dict(_delete_=True, grad_clip=dict(max_norm=35, norm_type=2))` in your config file. If your config does not inherits from any basic config that contains `optimizer_config=dict(grad_clip=None)`, you can simply add `optimizer_config=dict(grad_clip=dict(max_norm=35, norm_type=2))`.

- 'GPU out of memory"
  1. There are some scenarios when there are large amount of ground truth boxes, which may cause OOM during target assignment. You can set `gpu_assign_thr=N` in the config of assigner thus the assigner will calculate box overlaps through CPU when there are more than N GT boxes.
  2. Set `with_cp=True` in the backbone. This uses the sublinear strategy in PyTorch to reduce GPU memory cost in the backbone.
  3. Try mixed precision training using following the examples in `config/fp16`. The `loss_scale` might need further tuning for different models.
- "RuntimeError: Expected to have finished reduction in the prior iteration before starting a new one"
  1. This error indicates that your module has parameters that were not used in producing loss. This phenomenon may be caused by running different branches in your code in DDP mode.
  2. You can set `find_unused_parameters = True` in the config to solve the above problems or find those unused parameters manually.

34.4 Evaluation

- COCO Dataset, AP or AR = -1
  1. According to the definition of COCO dataset, the small and medium areas in an image are less than 1024 (32*32), 9216 (96*96), respectively.
  2. If the corresponding area has no object, the result of AP and AR will set to -1.

CHAPTER

THIRTYFIVE

ENGLISH

CHAPTER

THIRTYSIX

CHAPTER

THIRTYSEVEN

MMDET.APIS

async mmdet.apis.**async_inference_detector**(*model*, *imgs*)
Async inference image(s) with the detector.

Parameters

- **model** (*nn.Module*) – The loaded detector.
- **img** (*str | ndarray*) – Either image files or loaded images.

Returns Awaitable detection results.

mmdet.apis.**get_root_logger**(*log_file=None*, *log_level=20*)
Get root logger.

Parameters

- **log_file** (*str, optional*) – File path of log. Defaults to None.
- **log_level** (*int, optional*) – The level of logger. Defaults to logging.INFO.

Returns The obtained logger

Return type logging.Logger

mmdet.apis.**inference_detector**(*model*, *imgs*)
Inference image(s) with the detector.

Parameters

- **model** (*nn.Module*) – The loaded detector.
- **imgs** (*str/ndarray or list[str/ndarray] or tuple[str/ndarray]*) – Either image files or loaded images.

Returns If imgs is a list or tuple, the same length list type results will be returned, otherwise return the detection results directly.

mmdet.apis.**init_detector**(*config*, *checkpoint=None*, *device='cuda:0'*, *cfg_options=None*)
Initialize a detector from config file.

Parameters

- **config** (str or mmcv.Config) – Config file path or the config object.
- **checkpoint** (*str, optional*) – Checkpoint path. If left as None, the model will not load any weights.
- **cfg_options** (*dict*) – Options to override some settings in the used config.

Returns The constructed detector.

Return type nn.Module

mmdet.apis.**init_random_seed**(*seed=None*, *device='cuda'*)
Initialize random seed.

If the seed is not set, the seed will be automatically randomized, and then broadcast to all processes to prevent some potential bugs.

Parameters

- **seed** (*int, Optional*) – The seed. Default to None.
- **device** (*str*) – The device where the seed will be put on. Default to 'cuda'.

Returns Seed to be used.

Return type int

mmdet.apis.**multi_gpu_test**(*model*, *data_loader*, *tmpdir=None*, *gpu_collect=False*)
Test model with multiple gpus.

This method tests model with multiple gpus and collects the results under two different modes: gpu and cpu modes. By setting 'gpu_collect=True' it encodes results to gpu tensors and use gpu communication for results collection. On cpu mode it saves the results on different gpus to 'tmpdir' and collects them by the rank 0 worker.

Parameters

- **model** (*nn.Module*) – Model to be tested.
- **data_loader** (*nn.Dataloader*) – Pytorch data loader.
- **tmpdir** (*str*) – Path of directory to save the temporary results from different gpus under cpu mode.
- **gpu_collect** (*bool*) – Option to use either gpu or cpu to collect results.

Returns The prediction results.

Return type list

mmdet.apis.**set_random_seed**(*seed*, *deterministic=False*)
Set random seed.

Parameters

- **seed** (*int*) – Seed to be used.
- **deterministic** (*bool*) – Whether to set the deterministic option for CUDNN backend, i.e., set *torch.backends.cudnn.deterministic* to True and *torch.backends.cudnn.benchmark* to False. Default: False.

mmdet.apis.**show_result_pyplot**(*model*, *img*, *result*, *score_thr=0.3*, *title='result'*, *wait_time=0*)
Visualize the detection results on the image.

Parameters

- **model** (*nn.Module*) – The loaded detector.
- **img** (*str or np.ndarray*) – Image filename or loaded image.
- **result** (*tuple[list] or list*) – The detection result, can be either (bbox, segm) or just bbox.
- **score_thr** (*float*) – The threshold to visualize the bboxes and masks.
- **title** (*str*) – Title of the pyplot figure.
- **wait_time** (*float*) – Value of waitKey param. Default: 0.

CHAPTER

THIRTYEIGHT

MMDET.CORE

38.1 anchor

class mmdet.core.anchor.**AnchorGenerator**(*strides*, *ratios*, *scales=None*, *base_sizes=None*, *scale_major=True*, *octave_base_scale=None*, *scales_per_octave=None*, *centers=None*, *center_offset=0.0*)

Standard anchor generator for 2D anchor-based detectors.

Parameters

- **strides** (*list[int] | list[tuple[int, int]]*) – Strides of anchors in multiple feature levels in order (w, h).
- **ratios** (*list[float]*) – The list of ratios between the height and width of anchors in a single level.
- **scales** (*list[int] | None*) – Anchor scales for anchors in a single level. It cannot be set at the same time if *octave_base_scale* and *scales_per_octave* are set.
- **base_sizes** (*list[int] | None*) – The basic sizes of anchors in multiple levels. If None is given, strides will be used as base_sizes. (If strides are non square, the shortest stride is taken.)
- **scale_major** (*bool*) – Whether to multiply scales first when generating base anchors. If true, the anchors in the same row will have the same scales. By default it is True in V2.0
- **octave_base_scale** (*int*) – The base scale of octave.
- **scales_per_octave** (*int*) – Number of scales for each octave. *octave_base_scale* and *scales_per_octave* are usually used in retinanet and the *scales* should be None when they are set.
- **centers** (*list[tuple[float, float]] | None*) – The centers of the anchor relative to the feature grid center in multiple feature levels. By default it is set to be None and not used. If a list of tuple of float is given, they will be used to shift the centers of anchors.
- **center_offset** (*float*) – The offset of center in proportion to anchors' width and height. By default it is 0 in V2.0.

Examples

```
>>> from mmdet.core import AnchorGenerator
>>> self = AnchorGenerator([16], [1.], [1.], [9])
>>> all_anchors = self.grid_priors([(2, 2)], device='cpu')
>>> print(all_anchors)
[tensor([[-4.5000, -4.5000,  4.5000,  4.5000],
        [11.5000, -4.5000, 20.5000,  4.5000],
        [-4.5000, 11.5000,  4.5000, 20.5000],
        [11.5000, 11.5000, 20.5000, 20.5000]])]
>>> self = AnchorGenerator([16, 32], [1.], [1.], [9, 18])
>>> all_anchors = self.grid_priors([(2, 2), (1, 1)], device='cpu')
>>> print(all_anchors)
[tensor([[-4.5000, -4.5000,  4.5000,  4.5000],
        [11.5000, -4.5000, 20.5000,  4.5000],
        [-4.5000, 11.5000,  4.5000, 20.5000],
        [11.5000, 11.5000, 20.5000, 20.5000]]),         tensor([[-9., -9., 9., 9.
↪]])]
```

gen_base_anchors()
Generate base anchors.

Returns Base anchors of a feature grid in multiple feature levels.

Return type list(torch.Tensor)

gen_single_level_base_anchors(*base_size*, *scales*, *ratios*, *center=None*)
Generate base anchors of a single level.

Parameters

- **base_size** (*int | float*) – Basic size of an anchor.
- **scales** (*torch.Tensor*) – Scales of the anchor.
- **ratios** (*torch.Tensor*) – The ratio between between the height and width of anchors in a single level.
- **center** (*tuple[float], optional*) – The center of the base anchor related to a single feature grid. Defaults to None.

Returns Anchors in a single-level feature maps.

Return type torch.Tensor

grid_anchors(*featmap_sizes*, *device='cuda'*)
Generate grid anchors in multiple feature levels.

Parameters

- **featmap_sizes** (*list[tuple]*) – List of feature map sizes in multiple feature levels.
- **device** (*str*) – Device where the anchors will be put on.

Returns Anchors in multiple feature levels. The sizes of each tensor should be [N, 4], where N = width * height * num_base_anchors, width and height are the sizes of the corresponding feature level, num_base_anchors is the number of anchors for that level.

Return type list[torch.Tensor]

grid_priors(*featmap_sizes*, *dtype=torch.float32*, *device='cuda'*)
Generate grid anchors in multiple feature levels.

Parameters

- **featmap_sizes** (*list[tuple]*) – List of feature map sizes in multiple feature levels.
- **dtype** (torch.dtype) – Dtype of priors. Default: torch.float32.
- **device** (*str*) – The device where the anchors will be put on.

Returns Anchors in multiple feature levels. The sizes of each tensor should be [N, 4], where N = width * height * num_base_anchors, width and height are the sizes of the corresponding feature level, num_base_anchors is the number of anchors for that level.

Return type list[torch.Tensor]

property num_base_anchors
total number of base anchors in a feature grid

Type list[int]

property num_base_priors
The number of priors (anchors) at a point on the feature grid

Type list[int]

property num_levels
number of feature levels that the generator will be applied

Type int

single_level_grid_anchors(*base_anchors*, *featmap_size*, *stride=(16, 16)*, *device='cuda'*)
Generate grid anchors of a single level.

Note: This function is usually called by method `self.grid_anchors`.

Parameters

- **base_anchors** (*torch.Tensor*) – The base anchors of a feature grid.
- **featmap_size** (*tuple[int]*) – Size of the feature maps.
- **stride** (*tuple[int], optional*) – Stride of the feature map in order (w, h). Defaults to (16, 16).
- **device** (*str, optional*) – Device the tensor will be put on. Defaults to 'cuda'.

Returns Anchors in the overall feature maps.

Return type torch.Tensor

single_level_grid_priors(*featmap_size*, *level_idx*, *dtype=torch.float32*, *device='cuda'*)
Generate grid anchors of a single level.

Note: This function is usually called by method `self.grid_priors`.

Parameters

- **featmap_size** (*tuple[int]*) – Size of the feature maps.
- **level_idx** (*int*) – The index of corresponding feature map level.
- **(obj** (*dtype*) – *torch.dtype*): Date type of points.Defaults to `torch.float32`.

- **device** (*str, optional*) – The device the tensor will be put on. Defaults to 'cuda'.

Returns Anchors in the overall feature maps.

Return type torch.Tensor

single_level_valid_flags(*featmap_size*, *valid_size*, *num_base_anchors*, *device='cuda'*)
Generate the valid flags of anchor in a single feature map.

Parameters

- **featmap_size** (*tuple[int]*) – The size of feature maps, arrange as (h, w).
- **valid_size** (*tuple[int]*) – The valid size of the feature maps.
- **num_base_anchors** (*int*) – The number of base anchors.
- **device** (*str, optional*) – Device where the flags will be put on. Defaults to 'cuda'.

Returns The valid flags of each anchor in a single level feature map.

Return type torch.Tensor

sparse_priors(*prior_idxs*, *featmap_size*, *level_idx*, *dtype=torch.float32*, *device='cuda'*)
Generate sparse anchors according to the `prior_idxs`.

Parameters

- **prior_idxs** (*Tensor*) – The index of corresponding anchors in the feature map.
- **featmap_size** (*tuple[int]*) – feature map size arrange as (h, w).
- **level_idx** (*int*) – The level index of corresponding feature map.
- **(obj** (*device*) – *torch.dtype*): Date type of points.Defaults to `torch.float32`.
- **(obj** – *torch.device*): The device where the points is located.

Returns

Anchor with shape (N, 4), N should be equal to the length of `prior_idxs`.

Return type Tensor

valid_flags(*featmap_sizes*, *pad_shape*, *device='cuda'*)
Generate valid flags of anchors in multiple feature levels.

Parameters

- **featmap_sizes** (*list(tuple)*) – List of feature map sizes in multiple feature levels.
- **pad_shape** (*tuple*) – The padded shape of the image.
- **device** (*str*) – Device where the anchors will be put on.

Returns Valid flags of anchors in multiple levels.

Return type list(torch.Tensor)

class mmdet.core.anchor.**LegacyAnchorGenerator**(*strides*, *ratios*, *scales=None*, *base_sizes=None*, *scale_major=True*, *octave_base_scale=None*, *scales_per_octave=None*, *centers=None*, *center_offset=0.0*)
Legacy anchor generator used in MMDetection V1.x.

Note: Difference to the V2.0 anchor generator:

1. The center offset of V1.x anchors are set to be 0.5 rather than 0.
2. The width/height are minused by 1 when calculating the anchors' centers and corners to meet the V1.x coordinate system.
3. The anchors' corners are quantized.

Parameters

- **strides** (*list[int] | list[tuple[int]]*) – Strides of anchors in multiple feature levels.
- **ratios** (*list[float]*) – The list of ratios between the height and width of anchors in a single level.
- **scales** (*list[int] | None*) – Anchor scales for anchors in a single level. It cannot be set at the same time if *octave_base_scale* and *scales_per_octave* are set.
- **base_sizes** (*list[int]*) – The basic sizes of anchors in multiple levels. If None is given, strides will be used to generate base_sizes.
- **scale_major** (*bool*) – Whether to multiply scales first when generating base anchors. If true, the anchors in the same row will have the same scales. By default it is True in V2.0
- **octave_base_scale** (*int*) – The base scale of octave.
- **scales_per_octave** (*int*) – Number of scales for each octave. *octave_base_scale* and *scales_per_octave* are usually used in retinanet and the *scales* should be None when they are set.
- **centers** (*list[tuple[float, float]] | None*) – The centers of the anchor relative to the feature grid center in multiple feature levels. By default it is set to be None and not used. It a list of float is given, this list will be used to shift the centers of anchors.
- **center_offset** (*float*) – The offset of center in proportion to anchors' width and height. By default it is 0.5 in V2.0 but it should be 0.5 in v1.x models.

Examples

```
>>> from mmdet.core import LegacyAnchorGenerator
>>> self = LegacyAnchorGenerator(
>>>     [16], [1.], [1.], [9], center_offset=0.5)
>>> all_anchors = self.grid_anchors(((2, 2),), device='cpu')
>>> print(all_anchors)
[tensor([[ 0.,  0.,  8.,  8.],
        [16.,  0., 24.,  8.],
        [ 0., 16.,  8., 24.],
        [16., 16., 24., 24.]])]
```

gen_single_level_base_anchors(*base_size*, *scales*, *ratios*, *center=None*)

Generate base anchors of a single level.

Note: The width/height of anchors are minused by 1 when calculating the centers and corners to meet the V1.x coordinate system.

Parameters

- **base_size** (*int | float*) – Basic size of an anchor.
- **scales** (*torch.Tensor*) – Scales of the anchor.
- **ratios** (*torch.Tensor*) – The ratio between between the height. and width of anchors in a single level.
- **center** (*tuple[float], optional*) – The center of the base anchor related to a single feature grid. Defaults to None.

Returns Anchors in a single-level feature map.

Return type torch.Tensor

class mmdet.core.anchor.**MlvlPointGenerator**(*strides*, *offset=0.5*)

Standard points generator for multi-level (Mlvl) feature maps in 2D points-based detectors.

Parameters

- **strides** (*list[int] | list[tuple[int, int]]*) – Strides of anchors in multiple feature levels in order (w, h).
- **offset** (*float*) – The offset of points, the value is normalized with corresponding stride. Defaults to 0.5.

grid_priors(*featmap_sizes*, *dtype=torch.float32*, *device='cuda'*, *with_stride=False*)

Generate grid points of multiple feature levels.

Parameters

- **featmap_sizes** (*list[tuple]*) – List of feature map sizes in multiple feature levels, each size arrange as as (h, w).
- **dtype** (dtype) – Dtype of priors. Default: torch.float32.
- **device** (*str*) – The device where the anchors will be put on.
- **with_stride** (*bool*) – Whether to concatenate the stride to the last dimension of points.

Returns Points of multiple feature levels. The sizes of each tensor should be (N, 2) when with stride is `False`, where N = width * height, width and height are the sizes of the corresponding feature level, and the last dimension 2 represent (coord_x, coord_y), otherwise the shape should be (N, 4), and the last dimension 4 represent (coord_x, coord_y, stride_w, stride_h).

Return type list[torch.Tensor]

property num_base_priors

The number of priors (points) at a point on the feature grid

Type list[int]

property num_levels

number of feature levels that the generator will be applied

Type int

single_level_grid_priors(*featmap_size*, *level_idx*, *dtype=torch.float32*, *device='cuda'*, *with_stride=False*)

Generate grid Points of a single level.

Note: This function is usually called by method `self.grid_priors`.

Parameters

- **featmap_size** (*tuple[int]*) – Size of the feature maps, arrange as (h, w).
- **level_idx** (*int*) – The index of corresponding feature map level.
- **dtype** (dtype) – Dtype of priors. Default: torch.float32.
- **device** (*str, optional*) – The device the tensor will be put on. Defaults to 'cuda'.
- **with_stride** (*bool*) – Concatenate the stride to the last dimension of points.

Returns Points of single feature levels. The shape of tensor should be (N, 2) when with stride is `False`, where N = width * height, width and height are the sizes of the corresponding feature level, and the last dimension 2 represent (coord_x, coord_y), otherwise the shape should be (N, 4), and the last dimension 4 represent (coord_x, coord_y, stride_w, stride_h).

Return type Tensor

single_level_valid_flags(*featmap_size*, *valid_size*, *device='cuda'*)
Generate the valid flags of points of a single feature map.

Parameters

- **featmap_size** (*tuple[int]*) – The size of feature maps, arrange as as (h, w).
- **valid_size** (*tuple[int]*) – The valid size of the feature maps. The size arrange as as (h, w).
- **device** (*str, optional*) – The device where the flags will be put on. Defaults to 'cuda'.

Returns The valid flags of each points in a single level feature map.

Return type torch.Tensor

sparse_priors(*prior_idxs*, *featmap_size*, *level_idx*, *dtype=torch.float32*, *device='cuda'*)
Generate sparse points according to the `prior_idxs`.

Parameters

- **prior_idxs** (*Tensor*) – The index of corresponding anchors in the feature map.
- **featmap_size** (*tuple[int]*) – feature map size arrange as (w, h).
- **level_idx** (*int*) – The level index of corresponding feature map.
- **(obj** (*device*) – *torch.dtype*): Date type of points. Defaults to `torch.float32`.
- **(obj** – *torch.device*): The device where the points is located.

Returns Anchor with shape (N, 2), N should be equal to the length of `prior_idxs`. And last dimension 2 represent (coord_x, coord_y).

Return type Tensor

valid_flags(*featmap_sizes*, *pad_shape*, *device='cuda'*)
Generate valid flags of points of multiple feature levels.

Parameters

- **featmap_sizes** (*list(tuple)*) – List of feature map sizes in multiple feature levels, each size arrange as as (h, w).
- **pad_shape** (*tuple(int)*) – The padded shape of the image, arrange as (h, w).
- **device** (*str*) – The device where the anchors will be put on.

Returns Valid flags of points of multiple levels.

Return type list(torch.Tensor)

class mmdet.core.anchor.YOLOAnchorGenerator(*strides*, *base_sizes*)
Anchor generator for YOLO.

Parameters

- **strides** (*list[int] | list[tuple[int, int]]*) – Strides of anchors in multiple feature levels.
- **base_sizes** (*list[list[tuple[int, int]]]*) – The basic sizes of anchors in multiple levels.

gen_base_anchors()
Generate base anchors.

Returns Base anchors of a feature grid in multiple feature levels.

Return type list(torch.Tensor)

gen_single_level_base_anchors(*base_sizes_per_level*, *center=None*)
Generate base anchors of a single level.

Parameters

- **base_sizes_per_level** (*list[tuple[int, int]]*) – Basic sizes of anchors.
- **center** (*tuple[float], optional*) – The center of the base anchor related to a single feature grid. Defaults to None.

Returns Anchors in a single-level feature maps.

Return type torch.Tensor

property num_levels
number of feature levels that the generator will be applied

Type int

responsible_flags(*featmap_sizes*, *gt_bboxes*, *device='cuda'*)
Generate responsible anchor flags of grid cells in multiple scales.

Parameters

- **featmap_sizes** (*list(tuple)*) – List of feature map sizes in multiple feature levels.
- **gt_bboxes** (*Tensor*) – Ground truth boxes, shape (n, 4).
- **device** (*str*) – Device where the anchors will be put on.

Returns responsible flags of anchors in multiple level

Return type list(torch.Tensor)

single_level_responsible_flags(*featmap_size*, *gt_bboxes*, *stride*, *num_base_anchors*, *device='cuda'*)
Generate the responsible flags of anchor in a single feature map.

Parameters

- **featmap_size** (*tuple[int]*) – The size of feature maps.
- **gt_bboxes** (*Tensor*) – Ground truth boxes, shape (n, 4).
- **stride** (*tuple(int)*) – stride of current level
- **num_base_anchors** (*int*) – The number of base anchors.
- **device** (*str, optional*) – Device where the flags will be put on. Defaults to 'cuda'.

Returns The valid flags of each anchor in a single level feature map.

Return type torch.Tensor

mmdet.core.anchor.**anchor_inside_flags**(*flat_anchors*, *valid_flags*, *img_shape*, *allowed_border=0*)
Check whether the anchors are inside the border.

Parameters

- **flat_anchors** (*torch.Tensor*) – Flatten anchors, shape (n, 4).
- **valid_flags** (*torch.Tensor*) – An existing valid flags of anchors.
- **img_shape** (*tuple(int)*) – Shape of current image.
- **allowed_border** (*int, optional*) – The border to allow the valid anchor. Defaults to 0.

Returns Flags indicating whether the anchors are inside a valid range.

Return type torch.Tensor

mmdet.core.anchor.**calc_region**(*bbox*, *ratio*, *featmap_size=None*)
Calculate a proportional bbox region.

The bbox center are fixed and the new h' and w' is h * ratio and w * ratio.

Parameters

- **bbox** (*Tensor*) – Bboxes to calculate regions, shape (n, 4).
- **ratio** (*float*) – Ratio of the output region.
- **featmap_size** (*tuple*) – Feature map size used for clipping the boundary.

Returns x1, y1, x2, y2

Return type tuple

mmdet.core.anchor.**images_to_levels**(*target*, *num_levels*)
Convert targets by image to targets by feature level.

[target_img0, target_img1] -> [target_level0, target_level1, ...]

38.2 bbox

class mmdet.core.bbox.**AssignResult**(*num_gts*, *gt_inds*, *max_overlaps*, *labels=None*)
Stores assignments between predicted and truth boxes.

num_gts
the number of truth boxes considered when computing this assignment

Type int

gt_inds
for each predicted box indicates the 1-based index of the assigned truth box. 0 means unassigned and -1 means ignore.

Type LongTensor

max_overlaps
the iou between the predicted box and its assigned truth box.

Type FloatTensor

labels
If specified, for each predicted box indicates the category label of the assigned truth box.

Type None | LongTensor

Example

```
>>> # An assign result between 4 predicted boxes and 9 true boxes
>>> # where only two boxes were assigned.
>>> num_gts = 9
>>> max_overlaps = torch.LongTensor([0, .5, .9, 0])
>>> gt_inds = torch.LongTensor([-1, 1, 2, 0])
>>> labels = torch.LongTensor([0, 3, 4, 0])
>>> self = AssignResult(num_gts, gt_inds, max_overlaps, labels)
>>> print(str(self))  # xdoctest: +IGNORE_WANT
<AssignResult(num_gts=9, gt_inds.shape=(4,), max_overlaps.shape=(4,),
              labels.shape=(4,))>
>>> # Force addition of gt labels (when adding gt as proposals)
>>> new_labels = torch.LongTensor([3, 4, 5])
>>> self.add_gt_(new_labels)
>>> print(str(self))  # xdoctest: +IGNORE_WANT
<AssignResult(num_gts=9, gt_inds.shape=(7,), max_overlaps.shape=(7,),
              labels.shape=(7,))>
```

add_gt_(*gt_labels*)
Add ground truth as assigned results.

Parameters **gt_labels** (*torch.Tensor*) – Labels of gt boxes

get_extra_property(*key*)
Get user-defined property.

property info
a dictionary of info about the object

Type dict

property num_preds
the number of predictions in this assignment

Type int

classmethod random(***kwargs*)
Create random AssignResult for tests or debugging.

Parameters

- **num_preds** – number of predicted boxes
- **num_gts** – number of true boxes
- **p_ignore** (*float*) – probability of a predicted box assigned to an ignored truth
- **p_assigned** (*float*) – probability of a predicted box not being assigned
- **p_use_label** (*float | bool*) – with labels or not
- **rng** (*None | int | numpy.random.RandomState*) – seed or state

Returns Randomly generated assign results.

Return type *AssignResult*

Example

```
>>> from mmdet.core.bbox.assigners.assign_result import *  # NOQA
>>> self = AssignResult.random()
>>> print(self.info)
```

set_extra_property(*key*, *value*)
Set user-defined new property.

class mmdet.core.bbox.**BaseAssigner**
Base assigner that assigns boxes to ground truth boxes.

abstract assign(*bboxes*, *gt_bboxes*, *gt_bboxes_ignore=None*, *gt_labels=None*)
Assign boxes to either a ground truth boxes or a negative boxes.

class mmdet.core.bbox.**BaseBBoxCoder**(***kwargs*)
Base bounding box coder.

abstract decode(*bboxes*, *bboxes_pred*)
Decode the predicted bboxes according to prediction and base boxes.

abstract encode(*bboxes*, *gt_bboxes*)
Encode deltas between bboxes and ground truth boxes.

class mmdet.core.bbox.**BaseSampler**(*num*, *pos_fraction*, *neg_pos_ub=- 1*, *add_gt_as_proposals=True*, ***kwargs*)
Base class of samplers.

sample(*assign_result*, *bboxes*, *gt_bboxes*, *gt_labels=None*, ***kwargs*)
Sample positive and negative bboxes.

This is a simple implementation of bbox sampling given candidates, assigning results and ground truth bboxes.

Parameters

- **assign_result** (*AssignResult*) – Bbox assigning results.
- **bboxes** (*Tensor*) – Boxes to be sampled from.
- **gt_bboxes** (*Tensor*) – Ground truth bboxes.
- **gt_labels** (*Tensor, optional*) – Class labels of ground truth bboxes.

Returns Sampling result.

Return type *SamplingResult*

Example

```
>>> from mmdet.core.bbox import RandomSampler
>>> from mmdet.core.bbox import AssignResult
>>> from mmdet.core.bbox.demodata import ensure_rng, random_boxes
>>> rng = ensure_rng(None)
>>> assign_result = AssignResult.random(rng=rng)
>>> bboxes = random_boxes(assign_result.num_preds, rng=rng)
>>> gt_bboxes = random_boxes(assign_result.num_gts, rng=rng)
>>> gt_labels = None
>>> self = RandomSampler(num=32, pos_fraction=0.5, neg_pos_ub=-1,
```

(continues on next page)

(continued from previous page)

```
>>>                         add_gt_as_proposals=False)
>>> self = self.sample(assign_result, bboxes, gt_bboxes, gt_labels)
```

class mmdet.core.bbox.**BboxOverlaps2D**(*scale=1.0*, *dtype=None*)
2D Overlaps (e.g. IoUs, GIoUs) Calculator.

class mmdet.core.bbox.**CenterRegionAssigner**(*pos_scale*, *neg_scale*, *min_pos_iof=0.01*, *ignore_gt_scale=0.5*, *foreground_dominate=False*, *iou_calculator={'type': 'BboxOverlaps2D'}*)
Assign pixels at the center region of a bbox as positive.

Each proposals will be assigned with *-1*, *0*, or a positive integer indicating the ground truth index. - -1: negative samples - semi-positive numbers: positive sample, index (0-based) of assigned gt

Parameters

- **pos_scale** (*float*) – Threshold within which pixels are labelled as positive.
- **neg_scale** (*float*) – Threshold above which pixels are labelled as positive.
- **min_pos_iof** (*float*) – Minimum iof of a pixel with a gt to be labelled as positive. Default: 1e-2
- **ignore_gt_scale** (*float*) – Threshold within which the pixels are ignored when the gt is labelled as shadowed. Default: 0.5
- **foreground_dominate** (*bool*) – If True, the bbox will be assigned as positive when a gt's kernel region overlaps with another's shadowed (ignored) region, otherwise it is set as ignored. Default to False.

assign(*bboxes*, *gt_bboxes*, *gt_bboxes_ignore=None*, *gt_labels=None*)
Assign gt to bboxes.

This method assigns gts to every bbox (proposal/anchor), each bbox will be assigned with -1, or a semi-positive number. -1 means negative sample, semi-positive number is the index (0-based) of assigned gt.

Parameters

- **bboxes** (*Tensor*) – Bounding boxes to be assigned, shape(n, 4).
- **gt_bboxes** (*Tensor*) – Groundtruth boxes, shape (k, 4).
- **gt_bboxes_ignore** (*tensor, optional*) – Ground truth bboxes that are labelled as *ignored*, e.g., crowd boxes in COCO.
- **gt_labels** (*tensor, optional*) – Label of gt_bboxes, shape (num_gts,).

Returns The assigned result. Note that shadowed_labels of shape (N, 2) is also added as an *assign_result* attribute. *shadowed_labels* is a tensor composed of N pairs of anchor_ind, class_label], where N is the number of anchors that lie in the outer region of a gt, anchor_ind is the shadowed anchor index and class_label is the shadowed class label.

Return type *AssignResult*

Example

```
>>> self = CenterRegionAssigner(0.2, 0.2)
>>> bboxes = torch.Tensor([[0, 0, 10, 10], [10, 10, 20, 20]])
>>> gt_bboxes = torch.Tensor([[0, 0, 10, 10]])
>>> assign_result = self.assign(bboxes, gt_bboxes)
>>> expected_gt_inds = torch.LongTensor([1, 0])
>>> assert torch.all(assign_result.gt_inds == expected_gt_inds)
```

assign_one_hot_gt_indices(*is_bbox_in_gt_core*, *is_bbox_in_gt_shadow*, *gt_priority=None*)

Assign only one gt index to each prior box.

Gts with large gt_priority are more likely to be assigned.

Parameters

- **is_bbox_in_gt_core** (*Tensor*) – Bool tensor indicating the bbox center is in the core area of a gt (e.g. 0-0.2). Shape: (num_prior, num_gt).
- **is_bbox_in_gt_shadow** (*Tensor*) – Bool tensor indicating the bbox center is in the shadowed area of a gt (e.g. 0.2-0.5). Shape: (num_prior, num_gt).
- **gt_priority** (*Tensor*) – Priorities of gts. The gt with a higher priority is more likely to be assigned to the bbox when the bbox match with multiple gts. Shape: (num_gt,).

Returns

Returns (assigned_gt_inds, shadowed_gt_inds).

- assigned_gt_inds: The assigned gt index of each prior bbox (i.e. index from 1 to num_gts). Shape: (num_prior,).
- shadowed_gt_inds: shadowed gt indices. It is a tensor of shape (num_ignore, 2) with first column being the shadowed prior bbox indices and the second column the shadowed gt indices (1-based).

Return type tuple

get_gt_priorities(*gt_bboxes*)

Get gt priorities according to their areas.

Smaller gt has higher priority.

Parameters **gt_bboxes** (*Tensor*) – Ground truth boxes, shape (k, 4).

Returns The priority of gts so that gts with larger priority is more likely to be assigned. Shape (k,)

Return type Tensor

class mmdet.core.bbox.**CombinedSampler**(*pos_sampler*, *neg_sampler*, ***kwargs*)

A sampler that combines positive sampler and negative sampler.

class mmdet.core.bbox.**DeltaXYWHBBoxCoder**(*target_means=(0.0, 0.0, 0.0, 0.0)*, *target_stds=(1.0, 1.0, 1.0, 1.0)*, *clip_border=True*, *add_ctr_clamp=False*, *ctr_clamp=32*)

Delta XYWH BBox coder.

Following the practice in R-CNN, this coder encodes bbox (x1, y1, x2, y2) into delta (dx, dy, dw, dh) and decodes delta (dx, dy, dw, dh) back to original bbox (x1, y1, x2, y2).

Parameters

- **target_means** (*Sequence[float]*) – Denormalizing means of target for delta coordinates

- **target_stds** (*Sequence[float]*) – Denormalizing standard deviation of target for delta coordinates
- **clip_border** (*bool, optional*) – Whether clip the objects outside the border of the image. Defaults to True.
- **add_ctr_clamp** (*bool*) – Whether to add center clamp, when added, the predicted box is clamped is its center is too far away from the original anchor's center. Only used by YOLOF. Default False.
- **ctr_clamp** (*int*) – the maximum pixel shift to clamp. Only used by YOLOF. Default 32.

decode(*bboxes*, *pred_bboxes*, *max_shape=None*, *wh_ratio_clip=0.016*)

Apply transformation *pred_bboxes* to *boxes*.

Parameters

- **bboxes** (*torch.Tensor*) – Basic boxes. Shape (B, N, 4) or (N, 4)
- **pred_bboxes** (*Tensor*) – Encoded offsets with respect to each roi. Has shape (B, N, num_classes * 4) or (B, N, 4) or (N, num_classes * 4) or (N, 4). Note N = num_anchors * W * H when rois is a grid of anchors.Offset encoding follows[1].
- **(Sequence[int] or torch.Tensor or Sequence[** (*max_shape*) – Sequence[int]],optional): Maximum bounds for boxes, specifies (H, W, C) or (H, W). If bboxes shape is (B, N, 4), then the max_shape should be a Sequence[Sequence[int]] and the length of max_shape should also be B.
- **wh_ratio_clip** (*float, optional*) – The allowed ratio between width and height.

Returns Decoded boxes.

Return type torch.Tensor

encode(*bboxes*, *gt_bboxes*)

Get box regression transformation deltas that can be used to transform the `bboxes` into the `gt_bboxes`.

Parameters

- **bboxes** (*torch.Tensor*) – Source boxes, e.g., object proposals.
- **gt_bboxes** (*torch.Tensor*) – Target of the transformation, e.g., ground-truth boxes.

Returns Box transformation deltas

Return type torch.Tensor

class mmdet.core.bbox.**DistancePointBBoxCoder**(*clip_border=True*)

Distance Point BBox coder.

This coder encodes gt bboxes (x1, y1, x2, y2) into (top, bottom, left, right) and decode it back to the original.

Parameters **clip_border** (*bool, optional*) – Whether clip the objects outside the border of the image. Defaults to True.

decode(*points*, *pred_bboxes*, *max_shape=None*)

Decode distance prediction to bounding box.

Parameters

- **points** (*Tensor*) – Shape (B, N, 2) or (N, 2).
- **pred_bboxes** (*Tensor*) – Distance from the given point to 4 boundaries (left, top, right, bottom). Shape (B, N, 4) or (N, 4)

[1] https://gitlab.kitware.com/computer-vision/kwimage/-/blob/928cae35ca8/kwimage/structs/polygon.py#L379 # noqa: E501

- **(Sequence[int] or torch.Tensor or Sequence[** (*max_shape*) – Sequence[int]],optional): Maximum bounds for boxes, specifies (H, W, C) or (H, W). If priors shape is (B, N, 4), then the max_shape should be a Sequence[Sequence[int]], and the length of max_shape should also be B. Default None.

Returns Boxes with shape (N, 4) or (B, N, 4)

Return type Tensor

encode(*points*, *gt_bboxes*, *max_dis=None*, *eps=0.1*)

Encode bounding box to distances.

Parameters

- **points** (*Tensor*) – Shape (N, 2), The format is [x, y].
- **gt_bboxes** (*Tensor*) – Shape (N, 4), The format is "xyxy"
- **max_dis** (*float*) – Upper bound of the distance. Default None.
- **eps** (*float*) – a small value to ensure target < max_dis, instead <=. Default 0.1.

Returns Box transformation deltas. The shape is (N, 4).

Return type Tensor

class mmdet.core.bbox.**InstanceBalancedPosSampler**(*num*, *pos_fraction*, *neg_pos_ub=- 1*, *add_gt_as_proposals=True*, ***kwargs*)

Instance balanced sampler that samples equal number of positive samples for each instance.

class mmdet.core.bbox.**IoUBalancedNegSampler**(*num*, *pos_fraction*, *floor_thr=- 1*, *floor_fraction=0*, *num_bins=3*, ***kwargs*)

IoU Balanced Sampling.

arXiv: https://arxiv.org/pdf/1904.02701.pdf (CVPR 2019)

Sampling proposals according to their IoU. *floor_fraction* of needed RoIs are sampled from proposals whose IoU are lower than *floor_thr* randomly. The others are sampled from proposals whose IoU are higher than *floor_thr*. These proposals are sampled from some bins evenly, which are split by *num_bins* via IoU evenly.

Parameters

- **num** (*int*) – number of proposals.
- **pos_fraction** (*float*) – fraction of positive proposals.
- **floor_thr** (*float*) – threshold (minimum) IoU for IoU balanced sampling, set to -1 if all using IoU balanced sampling.
- **floor_fraction** (*float*) – sampling fraction of proposals under floor_thr.
- **num_bins** (*int*) – number of bins in IoU balanced sampling.

sample_via_interval(*max_overlaps*, *full_set*, *num_expected*)

Sample according to the iou interval.

Parameters

- **max_overlaps** (*torch.Tensor*) – IoU between bounding boxes and ground truth boxes.
- **full_set** (*set(int)*) – A full set of indices of boxes
- **num_expected** (*int*) – Number of expected samples

Returns Indices of samples

Return type np.ndarray

class mmdet.core.bbox.**MaxIoUAssigner**(*pos_iou_thr*, *neg_iou_thr*, *min_pos_iou=0.0*, *gt_max_assign_all=True*, *ignore_iof_thr=-1*, *ignore_wrt_candidates=True*, *match_low_quality=True*, *gpu_assign_thr=-1*, *iou_calculator={'type': 'BboxOverlaps2D'}*)

Assign a corresponding gt bbox or background to each bbox.

Each proposals will be assigned with *-1*, or a semi-positive integer indicating the ground truth index.

- -1: negative sample, no assigned gt
- semi-positive integer: positive sample, index (0-based) of assigned gt

Parameters

- **pos_iou_thr** (*float*) – IoU threshold for positive bboxes.
- **neg_iou_thr** (*float or tuple*) – IoU threshold for negative bboxes.
- **min_pos_iou** (*float*) – Minimum iou for a bbox to be considered as a positive bbox. Positive samples can have smaller IoU than pos_iou_thr due to the 4th step (assign max IoU sample to each gt).
- **gt_max_assign_all** (*bool*) – Whether to assign all bboxes with the same highest overlap with some gt to that gt.
- **ignore_iof_thr** (*float*) – IoF threshold for ignoring bboxes (if *gt_bboxes_ignore* is specified). Negative values mean not ignoring any bboxes.
- **ignore_wrt_candidates** (*bool*) – Whether to compute the iof between *bboxes* and *gt_bboxes_ignore*, or the contrary.
- **match_low_quality** (*bool*) – Whether to allow low quality matches. This is usually allowed for RPN and single stage detectors, but not allowed in the second stage. Details are demonstrated in Step 4.
- **gpu_assign_thr** (*int*) – The upper bound of the number of GT for GPU assign. When the number of gt is above this threshold, will assign on CPU device. Negative values mean not assign on CPU.

assign(*bboxes*, *gt_bboxes*, *gt_bboxes_ignore=None*, *gt_labels=None*)

Assign gt to bboxes.

This method assign a gt bbox to every bbox (proposal/anchor), each bbox will be assigned with -1, or a semi-positive number. -1 means negative sample, semi-positive number is the index (0-based) of assigned gt. The assignment is done in following steps, the order matters.

1. assign every bbox to the background
2. assign proposals whose iou with all gts < neg_iou_thr to 0
3. for each bbox, if the iou with its nearest gt >= pos_iou_thr, assign it to that bbox
4. for each gt bbox, assign its nearest proposals (may be more than one) to itself

Parameters

- **bboxes** (*Tensor*) – Bounding boxes to be assigned, shape(n, 4).
- **gt_bboxes** (*Tensor*) – Groundtruth boxes, shape (k, 4).
- **gt_bboxes_ignore** (*Tensor, optional*) – Ground truth bboxes that are labelled as *ignored*, e.g., crowd boxes in COCO.
- **gt_labels** (*Tensor, optional*) – Label of gt_bboxes, shape (k,).

Returns The assign result.

Return type *AssignResult*

Example

```
>>> self = MaxIoUAssigner(0.5, 0.5)
>>> bboxes = torch.Tensor([[0, 0, 10, 10], [10, 10, 20, 20]])
>>> gt_bboxes = torch.Tensor([[0, 0, 10, 9]])
>>> assign_result = self.assign(bboxes, gt_bboxes)
>>> expected_gt_inds = torch.LongTensor([1, 0])
>>> assert torch.all(assign_result.gt_inds == expected_gt_inds)
```

assign_wrt_overlaps(*overlaps*, *gt_labels=None*)
Assign w.r.t. the overlaps of bboxes with gts.

Parameters

- **overlaps** (*Tensor*) – Overlaps between k gt_bboxes and n bboxes, shape(k, n).
- **gt_labels** (*Tensor, optional*) – Labels of k gt_bboxes, shape (k,).

Returns The assign result.

Return type *AssignResult*

class mmdet.core.bbox.**OHEMSampler**(*num*, *pos_fraction*, *context*, *neg_pos_ub=- 1*, *add_gt_as_proposals=True*, *loss_key='loss_cls'*, ***kwargs*)
Online Hard Example Mining Sampler described in Training Region-based Object Detectors with Online Hard Example Mining.

class mmdet.core.bbox.**PseudoBBoxCoder**(***kwargs*)
Pseudo bounding box coder.

decode(*bboxes*, *pred_bboxes*)
torch.Tensor: return the given `pred_bboxes`

encode(*bboxes*, *gt_bboxes*)
torch.Tensor: return the given `bboxes`

class mmdet.core.bbox.**PseudoSampler**(***kwargs*)
A pseudo sampler that does not do sampling actually.

sample(*assign_result*, *bboxes*, *gt_bboxes*, ***kwargs*)
Directly returns the positive and negative indices of samples.

Parameters

- **assign_result** (*AssignResult*) – Assigned results
- **bboxes** (*torch.Tensor*) – Bounding boxes
- **gt_bboxes** (*torch.Tensor*) – Ground truth boxes

Returns sampler results

Return type *SamplingResult*

class mmdet.core.bbox.**RandomSampler**(*num*, *pos_fraction*, *neg_pos_ub=- 1*, *add_gt_as_proposals=True*, ***kwargs*)
Random sampler.

Parameters

- **num** (*int*) – Number of samples
- **pos_fraction** (*float*) – Fraction of positive samples
- **neg_pos_up** (*int, optional*) – Upper bound number of negative and positive samples. Defaults to -1.
- **add_gt_as_proposals** (*bool, optional*) – Whether to add ground truth boxes as proposals. Defaults to True.

random_choice(*gallery*, *num*)

Random select some elements from the gallery.

If *gallery* is a Tensor, the returned indices will be a Tensor; If *gallery* is a ndarray or list, the returned indices will be a ndarray.

Parameters

- **gallery** (*Tensor | ndarray | list*) – indices pool.
- **num** (*int*) – expected sample num.

Returns sampled indices.

Return type Tensor or ndarray

class mmdet.core.bbox.**RegionAssigner**(*center_ratio=0.2*, *ignore_ratio=0.5*)

Assign a corresponding gt bbox or background to each bbox.

Each proposals will be assigned with *-1*, *0*, or a positive integer indicating the ground truth index.

- -1: don't care
- 0: negative sample, no assigned gt
- positive integer: positive sample, index (1-based) of assigned gt

Parameters

- **center_ratio** – ratio of the region in the center of the bbox to define positive sample.
- **ignore_ratio** – ratio of the region to define ignore samples.

assign(*mlvl_anchors*, *mlvl_valid_flags*, *gt_bboxes*, *img_meta*, *featmap_sizes*, *anchor_scale*, *anchor_strides*, *gt_bboxes_ignore=None*, *gt_labels=None*, *allowed_border=0*)

Assign gt to anchors.

This method assign a gt bbox to every bbox (proposal/anchor), each bbox will be assigned with -1, 0, or a positive number. -1 means don't care, 0 means negative sample, positive number is the index (1-based) of assigned gt.

The assignment is done in following steps, and the order matters.

1. Assign every anchor to 0 (negative)
2. (For each gt_bboxes) Compute ignore flags based on ignore_region then assign -1 to anchors w.r.t. ignore flags
3. (For each gt_bboxes) Compute pos flags based on center_region then assign gt_bboxes to anchors w.r.t. pos flags
4. (For each gt_bboxes) Compute ignore flags based on adjacent anchor level then assign -1 to anchors w.r.t. ignore flags
5. Assign anchor outside of image to -1

Parameters

- **mlvl_anchors** (*list[Tensor]*) – Multi level anchors.
- **mlvl_valid_flags** (*list[Tensor]*) – Multi level valid flags.
- **gt_bboxes** (*Tensor*) – Ground truth bboxes of image
- **img_meta** (*dict*) – Meta info of image.
- **featmap_sizes** (*list[Tensor]*) – Feature mapsize each level
- **anchor_scale** (*int*) – Scale of the anchor.
- **anchor_strides** (*list[int]*) – Stride of the anchor.
- **gt_bboxes** – Groundtruth boxes, shape (k, 4).
- **gt_bboxes_ignore** (*Tensor, optional*) – Ground truth bboxes that are labelled as *ignored*, e.g., crowd boxes in COCO.
- **gt_labels** (*Tensor, optional*) – Label of gt_bboxes, shape (k,).
- **allowed_border** (*int, optional*) – The border to allow the valid anchor. Defaults to 0.

Returns The assign result.

Return type *AssignResult*

class mmdet.core.bbox.**SamplingResult**(*pos_inds*, *neg_inds*, *bboxes*, *gt_bboxes*, *assign_result*, *gt_flags*)

Bbox sampling result.

Example

```
>>> # xdoctest: +IGNORE_WANT
>>> from mmdet.core.bbox.samplers.sampling_result import *  # NOQA
>>> self = SamplingResult.random(rng=10)
>>> print(f'self = {self}')
self = <SamplingResult({
    'neg_bboxes': torch.Size([12, 4]),
    'neg_inds': tensor([ 0,  1,  2,  4,  5,  6,  7,  8,  9, 10, 11, 12]),
    'num_gts': 4,
    'pos_assigned_gt_inds': tensor([], dtype=torch.int64),
    'pos_bboxes': torch.Size([0, 4]),
    'pos_inds': tensor([], dtype=torch.int64),
    'pos_is_gt': tensor([], dtype=torch.uint8)
})>
```

property bboxes

concatenated positive and negative boxes

Type torch.Tensor

property info

Returns a dictionary of info about the object.

classmethod random(*rng=None*, ***kwargs*)

Parameters

- **rng** (*None | int | numpy.random.RandomState*) – seed or state.
- **kwargs** (*keyword arguments*) –
  - num_preds: number of predicted boxes
  - num_gts: number of true boxes
  - p_ignore (float): probability of a predicted box assigned to an ignored truth.
  - p_assigned (float): probability of a predicted box not being assigned.
  - p_use_label (float | bool): with labels or not.

Returns Randomly generated sampling result.

Return type *SamplingResult*

Example

```
>>> from mmdet.core.bbox.samplers.sampling_result import *  # NOQA
>>> self = SamplingResult.random()
>>> print(self.__dict__)
```

to(*device*)
Change the device of the data inplace.

Example

```
>>> self = SamplingResult.random()
>>> print(f'self = {self.to(None)}')
>>> # xdoctest: +REQUIRES(--gpu)
>>> print(f'self = {self.to(0)}')
```

class mmdet.core.bbox.**ScoreHLRSampler**(*num*, *pos_fraction*, *context*, *neg_pos_ub=- 1*, *add_gt_as_proposals=True*, *k=0.5*, *bias=0*, *score_thr=0.05*, *iou_thr=0.5*, ***kwargs*)

Importance-based Sample Reweighting (ISR_N), described in Prime Sample Attention in Object Detection.

Score hierarchical local rank (HLR) differentiates with RandomSampler in negative part. It firstly computes Score-HLR in a two-step way, then linearly maps score hlr to the loss weights.

Parameters

- **num** (*int*) – Total number of sampled RoIs.
- **pos_fraction** (*float*) – Fraction of positive samples.
- **context** (BaseRoIHead) – RoI head that the sampler belongs to.
- **neg_pos_ub** (*int*) – Upper bound of the ratio of num negative to num positive, -1 means no upper bound.
- **add_gt_as_proposals** (*bool*) – Whether to add ground truth as proposals.
- **k** (*float*) – Power of the non-linear mapping.
- **bias** (*float*) – Shift of the non-linear mapping.
- **score_thr** (*float*) – Minimum score that a negative sample is to be considered as valid bbox.

static random_choice(*gallery*, *num*)
Randomly select some elements from the gallery.

If *gallery* is a Tensor, the returned indices will be a Tensor; If *gallery* is a ndarray or list, the returned indices will be a ndarray.

Parameters

- **gallery** (*Tensor | ndarray | list*) – indices pool.
- **num** (*int*) – expected sample num.

Returns sampled indices.

Return type Tensor or ndarray

sample(*assign_result*, *bboxes*, *gt_bboxes*, *gt_labels=None*, *img_meta=None*, ***kwargs*)
Sample positive and negative bboxes.

This is a simple implementation of bbox sampling given candidates, assigning results and ground truth bboxes.

Parameters

- **assign_result** (*AssignResult*) – Bbox assigning results.
- **bboxes** (*Tensor*) – Boxes to be sampled from.
- **gt_bboxes** (*Tensor*) – Ground truth bboxes.
- **gt_labels** (*Tensor, optional*) – Class labels of ground truth bboxes.

Returns

Sampling result and negative label weights.

Return type tuple[*SamplingResult*, Tensor]

class mmdet.core.bbox.**TBLRBBoxCoder**(*normalizer=4.0*, *clip_border=True*)
TBLR BBox coder.

Following the practice in FSAF, this coder encodes gt bboxes (x1, y1, x2, y2) into (top, bottom, left, right) and decode it back to the original.

Parameters

- **normalizer** (*list | float*) – Normalization factor to be divided with when coding the coordinates. If it is a list, it should have length of 4 indicating normalization factor in tblr dims. Otherwise it is a unified float factor for all dims. Default: 4.0
- **clip_border** (*bool, optional*) – Whether clip the objects outside the border of the image. Defaults to True.

decode(*bboxes*, *pred_bboxes*, *max_shape=None*)
Apply transformation *pred_bboxes* to *boxes*.

Parameters

- **bboxes** (*torch.Tensor*) – Basic boxes.Shape (B, N, 4) or (N, 4)
- **pred_bboxes** (*torch.Tensor*) – Encoded boxes with shape (B, N, 4) or (N, 4)
- **(Sequence[int] or torch.Tensor or Sequence[** (*max_shape*) – Sequence[int]],optional): Maximum bounds for boxes, specifies (H, W, C) or (H, W). If bboxes shape is (B, N, 4), then the max_shape should be a Sequence[Sequence[int]] and the length of max_shape should also be B.

Returns Decoded boxes.

Return type torch.Tensor

encode(*bboxes*, *gt_bboxes*)
Get box regression transformation deltas that can be used to transform the `bboxes` into the `gt_bboxes` in the (top, left, bottom, right) order.

Parameters

- **bboxes** (*torch.Tensor*) – source boxes, e.g., object proposals.
- **gt_bboxes** (*torch.Tensor*) – target of the transformation, e.g., ground truth boxes.

Returns Box transformation deltas

Return type torch.Tensor

mmdet.core.bbox.**bbox2distance**(*points*, *bbox*, *max_dis=None*, *eps=0.1*)
Decode bounding box based on distances.

Parameters

- **points** (*Tensor*) – Shape (n, 2), [x, y].
- **bbox** (*Tensor*) – Shape (n, 4), "xyxy" format
- **max_dis** (*float*) – Upper bound of the distance.
- **eps** (*float*) – a small value to ensure target < max_dis, instead <=

Returns Decoded distances.

Return type Tensor

mmdet.core.bbox.**bbox2result**(*bboxes*, *labels*, *num_classes*)
Convert detection results to a list of numpy arrays.

Parameters

- **bboxes** (*torch.Tensor | np.ndarray*) – shape (n, 5)
- **labels** (*torch.Tensor | np.ndarray*) – shape (n,)
- **num_classes** (*int*) – class number, including background class

Returns bbox results of each class

Return type list(ndarray)

mmdet.core.bbox.**bbox2roi**(*bbox_list*)
Convert a list of bboxes to roi format.

Parameters **bbox_list** (*list[Tensor]*) – a list of bboxes corresponding to a batch of images.

Returns shape (n, 5), [batch_ind, x1, y1, x2, y2]

Return type Tensor

mmdet.core.bbox.**bbox_cxcywh_to_xyxy**(*bbox*)
Convert bbox coordinates from (cx, cy, w, h) to (x1, y1, x2, y2).

Parameters **bbox** (*Tensor*) – Shape (n, 4) for bboxes.

Returns Converted bboxes.

Return type Tensor

mmdet.core.bbox.**bbox_flip**(*bboxes*, *img_shape*, *direction='horizontal'*)
Flip bboxes horizontally or vertically.

Parameters

- **bboxes** (*Tensor*) – Shape (..., 4*k)
- **img_shape** (*tuple*) – Image shape.
- **direction** (*str*) – Flip direction, options are "horizontal", "vertical", "diagonal". Default: "horizontal"

Returns Flipped bboxes.

Return type Tensor

mmdet.core.bbox.**bbox_mapping**(*bboxes*, *img_shape*, *scale_factor*, *flip*, *flip_direction='horizontal'*)
Map bboxes from the original image scale to testing scale.

mmdet.core.bbox.**bbox_mapping_back**(*bboxes*, *img_shape*, *scale_factor*, *flip*, *flip_direction='horizontal'*)
Map bboxes from testing scale to original image scale.

mmdet.core.bbox.**bbox_overlaps**(*bboxes1*, *bboxes2*, *mode='iou'*, *is_aligned=False*, *eps=1e-06*)
Calculate overlap between two set of bboxes.

FP16 Contributed by https://github.com/open-mmlab/mmdetection/pull/4889 .. note:

```
Assume bboxes1 is M x 4, bboxes2 is N x 4, when mode is 'iou',
there are some new generated variable when calculating IOU
using bbox_overlaps function:

1) is_aligned is False
    area1: M x 1
    area2: N x 1
    lt: M x N x 2
    rb: M x N x 2
    wh: M x N x 2
    overlap: M x N x 1
    union: M x N x 1
    ious: M x N x 1

    Total memory:
        S = (9 x N x M + N + M) * 4 Byte,

    When using FP16, we can reduce:
        R = (9 x N x M + N + M) * 4 / 2 Byte
        R large than (N + M) * 4 * 2 is always true when N and M >= 1.
        Obviously, N + M <= N * M < 3 * N * M, when N >=2 and M >=2,
                   N + 1 < 3 * N, when N or M is 1.

    Given M = 40 (ground truth), N = 400000 (three anchor boxes
    in per grid, FPN, R-CNNs),
        R = 275 MB (one times)

    A special case (dense detection), M = 512 (ground truth),
        R = 3516 MB = 3.43 GB

    When the batch size is B, reduce:
```

(continues on next page)

(continued from previous page)

```
        B x R

    Therefore, CUDA memory runs out frequently.

    Experiments on GeForce RTX 2080Ti (11019 MiB):

    |   dtype   |   M   |   N   |   Use    |   Real   |   Ideal   |
    |:----:|:----:|:----:|:----:|:----:|:----:|
    |   FP32   |   512 | 400000 | 8020 MiB |   --   |   --   |
    |   FP16   |   512 | 400000 |   4504 MiB | 3516 MiB | 3516 MiB |
    |   FP32   |   40 | 400000 |   1540 MiB |   --   |   --   |
    |   FP16   |   40 | 400000 |   1264 MiB |   276MiB   | 275 MiB |

2) is_aligned is True
    area1: N x 1
    area2: N x 1
    lt: N x 2
    rb: N x 2
    wh: N x 2
    overlap: N x 1
    union: N x 1
    ious: N x 1

    Total memory:
        S = 11 x N * 4 Byte

    When using FP16, we can reduce:
        R = 11 x N * 4 / 2 Byte

So do the 'giou' (large than 'iou').

Time-wise, FP16 is generally faster than FP32.

When gpu_assign_thr is not -1, it takes more time on cpu
but not reduce memory.
There, we can reduce half the memory and keep the speed.
```

If `is_aligned` is `False`, then calculate the overlaps between each bbox of bboxes1 and bboxes2, otherwise the overlaps between each aligned pair of bboxes1 and bboxes2.

Parameters

- **bboxes1** (*Tensor*) – shape (B, m, 4) in <x1, y1, x2, y2> format or empty.
- **bboxes2** (*Tensor*) – shape (B, n, 4) in <x1, y1, x2, y2> format or empty. B indicates the batch dim, in shape (B1, B2, ..., Bn). If `is_aligned` is `True`, then m and n must be equal.
- **mode** (*str*) – "iou" (intersection over union), "iof" (intersection over foreground) or "giou" (generalized intersection over union). Default "iou".
- **is_aligned** (*bool, optional*) – If True, then m and n must be equal. Default False.
- **eps** (*float, optional*) – A value added to the denominator for numerical stability. Default 1e-6.

Returns shape (m, n) if `is_aligned` is False else shape (m,)

Return type Tensor

Example

```
>>> bboxes1 = torch.FloatTensor([
>>>     [0, 0, 10, 10],
>>>     [10, 10, 20, 20],
>>>     [32, 32, 38, 42],
>>> ])
>>> bboxes2 = torch.FloatTensor([
>>>     [0, 0, 10, 20],
>>>     [0, 10, 10, 19],
>>>     [10, 10, 20, 20],
>>> ])
>>> overlaps = bbox_overlaps(bboxes1, bboxes2)
>>> assert overlaps.shape == (3, 3)
>>> overlaps = bbox_overlaps(bboxes1, bboxes2, is_aligned=True)
>>> assert overlaps.shape == (3, )
```

Example

```
>>> empty = torch.empty(0, 4)
>>> nonempty = torch.FloatTensor([[0, 0, 10, 9]])
>>> assert tuple(bbox_overlaps(empty, nonempty).shape) == (0, 1)
>>> assert tuple(bbox_overlaps(nonempty, empty).shape) == (1, 0)
>>> assert tuple(bbox_overlaps(empty, empty).shape) == (0, 0)
```

mmdet.core.bbox.**bbox_rescale**(*bboxes*, *scale_factor=1.0*)
Rescale bounding box w.r.t. scale_factor.

Parameters

- **bboxes** (*Tensor*) – Shape (n, 4) for bboxes or (n, 5) for rois
- **scale_factor** (*float*) – rescale factor

Returns Rescaled bboxes.

Return type Tensor

mmdet.core.bbox.**bbox_xyxy_to_cxcywh**(*bbox*)
Convert bbox coordinates from (x1, y1, x2, y2) to (cx, cy, w, h).

Parameters **bbox** (*Tensor*) – Shape (n, 4) for bboxes.

Returns Converted bboxes.

Return type Tensor

mmdet.core.bbox.**build_assigner**(*cfg*, ***default_args*)
Builder of box assigner.

mmdet.core.bbox.**build_bbox_coder**(*cfg*, ***default_args*)
Builder of box coder.

mmdet.core.bbox.**build_sampler**(*cfg*, ***default_args*)
Builder of box sampler.

mmdet.core.bbox.**distance2bbox**(*points*, *distance*, *max_shape=None*)
Decode distance prediction to bounding box.

Parameters

- **points** (*Tensor*) – Shape (B, N, 2) or (N, 2).
- **distance** (*Tensor*) – Distance from the given point to 4 boundaries (left, top, right, bottom). Shape (B, N, 4) or (N, 4)
- **(Sequence[int] or torch.Tensor or Sequence[** (*max_shape*) – Sequence[int]],optional): Maximum bounds for boxes, specifies (H, W, C) or (H, W). If priors shape is (B, N, 4), then the max_shape should be a Sequence[Sequence[int]] and the length of max_shape should also be B.

Returns Boxes with shape (N, 4) or (B, N, 4)

Return type Tensor

mmdet.core.bbox.**roi2bbox**(*rois*)
Convert rois to bounding box format.

Parameters **rois** (*torch.Tensor*) – RoIs with the shape (n, 5) where the first column indicates batch id of each RoI.

Returns Converted boxes of corresponding rois.

Return type list[torch.Tensor]

38.3 export

mmdet.core.export.**add_dummy_nms_for_onnx**(*boxes*, *scores*, *max_output_boxes_per_class=1000*, *iou_threshold=0.5*, *score_threshold=0.05*, *pre_top_k=- 1*, *after_top_k=- 1*, *labels=None*)
Create a dummy onnx::NonMaxSuppression op while exporting to ONNX.

This function helps exporting to onnx with batch and multiclass NMS op. It only supports class-agnostic detection results. That is, the scores is of shape (N, num_bboxes, num_classes) and the boxes is of shape (N, num_boxes, 4).

Parameters

- **boxes** (*Tensor*) – The bounding boxes of shape [N, num_boxes, 4]
- **scores** (*Tensor*) – The detection scores of shape [N, num_boxes, num_classes]
- **max_output_boxes_per_class** (*int*) – Maximum number of output boxes per class of nms. Defaults to 1000.
- **iou_threshold** (*float*) – IOU threshold of nms. Defaults to 0.5
- **score_threshold** (*float*) – score threshold of nms. Defaults to 0.05.
- **pre_top_k** (*bool*) – Number of top K boxes to keep before nms. Defaults to -1.
- **after_top_k** (*int*) – Number of top K boxes to keep after nms. Defaults to -1.
- **labels** (*Tensor, optional*) – It not None, explicit labels would be used. Otherwise, labels would be automatically generated using num_classed. Defaults to None.

Returns

dets of shape [N, num_det, 5] and class labels of shape [N, num_det].

Return type tuple[Tensor, Tensor]

mmdet.core.export.**build_model_from_cfg**(*config_path*, *checkpoint_path*, *cfg_options=None*)
Build a model from config and load the given checkpoint.

Parameters

- **config_path** (*str*) – the OpenMMLab config for the model we want to export to ONNX
- **checkpoint_path** (*str*) – Path to the corresponding checkpoint

Returns the built model

Return type torch.nn.Module

mmdet.core.export.**dynamic_clip_for_onnx**(*x1*, *y1*, *x2*, *y2*, *max_shape*)
Clip boxes dynamically for onnx.

Since torch.clamp cannot have dynamic *min* and *max*, we scale the boxes by 1/max_shape and clamp in the range [0, 1].

Parameters

- **x1** (*Tensor*) – The x1 for bounding boxes.
- **y1** (*Tensor*) – The y1 for bounding boxes.
- **x2** (*Tensor*) – The x2 for bounding boxes.
- **y2** (*Tensor*) – The y2 for bounding boxes.
- **max_shape** (*Tensor or torch.Size*) – The (H,W) of original image.

Returns The clipped x1, y1, x2, y2.

Return type tuple(Tensor)

mmdet.core.export.**generate_inputs_and_wrap_model**(*config_path*, *checkpoint_path*, *input_config*, *cfg_options=None*)
Prepare sample input and wrap model for ONNX export.

The ONNX export API only accept args, and all inputs should be torch.Tensor or corresponding types (such as tuple of tensor). So we should call this function before exporting. This function will:

1. generate corresponding inputs which are used to execute the model.
2. Wrap the model's forward function.

For example, the MMDet models' forward function has a parameter `return_loss:bool`. As we want to set it as False while export API supports neither bool type or kwargs. So we have to replace the forward method like `model.forward = partial(model.forward, return_loss=False)`.

Parameters

- **config_path** (*str*) – the OpenMMLab config for the model we want to export to ONNX
- **checkpoint_path** (*str*) – Path to the corresponding checkpoint
- **input_config** (*dict*) – the exactly data in this dict depends on the framework. For MMSeg, we can just declare the input shape, and generate the dummy data accordingly. However, for MMDet, we may pass the real img path, or the NMS will return None as there is no legal bbox.

Returns

(model, tensor_data) wrapped model which can be called by `model(*tensor_data)` and a list of inputs which are used to execute the model while exporting.

Return type tuple

`mmdet.core.export.get_k_for_topk(k, size)`
Get k of TopK for onnx exporting.

The K of TopK in TensorRT should not be a Tensor, while in ONNX Runtime it could be a Tensor.Due to dynamic shape feature, we have to decide whether to do TopK and what K it should be while exporting to ONNX.

If returned K is less than zero, it means we do not have to do TopK operation.

Parameters

- **k** (*int or Tensor*) – The set k value for nms from config file.
- **size** (*Tensor or torch.Size*) – The number of elements of TopK's input tensor

Returns (int or Tensor): The final K for TopK.

Return type tuple

`mmdet.core.export.preprocess_example_input(input_config)`
Prepare an example input image for `generate_inputs_and_wrap_model`.

Parameters **input_config** (*dict*) – customized config describing the example input.

Returns (one_img, one_meta), tensor of the example input image and meta information for the example input image.

Return type tuple

Examples

```
>>> from mmdet.core.export import preprocess_example_input
>>> input_config = {
>>>         'input_shape': (1,3,224,224),
>>>         'input_path': 'demo/demo.jpg',
>>>         'normalize_cfg': {
>>>             'mean': (123.675, 116.28, 103.53),
>>>             'std': (58.395, 57.12, 57.375)
>>>             }
>>>         }
>>> one_img, one_meta = preprocess_example_input(input_config)
>>> print(one_img.shape)
torch.Size([1, 3, 224, 224])
>>> print(one_meta)
{'img_shape': (224, 224, 3),
'ori_shape': (224, 224, 3),
'pad_shape': (224, 224, 3),
'filename': '<demo>.png',
'scale_factor': 1.0,
'flip': False}
```

38.4 mask

class mmdet.core.mask.**BaseInstanceMasks**

Base class for instance masks.

abstract property areas

areas of each instance.

Type ndarray

abstract crop(*bbox*)

Crop each mask by the given bbox.

Parameters **bbox** (*ndarray*) – Bbox in format [x1, y1, x2, y2], shape (4,).

Returns The cropped masks.

Return type *BaseInstanceMasks*

abstract crop_and_resize(*bboxes*, *out_shape*, *inds*, *device*, *interpolation='bilinear'*, *binarize=True*)

Crop and resize masks by the given bboxes.

This function is mainly used in mask targets computation. It firstly align mask to bboxes by assigned_inds, then crop mask by the assigned bbox and resize to the size of (mask_h, mask_w)

Parameters

- **bboxes** (*Tensor*) – Bboxes in format [x1, y1, x2, y2], shape (N, 4)
- **out_shape** (*tuple[int]*) – Target (h, w) of resized mask
- **inds** (*ndarray*) – Indexes to assign masks to each bbox, shape (N,) and values should be between [0, num_masks - 1].
- **device** (*str*) – Device of bboxes
- **interpolation** (*str*) – See *mmcv.imresize*
- **binarize** (*bool*) – if True fractional values are rounded to 0 or 1 after the resize operation. if False and unsupported an error will be raised. Defaults to True.

Returns the cropped and resized masks.

Return type *BaseInstanceMasks*

abstract expand(*expanded_h*, *expanded_w*, *top*, *left*)

see Expand.

abstract flip(*flip_direction='horizontal'*)

Flip masks alone the given direction.

Parameters **flip_direction** (*str*) – Either 'horizontal' or 'vertical'.

Returns The flipped masks.

Return type *BaseInstanceMasks*

abstract pad(*out_shape*, *pad_val*)

Pad masks to the given size of (h, w).

Parameters

- **out_shape** (*tuple[int]*) – Target (h, w) of padded mask.
- **pad_val** (*int*) – The padded value.

Returns The padded masks.

Return type *BaseInstanceMasks*

abstract rescale(*scale*, *interpolation='nearest'*)
Rescale masks as large as possible while keeping the aspect ratio. For details can refer to *mmcv.imrescale*.

Parameters

- **scale** (*tuple[int]*) – The maximum size (h, w) of rescaled mask.
- **interpolation** (*str*) – Same as `mmcv.imrescale()`.

Returns The rescaled masks.

Return type *BaseInstanceMasks*

abstract resize(*out_shape*, *interpolation='nearest'*)
Resize masks to the given out_shape.

Parameters

- **out_shape** – Target (h, w) of resized mask.
- **interpolation** (*str*) – See `mmcv.imresize()`.

Returns The resized masks.

Return type *BaseInstanceMasks*

abstract rotate(*out_shape*, *angle*, *center=None*, *scale=1.0*, *fill_val=0*)
Rotate the masks.

Parameters

- **out_shape** (*tuple[int]*) – Shape for output mask, format (h, w).
- **angle** (*int | float*) – Rotation angle in degrees. Positive values mean counter-clockwise rotation.
- **center** (*tuple[float], optional*) – Center point (w, h) of the rotation in source image. If not specified, the center of the image will be used.
- **scale** (*int | float*) – Isotropic scale factor.
- **fill_val** (*int | float*) – Border value. Default 0 for masks.

Returns Rotated masks.

shear(*out_shape*, *magnitude*, *direction='horizontal'*, *border_value=0*, *interpolation='bilinear'*)
Shear the masks.

Parameters

- **out_shape** (*tuple[int]*) – Shape for output mask, format (h, w).
- **magnitude** (*int | float*) – The magnitude used for shear.
- **direction** (*str*) – The shear direction, either "horizontal" or "vertical".
- **border_value** (*int | tuple[int]*) – Value used in case of a constant border. Default 0.
- **interpolation** (*str*) – Same as in `mmcv.imshear()`.

Returns Sheared masks.

Return type ndarray

abstract to_ndarray()
Convert masks to the format of ndarray.

Returns Converted masks in the format of ndarray.

Return type ndarray

abstract to_tensor(*dtype*, *device*)
Convert masks to the format of Tensor.

Parameters

- **dtype** (*str*) – Dtype of converted mask.
- **device** (*torch.device*) – Device of converted masks.

Returns Converted masks in the format of Tensor.

Return type Tensor

abstract translate(*out_shape*, *offset*, *direction='horizontal'*, *fill_val=0*, *interpolation='bilinear'*)
Translate the masks.

Parameters

- **out_shape** (*tuple[int]*) – Shape for output mask, format (h, w).
- **offset** (*int | float*) – The offset for translate.
- **direction** (*str*) – The translate direction, either "horizontal" or "vertical".
- **fill_val** (*int | float*) – Border value. Default 0.
- **interpolation** (*str*) – Same as `mmcv.imtranslate()`.

Returns Translated masks.

class mmdet.core.mask.**BitmapMasks**(*masks*, *height*, *width*)
This class represents masks in the form of bitmaps.

Parameters

- **masks** (*ndarray*) – ndarray of masks in shape (N, H, W), where N is the number of objects.
- **height** (*int*) – height of masks
- **width** (*int*) – width of masks

Example

```
>>> from mmdet.core.mask.structures import *  # NOQA
>>> num_masks, H, W = 3, 32, 32
>>> rng = np.random.RandomState(0)
>>> masks = (rng.rand(num_masks, H, W) > 0.1).astype(np.int)
>>> self = BitmapMasks(masks, height=H, width=W)
```

```
>>> # demo crop_and_resize
>>> num_boxes = 5
>>> bboxes = np.array([[0, 0, 30, 10.0]] * num_boxes)
>>> out_shape = (14, 14)
>>> inds = torch.randint(0, len(self), size=(num_boxes,))
>>> device = 'cpu'
>>> interpolation = 'bilinear'
>>> new = self.crop_and_resize(
...     bboxes, out_shape, inds, device, interpolation)
```

(continues on next page)

(continued from previous page)

```
>>> assert len(new) == num_boxes
>>> assert new.height, new.width == out_shape
```

property areas
See *BaseInstanceMasks.areas*.

crop(*bbox*)
See *BaseInstanceMasks.crop()*.

crop_and_resize(*bboxes*, *out_shape*, *inds*, *device='cpu'*, *interpolation='bilinear'*, *binarize=True*)
See *BaseInstanceMasks.crop_and_resize()*.

expand(*expanded_h*, *expanded_w*, *top*, *left*)
See *BaseInstanceMasks.expand()*.

flip(*flip_direction='horizontal'*)
See *BaseInstanceMasks.flip()*.

pad(*out_shape*, *pad_val=0*)
See *BaseInstanceMasks.pad()*.

classmethod random(*num_masks=3*, *height=32*, *width=32*, *dtype=<class 'numpy.uint8'>*, *rng=None*)
Generate random bitmap masks for demo / testing purposes.

Example

```
>>> from mmdet.core.mask.structures import BitmapMasks
>>> self = BitmapMasks.random()
>>> print('self = {}'.format(self))
self = BitmapMasks(num_masks=3, height=32, width=32)
```

rescale(*scale*, *interpolation='nearest'*)
See *BaseInstanceMasks.rescale()*.

resize(*out_shape*, *interpolation='nearest'*)
See *BaseInstanceMasks.resize()*.

rotate(*out_shape*, *angle*, *center=None*, *scale=1.0*, *fill_val=0*)
Rotate the BitmapMasks.

Parameters

- **out_shape** (*tuple[int]*) – Shape for output mask, format (h, w).
- **angle** (*int | float*) – Rotation angle in degrees. Positive values mean counter-clockwise rotation.
- **center** (*tuple[float], optional*) – Center point (w, h) of the rotation in source image. If not specified, the center of the image will be used.
- **scale** (*int | float*) – Isotropic scale factor.
- **fill_val** (*int | float*) – Border value. Default 0 for masks.

Returns Rotated BitmapMasks.

Return type *BitmapMasks*

shear(*out_shape*, *magnitude*, *direction='horizontal'*, *border_value=0*, *interpolation='bilinear'*)
Shear the BitmapMasks.

Parameters

- **out_shape** (*tuple[int]*) – Shape for output mask, format (h, w).
- **magnitude** (*int | float*) – The magnitude used for shear.
- **direction** (*str*) – The shear direction, either "horizontal" or "vertical".
- **border_value** (*int | tuple[int]*) – Value used in case of a constant border.
- **interpolation** (*str*) – Same as in `mmcv.imshear()`.

Returns The sheared masks.

Return type *BitmapMasks*

to_ndarray()
See *BaseInstanceMasks.to_ndarray()*.

to_tensor(*dtype*, *device*)
See *BaseInstanceMasks.to_tensor()*.

translate(*out_shape*, *offset*, *direction='horizontal'*, *fill_val=0*, *interpolation='bilinear'*)
Translate the BitmapMasks.

Parameters

- **out_shape** (*tuple[int]*) – Shape for output mask, format (h, w).
- **offset** (*int | float*) – The offset for translate.
- **direction** (*str*) – The translate direction, either "horizontal" or "vertical".
- **fill_val** (*int | float*) – Border value. Default 0 for masks.
- **interpolation** (*str*) – Same as `mmcv.imtranslate()`.

Returns Translated BitmapMasks.

Return type *BitmapMasks*

Example

```
>>> from mmdet.core.mask.structures import BitmapMasks
>>> self = BitmapMasks.random(dtype=np.uint8)
>>> out_shape = (32, 32)
>>> offset = 4
>>> direction = 'horizontal'
>>> fill_val = 0
>>> interpolation = 'bilinear'
>>> # Note, There seem to be issues when:
>>> # * out_shape is different than self's shape
>>> # * the mask dtype is not supported by cv2.AffineWarp
>>> new = self.translate(out_shape, offset, direction, fill_val,
>>>                      interpolation)
>>> assert len(new) == len(self)
>>> assert new.height, new.width == out_shape
```

class mmdet.core.mask.**PolygonMasks**(*masks*, *height*, *width*)
This class represents masks in the form of polygons.

Polygons is a list of three levels. The first level of the list corresponds to objects, the second level to the polys that compose the object, the third level to the poly coordinates

Parameters

- **masks** (*list[list[ndarray]]*) – The first level of the list corresponds to objects, the second level to the polys that compose the object, the third level to the poly coordinates
- **height** (*int*) – height of masks
- **width** (*int*) – width of masks

Example

```
>>> from mmdet.core.mask.structures import *  # NOQA
>>> masks = [
>>>     [ np.array([0, 0, 10, 0, 10, 10., 0, 10, 0, 0]) ]
>>> ]
>>> height, width = 16, 16
>>> self = PolygonMasks(masks, height, width)
```

```
>>> # demo translate
>>> new = self.translate((16, 16), 4., direction='horizontal')
>>> assert np.all(new.masks[0][0][1::2] == masks[0][0][1::2])
>>> assert np.all(new.masks[0][0][0::2] == masks[0][0][0::2] + 4)
```

```
>>> # demo crop_and_resize
>>> num_boxes = 3
>>> bboxes = np.array([[0, 0, 30, 10.0]] * num_boxes)
>>> out_shape = (16, 16)
>>> inds = torch.randint(0, len(self), size=(num_boxes,))
>>> device = 'cpu'
>>> interpolation = 'bilinear'
>>> new = self.crop_and_resize(
...     bboxes, out_shape, inds, device, interpolation)
>>> assert len(new) == num_boxes
>>> assert new.height, new.width == out_shape
```

property areas
Compute areas of masks.

This func is modified from detectron2. The function only works with Polygons using the shoelace formula.

Returns areas of each instance

Return type ndarray

crop(*bbox*)
see *BaseInstanceMasks.crop()*

crop_and_resize(*bboxes*, *out_shape*, *inds*, *device='cpu'*, *interpolation='bilinear'*, *binarize=True*)
see *BaseInstanceMasks.crop_and_resize()*

expand(**args*, ***kwargs*)
TODO: Add expand for polygon

flip(*flip_direction='horizontal'*)
see *BaseInstanceMasks.flip()*

pad(*out_shape*, *pad_val=0*)
padding has no effect on polygons`

classmethod random(*num_masks=3*, *height=32*, *width=32*, *n_verts=5*, *dtype=<class 'numpy.float32'>*, *rng=None*)
Generate random polygon masks for demo / testing purposes.

Adapted from[Page 202, 1]

References

Example

```
>>> from mmdet.core.mask.structures import PolygonMasks
>>> self = PolygonMasks.random()
>>> print('self = {}'.format(self))
```

rescale(*scale*, *interpolation=None*)
see *BaseInstanceMasks.rescale()*

resize(*out_shape*, *interpolation=None*)
see *BaseInstanceMasks.resize()*

rotate(*out_shape*, *angle*, *center=None*, *scale=1.0*, *fill_val=0*)
See *BaseInstanceMasks.rotate()*.

shear(*out_shape*, *magnitude*, *direction='horizontal'*, *border_value=0*, *interpolation='bilinear'*)
See *BaseInstanceMasks.shear()*.

to_bitmap()
convert polygon masks to bitmap masks.

to_ndarray()
Convert masks to the format of ndarray.

to_tensor(*dtype*, *device*)
See *BaseInstanceMasks.to_tensor()*.

translate(*out_shape*, *offset*, *direction='horizontal'*, *fill_val=None*, *interpolation=None*)
Translate the PolygonMasks.

Example

```
>>> self = PolygonMasks.random(dtype=np.int)
>>> out_shape = (self.height, self.width)
>>> new = self.translate(out_shape, 4., direction='horizontal')
>>> assert np.all(new.masks[0][0][1::2] == self.masks[0][0][1::2])
>>> assert np.all(new.masks[0][0][0::2] == self.masks[0][0][0::2] + 4)  # noqa:␣
↪E501
```

mmdet.core.mask.**encode_mask_results**(*mask_results*)
Encode bitmap mask to RLE code.

Parameters **mask_results** (*list | tuple[list]*) – bitmap mask results. In mask scoring rcnn, mask_results is a tuple of (segm_results, segm_cls_score).

Returns RLE encoded mask.

Return type list | tuple

mmdet.core.mask.**mask_target**(*pos_proposals_list*, *pos_assigned_gt_inds_list*, *gt_masks_list*, *cfg*)

Compute mask target for positive proposals in multiple images.

Parameters

- **pos_proposals_list** (*list[Tensor]*) – Positive proposals in multiple images.
- **pos_assigned_gt_inds_list** (*list[Tensor]*) – Assigned GT indices for each positive proposals.
- **gt_masks_list** (list[*BaseInstanceMasks*]) – Ground truth masks of each image.
- **cfg** (*dict*) – Config dict that specifies the mask size.

Returns Mask target of each image.

Return type list[Tensor]

Example

```
>>> import mmcv
>>> import mmdet
>>> from mmdet.core.mask import BitmapMasks
>>> from mmdet.core.mask.mask_target import *
>>> H, W = 17, 18
>>> cfg = mmcv.Config({'mask_size': (13, 14)})
>>> rng = np.random.RandomState(0)
>>> # Positive proposals (tl_x, tl_y, br_x, br_y) for each image
>>> pos_proposals_list = [
>>>     torch.Tensor([
>>>         [ 7.2425,  5.5929, 13.9414, 14.9541],
>>>         [ 7.3241,  3.6170, 16.3850, 15.3102],
>>>     ]),
>>>     torch.Tensor([
>>>         [ 4.8448, 6.4010, 7.0314, 9.7681],
>>>         [ 5.9790, 2.6989, 7.4416, 4.8580],
>>>         [ 0.0000, 0.0000, 0.1398, 9.8232],
>>>     ]),
>>> ]
>>> # Corresponding class index for each proposal for each image
>>> pos_assigned_gt_inds_list = [
>>>     torch.LongTensor([7, 0]),
>>>     torch.LongTensor([5, 4, 1]),
>>> ]
>>> # Ground truth mask for each true object for each image
>>> gt_masks_list = [
>>>     BitmapMasks(rng.rand(8, H, W), height=H, width=W),
>>>     BitmapMasks(rng.rand(6, H, W), height=H, width=W),
>>> ]
>>> mask_targets = mask_target(
>>>     pos_proposals_list, pos_assigned_gt_inds_list,
>>>     gt_masks_list, cfg)
>>> assert mask_targets.shape == (5,) + cfg['mask_size']
```

mmdet.core.mask.**split_combined_polys**(*polys*, *poly_lens*, *polys_per_mask*)
Split the combined 1-D polys into masks.

A mask is represented as a list of polys, and a poly is represented as a 1-D array. In dataset, all masks are concatenated into a single 1-D tensor. Here we need to split the tensor into original representations.

Parameters

- **polys** (*list*) – a list (length = image num) of 1-D tensors
- **poly_lens** (*list*) – a list (length = image num) of poly length
- **polys_per_mask** (*list*) – a list (length = image num) of poly number of each mask

Returns a list (length = image num) of list (length = mask num) of list (length = poly num) of numpy array.

Return type list

38.5 evaluation

class mmdet.core.evaluation.**DistEvalHook**(**args*, *dynamic_intervals=None*, ***kwargs*)

before_train_epoch(*runner*)
Evaluate the model only at the start of training by epoch.

before_train_iter(*runner*)
Evaluate the model only at the start of training by iteration.

class mmdet.core.evaluation.**EvalHook**(**args*, *dynamic_intervals=None*, ***kwargs*)

before_train_epoch(*runner*)
Evaluate the model only at the start of training by epoch.

before_train_iter(*runner*)
Evaluate the model only at the start of training by iteration.

mmdet.core.evaluation.**average_precision**(*recalls*, *precisions*, *mode='area'*)
Calculate average precision (for single or multiple scales).

Parameters

- **recalls** (*ndarray*) – shape (num_scales, num_dets) or (num_dets,)
- **precisions** (*ndarray*) – shape (num_scales, num_dets) or (num_dets,)
- **mode** (*str*) – 'area' or '11points', 'area' means calculating the area under precision-recall curve, '11points' means calculating the average precision of recalls at [0, 0.1, . . . , 1]

Returns calculated average precision

Return type float or ndarray

mmdet.core.evaluation.**eval_map**(*det_results*, *annotations*, *scale_ranges=None*, *iou_thr=0.5*, *dataset=None*, *logger=None*, *tpfp_fn=None*, *nproc=4*, *use_legacy_coordinate=False*)
Evaluate mAP of a dataset.

Parameters

- **det_results** (*list[list]*) – [[cls1_det, cls2_det, . . .], . . .]. The outer list indicates images, and the inner list indicates per-class detected bboxes.

- **annotations** (*list[dict]*) – Ground truth annotations where each item of the list indicates an image. Keys of annotations are:
  - *bboxes*: numpy array of shape (n, 4)
  - *labels*: numpy array of shape (n,)
  - *bboxes_ignore* (optional): numpy array of shape (k, 4)
  - *labels_ignore* (optional): numpy array of shape (k,)
- **scale_ranges** (*list[tuple] | None*) – Range of scales to be evaluated, in the format [(min1, max1), (min2, max2), ...]. A range of (32, 64) means the area range between (32**2, 64**2). Default: None.
- **iou_thr** (*float*) – IoU threshold to be considered as matched. Default: 0.5.
- **dataset** (*list[str] | str | None*) – Dataset name or dataset classes, there are minor differences in metrics for different datasets, e.g. "voc07", "imagenet_det", etc. Default: None.
- **logger** (*logging.Logger | str | None*) – The way to print the mAP summary. See *mmcv.utils.print_log()* for details. Default: None.
- **tpfp_fn** (*callable | None*) – The function used to determine true/ false positives. If None, `tpfp_default()` is used as default unless dataset is 'det' or 'vid' (`tpfp_imagenet()` in this case). If it is given as a function, then this function is used to evaluate tp & fp. Default None.
- **nproc** (*int*) – Processes used for computing TP and FP. Default: 4.
- **use_legacy_coordinate** (*bool*) – Whether to use coordinate system in mmdet v1.x. which means width, height should be calculated as 'x2 - x1 + 1` and 'y2 - y1 + 1' respectively. Default: False.

Returns (mAP, [dict, dict, ...])

Return type tuple

mmdet.core.evaluation.**eval_recalls**(*gts*, *proposals*, *proposal_nums=None*, *iou_thrs=0.5*, *logger=None*, *use_legacy_coordinate=False*)

Calculate recalls.

Parameters

- **gts** (*list[ndarray]*) – a list of arrays of shape (n, 4)
- **proposals** (*list[ndarray]*) – a list of arrays of shape (k, 4) or (k, 5)
- **proposal_nums** (*int | Sequence[int]*) – Top N proposals to be evaluated.
- **iou_thrs** (*float | Sequence[float]*) – IoU thresholds. Default: 0.5.
- **logger** (*logging.Logger | str | None*) – The way to print the recall summary. See *mmcv.utils.print_log()* for details. Default: None.
- **use_legacy_coordinate** (*bool*) – Whether use coordinate system in mmdet v1.x. "1" was added to both height and width which means w, h should be computed as 'x2 - x1 + 1` and 'y2 - y1 + 1'. Default: False.

Returns recalls of different ious and proposal nums

Return type ndarray

mmdet.core.evaluation.**get_classes**(*dataset*)

Get class names of a dataset.

mmdet.core.evaluation.**plot_iou_recall**(*recalls*, *iou_thrs*)
Plot IoU-Recalls curve.

Parameters

- **recalls** (*ndarray or list*) – shape (k,)
- **iou_thrs** (*ndarray or list*) – same shape as *recalls*

mmdet.core.evaluation.**plot_num_recall**(*recalls*, *proposal_nums*)
Plot Proposal_num-Recalls curve.

Parameters

- **recalls** (*ndarray or list*) – shape (k,)
- **proposal_nums** (*ndarray or list*) – same shape as *recalls*

mmdet.core.evaluation.**print_map_summary**(*mean_ap*, *results*, *dataset=None*, *scale_ranges=None*, *logger=None*)
Print mAP and results of each class.

A table will be printed to show the gts/dets/recall/AP of each class and the mAP.

Parameters

- **mean_ap** (*float*) – Calculated from *eval_map()*.
- **results** (*list[dict]*) – Calculated from *eval_map()*.
- **dataset** (*list[str] | str | None*) – Dataset name or dataset classes.
- **scale_ranges** (*list[tuple] | None*) – Range of scales to be evaluated.
- **logger** (*logging.Logger | str | None*) – The way to print the mAP summary. See *mmcv.utils.print_log()* for details. Default: None.

mmdet.core.evaluation.**print_recall_summary**(*recalls*, *proposal_nums*, *iou_thrs*, *row_idxs=None*, *col_idxs=None*, *logger=None*)
Print recalls in a table.

Parameters

- **recalls** (*ndarray*) – calculated from *bbox_recalls*
- **proposal_nums** (*ndarray or list*) – top N proposals
- **iou_thrs** (*ndarray or list*) – iou thresholds
- **row_idxs** (*ndarray*) – which rows(proposal nums) to print
- **col_idxs** (*ndarray*) – which cols(iou thresholds) to print
- **logger** (*logging.Logger | str | None*) – The way to print the recall summary. See *mmcv.utils.print_log()* for details. Default: None.

38.6 post_processing

mmdet.core.post_processing.**fast_nms**(*multi_bboxes*, *multi_scores*, *multi_coeffs*, *score_thr*, *iou_thr*, *top_k*, *max_num=- 1*)

Fast NMS in YOLACT.

Fast NMS allows already-removed detections to suppress other detections so that every instance can be decided to be kept or discarded in parallel, which is not possible in traditional NMS. This relaxation allows us to implement Fast NMS entirely in standard GPU-accelerated matrix operations.

Parameters

- **multi_bboxes** (*Tensor*) – shape (n, #class*4) or (n, 4)
- **multi_scores** (*Tensor*) – shape (n, #class+1), where the last column contains scores of the background class, but this will be ignored.
- **multi_coeffs** (*Tensor*) – shape (n, #class*coeffs_dim).
- **score_thr** (*float*) – bbox threshold, bboxes with scores lower than it will not be considered.
- **iou_thr** (*float*) – IoU threshold to be considered as conflicted.
- **top_k** (*int*) – if there are more than top_k bboxes before NMS, only top top_k will be kept.
- **max_num** (*int*) – if there are more than max_num bboxes after NMS, only top max_num will be kept. If -1, keep all the bboxes. Default: -1.

Returns

(dets, labels, coefficients), tensors of shape (k, 5), (k, 1), and (k, coeffs_dim). Dets are boxes with scores. Labels are 0-based.

Return type tuple

mmdet.core.post_processing.**mask_matrix_nms**(*masks*, *labels*, *scores*, *filter_thr=- 1*, *nms_pre=- 1*, *max_num=- 1*, *kernel='gaussian'*, *sigma=2.0*, *mask_area=None*)

Matrix NMS for multi-class masks.

Parameters

- **masks** (*Tensor*) – Has shape (num_instances, h, w)
- **labels** (*Tensor*) – Labels of corresponding masks, has shape (num_instances,).
- **scores** (*Tensor*) – Mask scores of corresponding masks, has shape (num_instances).
- **filter_thr** (*float*) – Score threshold to filter the masks after matrix nms. Default: -1, which means do not use filter_thr.
- **nms_pre** (*int*) – The max number of instances to do the matrix nms. Default: -1, which means do not use nms_pre.
- **max_num** (*int, optional*) – If there are more than max_num masks after matrix, only top max_num will be kept. Default: -1, which means do not use max_num.
- **kernel** (*str*) – 'linear' or 'gaussian'.
- **sigma** (*float*) – std in gaussian method.
- **mask_area** (*Tensor*) – The sum of seg_masks.

Returns

Processed mask results.

- scores (Tensor): Updated scores, has shape (n,).
- labels (Tensor): Remained labels, has shape (n,).
- masks (Tensor): Remained masks, has shape (n, w, h).
- **keep_inds (Tensor): The indices number of** the remaining mask in the input mask, has shape (n,).

Return type tuple(Tensor)

mmdet.core.post_processing.**merge_aug_bboxes**(*aug_bboxes*, *aug_scores*, *img_metas*, *rcnn_test_cfg*)
Merge augmented detection bboxes and scores.

Parameters

- **aug_bboxes** (*list[Tensor]*) – shape (n, 4*#class)
- **aug_scores** (*list[Tensor] or None*) – shape (n, #class)
- **img_shapes** (*list[Tensor]*) – shape (3,).
- **rcnn_test_cfg** (*dict*) – rcnn test config.

Returns (bboxes, scores)

Return type tuple

mmdet.core.post_processing.**merge_aug_masks**(*aug_masks*, *img_metas*, *rcnn_test_cfg*, *weights=None*)
Merge augmented mask prediction.

Parameters

- **aug_masks** (*list[ndarray]*) – shape (n, #class, h, w)
- **img_shapes** (*list[ndarray]*) – shape (3,).
- **rcnn_test_cfg** (*dict*) – rcnn test config.

Returns (bboxes, scores)

Return type tuple

mmdet.core.post_processing.**merge_aug_proposals**(*aug_proposals*, *img_metas*, *cfg*)
Merge augmented proposals (multiscale, flip, etc.)

Parameters

- **aug_proposals** (*list[Tensor]*) – proposals from different testing schemes, shape (n, 5). Note that they are not rescaled to the original image size.
- **img_metas** (*list[dict]*) – list of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmdet/datasets/pipelines/formatting.py:Collect*.
- **cfg** (*dict*) – rpn test config.

Returns shape (n, 4), proposals corresponding to original image scale.

Return type Tensor

mmdet.core.post_processing.**merge_aug_scores**(*aug_scores*)
Merge augmented bbox scores.

mmdet.core.post_processing.**multiclass_nms**(*multi_bboxes*, *multi_scores*, *score_thr*, *nms_cfg*, *max_num=-1*, *score_factors=None*, *return_inds=False*)

NMS for multi-class bboxes.

Parameters

- **multi_bboxes** (*Tensor*) – shape (n, #class*4) or (n, 4)
- **multi_scores** (*Tensor*) – shape (n, #class), where the last column contains scores of the background class, but this will be ignored.
- **score_thr** (*float*) – bbox threshold, bboxes with scores lower than it will not be considered.
- **nms_thr** (*float*) – NMS IoU threshold
- **max_num** (*int, optional*) – if there are more than max_num bboxes after NMS, only top max_num will be kept. Default to -1.
- **score_factors** (*Tensor, optional*) – The factors multiplied to scores before applying NMS. Default to None.
- **return_inds** (*bool, optional*) – Whether return the indices of kept bboxes. Default to False.

Returns

(dets, labels, indices (optional)), tensors of shape (k, 5), (k), and (k). Dets are boxes with scores. Labels are 0-based.

Return type tuple

38.7 utils

class mmdet.core.utils.**DistOptimizerHook**(**args*, ***kwargs*)

Deprecated optimizer hook for distributed training.

mmdet.core.utils.**all_reduce_dict**(*py_dict*, *op='sum'*, *group=None*, *to_float=True*)

Apply all reduce function for python dict object.

The code is modified from https://github.com/Megvii- BaseDetection/YOLOX/blob/main/yolox/utils/allreduce_norm.py.

NOTE: make sure that py_dict in different ranks has the same keys and the values should be in the same shape. Currently only supports nccl backend.

Parameters

- **py_dict** (*dict*) – Dict to be applied all reduce op.
- **op** (*str*) – Operator, could be 'sum' or 'mean'. Default: 'sum'
- **group** (torch.distributed.group, optional) – Distributed group, Default: None.
- **to_float** (*bool*) – Whether to convert all values of dict to float. Default: True.

Returns reduced python dict object.

Return type OrderedDict

mmdet.core.utils.**allreduce_grads**(*params*, *coalesce=True*, *bucket_size_mb=-1*)

Allreduce gradients.

Parameters

- **params** (*list[torch.Parameters]*) – List of parameters of a model
- **coalesce** (*bool, optional*) – Whether allreduce parameters as a whole. Defaults to True.
- **bucket_size_mb** (*int, optional*) – Size of bucket, the unit is MB. Defaults to -1.

mmdet.core.utils.**center_of_mass**(*mask*, *esp=1e-06*)
Calculate the centroid coordinates of the mask.

Parameters

- **mask** (*Tensor*) – The mask to be calculated, shape (h, w).
- **esp** (*float*) – Avoid dividing by zero. Default: 1e-6.

Returns

the coordinates of the center point of the mask.

- center_h (Tensor): the center point of the height.
- center_w (Tensor): the center point of the width.

Return type tuple[Tensor]

mmdet.core.utils.**filter_scores_and_topk**(*scores*, *score_thr*, *topk*, *results=None*)
Filter results using score threshold and topk candidates.

Parameters

- **scores** (*Tensor*) – The scores, shape (num_bboxes, K).
- **score_thr** (*float*) – The score filter threshold.
- **topk** (*int*) – The number of topk candidates.
- **results** (*dict or list or Tensor, Optional*) – The results to which the filtering rule is to be applied. The shape of each item is (num_bboxes, N).

Returns

Filtered results

- scores (Tensor): The scores after being filtered, shape (num_bboxes_filtered,).
- labels (Tensor): The class labels, shape (num_bboxes_filtered,).
- anchor_idxs (Tensor): The anchor indexes, shape (num_bboxes_filtered,).
- filtered_results (dict or list or Tensor, Optional): The filtered results. The shape of each item is (num_bboxes_filtered, N).

Return type tuple

mmdet.core.utils.**flip_tensor**(*src_tensor*, *flip_direction*)
flip tensor base on flip_direction.

Parameters

- **src_tensor** (*Tensor*) – input feature map, shape (B, C, H, W).
- **flip_direction** (*str*) – The flipping direction. Options are 'horizontal', 'vertical', 'diagonal'.

Returns Flipped tensor.

Return type out_tensor (Tensor)

mmdet.core.utils.**generate_coordinate**(*featmap_sizes*, *device='cuda'*)
Generate the coordinate.

Parameters

- **featmap_sizes** (*tuple*) – The feature to be calculated, of shape (N, C, W, H).
- **device** (*str*) – The device where the feature will be put on.

Returns The coordinate feature, of shape (N, 2, W, H).

Return type coord_feat (Tensor)

mmdet.core.utils.**mask2ndarray**(*mask*)
Convert Mask to ndarray..

:param mask (BitmapMasks or PolygonMasks or: :param torch.Tensor or np.ndarray): The mask to be converted.

Returns Ndarray mask of shape (n, h, w) that has been converted

Return type np.ndarray

mmdet.core.utils.**multi_apply**(*func*, **args*, ***kwargs*)
Apply function to a list of arguments.

Note: This function applies the func to multiple inputs and map the multiple outputs of the func into different list. Each list contains the same type of outputs corresponding to different inputs.

Parameters **func** (*Function*) – A function that will be applied to a list of arguments

Returns A tuple containing multiple list, each list contains a kind of returned results by the function

Return type tuple(list)

mmdet.core.utils.**reduce_mean**(*tensor*)
"Obtain the mean of tensor on different GPUs.

mmdet.core.utils.**select_single_mlvl**(*mlvl_tensors*, *batch_id*, *detach=True*)
Extract a multi-scale single image tensor from a multi-scale batch tensor based on batch index.

Note: The default value of detach is True, because the proposal gradient needs to be detached during the training of the two-stage model. E.g Cascade Mask R-CNN.

Parameters

- **mlvl_tensors** (*list[Tensor]*) – Batch tensor for all scale levels, each is a 4D-tensor.
- **batch_id** (*int*) – Batch index.
- **detach** (*bool*) – Whether detach gradient. Default True.

Returns Multi-scale single image tensor.

Return type list[Tensor]

mmdet.core.utils.**unmap**(*data*, *count*, *inds*, *fill=0*)
Unmap a subset of item (data) back to the original set of items (of size count)

CHAPTER

THIRTYNINE

MMDET.DATASETS

39.1 datasets

class mmdet.datasets.**CityscapesDataset**(*ann_file*, *pipeline*, *classes=None*, *data_root=None*, *img_prefix=''*, *seg_prefix=None*, *proposal_file=None*, *test_mode=False*, *filter_empty_gt=True*, *file_client_args={'backend': 'disk'}*)

evaluate(*results*, *metric='bbox'*, *logger=None*, *outfile_prefix=None*, *classwise=False*, *proposal_nums=(100, 300, 1000)*, *iou_thrs=array([0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95])*)

Evaluation in Cityscapes/COCO protocol.

Parameters

- **results** (*list[list | tuple]*) – Testing results of the dataset.
- **metric** (*str | list[str]*) – Metrics to be evaluated. Options are 'bbox', 'segm', 'proposal', 'proposal_fast'.
- **logger** (*logging.Logger | str | None*) – Logger used for printing related information during evaluation. Default: None.
- **outfile_prefix** (*str | None*) – The prefix of output file. It includes the file path and the prefix of filename, e.g., "a/b/prefix". If results are evaluated with COCO protocol, it would be the prefix of output json file. For example, the metric is 'bbox' and 'segm', then json files would be "a/b/prefix.bbox.json" and "a/b/prefix.segm.json". If results are evaluated with cityscapes protocol, it would be the prefix of output txt/png files. The output files would be png images under folder "a/b/prefix/xxx/" and the file name of images would be written into a txt file "a/b/prefix/xxx_pred.txt", where "xxx" is the video name of cityscapes. If not specified, a temp file will be created. Default: None.
- **classwise** (*bool*) – Whether to evaluating the AP for each class.
- **proposal_nums** (*Sequence[int]*) – Proposal number used for evaluating recalls, such as recall@100, recall@1000. Default: (100, 300, 1000).
- **iou_thrs** (*Sequence[float]*) – IoU threshold used for evaluating recalls. If set to a list, the average recall of all IoUs will also be computed. Default: 0.5.

Returns COCO style evaluation metric or cityscapes mAP and AP@50.

Return type dict[str, float]

format_results(*results*, *txtfile_prefix=None*)

Format the results to txt (standard format for Cityscapes evaluation).

Parameters

- **results** (*list*) – Testing results of the dataset.
- **txtfile_prefix** (*str | None*) – The prefix of txt files. It includes the file path and the prefix of filename, e.g., "a/b/prefix". If not specified, a temp file will be created. Default: None.

Returns (result_files, tmp_dir), result_files is a dict containing the json filepaths, tmp_dir is the temporal directory created for saving txt/png files when txtfile_prefix is not specified.

Return type tuple

results2txt(*results*, *outfile_prefix*)
Dump the detection results to a txt file.

Parameters

- **results** (*list[list | tuple]*) – Testing results of the dataset.
- **outfile_prefix** (*str*) – The filename prefix of the json files. If the prefix is "somepath/xxx", the txt files will be named "somepath/xxx.txt".

Returns Result txt files which contains corresponding instance segmentation images.

Return type list[str]

class mmdet.datasets.**ClassBalancedDataset**(*dataset*, *oversample_thr*, *filter_empty_gt=True*)
A wrapper of repeated dataset with repeat factor.

Suitable for training on class imbalanced datasets like LVIS. Following the sampling strategy in the paper, in each epoch, an image may appear multiple times based on its "repeat factor". The repeat factor for an image is a function of the frequency the rarest category labeled in that image. The "frequency of category c" in [0, 1] is defined by the fraction of images in the training set (without repeats) in which category c appears. The dataset needs to instantiate `self.get_cat_ids()` to support ClassBalancedDataset.

The repeat factor is computed as followed.

1. For each category c, compute the fraction # of images that contain it: $f(c)$
2. For each category c, compute the category-level repeat factor: $r(c) = max(1, sqrt(t/f(c)))$
3. For each image I, compute the image-level repeat factor: $r(I) = max_{cinI} r(c)$

Parameters

- **dataset** (*CustomDataset*) – The dataset to be repeated.
- **oversample_thr** (*float*) – frequency threshold below which data is repeated. For categories with `f_c >= oversample_thr`, there is no oversampling. For categories with `f_c < oversample_thr`, the degree of oversampling following the square-root inverse frequency heuristic above.
- **filter_empty_gt** (*bool, optional*) – If set true, images without bounding boxes will not be oversampled. Otherwise, they will be categorized as the pure background class and involved into the oversampling. Default: True.

class mmdet.datasets.**CocoDataset**(*ann_file*, *pipeline*, *classes=None*, *data_root=None*, *img_prefix=''*, *seg_prefix=None*, *proposal_file=None*, *test_mode=False*, *filter_empty_gt=True*, *file_client_args={'backend': 'disk'}*)

evaluate(*results*, *metric='bbox'*, *logger=None*, *jsonfile_prefix=None*, *classwise=False*, *proposal_nums=(100, 300, 1000)*, *iou_thrs=None*, *metric_items=None*)
Evaluation in COCO protocol.

Parameters

- **results** (*list[list | tuple]*) – Testing results of the dataset.
- **metric** (*str | list[str]*) – Metrics to be evaluated. Options are 'bbox', 'segm', 'proposal', 'proposal_fast'.
- **logger** (*logging.Logger | str | None*) – Logger used for printing related information during evaluation. Default: None.
- **jsonfile_prefix** (*str | None*) – The prefix of json files. It includes the file path and the prefix of filename, e.g., "a/b/prefix". If not specified, a temp file will be created. Default: None.
- **classwise** (*bool*) – Whether to evaluating the AP for each class.
- **proposal_nums** (*Sequence[int]*) – Proposal number used for evaluating recalls, such as recall@100, recall@1000. Default: (100, 300, 1000).
- **iou_thrs** (*Sequence[float], optional*) – IoU threshold used for evaluating recalls/mAPs. If set to a list, the average of all IoUs will also be computed. If not specified, [0.50, 0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95] will be used. Default: None.
- **metric_items** (*list[str] | str, optional*) – Metric items that will be returned. If not specified, `['AR@100', 'AR@300', 'AR@1000', 'AR_s@1000', 'AR_m@1000', 'AR_l@1000' ]` will be used when `metric=='proposal'`, `['mAP', 'mAP_50', 'mAP_75', 'mAP_s', 'mAP_m', 'mAP_l']` will be used when `metric=='bbox' or metric=='segm'`.

Returns COCO style evaluation metric.

Return type dict[str, float]

format_results(*results*, *jsonfile_prefix=None*, ***kwargs*)

Format the results to json (standard format for COCO evaluation).

Parameters

- **results** (*list[tuple | numpy.ndarray]*) – Testing results of the dataset.
- **jsonfile_prefix** (*str | None*) – The prefix of json files. It includes the file path and the prefix of filename, e.g., "a/b/prefix". If not specified, a temp file will be created. Default: None.

Returns (result_files, tmp_dir), result_files is a dict containing the json filepaths, tmp_dir is the temporal directory created for saving json files when jsonfile_prefix is not specified.

Return type tuple

get_ann_info(*idx*)

Get COCO annotation by index.

Parameters **idx** (*int*) – Index of data.

Returns Annotation info of specified index.

Return type dict

get_cat_ids(*idx*)

Get COCO category ids by index.

Parameters **idx** (*int*) – Index of data.

Returns All categories in the image of specified index.

Return type list[int]

load_annotations(*ann_file*)
Load annotation from COCO style annotation file.

Parameters **ann_file** (*str*) – Path of annotation file.

Returns Annotation info from COCO api.

Return type list[dict]

results2json(*results*, *outfile_prefix*)
Dump the detection results to a COCO style json file.

There are 3 types of results: proposals, bbox predictions, mask predictions, and they have different data types. This method will automatically recognize the type, and dump them to json files.

Parameters

- **results** (*list[list | tuple | ndarray]*) – Testing results of the dataset.
- **outfile_prefix** (*str*) – The filename prefix of the json files. If the prefix is "somepath/xxx", the json files will be named "somepath/xxx.bbox.json", "somepath/xxx.segm.json", "somepath/xxx.proposal.json".

Returns str]: Possible keys are "bbox", "segm", "proposal", and values are corresponding filenames.

Return type dict[str

xyxy2xywh(*bbox*)
Convert xyxy style bounding boxes to xywh style for COCO evaluation.

Parameters **bbox** (*numpy.ndarray*) – The bounding boxes, shape (4,), in xyxy order.

Returns The converted bounding boxes, in xywh order.

Return type list[float]

class mmdet.datasets.**CocoPanopticDataset**(*ann_file*, *pipeline*, *classes=None*, *data_root=None*, *img_prefix=''*, *seg_prefix=None*, *proposal_file=None*, *test_mode=False*, *filter_empty_gt=True*, *file_client_args={'backend': 'disk'}*)
Coco dataset for Panoptic segmentation.

The annotation format is shown as follows. The *ann* field is optional for testing.

```
[
    {
        'filename': f'{image_id:012}.png',
        'image_id':9
        'segments_info': {
            [
                {
                    'id': 8345037, (segment_id in panoptic png,
                                    convert from rgb)
                    'category_id': 51,
                    'iscrowd': 0,
                    'bbox': (x1, y1, w, h),
                    'area': 24315,
                    'segmentation': list,(coded mask)
                },
                ...
```

(continues on next page)

(continued from previous page)

```
            }
        }
    },
    ...
]
```

evaluate(*results*, *metric='PQ'*, *logger=None*, *jsonfile_prefix=None*, *classwise=False*, ***kwargs*)
Evaluation in COCO Panoptic protocol.

Parameters

- **results** (*list[dict]*) – Testing results of the dataset.
- **metric** (*str | list[str]*) – Metrics to be evaluated. Only support 'PQ' at present. 'pq' will be regarded as 'PQ.
- **logger** (*logging.Logger | str | None*) – Logger used for printing related information during evaluation. Default: None.
- **jsonfile_prefix** (*str | None*) – The prefix of json files. It includes the file path and the prefix of filename, e.g., "a/b/prefix". If not specified, a temp file will be created. Default: None.
- **classwise** (*bool*) – Whether to print classwise evaluation results. Default: False.

Returns COCO Panoptic style evaluation metric.

Return type dict[str, float]

evaluate_pan_json(*result_files*, *outfile_prefix*, *logger=None*, *classwise=False*)
Evaluate PQ according to the panoptic results json file.

get_ann_info(*idx*)
Get COCO annotation by index.

Parameters **idx** (*int*) – Index of data.

Returns Annotation info of specified index.

Return type dict

load_annotations(*ann_file*)
Load annotation from COCO Panoptic style annotation file.

Parameters **ann_file** (*str*) – Path of annotation file.

Returns Annotation info from COCO api.

Return type list[dict]

results2json(*results*, *outfile_prefix*)
Dump the panoptic results to a COCO panoptic style json file.

Parameters

- **results** (*dict*) – Testing results of the dataset.
- **outfile_prefix** (*str*) – The filename prefix of the json files. If the prefix is "somepath/xxx", the json files will be named "somepath/xxx.panoptic.json"

Returns

str]: The key is 'panoptic' and the value is corresponding filename.

Return type dict[str

class mmdet.datasets.**ConcatDataset**(*datasets*, *separate_eval=True*)

A wrapper of concatenated dataset.

Same as `torch.utils.data.dataset.ConcatDataset`, but concat the group flag for image aspect ratio.

Parameters

- **datasets** (list[Dataset]) – A list of datasets.
- **separate_eval** (*bool*) – Whether to evaluate the results separately if it is used as validation dataset. Defaults to True.

evaluate(*results*, *logger=None*, ***kwargs*)

Evaluate the results.

Parameters

- **results** (*list[list | tuple]*) – Testing results of the dataset.
- **logger** (*logging.Logger | str | None*) – Logger used for printing related information during evaluation. Default: None.

Returns float]: AP results of the total dataset or each separate dataset if *self.separate_eval=True*.

Return type dict[str

get_cat_ids(*idx*)

Get category ids of concatenated dataset by index.

Parameters **idx** (*int*) – Index of data.

Returns All categories in the image of specified index.

Return type list[int]

class mmdet.datasets.**CustomDataset**(*ann_file*, *pipeline*, *classes=None*, *data_root=None*, *img_prefix=''*, *seg_prefix=None*, *proposal_file=None*, *test_mode=False*, *filter_empty_gt=True*, *file_client_args={'backend': 'disk'}*)

Custom dataset for detection.

The annotation format is shown as follows. The *ann* field is optional for testing.

```
[
    {
        'filename': 'a.jpg',
        'width': 1280,
        'height': 720,
        'ann': {
            'bboxes': <np.ndarray> (n, 4) in (x1, y1, x2, y2) order.
            'labels': <np.ndarray> (n, ),
            'bboxes_ignore': <np.ndarray> (k, 4), (optional field)
            'labels_ignore': <np.ndarray> (k, 4) (optional field)
        }
    },
    ...
]
```

Parameters

- **ann_file** (*str*) – Annotation file path.
- **pipeline** (*list[dict]*) – Processing pipeline.

- **classes** (*str | Sequence[str], optional*) – Specify classes to load. If is None, `cls.CLASSES` will be used. Default: None.
- **data_root** (*str, optional*) – Data root for `ann_file`, `img_prefix`, `seg_prefix`, `proposal_file` if specified.
- **test_mode** (*bool, optional*) – If set True, annotation will not be loaded.
- **filter_empty_gt** (*bool, optional*) – If set true, images without bounding boxes of the dataset's classes will be filtered out. This option only works when *test_mode=False*, i.e., we never filter images during tests.

evaluate(*results*, *metric='mAP'*, *logger=None*, *proposal_nums=(100, 300, 1000)*, *iou_thr=0.5*, *scale_ranges=None*)

Evaluate the dataset.

Parameters

- **results** (*list*) – Testing results of the dataset.
- **metric** (*str | list[str]*) – Metrics to be evaluated.
- **logger** (*logging.Logger | None | str*) – Logger used for printing related information during evaluation. Default: None.
- **proposal_nums** (*Sequence[int]*) – Proposal number used for evaluating recalls, such as recall@100, recall@1000. Default: (100, 300, 1000).
- **iou_thr** (*float | list[float]*) – IoU threshold. Default: 0.5.
- **scale_ranges** (*list[tuple] | None*) – Scale ranges for evaluating mAP. Default: None.

format_results(*results*, ***kwargs*)

Place holder to format result to dataset specific output.

get_ann_info(*idx*)

Get annotation by index.

Parameters **idx** (*int*) – Index of data.

Returns Annotation info of specified index.

Return type dict

get_cat_ids(*idx*)

Get category ids by index.

Parameters **idx** (*int*) – Index of data.

Returns All categories in the image of specified index.

Return type list[int]

classmethod get_classes(*classes=None*)

Get class names of current dataset.

Parameters **classes** (*Sequence[str] | str | None*) – If classes is None, use default CLASSES defined by builtin dataset. If classes is a string, take it as a file name. The file contains the name of classes where each line contains one class name. If classes is a tuple or list, override the CLASSES defined by the dataset.

Returns Names of categories of the dataset.

Return type tuple[str] or list[str]

load_annotations(*ann_file*)
Load annotation from annotation file.

load_proposals(*proposal_file*)
Load proposal from proposal file.

pre_pipeline(*results*)
Prepare results dict for pipeline.

prepare_test_img(*idx*)
Get testing data after pipeline.

Parameters **idx** (*int*) – Index of data.

Returns Testing data after pipeline with new keys introduced by pipeline.

Return type dict

prepare_train_img(*idx*)
Get training data and annotations after pipeline.

Parameters **idx** (*int*) – Index of data.

Returns Training data and annotation after pipeline with new keys introduced by pipeline.

Return type dict

class mmdet.datasets.**DeepFashionDataset**(*ann_file*, *pipeline*, *classes=None*, *data_root=None*, *img_prefix=''*, *seg_prefix=None*, *proposal_file=None*, *test_mode=False*, *filter_empty_gt=True*, *file_client_args={'backend': 'disk'}*)

class mmdet.datasets.**DistributedGroupSampler**(*dataset*, *samples_per_gpu=1*, *num_replicas=None*, *rank=None*, *seed=0*)
Sampler that restricts data loading to a subset of the dataset.

It is especially useful in conjunction with `torch.nn.parallel.DistributedDataParallel`. In such case, each process can pass a DistributedSampler instance as a DataLoader sampler, and load a subset of the original dataset that is exclusive to it.

Note: Dataset is assumed to be of constant size.

Parameters

- **dataset** – Dataset used for sampling.
- **num_replicas** (*optional*) – Number of processes participating in distributed training.
- **rank** (*optional*) – Rank of the current process within num_replicas.
- **seed** (*int, optional*) – random seed used to shuffle the sampler if `shuffle=True`. This number should be identical across all processes in the distributed group. Default: 0.

class mmdet.datasets.**DistributedSampler**(*dataset*, *num_replicas=None*, *rank=None*, *shuffle=True*, *seed=0*)

class mmdet.datasets.**GroupSampler**(*dataset*, *samples_per_gpu=1*)

mmdet.datasets.**LVISDataset**
alias of *mmdet.datasets.lvis.LVISV05Dataset*

class mmdet.datasets.**LVISV05Dataset**(*ann_file*, *pipeline*, *classes=None*, *data_root=None*, *img_prefix=''*, *seg_prefix=None*, *proposal_file=None*, *test_mode=False*, *filter_empty_gt=True*, *file_client_args={'backend': 'disk'}*)

evaluate(*results*, *metric='bbox'*, *logger=None*, *jsonfile_prefix=None*, *classwise=False*, *proposal_nums=(100, 300, 1000)*, *iou_thrs=array([0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95])*)

Evaluation in LVIS protocol.

Parameters

- **results** (*list[list | tuple]*) – Testing results of the dataset.
- **metric** (*str | list[str]*) – Metrics to be evaluated. Options are 'bbox', 'segm', 'proposal', 'proposal_fast'.
- **logger** (*logging.Logger | str | None*) – Logger used for printing related information during evaluation. Default: None.
- **jsonfile_prefix** (*str | None*) –
- **classwise** (*bool*) – Whether to evaluating the AP for each class.
- **proposal_nums** (*Sequence[int]*) – Proposal number used for evaluating recalls, such as recall@100, recall@1000. Default: (100, 300, 1000).
- **iou_thrs** (*Sequence[float]*) – IoU threshold used for evaluating recalls. If set to a list, the average recall of all IoUs will also be computed. Default: 0.5.

Returns LVIS style metrics.

Return type dict[str, float]

load_annotations(*ann_file*)

Load annotation from lvis style annotation file.

Parameters **ann_file** (*str*) – Path of annotation file.

Returns Annotation info from LVIS api.

Return type list[dict]

class mmdet.datasets.**LVISV1Dataset**(*ann_file*, *pipeline*, *classes=None*, *data_root=None*, *img_prefix=''*, *seg_prefix=None*, *proposal_file=None*, *test_mode=False*, *filter_empty_gt=True*, *file_client_args={'backend': 'disk'}*)

load_annotations(*ann_file*)

Load annotation from lvis style annotation file.

Parameters **ann_file** (*str*) – Path of annotation file.

Returns Annotation info from LVIS api.

Return type list[dict]

class mmdet.datasets.**MultiImageMixDataset**(*dataset*, *pipeline*, *dynamic_scale=None*, *skip_type_keys=None*)

A wrapper of multiple images mixed dataset.

Suitable for training on multiple images mixed data augmentation like mosaic and mixup. For the augmentation pipeline of mixed image data, the *get_indexes* method needs to be provided to obtain the image indexes, and you can set *skip_flags* to change the pipeline running process. At the same time, we provide the *dynamic_scale* parameter to dynamically change the output image size.

Parameters

- **dataset** (*CustomDataset*) – The dataset to be mixed.
- **pipeline** (*Sequence[dict]*) – Sequence of transform object or config dict to be composed.
- **dynamic_scale** (*tuple[int], optional*) – The image scale can be changed dynamically. Default to None. It is deprecated.
- **skip_type_keys** (*list[str], optional*) – Sequence of type string to be skip pipeline. Default to None.

update_skip_type_keys(*skip_type_keys*)

Update skip_type_keys. It is called by an external hook.

Parameters **skip_type_keys** (*list[str], optional*) – Sequence of type string to be skip pipeline.

class mmdet.datasets.**RepeatDataset**(*dataset*, *times*)

A wrapper of repeated dataset.

The length of repeated dataset will be *times* larger than the original dataset. This is useful when the data loading time is long but the dataset is small. Using RepeatDataset can reduce the data loading time between epochs.

Parameters

- **dataset** (Dataset) – The dataset to be repeated.
- **times** (*int*) – Repeat times.

get_cat_ids(*idx*)

Get category ids of repeat dataset by index.

Parameters **idx** (*int*) – Index of data.

Returns All categories in the image of specified index.

Return type list[int]

class mmdet.datasets.**VOCDataset**(***kwargs*)

evaluate(*results*, *metric='mAP'*, *logger=None*, *proposal_nums=(100, 300, 1000)*, *iou_thr=0.5*, *scale_ranges=None*)

Evaluate in VOC protocol.

Parameters

- **results** (*list[list | tuple]*) – Testing results of the dataset.
- **metric** (*str | list[str]*) – Metrics to be evaluated. Options are 'mAP', 'recall'.
- **logger** (*logging.Logger | str, optional*) – Logger used for printing related information during evaluation. Default: None.
- **proposal_nums** (*Sequence[int]*) – Proposal number used for evaluating recalls, such as recall@100, recall@1000. Default: (100, 300, 1000).
- **iou_thr** (*float | list[float]*) – IoU threshold. Default: 0.5.
- **scale_ranges** (*list[tuple], optional*) – Scale ranges for evaluating mAP. If not specified, all bounding boxes would be included in evaluation. Default: None.

Returns AP/recall metrics.

Return type dict[str, float]

class mmdet.datasets.**WIDERFaceDataset**(***kwargs*)
Reader for the WIDER Face dataset in PASCAL VOC format.

Conversion scripts can be found in https://github.com/sovrasov/wider-face-pascal-voc-annotations

load_annotations(*ann_file*)
Load annotation from WIDERFace XML style annotation file.

Parameters **ann_file** (*str*) – Path of XML file.

Returns Annotation info from XML file.

Return type list[dict]

class mmdet.datasets.**XMLDataset**(*min_size=None*, *img_subdir='JPEGImages'*, *ann_subdir='Annotations'*, ***kwargs*)
XML dataset for detection.

Parameters

- **min_size** (*int | float, optional*) – The minimum size of bounding boxes in the images. If the size of a bounding box is less than min_size, it would be add to ignored field.
- **img_subdir** (*str*) – Subdir where images are stored. Default: JPEGImages.
- **ann_subdir** (*str*) – Subdir where annotations are. Default: Annotations.

get_ann_info(*idx*)
Get annotation from XML file by index.

Parameters **idx** (*int*) – Index of data.

Returns Annotation info of specified index.

Return type dict

get_cat_ids(*idx*)
Get category ids in XML file by index.

Parameters **idx** (*int*) – Index of data.

Returns All categories in the image of specified index.

Return type list[int]

load_annotations(*ann_file*)
Load annotation from XML style ann_file.

Parameters **ann_file** (*str*) – Path of XML file.

Returns Annotation info from XML file.

Return type list[dict]

mmdet.datasets.**build_dataloader**(*dataset*, *samples_per_gpu*, *workers_per_gpu*, *num_gpus=1*, *dist=True*, *shuffle=True*, *seed=None*, *runner_type='EpochBasedRunner'*, *persistent_workers=False*, ***kwargs*)
Build PyTorch DataLoader.

In distributed training, each GPU/process has a dataloader. In non-distributed training, there is only one dataloader for all GPUs.

Parameters

- **dataset** (*Dataset*) – A PyTorch dataset.

- **samples_per_gpu** (*int*) – Number of training samples on each GPU, i.e., batch size of each GPU.
- **workers_per_gpu** (*int*) – How many subprocesses to use for data loading for each GPU.
- **num_gpus** (*int*) – Number of GPUs. Only used in non-distributed training.
- **dist** (*bool*) – Distributed training/test or not. Default: True.
- **shuffle** (*bool*) – Whether to shuffle the data at every epoch. Default: True.
- **seed** (*int, Optional*) – Seed to be used. Default: None.
- **runner_type** (*str*) – Type of runner. Default: *EpochBasedRunner*
- **persistent_workers** (*bool*) – If True, the data loader will not shutdown the worker processes after a dataset has been consumed once. This allows to maintain the workers *Dataset* instances alive. This argument is only valid when PyTorch>=1.7.0. Default: False.
- **kwargs** – any keyword argument to be used to initialize DataLoader

Returns A PyTorch dataloader.

Return type DataLoader

mmdet.datasets.**get_loading_pipeline**(*pipeline*)

Only keep loading image and annotations related configuration.

Parameters **pipeline** (*list[dict]*) – Data pipeline configs.

Returns

The new pipeline list with only keep loading image and annotations related configuration.

Return type list[dict]

Examples

```
>>> pipelines = [
...     dict(type='LoadImageFromFile'),
...     dict(type='LoadAnnotations', with_bbox=True),
...     dict(type='Resize', img_scale=(1333, 800), keep_ratio=True),
...     dict(type='RandomFlip', flip_ratio=0.5),
...     dict(type='Normalize', **img_norm_cfg),
...     dict(type='Pad', size_divisor=32),
...     dict(type='DefaultFormatBundle'),
...     dict(type='Collect', keys=['img', 'gt_bboxes', 'gt_labels'])
...     ]
>>> expected_pipelines = [
...     dict(type='LoadImageFromFile'),
...     dict(type='LoadAnnotations', with_bbox=True)
...     ]
>>> assert expected_pipelines ==         ...         get_loading_pipeline(pipelines)
```

mmdet.datasets.**replace_ImageToTensor**(*pipelines*)

Replace the ImageToTensor transform in a data pipeline to DefaultFormatBundle, which is normally useful in batch inference.

Parameters **pipelines** (*list[dict]*) – Data pipeline configs.

Returns

The new pipeline list with all ImageToTensor replaced by DefaultFormatBundle.

Return type list

Examples

```
>>> pipelines = [
...     dict(type='LoadImageFromFile'),
...     dict(
...         type='MultiScaleFlipAug',
...         img_scale=(1333, 800),
...         flip=False,
...         transforms=[
...             dict(type='Resize', keep_ratio=True),
...             dict(type='RandomFlip'),
...             dict(type='Normalize', mean=[0, 0, 0], std=[1, 1, 1]),
...             dict(type='Pad', size_divisor=32),
...             dict(type='ImageToTensor', keys=['img']),
...             dict(type='Collect', keys=['img']),
...         ])
...     ]
>>> expected_pipelines = [
...     dict(type='LoadImageFromFile'),
...     dict(
...         type='MultiScaleFlipAug',
...         img_scale=(1333, 800),
...         flip=False,
...         transforms=[
...             dict(type='Resize', keep_ratio=True),
...             dict(type='RandomFlip'),
...             dict(type='Normalize', mean=[0, 0, 0], std=[1, 1, 1]),
...             dict(type='Pad', size_divisor=32),
...             dict(type='DefaultFormatBundle'),
...             dict(type='Collect', keys=['img']),
...         ])
...     ]
>>> assert expected_pipelines == replace_ImageToTensor(pipelines)
```

39.2 pipelines

class mmdet.datasets.pipelines.**Albu**(*transforms*, *bbox_params=None*, *keymap=None*, *update_pad_shape=False*, *skip_img_without_anno=False*)

Albumentation augmentation.

Adds custom transformations from Albumentations library. Please, visit *https://albumentations.readthedocs.io* to get more information.

An example of `transforms` is as followed:

```
[
    dict(
        type='ShiftScaleRotate',
```

(continues on next page)

(continued from previous page)

```
        shift_limit=0.0625,
        scale_limit=0.0,
        rotate_limit=0,
        interpolation=1,
        p=0.5),
    dict(
        type='RandomBrightnessContrast',
        brightness_limit=[0.1, 0.3],
        contrast_limit=[0.1, 0.3],
        p=0.2),
    dict(type='ChannelShuffle', p=0.1),
    dict(
        type='OneOf',
        transforms=[
            dict(type='Blur', blur_limit=3, p=1.0),
            dict(type='MedianBlur', blur_limit=3, p=1.0)
        ],
        p=0.1),
]
```

Parameters

- **transforms** (*list[dict]*) – A list of albu transformations
- **bbox_params** (*dict*) – Bbox_params for albumentation *Compose*
- **keymap** (*dict*) – Contains {'input key':'albumentation-style key'}
- **skip_img_without_anno** (*bool*) – Whether to skip the image if no ann left after aug

albu_builder(*cfg*)

Import a module from albumentations.

It inherits some of `build_from_cfg()` logic.

Parameters **cfg** (*dict*) – Config dict. It should at least contain the key "type".

Returns The constructed object.

Return type obj

static mapper(*d*, *keymap*)

Dictionary mapper. Renames keys according to keymap provided.

Parameters

- **d** (*dict*) – old dict
- **keymap** (*dict*) – {'old_key':'new_key'}

Returns new dict.

Return type dict

class mmdet.datasets.pipelines.**AutoAugment**(*policies*)

Auto augmentation.

This data augmentation is proposed in Learning Data Augmentation Strategies for Object Detection.

TODO: Implement 'Shear', 'Sharpness' and 'Rotate' transforms

Parameters **policies** (*list[list[dict]]*) – The policies of auto augmentation. Each policy in `policies` is a specific augmentation policy, and is composed by several augmentations (dict). When AutoAugment is called, a random policy in `policies` will be selected to augment images.

Examples

```
>>> replace = (104, 116, 124)
>>> policies = [
>>>     [
>>>         dict(type='Sharpness', prob=0.0, level=8),
>>>         dict(
>>>             type='Shear',
>>>             prob=0.4,
>>>             level=0,
>>>             replace=replace,
>>>             axis='x')
>>>     ],
>>>     [
>>>         dict(
>>>             type='Rotate',
>>>             prob=0.6,
>>>             level=10,
>>>             replace=replace),
>>>         dict(type='Color', prob=1.0, level=6)
>>>     ]
>>> ]
>>> augmentation = AutoAugment(policies)
>>> img = np.ones(100, 100, 3)
>>> gt_bboxes = np.ones(10, 4)
>>> results = dict(img=img, gt_bboxes=gt_bboxes)
>>> results = augmentation(results)
```

class mmdet.datasets.pipelines.**BrightnessTransform**(*level*, *prob=0.5*)

Apply Brightness transformation to image. The bboxes, masks and segmentations are not modified.

Parameters

- **level** (*int | float*) – Should be in range [0,_MAX_LEVEL].
- **prob** (*float*) – The probability for performing Brightness transformation.

class mmdet.datasets.pipelines.**Collect**(*keys*, *meta_keys=('filename', 'ori_filename', 'ori_shape', 'img_shape', 'pad_shape', 'scale_factor', 'flip', 'flip_direction', 'img_norm_cfg')*)

Collect data from the loader relevant to the specific task.

This is usually the last stage of the data loader pipeline. Typically keys is set to some subset of "img", "proposals", "gt_bboxes", "gt_bboxes_ignore", "gt_labels", and/or "gt_masks".

The "img_meta" item is always populated. The contents of the "img_meta" dictionary depends on "meta_keys". By default this includes:

- "img_shape": shape of the image input to the network as a tuple (h, w, c). Note that images may be zero padded on the bottom/right if the batch tensor is larger than this shape.
- "scale_factor": a float indicating the preprocessing scale

- "flip": a boolean indicating if image flip transform was used
- "filename": path to the image file
- "ori_shape": original shape of the image as a tuple (h, w, c)
- "pad_shape": image shape after padding
- "img_norm_cfg": a dict of normalization information:
  - mean - per channel mean subtraction
  - std - per channel std divisor
  - to_rgb - bool indicating if bgr was converted to rgb

Parameters

- **keys** (*Sequence[str]*) – Keys of results to be collected in `data`.
- **meta_keys** (*Sequence[str], optional*) – Meta keys to be converted to `mmcv.DataContainer` and collected in `data[img_metas]`. Default: `('filename', 'ori_filename', 'ori_shape', 'img_shape', 'pad_shape', 'scale_factor', 'flip', 'flip_direction', 'img_norm_cfg')`

class mmdet.datasets.pipelines.**ColorTransform**(*level*, *prob=0.5*)

Apply Color transformation to image. The bboxes, masks, and segmentations are not modified.

Parameters

- **level** (*int | float*) – Should be in range [0,_MAX_LEVEL].
- **prob** (*float*) – The probability for performing Color transformation.

class mmdet.datasets.pipelines.**Compose**(*transforms*)

Compose multiple transforms sequentially.

Parameters **transforms** (*Sequence[dict | callable]*) – Sequence of transform object or config dict to be composed.

class mmdet.datasets.pipelines.**ContrastTransform**(*level*, *prob=0.5*)

Apply Contrast transformation to image. The bboxes, masks and segmentations are not modified.

Parameters

- **level** (*int | float*) – Should be in range [0,_MAX_LEVEL].
- **prob** (*float*) – The probability for performing Contrast transformation.

class mmdet.datasets.pipelines.**CutOut**(*n_holes*, *cutout_shape=None*, *cutout_ratio=None*, *fill_in=(0, 0, 0)*)

CutOut operation.

Randomly drop some regions of image used in Cutout.

Parameters

- **n_holes** (*int | tuple[int, int]*) – Number of regions to be dropped. If it is given as a list, number of holes will be randomly selected from the closed interval [*n_holes[0]*, *n_holes[1]*].
- **cutout_shape** (*tuple[int, int] | list[tuple[int, int]]*) – The candidate shape of dropped regions. It can be *tuple[int, int]* to use a fixed cutout shape, or *list[tuple[int, int]]* to randomly choose shape from the list.

- **cutout_ratio** (*tuple[float, float] | list[tuple[float, float]]*) – The candidate ratio of dropped regions. It can be *tuple[float, float]* to use a fixed ratio or *list[tuple[float, float]]* to randomly choose ratio from the list. Please note that *cutout_shape* and *cutout_ratio* cannot be both given at the same time.
- **fill_in** (*tuple[float, float, float] | tuple[int, int, int]*) – The value of pixel to fill in the dropped regions. Default: (0, 0, 0).

class mmdet.datasets.pipelines.**DefaultFormatBundle**(*img_to_float=True*)

Default formatting bundle.

It simplifies the pipeline of formatting common fields, including "img", "proposals", "gt_bboxes", "gt_labels", "gt_masks" and "gt_semantic_seg". These fields are formatted as follows.

- img: (1)transpose, (2)to tensor, (3)to DataContainer (stack=True)
- proposals: (1)to tensor, (2)to DataContainer
- gt_bboxes: (1)to tensor, (2)to DataContainer
- gt_bboxes_ignore: (1)to tensor, (2)to DataContainer
- gt_labels: (1)to tensor, (2)to DataContainer
- gt_masks: (1)to tensor, (2)to DataContainer (cpu_only=True)
- gt_semantic_seg: (1)unsqueeze dim-0 (2)to tensor, (3)to DataContainer (stack=True)

Parameters **img_to_float** (*bool*) – Whether to force the image to be converted to float type. Default: True.

class mmdet.datasets.pipelines.**EqualizeTransform**(*prob=0.5*)

Apply Equalize transformation to image. The bboxes, masks and segmentations are not modified.

Parameters **prob** (*float*) – The probability for performing Equalize transformation.

class mmdet.datasets.pipelines.**Expand**(*mean=(0, 0, 0)*, *to_rgb=True*, *ratio_range=(1, 4)*, *seg_ignore_label=None*, *prob=0.5*)

Random expand the image & bboxes.

Randomly place the original image on a canvas of 'ratio' x original image size filled with mean values. The ratio is in the range of ratio_range.

Parameters

- **mean** (*tuple*) – mean value of dataset.
- **to_rgb** (*bool*) – if need to convert the order of mean to align with RGB.
- **ratio_range** (*tuple*) – range of expand ratio.
- **prob** (*float*) – probability of applying this transformation

class mmdet.datasets.pipelines.**ImageToTensor**(*keys*)

Convert image to `torch.Tensor` by given keys.

The dimension order of input image is (H, W, C). The pipeline will convert it to (C, H, W). If only 2 dimension (H, W) is given, the output would be (1, H, W).

Parameters **keys** (*Sequence[str]*) – Key of images to be converted to Tensor.

class mmdet.datasets.pipelines.**InstaBoost**(*action_candidate=('normal', 'horizontal', 'skip')*, *action_prob=(1, 0, 0)*, *scale=(0.8, 1.2)*, *dx=15*, *dy=15*, *theta=(- 1, 1)*, *color_prob=0.5*, *hflag=False*, *aug_ratio=0.5*)

Data augmentation method in InstaBoost: Boosting Instance Segmentation Via Probability Map Guided Copy-Pasting.

Refer to https://github.com/GothicAi/Instaboost for implementation details.

Parameters

- **action_candidate** (*tuple*) – Action candidates. "normal", "horizontal", "vertical", "skip" are supported. Default: ('normal', 'horizontal', 'skip').
- **action_prob** (*tuple*) – Corresponding action probabilities. Should be the same length as action_candidate. Default: (1, 0, 0).
- **scale** (*tuple*) – (min scale, max scale). Default: (0.8, 1.2).
- **dx** (*int*) – The maximum x-axis shift will be (instance width) / dx. Default 15.
- **dy** (*int*) – The maximum y-axis shift will be (instance height) / dy. Default 15.
- **theta** (*tuple*) – (min rotation degree, max rotation degree). Default: (-1, 1).
- **color_prob** (*float*) – Probability of images for color augmentation. Default 0.5.
- **heatmap_flag** (*bool*) – Whether to use heatmap guided. Default False.
- **aug_ratio** (*float*) – Probability of applying this transformation. Default 0.5.

class mmdet.datasets.pipelines.**LoadAnnotations**(*with_bbox=True*, *with_label=True*, *with_mask=False*, *with_seg=False*, *poly2mask=True*, *file_client_args={'backend': 'disk'}*)

Load multiple types of annotations.

Parameters

- **with_bbox** (*bool*) – Whether to parse and load the bbox annotation. Default: True.
- **with_label** (*bool*) – Whether to parse and load the label annotation. Default: True.
- **with_mask** (*bool*) – Whether to parse and load the mask annotation. Default: False.
- **with_seg** (*bool*) – Whether to parse and load the semantic segmentation annotation. Default: False.
- **poly2mask** (*bool*) – Whether to convert the instance masks from polygons to bitmaps. Default: True.
- **file_client_args** (*dict*) – Arguments to instantiate a FileClient. See `mmcv.fileio.FileClient` for details. Defaults to `dict(backend='disk')`.

process_polygons(*polygons*)

Convert polygons to list of ndarray and filter invalid polygons.

Parameters **polygons** (*list[list]*) – Polygons of one instance.

Returns Processed polygons.

Return type list[numpy.ndarray]

class mmdet.datasets.pipelines.**LoadImageFromFile**(*to_float32=False*, *color_type='color'*, *file_client_args={'backend': 'disk'}*)

Load an image from file.

Required keys are "img_prefix" and "img_info" (a dict that must contain the key "filename"). Added or updated keys are "filename", "img", "img_shape", "ori_shape" (same as *img_shape*), "pad_shape" (same as *img_shape*), "scale_factor" (1.0) and "img_norm_cfg" (means=0 and stds=1).

Parameters

- **to_float32** (*bool*) – Whether to convert the loaded image to a float32 numpy array. If set to False, the loaded image is an uint8 array. Defaults to False.
- **color_type** (*str*) – The flag argument for `mmcv.imfrombytes()`. Defaults to 'color'.
- **file_client_args** (*dict*) – Arguments to instantiate a FileClient. See `mmcv.fileio.FileClient` for details. Defaults to `dict(backend='disk')`.

class mmdet.datasets.pipelines.**LoadImageFromWebcam**(*to_float32=False*, *color_type='color'*, *file_client_args={'backend': 'disk'}*)

Load an image from webcam.

Similar with *LoadImageFromFile*, but the image read from webcam is in `results['img']`.

class mmdet.datasets.pipelines.**LoadMultiChannelImageFromFiles**(*to_float32=False*, *color_type='unchanged'*, *file_client_args={'backend': 'disk'}*)

Load multi-channel images from a list of separate channel files.

Required keys are "img_prefix" and "img_info" (a dict that must contain the key "filename", which is expected to be a list of filenames). Added or updated keys are "filename", "img", "img_shape", "ori_shape" (same as *img_shape*), "pad_shape" (same as *img_shape*), "scale_factor" (1.0) and "img_norm_cfg" (means=0 and stds=1).

Parameters

- **to_float32** (*bool*) – Whether to convert the loaded image to a float32 numpy array. If set to False, the loaded image is an uint8 array. Defaults to False.
- **color_type** (*str*) – The flag argument for `mmcv.imfrombytes()`. Defaults to 'color'.
- **file_client_args** (*dict*) – Arguments to instantiate a FileClient. See `mmcv.fileio.FileClient` for details. Defaults to `dict(backend='disk')`.

class mmdet.datasets.pipelines.**LoadProposals**(*num_max_proposals=None*)

Load proposal pipeline.

Required key is "proposals". Updated keys are "proposals", "bbox_fields".

Parameters **num_max_proposals** (*int, optional*) – Maximum number of proposals to load. If not specified, all proposals will be loaded.

class mmdet.datasets.pipelines.**MinIoURandomCrop**(*min_ious=(0.1, 0.3, 0.5, 0.7, 0.9)*, *min_crop_size=0.3*, *bbox_clip_border=True*)

Random crop the image & bboxes, the cropped patches have minimum IoU requirement with original image & bboxes, the IoU threshold is randomly selected from min_ious.

Parameters

- **min_ious** (*tuple*) – minimum IoU threshold for all intersections with
- **boxes** (*bounding*) –
- **min_crop_size** (*float*) – minimum crop's size (i.e. h,w := a*h, a*w,
- **a >= min_crop_size)** (*where*) –

- **bbox_clip_border** (*bool, optional*) – Whether clip the objects outside the border of the image. Defaults to True.

Note: The keys for bboxes, labels and masks should be paired. That is, *gt_bboxes* corresponds to *gt_labels* and *gt_masks*, and *gt_bboxes_ignore* to *gt_labels_ignore* and *gt_masks_ignore*.

class mmdet.datasets.pipelines.**MixUp**(*img_scale=(640, 640)*, *ratio_range=(0.5, 1.5)*, *flip_ratio=0.5*, *pad_val=114*, *max_iters=15*, *min_bbox_size=5*, *min_area_ratio=0.2*, *max_aspect_ratio=20*, *skip_filter=True*)

MixUp data augmentation.

Parameters

- **img_scale** (*Sequence[int]*) – Image output size after mixup pipeline. Default: (640, 640).
- **ratio_range** (*Sequence[float]*) – Scale ratio of mixup image. Default: (0.5, 1.5).
- **flip_ratio** (*float*) – Horizontal flip ratio of mixup image. Default: 0.5.
- **pad_val** (*int*) – Pad value. Default: 114.
- **max_iters** (*int*) – The maximum number of iterations. If the number of iterations is greater than *max_iters*, but gt_bbox is still empty, then the iteration is terminated. Default: 15.
- **min_bbox_size** (*float*) – Width and height threshold to filter bboxes. If the height or width of a box is smaller than this value, it will be removed. Default: 5.
- **min_area_ratio** (*float*) – Threshold of area ratio between original bboxes and wrapped bboxes. If smaller than this value, the box will be removed. Default: 0.2.
- **max_aspect_ratio** (*float*) – Aspect ratio of width and height threshold to filter bboxes. If max(h/w, w/h) larger than this value, the box will be removed. Default: 20.
- **skip_filter** (*bool*) – Whether to skip filtering rules. If it is True, the filter rule will not be applied, and the *min_bbox_size* and *min_area_ratio* and *max_aspect_ratio* is invalid. Default to True.

get_indexes(*dataset*)

Call function to collect indexes.

Parameters **dataset** (MultiImageMixDataset) – The dataset.

Returns indexes.

Return type list

class mmdet.datasets.pipelines.**Mosaic**(*img_scale=(640, 640)*, *center_ratio_range=(0.5, 1.5)*, *min_bbox_size=0*, *skip_filter=True*, *pad_val=114*)

Mosaic augmentation.

Given 4 images, mosaic transform combines them into one output image. The output image is composed of the parts from each sub- image.

```
                mosaic transform
                   center_x
          +------------------------------+
          |       pad        |  pad      |
          |      +-----------+           |
          |      |           |           |
```

(continues on next page)

(continued from previous page)

```
            |      |  image1   |--------+  |
            |      |           |        |  |
            |      |           | image2 |  |
center_y    |----+-------------+-----------|
            |    |   cropped   |           |
            |pad |   image3    |  image4   |
            |    |             |           |
            +----|-------------+-----------+
                 |             |
                 +-------------+

The mosaic transform steps are as follows:

    1. Choose the mosaic center as the intersections of 4 images
    2. Get the left top image according to the index, and randomly
       sample another 3 images from the custom dataset.
    3. Sub image will be cropped if image is larger than mosaic patch
```

Parameters

- **img_scale** (*Sequence[int]*) – Image size after mosaic pipeline of single image. Default to (640, 640).
- **center_ratio_range** (*Sequence[float]*) – Center ratio range of mosaic output. Default to (0.5, 1.5).
- **min_bbox_size** (*int | float*) – The minimum pixel for filtering invalid bboxes after the mosaic pipeline. Default to 0.
- **skip_filter** (*bool*) – Whether to skip filtering rules. If it is True, the filter rule will not be applied, and the *min_bbox_size* is invalid. Default to True.
- **pad_val** (*int*) – Pad value. Default to 114.

get_indexes(*dataset*)

Call function to collect indexes.

Parameters **dataset** (MultiImageMixDataset) – The dataset.

Returns indexes.

Return type list

class mmdet.datasets.pipelines.**MultiScaleFlipAug**(*transforms*, *img_scale=None*, *scale_factor=None*, *flip=False*, *flip_direction='horizontal'*)

Test-time augmentation with multiple scales and flipping.

An example configuration is as followed:

```
img_scale=[(1333, 400), (1333, 800)],
flip=True,
transforms=[
    dict(type='Resize', keep_ratio=True),
    dict(type='RandomFlip'),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='Pad', size_divisor=32),
    dict(type='ImageToTensor', keys=['img']),
```

(continues on next page)

(continued from previous page)

```
    dict(type='Collect', keys=['img']),
]
```

After MultiScaleFLipAug with above configuration, the results are wrapped into lists of the same length as followed:

```
dict(
    img=[...],
    img_shape=[...],
    scale=[(1333, 400), (1333, 400), (1333, 800), (1333, 800)]
    flip=[False, True, False, True]
    ...
)
```

Parameters

- **transforms** (*list[dict]*) – Transforms to apply in each augmentation.
- **img_scale** (*tuple | list[tuple] | None*) – Images scales for resizing.
- **scale_factor** (*float | list[float] | None*) – Scale factors for resizing.
- **flip** (*bool*) – Whether apply flip augmentation. Default: False.
- **flip_direction** (*str | list[str]*) – Flip augmentation directions, options are "horizontal", "vertical" and "diagonal". If flip_direction is a list, multiple flip augmentations will be applied. It has no effect when flip == False. Default: "horizontal".

class mmdet.datasets.pipelines.**Normalize**(*mean*, *std*, *to_rgb=True*)

Normalize the image.

Added key is "img_norm_cfg".

Parameters

- **mean** (*sequence*) – Mean values of 3 channels.
- **std** (*sequence*) – Std values of 3 channels.
- **to_rgb** (*bool*) – Whether to convert the image from BGR to RGB, default is true.

class mmdet.datasets.pipelines.**Pad**(*size=None*, *size_divisor=None*, *pad_to_square=False*, *pad_val={'img': 0, 'masks': 0, 'seg': 255}*)

Pad the image & masks & segmentation map.

There are two padding modes: (1) pad to a fixed size and (2) pad to the minimum size that is divisible by some number. Added keys are "pad_shape", "pad_fixed_size", "pad_size_divisor",

Parameters

- **size** (*tuple, optional*) – Fixed padding size.
- **size_divisor** (*int, optional*) – The divisor of padded size.
- **pad_to_square** (*bool*) – Whether to pad the image into a square. Currently only used for YOLOX. Default: False.
- **pad_val** (*dict, optional*) – A dict for padding value, the default value is *dict(img=0, masks=0, seg=255)*.

class mmdet.datasets.pipelines.**PhotoMetricDistortion**(*brightness_delta=32*, *contrast_range=(0.5, 1.5)*, *saturation_range=(0.5, 1.5)*, *hue_delta=18*)

Apply photometric distortion to image sequentially, every transformation is applied with a probability of 0.5. The position of random contrast is in second or second to last.

1. random brightness
2. random contrast (mode 0)
3. convert color from BGR to HSV
4. random saturation
5. random hue
6. convert color from HSV to BGR
7. random contrast (mode 1)
8. randomly swap channels

Parameters

- **brightness_delta** (*int*) – delta of brightness.
- **contrast_range** (*tuple*) – range of contrast.
- **saturation_range** (*tuple*) – range of saturation.
- **hue_delta** (*int*) – delta of hue.

class mmdet.datasets.pipelines.**RandomAffine**(*max_rotate_degree=10.0*, *max_translate_ratio=0.1*, *scaling_ratio_range=(0.5, 1.5)*, *max_shear_degree=2.0*, *border=(0, 0)*, *border_val=(114, 114, 114)*, *min_bbox_size=2*, *min_area_ratio=0.2*, *max_aspect_ratio=20*, *skip_filter=True*)

Random affine transform data augmentation.

This operation randomly generates affine transform matrix which including rotation, translation, shear and scaling transforms.

Parameters

- **max_rotate_degree** (*float*) – Maximum degrees of rotation transform. Default: 10.
- **max_translate_ratio** (*float*) – Maximum ratio of translation. Default: 0.1.
- **scaling_ratio_range** (*tuple[float]*) – Min and max ratio of scaling transform. Default: (0.5, 1.5).
- **max_shear_degree** (*float*) – Maximum degrees of shear transform. Default: 2.
- **border** (*tuple[int]*) – Distance from height and width sides of input image to adjust output shape. Only used in mosaic dataset. Default: (0, 0).
- **border_val** (*tuple[int]*) – Border padding values of 3 channels. Default: (114, 114, 114).
- **min_bbox_size** (*float*) – Width and height threshold to filter bboxes. If the height or width of a box is smaller than this value, it will be removed. Default: 2.
- **min_area_ratio** (*float*) – Threshold of area ratio between original bboxes and wrapped bboxes. If smaller than this value, the box will be removed. Default: 0.2.

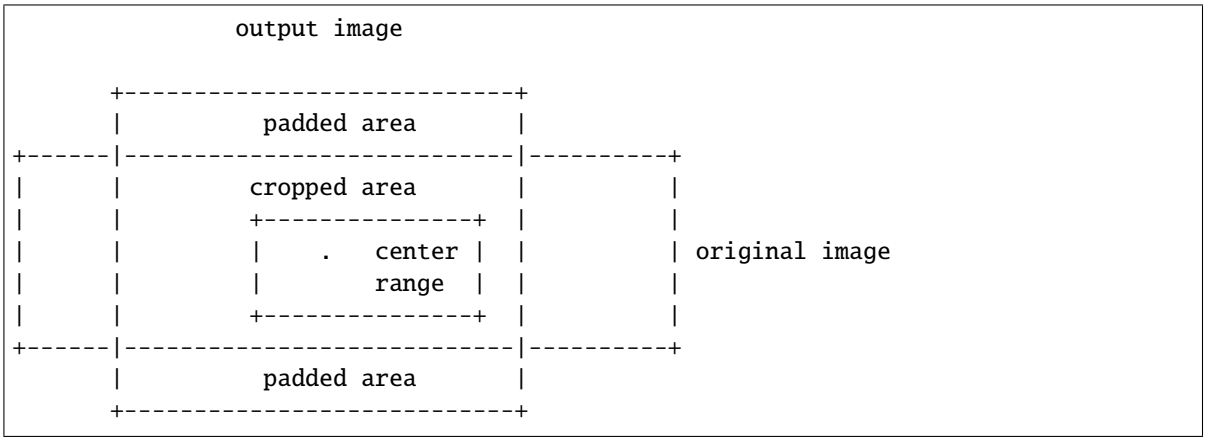
- **max_aspect_ratio** (*float*) – Aspect ratio of width and height threshold to filter bboxes. If max(h/w, w/h) larger than this value, the box will be removed.
- **skip_filter** (*bool*) – Whether to skip filtering rules. If it is True, the filter rule will not be applied, and the *min_bbox_size* and *min_area_ratio* and *max_aspect_ratio* is invalid. Default to True.

class mmdet.datasets.pipelines.**RandomCenterCropPad**(*crop_size=None*, *ratios=(0.9, 1.0, 1.1)*, *border=128*, *mean=None*, *std=None*, *to_rgb=None*, *test_mode=False*, *test_pad_mode=('logical_or', 127)*, *test_pad_add_pix=0*, *bbox_clip_border=True*)

Random center crop and random around padding for CornerNet.

This operation generates randomly cropped image from the original image and pads it simultaneously. Different from *RandomCrop*, the output shape may not equal to `crop_size` strictly. We choose a random value from `ratios` and the output shape could be larger or smaller than `crop_size`. The padding operation is also different from *Pad*, here we use around padding instead of right-bottom padding.

The relation between output image (padding image) and original image:



There are 5 main areas in the figure:

- output image: output image of this operation, also called padding image in following instruction.
- original image: input image of this operation.
- padded area: non-intersect area of output image and original image.
- cropped area: the overlap of output image and original image.
- center range: a smaller area where random center chosen from. center range is computed by `border` and original image's shape to avoid our random center is too close to original image's border.

Also this operation act differently in train and test mode, the summary pipeline is listed below.

Train pipeline:

1. Choose a `random_ratio` from `ratios`, the shape of padding image will be `random_ratio * crop_size`.
2. Choose a `random_center` in center range.
3. Generate padding image with center matches the `random_center`.
4. Initialize the padding image with pixel value equals to `mean`.
5. Copy the cropped area to padding image.

6. Refine annotations.

Test pipeline:

1. Compute output shape according to `test_pad_mode`.
2. Generate padding image with center matches the original image center.
3. Initialize the padding image with pixel value equals to `mean`.
4. Copy the `cropped area` to padding image.

Parameters

- **crop_size** (*tuple | None*) – expected size after crop, final size will computed according to ratio. Requires (h, w) in train mode, and None in test mode.
- **ratios** (*tuple*) – random select a ratio from tuple and crop image to (crop_size[0] * ratio) * (crop_size[1] * ratio). Only available in train mode.
- **border** (*int*) – max distance from center select area to image border. Only available in train mode.
- **mean** (*sequence*) – Mean values of 3 channels.
- **std** (*sequence*) – Std values of 3 channels.
- **to_rgb** (*bool*) – Whether to convert the image from BGR to RGB.
- **test_mode** (*bool*) – whether involve random variables in transform. In train mode, crop_size is fixed, center coords and ratio is random selected from predefined lists. In test mode, crop_size is image's original shape, center coords and ratio is fixed.
- **test_pad_mode** (*tuple*) – padding method and padding shape value, only available in test mode. Default is using 'logical_or' with 127 as padding shape value.
  - 'logical_or': final_shape = input_shape | padding_shape_value
  - 'size_divisor': final_shape = int(ceil(input_shape / padding_shape_value) * padding_shape_value)
- **test_pad_add_pix** (*int*) – Extra padding pixel in test mode. Default 0.
- **bbox_clip_border** (*bool, optional*) – Whether clip the objects outside the border of the image. Defaults to True.

class mmdet.datasets.pipelines.**RandomCrop**(*crop_size*, *crop_type='absolute'*, *allow_negative_crop=False*, *recompute_bbox=False*, *bbox_clip_border=True*)

Random crop the image & bboxes & masks.

The absolute *crop_size* is sampled based on *crop_type* and *image_size*, then the cropped results are generated.

Parameters

- **crop_size** (*tuple*) – The relative ratio or absolute pixels of height and width.
- **crop_type** (*str, optional*) – one of "relative_range", "relative", "absolute", "absolute_range". "relative" randomly crops (h * crop_size[0], w * crop_size[1]) part from an input of size (h, w). "relative_range" uniformly samples relative crop size from range [crop_size[0], 1] and [crop_size[1], 1] for height and width respectively. "absolute" crops from an input with absolute size (crop_size[0], crop_size[1]). "absolute_range" uniformly samples crop_h in range [crop_size[0], min(h, crop_size[1])] and crop_w in range [crop_size[0], min(w, crop_size[1])]. Default "absolute".

- **allow_negative_crop** (*bool, optional*) – Whether to allow a crop that does not contain any bbox area. Default False.
- **recompute_bbox** (*bool, optional*) – Whether to re-compute the boxes based on cropped instance masks. Default False.
- **bbox_clip_border** (*bool, optional*) – Whether clip the objects outside the border of the image. Defaults to True.

Note:

- **If the image is smaller than the absolute crop size, return the** original image.
- The keys for bboxes, labels and masks must be aligned. That is, *gt_bboxes* corresponds to *gt_labels* and *gt_masks*, and *gt_bboxes_ignore* corresponds to *gt_labels_ignore* and *gt_masks_ignore*.
- If the crop does not contain any gt-bbox region and *allow_negative_crop* is set to False, skip this image.

---

class mmdet.datasets.pipelines.**RandomFlip**(*flip_ratio=None*, *direction='horizontal'*)

Flip the image & bbox & mask.

If the input dict contains the key "flip", then the flag will be used, otherwise it will be randomly decided by a ratio specified in the init method.

When random flip is enabled, `flip_ratio`/`direction` can either be a float/string or tuple of float/string. There are 3 flip modes:

- **`flip_ratio` is float, `direction` is string: the image will be** `direction``ly flipped with probability of ``flip_ratio` . E.g., `flip_ratio=0.5`, `direction='horizontal'`, then image will be horizontally flipped with probability of 0.5.
- **`flip_ratio` is float, `direction` is list of string: the image will** be `direction[i]``ly flipped with probability of ``flip_ratio/len(direction)`. E.g., `flip_ratio=0.5`, `direction=['horizontal', 'vertical']`, then image will be horizontally flipped with probability of 0.25, vertically with probability of 0.25.
- **`flip_ratio` is list of float, `direction` is list of string:** given `len(flip_ratio) == len(direction)`, the image will be `direction[i]``ly flipped with probability of ``flip_ratio[i]`. E.g., `flip_ratio=[0.3, 0.5]`, `direction=['horizontal', 'vertical']`, then image will be horizontally flipped with probability of 0.3, vertically with probability of 0.5.

Parameters

- **flip_ratio** (*float | list[float], optional*) – The flipping probability. Default: None.
- **direction** (*str | list[str], optional*) – The flipping direction. Options are 'horizontal', 'vertical', 'diagonal'. Default: 'horizontal'. If input is a list, the length must equal `flip_ratio`. Each element in `flip_ratio` indicates the flip probability of corresponding direction.

bbox_flip(*bboxes*, *img_shape*, *direction*)

Flip bboxes horizontally.

Parameters

- **bboxes** (*numpy.ndarray*) – Bounding boxes, shape (..., 4*k)
- **img_shape** (*tuple[int]*) – Image shape (height, width)

- **direction** (*str*) – Flip direction. Options are 'horizontal', 'vertical'.

Returns Flipped bounding boxes.

Return type numpy.ndarray

class mmdet.datasets.pipelines.**RandomShift**(*shift_ratio=0.5*, *max_shift_px=32*, *filter_thr_px=1*)

Shift the image and box given shift pixels and probability.

Parameters

- **shift_ratio** (*float*) – Probability of shifts. Default 0.5.
- **max_shift_px** (*int*) – The max pixels for shifting. Default 32.
- **filter_thr_px** (*int*) – The width and height threshold for filtering. The bbox and the rest of the targets below the width and height threshold will be filtered. Default 1.

class mmdet.datasets.pipelines.**Resize**(*img_scale=None*, *multiscale_mode='range'*, *ratio_range=None*, *keep_ratio=True*, *bbox_clip_border=True*, *backend='cv2'*, *override=False*)

Resize images & bbox & mask.

This transform resizes the input image to some scale. Bboxes and masks are then resized with the same scale factor. If the input dict contains the key "scale", then the scale in the input dict is used, otherwise the specified scale in the init method is used. If the input dict contains the key "scale_factor" (if MultiScaleFlipAug does not give img_scale but scale_factor), the actual scale will be computed by image shape and scale_factor.

img_scale can either be a tuple (single-scale) or a list of tuple (multi-scale). There are 3 multiscale modes:

- `ratio_range is not None`: randomly sample a ratio from the ratio range and multiply it with the image scale.
- `ratio_range is None` and `multiscale_mode == "range"`: randomly sample a scale from the multi-scale range.
- `ratio_range is None` and `multiscale_mode == "value"`: randomly sample a scale from multiple scales.

Parameters

- **img_scale** (*tuple or list[tuple]*) – Images scales for resizing.
- **multiscale_mode** (*str*) – Either "range" or "value".
- **ratio_range** (*tuple[float]*) – (min_ratio, max_ratio)
- **keep_ratio** (*bool*) – Whether to keep the aspect ratio when resizing the image.
- **bbox_clip_border** (*bool, optional*) – Whether clip the objects outside the border of the image. Defaults to True.
- **backend** (*str*) – Image resize backend, choices are 'cv2' and 'pillow'. These two backends generates slightly different results. Defaults to 'cv2'.
- **override** (*bool, optional*) – Whether to override *scale* and *scale_factor* so as to call resize twice. Default False. If True, after the first resizing, the existed *scale* and *scale_factor* will be ignored so the second resizing can be allowed. This option is a work-around for multiple times of resize in DETR. Defaults to False.

static **random_sample**(*img_scales*)

Randomly sample an img_scale when `multiscale_mode=='range'`.

Parameters **img_scales** (*list[tuple]*) – Images scale range for sampling. There must be two tuples in img_scales, which specify the lower and upper bound of image scales.

Returns Returns a tuple `(img_scale, None)`, where `img_scale` is sampled scale and None is just a placeholder to be consistent with *random_select()*.

Return type (tuple, None)

static random_sample_ratio(*img_scale*, *ratio_range*)

Randomly sample an img_scale when `ratio_range` is specified.

A ratio will be randomly sampled from the range specified by `ratio_range`. Then it would be multiplied with `img_scale` to generate sampled scale.

Parameters

- **img_scale** (*tuple*) – Images scale base to multiply with ratio.
- **ratio_range** (*tuple[float]*) – The minimum and maximum ratio to scale the `img_scale`.

Returns Returns a tuple `(scale, None)`, where `scale` is sampled ratio multiplied with `img_scale` and None is just a placeholder to be consistent with *random_select()*.

Return type (tuple, None)

static random_select(*img_scales*)

Randomly select an img_scale from given candidates.

Parameters **img_scales** (*list[tuple]*) – Images scales for selection.

Returns Returns a tuple `(img_scale, scale_dix)`, where `img_scale` is the selected image scale and `scale_idx` is the selected index in the given candidates.

Return type (tuple, int)

class mmdet.datasets.pipelines.**Rotate**(*level*, *scale=1*, *center=None*, *img_fill_val=128*, *seg_ignore_label=255*, *prob=0.5*, *max_rotate_angle=30*, *random_negative_prob=0.5*)

Apply Rotate Transformation to image (and its corresponding bbox, mask, segmentation).

Parameters

- **level** (*int | float*) – The level should be in range (0,_MAX_LEVEL].
- **scale** (*int | float*) – Isotropic scale factor. Same in `mmcv.imrotate`.
- **center** (*int | float | tuple[float]*) – Center point (w, h) of the rotation in the source image. If None, the center of the image will be used. Same in `mmcv.imrotate`.
- **img_fill_val** (*int | float | tuple*) – The fill value for image border. If float, the same value will be used for all the three channels of image. If tuple, the should be 3 elements (e.g. equals the number of channels for image).
- **seg_ignore_label** (*int*) – The fill value used for segmentation map. Note this value must equals `ignore_label` in `semantic_head` of the corresponding config. Default 255.
- **prob** (*float*) – The probability for perform transformation and should be in range 0 to 1.
- **max_rotate_angle** (*int | float*) – The maximum angles for rotate transformation.
- **random_negative_prob** (*float*) – The probability that turns the offset negative.

class mmdet.datasets.pipelines.**SegRescale**(*scale_factor=1*, *backend='cv2'*)

Rescale semantic segmentation maps.

Parameters

- **scale_factor** (*float*) – The scale factor of the final output.
- **backend** (*str*) – Image rescale backend, choices are 'cv2' and 'pillow'. These two backends generates slightly different results. Defaults to 'cv2'.

class mmdet.datasets.pipelines.**Shear**(*level*, *img_fill_val=128*, *seg_ignore_label=255*, *prob=0.5*, *direction='horizontal'*, *max_shear_magnitude=0.3*, *random_negative_prob=0.5*, *interpolation='bilinear'*)

Apply Shear Transformation to image (and its corresponding bbox, mask, segmentation).

Parameters

- **level** (*int | float*) – The level should be in range [0,_MAX_LEVEL].
- **img_fill_val** (*int | float | tuple*) – The filled values for image border. If float, the same fill value will be used for all the three channels of image. If tuple, the should be 3 elements.
- **seg_ignore_label** (*int*) – The fill value used for segmentation map. Note this value must equals `ignore_label` in `semantic_head` of the corresponding config. Default 255.
- **prob** (*float*) – The probability for performing Shear and should be in range [0, 1].
- **direction** (*str*) – The direction for shear, either "horizontal" or "vertical".
- **max_shear_magnitude** (*float*) – The maximum magnitude for Shear transformation.
- **random_negative_prob** (*float*) – The probability that turns the offset negative. Should be in range [0,1]
- **interpolation** (*str*) – Same as in `mmcv.imshear()`.

class mmdet.datasets.pipelines.**ToDataContainer**(*fields=({'key': 'img', 'stack': True}, {'key': 'gt_bboxes'}, {'key': 'gt_labels'})*)

Convert results to `mmcv.DataContainer` by given fields.

Parameters **fields** (*Sequence[dict]*) – Each field is a dict like `dict(key='xxx', **kwargs)`. The `key` in result will be converted to `mmcv.DataContainer` with `**kwargs`. Default: `(dict(key='img', stack=True), dict(key='gt_bboxes'), dict(key='gt_labels'))`.

class mmdet.datasets.pipelines.**ToTensor**(*keys*)

Convert some results to `torch.Tensor` by given keys.

Parameters **keys** (*Sequence[str]*) – Keys that need to be converted to Tensor.

class mmdet.datasets.pipelines.**Translate**(*level*, *prob=0.5*, *img_fill_val=128*, *seg_ignore_label=255*, *direction='horizontal'*, *max_translate_offset=250.0*, *random_negative_prob=0.5*, *min_size=0*)

Translate the images, bboxes, masks and segmentation maps horizontally or vertically.

Parameters

- **level** (*int | float*) – The level for Translate and should be in range [0,_MAX_LEVEL].
- **prob** (*float*) – The probability for performing translation and should be in range [0, 1].
- **img_fill_val** (*int | float | tuple*) – The filled value for image border. If float, the same fill value will be used for all the three channels of image. If tuple, the should be 3 elements (e.g. equals the number of channels for image).
- **seg_ignore_label** (*int*) – The fill value used for segmentation map. Note this value must equals `ignore_label` in `semantic_head` of the corresponding config. Default 255.

- **direction** (*str*) – The translate direction, either "horizontal" or "vertical".
- **max_translate_offset** (*int | float*) – The maximum pixel's offset for Translate.
- **random_negative_prob** (*float*) – The probability that turns the offset negative.
- **min_size** (*int | float*) – The minimum pixel for filtering invalid bboxes after the translation.

class mmdet.datasets.pipelines.**Transpose**(*keys*, *order*)
Transpose some results by given keys.

Parameters

- **keys** (*Sequence[str]*) – Keys of results to be transposed.
- **order** (*Sequence[int]*) – Order of transpose.

class mmdet.datasets.pipelines.**YOLOXHSVRandomAug**(*hue_delta=5*, *saturation_delta=30*, *value_delta=30*)
Apply HSV augmentation to image sequentially. It is referenced from https://github.com/Megvii- BaseDetection/YOLOX/blob/main/yolox/data/data_augment.py#L21.

Parameters

- **hue_delta** (*int*) – delta of hue. Default: 5.
- **saturation_delta** (*int*) – delta of saturation. Default: 30.
- **value_delta** (*int*) – delat of value. Default: 30.

mmdet.datasets.pipelines.**to_tensor**(*data*)
Convert objects of various python types to `torch.Tensor`.

Supported types are: `numpy.ndarray`, `torch.Tensor`, `Sequence`, `int` and `float`.

Parameters **data** (*torch.Tensor | numpy.ndarray | Sequence | int | float*) – Data to be converted.

39.3 samplers

class mmdet.datasets.samplers.**DistributedGroupSampler**(*dataset*, *samples_per_gpu=1*, *num_replicas=None*, *rank=None*, *seed=0*)
Sampler that restricts data loading to a subset of the dataset.

It is especially useful in conjunction with `torch.nn.parallel.DistributedDataParallel`. In such case, each process can pass a DistributedSampler instance as a DataLoader sampler, and load a subset of the original dataset that is exclusive to it.

Note: Dataset is assumed to be of constant size.

Parameters

- **dataset** – Dataset used for sampling.
- **num_replicas** (*optional*) – Number of processes participating in distributed training.
- **rank** (*optional*) – Rank of the current process within num_replicas.
- **seed** (*int, optional*) – random seed used to shuffle the sampler if `shuffle=True`. This number should be identical across all processes in the distributed group. Default: 0.

class mmdet.datasets.samplers.**DistributedSampler**(*dataset*, *num_replicas=None*, *rank=None*, *shuffle=True*, *seed=0*)

class mmdet.datasets.samplers.**GroupSampler**(*dataset*, *samples_per_gpu=1*)

class mmdet.datasets.samplers.**InfiniteBatchSampler**(*dataset*, *batch_size=1*, *world_size=None*, *rank=None*, *seed=0*, *shuffle=True*)

Similar to *BatchSampler* warping a *DistributedSampler. It is designed iteration-based runners like `IterBasedRunner* and yields a mini-batch indices each time.

The implementation logic is referred to https://github.com/facebookresearch/detectron2/blob/main/detectron2/data/samplers/grouped_batch_sampler.py

Parameters

- **dataset** (*object*) – The dataset.
- **batch_size** (*int*) – When model is `DistributedDataParallel`, it is the number of training samples on each GPU, When model is `DataParallel`, it is *num_gpus * samples_per_gpu*. Default : 1.
- **world_size** (*int, optional*) – Number of processes participating in distributed training. Default: None.
- **rank** (*int, optional*) – Rank of current process. Default: None.
- **seed** (*int*) – Random seed. Default: 0.
- **shuffle** (*bool*) – Whether shuffle the dataset or not. Default: True.

set_epoch(*epoch*)

Not supported in *IterationBased* runner.

class mmdet.datasets.samplers.**InfiniteGroupBatchSampler**(*dataset*, *batch_size=1*, *world_size=None*, *rank=None*, *seed=0*, *shuffle=True*)

Similar to *BatchSampler* warping a *GroupSampler. It is designed for iteration-based runners like `IterBasedRunner* and yields a mini-batch indices each time, all indices in a batch should be in the same group.

The implementation logic is referred to https://github.com/facebookresearch/detectron2/blob/main/detectron2/data/samplers/grouped_batch_sampler.py

Parameters

- **dataset** (*object*) – The dataset.
- **batch_size** (*int*) – When model is `DistributedDataParallel`, it is the number of training samples on each GPU. When model is `DataParallel`, it is *num_gpus * samples_per_gpu*. Default : 1.
- **world_size** (*int, optional*) – Number of processes participating in distributed training. Default: None.
- **rank** (*int, optional*) – Rank of current process. Default: None.
- **seed** (*int*) – Random seed. Default: 0.
- **shuffle** (*bool*) – Whether shuffle the indices of a dummy *epoch*, it should be noted that *shuffle* can not guarantee that you can generate sequential indices because it need to ensure that all indices in a batch is in a group. Default: True.

set_epoch(*epoch*)

Not supported in *IterationBased* runner.

39.4 api_wrappers

class mmdet.datasets.api_wrappers.**COCO**(**args: Any*, ***kwargs: Any*)
This class is almost the same as official pycocotools package.

It implements some snake case function aliases. So that the COCO class has the same interface as LVIS class.

mmdet.datasets.api_wrappers.**pq_compute_multi_core**(*matched_annotations_list*, *gt_folder*, *pred_folder*, *categories*, *file_client=None*)
Evaluate the metrics of Panoptic Segmentation with multithreading.

Same as the function with the same name in *panopticapi*.

Parameters

- **matched_annotations_list** (*list*) – The matched annotation list. Each element is a tuple of annotations of the same image with the format (gt_anns, pred_anns).
- **gt_folder** (*str*) – The path of the ground truth images.
- **pred_folder** (*str*) – The path of the prediction images.
- **categories** (*str*) – The categories of the dataset.
- **file_client** (*object*) – The file client of the dataset. If None, the backend will be set to *disk*.

mmdet.datasets.api_wrappers.**pq_compute_single_core**(*proc_id*, *annotation_set*, *gt_folder*, *pred_folder*, *categories*, *file_client=None*)
The single core function to evaluate the metric of Panoptic Segmentation.

Same as the function with the same name in *panopticapi*. Only the function to load the images is changed to use the file client.

Parameters

- **proc_id** (*int*) – The id of the mini process.
- **gt_folder** (*str*) – The path of the ground truth images.
- **pred_folder** (*str*) – The path of the prediction images.
- **categories** (*str*) – The categories of the dataset.
- **file_client** (*object*) – The file client of the dataset. If None, the backend will be set to *disk*.

CHAPTER

FORTY

MMDET.MODELS

40.1 detectors

class mmdet.models.detectors.**ATSS**(*backbone*, *neck*, *bbox_head*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

Implementation of ATSS.

class mmdet.models.detectors.**AutoAssign**(*backbone*, *neck*, *bbox_head*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*)

Implementation of AutoAssign: Differentiable Label Assignment for Dense Object Detection.

class mmdet.models.detectors.**BaseDetector**(*init_cfg=None*)

Base class for detectors.

abstract aug_test(*imgs*, *img_metas*, ***kwargs*)

Test function with test time augmentation.

abstract extract_feat(*imgs*)

Extract features from images.

extract_feats(*imgs*)

Extract features from multiple images.

Parameters **imgs** (*list[torch.Tensor]*) – A list of images. The images are augmented from the same image but in different ways.

Returns Features of different images

Return type list[torch.Tensor]

forward(*img*, *img_metas*, *return_loss=True*, ***kwargs*)

Calls either *forward_train()* or *forward_test()* depending on whether `return_loss` is `True`.

Note this setting will change the expected inputs. When `return_loss=True`, img and img_meta are single-nested (i.e. Tensor and List[dict]), and when `resturn_loss=False`, img and img_meta should be double nested (i.e. List[Tensor], List[List[dict]]), with the outer list indicating test time augmentations.

forward_test(*imgs*, *img_metas*, ***kwargs*)

Parameters

- **imgs** (*List[Tensor]*) – the outer list indicates test-time augmentations and inner Tensor should have a shape NxCxHxW, which contains all images in the batch.
- **img_metas** (*List[List[dict]]*) – the outer list indicates test-time augs (multiscale, flip, etc.) and the inner list indicates images in a batch.

forward_train(*imgs*, *img_metas*, ***kwargs*)

Parameters

- **img** (*list[Tensor]*) – List of tensors of shape (1, C, H, W). Typically these should be mean centered and std scaled.
- **img_metas** (*list[dict]*) – List of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys, see *mmdet.datasets.pipelines.Collect*.
- **kwargs** (*keyword arguments*) – Specific to concrete implementation.

show_result(*img*, *result*, *score_thr=0.3*, *bbox_color=(72, 101, 241)*, *text_color=(72, 101, 241)*, *mask_color=None*, *thickness=2*, *font_size=13*, *win_name=''*, *show=False*, *wait_time=0*, *out_file=None*)

Draw *result* over *img*.

Parameters

- **img** (*str or Tensor*) – The image to be displayed.
- **result** (*Tensor or tuple*) – The results to draw over *img* bbox_result or (bbox_result, segm_result).
- **score_thr** (*float, optional*) – Minimum score of bboxes to be shown. Default: 0.3.
- **bbox_color** (str or tuple(int) or Color) – Color of bbox lines. The tuple of color should be in BGR order. Default: 'green'
- **text_color** (str or tuple(int) or Color) – Color of texts. The tuple of color should be in BGR order. Default: 'green'
- **mask_color** (None or str or tuple(int) or Color) – Color of masks. The tuple of color should be in BGR order. Default: None
- **thickness** (*int*) – Thickness of lines. Default: 2
- **font_size** (*int*) – Font size of texts. Default: 13
- **win_name** (*str*) – The window name. Default: ''
- **wait_time** (*float*) – Value of waitKey param. Default: 0.
- **show** (*bool*) – Whether to show the image. Default: False.
- **out_file** (*str or None*) – The filename to write the image. Default: None.

Returns Only if not *show* or *out_file*

Return type img (Tensor)

train_step(*data*, *optimizer*)

The iteration step during training.

This method defines an iteration step during training, except for the back propagation and optimizer updating, which are done in an optimizer hook. Note that in some complicated cases or models, the whole process including back propagation and optimizer updating is also defined in this method, such as GAN.

Parameters

- **data** (*dict*) – The output of dataloader.

- **optimizer** (torch.optim.Optimizer | dict) – The optimizer of runner is passed to `train_step()`. This argument is unused and reserved.

Returns

It should contain at least 3 keys: `loss`, `log_vars`, `num_samples`.

- `loss` is a tensor for back propagation, which can be a weighted sum of multiple losses.
- `log_vars` contains all the variables to be sent to the logger.
- `num_samples` indicates the batch size (when the model is DDP, it means the batch size on each GPU), which is used for averaging the logs.

Return type dict

val_step(*data*, *optimizer=None*)
The iteration step during validation.

This method shares the same signature as *train_step()*, but used during val epochs. Note that the evaluation after training epochs is not implemented with this method, but an evaluation hook.

property with_bbox
whether the detector has a bbox head

Type bool

property with_mask
whether the detector has a mask head

Type bool

property with_neck
whether the detector has a neck

Type bool

property with_shared_head
whether the detector has a shared head in the RoI Head

Type bool

class mmdet.models.detectors.**CascadeRCNN**(*backbone*, *neck=None*, *rpn_head=None*, *roi_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)
Implementation of Cascade R-CNN: Delving into High Quality Object Detection

show_result(*data*, *result*, ***kwargs*)
Show prediction results of the detector.

Parameters

- **data** (*str or np.ndarray*) – Image filename or loaded image.
- **result** (*Tensor or tuple*) – The results to draw over *img* bbox_result or (bbox_result, segm_result).

Returns The image with bboxes drawn on it.

Return type np.ndarray

class mmdet.models.detectors.**CenterNet**(*backbone*, *neck*, *bbox_head*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)
Implementation of CenterNet(Objects as Points)

<https://arxiv.org/abs/1904.07850>.

aug_test(*imgs*, *img_metas*, *rescale=True*)
Augment testing of CenterNet. Aug test must have flipped image pair, and unlike CornerNet, it will perform an averaging operation on the feature map instead of detecting bbox.

Parameters

- **imgs** (*list[Tensor]*) – Augmented images.
- **img_metas** (*list[list[dict]]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **rescale** (*bool*) – If True, return boxes in original image space. Default: True.

Note: `imgs` must including flipped image pairs.

Returns

BBox results of each image and classes. The outer list corresponds to each image. The inner list corresponds to each class.

Return type list[list[np.ndarray]]

merge_aug_results(*aug_results*, *with_nms*)
Merge augmented detection bboxes and score.

Parameters

- **aug_results** (*list[list[Tensor]]*) – Det_bboxes and det_labels of each image.
- **with_nms** (*bool*) – If True, do nms before return boxes.

Returns (out_bboxes, out_labels)

Return type tuple

class mmdet.models.detectors.**CornerNet**(*backbone*, *neck*, *bbox_head*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)
CornerNet.

This detector is the implementation of the paper CornerNet: Detecting Objects as Paired Keypoints .

aug_test(*imgs*, *img_metas*, *rescale=False*)
Augment testing of CornerNet.

Parameters

- **imgs** (*list[Tensor]*) – Augmented images.
- **img_metas** (*list[list[dict]]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **rescale** (*bool*) – If True, return boxes in original image space. Default: False.

Note: `imgs` must including flipped image pairs.

Returns

BBox results of each image and classes. The outer list corresponds to each image. The inner list corresponds to each class.

Return type list[list[np.ndarray]]

merge_aug_results(*aug_results*, *img_metas*)
Merge augmented detection bboxes and score.

Parameters

- **aug_results** (*list[list[Tensor]]*) – Det_bboxes and det_labels of each image.
- **img_metas** (*list[list[dict]]*) – Meta information of each image, e.g., image size, scaling factor, etc.

Returns (bboxes, labels)

Return type tuple

class mmdet.models.detectors.**DETR**(*backbone*, *bbox_head*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)
Implementation of DETR: End-to-End Object Detection with Transformers

forward_dummy(*img*)
Used for computing network flops.

See *mmdetection/tools/analysis_tools/get_flops.py*

onnx_export(*img*, *img_metas*)
Test function for exporting to ONNX, without test time augmentation.

Parameters

- **img** (*torch.Tensor*) – input images.
- **img_metas** (*list[dict]*) – List of image information.

Returns

dets of shape [N, num_det, 5] and class labels of shape [N, num_det].

Return type tuple[Tensor, Tensor]

class mmdet.models.detectors.**DeformableDETR**(**args*, ***kwargs*)

class mmdet.models.detectors.**FCOS**(*backbone*, *neck*, *bbox_head*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)
Implementation of FCOS

class mmdet.models.detectors.**FOVEA**(*backbone*, *neck*, *bbox_head*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)
Implementation of FoveaBox

class mmdet.models.detectors.**FSAF**(*backbone*, *neck*, *bbox_head*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)
Implementation of FSAF

class mmdet.models.detectors.**FastRCNN**(*backbone*, *roi_head*, *train_cfg*, *test_cfg*, *neck=None*, *pretrained=None*, *init_cfg=None*)
Implementation of Fast R-CNN

forward_test(*imgs*, *img_metas*, *proposals*, ***kwargs*)

Parameters

- **imgs** (*List[Tensor]*) – the outer list indicates test-time augmentations and inner Tensor should have a shape NxCxHxW, which contains all images in the batch.
- **img_metas** (*List[List[dict]]*) – the outer list indicates test-time augs (multiscale, flip, etc.) and the inner list indicates images in a batch.

- **proposals** (*List[List[Tensor]]*) – the outer list indicates test-time augs (multiscale, flip, etc.) and the inner list indicates images in a batch. The Tensor should have a shape Px4, where P is the number of proposals.

class mmdet.models.detectors.**FasterRCNN**(*backbone*, *rpn_head*, *roi_head*, *train_cfg*, *test_cfg*, *neck=None*, *pretrained=None*, *init_cfg=None*)

Implementation of Faster R-CNN

class mmdet.models.detectors.**GFL**(*backbone*, *neck*, *bbox_head*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

class mmdet.models.detectors.**GridRCNN**(*backbone*, *rpn_head*, *roi_head*, *train_cfg*, *test_cfg*, *neck=None*, *pretrained=None*, *init_cfg=None*)

Grid R-CNN.

This detector is the implementation of: - Grid R-CNN (https://arxiv.org/abs/1811.12030) - Grid R-CNN Plus: Faster and Better (https://arxiv.org/abs/1906.05688)

class mmdet.models.detectors.**HybridTaskCascade**(***kwargs*)

Implementation of HTC

property **with_semantic**

whether the detector has a semantic head

Type bool

class mmdet.models.detectors.**KnowledgeDistillationSingleStageDetector**(*backbone*, *neck*, *bbox_head*, *teacher_config*, *teacher_ckpt=None*, *eval_teacher=True*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*)

Implementation of Distilling the Knowledge in a Neural Network..

Parameters

- **teacher_config** (*str | dict*) – Config file path or the config object of teacher model.
- **teacher_ckpt** (*str, optional*) – Checkpoint path of teacher model. If left as None, the model will not load any weights.

cuda(*device=None*)

Since teacher_model is registered as a plain object, it is necessary to put the teacher model to cuda when calling cuda function.

forward_train(*img*, *img_metas*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*)

Parameters

- **img** (*Tensor*) – Input images of shape (N, C, H, W). Typically these should be mean centered and std scaled.
- **img_metas** (*list[dict]*) – A List of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmdet.datasets.pipelines.Collect*.
- **gt_bboxes** (*list[Tensor]*) – Each item are the truth boxes for each image in [tl_x, tl_y, br_x, br_y] format.

- **gt_labels** (*list[Tensor]*) – Class indices corresponding to each box
- **gt_bboxes_ignore** (*None | list[Tensor]*) – Specify which bounding boxes can be ignored when computing the loss.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

train(*mode=True*)
Set the same train mode for teacher and student model.

class mmdet.models.detectors.**LAD**(*backbone*, *neck*, *bbox_head*, *teacher_backbone*, *teacher_neck*, *teacher_bbox_head*, *teacher_ckpt*, *eval_teacher=True*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*)
Implementation of LAD.

extract_teacher_feat(*img*)
Directly extract teacher features from the backbone+neck.

forward_train(*img*, *img_metas*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*)

Parameters

- **img** (*Tensor*) – Input images of shape (N, C, H, W). Typically these should be mean centered and std scaled.
- **img_metas** (*list[dict]*) – A List of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmdet.datasets.pipelines.Collect*.
- **gt_bboxes** (*list[Tensor]*) – Each item are the truth boxes for each image in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – Class indices corresponding to each box
- **gt_bboxes_ignore** (*None | list[Tensor]*) – Specify which bounding boxes can be ignored when computing the loss.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

property **with_teacher_neck**
whether the detector has a teacher_neck

Type bool

class mmdet.models.detectors.**MaskRCNN**(*backbone*, *rpn_head*, *roi_head*, *train_cfg*, *test_cfg*, *neck=None*, *pretrained=None*, *init_cfg=None*)
Implementation of Mask R-CNN

class mmdet.models.detectors.**MaskScoringRCNN**(*backbone*, *rpn_head*, *roi_head*, *train_cfg*, *test_cfg*, *neck=None*, *pretrained=None*, *init_cfg=None*)
Mask Scoring RCNN.

https://arxiv.org/abs/1903.00241

class mmdet.models.detectors.**NASFCOS**(*backbone*, *neck*, *bbox_head*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)
NAS-FCOS: Fast Neural Architecture Search for Object Detection.

https://arxiv.org/abs/1906.0442

class mmdet.models.detectors.**PAA**(*backbone*, *neck*, *bbox_head*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

Implementation of PAA.

class mmdet.models.detectors.**PanopticFPN**(*backbone*, *neck=None*, *rpn_head=None*, *roi_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*, *semantic_head=None*, *panoptic_fusion_head=None*)

Implementation of Panoptic feature pyramid networks

class mmdet.models.detectors.**PointRend**(*backbone*, *rpn_head*, *roi_head*, *train_cfg*, *test_cfg*, *neck=None*, *pretrained=None*, *init_cfg=None*)

PointRend: Image Segmentation as Rendering

This detector is the implementation of PointRend.

class mmdet.models.detectors.**QueryInst**(*backbone*, *rpn_head*, *roi_head*, *train_cfg*, *test_cfg*, *neck=None*, *pretrained=None*, *init_cfg=None*)

Implementation of Instances as Queries

class mmdet.models.detectors.**RPN**(*backbone*, *neck*, *rpn_head*, *train_cfg*, *test_cfg*, *pretrained=None*, *init_cfg=None*)

Implementation of Region Proposal Network.

aug_test(*imgs*, *img_metas*, *rescale=False*)

Test function with test time augmentation.

Parameters

- **imgs** (*list[torch.Tensor]*) – List of multiple images
- **img_metas** (*list[dict]*) – List of image information.
- **rescale** (*bool, optional*) – Whether to rescale the results. Defaults to False.

Returns proposals

Return type list[np.ndarray]

extract_feat(*img*)

Extract features.

Parameters **img** (*torch.Tensor*) – Image tensor with shape (n, c, h ,w).

Returns

Multi-level features that may have different resolutions.

Return type list[torch.Tensor]

forward_dummy(*img*)

Dummy forward function.

forward_train(*img*, *img_metas*, *gt_bboxes=None*, *gt_bboxes_ignore=None*)

Parameters

- **img** (*Tensor*) – Input images of shape (N, C, H, W). Typically these should be mean centered and std scaled.
- **img_metas** (*list[dict]*) – A List of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmdet.datasets.pipelines.Collect*.

- **gt_bboxes** (*list[Tensor]*) – Each item are the truth boxes for each image in [tl_x, tl_y, br_x, br_y] format.
- **gt_bboxes_ignore** (*None | list[Tensor]*) – Specify which bounding boxes can be ignored when computing the loss.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

show_result(*data*, *result*, *top_k=20*, ***kwargs*)
Show RPN proposals on the image.

Parameters

- **data** (*str or np.ndarray*) – Image filename or loaded image.
- **result** (*Tensor or tuple*) – The results to draw over *img* bbox_result or (bbox_result, segm_result).
- **top_k** (*int*) – Plot the first k bboxes only if set positive. Default: 20

Returns The image with bboxes drawn on it.

Return type np.ndarray

simple_test(*img*, *img_metas*, *rescale=False*)
Test function without test time augmentation.

Parameters

- **imgs** (*list[torch.Tensor]*) – List of multiple images
- **img_metas** (*list[dict]*) – List of image information.
- **rescale** (*bool, optional*) – Whether to rescale the results. Defaults to False.

Returns proposals

Return type list[np.ndarray]

class mmdet.models.detectors.**RepPointsDetector**(*backbone*, *neck*, *bbox_head*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)
RepPoints: Point Set Representation for Object Detection.

This detector is the implementation of: - RepPoints detector (https://arxiv.org/pdf/1904.11490)

class mmdet.models.detectors.**RetinaNet**(*backbone*, *neck*, *bbox_head*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)
Implementation of RetinaNet

class mmdet.models.detectors.**SCNet**(***kwargs*)
Implementation of SCNet

class mmdet.models.detectors.**SOLO**(*backbone*, *neck=None*, *bbox_head=None*, *mask_head=None*, *train_cfg=None*, *test_cfg=None*, *init_cfg=None*, *pretrained=None*)
SOLO: Segmenting Objects by Locations

class mmdet.models.detectors.**SingleStageDetector**(*backbone*, *neck=None*, *bbox_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)
Base class for single-stage detectors.

Single-stage detectors directly and densely predict bounding boxes on the output features of the backbone+neck.

aug_test(*imgs*, *img_metas*, *rescale=False*)
Test function with test time augmentation.

Parameters

- **imgs** (*list[Tensor]*) – the outer list indicates test-time augmentations and inner Tensor should have a shape NxCxHxW, which contains all images in the batch.
- **img_metas** (*list[list[dict]]*) – the outer list indicates test-time augs (multiscale, flip, etc.) and the inner list indicates images in a batch. each dict has image information.
- **rescale** (*bool, optional*) – Whether to rescale the results. Defaults to False.

Returns

BBox results of each image and classes. The outer list corresponds to each image. The inner list corresponds to each class.

Return type list[list[np.ndarray]]

extract_feat(*img*)
Directly extract features from the backbone+neck.

forward_dummy(*img*)
Used for computing network flops.

See *mmdetection/tools/analysis_tools/get_flops.py*

forward_train(*img*, *img_metas*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*)

Parameters

- **img** (*Tensor*) – Input images of shape (N, C, H, W). Typically these should be mean centered and std scaled.
- **img_metas** (*list[dict]*) – A List of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmdet.datasets.pipelines.Collect*.
- **gt_bboxes** (*list[Tensor]*) – Each item are the truth boxes for each image in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – Class indices corresponding to each box
- **gt_bboxes_ignore** (*None | list[Tensor]*) – Specify which bounding boxes can be ignored when computing the loss.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

onnx_export(*img*, *img_metas*, *with_nms=True*)
Test function without test time augmentation.

Parameters

- **img** (*torch.Tensor*) – input images.
- **img_metas** (*list[dict]*) – List of image information.

Returns

dets of shape [N, num_det, 5] and class labels of shape [N, num_det].

Return type tuple[Tensor, Tensor]

simple_test(*img*, *img_metas*, *rescale=False*)
Test function without test-time augmentation.

Parameters

- **img** (*torch.Tensor*) – Images with shape (N, C, H, W).
- **img_metas** (*list[dict]*) – List of image information.
- **rescale** (*bool, optional*) – Whether to rescale the results. Defaults to False.

Returns

BBox results of each image and classes. The outer list corresponds to each image. The inner list corresponds to each class.

Return type list[list[np.ndarray]]

class mmdet.models.detectors.**SparseRCNN**(**args*, ***kwargs*)
Implementation of Sparse R-CNN: End-to-End Object Detection with Learnable Proposals

forward_dummy(*img*)
Used for computing network flops.

See *mmdetection/tools/analysis_tools/get_flops.py*

forward_train(*img*, *img_metas*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*, *gt_masks=None*, *proposals=None*, ***kwargs*)
Forward function of SparseR-CNN and QueryInst in train stage.

Parameters

- **img** (*Tensor*) – of shape (N, C, H, W) encoding input images. Typically these should be mean centered and std scaled.
- **img_metas** (*list[dict]*) – list of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmdet.datasets.pipelines.Collect*.
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss.
- **gt_masks** (*List[Tensor], optional*) – Segmentation masks for each box. This is required to train QueryInst.
- **proposals** (*List[Tensor], optional*) – override rpn proposals with custom proposals. Use when *with_rpn* is False.

Returns a dictionary of loss components

Return type dict[str, Tensor]

simple_test(*img*, *img_metas*, *rescale=False*)
Test function without test time augmentation.

Parameters

- **imgs** (*list[torch.Tensor]*) – List of multiple images
- **img_metas** (*list[dict]*) – List of image information.

- **rescale** (*bool*) – Whether to rescale the results. Defaults to False.

Returns

BBox results of each image and classes. The outer list corresponds to each image. The inner list corresponds to each class.

Return type list[list[np.ndarray]]

class mmdet.models.detectors.**TridentFasterRCNN**(*backbone*, *rpn_head*, *roi_head*, *train_cfg*, *test_cfg*, *neck=None*, *pretrained=None*, *init_cfg=None*)

Implementation of TridentNet

aug_test(*imgs*, *img_metas*, *rescale=False*)
Test with augmentations.

If rescale is False, then returned bboxes and masks will fit the scale of imgs[0].

forward_train(*img*, *img_metas*, *gt_bboxes*, *gt_labels*, ***kwargs*)
make copies of img and gts to fit multi-branch.

simple_test(*img*, *img_metas*, *proposals=None*, *rescale=False*)
Test without augmentation.

class mmdet.models.detectors.**TwoStageDetector**(*backbone*, *neck=None*, *rpn_head=None*, *roi_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

Base class for two-stage detectors.

Two-stage detectors typically consisting of a region proposal network and a task-specific regression head.

async async_simple_test(*img*, *img_meta*, *proposals=None*, *rescale=False*)
Async test without augmentation.

aug_test(*imgs*, *img_metas*, *rescale=False*)
Test with augmentations.

If rescale is False, then returned bboxes and masks will fit the scale of imgs[0].

extract_feat(*img*)
Directly extract features from the backbone+neck.

forward_dummy(*img*)
Used for computing network flops.

See *mmdetection/tools/analysis_tools/get_flops.py*

forward_train(*img*, *img_metas*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*, *gt_masks=None*, *proposals=None*, ***kwargs*)

Parameters

- **img** (*Tensor*) – of shape (N, C, H, W) encoding input images. Typically these should be mean centered and std scaled.
- **img_metas** (*list[dict]*) – list of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmdet/datasets/pipelines/formatting.py:Collect*.
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box

- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss.
- **gt_masks** (*None | Tensor*) – true segmentation masks for each box used if the architecture supports a segmentation task.
- **proposals** – override rpn proposals with custom proposals. Use when *with_rpn* is False.

Returns a dictionary of loss components

Return type dict[str, Tensor]

simple_test(*img*, *img_metas*, *proposals=None*, *rescale=False*)
Test without augmentation.

property with_roi_head
whether the detector has a RoI head

Type bool

property with_rpn
whether the detector has RPN

Type bool

class mmdet.models.detectors.**TwoStagePanopticSegmentor**(*backbone*, *neck=None*, *rpn_head=None*, *roi_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*, *semantic_head=None*, *panoptic_fusion_head=None*)

Base class of Two-stage Panoptic Segmentor.

As well as the components in TwoStageDetector, Panoptic Segmentor has extra semantic_head and panoptic_fusion_head.

forward_dummy(*img*)
Used for computing network flops.

See *mmdetection/tools/get_flops.py*

forward_train(*img*, *img_metas*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*, *gt_masks=None*, *gt_semantic_seg=None*, *proposals=None*, ***kwargs*)

Parameters

- **img** (*Tensor*) – of shape (N, C, H, W) encoding input images. Typically these should be mean centered and std scaled.
- **img_metas** (*list[dict]*) – list of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmdet/datasets/pipelines/formatting.py:Collect*.
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss.
- **gt_masks** (*None | Tensor*) – true segmentation masks for each box used if the architecture supports a segmentation task.

- **proposals** – override rpn proposals with custom proposals. Use when *with_rpn* is False.

Returns a dictionary of loss components

Return type dict[str, Tensor]

simple_test(*img*, *img_metas*, *proposals=None*, *rescale=False*)
Test without Augmentation.

simple_test_mask(*x*, *img_metas*, *det_bboxes*, *det_labels*, *rescale=False*)
Simple test for mask head without augmentation.

class mmdet.models.detectors.**VFNet**(*backbone*, *neck*, *bbox_head*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)
Implementation of **`VarifocalNet (VFNet).<https://arxiv.org/abs/2008.13367>`_**

class mmdet.models.detectors.**YOLACT**(*backbone*, *neck*, *bbox_head*, *segm_head*, *mask_head*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)
Implementation of YOLACT

aug_test(*imgs*, *img_metas*, *rescale=False*)
Test with augmentations.

forward_dummy(*img*)
Used for computing network flops.

See *mmdetection/tools/analysis_tools/get_flops.py*

forward_train(*img*, *img_metas*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*, *gt_masks=None*)

Parameters

- **img** (*Tensor*) – of shape (N, C, H, W) encoding input images. Typically these should be mean centered and std scaled.
- **img_metas** (*list[dict]*) – list of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmdet/datasets/pipelines/formatting.py:Collect*.
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss.
- **gt_masks** (*None | Tensor*) – true segmentation masks for each box used if the architecture supports a segmentation task.

Returns a dictionary of loss components

Return type dict[str, Tensor]

simple_test(*img*, *img_metas*, *rescale=False*)
Test function without test-time augmentation.

class mmdet.models.detectors.**YOLOF**(*backbone*, *neck*, *bbox_head*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*)
Implementation of You Only Look One-level Feature

class mmdet.models.detectors.**YOLOV3**(*backbone*, *neck*, *bbox_head*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

onnx_export(*img*, *img_metas*)

Test function for exporting to ONNX, without test time augmentation.

Parameters

- **img** (*torch.Tensor*) – input images.
- **img_metas** (*list[dict]*) – List of image information.

Returns

dets of shape [N, num_det, 5] and class labels of shape [N, num_det].

Return type tuple[Tensor, Tensor]

class mmdet.models.detectors.**YOLOX**(*backbone*, *neck*, *bbox_head*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *input_size=(640, 640)*, *size_multiplier=32*, *random_size_range=(15, 25)*, *random_size_interval=10*, *init_cfg=None*)

Implementation of YOLOX: Exceeding YOLO Series in 2021

Note: Considering the trade-off between training speed and accuracy, multi-scale training is temporarily kept. More elegant implementation will be adopted in the future.

Parameters

- **backbone** (*nn.Module*) – The backbone module.
- **neck** (*nn.Module*) – The neck module.
- **bbox_head** (*nn.Module*) – The bbox head module.
- **(obj** (*test_cfg*) – *ConfigDict*, optional): The training config of YOLOX. Default: None.
- **(obj** – *ConfigDict*, optional): The testing config of YOLOX. Default: None.
- **pretrained** (*str, optional*) – model pretrained path. Default: None.
- **input_size** (*tuple*) – The model default input image size. Default: (640, 640).
- **size_multiplier** (*int*) – Image size multiplication factor. Default: 32.
- **random_size_range** (*tuple*) – The multi-scale random range during multi-scale training. The real training image size will be multiplied by size_multiplier. Default: (15, 25).
- **random_size_interval** (*int*) – The iter interval of change image size. Default: 10.
- **init_cfg** (*dict, optional*) – Initialization config dict. Default: None.

forward_train(*img*, *img_metas*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*)

Parameters

- **img** (*Tensor*) – Input images of shape (N, C, H, W). Typically these should be mean centered and std scaled.
- **img_metas** (*list[dict]*) – A List of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmdet.datasets.pipelines.Collect*.

- **gt_bboxes** (*list[Tensor]*) – Each item are the truth boxes for each image in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – Class indices corresponding to each box
- **gt_bboxes_ignore** (*None | list[Tensor]*) – Specify which bounding boxes can be ignored when computing the loss.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

40.2 backbones

class mmdet.models.backbones.**CSPDarknet**(*arch='P5'*, *deepen_factor=1.0*, *widen_factor=1.0*, *out_indices=(2, 3, 4)*, *frozen_stages=- 1*, *use_depthwise=False*, *arch_ovewrite=None*, *spp_kernal_sizes=(5, 9, 13)*, *conv_cfg=None*, *norm_cfg={'eps': 0.001, 'momentum': 0.03, 'type': 'BN'}*, *act_cfg={'type': 'Swish'}*, *norm_eval=False*, *init_cfg={'a': 2.23606797749979, 'distribution': 'uniform', 'layer': 'Conv2d', 'mode': 'fan_in', 'nonlinearity': 'leaky_relu', 'type': 'Kaiming'}*)

CSP-Darknet backbone used in YOLOv5 and YOLOX.

Parameters

- **arch** (*str*) – Architecture of CSP-Darknet, from {P5, P6}. Default: P5.
- **deepen_factor** (*float*) – Depth multiplier, multiply number of channels in each layer by this amount. Default: 1.0.
- **widen_factor** (*float*) – Width multiplier, multiply number of blocks in CSP layer by this amount. Default: 1.0.
- **out_indices** (*Sequence[int]*) – Output from which stages. Default: (2, 3, 4).
- **frozen_stages** (*int*) – Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Default: -1.
- **use_depthwise** (*bool*) – Whether to use depthwise separable convolution. Default: False.
- **arch_ovewrite** (*list*) – Overwrite default arch settings. Default: None.
- **spp_kernal_sizes** – (tuple[int]): Sequential of kernel sizes of SPP layers. Default: (5, 9, 13).
- **conv_cfg** (*dict*) – Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict*) – Dictionary to construct and config norm layer. Default: dict(type='BN', requires_grad=True).
- **act_cfg** (*dict*) – Config dict for activation layer. Default: dict(type='LeakyReLU', negative_slope=0.1).
- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None.

Example

```
>>> from mmdet.models import CSPDarknet
>>> import torch
>>> self = CSPDarknet(depth=53)
>>> self.eval()
>>> inputs = torch.rand(1, 3, 416, 416)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
...
(1, 256, 52, 52)
(1, 512, 26, 26)
(1, 1024, 13, 13)
```

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

train(*mode=True*)

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. `Dropout`, `BatchNorm`, etc.

Parameters **mode** (*bool*) – whether to set training mode (`True`) or evaluation mode (`False`). Default: `True`.

Returns self

Return type Module

class mmdet.models.backbones.**Darknet**(*depth=53*, *out_indices=(3, 4, 5)*, *frozen_stages=- 1*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *act_cfg={'negative_slope': 0.1, 'type': 'LeakyReLU'}*, *norm_eval=True*, *pretrained=None*, *init_cfg=None*)

Darknet backbone.

Parameters

- **depth** (*int*) – Depth of Darknet. Currently only support 53.
- **out_indices** (*Sequence[int]*) – Output from which stages.
- **frozen_stages** (*int*) – Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Default: -1.
- **conv_cfg** (*dict*) – Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict*) – Dictionary to construct and config norm layer. Default: dict(type='BN', requires_grad=True)
- **act_cfg** (*dict*) – Config dict for activation layer. Default: dict(type='LeakyReLU', negative_slope=0.1).

- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only.
- **pretrained** (*str, optional*) – model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

Example

```
>>> from mmdet.models import Darknet
>>> import torch
>>> self = Darknet(depth=53)
>>> self.eval()
>>> inputs = torch.rand(1, 3, 416, 416)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
...
(1, 256, 52, 52)
(1, 512, 26, 26)
(1, 1024, 13, 13)
```

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

static make_conv_res_block(*in_channels*, *out_channels*, *res_repeat*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *act_cfg={'negative_slope': 0.1, 'type': 'LeakyReLU'}*)

In Darknet backbone, ConvLayer is usually followed by ResBlock. This function will make that. The Conv layers always have 3x3 filters with stride=2. The number of the filters in Conv layer is the same as the out channels of the ResBlock.

Parameters

- **in_channels** (*int*) – The number of input channels.
- **out_channels** (*int*) – The number of output channels.
- **res_repeat** (*int*) – The number of ResBlocks.
- **conv_cfg** (*dict*) – Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict*) – Dictionary to construct and config norm layer. Default: dict(type='BN', requires_grad=True)
- **act_cfg** (*dict*) – Config dict for activation layer. Default: dict(type='LeakyReLU', negative_slope=0.1).

train(*mode=True*)

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. `Dropout`, `BatchNorm`, etc.

Parameters **mode** (*bool*) – whether to set training mode (`True`) or evaluation mode (`False`). Default: `True`.

Returns self

Return type Module

class mmdet.models.backbones.**DetectoRS_ResNeXt**(*groups=1*, *base_width=4*, ***kwargs*)

ResNeXt backbone for DetectoRS.

Parameters

- **groups** (*int*) – The number of groups in ResNeXt.
- **base_width** (*int*) – The base width of ResNeXt.

make_res_layer(***kwargs*)

Pack all blocks in a stage into a `ResLayer` for DetectoRS.

class mmdet.models.backbones.**DetectoRS_ResNet**(*sac=None*, *stage_with_sac=(False, False, False, False)*, *rfp_inplanes=None*, *output_img=False*, *pretrained=None*, *init_cfg=None*, ***kwargs*)

ResNet backbone for DetectoRS.

Parameters

- **sac** (*dict, optional*) – Dictionary to construct SAC (Switchable Atrous Convolution). Default: None.
- **stage_with_sac** (*list*) – Which stage to use sac. Default: (False, False, False, False).
- **rfp_inplanes** (*int, optional*) – The number of channels from RFP. Default: None. If specified, an additional conv layer will be added for `rfp_feat`. Otherwise, the structure is the same as base class.
- **output_img** (*bool*) – If `True`, the input image will be inserted into the starting position of output. Default: False.

forward(*x*)

Forward function.

init_weights()

Initialize the weights.

make_res_layer(***kwargs*)

Pack all blocks in a stage into a `ResLayer` for DetectoRS.

rfp_forward(*x*, *rfp_feats*)

Forward function for RFP.

class mmdet.models.backbones.**HRNet**(*extra*, *in_channels=3*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *norm_eval=True*, *with_cp=False*, *zero_init_residual=False*, *multiscale_output=True*, *pretrained=None*, *init_cfg=None*)

HRNet backbone.

High-Resolution Representations for Labeling Pixels and Regions arXiv:.

Parameters

- **extra** (*dict*) – Detailed configuration for each stage of HRNet. There must be 4 stages, the configuration for each stage must have 5 keys:

  - num_modules(int): The number of HRModule in this stage.
  - num_branches(int): The number of branches in the HRModule.
  - block(str): The type of convolution block.
  - **num_blocks(tuple): The number of blocks in each branch.** The length must be equal to num_branches.
  - **num_channels(tuple): The number of channels in each branch.** The length must be equal to num_branches.
- **in_channels** (*int*) – Number of input image channels. Default: 3.
- **conv_cfg** (*dict*) – Dictionary to construct and config conv layer.
- **norm_cfg** (*dict*) – Dictionary to construct and config norm layer.
- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: True.
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **zero_init_residual** (*bool*) – Whether to use zero init for last norm layer in resblocks to let them behave as identity. Default: False.
- **multiscale_output** (*bool*) – Whether to output multi-level features produced by multiple branches. If False, only the first level feature will be output. Default: True.
- **pretrained** (*str, optional*) – Model pretrained path. Default: None.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None.

Example

```
>>> from mmdet.models import HRNet
>>> import torch
>>> extra = dict(
>>>     stage1=dict(
>>>         num_modules=1,
>>>         num_branches=1,
>>>         block='BOTTLENECK',
>>>         num_blocks=(4, ),
>>>         num_channels=(64, )),
>>>     stage2=dict(
>>>         num_modules=1,
>>>         num_branches=2,
>>>         block='BASIC',
>>>         num_blocks=(4, 4),
>>>         num_channels=(32, 64)),
>>>     stage3=dict(
>>>         num_modules=4,
>>>         num_branches=3,
>>>         block='BASIC',
>>>         num_blocks=(4, 4, 4),
>>>         num_channels=(32, 64, 128)),
>>>     stage4=dict(
```

(continues on next page)

(continued from previous page)

```
>>>         num_modules=3,
>>>         num_branches=4,
>>>         block='BASIC',
>>>         num_blocks=(4, 4, 4, 4),
>>>         num_channels=(32, 64, 128, 256)))
>>> self = HRNet(extra, in_channels=1)
>>> self.eval()
>>> inputs = torch.rand(1, 1, 32, 32)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 32, 8, 8)
(1, 64, 4, 4)
(1, 128, 2, 2)
(1, 256, 1, 1)
```

forward(*x*)
Forward function.

property norm1
the normalization layer named "norm1"

Type nn.Module

property norm2
the normalization layer named "norm2"

Type nn.Module

train(*mode=True*)
Convert the model into training mode will keeping the normalization layer freezed.

class mmdet.models.backbones.**HourglassNet**(*downsample_times=5*, *num_stacks=2*, *stage_channels=(256, 256, 384, 384, 384, 512)*, *stage_blocks=(2, 2, 2, 2, 2, 4)*, *feat_channel=256*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *pretrained=None*, *init_cfg=None*)

HourglassNet backbone.

Stacked Hourglass Networks for Human Pose Estimation. More details can be found in the paper .

Parameters

- **downsample_times** (*int*) – Downsample times in a HourglassModule.
- **num_stacks** (*int*) – Number of HourglassModule modules stacked, 1 for Hourglass-52, 2 for Hourglass-104.
- **stage_channels** (*list[int]*) – Feature channel of each sub-module in a HourglassModule.
- **stage_blocks** (*list[int]*) – Number of sub-modules stacked in a HourglassModule.
- **feat_channel** (*int*) – Feature channel of conv after a HourglassModule.
- **norm_cfg** (*dict*) – Dictionary to construct and config norm layer.
- **pretrained** (*str, optional*) – model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

Example

```
>>> from mmdet.models import HourglassNet
>>> import torch
>>> self = HourglassNet()
>>> self.eval()
>>> inputs = torch.rand(1, 3, 511, 511)
>>> level_outputs = self.forward(inputs)
>>> for level_output in level_outputs:
...     print(tuple(level_output.shape))
(1, 256, 128, 128)
(1, 256, 128, 128)
```

forward(*x*)
Forward function.

init_weights()
Init module weights.

class mmdet.models.backbones.**MobileNetV2**(*widen_factor=1.0*, *out_indices=(1, 2, 4, 7)*, *frozen_stages=-1*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU6'}*, *norm_eval=False*, *with_cp=False*, *pretrained=None*, *init_cfg=None*)

MobileNetV2 backbone.

Parameters

- **widen_factor** (*float*) – Width multiplier, multiply number of channels in each layer by this amount. Default: 1.0.
- **out_indices** (*Sequence[int], optional*) – Output from which stages. Default: (1, 2, 4, 7).
- **frozen_stages** (*int*) – Stages to be frozen (all param fixed). Default: -1, which means not freezing any parameters.
- **conv_cfg** (*dict, optional*) – Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: dict(type='BN').
- **act_cfg** (*dict*) – Config dict for activation layer. Default: dict(type='ReLU6').
- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **pretrained** (*str, optional*) – model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

forward(*x*)
Forward function.

make_layer(*out_channels*, *num_blocks*, *stride*, *expand_ratio*)
Stack InvertedResidual blocks to build a layer for MobileNetV2.

Parameters

- **out_channels** (*int*) – out_channels of block.

- **num_blocks** (*int*) – number of blocks.
- **stride** (*int*) – stride of the first block. Default: 1
- **expand_ratio** (*int*) – Expand the number of channels of the hidden layer in InvertedResidual by this ratio. Default: 6.

train(*mode=True*)

Convert the model into training mode while keep normalization layer frozen.

class mmdet.models.backbones.**PyramidVisionTransformer**(*pretrain_img_size=224*, *in_channels=3*, *embed_dims=64*, *num_stages=4*, *num_layers=[3, 4, 6, 3]*, *num_heads=[1, 2, 5, 8]*, *patch_sizes=[4, 2, 2, 2]*, *strides=[4, 2, 2, 2]*, *paddings=[0, 0, 0, 0]*, *sr_ratios=[8, 4, 2, 1]*, *out_indices=(0, 1, 2, 3)*, *mlp_ratios=[8, 8, 4, 4]*, *qkv_bias=True*, *drop_rate=0.0*, *attn_drop_rate=0.0*, *drop_path_rate=0.1*, *use_abs_pos_embed=True*, *norm_after_stage=False*, *use_conv_ffn=False*, *act_cfg={'type': 'GELU'}*, *norm_cfg={'eps': 1e-06, 'type': 'LN'}*, *pretrained=None*, *convert_weights=True*, *init_cfg=None*)

Pyramid Vision Transformer (PVT)

Implementation of Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction without Convolutions.

Parameters

- **pretrain_img_size** (*int | tuple[int]*) – The size of input image when pretrain. Defaults: 224.
- **in_channels** (*int*) – Number of input channels. Default: 3.
- **embed_dims** (*int*) – Embedding dimension. Default: 64.
- **num_stags** (*int*) – The num of stages. Default: 4.
- **num_layers** (*Sequence[int]*) – The layer number of each transformer encode layer. Default: [3, 4, 6, 3].
- **num_heads** (*Sequence[int]*) – The attention heads of each transformer encode layer. Default: [1, 2, 5, 8].
- **patch_sizes** (*Sequence[int]*) – The patch_size of each patch embedding. Default: [4, 2, 2, 2].
- **strides** (*Sequence[int]*) – The stride of each patch embedding. Default: [4, 2, 2, 2].
- **paddings** (*Sequence[int]*) – The padding of each patch embedding. Default: [0, 0, 0, 0].
- **sr_ratios** (*Sequence[int]*) – The spatial reduction rate of each transformer encode layer. Default: [8, 4, 2, 1].
- **out_indices** (*Sequence[int] | int*) – Output from which stages. Default: (0, 1, 2, 3).
- **mlp_ratios** (*Sequence[int]*) – The ratio of the mlp hidden dim to the embedding dim of each transformer encode layer. Default: [8, 8, 4, 4].
- **qkv_bias** (*bool*) – Enable bias for qkv if True. Default: True.
- **drop_rate** (*float*) – Probability of an element to be zeroed. Default 0.0.

- **attn_drop_rate** (*float*) – The drop out rate for attention layer. Default 0.0.
- **drop_path_rate** (*float*) – stochastic depth rate. Default 0.1.
- **use_abs_pos_embed** (*bool*) – If True, add absolute position embedding to the patch embedding. Defaults: True.
- **use_conv_ffn** (*bool*) – If True, use Convolutional FFN to replace FFN. Default: False.
- **act_cfg** (*dict*) – The activation config for FFNs. Default: dict(type='GELU').
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: dict(type='LN').
- **pretrained** (*str, optional*) – model pretrained path. Default: None.
- **convert_weights** (*bool*) – The flag indicates whether the pre-trained model is from the original repo. We may need to convert some keys to make it compatible. Default: True.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

init_weights()

Initialize the weights.

class mmdet.models.backbones.**PyramidVisionTransformerV2**(***kwargs*)

Implementation of PVTv2: Improved Baselines with Pyramid Vision Transformer.

class mmdet.models.backbones.**RegNet**(*arch*, *in_channels=3*, *stem_channels=32*, *base_channels=32*, *strides=(2, 2, 2, 2)*, *dilations=(1, 1, 1, 1)*, *out_indices=(0, 1, 2, 3)*, *style='pytorch'*, *deep_stem=False*, *avg_down=False*, *frozen_stages=-1*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *norm_eval=True*, *dcn=None*, *stage_with_dcn=(False, False, False, False)*, *plugins=None*, *with_cp=False*, *zero_init_residual=True*, *pretrained=None*, *init_cfg=None*)

RegNet backbone.

More details can be found in paper .

Parameters

- **arch** (*dict*) – The parameter of RegNets.
  - w0 (int): initial width
  - wa (float): slope of width
  - wm (float): quantization parameter to quantize the width
  - depth (int): depth of the backbone
  - group_w (int): width of group
  - bot_mul (float): bottleneck ratio, i.e. expansion of bottleneck.
- **strides** (*Sequence[int]*) – Strides of the first block of each stage.

- **base_channels** (*int*) – Base channels after stem layer.
- **in_channels** (*int*) – Number of input image channels. Default: 3.
- **dilations** (*Sequence[int]*) – Dilation of each stage.
- **out_indices** (*Sequence[int]*) – Output from which stages.
- **style** (*str*) – *pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer.
- **frozen_stages** (*int*) – Stages to be frozen (all param fixed). -1 means not freezing any parameters.
- **norm_cfg** (*dict*) – dictionary to construct and config norm layer.
- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only.
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed.
- **zero_init_residual** (*bool*) – whether to use zero init for last norm layer in resblocks to let them behave as identity.
- **pretrained** (*str, optional*) – model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

Example

```
>>> from mmdet.models import RegNet
>>> import torch
>>> self = RegNet(
        arch=dict(
            w0=88,
            wa=26.31,
            wm=2.25,
            group_w=48,
            depth=25,
            bot_mul=1.0))
>>> self.eval()
>>> inputs = torch.rand(1, 3, 32, 32)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 96, 8, 8)
(1, 192, 4, 4)
(1, 432, 2, 2)
(1, 1008, 1, 1)
```

adjust_width_group(*widths*, *bottleneck_ratio*, *groups*)

Adjusts the compatibility of widths and groups.

Parameters

- **widths** (*list[int]*) – Width of each stage.
- **bottleneck_ratio** (*float*) – Bottleneck ratio.

- **groups** (*int*) – number of groups in each stage

Returns The adjusted widths and groups of each stage.

Return type tuple(list)

forward(*x*)
Forward function.

generate_regnet(*initial_width*, *width_slope*, *width_parameter*, *depth*, *divisor=8*)
Generates per block width from RegNet parameters.

Parameters

- **initial_width** (*[int]*) – Initial width of the backbone
- **width_slope** (*[float]*) – Slope of the quantized linear function
- **width_parameter** (*[int]*) – Parameter used to quantize the width.
- **depth** (*[int]*) – Depth of the backbone.
- **divisor** (*int, optional*) – The divisor of channels. Defaults to 8.

Returns return a list of widths of each stage and the number of stages

Return type list, int

get_stages_from_blocks(*widths*)
Gets widths/stage_blocks of network at each stage.

Parameters **widths** (*list[int]*) – Width in each stage.

Returns width and depth of each stage

Return type tuple(list)

static quantize_float(*number*, *divisor*)
Converts a float to closest non-zero int divisible by divisor.

Parameters

- **number** (*int*) – Original number to be quantized.
- **divisor** (*int*) – Divisor used to quantize the number.

Returns quantized number that is divisible by devisor.

Return type int

class mmdet.models.backbones.**Res2Net**(*scales=4*, *base_width=26*, *style='pytorch'*, *deep_stem=True*, *avg_down=True*, *pretrained=None*, *init_cfg=None*, ***kwargs*)
Res2Net backbone.

Parameters

- **scales** (*int*) – Scales used in Res2Net. Default: 4
- **base_width** (*int*) – Basic width of each scale. Default: 26
- **depth** (*int*) – Depth of res2net, from {50, 101, 152}.
- **in_channels** (*int*) – Number of input image channels. Default: 3.
- **num_stages** (*int*) – Res2net stages. Default: 4.
- **strides** (*Sequence[int]*) – Strides of the first block of each stage.
- **dilations** (*Sequence[int]*) – Dilation of each stage.

- **out_indices** (*Sequence[int]*) – Output from which stages.
- **style** (*str*) – *pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer.
- **deep_stem** (*bool*) – Replace 7x7 conv in input stem with 3 3x3 conv
- **avg_down** (*bool*) – Use AvgPool instead of stride conv when downsampling in the bottle2neck.
- **frozen_stages** (*int*) – Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters.
- **norm_cfg** (*dict*) – Dictionary to construct and config norm layer.
- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only.
- **plugins** (*list[dict]*) – List of plugins for stages, each dict contains:
  - cfg (dict, required): Cfg dict to build plugin.
  - position (str, required): Position inside block to insert plugin, options are 'after_conv1', 'after_conv2', 'after_conv3'.
  - stages (tuple[bool], optional): Stages to apply plugin, length should be same as 'num_stages'.
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed.
- **zero_init_residual** (*bool*) – Whether to use zero init for last norm layer in resblocks to let them behave as identity.
- **pretrained** (*str, optional*) – model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

Example

```
>>> from mmdet.models import Res2Net
>>> import torch
>>> self = Res2Net(depth=50, scales=4, base_width=26)
>>> self.eval()
>>> inputs = torch.rand(1, 3, 32, 32)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 256, 8, 8)
(1, 512, 4, 4)
(1, 1024, 2, 2)
(1, 2048, 1, 1)
```

make_res_layer(***kwargs*)
Pack all blocks in a stage into a ResLayer.

class mmdet.models.backbones.**ResNeSt**(*groups=1*, *base_width=4*, *radix=2*, *reduction_factor=4*, *avg_down_stride=True*, ***kwargs*)

ResNeSt backbone.

Parameters

- **groups** (*int*) – Number of groups of Bottleneck. Default: 1
- **base_width** (*int*) – Base width of Bottleneck. Default: 4
- **radix** (*int*) – Radix of SplitAttentionConv2d. Default: 2
- **reduction_factor** (*int*) – Reduction factor of inter_channels in SplitAttentionConv2d. Default: 4.
- **avg_down_stride** (*bool*) – Whether to use average pool for stride in Bottleneck. Default: True.
- **kwargs** (*dict*) – Keyword arguments for ResNet.

make_res_layer(***kwargs*)
Pack all blocks in a stage into a ResLayer.

class mmdet.models.backbones.**ResNeXt**(*groups=1*, *base_width=4*, ***kwargs*)
ResNeXt backbone.

Parameters

- **depth** (*int*) – Depth of resnet, from {18, 34, 50, 101, 152}.
- **in_channels** (*int*) – Number of input image channels. Default: 3.
- **num_stages** (*int*) – Resnet stages. Default: 4.
- **groups** (*int*) – Group of resnext.
- **base_width** (*int*) – Base width of resnext.
- **strides** (*Sequence[int]*) – Strides of the first block of each stage.
- **dilations** (*Sequence[int]*) – Dilation of each stage.
- **out_indices** (*Sequence[int]*) – Output from which stages.
- **style** (*str*) – *pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer.
- **frozen_stages** (*int*) – Stages to be frozen (all param fixed). -1 means not freezing any parameters.
- **norm_cfg** (*dict*) – dictionary to construct and config norm layer.
- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only.
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed.
- **zero_init_residual** (*bool*) – whether to use zero init for last norm layer in resblocks to let them behave as identity.

make_res_layer(***kwargs*)
Pack all blocks in a stage into a ResLayer

class mmdet.models.backbones.**ResNet**(*depth*, *in_channels=3*, *stem_channels=None*, *base_channels=64*, *num_stages=4*, *strides=(1, 2, 2, 2)*, *dilations=(1, 1, 1, 1)*, *out_indices=(0, 1, 2, 3)*, *style='pytorch'*, *deep_stem=False*, *avg_down=False*, *frozen_stages=- 1*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *norm_eval=True*, *dcn=None*, *stage_with_dcn=(False, False, False, False)*, *plugins=None*, *with_cp=False*, *zero_init_residual=True*, *pretrained=None*, *init_cfg=None*)

ResNet backbone.

Parameters

- **depth** (*int*) – Depth of resnet, from {18, 34, 50, 101, 152}.
- **stem_channels** (*int | None*) – Number of stem channels. If not specified, it will be the same as *base_channels*. Default: None.
- **base_channels** (*int*) – Number of base channels of res layer. Default: 64.
- **in_channels** (*int*) – Number of input image channels. Default: 3.
- **num_stages** (*int*) – Resnet stages. Default: 4.
- **strides** (*Sequence[int]*) – Strides of the first block of each stage.
- **dilations** (*Sequence[int]*) – Dilation of each stage.
- **out_indices** (*Sequence[int]*) – Output from which stages.
- **style** (*str*) – *pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer.
- **deep_stem** (*bool*) – Replace 7x7 conv in input stem with 3 3x3 conv
- **avg_down** (*bool*) – Use AvgPool instead of stride conv when downsampling in the bottleneck.
- **frozen_stages** (*int*) – Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters.
- **norm_cfg** (*dict*) – Dictionary to construct and config norm layer.
- **norm_eval** (*bool*) – Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only.
- **plugins** (*list[dict]*) – List of plugins for stages, each dict contains:
  - cfg (dict, required): Cfg dict to build plugin.
  - position (str, required): Position inside block to insert plugin, options are 'after_conv1', 'after_conv2', 'after_conv3'.
  - stages (tuple[bool], optional): Stages to apply plugin, length should be same as 'num_stages'.
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed.
- **zero_init_residual** (*bool*) – Whether to use zero init for last norm layer in resblocks to let them behave as identity.
- **pretrained** (*str, optional*) – model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

Example

```
>>> from mmdet.models import ResNet
>>> import torch
>>> self = ResNet(depth=18)
>>> self.eval()
>>> inputs = torch.rand(1, 3, 32, 32)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 64, 8, 8)
(1, 128, 4, 4)
(1, 256, 2, 2)
(1, 512, 1, 1)
```

forward(*x*)
Forward function.

make_res_layer(***kwargs*)
Pack all blocks in a stage into a `ResLayer`.

make_stage_plugins(*plugins*, *stage_idx*)
Make plugins for ResNet `stage_idx` th stage.

Currently we support to insert `context_block`, `empirical_attention_block`, `nonlocal_block` into the backbone like ResNet/ResNeXt. They could be inserted after conv1/conv2/conv3 of Bottleneck.

An example of plugins format could be:

Examples

```
>>> plugins=[
...     dict(cfg=dict(type='xxx', arg1='xxx'),
...          stages=(False, True, True, True),
...          position='after_conv2'),
...     dict(cfg=dict(type='yyy'),
...          stages=(True, True, True, True),
...          position='after_conv3'),
...     dict(cfg=dict(type='zzz', postfix='1'),
...          stages=(True, True, True, True),
...          position='after_conv3'),
...     dict(cfg=dict(type='zzz', postfix='2'),
...          stages=(True, True, True, True),
...          position='after_conv3')
... ]
>>> self = ResNet(depth=18)
>>> stage_plugins = self.make_stage_plugins(plugins, 0)
>>> assert len(stage_plugins) == 3
```

Suppose `stage_idx=0`, the structure of blocks in the stage would be:

```
conv1-> conv2->conv3->yyy->zzz1->zzz2
```

Suppose 'stage_idx=1', the structure of blocks in the stage would be:

```
conv1-> conv2->xxx->conv3->yyy->zzz1->zzz2
```

If stages is missing, the plugin would be applied to all stages.

Parameters

- **plugins** (*list[dict]*) – List of plugins cfg to build. The postfix is required if multiple same type plugins are inserted.
- **stage_idx** (*int*) – Index of stage to build

Returns Plugins for current stage

Return type list[dict]

property norm1

the normalization layer named "norm1"

Type nn.Module

train(*mode=True*)

Convert the model into training mode while keep normalization layer freezed.

class mmdet.models.backbones.**ResNetV1d**(***kwargs*)

ResNetV1d variant described in Bag of Tricks.

Compared with default ResNet(ResNetV1b), ResNetV1d replaces the 7x7 conv in the input stem with three 3x3 convs. And in the downsampling block, a 2x2 avg_pool with stride 2 is added before conv, whose stride is changed to 1.

class mmdet.models.backbones.**SSDVGG**(*depth*, *with_last_pool=False*, *ceil_mode=True*, *out_indices=(3, 4)*, *out_feature_indices=(22, 34)*, *pretrained=None*, *init_cfg=None*, *input_size=None*, *l2_norm_scale=None*)

VGG Backbone network for single-shot-detection.

Parameters

- **depth** (*int*) – Depth of vgg, from {11, 13, 16, 19}.
- **with_last_pool** (*bool*) – Whether to add a pooling layer at the last of the model
- **ceil_mode** (*bool*) – When True, will use *ceil* instead of *floor* to compute the output shape.
- **out_indices** (*Sequence[int]*) – Output from which stages.
- **out_feature_indices** (*Sequence[int]*) – Output from which feature map.
- **pretrained** (*str, optional*) – model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None
- **input_size** (*int, optional*) – Deprecated argumment. Width and height of input, from {300, 512}.
- **l2_norm_scale** (*float, optional*) – Deprecated argumment. L2 normalization layer init scale.

Example

```
>>> self = SSDVGG(input_size=300, depth=11)
>>> self.eval()
>>> inputs = torch.rand(1, 3, 300, 300)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 1024, 19, 19)
(1, 512, 10, 10)
(1, 256, 5, 5)
(1, 256, 3, 3)
(1, 256, 1, 1)
```

forward(*x*)
Forward function.

init_weights(*pretrained=None*)
Initialize the weights.

class mmdet.models.backbones.**SwinTransformer**(*pretrain_img_size=224*, *in_channels=3*, *embed_dims=96*, *patch_size=4*, *window_size=7*, *mlp_ratio=4*, *depths=(2, 2, 6, 2)*, *num_heads=(3, 6, 12, 24)*, *strides=(4, 2, 2, 2)*, *out_indices=(0, 1, 2, 3)*, *qkv_bias=True*, *qk_scale=None*, *patch_norm=True*, *drop_rate=0.0*, *attn_drop_rate=0.0*, *drop_path_rate=0.1*, *use_abs_pos_embed=False*, *act_cfg={'type': 'GELU'}*, *norm_cfg={'type': 'LN'}*, *with_cp=False*, *pretrained=None*, *convert_weights=False*, *frozen_stages=- 1*, *init_cfg=None*)

Swin Transformer A PyTorch implement of : *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows* -

https://arxiv.org/abs/2103.14030

Inspiration from https://github.com/microsoft/Swin-Transformer

Parameters

- **pretrain_img_size** (*int | tuple[int]*) – The size of input image when pretrain. Defaults: 224.
- **in_channels** (*int*) – The num of input channels. Defaults: 3.
- **embed_dims** (*int*) – The feature dimension. Default: 96.
- **patch_size** (*int | tuple[int]*) – Patch size. Default: 4.
- **window_size** (*int*) – Window size. Default: 7.
- **mlp_ratio** (*int*) – Ratio of mlp hidden dim to embedding dim. Default: 4.
- **depths** (*tuple[int]*) – Depths of each Swin Transformer stage. Default: (2, 2, 6, 2).
- **num_heads** (*tuple[int]*) – Parallel attention heads of each Swin Transformer stage. Default: (3, 6, 12, 24).
- **strides** (*tuple[int]*) – The patch merging or patch embedding stride of each Swin Transformer stage. (In swin, we set kernel size equal to stride.) Default: (4, 2, 2, 2).
- **out_indices** (*tuple[int]*) – Output from which stages. Default: (0, 1, 2, 3).

- **qkv_bias** (*bool, optional*) – If True, add a learnable bias to query, key, value. Default: True
- **qk_scale** (*float | None, optional*) – Override default qk scale of head_dim ** -0.5 if set. Default: None.
- **patch_norm** (*bool*) – If add a norm layer for patch embed and patch merging. Default: True.
- **drop_rate** (*float*) – Dropout rate. Defaults: 0.
- **attn_drop_rate** (*float*) – Attention dropout rate. Default: 0.
- **drop_path_rate** (*float*) – Stochastic depth rate. Defaults: 0.1.
- **use_abs_pos_embed** (*bool*) – If True, add absolute position embedding to the patch embedding. Defaults: False.
- **act_cfg** (*dict*) – Config dict for activation layer. Default: dict(type='LN').
- **norm_cfg** (*dict*) – Config dict for normalization layer at output of backone. Defaults: dict(type='LN').
- **with_cp** (*bool, optional*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **pretrained** (*str, optional*) – model pretrained path. Default: None.
- **convert_weights** (*bool*) – The flag indicates whether the pre-trained model is from the original repo. We may need to convert some keys to make it compatible. Default: False.
- **frozen_stages** (*int*) – Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters.
- **init_cfg** (*dict, optional*) – The Config for initialization. Defaults to None.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

init_weights()

Initialize the weights.

train(*mode=True*)

Convert the model into training mode while keep layers freezed.

class mmdet.models.backbones.**TridentResNet**(*depth*, *num_branch*, *test_branch_idx*, *trident_dilations*, ***kwargs*)

The stem layer, stage 1 and stage 2 in Trident ResNet are identical to ResNet, while in stage 3, Trident BottleBlock is utilized to replace the normal BottleBlock to yield trident output. Different branch shares the convolution weight but uses different dilations to achieve multi-scale output.

/ stage3(b0) x - stem - stage1 - stage2 - stage3(b1) - output stage3(b2) /

Parameters

- **depth** (*int*) – Depth of resnet, from {50, 101, 152}.

- **num_branch** (*int*) – Number of branches in TridentNet.
- **test_branch_idx** (*int*) – In inference, all 3 branches will be used if *test_branch_idx==-1*, otherwise only branch with index *test_branch_idx* will be used.
- **trident_dilations** (*tuple[int]*) – Dilations of different trident branch. len(trident_dilations) should be equal to num_branch.

40.3 necks

class mmdet.models.necks.**BFP**(*Balanced Feature Pyramids*)

BFP takes multi-level features as inputs and gather them into a single one, then refine the gathered feature and scatter the refined results to multi-level features. This module is used in Libra R-CNN (CVPR 2019), see the paper Libra R-CNN: Towards Balanced Learning for Object Detection for details.

Parameters

- **in_channels** (*int*) – Number of input channels (feature maps of all levels should have the same channels).
- **num_levels** (*int*) – Number of input feature levels.
- **conv_cfg** (*dict*) – The config dict for convolution layers.
- **norm_cfg** (*dict*) – The config dict for normalization layers.
- **refine_level** (*int*) – Index of integration and refine level of BSF in multi-level features from bottom to top.
- **refine_type** (*str*) – Type of the refine op, currently support [None, 'conv', 'non_local'].
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

forward(*inputs*)

Forward function.

class mmdet.models.necks.**CTResNetNeck**(*in_channel*, *num_deconv_filters*, *num_deconv_kernels*, *use_dcn=True*, *init_cfg=None*)

The neck used in CenterNet for object classification and box regression.

Parameters

- **in_channel** (*int*) – Number of input channels.
- **num_deconv_filters** (*tuple[int]*) – Number of filters per stage.
- **num_deconv_kernels** (*tuple[int]*) – Number of kernels per stage.
- **use_dcn** (*bool*) – If True, use DCNv2. Default: True.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

forward(*inputs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

init_weights()
Initialize the weights.

class mmdet.models.necks.**ChannelMapper**(*in_channels*, *out_channels*, *kernel_size=3*, *conv_cfg=None*, *norm_cfg=None*, *act_cfg={'type': 'ReLU'}*, *num_outs=None*, *init_cfg={'distribution': 'uniform', 'layer': 'Conv2d', 'type': 'Xavier'}*)
Channel Mapper to reduce/increase channels of backbone features.

This is used to reduce/increase channels of backbone features.

Parameters

- **in_channels** (*List[int]*) – Number of input channels per scale.
- **out_channels** (*int*) – Number of output channels (used at each scale).
- **kernel_size** (*int, optional*) – kernel_size for reducing channels (used at each scale). Default: 3.
- **conv_cfg** (*dict, optional*) – Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict, optional*) – Config dict for normalization layer. Default: None.
- **act_cfg** (*dict, optional*) – Config dict for activation layer in ConvModule. Default: dict(type='ReLU').
- **num_outs** (*int, optional*) – Number of output feature maps. There would be extra_convs when num_outs larger than the length of in_channels.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

Example

```
>>> import torch
>>> in_channels = [2, 3, 5, 7]
>>> scales = [340, 170, 84, 43]
>>> inputs = [torch.rand(1, c, s, s)
...           for c, s in zip(in_channels, scales)]
>>> self = ChannelMapper(in_channels, 11, 3).eval()
>>> outputs = self.forward(inputs)
>>> for i in range(len(outputs)):
...     print(f'outputs[{i}].shape = {outputs[i].shape}')
outputs[0].shape = torch.Size([1, 11, 340, 340])
outputs[1].shape = torch.Size([1, 11, 170, 170])
outputs[2].shape = torch.Size([1, 11, 84, 84])
outputs[3].shape = torch.Size([1, 11, 43, 43])
```

forward(*inputs*)
Forward function.

class mmdet.models.necks.**DilatedEncoder**(*in_channels*, *out_channels*, *block_mid_channels*, *num_residual_blocks*)
Dilated Encoder for YOLOF <https://arxiv.org/abs/2103.09460>`.

This module contains two types of components:

- **the original FPN lateral convolution layer and fpn convolution layer,** which are 1x1 conv + 3x3 conv

- the dilated residual block

Parameters

- **in_channels** (*int*) – The number of input channels.
- **out_channels** (*int*) – The number of output channels.
- **block_mid_channels** (*int*) – The number of middle block output channels
- **num_residual_blocks** (*int*) – The number of residual blocks.

forward(*feature*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.necks.**FPG**(*in_channels*, *out_channels*, *num_outs*, *stack_times*, *paths*, *inter_channels=None*, *same_down_trans=None*, *same_up_trans={'kernel_size': 3, 'padding': 1, 'stride': 2, 'type': 'conv'}*, *across_lateral_trans={'kernel_size': 1, 'type': 'conv'}*, *across_down_trans={'kernel_size': 3, 'type': 'conv'}*, *across_up_trans=None*, *across_skip_trans={'type': 'identity'}*, *output_trans={'kernel_size': 3, 'type': 'last_conv'}*, *start_level=0*, *end_level=- 1*, *add_extra_convs=False*, *norm_cfg=None*, *skip_inds=None*, *init_cfg=[{'type': 'Caffe2Xavier', 'layer': 'Conv2d'}, {'type': 'Constant', 'layer': ['_BatchNorm', '_InstanceNorm', 'GroupNorm', 'LayerNorm'], 'val': 1.0}]*)

FPG.

Implementation of Feature Pyramid Grids (FPG). This implementation only gives the basic structure stated in the paper. But users can implement different type of transitions to fully explore the the potential power of the structure of FPG.

Parameters

- **in_channels** (*int*) – Number of input channels (feature maps of all levels should have the same channels).
- **out_channels** (*int*) – Number of output channels (used at each scale)
- **num_outs** (*int*) – Number of output scales.
- **stack_times** (*int*) – The number of times the pyramid architecture will be stacked.
- **paths** (*list[str]*) – Specify the path order of each stack level. Each element in the list should be either 'bu' (bottom-up) or 'td' (top-down).
- **inter_channels** (*int*) – Number of inter channels.
- **same_up_trans** (*dict*) – Transition that goes down at the same stage.
- **same_down_trans** (*dict*) – Transition that goes up at the same stage.
- **across_lateral_trans** (*dict*) – Across-pathway same-stage
- **across_down_trans** (*dict*) – Across-pathway bottom-up connection.
- **across_up_trans** (*dict*) – Across-pathway top-down connection.

- **across_skip_trans** (*dict*) – Across-pathway skip connection.
- **output_trans** (*dict*) – Transition that trans the output of the last stage.
- **start_level** (*int*) – Index of the start input backbone level used to build the feature pyramid. Default: 0.
- **end_level** (*int*) – Index of the end input backbone level (exclusive) to build the feature pyramid. Default: -1, which means the last level.
- **add_extra_convs** (*bool*) – It decides whether to add conv layers on top of the original feature maps. Default to False. If True, its actual mode is specified by *extra_convs_on_inputs*.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: None.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

forward(*inputs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.necks.**FPN**(*in_channels*, *out_channels*, *num_outs*, *start_level=0*, *end_level=- 1*, *add_extra_convs=False*, *relu_before_extra_convs=False*, *no_norm_on_lateral=False*, *conv_cfg=None*, *norm_cfg=None*, *act_cfg=None*, *upsample_cfg={'mode': 'nearest'}*, *init_cfg={'distribution': 'uniform', 'layer': 'Conv2d', 'type': 'Xavier'}*)

Feature Pyramid Network.

This is an implementation of paper Feature Pyramid Networks for Object Detection.

Parameters

- **in_channels** (*List[int]*) – Number of input channels per scale.
- **out_channels** (*int*) – Number of output channels (used at each scale)
- **num_outs** (*int*) – Number of output scales.
- **start_level** (*int*) – Index of the start input backbone level used to build the feature pyramid. Default: 0.
- **end_level** (*int*) – Index of the end input backbone level (exclusive) to build the feature pyramid. Default: -1, which means the last level.
- **add_extra_convs** (*bool | str*) – If bool, it decides whether to add conv layers on top of the original feature maps. Default to False. If True, it is equivalent to *add_extra_convs='on_input'*. If str, it specifies the source feature map of the extra convs. Only the following options are allowed
  - 'on_input': Last feat map of neck inputs (i.e. backbone feature).
  - 'on_lateral': Last feature map after lateral convs.
  - 'on_output': The last output feature map after fpn convs.
- **relu_before_extra_convs** (*bool*) – Whether to apply relu before the extra conv. Default: False.

- **no_norm_on_lateral** (*bool*) – Whether to apply norm on lateral. Default: False.
- **conv_cfg** (*dict*) – Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: None.
- **act_cfg** (*str*) – Config dict for activation layer in ConvModule. Default: None.
- **upsample_cfg** (*dict*) – Config dict for interpolate layer. Default: *dict(mode='nearest')*
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

Example

```
>>> import torch
>>> in_channels = [2, 3, 5, 7]
>>> scales = [340, 170, 84, 43]
>>> inputs = [torch.rand(1, c, s, s)
...           for c, s in zip(in_channels, scales)]
>>> self = FPN(in_channels, 11, len(in_channels)).eval()
>>> outputs = self.forward(inputs)
>>> for i in range(len(outputs)):
...     print(f'outputs[{i}].shape = {outputs[i].shape}')
outputs[0].shape = torch.Size([1, 11, 340, 340])
outputs[1].shape = torch.Size([1, 11, 170, 170])
outputs[2].shape = torch.Size([1, 11, 84, 84])
outputs[3].shape = torch.Size([1, 11, 43, 43])
```

forward(*inputs*)
Forward function.

class mmdet.models.necks.**FPN_CARAFE**(*in_channels*, *out_channels*, *num_outs*, *start_level=0*, *end_level=- 1*, *norm_cfg=None*, *act_cfg=None*, *order=('conv', 'norm', 'act')*, *upsample_cfg={'encoder_dilation': 1, 'encoder_kernel': 3, 'type': 'carafe', 'up_group': 1, 'up_kernel': 5}*, *init_cfg=None*)

FPN_CARAFE is a more flexible implementation of FPN. It allows more choice for upsample methods during the top-down pathway.

It can reproduce the performance of ICCV 2019 paper CARAFE: Content-Aware ReAssembly of FEatures Please refer to https://arxiv.org/abs/1905.02188 for more details.

Parameters

- **in_channels** (*list[int]*) – Number of channels for each input feature map.
- **out_channels** (*int*) – Output channels of feature pyramids.
- **num_outs** (*int*) – Number of output stages.
- **start_level** (*int*) – Start level of feature pyramids. (Default: 0)
- **end_level** (*int*) – End level of feature pyramids. (Default: -1 indicates the last level).
- **norm_cfg** (*dict*) – Dictionary to construct and config norm layer.
- **activate** (*str*) – Type of activation function in ConvModule (Default: None indicates w/o activation).
- **order** (*dict*) – Order of components in ConvModule.
- **upsample** (*str*) – Type of upsample layer.

- **upsample_cfg** (*dict*) – Dictionary to construct and config upsample layer.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

forward(*inputs*)
Forward function.

init_weights()
Initialize the weights of module.

slice_as(*src*, *dst*)
Slice `src` as `dst`

Note: `src` should have the same or larger size than `dst`.

Parameters

- **src** (*torch.Tensor*) – Tensors to be sliced.
- **dst** (*torch.Tensor*) – `src` will be sliced to have the same size as `dst`.

Returns Sliced tensor.

Return type torch.Tensor

tensor_add(*a*, *b*)
Add tensors `a` and `b` that might have different sizes.

class mmdet.models.necks.**HRFPN**(*High Resolution Feature Pyramids*)
paper: High-Resolution Representations for Labeling Pixels and Regions.

Parameters

- **in_channels** (*list*) – number of channels for each branch.
- **out_channels** (*int*) – output channels of feature pyramids.
- **num_outs** (*int*) – number of output stages.
- **pooling_type** (*str*) – pooling for generating feature pyramids from {MAX, AVG}.
- **conv_cfg** (*dict*) – dictionary to construct and config conv layer.
- **norm_cfg** (*dict*) – dictionary to construct and config norm layer.
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed.
- **stride** (*int*) – stride of 3x3 convolutional layers
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

forward(*inputs*)
Forward function.

class mmdet.models.necks.**NASFCOS_FPN**(*in_channels*, *out_channels*, *num_outs*, *start_level=1*, *end_level=-1*, *add_extra_convs=False*, *conv_cfg=None*, *norm_cfg=None*, *init_cfg=None*)
FPN structure in NASFPN.

Implementation of paper NAS-FCOS: Fast Neural Architecture Search for Object Detection

Parameters

- **in_channels** (*List[int]*) – Number of input channels per scale.
- **out_channels** (*int*) – Number of output channels (used at each scale)
- **num_outs** (*int*) – Number of output scales.
- **start_level** (*int*) – Index of the start input backbone level used to build the feature pyramid. Default: 0.
- **end_level** (*int*) – Index of the end input backbone level (exclusive) to build the feature pyramid. Default: -1, which means the last level.
- **add_extra_convs** (*bool*) – It decides whether to add conv layers on top of the original feature maps. Default to False. If True, its actual mode is specified by *extra_convs_on_inputs*.
- **conv_cfg** (*dict*) – dictionary to construct and config conv layer.
- **norm_cfg** (*dict*) – dictionary to construct and config norm layer.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

forward(*inputs*)
Forward function.

init_weights()
Initialize the weights of module.

class mmdet.models.necks.**NASFPN**(*in_channels*, *out_channels*, *num_outs*, *stack_times*, *start_level=0*, *end_level=- 1*, *add_extra_convs=False*, *norm_cfg=None*, *init_cfg={'layer': 'Conv2d', 'type': 'Caffe2Xavier'}*)

NAS-FPN.

Implementation of NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection

Parameters

- **in_channels** (*List[int]*) – Number of input channels per scale.
- **out_channels** (*int*) – Number of output channels (used at each scale)
- **num_outs** (*int*) – Number of output scales.
- **stack_times** (*int*) – The number of times the pyramid architecture will be stacked.
- **start_level** (*int*) – Index of the start input backbone level used to build the feature pyramid. Default: 0.
- **end_level** (*int*) – Index of the end input backbone level (exclusive) to build the feature pyramid. Default: -1, which means the last level.
- **add_extra_convs** (*bool*) – It decides whether to add conv layers on top of the original feature maps. Default to False. If True, its actual mode is specified by *extra_convs_on_inputs*.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

forward(*inputs*)
Forward function.

class mmdet.models.necks.**PAFPN**(*in_channels*, *out_channels*, *num_outs*, *start_level=0*, *end_level=- 1*, *add_extra_convs=False*, *relu_before_extra_convs=False*, *no_norm_on_lateral=False*, *conv_cfg=None*, *norm_cfg=None*, *act_cfg=None*, *init_cfg={'distribution': 'uniform', 'layer': 'Conv2d', 'type': 'Xavier'}*)

Path Aggregation Network for Instance Segmentation.

This is an implementation of the PAFPN in Path Aggregation Network.

Parameters

- **in_channels** (*List[int]*) – Number of input channels per scale.
- **out_channels** (*int*) – Number of output channels (used at each scale)
- **num_outs** (*int*) – Number of output scales.
- **start_level** (*int*) – Index of the start input backbone level used to build the feature pyramid. Default: 0.
- **end_level** (*int*) – Index of the end input backbone level (exclusive) to build the feature pyramid. Default: -1, which means the last level.
- **add_extra_convs** (*bool | str*) – If bool, it decides whether to add conv layers on top of the original feature maps. Default to False. If True, it is equivalent to *add_extra_convs='on_input'*. If str, it specifies the source feature map of the extra convs. Only the following options are allowed
  - 'on_input': Last feat map of neck inputs (i.e. backbone feature).
  - 'on_lateral': Last feature map after lateral convs.
  - 'on_output': The last output feature map after fpn convs.
- **relu_before_extra_convs** (*bool*) – Whether to apply relu before the extra conv. Default: False.
- **no_norm_on_lateral** (*bool*) – Whether to apply norm on lateral. Default: False.
- **conv_cfg** (*dict*) – Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: None.
- **act_cfg** (*str*) – Config dict for activation layer in ConvModule. Default: None.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

forward(*inputs*)
Forward function.

class mmdet.models.necks.**RFP**(*Recursive Feature Pyramid*)
This is an implementation of RFP in DetectoRS. Different from standard FPN, the input of RFP should be multi level features along with origin input image of backbone.

Parameters

- **rfp_steps** (*int*) – Number of unrolled steps of RFP.
- **rfp_backbone** (*dict*) – Configuration of the backbone for RFP.
- **aspp_out_channels** (*int*) – Number of output channels of ASPP module.
- **aspp_dilations** (*tuple[int]*) – Dilation rates of four branches. Default: (1, 3, 6, 1)
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

forward(*inputs*)
Forward function.

init_weights()
Initialize the weights.

class mmdet.models.necks.**SSDNeck**(*in_channels*, *out_channels*, *level_strides*, *level_paddings*, *l2_norm_scale=20.0*, *last_kernel_size=3*, *use_depthwise=False*, *conv_cfg=None*, *norm_cfg=None*, *act_cfg={'type': 'ReLU'}*, *init_cfg=[{'type': 'Xavier', 'distribution': 'uniform', 'layer': 'Conv2d'}, {'type': 'Constant', 'val': 1, 'layer': 'BatchNorm2d'}]*)

Extra layers of SSD backbone to generate multi-scale feature maps.

Parameters

- **in_channels** (*Sequence[int]*) – Number of input channels per scale.
- **out_channels** (*Sequence[int]*) – Number of output channels per scale.
- **level_strides** (*Sequence[int]*) – Stride of 3x3 conv per level.
- **level_paddings** (*Sequence[int]*) – Padding size of 3x3 conv per level.
- **l2_norm_scale** (*float|None*) – L2 normalization layer init scale. If None, not use L2 normalization on the first input feature.
- **last_kernel_size** (*int*) – Kernel size of the last conv layer. Default: 3.
- **use_depthwise** (*bool*) – Whether to use DepthwiseSeparableConv. Default: False.
- **conv_cfg** (*dict*) – Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict*) – Dictionary to construct and config norm layer. Default: None.
- **act_cfg** (*dict*) – Config dict for activation layer. Default: dict(type='ReLU').
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

forward(*inputs*)

Forward function.

class mmdet.models.necks.**YOLOV3Neck**(*num_scales*, *in_channels*, *out_channels*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *act_cfg={'negative_slope': 0.1, 'type': 'LeakyReLU'}*, *init_cfg=None*)

The neck of YOLOV3.

It can be treated as a simplified version of FPN. It will take the result from Darknet backbone and do some upsampling and concatenation. It will finally output the detection result.

Note:

The input feats should be from top to bottom. i.e., from high-lvl to low-lvl

But YOLOV3Neck will process them in reversed order. i.e., from bottom (high-lvl) to top (low-lvl)

Parameters

- **num_scales** (*int*) – The number of scales / stages.
- **in_channels** (*List[int]*) – The number of input channels per scale.
- **out_channels** (*List[int]*) – The number of output channels per scale.
- **conv_cfg** (*dict, optional*) – Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict, optional*) – Dictionary to construct and config norm layer. Default: dict(type='BN', requires_grad=True)
- **act_cfg** (*dict, optional*) – Config dict for activation layer. Default: dict(type='LeakyReLU', negative_slope=0.1).

- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

forward(*feats*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.necks.**YOLOXPAFPN**(*in_channels*, *out_channels*, *num_csp_blocks=3*, *use_depthwise=False*, *upsample_cfg={'mode': 'nearest', 'scale_factor': 2}*, *conv_cfg=None*, *norm_cfg={'eps': 0.001, 'momentum': 0.03, 'type': 'BN'}*, *act_cfg={'type': 'Swish'}*, *init_cfg={'a': 2.23606797749979, 'distribution': 'uniform', 'layer': 'Conv2d', 'mode': 'fan_in', 'nonlinearity': 'leaky_relu', 'type': 'Kaiming'}*)

Path Aggregation Network used in YOLOX.

Parameters

- **in_channels** (*List[int]*) – Number of input channels per scale.
- **out_channels** (*int*) – Number of output channels (used at each scale)
- **num_csp_blocks** (*int*) – Number of bottlenecks in CSPLayer. Default: 3
- **use_depthwise** (*bool*) – Whether to depthwise separable convolution in blocks. Default: False
- **upsample_cfg** (*dict*) – Config dict for interpolate layer. Default: *dict(scale_factor=2, mode='nearest')*
- **conv_cfg** (*dict, optional*) – Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: dict(type='BN')
- **act_cfg** (*dict*) – Config dict for activation layer. Default: dict(type='Swish')
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None.

forward(*inputs*)

Parameters **inputs** (*tuple[Tensor]*) – input features.

Returns YOLOXPAFPN features.

Return type tuple[Tensor]

40.4 dense_heads

class mmdet.models.dense_heads.**ATSSHead**(*num_classes*, *in_channels*, *stacked_convs=4*, *conv_cfg=None*, *norm_cfg={'num_groups': 32, 'requires_grad': True, 'type': 'GN'}*, *reg_decoded_bbox=True*, *loss_centerness={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, *init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'atss_cls', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'}*, ***kwargs*)

Bridging the Gap Between Anchor-based and Anchor-free Detection via Adaptive Training Sample Selection.

ATSS head structure is similar with FCOS, however ATSS use anchor boxes and assign label by Adaptive Training Sample Selection instead max-iou.

https://arxiv.org/abs/1912.02424

forward(*feats*)

Forward features from the upstream network.

Parameters **feats** (*tuple[Tensor]*) – Features from the upstream network, each is a 4D-tensor.

Returns

Usually a tuple of classification scores and bbox prediction

cls_scores (list[Tensor]): Classification scores for all scale levels, each is a 4D-tensor, the channels number is num_anchors * num_classes.

bbox_preds (list[Tensor]): Box energies / deltas for all scale levels, each is a 4D-tensor, the channels number is num_anchors * 4.

Return type tuple

forward_single(*x*, *scale*)

Forward feature of a single scale level.

Parameters

- **x** (*Tensor*) – Features of a single scale level.
- **(** (*scale*) – obj: *mmcv.cnn.Scale*): Learnable scale module to resize the bbox prediction.

Returns

cls_score (Tensor): Cls scores for a single scale level the channels number is num_anchors * num_classes.

bbox_pred (Tensor): Box energies / deltas for a single scale level, the channels number is num_anchors * 4.

centerness (Tensor): Centerness for a single scale level, the channel number is (N, num_anchors * 1, H, W).

Return type tuple

get_targets(*anchor_list*, *valid_flag_list*, *gt_bboxes_list*, *img_metas*, *gt_bboxes_ignore_list=None*, *gt_labels_list=None*, *label_channels=1*, *unmap_outputs=True*)

Get targets for ATSS head.

This method is almost the same as *AnchorHead.get_targets()*. Besides returning the targets as the parent method does, it also returns the anchors as the first element of the returned tuple.

loss(*cls_scores*, *bbox_preds*, *centernesses*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)
Compute losses of the head.

Parameters

- **cls_scores** (*list[Tensor]*) – Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level with shape (N, num_anchors * 4, H, W)
- **centernesses** (*list[Tensor]*) – Centerness for each scale level with shape (N, num_anchors * 1, H, W)
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*list[Tensor] | None*) – specify which bounding boxes can be ignored when computing the loss.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

loss_single(*anchors*, *cls_score*, *bbox_pred*, *centerness*, *labels*, *label_weights*, *bbox_targets*, *num_total_samples*)
Compute loss of a single scale level.

Parameters

- **cls_score** (*Tensor*) – Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W).
- **bbox_pred** (*Tensor*) – Box energies / deltas for each scale level with shape (N, num_anchors * 4, H, W).
- **anchors** (*Tensor*) – Box reference for each scale level with shape (N, num_total_anchors, 4).
- **labels** (*Tensor*) – Labels of each anchors with shape (N, num_total_anchors).
- **label_weights** (*Tensor*) – Label weights of each anchor with shape (N, num_total_anchors)
- **bbox_targets** (*Tensor*) – BBox regression targets of each anchor weight shape (N, num_total_anchors, 4).
- **num_total_samples** (*int*) – Number os positive samples that is reduced over all GPUs.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

class mmdet.models.dense_heads.**AnchorFreeHead**(*num_classes*, *in_channels*, *feat_channels=256*, *stacked_convs=4*, *strides=(4, 8, 16, 32, 64)*, *dcn_on_last_conv=False*, *conv_bias='auto'*, *loss_cls={'alpha': 0.25, 'gamma': 2.0, 'loss_weight': 1.0, 'type': 'FocalLoss', 'use_sigmoid': True}*, *loss_bbox={'loss_weight': 1.0, 'type': 'IoULoss'}*, *bbox_coder={'type': 'DistancePointBBoxCoder'}*, *conv_cfg=None*, *norm_cfg=None*, *train_cfg=None*, *test_cfg=None*, *init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'conv_cls', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'}*)

Anchor-free head (FCOS, Fovea, RepPoints, etc.).

Parameters

- **num_classes** (*int*) – Number of categories excluding the background category.
- **in_channels** (*int*) – Number of channels in the input feature map.
- **feat_channels** (*int*) – Number of hidden channels. Used in child classes.
- **stacked_convs** (*int*) – Number of stacking convs of the head.
- **strides** (*tuple*) – Downsample factor of each feature map.
- **dcn_on_last_conv** (*bool*) – If true, use dcn in the last layer of towers. Default: False.
- **conv_bias** (*bool | str*) – If specified as *auto*, it will be decided by the norm_cfg. Bias of conv will be set as True if *norm_cfg* is None, otherwise False. Default: "auto".
- **loss_cls** (*dict*) – Config of classification loss.
- **loss_bbox** (*dict*) – Config of localization loss.
- **bbox_coder** (*dict*) – Config of bbox coder. Defaults 'DistancePointBBoxCoder'.
- **conv_cfg** (*dict*) – Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: None.
- **train_cfg** (*dict*) – Training config of anchor head.
- **test_cfg** (*dict*) – Testing config of anchor head.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

aug_test(*feats*, *img_metas*, *rescale=False*)

Test function with test time augmentation.

Parameters

- **feats** (*list[Tensor]*) – the outer list indicates test-time augmentations and inner Tensor should have a shape NxCxHxW, which contains features for all images in the batch.
- **img_metas** (*list[list[dict]]*) – the outer list indicates test-time augs (multiscale, flip, etc.) and the inner list indicates images in a batch. each dict has image information.
- **rescale** (*bool, optional*) – Whether to rescale the results. Defaults to False.

Returns bbox results of each class

Return type list[ndarray]

forward(*feats*)

Forward features from the upstream network.

Parameters **feats** (*tuple[Tensor]*) – Features from the upstream network, each is a 4D-tensor.

Returns

Usually contain classification scores and bbox predictions.

cls_scores (list[Tensor]): Box scores for each scale level, each is a 4D-tensor, the channel number is num_points * num_classes.

bbox_preds (list[Tensor]): Box energies / deltas for each scale level, each is a 4D-tensor, the channel number is num_points * 4.

Return type tuple

forward_single(*x*)
Forward features of a single scale level.

Parameters **x** (*Tensor*) – FPN feature maps of the specified stride.

Returns

Scores for each class, bbox predictions, features after classification and regression conv layers, some models needs these features like FCOS.

Return type tuple

get_points(*featmap_sizes*, *dtype*, *device*, *flatten=False*)
Get points according to feature map sizes.

Parameters

- **featmap_sizes** (*list[tuple]*) – Multi-level feature map sizes.
- **dtype** (*torch.dtype*) – Type of points.
- **device** (*torch.device*) – Device of points.

Returns points of each image.

Return type tuple

abstract get_targets(*points*, *gt_bboxes_list*, *gt_labels_list*)
Compute regression, classification and centerness targets for points in multiple images.

Parameters

- **points** (*list[Tensor]*) – Points of each fpn level, each has shape (num_points, 2).
- **gt_bboxes_list** (*list[Tensor]*) – Ground truth bboxes of each image, each has shape (num_gt, 4).
- **gt_labels_list** (*list[Tensor]*) – Ground truth labels of each box, each has shape (num_gt,).

abstract loss(*cls_scores*, *bbox_preds*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)
Compute loss of the head.

Parameters

- **cls_scores** (*list[Tensor]*) – Box scores for each scale level, each is a 4D-tensor, the channel number is num_points * num_classes.
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level, each is a 4D-tensor, the channel number is num_points * 4.

- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss.

class mmdet.models.dense_heads.**AnchorHead**(*num_classes*, *in_channels*, *feat_channels=256*, *anchor_generator={'ratios': [0.5, 1.0, 2.0], 'scales': [8, 16, 32], 'strides': [4, 8, 16, 32, 64], 'type': 'AnchorGenerator'}*, *bbox_coder={'clip_border': True, 'target_means': (0.0, 0.0, 0.0, 0.0), 'target_stds': (1.0, 1.0, 1.0, 1.0), 'type': 'DeltaXYWHBBoxCoder'}*, *reg_decoded_bbox=False*, *loss_cls={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, *loss_bbox={'beta': 0.1111111111111111, 'loss_weight': 1.0, 'type': 'SmoothL1Loss'}*, *train_cfg=None*, *test_cfg=None*, *init_cfg={'layer': 'Conv2d', 'std': 0.01, 'type': 'Normal'}*)

Anchor-based head (RPN, RetinaNet, SSD, etc.).

Parameters

- **num_classes** (*int*) – Number of categories excluding the background category.
- **in_channels** (*int*) – Number of channels in the input feature map.
- **feat_channels** (*int*) – Number of hidden channels. Used in child classes.
- **anchor_generator** (*dict*) – Config dict for anchor generator
- **bbox_coder** (*dict*) – Config of bounding box coder.
- **reg_decoded_bbox** (*bool*) – If true, the regression loss would be applied directly on decoded bounding boxes, converting both the predicted boxes and regression targets to absolute coordinates format. Default False. It should be *True* when using *IoULoss*, *GIoULoss*, or *DIoULoss* in the bbox head.
- **loss_cls** (*dict*) – Config of classification loss.
- **loss_bbox** (*dict*) – Config of localization loss.
- **train_cfg** (*dict*) – Training config of anchor head.
- **test_cfg** (*dict*) – Testing config of anchor head.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

aug_test(*feats*, *img_metas*, *rescale=False*)

Test function with test time augmentation.

Parameters

- **feats** (*list[Tensor]*) – the outer list indicates test-time augmentations and inner Tensor should have a shape NxCxHxW, which contains features for all images in the batch.
- **img_metas** (*list[list[dict]]*) – the outer list indicates test-time augs (multiscale, flip, etc.) and the inner list indicates images in a batch. each dict has image information.
- **rescale** (*bool, optional*) – Whether to rescale the results. Defaults to False.

Returns

Each item in result_list is 2-tuple. The first item is `bboxes` with shape (n, 5), where 5 represent (tl_x, tl_y, br_x, br_y, score). The shape of the second tensor in the tuple is `labels` with shape (n,), The length of list should always be 1.

Return type list[tuple[Tensor, Tensor]]

`forward`(*feats*)

Forward features from the upstream network.

Parameters **feats** (*tuple[Tensor]*) – Features from the upstream network, each is a 4D-tensor.

Returns

A tuple of classification scores and bbox prediction.

- cls_scores (list[Tensor]): Classification scores for all scale levels, each is a 4D-tensor, the channels number is num_base_priors * num_classes.
- bbox_preds (list[Tensor]): Box energies / deltas for all scale levels, each is a 4D-tensor, the channels number is num_base_priors * 4.

Return type tuple

`forward_single`(*x*)

Forward feature of a single scale level.

Parameters **x** (*Tensor*) – Features of a single scale level.

Returns cls_score (Tensor): Cls scores for a single scale level the channels number is num_base_priors * num_classes. bbox_pred (Tensor): Box energies / deltas for a single scale level, the channels number is num_base_priors * 4.

Return type tuple

`get_anchors`(*featmap_sizes*, *img_metas*, *device='cuda'*)

Get anchors according to feature map sizes.

Parameters

- **featmap_sizes** (*list[tuple]*) – Multi-level feature map sizes.
- **img_metas** (*list[dict]*) – Image meta info.
- **device** (*torch.device | str*) – Device for returned tensors

Returns anchor_list (list[Tensor]): Anchors of each image. valid_flag_list (list[Tensor]): Valid flags of each image.

Return type tuple

`get_targets`(*anchor_list*, *valid_flag_list*, *gt_bboxes_list*, *img_metas*, *gt_bboxes_ignore_list=None*, *gt_labels_list=None*, *label_channels=1*, *unmap_outputs=True*, *return_sampling_results=False*)

Compute regression and classification targets for anchors in multiple images.

Parameters

- **anchor_list** (*list[list[Tensor]]*) – Multi level anchors of each image. The outer list indicates images, and the inner list corresponds to feature levels of the image. Each element of the inner list is a tensor of shape (num_anchors, 4).
- **valid_flag_list** (*list[list[Tensor]]*) – Multi level valid flags of each image. The outer list indicates images, and the inner list corresponds to feature levels of the image. Each element of the inner list is a tensor of shape (num_anchors,)

- **gt_bboxes_list** (*list[Tensor]*) – Ground truth bboxes of each image.
- **img_metas** (*list[dict]*) – Meta info of each image.
- **gt_bboxes_ignore_list** (*list[Tensor]*) – Ground truth bboxes to be ignored.
- **gt_labels_list** (*list[Tensor]*) – Ground truth labels of each box.
- **label_channels** (*int*) – Channel of label.
- **unmap_outputs** (*bool*) – Whether to map outputs back to the original set of anchors.

Returns

Usually returns a tuple containing learning targets.

- labels_list (list[Tensor]): Labels of each level.
- label_weights_list (list[Tensor]): Label weights of each level.
- bbox_targets_list (list[Tensor]): BBox targets of each level.
- bbox_weights_list (list[Tensor]): BBox weights of each level.
- num_total_pos (int): Number of positive samples in all images.
- num_total_neg (int): Number of negative samples in all images.

additional_returns: This function enables user-defined returns from *self._get_targets_single*. These returns are currently refined to properties at each feature map (i.e. having HxW dimension). The results will be concatenated after the end

Return type tuple

loss(*cls_scores*, *bbox_preds*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)

Compute losses of the head.

Parameters

- **cls_scores** (*list[Tensor]*) – Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level with shape (N, num_anchors * 4, H, W)
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss. Default: None

Returns A dictionary of loss components.

Return type dict[str, Tensor]

loss_single(*cls_score*, *bbox_pred*, *anchors*, *labels*, *label_weights*, *bbox_targets*, *bbox_weights*, *num_total_samples*)

Compute loss of a single scale level.

Parameters

- **cls_score** (*Tensor*) – Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W).
- **bbox_pred** (*Tensor*) – Box energies / deltas for each scale level with shape (N, num_anchors * 4, H, W).
- **anchors** (*Tensor*) – Box reference for each scale level with shape (N, num_total_anchors, 4).
- **labels** (*Tensor*) – Labels of each anchors with shape (N, num_total_anchors).
- **label_weights** (*Tensor*) – Label weights of each anchor with shape (N, num_total_anchors)
- **bbox_targets** (*Tensor*) – BBox regression targets of each anchor weight shape (N, num_total_anchors, 4).
- **bbox_weights** (*Tensor*) – BBox regression loss weights of each anchor with shape (N, num_total_anchors, 4).
- **num_total_samples** (*int*) – If sampling, num total samples equal to the number of total anchors; Otherwise, it is the number of positive anchors.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

class mmdet.models.dense_heads.**AutoAssignHead**(**args*, *force_topk=False*, *topk=9*, *pos_loss_weight=0.25*, *neg_loss_weight=0.75*, *center_loss_weight=0.75*, ***kwargs*)

AutoAssignHead head used in AutoAssign.

More details can be found in the paper .

Parameters

- **force_topk** (*bool*) – Used in center prior initialization to handle extremely small gt. Default is False.
- **topk** (*int*) – The number of points used to calculate the center prior when no point falls in gt_bbox. Only work when force_topk if True. Defaults to 9.
- **pos_loss_weight** (*float*) – The loss weight of positive loss and with default value 0.25.
- **neg_loss_weight** (*float*) – The loss weight of negative loss and with default value 0.75.
- **center_loss_weight** (*float*) – The loss weight of center prior loss and with default value 0.75.

forward_single(*x*, *scale*, *stride*)

Forward features of a single scale level.

Parameters

- **x** (*Tensor*) – FPN feature maps of the specified stride.
- **(** (*scale*) – obj: *mmcv.cnn.Scale*): Learnable scale module to resize the bbox prediction.
- **stride** (*int*) – The corresponding stride for feature maps, only used to normalize the bbox prediction when self.norm_on_bbox is True.

Returns scores for each class, bbox predictions and centerness predictions of input feature maps.

Return type tuple

get_neg_loss_single(*cls_score*, *objectness*, *gt_labels*, *ious*, *inside_gt_bbox_mask*)
Calculate the negative loss of all points in feature map.

Parameters

- **cls_score** (*Tensor*) – All category scores for each point on the feature map. The shape is (num_points, num_class).
- **objectness** (*Tensor*) – Foreground probability of all points and is shape of (num_points, 1).
- **gt_labels** (*Tensor*) – The zeros based label of all gt with shape of (num_gt).
- **ious** (*Tensor*) – Float tensor with shape of (num_points, num_gt). Each value represent the iou of pred_bbox and gt_bboxes.
- **inside_gt_bbox_mask** (*Tensor*) – Tensor of bool type, with shape of (num_points, num_gt), each value is used to mark whether this point falls within a certain gt.

Returns

- neg_loss (Tensor): The negative loss of all points in the feature map.

Return type tuple[Tensor]

get_pos_loss_single(*cls_score*, *objectness*, *reg_loss*, *gt_labels*, *center_prior_weights*)
Calculate the positive loss of all points in gt_bboxes.

Parameters

- **cls_score** (*Tensor*) – All category scores for each point on the feature map. The shape is (num_points, num_class).
- **objectness** (*Tensor*) – Foreground probability of all points, has shape (num_points, 1).
- **reg_loss** (*Tensor*) – The regression loss of each gt_bbox and each prediction box, has shape of (num_points, num_gt).
- **gt_labels** (*Tensor*) – The zeros based gt_labels of all gt with shape of (num_gt,).
- **center_prior_weights** (*Tensor*) – Float tensor with shape of (num_points, num_gt). Each value represents the center weighting coefficient.

Returns

- pos_loss (Tensor): The positive loss of all points in the gt_bboxes.

Return type tuple[Tensor]

get_targets(*points*, *gt_bboxes_list*)
Compute regression targets and each point inside or outside gt_bbox in multiple images.

Parameters

- **points** (*list[Tensor]*) – Points of all fpn level, each has shape (num_points, 2).
- **gt_bboxes_list** (*list[Tensor]*) – Ground truth bboxes of each image, each has shape (num_gt, 4).

Returns

- inside_gt_bbox_mask_list (list[Tensor]): Each Tensor is with bool type and shape of (num_points, num_gt), each value is used to mark whether this point falls within a certain gt.
- concat_lvl_bbox_targets (list[Tensor]): BBox targets of each level. Each tensor has shape (num_points, num_gt, 4).

Return type tuple(list[Tensor])

init_weights()
Initialize weights of the head.

In particular, we have special initialization for classified conv's and regression conv's bias

loss(*cls_scores*, *bbox_preds*, *objectnesses*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)
Compute loss of the head.

Parameters

- **cls_scores** (*list[Tensor]*) – Box scores for each scale level, each is a 4D-tensor, the channel number is num_points * num_classes.
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level, each is a 4D-tensor, the channel number is num_points * 4.
- **objectnesses** (*list[Tensor]*) – objectness for each scale level, each is a 4D-tensor, the channel number is num_points * 1.
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

class mmdet.models.dense_heads.**CascadeRPNHead**(*num_stages*, *stages*, *train_cfg*, *test_cfg*, *init_cfg=None*)
The CascadeRPNHead will predict more accurate region proposals, which is required for two-stage detectors (such as Fast/Faster R-CNN). CascadeRPN consists of a sequence of RPNStage to progressively improve the accuracy of the detected proposals.

More details can be found in https://arxiv.org/abs/1909.06720.

Parameters

- **num_stages** (*int*) – number of CascadeRPN stages.
- **stages** (*list[dict]*) – list of configs to build the stages.
- **train_cfg** (*list[dict]*) – list of configs at training time each stage.
- **test_cfg** (*dict*) – config at testing time.

aug_test_rpn(*x*, *img_metas*)
Augmented forward test function.

forward_train(*x*, *img_metas*, *gt_bboxes*, *gt_labels=None*, *gt_bboxes_ignore=None*, *proposal_cfg=None*)
Forward train function.

get_bboxes()
get_bboxes() is implemented in StageCascadeRPNHead.

loss()
loss() is implemented in StageCascadeRPNHead.

simple_test_rpn(*x*, *img_metas*)
Simple forward test function.

class mmdet.models.dense_heads.**CenterNetHead**(*in_channel*, *feat_channel*, *num_classes*, *loss_center_heatmap={'loss_weight': 1.0, 'type': 'GaussianFocalLoss'}*, *loss_wh={'loss_weight': 0.1, 'type': 'L1Loss'}*, *loss_offset={'loss_weight': 1.0, 'type': 'L1Loss'}*, *train_cfg=None*, *test_cfg=None*, *init_cfg=None*)
Objects as Points Head. CenterHead use center_point to indicate object's position. Paper link <https://arxiv.org/abs/1904.07850>

Parameters

- **in_channel** (*int*) – Number of channel in the input feature map.
- **feat_channel** (*int*) – Number of channel in the intermediate feature map.
- **num_classes** (*int*) – Number of categories excluding the background category.
- **loss_center_heatmap** (*dict | None*) – Config of center heatmap loss. Default: GaussianFocalLoss.
- **loss_wh** (*dict | None*) – Config of wh loss. Default: L1Loss.
- **loss_offset** (*dict | None*) – Config of offset loss. Default: L1Loss.
- **train_cfg** (*dict | None*) – Training config. Useless in CenterNet, but we keep this variable for SingleStageDetector. Default: None.
- **test_cfg** (*dict | None*) – Testing config of CenterNet. Default: None.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

decode_heatmap(*center_heatmap_pred*, *wh_pred*, *offset_pred*, *img_shape*, *k=100*, *kernel=3*)
Transform outputs into detections raw bbox prediction.

Parameters

- **center_heatmap_pred** (*Tensor*) – center predict heatmap, shape (B, num_classes, H, W).
- **wh_pred** (*Tensor*) – wh predict, shape (B, 2, H, W).
- **offset_pred** (*Tensor*) – offset predict, shape (B, 2, H, W).
- **img_shape** (*list[int]*) – image shape in [h, w] format.
- **k** (*int*) – Get top k center keypoints from heatmap. Default 100.
- **kernel** (*int*) – Max pooling kernel for extract local maximum pixels. Default 3.

Returns

Decoded output of CenterNetHead, containing

the following Tensors:

- batch_bboxes (Tensor): Coords of each box with shape (B, k, 5)
- batch_topk_labels (Tensor): Categories of each box with shape (B, k)

Return type tuple[torch.Tensor]

forward(*feats*)
Forward features. Notice CenterNet head does not use FPN.

Parameters **feats** (*tuple[Tensor]*) – Features from the upstream network, each is a 4D-tensor.

Returns

center predict heatmaps for all levels, the channels number is num_classes.

wh_preds (List[Tensor]): wh predicts for all levels, the channels number is 2.

offset_preds (List[Tensor]): offset predicts for all levels, the channels number is 2.

Return type center_heatmap_preds (List[Tensor])

forward_single(*feat*)
Forward feature of a single level.

Parameters **feat** (*Tensor*) – Feature of a single level.

Returns

center predict heatmaps, the channels number is num_classes.

wh_pred (Tensor): wh predicts, the channels number is 2. offset_pred (Tensor): offset predicts, the channels number is 2.

Return type center_heatmap_pred (Tensor)

get_bboxes(*center_heatmap_preds*, *wh_preds*, *offset_preds*, *img_metas*, *rescale=True*, *with_nms=False*)
Transform network output for a batch into bbox predictions.

Parameters

- **center_heatmap_preds** (*list[Tensor]*) – Center predict heatmaps for all levels with shape (B, num_classes, H, W).
- **wh_preds** (*list[Tensor]*) – WH predicts for all levels with shape (B, 2, H, W).
- **offset_preds** (*list[Tensor]*) – Offset predicts for all levels with shape (B, 2, H, W).
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **rescale** (*bool*) – If True, return boxes in original image space. Default: True.
- **with_nms** (*bool*) – If True, do nms before return boxes. Default: False.

Returns

Each item in result_list is 2-tuple. The first item is an (n, 5) tensor, where 5 represent (tl_x, tl_y, br_x, br_y, score) and the score between 0 and 1. The shape of the second tensor in the tuple is (n,), and each element represents the class label of the corresponding box.

Return type list[tuple[Tensor, Tensor]]

get_targets(*gt_bboxes*, *gt_labels*, *feat_shape*, *img_shape*)
Compute regression and classification targets in multiple images.

Parameters

- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box.
- **feat_shape** (*list[int]*) – feature map shape with value [B, _, H, W]
- **img_shape** (*list[int]*) – image shape in [h, w] format.

Returns

The float value is mean avg_factor, the dict has components below: - center_heatmap_target (Tensor): targets of center heatmap, shape (B, num_classes, H, W). - wh_target (Tensor): targets of wh predict, shape (B, 2, H, W). - offset_target (Tensor): targets of offset predict, shape (B, 2, H, W). - wh_offset_target_weight (Tensor): weights of wh and offset predict, shape (B, 2, H, W).

Return type tuple[dict,float]

`init_weights()`
Initialize weights of the head.

`loss(`*center_heatmap_preds*, *wh_preds*, *offset_preds*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*`)`
Compute losses of the head.

Parameters

- **center_heatmap_preds** (*list[Tensor]*) – center predict heatmaps for all levels with shape (B, num_classes, H, W).
- **wh_preds** (*list[Tensor]*) – wh predicts for all levels with shape (B, 2, H, W).
- **offset_preds** (*list[Tensor]*) – offset predicts for all levels with shape (B, 2, H, W).
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box.
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss. Default: None

Returns

which has components below:

- loss_center_heatmap (Tensor): loss of center heatmap.
- loss_wh (Tensor): loss of hw heatmap
- loss_offset (Tensor): loss of offset heatmap.

Return type dict[str, Tensor]

`class mmdet.models.dense_heads.CentripetalHead(`**args*, *centripetal_shift_channels=2*, *guiding_shift_channels=2*, *feat_adaption_conv_kernel=3*, *loss_guiding_shift={'beta': 1.0, 'loss_weight': 0.05, 'type': 'SmoothL1Loss'}*, *loss_centripetal_shift={'beta': 1.0, 'loss_weight': 1, 'type': 'SmoothL1Loss'}*, *init_cfg=None*, ***kwargs*`)`
Head of CentripetalNet: Pursuing High-quality Keypoint Pairs for Object Detection.

CentripetalHead inherits from *CornerHead*. It removes the embedding branch and adds guiding shift and centripetal shift branches. More details can be found in the paper .

Parameters

- **num_classes** (*int*) – Number of categories excluding the background category.

- **in_channels** (*int*) – Number of channels in the input feature map.
- **num_feat_levels** (*int*) – Levels of feature from the previous module. 2 for HourglassNet-104 and 1 for HourglassNet-52. HourglassNet-104 outputs the final feature and intermediate supervision feature and HourglassNet-52 only outputs the final feature. Default: 2.
- **corner_emb_channels** (*int*) – Channel of embedding vector. Default: 1.
- **train_cfg** (*dict | None*) – Training config. Useless in CornerHead, but we keep this variable for SingleStageDetector. Default: None.
- **test_cfg** (*dict | None*) – Testing config of CornerHead. Default: None.
- **loss_heatmap** (*dict | None*) – Config of corner heatmap loss. Default: GaussianFocalLoss.
- **loss_embedding** (*dict | None*) – Config of corner embedding loss. Default: AssociativeEmbeddingLoss.
- **loss_offset** (*dict | None*) – Config of corner offset loss. Default: SmoothL1Loss.
- **loss_guiding_shift** (*dict*) – Config of guiding shift loss. Default: SmoothL1Loss.
- **loss_centripetal_shift** (*dict*) – Config of centripetal shift loss. Default: SmoothL1Loss.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

forward_single(*x*, *lvl_ind*)

Forward feature of a single level.

Parameters

- **x** (*Tensor*) – Feature of a single level.
- **lvl_ind** (*int*) – Level index of current feature.

Returns

A tuple of CentripetalHead's output for current feature level. Containing the following Tensors:

- tl_heat (Tensor): Predicted top-left corner heatmap.
- br_heat (Tensor): Predicted bottom-right corner heatmap.
- tl_off (Tensor): Predicted top-left offset heatmap.
- br_off (Tensor): Predicted bottom-right offset heatmap.
- tl_guiding_shift (Tensor): Predicted top-left guiding shift heatmap.
- br_guiding_shift (Tensor): Predicted bottom-right guiding shift heatmap.
- tl_centripetal_shift (Tensor): Predicted top-left centripetal shift heatmap.
- br_centripetal_shift (Tensor): Predicted bottom-right centripetal shift heatmap.

Return type tuple[Tensor]

get_bboxes(*tl_heats*, *br_heats*, *tl_offs*, *br_offs*, *tl_guiding_shifts*, *br_guiding_shifts*, *tl_centripetal_shifts*, *br_centripetal_shifts*, *img_metas*, *rescale=False*, *with_nms=True*)

Transform network output for a batch into bbox predictions.

Parameters

- **tl_heats** (*list[Tensor]*) – Top-left corner heatmaps for each level with shape (N, num_classes, H, W).

- **br_heats** (*list[Tensor]*) – Bottom-right corner heatmaps for each level with shape (N, num_classes, H, W).
- **tl_offs** (*list[Tensor]*) – Top-left corner offsets for each level with shape (N, corner_offset_channels, H, W).
- **br_offs** (*list[Tensor]*) – Bottom-right corner offsets for each level with shape (N, corner_offset_channels, H, W).
- **tl_guiding_shifts** (*list[Tensor]*) – Top-left guiding shifts for each level with shape (N, guiding_shift_channels, H, W). Useless in this function, we keep this arg because it's the raw output from CentripetalHead.
- **br_guiding_shifts** (*list[Tensor]*) – Bottom-right guiding shifts for each level with shape (N, guiding_shift_channels, H, W). Useless in this function, we keep this arg because it's the raw output from CentripetalHead.
- **tl_centripetal_shifts** (*list[Tensor]*) – Top-left centripetal shifts for each level with shape (N, centripetal_shift_channels, H, W).
- **br_centripetal_shifts** (*list[Tensor]*) – Bottom-right centripetal shifts for each level with shape (N, centripetal_shift_channels, H, W).
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **rescale** (*bool*) – If True, return boxes in original image space. Default: False.
- **with_nms** (*bool*) – If True, do nms before return boxes. Default: True.

init_weights()
Initialize the weights.

loss(*tl_heats*, *br_heats*, *tl_offs*, *br_offs*, *tl_guiding_shifts*, *br_guiding_shifts*, *tl_centripetal_shifts*, *br_centripetal_shifts*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)
Compute losses of the head.

Parameters

- **tl_heats** (*list[Tensor]*) – Top-left corner heatmaps for each level with shape (N, num_classes, H, W).
- **br_heats** (*list[Tensor]*) – Bottom-right corner heatmaps for each level with shape (N, num_classes, H, W).
- **tl_offs** (*list[Tensor]*) – Top-left corner offsets for each level with shape (N, corner_offset_channels, H, W).
- **br_offs** (*list[Tensor]*) – Bottom-right corner offsets for each level with shape (N, corner_offset_channels, H, W).
- **tl_guiding_shifts** (*list[Tensor]*) – Top-left guiding shifts for each level with shape (N, guiding_shift_channels, H, W).
- **br_guiding_shifts** (*list[Tensor]*) – Bottom-right guiding shifts for each level with shape (N, guiding_shift_channels, H, W).
- **tl_centripetal_shifts** (*list[Tensor]*) – Top-left centripetal shifts for each level with shape (N, centripetal_shift_channels, H, W).
- **br_centripetal_shifts** (*list[Tensor]*) – Bottom-right centripetal shifts for each level with shape (N, centripetal_shift_channels, H, W).

- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [left, top, right, bottom] format.
- **gt_labels** (*list[Tensor]*) – Class indices corresponding to each box.
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*list[Tensor] | None*) – Specify which bounding boxes can be ignored when computing the loss.

Returns

A dictionary of loss components. Containing the following losses:

- det_loss (list[Tensor]): Corner keypoint losses of all feature levels.
- off_loss (list[Tensor]): Corner offset losses of all feature levels.
- guiding_loss (list[Tensor]): Guiding shift losses of all feature levels.
- centripetal_loss (list[Tensor]): Centripetal shift losses of all feature levels.

Return type dict[str, Tensor]

loss_single(*tl_hmp*, *br_hmp*, *tl_off*, *br_off*, *tl_guiding_shift*, *br_guiding_shift*, *tl_centripetal_shift*, *br_centripetal_shift*, *targets*)

Compute losses for single level.

Parameters

- **tl_hmp** (*Tensor*) – Top-left corner heatmap for current level with shape (N, num_classes, H, W).
- **br_hmp** (*Tensor*) – Bottom-right corner heatmap for current level with shape (N, num_classes, H, W).
- **tl_off** (*Tensor*) – Top-left corner offset for current level with shape (N, corner_offset_channels, H, W).
- **br_off** (*Tensor*) – Bottom-right corner offset for current level with shape (N, corner_offset_channels, H, W).
- **tl_guiding_shift** (*Tensor*) – Top-left guiding shift for current level with shape (N, guiding_shift_channels, H, W).
- **br_guiding_shift** (*Tensor*) – Bottom-right guiding shift for current level with shape (N, guiding_shift_channels, H, W).
- **tl_centripetal_shift** (*Tensor*) – Top-left centripetal shift for current level with shape (N, centripetal_shift_channels, H, W).
- **br_centripetal_shift** (*Tensor*) – Bottom-right centripetal shift for current level with shape (N, centripetal_shift_channels, H, W).
- **targets** (*dict*) – Corner target generated by *get_targets*.

Returns

Losses of the head's different branches containing the following losses:

- det_loss (Tensor): Corner keypoint loss.
- off_loss (Tensor): Corner offset loss.
- guiding_loss (Tensor): Guiding shift loss.

- centripetal_loss (Tensor): Centripetal shift loss.

Return type tuple[torch.Tensor]

class mmdet.models.dense_heads.**CornerHead**(*num_classes*, *in_channels*, *num_feat_levels=2*, *corner_emb_channels=1*, *train_cfg=None*, *test_cfg=None*, *loss_heatmap={'alpha': 2.0, 'gamma': 4.0, 'loss_weight': 1, 'type': 'GaussianFocalLoss'}*, *loss_embedding={'pull_weight': 0.25, 'push_weight': 0.25, 'type': 'AssociativeEmbeddingLoss'}*, *loss_offset={'beta': 1.0, 'loss_weight': 1, 'type': 'SmoothL1Loss'}*, *init_cfg=None*)

Head of CornerNet: Detecting Objects as Paired Keypoints.

Code is modified from the official github repo .

More details can be found in the paper .

Parameters

- **num_classes** (*int*) – Number of categories excluding the background category.
- **in_channels** (*int*) – Number of channels in the input feature map.
- **num_feat_levels** (*int*) – Levels of feature from the previous module. 2 for HourglassNet-104 and 1 for HourglassNet-52. Because HourglassNet-104 outputs the final feature and intermediate supervision feature and HourglassNet-52 only outputs the final feature. Default: 2.
- **corner_emb_channels** (*int*) – Channel of embedding vector. Default: 1.
- **train_cfg** (*dict | None*) – Training config. Useless in CornerHead, but we keep this variable for SingleStageDetector. Default: None.
- **test_cfg** (*dict | None*) – Testing config of CornerHead. Default: None.
- **loss_heatmap** (*dict | None*) – Config of corner heatmap loss. Default: GaussianFocalLoss.
- **loss_embedding** (*dict | None*) – Config of corner embedding loss. Default: AssociativeEmbeddingLoss.
- **loss_offset** (*dict | None*) – Config of corner offset loss. Default: SmoothL1Loss.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

decode_heatmap(*tl_heat*, *br_heat*, *tl_off*, *br_off*, *tl_emb=None*, *br_emb=None*, *tl_centripetal_shift=None*, *br_centripetal_shift=None*, *img_meta=None*, *k=100*, *kernel=3*, *distance_threshold=0.5*, *num_dets=1000*)

Transform outputs for a single batch item into raw bbox predictions.

Parameters

- **tl_heat** (*Tensor*) – Top-left corner heatmap for current level with shape (N, num_classes, H, W).
- **br_heat** (*Tensor*) – Bottom-right corner heatmap for current level with shape (N, num_classes, H, W).
- **tl_off** (*Tensor*) – Top-left corner offset for current level with shape (N, corner_offset_channels, H, W).
- **br_off** (*Tensor*) – Bottom-right corner offset for current level with shape (N, corner_offset_channels, H, W).

- **tl_emb** (*Tensor | None*) – Top-left corner embedding for current level with shape (N, corner_emb_channels, H, W).
- **br_emb** (*Tensor | None*) – Bottom-right corner embedding for current level with shape (N, corner_emb_channels, H, W).
- **tl_centripetal_shift** (*Tensor | None*) – Top-left centripetal shift for current level with shape (N, 2, H, W).
- **br_centripetal_shift** (*Tensor | None*) – Bottom-right centripetal shift for current level with shape (N, 2, H, W).
- **img_meta** (*dict*) – Meta information of current image, e.g., image size, scaling factor, etc.
- **k** (*int*) – Get top k corner keypoints from heatmap.
- **kernel** (*int*) – Max pooling kernel for extract local maximum pixels.
- **distance_threshold** (*float*) – Distance threshold. Top-left and bottom-right corner keypoints with feature distance less than the threshold will be regarded as keypoints from same object.
- **num_dets** (*int*) – Num of raw boxes before doing nms.

Returns

Decoded output of CornerHead, containing the following Tensors:

- bboxes (Tensor): Coords of each box.
- scores (Tensor): Scores of each box.
- clses (Tensor): Categories of each box.

Return type tuple[torch.Tensor]

forward(*feats*)

Forward features from the upstream network.

Parameters **feats** (*tuple[Tensor]*) – Features from the upstream network, each is a 4D-tensor.

Returns

Usually a tuple of corner heatmaps, offset heatmaps and embedding heatmaps.

- tl_heats (list[Tensor]): Top-left corner heatmaps for all levels, each is a 4D-tensor, the channels number is num_classes.
- br_heats (list[Tensor]): Bottom-right corner heatmaps for all levels, each is a 4D-tensor, the channels number is num_classes.
- tl_embs (list[Tensor] | list[None]): Top-left embedding heatmaps for all levels, each is a 4D-tensor or None. If not None, the channels number is corner_emb_channels.
- br_embs (list[Tensor] | list[None]): Bottom-right embedding heatmaps for all levels, each is a 4D-tensor or None. If not None, the channels number is corner_emb_channels.
- tl_offs (list[Tensor]): Top-left offset heatmaps for all levels, each is a 4D-tensor. The channels number is corner_offset_channels.
- br_offs (list[Tensor]): Bottom-right offset heatmaps for all levels, each is a 4D-tensor. The channels number is corner_offset_channels.

Return type tuple

forward_single(*x*, *lvl_ind*, *return_pool=False*)
Forward feature of a single level.

Parameters

- **x** (*Tensor*) – Feature of a single level.
- **lvl_ind** (*int*) – Level index of current feature.
- **return_pool** (*bool*) – Return corner pool feature or not.

Returns

A tuple of CornerHead's output for current feature level. Containing the following Tensors:

- tl_heat (Tensor): Predicted top-left corner heatmap.
- br_heat (Tensor): Predicted bottom-right corner heatmap.
- tl_emb (Tensor | None): Predicted top-left embedding heatmap. None for *self.with_corner_emb == False*.
- br_emb (Tensor | None): Predicted bottom-right embedding heatmap. None for *self.with_corner_emb == False*.
- tl_off (Tensor): Predicted top-left offset heatmap.
- br_off (Tensor): Predicted bottom-right offset heatmap.
- tl_pool (Tensor): Top-left corner pool feature. Not must have.
- br_pool (Tensor): Bottom-right corner pool feature. Not must have.

Return type tuple[Tensor]

get_bboxes(*tl_heats*, *br_heats*, *tl_embs*, *br_embs*, *tl_offs*, *br_offs*, *img_metas*, *rescale=False*, *with_nms=True*)
Transform network output for a batch into bbox predictions.

Parameters

- **tl_heats** (*list[Tensor]*) – Top-left corner heatmaps for each level with shape (N, num_classes, H, W).
- **br_heats** (*list[Tensor]*) – Bottom-right corner heatmaps for each level with shape (N, num_classes, H, W).
- **tl_embs** (*list[Tensor]*) – Top-left corner embeddings for each level with shape (N, corner_emb_channels, H, W).
- **br_embs** (*list[Tensor]*) – Bottom-right corner embeddings for each level with shape (N, corner_emb_channels, H, W).
- **tl_offs** (*list[Tensor]*) – Top-left corner offsets for each level with shape (N, corner_offset_channels, H, W).
- **br_offs** (*list[Tensor]*) – Bottom-right corner offsets for each level with shape (N, corner_offset_channels, H, W).
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **rescale** (*bool*) – If True, return boxes in original image space. Default: False.
- **with_nms** (*bool*) – If True, do nms before return boxes. Default: True.

get_targets(*gt_bboxes*, *gt_labels*, *feat_shape*, *img_shape*, *with_corner_emb=False*, *with_guiding_shift=False*, *with_centripetal_shift=False*)

Generate corner targets.

Including corner heatmap, corner offset.

Optional: corner embedding, corner guiding shift, centripetal shift.

For CornerNet, we generate corner heatmap, corner offset and corner embedding from this function.

For CentripetalNet, we generate corner heatmap, corner offset, guiding shift and centripetal shift from this function.

Parameters

- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes of each image, each has shape (num_gt, 4).
- **gt_labels** (*list[Tensor]*) – Ground truth labels of each box, each has shape (num_gt,).
- **feat_shape** (*list[int]*) – Shape of output feature, [batch, channel, height, width].
- **img_shape** (*list[int]*) – Shape of input image, [height, width, channel].
- **with_corner_emb** (*bool*) – Generate corner embedding target or not. Default: False.
- **with_guiding_shift** (*bool*) – Generate guiding shift target or not. Default: False.
- **with_centripetal_shift** (*bool*) – Generate centripetal shift target or not. Default: False.

Returns

Ground truth of corner heatmap, corner offset, corner embedding, guiding shift and centripetal shift. Containing the following keys:

- topleft_heatmap (Tensor): Ground truth top-left corner heatmap.
- bottomright_heatmap (Tensor): Ground truth bottom-right corner heatmap.
- topleft_offset (Tensor): Ground truth top-left corner offset.
- bottomright_offset (Tensor): Ground truth bottom-right corner offset.
- corner_embedding (list[list[list[int]]]): Ground truth corner embedding. Not must have.
- topleft_guiding_shift (Tensor): Ground truth top-left corner guiding shift. Not must have.
- bottomright_guiding_shift (Tensor): Ground truth bottom-right corner guiding shift. Not must have.
- topleft_centripetal_shift (Tensor): Ground truth top-left corner centripetal shift. Not must have.
- bottomright_centripetal_shift (Tensor): Ground truth bottom-right corner centripetal shift. Not must have.

Return type dict

init_weights()

Initialize the weights.

loss(*tl_heats*, *br_heats*, *tl_embs*, *br_embs*, *tl_offs*, *br_offs*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)

Compute losses of the head.

Parameters

- **tl_heats** (*list[Tensor]*) – Top-left corner heatmaps for each level with shape (N, num_classes, H, W).
- **br_heats** (*list[Tensor]*) – Bottom-right corner heatmaps for each level with shape (N, num_classes, H, W).
- **tl_embs** (*list[Tensor]*) – Top-left corner embeddings for each level with shape (N, corner_emb_channels, H, W).
- **br_embs** (*list[Tensor]*) – Bottom-right corner embeddings for each level with shape (N, corner_emb_channels, H, W).
- **tl_offs** (*list[Tensor]*) – Top-left corner offsets for each level with shape (N, corner_offset_channels, H, W).
- **br_offs** (*list[Tensor]*) – Bottom-right corner offsets for each level with shape (N, corner_offset_channels, H, W).
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [left, top, right, bottom] format.
- **gt_labels** (*list[Tensor]*) – Class indices corresponding to each box.
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*list[Tensor] | None*) – Specify which bounding boxes can be ignored when computing the loss.

Returns

A dictionary of loss components. Containing the following losses:

- det_loss (list[Tensor]): Corner keypoint losses of all feature levels.
- pull_loss (list[Tensor]): Part one of AssociativeEmbedding losses of all feature levels.
- push_loss (list[Tensor]): Part two of AssociativeEmbedding losses of all feature levels.
- off_loss (list[Tensor]): Corner offset losses of all feature levels.

Return type dict[str, Tensor]

loss_single(*tl_hmp*, *br_hmp*, *tl_emb*, *br_emb*, *tl_off*, *br_off*, *targets*)

Compute losses for single level.

Parameters

- **tl_hmp** (*Tensor*) – Top-left corner heatmap for current level with shape (N, num_classes, H, W).
- **br_hmp** (*Tensor*) – Bottom-right corner heatmap for current level with shape (N, num_classes, H, W).
- **tl_emb** (*Tensor*) – Top-left corner embedding for current level with shape (N, corner_emb_channels, H, W).
- **br_emb** (*Tensor*) – Bottom-right corner embedding for current level with shape (N, corner_emb_channels, H, W).
- **tl_off** (*Tensor*) – Top-left corner offset for current level with shape (N, corner_offset_channels, H, W).
- **br_off** (*Tensor*) – Bottom-right corner offset for current level with shape (N, corner_offset_channels, H, W).

- **targets** (*dict*) – Corner target generated by *get_targets*.

Returns

Losses of the head's different branches containing the following losses:

- det_loss (Tensor): Corner keypoint loss.
- pull_loss (Tensor): Part one of AssociativeEmbedding loss.
- push_loss (Tensor): Part two of AssociativeEmbedding loss.
- off_loss (Tensor): Corner offset loss.

Return type tuple[torch.Tensor]

onnx_export(*tl_heats*, *br_heats*, *tl_embs*, *br_embs*, *tl_offs*, *br_offs*, *img_metas*, *rescale=False*, *with_nms=True*)

Transform network output for a batch into bbox predictions.

Parameters

- **tl_heats** (*list[Tensor]*) – Top-left corner heatmaps for each level with shape (N, num_classes, H, W).
- **br_heats** (*list[Tensor]*) – Bottom-right corner heatmaps for each level with shape (N, num_classes, H, W).
- **tl_embs** (*list[Tensor]*) – Top-left corner embeddings for each level with shape (N, corner_emb_channels, H, W).
- **br_embs** (*list[Tensor]*) – Bottom-right corner embeddings for each level with shape (N, corner_emb_channels, H, W).
- **tl_offs** (*list[Tensor]*) – Top-left corner offsets for each level with shape (N, corner_offset_channels, H, W).
- **br_offs** (*list[Tensor]*) – Bottom-right corner offsets for each level with shape (N, corner_offset_channels, H, W).
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **rescale** (*bool*) – If True, return boxes in original image space. Default: False.
- **with_nms** (*bool*) – If True, do nms before return boxes. Default: True.

Returns First tensor bboxes with shape [N, num_det, 5], 5 arrange as (x1, y1, x2, y2, score) and second element is class labels of shape [N, num_det].

Return type tuple[Tensor, Tensor]

class mmdet.models.dense_heads.**DETRHead**(*num_classes*, *in_channels*, *num_query=100*, *num_reg_fcs=2*, *transformer=None*, *sync_cls_avg_factor=False*, *positional_encoding={'normalize': True, 'num_feats': 128, 'type': 'SinePositionalEncoding'}*, *loss_cls={'bg_cls_weight': 0.1, 'class_weight': 1.0, 'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_sigmoid': False}*, *loss_bbox={'loss_weight': 5.0, 'type': 'L1Loss'}*, *loss_iou={'loss_weight': 2.0, 'type': 'GIoULoss'}*, *train_cfg={'assigner': {'cls_cost': {'type': 'ClassificationCost', 'weight': 1.0}, 'iou_cost': {'iou_mode': 'giou', 'type': 'IoUCost', 'weight': 2.0}, 'reg_cost': {'type': 'BBoxL1Cost', 'weight': 5.0}, 'type': 'HungarianAssigner'}}*, *test_cfg={'max_per_img': 100}*, *init_cfg=None*, ***kwargs*)

Implements the DETR transformer head.

See paper: End-to-End Object Detection with Transformers for details.

Parameters

- **num_classes** (*int*) – Number of categories excluding the background.
- **in_channels** (*int*) – Number of channels in the input feature map.
- **num_query** (*int*) – Number of query in Transformer.
- **num_reg_fcs** (*int, optional*) – Number of fully-connected layers used in *FFN*, which is then used for the regression head. Default 2.
- **(obj** (*test_cfg*) – `mmcv.ConfigDict`|dict): Config for transformer. Default: None.
- **sync_cls_avg_factor** (*bool*) – Whether to sync the avg_factor of all ranks. Default to False.
- **(obj** – `mmcv.ConfigDict`|dict): Config for position encoding.
- **(obj** – *mmcv.ConfigDict`|dict): Config of the classification loss. Default `CrossEntropyLoss.*
- **(obj** – *mmcv.ConfigDict`|dict): Config of the regression loss. Default `L1Loss.*
- **(obj** – *mmcv.ConfigDict`|dict): Config of the regression iou loss. Default `GIoULoss.*
- **(obj** – `mmcv.ConfigDict`|dict): Training config of transformer head.
- **(obj** – `mmcv.ConfigDict`|dict): Testing config of transformer head.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

forward(*feats*, *img_metas*)

Forward function.

Parameters

- **feats** (*tuple[Tensor]*) – Features from the upstream network, each is a 4D-tensor.
- **img_metas** (*list[dict]*) – List of image information.

Returns

Outputs for all scale levels.

- all_cls_scores_list (list[Tensor]): Classification scores for each scale level. Each is a 4D-tensor with shape [nb_dec, bs, num_query, cls_out_channels]. Note *cls_out_channels* should includes background.

- all_bbox_preds_list (list[Tensor]): Sigmoid regression outputs for each scale level. Each is a 4D-tensor with normalized coordinate format (cx, cy, w, h) and shape [nb_dec, bs, num_query, 4].

Return type tuple[list[Tensor], list[Tensor]]

`forward_onnx`(*feats*, *img_metas*)

Forward function for exporting to ONNX.

Over-write *forward* because: *masks* is directly created with zero (valid position tag) and has the same spatial size as *x*. Thus the construction of *masks* is different from that in *forward*.

Parameters

- **feats** (*tuple[Tensor]*) – Features from the upstream network, each is a 4D-tensor.
- **img_metas** (*list[dict]*) – List of image information.

Returns

Outputs for all scale levels.

- all_cls_scores_list (list[Tensor]): Classification scores for each scale level. Each is a 4D-tensor with shape [nb_dec, bs, num_query, cls_out_channels]. Note *cls_out_channels* should includes background.
- all_bbox_preds_list (list[Tensor]): Sigmoid regression outputs for each scale level. Each is a 4D-tensor with normalized coordinate format (cx, cy, w, h) and shape [nb_dec, bs, num_query, 4].

Return type tuple[list[Tensor], list[Tensor]]

`forward_single`(*x*, *img_metas*)

"Forward function for a single feature level.

Parameters

- **x** (*Tensor*) – Input feature from backbone's single stage, shape [bs, c, h, w].
- **img_metas** (*list[dict]*) – List of image information.

Returns

Outputs from the classification head, shape [nb_dec, bs, num_query, cls_out_channels]. Note cls_out_channels should includes background.

all_bbox_preds (Tensor): Sigmoid outputs from the regression head with normalized coordinate format (cx, cy, w, h). Shape [nb_dec, bs, num_query, 4].

Return type all_cls_scores (Tensor)

`forward_single_onnx`(*x*, *img_metas*)

"Forward function for a single feature level with ONNX exportation.

Parameters

- **x** (*Tensor*) – Input feature from backbone's single stage, shape [bs, c, h, w].
- **img_metas** (*list[dict]*) – List of image information.

Returns

Outputs from the classification head, shape [nb_dec, bs, num_query, cls_out_channels]. Note cls_out_channels should includes background.

all_bbox_preds (Tensor): Sigmoid outputs from the regression head with normalized coordinate format (cx, cy, w, h). Shape [nb_dec, bs, num_query, 4].

Return type all_cls_scores (Tensor)

forward_train(*x*, *img_metas*, *gt_bboxes*, *gt_labels=None*, *gt_bboxes_ignore=None*, *proposal_cfg=None*, ***kwargs*)

Forward function for training mode.

Parameters

- **x** (*list[Tensor]*) – Features from backbone.
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes** (*Tensor*) – Ground truth bboxes of the image, shape (num_gts, 4).
- **gt_labels** (*Tensor*) – Ground truth labels of each box, shape (num_gts,).
- **gt_bboxes_ignore** (*Tensor*) – Ground truth bboxes to be ignored, shape (num_ignored_gts, 4).
- **proposal_cfg** (*mmcv.Config*) – Test / postprocessing configuration, if None, test_cfg would be used.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

get_bboxes(*all_cls_scores_list*, *all_bbox_preds_list*, *img_metas*, *rescale=False*)

Transform network outputs for a batch into bbox predictions.

Parameters

- **all_cls_scores_list** (*list[Tensor]*) – Classification outputs for each feature level. Each is a 4D-tensor with shape [nb_dec, bs, num_query, cls_out_channels].
- **all_bbox_preds_list** (*list[Tensor]*) – Sigmoid regression outputs for each feature level. Each is a 4D-tensor with normalized coordinate format (cx, cy, w, h) and shape [nb_dec, bs, num_query, 4].
- **img_metas** (*list[dict]*) – Meta information of each image.
- **rescale** (*bool, optional*) – If True, return boxes in original image space. Default False.

Returns Each item in result_list is 2-tuple. The first item is an (n, 5) tensor, where the first 4 columns are bounding box positions (tl_x, tl_y, br_x, br_y) and the 5-th column is a score between 0 and 1. The second item is a (n,) tensor where each item is the predicted class label of the corresponding box.

Return type list[list[Tensor, Tensor]]

get_targets(*cls_scores_list*, *bbox_preds_list*, *gt_bboxes_list*, *gt_labels_list*, *img_metas*, *gt_bboxes_ignore_list=None*)

"Compute regression and classification targets for a batch image.

Outputs from a single decoder layer of a single feature level are used.

Parameters

- **cls_scores_list** (*list[Tensor]*) – Box score logits from a single decoder layer for each image with shape [num_query, cls_out_channels].
- **bbox_preds_list** (*list[Tensor]*) – Sigmoid outputs from a single decoder layer for each image, with normalized coordinate (cx, cy, w, h) and shape [num_query, 4].

- **gt_bboxes_list** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels_list** (*list[Tensor]*) – Ground truth class indices for each image with shape (num_gts,).
- **img_metas** (*list[dict]*) – List of image meta information.
- **gt_bboxes_ignore_list** (*list[Tensor], optional*) – Bounding boxes which can be ignored for each image. Default None.

Returns

a tuple containing the following targets.

- labels_list (list[Tensor]): Labels for all images.
- label_weights_list (list[Tensor]): Label weights for all images.
- bbox_targets_list (list[Tensor]): BBox targets for all images.
- bbox_weights_list (list[Tensor]): BBox weights for all images.
- num_total_pos (int): Number of positive samples in all images.
- num_total_neg (int): Number of negative samples in all images.

Return type tuple

init_weights()
Initialize weights of the transformer head.

loss(*all_cls_scores_list*, *all_bbox_preds_list*, *gt_bboxes_list*, *gt_labels_list*, *img_metas*, *gt_bboxes_ignore=None*)
"Loss function.

Only outputs from the last feature level are used for computing losses by default.

Parameters

- **all_cls_scores_list** (*list[Tensor]*) – Classification outputs for each feature level. Each is a 4D-tensor with shape [nb_dec, bs, num_query, cls_out_channels].
- **all_bbox_preds_list** (*list[Tensor]*) – Sigmoid regression outputs for each feature level. Each is a 4D-tensor with normalized coordinate format (cx, cy, w, h) and shape [nb_dec, bs, num_query, 4].
- **gt_bboxes_list** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels_list** (*list[Tensor]*) – Ground truth class indices for each image with shape (num_gts,).
- **img_metas** (*list[dict]*) – List of image meta information.
- **gt_bboxes_ignore** (*list[Tensor], optional*) – Bounding boxes which can be ignored for each image. Default None.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

loss_single(*cls_scores*, *bbox_preds*, *gt_bboxes_list*, *gt_labels_list*, *img_metas*, *gt_bboxes_ignore_list=None*)
"Loss function for outputs from a single decoder layer of a single feature level.

Parameters

- **cls_scores** (*Tensor*) – Box score logits from a single decoder layer for all images. Shape [bs, num_query, cls_out_channels].
- **bbox_preds** (*Tensor*) – Sigmoid outputs from a single decoder layer for all images, with normalized coordinate (cx, cy, w, h) and shape [bs, num_query, 4].
- **gt_bboxes_list** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels_list** (*list[Tensor]*) – Ground truth class indices for each image with shape (num_gts,).
- **img_metas** (*list[dict]*) – List of image meta information.
- **gt_bboxes_ignore_list** (*list[Tensor], optional*) – Bounding boxes which can be ignored for each image. Default None.

Returns

A dictionary of loss components for outputs from a single decoder layer.

Return type dict[str, Tensor]

onnx_export(*all_cls_scores_list*, *all_bbox_preds_list*, *img_metas*)

Transform network outputs into bbox predictions, with ONNX exportation.

Parameters

- **all_cls_scores_list** (*list[Tensor]*) – Classification outputs for each feature level. Each is a 4D-tensor with shape [nb_dec, bs, num_query, cls_out_channels].
- **all_bbox_preds_list** (*list[Tensor]*) – Sigmoid regression outputs for each feature level. Each is a 4D-tensor with normalized coordinate format (cx, cy, w, h) and shape [nb_dec, bs, num_query, 4].
- **img_metas** (*list[dict]*) – Meta information of each image.

Returns

dets of shape [N, num_det, 5] and class labels of shape [N, num_det].

Return type tuple[Tensor, Tensor]

simple_test_bboxes(*feats*, *img_metas*, *rescale=False*)

Test det bboxes without test-time augmentation.

Parameters

- **feats** (*tuple[torch.Tensor]*) – Multi-level features from the upstream network, each is a 4D-tensor.
- **img_metas** (*list[dict]*) – List of image information.
- **rescale** (*bool, optional*) – Whether to rescale the results. Defaults to False.

Returns

Each item in result_list is 2-tuple. The first item is `bboxes` with shape (n, 5), where 5 represent (tl_x, tl_y, br_x, br_y, score). The shape of the second tensor in the tuple is `labels` with shape (n,)

Return type list[tuple[Tensor, Tensor]]

class mmdet.models.dense_heads.**DecoupledSOLOHead**(**args*, *init_cfg=[{'type': 'Normal', 'layer': 'Conv2d', 'std': 0.01}, {'type': 'Normal', 'std': 0.01, 'bias_prob': 0.01, 'override': {'name': 'conv_mask_list_x'}}, {'type': 'Normal', 'std': 0.01, 'bias_prob': 0.01, 'override': {'name': 'conv_mask_list_y'}}, {'type': 'Normal', 'std': 0.01, 'bias_prob': 0.01, 'override': {'name': 'conv_cls'}}]*, ***kwargs*)

Decoupled SOLO mask head used in `SOLO: Segmenting Objects by Locations.

<https://arxiv.org/abs/1912.04488>`_

Parameters **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

forward(*feats*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

get_results(*mlvl_mask_preds_x*, *mlvl_mask_preds_y*, *mlvl_cls_scores*, *img_metas*, *rescale=None*, ***kwargs*)

Get multi-image mask results.

Parameters

- **mlvl_mask_preds_x** (*list[Tensor]*) – Multi-level mask prediction from x branch. Each element in the list has shape (batch_size, num_grids ,h ,w).
- **mlvl_mask_preds_y** (*list[Tensor]*) – Multi-level mask prediction from y branch. Each element in the list has shape (batch_size, num_grids ,h ,w).
- **mlvl_cls_scores** (*list[Tensor]*) – Multi-level scores. Each element in the list has shape (batch_size, num_classes ,num_grids ,num_grids).
- **img_metas** (*list[dict]*) – Meta information of all images.

Returns

Processed results of multiple images.Each `InstanceData` usually contains following keys.

- scores (Tensor): Classification scores, has shape (num_instance,).
- labels (Tensor): Has shape (num_instances,).
- masks (Tensor): Processed mask results, has shape (num_instances, h, w).

Return type list[`InstanceData`]

loss(*mlvl_mask_preds_x*, *mlvl_mask_preds_y*, *mlvl_cls_preds*, *gt_labels*, *gt_masks*, *img_metas*, *gt_bboxes=None*, ***kwargs*)

Calculate the loss of total batch.

Parameters

- **mlvl_mask_preds_x** (*list[Tensor]*) – Multi-level mask prediction from x branch. Each element in the list has shape (batch_size, num_grids ,h ,w).
- **mlvl_mask_preds_x** – Multi-level mask prediction from y branch. Each element in the list has shape (batch_size, num_grids ,h ,w).

- **mlvl_cls_preds** (*list[Tensor]*) – Multi-level scores. Each element in the list has shape (batch_size, num_classes, num_grids ,num_grids).
- **gt_labels** (*list[Tensor]*) – Labels of multiple images.
- **gt_masks** (*list[Tensor]*) – Ground truth masks of multiple images. Each has shape (num_instances, h, w).
- **img_metas** (*list[dict]*) – Meta information of multiple images.
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes of multiple images. Default: None.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

class mmdet.models.dense_heads.**DecoupledSOLOLightHead**(**args*, *dcn_cfg=None*, *init_cfg=[{'type': 'Normal', 'layer': 'Conv2d', 'std': 0.01}, {'type': 'Normal', 'std': 0.01, 'bias_prob': 0.01, 'override': {'name': 'conv_mask_list_x'}}, {'type': 'Normal', 'std': 0.01, 'bias_prob': 0.01, 'override': {'name': 'conv_mask_list_y'}}, {'type': 'Normal', 'std': 0.01, 'bias_prob': 0.01, 'override': {'name': 'conv_cls'}}]*, ***kwargs*)

Decoupled Light SOLO mask head used in SOLO: Segmenting Objects by Locations

Parameters

- **with_dcn** (*bool*) – Whether use dcn in mask_convs and cls_convs, default: False.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

forward(*feats*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.dense_heads.**DeformableDETRHead**(**args*, *with_box_refine=False*, *as_two_stage=False*, *transformer=None*, ***kwargs*)

Head of DeformDETR: Deformable DETR: Deformable Transformers for End-to- End Object Detection.

Code is modified from the official github repo.

More details can be found in the paper .

Parameters

- **with_box_refine** (*bool*) – Whether to refine the reference points in the decoder. Defaults to False.
- **as_two_stage** (*bool*) – Whether to generate the proposal from the outputs of encoder.
- **(obj** (*transformer*) – *ConfigDict*): ConfigDict is used for building the Encoder and Decoder.

forward(*mlvl_feats*, *img_metas*)
Forward function.

Parameters

- **mlvl_feats** (*tuple[Tensor]*) – Features from the upstream network, each is a 4D-tensor with shape (N, C, H, W).
- **img_metas** (*list[dict]*) – List of image information.

Returns Outputs from the classification head, shape [nb_dec, bs, num_query, cls_out_channels]. Note cls_out_channels should includes background. all_bbox_preds (Tensor): Sigmoid outputs from the regression head with normalized coordinate format (cx, cy, w, h). Shape [nb_dec, bs, num_query, 4]. enc_outputs_class (Tensor): The score of each point on encode feature map, has shape (N, h*w, num_class). Only when as_two_stage is True it would be returned, otherwise *None* would be returned. enc_outputs_coord (Tensor): The proposal generate from the encode feature map, has shape (N, h*w, 4). Only when as_two_stage is True it would be returned, otherwise *None* would be returned.

Return type all_cls_scores (Tensor)

get_bboxes(*all_cls_scores*, *all_bbox_preds*, *enc_cls_scores*, *enc_bbox_preds*, *img_metas*, *rescale=False*)
Transform network outputs for a batch into bbox predictions.

Parameters

- **all_cls_scores** (*Tensor*) – Classification score of all decoder layers, has shape [nb_dec, bs, num_query, cls_out_channels].
- **all_bbox_preds** (*Tensor*) – Sigmoid regression outputs of all decode layers. Each is a 4D-tensor with normalized coordinate format (cx, cy, w, h) and shape [nb_dec, bs, num_query, 4].
- **enc_cls_scores** (*Tensor*) – Classification scores of points on encode feature map , has shape (N, h*w, num_classes). Only be passed when as_two_stage is True, otherwise is None.
- **enc_bbox_preds** (*Tensor*) – Regression results of each points on the encode feature map, has shape (N, h*w, 4). Only be passed when as_two_stage is True, otherwise is None.
- **img_metas** (*list[dict]*) – Meta information of each image.
- **rescale** (*bool, optional*) – If True, return boxes in original image space. Default False.

Returns Each item in result_list is 2-tuple. The first item is an (n, 5) tensor, where the first 4 columns are bounding box positions (tl_x, tl_y, br_x, br_y) and the 5-th column is a score between 0 and 1. The second item is a (n,) tensor where each item is the predicted class label of the corresponding box.

Return type list[list[Tensor, Tensor]]

init_weights()
Initialize weights of the DeformDETR head.

loss(*all_cls_scores*, *all_bbox_preds*, *enc_cls_scores*, *enc_bbox_preds*, *gt_bboxes_list*, *gt_labels_list*, *img_metas*, *gt_bboxes_ignore=None*)
"Loss function.

Parameters

- **all_cls_scores** (*Tensor*) – Classification score of all decoder layers, has shape [nb_dec, bs, num_query, cls_out_channels].

- **all_bbox_preds** (*Tensor*) – Sigmoid regression outputs of all decode layers. Each is a 4D-tensor with normalized coordinate format (cx, cy, w, h) and shape [nb_dec, bs, num_query, 4].
- **enc_cls_scores** (*Tensor*) – Classification scores of points on encode feature map , has shape (N, h*w, num_classes). Only be passed when as_two_stage is True, otherwise is None.
- **enc_bbox_preds** (*Tensor*) – Regression results of each points on the encode feature map, has shape (N, h*w, 4). Only be passed when as_two_stage is True, otherwise is None.
- **gt_bboxes_list** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels_list** (*list[Tensor]*) – Ground truth class indices for each image with shape (num_gts,).
- **img_metas** (*list[dict]*) – List of image meta information.
- **gt_bboxes_ignore** (*list[Tensor], optional*) – Bounding boxes which can be ignored for each image. Default None.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

class mmdet.models.dense_heads.**EmbeddingRPNHead**(*num_proposals=100*, *proposal_feature_channel=256*, *init_cfg=None*, ***kwargs*)

RPNHead in the Sparse R-CNN .

Unlike traditional RPNHead, this module does not need FPN input, but just decode *init_proposal_bboxes* and expand the first dimension of *init_proposal_bboxes* and *init_proposal_features* to the batch_size.

Parameters

- **num_proposals** (*int*) – Number of init_proposals. Default 100.
- **proposal_feature_channel** (*int*) – Channel number of init_proposal_feature. Defaults to 256.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

forward_dummy(*img*, *img_metas*)
Dummy forward function.

Used in flops calculation.

forward_train(*img*, *img_metas*)
Forward function in training stage.

init_weights()
Initialize the init_proposal_bboxes as normalized.

[c_x, c_y, w, h], and we initialize it to the size of the entire image.

simple_test(*img*, *img_metas*)
Forward function in testing stage.

simple_test_rpn(*img*, *img_metas*)
Forward function in testing stage.

class mmdet.models.dense_heads.**FCOSHead**(*num_classes*, *in_channels*, *regress_ranges=((- 1, 64), (64, 128), (128, 256), (256, 512), (512, 100000000.0))*, *center_sampling=False*, *center_sample_radius=1.5*, *norm_on_bbox=False*, *centerness_on_reg=False*, *loss_cls={'alpha': 0.25, 'gamma': 2.0, 'loss_weight': 1.0, 'type': 'FocalLoss', 'use_sigmoid': True}*, *loss_bbox={'loss_weight': 1.0, 'type': 'IoULoss'}*, *loss_centerness={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, *norm_cfg={'num_groups': 32, 'requires_grad': True, 'type': 'GN'}*, *init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'conv_cls', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'}*, ***kwargs*)

Anchor-free head used in FCOS.

The FCOS head does not use anchor boxes. Instead bounding boxes are predicted at each pixel and a centerness measure is used to suppress low-quality predictions. Here norm_on_bbox, centerness_on_reg, dcn_on_last_conv are training tricks used in official repo, which will bring remarkable mAP gains of up to 4.9. Please see https://github.com/tianzhi0549/FCOS for more detail.

Parameters

- **num_classes** (*int*) – Number of categories excluding the background category.
- **in_channels** (*int*) – Number of channels in the input feature map.
- **strides** (*list[int] | list[tuple[int, int]]*) – Strides of points in multiple feature levels. Default: (4, 8, 16, 32, 64).
- **regress_ranges** (*tuple[tuple[int, int]]*) – Regress range of multiple level points.
- **center_sampling** (*bool*) – If true, use center sampling. Default: False.
- **center_sample_radius** (*float*) – Radius of center sampling. Default: 1.5.
- **norm_on_bbox** (*bool*) – If true, normalize the regression targets with FPN strides. Default: False.
- **centerness_on_reg** (*bool*) – If true, position centerness on the regress branch. Please refer to https://github.com/tianzhi0549/FCOS/issues/89#issuecomment-516877042. Default: False.
- **conv_bias** (*bool | str*) – If specified as *auto*, it will be decided by the norm_cfg. Bias of conv will be set as True if *norm_cfg* is None, otherwise False. Default: "auto".
- **loss_cls** (*dict*) – Config of classification loss.
- **loss_bbox** (*dict*) – Config of localization loss.
- **loss_centerness** (*dict*) – Config of centerness loss.
- **norm_cfg** (*dict*) – dictionary to construct and config norm layer. Default: norm_cfg=dict(type='GN', num_groups=32, requires_grad=True).
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

Example

```
>>> self = FCOSHead(11, 7)
>>> feats = [torch.rand(1, 7, s, s) for s in [4, 8, 16, 32, 64]]
>>> cls_score, bbox_pred, centerness = self.forward(feats)
>>> assert len(cls_score) == len(self.scales)
```

centerness_target(*pos_bbox_targets*)
Compute centerness targets.

Parameters **pos_bbox_targets** (*Tensor*) – BBox targets of positive bboxes in shape (num_pos, 4)

Returns Centerness target.

Return type Tensor

forward(*feats*)
Forward features from the upstream network.

Parameters **feats** (*tuple[Tensor]*) – Features from the upstream network, each is a 4D-tensor.

Returns cls_scores (list[Tensor]): Box scores for each scale level, each is a 4D-tensor, the channel number is num_points * num_classes. bbox_preds (list[Tensor]): Box energies / deltas for each scale level, each is a 4D-tensor, the channel number is num_points * 4. centernesses (list[Tensor]): centerness for each scale level, each is a 4D-tensor, the channel number is num_points * 1.

Return type tuple

forward_single(*x*, *scale*, *stride*)
Forward features of a single scale level.

Parameters

- **x** (*Tensor*) – FPN feature maps of the specified stride.
- **(** (*scale*) – obj: *mmcv.cnn.Scale*): Learnable scale module to resize the bbox prediction.
- **stride** (*int*) – The corresponding stride for feature maps, only used to normalize the bbox prediction when self.norm_on_bbox is True.

Returns scores for each class, bbox predictions and centerness predictions of input feature maps.

Return type tuple

get_targets(*points*, *gt_bboxes_list*, *gt_labels_list*)
Compute regression, classification and centerness targets for points in multiple images.

Parameters

- **points** (*list[Tensor]*) – Points of each fpn level, each has shape (num_points, 2).
- **gt_bboxes_list** (*list[Tensor]*) – Ground truth bboxes of each image, each has shape (num_gt, 4).
- **gt_labels_list** (*list[Tensor]*) – Ground truth labels of each box, each has shape (num_gt,).

Returns concat_lvl_labels (list[Tensor]): Labels of each level. concat_lvl_bbox_targets (list[Tensor]): BBox targets of each level.

Return type tuple

loss(*cls_scores*, *bbox_preds*, *centernesses*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)
Compute loss of the head.

Parameters

- **cls_scores** (*list[Tensor]*) – Box scores for each scale level, each is a 4D-tensor, the channel number is num_points * num_classes.
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level, each is a 4D-tensor, the channel number is num_points * 4.
- **centernesses** (*list[Tensor]*) – centerness for each scale level, each is a 4D-tensor, the channel number is num_points * 1.
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

class mmdet.models.dense_heads.**FSAFHead**(**args*, *score_threshold=None*, *init_cfg=None*, ***kwargs*)
Anchor-free head used in FSAF.

The head contains two subnetworks. The first classifies anchor boxes and the second regresses deltas for the anchors (num_anchors is 1 for anchor- free methods)

Parameters

- ***args** – Same as its base class in *RetinaHead*
- **score_threshold** (*float, optional*) – The score_threshold to calculate positive recall. If given, prediction scores lower than this value is counted as incorrect prediction. Default to None.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None
- ****kwargs** – Same as its base class in *RetinaHead*

Example

```
>>> import torch
>>> self = FSAFHead(11, 7)
>>> x = torch.rand(1, 7, 32, 32)
>>> cls_score, bbox_pred = self.forward_single(x)
>>> # Each anchor predicts a score for each class except background
>>> cls_per_anchor = cls_score.shape[1] / self.num_anchors
>>> box_per_anchor = bbox_pred.shape[1] / self.num_anchors
>>> assert cls_per_anchor == self.num_classes
>>> assert box_per_anchor == 4
```

calculate_pos_recall(*cls_scores*, *labels_list*, *pos_inds*)
Calculate positive recall with score threshold.

Parameters

- **cls_scores** (*list[Tensor]*) – Classification scores at all fpn levels. Each tensor is in shape (N, num_classes * num_anchors, H, W)
- **labels_list** (*list[Tensor]*) – The label that each anchor is assigned to. Shape (N * H * W * num_anchors,)
- **pos_inds** (*list[Tensor]*) – List of bool tensors indicating whether the anchor is assigned to a positive label. Shape (N * H * W * num_anchors,)

Returns A single float number indicating the positive recall.

Return type Tensor

collect_loss_level_single(*cls_loss*, *reg_loss*, *assigned_gt_inds*, *labels_seq*)

Get the average loss in each FPN level w.r.t. each gt label.

Parameters

- **cls_loss** (*Tensor*) – Classification loss of each feature map pixel, shape (num_anchor, num_class)
- **reg_loss** (*Tensor*) – Regression loss of each feature map pixel, shape (num_anchor, 4)
- **assigned_gt_inds** (*Tensor*) – It indicates which gt the prior is assigned to (0-based, -1: no assignment). shape (num_anchor),
- **labels_seq** – The rank of labels. shape (num_gt)

Returns (num_gt), average loss of each gt in this level

Return type shape

forward_single(*x*)

Forward feature map of a single scale level.

Parameters **x** (*Tensor*) – Feature map of a single scale level.

Returns

cls_score (Tensor): Box scores for each scale level Has shape (N, num_points * num_classes, H, W).

bbox_pred (Tensor): Box energies / deltas for each scale level with shape (N, num_points * 4, H, W).

Return type tuple (Tensor)

loss(*cls_scores*, *bbox_preds*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)

Compute loss of the head.

Parameters

- **cls_scores** (*list[Tensor]*) – Box scores for each scale level Has shape (N, num_points * num_classes, H, W).
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level with shape (N, num_points * 4, H, W).
- **gt_bboxes** (*list[Tensor]*) – each item are the truth boxes for each image in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.

- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

reweight_loss_single(*cls_loss*, *reg_loss*, *assigned_gt_inds*, *labels*, *level*, *min_levels*)

Reweight loss values at each level.

Reassign loss values at each level by masking those where the pre-calculated loss is too large. Then return the reduced losses.

Parameters

- **cls_loss** (*Tensor*) – Element-wise classification loss. Shape: (num_anchors, num_classes)
- **reg_loss** (*Tensor*) – Element-wise regression loss. Shape: (num_anchors, 4)
- **assigned_gt_inds** (*Tensor*) – The gt indices that each anchor bbox is assigned to. -1 denotes a negative anchor, otherwise it is the gt index (0-based). Shape: (num_anchors,),
- **labels** (*Tensor*) – Label assigned to anchors. Shape: (num_anchors,).
- **level** (*int*) – The current level index in the pyramid (0-4 for RetinaNet)
- **min_levels** (*Tensor*) – The best-matching level for each gt. Shape: (num_gts,),

Returns

- cls_loss: Reduced corrected classification loss. Scalar.
- reg_loss: Reduced corrected regression loss. Scalar.
- pos_flags (Tensor): Corrected bool tensor indicating the final positive anchors. Shape: (num_anchors,).

Return type tuple

class mmdet.models.dense_heads.**FeatureAdaption**(*in_channels*, *out_channels*, *kernel_size=3*, *deform_groups=4*, *init_cfg={'layer': 'Conv2d', 'override': {'name': 'conv_adaption', 'std': 0.01, 'type': 'Normal'}, 'std': 0.1, 'type': 'Normal'}*)

Feature Adaption Module.

Feature Adaption Module is implemented based on DCN v1. It uses anchor shape prediction rather than feature map to predict offsets of deform conv layer.

Parameters

- **in_channels** (*int*) – Number of channels in the input feature map.
- **out_channels** (*int*) – Number of channels in the output feature map.
- **kernel_size** (*int*) – Deformable conv kernel size.
- **deform_groups** (*int*) – Deformable conv group size.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

forward(*x*, *shape*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

class mmdet.models.dense_heads.**FoveaHead**(*num_classes*, *in_channels*, *base_edge_list=(16, 32, 64, 128, 256)*, *scale_ranges=((8, 32), (16, 64), (32, 128), (64, 256), (128, 512))*, *sigma=0.4*, *with_deform=False*, *deform_groups=4*, *init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'conv_cls', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'}*, ***kwargs*)

FoveaBox: Beyond Anchor-based Object Detector https://arxiv.org/abs/1904.03797

forward_single(*x*)

Forward features of a single scale level.

Parameters **x** (*Tensor*) – FPN feature maps of the specified stride.

Returns

Scores for each class, bbox predictions, features after classification and regression conv layers, some models needs these features like FCOS.

Return type tuple

get_targets(*gt_bbox_list*, *gt_label_list*, *featmap_sizes*, *points*)

Compute regression, classification and centerness targets for points in multiple images.

Parameters

- **points** (*list[Tensor]*) – Points of each fpn level, each has shape (num_points, 2).
- **gt_bboxes_list** (*list[Tensor]*) – Ground truth bboxes of each image, each has shape (num_gt, 4).
- **gt_labels_list** (*list[Tensor]*) – Ground truth labels of each box, each has shape (num_gt,).

loss(*cls_scores*, *bbox_preds*, *gt_bbox_list*, *gt_label_list*, *img_metas*, *gt_bboxes_ignore=None*)

Compute loss of the head.

Parameters

- **cls_scores** (*list[Tensor]*) – Box scores for each scale level, each is a 4D-tensor, the channel number is num_points * num_classes.
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level, each is a 4D-tensor, the channel number is num_points * 4.
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss.

class mmdet.models.dense_heads.**FreeAnchorRetinaHead**(*num_classes*, *in_channels*, *stacked_convs=4*, *conv_cfg=None*, *norm_cfg=None*, *pre_anchor_topk=50*, *bbox_thr=0.6*, *gamma=2.0*, *alpha=0.5*, ***kwargs*)

FreeAnchor RetinaHead used in https://arxiv.org/abs/1909.02466.

Parameters

- **num_classes** (*int*) – Number of categories excluding the background category.
- **in_channels** (*int*) – Number of channels in the input feature map.
- **stacked_convs** (*int*) – Number of conv layers in cls and reg tower. Default: 4.
- **conv_cfg** (*dict*) – dictionary to construct and config conv layer. Default: None.
- **norm_cfg** (*dict*) – dictionary to construct and config norm layer. Default: norm_cfg=dict(type='GN', num_groups=32, requires_grad=True).
- **pre_anchor_topk** (*int*) – Number of boxes that be token in each bag.
- **bbox_thr** (*float*) – The threshold of the saturated linear function. It is usually the same with the IoU threshold used in NMS.
- **gamma** (*float*) – Gamma parameter in focal loss.
- **alpha** (*float*) – Alpha parameter in focal loss.

loss(*cls_scores*, *bbox_preds*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)

Compute losses of the head.

Parameters

- **cls_scores** (*list[Tensor]*) – Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level with shape (N, num_anchors * 4, H, W)
- **gt_bboxes** (*list[Tensor]*) – each item are the truth boxes for each image in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

negative_bag_loss(*cls_prob*, *box_prob*)

Compute negative bag loss.

$FL((1 - P_{a_j \in A_+}) * (1 - P_j^{bg}))$.

$P_{a_j \in A_+}$: Box_probability of matched samples.

P_j^{bg}: Classification probability of negative samples.

Parameters

- **cls_prob** (*Tensor*) – Classification probability, in shape (num_img, num_anchors, num_classes).

- **box_prob** (*Tensor*) – Box probability, in shape (num_img, num_anchors, num_classes).

Returns Negative bag loss in shape (num_img, num_anchors, num_classes).

Return type Tensor

positive_bag_loss(*matched_cls_prob*, *matched_box_prob*)

Compute positive bag loss.

$-log(Mean - max(P_{ij}^{cls} * P_{ij}^{loc}))$.

P_{ij}^{cls}: matched_cls_prob, classification probability of matched samples.

P_{ij}^{loc}: matched_box_prob, box probability of matched samples.

Parameters

- **matched_cls_prob** (*Tensor*) – Classification probability of matched samples in shape (num_gt, pre_anchor_topk).
- **matched_box_prob** (*Tensor*) – BBox probability of matched samples, in shape (num_gt, pre_anchor_topk).

Returns Positive bag loss in shape (num_gt,).

Return type Tensor

class mmdet.models.dense_heads.**GARPNHead**(*in_channels*, *init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'conv_loc', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'}*, ***kwargs*)

Guided-Anchor-based RPN head.

forward_single(*x*)

Forward feature of a single scale level.

loss(*cls_scores*, *bbox_preds*, *shape_preds*, *loc_preds*, *gt_bboxes*, *img_metas*, *gt_bboxes_ignore=None*)

Compute losses of the head.

Parameters

- **cls_scores** (*list[Tensor]*) – Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level with shape (N, num_anchors * 4, H, W)
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss. Default: None

Returns A dictionary of loss components.

Return type dict[str, Tensor]

class mmdet.models.dense_heads.**GARetinaHead**(*num_classes*, *in_channels*, *stacked_convs=4*, *conv_cfg=None*, *norm_cfg=None*, *init_cfg=None*, ***kwargs*)

Guided-Anchor-based RetinaNet head.

forward_single(*x*)
Forward feature map of a single scale level.

class mmdet.models.dense_heads.**GFLHead**(*num_classes*, *in_channels*, *stacked_convs=4*, *conv_cfg=None*, *norm_cfg={'num_groups': 32, 'requires_grad': True, 'type': 'GN'}*, *loss_dfl={'loss_weight': 0.25, 'type': 'DistributionFocalLoss'}*, *bbox_coder={'type': 'DistancePointBBoxCoder'}*, *reg_max=16*, *init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'gfl_cls', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'}*, ***kwargs*)
Generalized Focal Loss: Learning Qualified and Distributed Bounding Boxes for Dense Object Detection.

GFL head structure is similar with ATSS, however GFL uses 1) joint representation for classification and localization quality, and 2) flexible General distribution for bounding box locations, which are supervised by Quality Focal Loss (QFL) and Distribution Focal Loss (DFL), respectively

https://arxiv.org/abs/2006.04388

Parameters

- **num_classes** (*int*) – Number of categories excluding the background category.
- **in_channels** (*int*) – Number of channels in the input feature map.
- **stacked_convs** (*int*) – Number of conv layers in cls and reg tower. Default: 4.
- **conv_cfg** (*dict*) – dictionary to construct and config conv layer. Default: None.
- **norm_cfg** (*dict*) – dictionary to construct and config norm layer. Default: dict(type='GN', num_groups=32, requires_grad=True).
- **loss_qfl** (*dict*) – Config of Quality Focal Loss (QFL).
- **bbox_coder** (*dict*) – Config of bbox coder. Defaults 'DistancePointBBoxCoder'.
- **reg_max** (*int*) – Max value of integral set :math: *{0, ..., reg_max}* in QFL setting. Default: 16.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

Example

```
>>> self = GFLHead(11, 7)
>>> feats = [torch.rand(1, 7, s, s) for s in [4, 8, 16, 32, 64]]
>>> cls_quality_score, bbox_pred = self.forward(feats)
>>> assert len(cls_quality_score) == len(self.scales)
```

anchor_center(*anchors*)
Get anchor centers from anchors.

Parameters **anchors** (*Tensor*) – Anchor list with shape (N, 4), "xyxy" format.

Returns Anchor centers with shape (N, 2), "xy" format.

Return type Tensor

forward(*feats*)
Forward features from the upstream network.

Parameters **feats** (*tuple[Tensor]*) – Features from the upstream network, each is a 4D-tensor.

Returns

Usually a tuple of classification scores and bbox prediction

cls_scores (list[Tensor]): Classification and quality (IoU) joint scores for all scale levels, each is a 4D-tensor, the channel number is num_classes.

bbox_preds (list[Tensor]): Box distribution logits for all scale levels, each is a 4D-tensor, the channel number is 4*(n+1), n is max value of integral set.

Return type tuple

`forward_single`(*x*, *scale*)
Forward feature of a single scale level.

Parameters

- **x** (*Tensor*) – Features of a single scale level.
- **(** (*scale*) – obj: *mmcv.cnn.Scale*): Learnable scale module to resize the bbox prediction.

Returns

cls_score (Tensor): Cls and quality joint scores for a single scale level the channel number is num_classes.

bbox_pred (Tensor): Box distribution logits for a single scale level, the channel number is 4*(n+1), n is max value of integral set.

Return type tuple

`get_targets`(*anchor_list*, *valid_flag_list*, *gt_bboxes_list*, *img_metas*, *gt_bboxes_ignore_list=None*, *gt_labels_list=None*, *label_channels=1*, *unmap_outputs=True*)
Get targets for GFL head.

This method is almost the same as *AnchorHead.get_targets()*. Besides returning the targets as the parent method does, it also returns the anchors as the first element of the returned tuple.

`loss`(*cls_scores*, *bbox_preds*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)
Compute losses of the head.

Parameters

- **cls_scores** (*list[Tensor]*) – Cls and quality scores for each scale level has shape (N, num_classes, H, W).
- **bbox_preds** (*list[Tensor]*) – Box distribution logits for each scale level with shape (N, 4*(n+1), H, W), n is max value of integral set.
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*list[Tensor] | None*) – specify which bounding boxes can be ignored when computing the loss.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

`loss_single`(*anchors*, *cls_score*, *bbox_pred*, *labels*, *label_weights*, *bbox_targets*, *stride*, *num_total_samples*)
Compute loss of a single scale level.

Parameters

- **anchors** (*Tensor*) – Box reference for each scale level with shape (N, num_total_anchors, 4).
- **cls_score** (*Tensor*) – Cls and quality joint scores for each scale level has shape (N, num_classes, H, W).
- **bbox_pred** (*Tensor*) – Box distribution logits for each scale level with shape (N, 4*(n+1), H, W), n is max value of integral set.
- **labels** (*Tensor*) – Labels of each anchors with shape (N, num_total_anchors).
- **label_weights** (*Tensor*) – Label weights of each anchor with shape (N, num_total_anchors)
- **bbox_targets** (*Tensor*) – BBox regression targets of each anchor weight shape (N, num_total_anchors, 4).
- **stride** (*tuple*) – Stride in this scale level.
- **num_total_samples** (*int*) – Number of positive samples that is reduced over all GPUs.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

class mmdet.models.dense_heads.**GuidedAnchorHead**(*num_classes*, *in_channels*, *feat_channels=256*, *approx_anchor_generator={'octave_base_scale': 8, 'ratios': [0.5, 1.0, 2.0], 'scales_per_octave': 3, 'strides': [4, 8, 16, 32, 64], 'type': 'AnchorGenerator'}*, *square_anchor_generator={'ratios': [1.0], 'scales': [8], 'strides': [4, 8, 16, 32, 64], 'type': 'AnchorGenerator'}*, *anchor_coder={'target_means': [0.0, 0.0, 0.0, 0.0], 'target_stds': [1.0, 1.0, 1.0, 1.0], 'type': 'DeltaXYWHBBoxCoder'}*, *bbox_coder={'target_means': [0.0, 0.0, 0.0, 0.0], 'target_stds': [1.0, 1.0, 1.0, 1.0], 'type': 'DeltaXYWHBBoxCoder'}*, *reg_decoded_bbox=False*, *deform_groups=4*, *loc_filter_thr=0.01*, *train_cfg=None*, *test_cfg=None*, *loss_loc={'alpha': 0.25, 'gamma': 2.0, 'loss_weight': 1.0, 'type': 'FocalLoss', 'use_sigmoid': True}*, *loss_shape={'beta': 0.2, 'loss_weight': 1.0, 'type': 'BoundedIoULoss'}*, *loss_cls={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, *loss_bbox={'beta': 1.0, 'loss_weight': 1.0, 'type': 'SmoothL1Loss'}*, *init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'conv_loc', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'}*)

Guided-Anchor-based head (GA-RPN, GA-RetinaNet, etc.).

This GuidedAnchorHead will predict high-quality feature guided anchors and locations where anchors will be kept in inference. There are mainly 3 categories of bounding-boxes.

- Sampled 9 pairs for target assignment. (approxes)
- The square boxes where the predicted anchors are based on. (squares)
- Guided anchors.

Please refer to https://arxiv.org/abs/1901.03278 for more details.

Parameters

- **num_classes** (*int*) – Number of classes.
- **in_channels** (*int*) – Number of channels in the input feature map.
- **feat_channels** (*int*) – Number of hidden channels.
- **approx_anchor_generator** (*dict*) – Config dict for approx generator
- **square_anchor_generator** (*dict*) – Config dict for square generator
- **anchor_coder** (*dict*) – Config dict for anchor coder
- **bbox_coder** (*dict*) – Config dict for bbox coder
- **reg_decoded_bbox** (*bool*) – If true, the regression loss would be applied directly on decoded bounding boxes, converting both the predicted boxes and regression targets to absolute coordinates format. Default False. It should be *True* when using *IoULoss*, *GIoULoss*, or *DIoULoss* in the bbox head.
- **deform_groups** – (int): Group number of DCN in FeatureAdaption module.
- **loc_filter_thr** (*float*) – Threshold to filter out unconcerned regions.
- **loss_loc** (*dict*) – Config of location loss.
- **loss_shape** (*dict*) – Config of anchor shape loss.
- **loss_cls** (*dict*) – Config of classification loss.
- **loss_bbox** (*dict*) – Config of bbox regression loss.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

forward(*feats*)

Forward features from the upstream network.

Parameters **feats** (*tuple[Tensor]*) – Features from the upstream network, each is a 4D-tensor.

Returns

A tuple of classification scores and bbox prediction.

- cls_scores (list[Tensor]): Classification scores for all scale levels, each is a 4D-tensor, the channels number is num_base_priors * num_classes.
- bbox_preds (list[Tensor]): Box energies / deltas for all scale levels, each is a 4D-tensor, the channels number is num_base_priors * 4.

Return type tuple

forward_single(*x*)

Forward feature of a single scale level.

Parameters **x** (*Tensor*) – Features of a single scale level.

Returns cls_score (Tensor): Cls scores for a single scale level the channels number is num_base_priors * num_classes. bbox_pred (Tensor): Box energies / deltas for a single scale level, the channels number is num_base_priors * 4.

Return type tuple

ga_loc_targets(*gt_bboxes_list*, *featmap_sizes*)
Compute location targets for guided anchoring.

Each feature map is divided into positive, negative and ignore regions. - positive regions: target 1, weight 1 - ignore regions: target 0, weight 0 - negative regions: target 0, weight 0.1

Parameters

- **gt_bboxes_list** (*list[Tensor]*) – Gt bboxes of each image.
- **featmap_sizes** (*list[tuple]*) – Multi level sizes of each feature maps.

Returns tuple

ga_shape_targets(*approx_list*, *inside_flag_list*, *square_list*, *gt_bboxes_list*, *img_metas*, *gt_bboxes_ignore_list=None*, *unmap_outputs=True*)
Compute guided anchoring targets.

Parameters

- **approx_list** (*list[list]*) – Multi level approxs of each image.
- **inside_flag_list** (*list[list]*) – Multi level inside flags of each image.
- **square_list** (*list[list]*) – Multi level squares of each image.
- **gt_bboxes_list** (*list[Tensor]*) – Ground truth bboxes of each image.
- **img_metas** (*list[dict]*) – Meta info of each image.
- **gt_bboxes_ignore_list** (*list[Tensor]*) – ignore list of gt bboxes.
- **unmap_outputs** (*bool*) – unmap outputs or not.

Returns tuple

get_anchors(*featmap_sizes*, *shape_preds*, *loc_preds*, *img_metas*, *use_loc_filter=False*, *device='cuda'*)
Get squares according to feature map sizes and guided anchors.

Parameters

- **featmap_sizes** (*list[tuple]*) – Multi-level feature map sizes.
- **shape_preds** (*list[tensor]*) – Multi-level shape predictions.
- **loc_preds** (*list[tensor]*) – Multi-level location predictions.
- **img_metas** (*list[dict]*) – Image meta info.
- **use_loc_filter** (*bool*) – Use loc filter or not.
- **device** (*torch.device | str*) – device for returned tensors

Returns

square approxs of each image, guided anchors of each image, loc masks of each image

Return type tuple

get_bboxes(*cls_scores*, *bbox_preds*, *shape_preds*, *loc_preds*, *img_metas*, *cfg=None*, *rescale=False*)
Transform network outputs of a batch into bbox results.

Note: When score_factors is not None, the cls_scores are usually multiplied by it then obtain the real score used in NMS, such as CenterNess in FCOS, IoU branch in ATSS.

Parameters

- **cls_scores** (*list[Tensor]*) – Classification scores for all scale levels, each is a 4D-tensor, has shape (batch_size, num_priors * num_classes, H, W).

- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for all scale levels, each is a 4D-tensor, has shape (batch_size, num_priors * 4, H, W).
- **score_factors** (*list[Tensor], Optional*) – Score factor for all scale level, each is a 4D-tensor, has shape (batch_size, num_priors * 1, H, W). Default None.
- **img_metas** (*list[dict], Optional*) – Image meta info. Default None.
- **cfg** (*mmcv.Config, Optional*) – Test / postprocessing configuration, if None, test_cfg would be used. Default None.
- **rescale** (*bool*) – If True, return boxes in original image space. Default False.
- **with_nms** (*bool*) – If True, do nms before return boxes. Default True.

Returns

Each item in result_list is 2-tuple. The first item is an (n, 5) tensor, where the first 4 columns are bounding box positions (tl_x, tl_y, br_x, br_y) and the 5-th column is a score between 0 and 1. The second item is a (n,) tensor where each item is the predicted class label of the corresponding box.

Return type list[list[Tensor, Tensor]]

get_sampled_approxs(*featmap_sizes*, *img_metas*, *device='cuda'*)

Get sampled approxs and inside flags according to feature map sizes.

Parameters

- **featmap_sizes** (*list[tuple]*) – Multi-level feature map sizes.
- **img_metas** (*list[dict]*) – Image meta info.
- **device** (*torch.device | str*) – device for returned tensors

Returns approxes of each image, inside flags of each image

Return type tuple

loss(*cls_scores*, *bbox_preds*, *shape_preds*, *loc_preds*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)

Compute losses of the head.

Parameters

- **cls_scores** (*list[Tensor]*) – Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level with shape (N, num_anchors * 4, H, W)
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss. Default: None

Returns A dictionary of loss components.

Return type dict[str, Tensor]

class mmdet.models.dense_heads.**LADHead**(**args*, *topk=9*, *score_voting=True*, *covariance_type='diag'*, ***kwargs*)

Label Assignment Head from the paper: Improving Object Detection by Label Assignment Distillation

forward_train(*x*, *label_assignment_results*, *img_metas*, *gt_bboxes*, *gt_labels=None*, *gt_bboxes_ignore=None*, ***kwargs*)

Forward train with the available label assignment (student receives from teacher).

Parameters

- **x** (*list[Tensor]*) – Features from FPN.
- **label_assignment_results** (*tuple*) – As the outputs defined in the function *self.get_label_assignment*.
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes** (*Tensor*) – Ground truth bboxes of the image, shape (num_gts, 4).
- **gt_labels** (*Tensor*) – Ground truth labels of each box, shape (num_gts,).
- **gt_bboxes_ignore** (*Tensor*) – Ground truth bboxes to be ignored, shape (num_ignored_gts, 4).

Returns (dict[str, Tensor]): A dictionary of loss components.

Return type losses

get_label_assignment(*cls_scores*, *bbox_preds*, *iou_preds*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)

Get label assignment (from teacher).

Parameters

- **cls_scores** (*list[Tensor]*) – Box scores for each scale level. Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level with shape (N, num_anchors * 4, H, W)
- **iou_preds** (*list[Tensor]*) – iou_preds for each scale level with shape (N, num_anchors * 1, H, W)
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*list[Tensor] | None*) – Specify which bounding boxes can be ignored when are computing the loss.

Returns

Returns a tuple containing label assignment variables.

- **labels (Tensor): Labels of all anchors, each with** shape (num_anchors,).
- **labels_weight (Tensor): Label weights of all anchor.** each with shape (num_anchors,).
- **bboxes_target (Tensor): BBox targets of all anchors.** each with shape (num_anchors, 4).

- **bboxes_weight (Tensor): BBox weights of all anchors.** each with shape (num_anchors, 4).
- **pos_inds_flatten (Tensor): Contains all index of positive** sample in all anchor.
- pos_anchors (Tensor): Positive anchors.
- num_pos (int): Number of positive anchors.

Return type tuple

loss(*cls_scores*, *bbox_preds*, *iou_preds*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*, *label_assignment_results=None*)
Compute losses of the head.

Parameters

- **cls_scores** (*list[Tensor]*) – Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level with shape (N, num_anchors * 4, H, W)
- **iou_preds** (*list[Tensor]*) – iou_preds for each scale level with shape (N, num_anchors * 1, H, W)
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*list[Tensor] | None*) – Specify which bounding boxes can be ignored when are computing the loss.
- **label_assignment_results** (*tuple*) – As the outputs defined in the function *self.get_label_assignment*.

Returns A dictionary of loss gmm_assignment.

Return type dict[str, Tensor]

class mmdet.models.dense_heads.**LDHead**(*num_classes*, *in_channels*, *loss_ld={'T': 10, 'loss_weight': 0.25, 'type': 'LocalizationDistillationLoss'}*, ***kwargs*)
Localization distillation Head. (Short description)

It utilizes the learned bbox distributions to transfer the localization dark knowledge from teacher to student. Original paper: Localization Distillation for Object Detection.

Parameters

- **num_classes** (*int*) – Number of categories excluding the background category.
- **in_channels** (*int*) – Number of channels in the input feature map.
- **loss_ld** (*dict*) – Config of Localization Distillation Loss (LD), T is the temperature for distillation.

forward_train(*x*, *out_teacher*, *img_metas*, *gt_bboxes*, *gt_labels=None*, *gt_bboxes_ignore=None*, *proposal_cfg=None*, ***kwargs*)

Parameters

- **x** (*list[Tensor]*) – Features from FPN.
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes** (*Tensor*) – Ground truth bboxes of the image, shape (num_gts, 4).
- **gt_labels** (*Tensor*) – Ground truth labels of each box, shape (num_gts,).
- **gt_bboxes_ignore** (*Tensor*) – Ground truth bboxes to be ignored, shape (num_ignored_gts, 4).
- **proposal_cfg** (*mmcv.Config*) – Test / postprocessing configuration, if None, test_cfg would be used

Returns

The loss components and proposals of each image.

- losses (dict[str, Tensor]): A dictionary of loss components.
- proposal_list (list[Tensor]): Proposals of each image.

Return type tuple[dict, list]

loss(*cls_scores*, *bbox_preds*, *gt_bboxes*, *gt_labels*, *soft_target*, *img_metas*, *gt_bboxes_ignore=None*)

Compute losses of the head.

Parameters

- **cls_scores** (*list[Tensor]*) – Cls and quality scores for each scale level has shape (N, num_classes, H, W).
- **bbox_preds** (*list[Tensor]*) – Box distribution logits for each scale level with shape (N, 4*(n+1), H, W), n is max value of integral set.
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*list[Tensor] | None*) – specify which bounding boxes can be ignored when computing the loss.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

loss_single(*anchors*, *cls_score*, *bbox_pred*, *labels*, *label_weights*, *bbox_targets*, *stride*, *soft_targets*, *num_total_samples*)

Compute loss of a single scale level.

Parameters

- **anchors** (*Tensor*) – Box reference for each scale level with shape (N, num_total_anchors, 4).
- **cls_score** (*Tensor*) – Cls and quality joint scores for each scale level has shape (N, num_classes, H, W).
- **bbox_pred** (*Tensor*) – Box distribution logits for each scale level with shape (N, 4*(n+1), H, W), n is max value of integral set.
- **labels** (*Tensor*) – Labels of each anchors with shape (N, num_total_anchors).

- **label_weights** (*Tensor*) – Label weights of each anchor with shape (N, num_total_anchors)
- **bbox_targets** (*Tensor*) – BBox regression targets of each anchor weight shape (N, num_total_anchors, 4).
- **stride** (*tuple*) – Stride in this scale level.
- **num_total_samples** (*int*) – Number of positive samples that is reduced over all GPUs.

Returns Loss components and weight targets.

Return type dict[tuple, Tensor]

class mmdet.models.dense_heads.**NASFCOSHead**(**args*, *init_cfg=None*, ***kwargs*)

Anchor-free head used in NASFCOS.

It is quite similar with FCOS head, except for the searched structure of classification branch and bbox regression branch, where a structure of "dconv3x3, conv3x3, dconv3x3, conv1x1" is utilized instead.

class mmdet.models.dense_heads.**PAAHead**(**args*, *topk=9*, *score_voting=True*, *covariance_type='diag'*, ***kwargs*)

Head of PAAAssignment: Probabilistic Anchor Assignment with IoU Prediction for Object Detection.

Code is modified from the official github repo.

More details can be found in the paper .

Parameters

- **topk** (*int*) – Select topk samples with smallest loss in each level.
- **score_voting** (*bool*) – Whether to use score voting in post-process.
- **covariance_type** – String describing the type of covariance parameters to be used in `sklearn.mixture.GaussianMixture`. It must be one of:
  - 'full': each component has its own general covariance matrix
  - 'tied': all components share the same general covariance matrix
  - 'diag': each component has its own diagonal covariance matrix
  - 'spherical': each component has its own single variance

  Default: 'diag'. From 'full' to 'spherical', the gmm fitting process is faster yet the performance could be influenced. For most cases, 'diag' should be a good choice.

get_bboxes(*cls_scores*, *bbox_preds*, *score_factors=None*, *img_metas=None*, *cfg=None*, *rescale=False*, *with_nms=True*, ***kwargs*)

Transform network outputs of a batch into bbox results.

Note: When score_factors is not None, the cls_scores are usually multiplied by it then obtain the real score used in NMS, such as CenterNess in FCOS, IoU branch in ATSS.

Parameters

- **cls_scores** (*list[Tensor]*) – Classification scores for all scale levels, each is a 4D-tensor, has shape (batch_size, num_priors * num_classes, H, W).
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for all scale levels, each is a 4D-tensor, has shape (batch_size, num_priors * 4, H, W).
- **score_factors** (*list[Tensor], Optional*) – Score factor for all scale level, each is a 4D-tensor, has shape (batch_size, num_priors * 1, H, W). Default None.
- **img_metas** (*list[dict], Optional*) – Image meta info. Default None.

- **cfg** (*mmcv.Config, Optional*) – Test / postprocessing configuration, if None, test_cfg would be used. Default None.
- **rescale** (*bool*) – If True, return boxes in original image space. Default False.
- **with_nms** (*bool*) – If True, do nms before return boxes. Default True.

Returns

Each item in result_list is 2-tuple. The first item is an (n, 5) tensor, where the first 4 columns are bounding box positions (tl_x, tl_y, br_x, br_y) and the 5-th column is a score between 0 and 1. The second item is a (n,) tensor where each item is the predicted class label of the corresponding box.

Return type list[list[Tensor, Tensor]]

get_pos_loss(*anchors*, *cls_score*, *bbox_pred*, *label*, *label_weight*, *bbox_target*, *bbox_weight*, *pos_inds*)

Calculate loss of all potential positive samples obtained from first match process.

Parameters

- **anchors** (*list[Tensor]*) – Anchors of each scale.
- **cls_score** (*Tensor*) – Box scores of single image with shape (num_anchors, num_classes)
- **bbox_pred** (*Tensor*) – Box energies / deltas of single image with shape (num_anchors, 4)
- **label** (*Tensor*) – classification target of each anchor with shape (num_anchors,)
- **label_weight** (*Tensor*) – Classification loss weight of each anchor with shape (num_anchors).
- **bbox_target** (*dict*) – Regression target of each anchor with shape (num_anchors, 4).
- **bbox_weight** (*Tensor*) – Bbox weight of each anchor with shape (num_anchors, 4).
- **pos_inds** (*Tensor*) – Index of all positive samples got from first assign process.

Returns Losses of all positive samples in single image.

Return type Tensor

get_targets(*anchor_list*, *valid_flag_list*, *gt_bboxes_list*, *img_metas*, *gt_bboxes_ignore_list=None*, *gt_labels_list=None*, *label_channels=1*, *unmap_outputs=True*)

Get targets for PAA head.

This method is almost the same as *AnchorHead.get_targets()*. We direct return the results from _get_targets_single instead map it to levels by images_to_levels function.

Parameters

- **anchor_list** (*list[list[Tensor]]*) – Multi level anchors of each image. The outer list indicates images, and the inner list corresponds to feature levels of the image. Each element of the inner list is a tensor of shape (num_anchors, 4).
- **valid_flag_list** (*list[list[Tensor]]*) – Multi level valid flags of each image. The outer list indicates images, and the inner list corresponds to feature levels of the image. Each element of the inner list is a tensor of shape (num_anchors,)
- **gt_bboxes_list** (*list[Tensor]*) – Ground truth bboxes of each image.
- **img_metas** (*list[dict]*) – Meta info of each image.
- **gt_bboxes_ignore_list** (*list[Tensor]*) – Ground truth bboxes to be ignored.

- **gt_labels_list** (*list[Tensor]*) – Ground truth labels of each box.
- **label_channels** (*int*) – Channel of label.
- **unmap_outputs** (*bool*) – Whether to map outputs back to the original set of anchors.

Returns

Usually returns a tuple containing learning targets.

- **labels (list[Tensor]): Labels of all anchors, each with** shape (num_anchors,).
- **label_weights (list[Tensor]): Label weights of all anchor.** each with shape (num_anchors,).
- **bbox_targets (list[Tensor]): BBox targets of all anchors.** each with shape (num_anchors, 4).
- **bbox_weights (list[Tensor]): BBox weights of all anchors.** each with shape (num_anchors, 4).
- **pos_inds (list[Tensor]): Contains all index of positive** sample in all anchor.
- **gt_inds (list[Tensor]): Contains all gt_index of positive** sample in all anchor.

Return type tuple

gmm_separation_scheme(*gmm_assignment*, *scores*, *pos_inds_gmm*)

A general separation scheme for gmm model.

It separates a GMM distribution of candidate samples into three parts, 0 1 and uncertain areas, and you can implement other separation schemes by rewriting this function.

Parameters

- **gmm_assignment** (*Tensor*) – The prediction of GMM which is of shape (num_samples,). The 0/1 value indicates the distribution that each sample comes from.
- **scores** (*Tensor*) – The probability of sample coming from the fit GMM distribution. The tensor is of shape (num_samples,).
- **pos_inds_gmm** (*Tensor*) – All the indexes of samples which are used to fit GMM model. The tensor is of shape (num_samples,)

Returns

The indices of positive and ignored samples.

- pos_inds_temp (Tensor): Indices of positive samples.
- ignore_inds_temp (Tensor): Indices of ignore samples.

Return type tuple[Tensor]

loss(*cls_scores*, *bbox_preds*, *iou_preds*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)

Compute losses of the head.

Parameters

- **cls_scores** (*list[Tensor]*) – Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level with shape (N, num_anchors * 4, H, W)
- **iou_preds** (*list[Tensor]*) – iou_preds for each scale level with shape (N, num_anchors * 1, H, W)

- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*list[Tensor] | None*) – Specify which bounding boxes can be ignored when are computing the loss.

Returns A dictionary of loss gmm_assignment.

Return type dict[str, Tensor]

paa_reassign(*pos_losses*, *label*, *label_weight*, *bbox_weight*, *pos_inds*, *pos_gt_inds*, *anchors*)

Fit loss to GMM distribution and separate positive, ignore, negative samples again with GMM model.

Parameters

- **pos_losses** (*Tensor*) – Losses of all positive samples in single image.
- **label** (*Tensor*) – classification target of each anchor with shape (num_anchors,)
- **label_weight** (*Tensor*) – Classification loss weight of each anchor with shape (num_anchors).
- **bbox_weight** (*Tensor*) – Bbox weight of each anchor with shape (num_anchors, 4).
- **pos_inds** (*Tensor*) – Index of all positive samples got from first assign process.
- **pos_gt_inds** (*Tensor*) – Gt_index of all positive samples got from first assign process.
- **anchors** (*list[Tensor]*) – Anchors of each scale.

Returns

Usually returns a tuple containing learning targets.

- label (Tensor): classification target of each anchor after paa assign, with shape (num_anchors,)
- label_weight (Tensor): Classification loss weight of each anchor after paa assign, with shape (num_anchors).
- bbox_weight (Tensor): Bbox weight of each anchor with shape (num_anchors, 4).
- num_pos (int): The number of positive samples after paa assign.

Return type tuple

score_voting(*det_bboxes*, *det_labels*, *mlvl_bboxes*, *mlvl_nms_scores*, *score_thr*)

Implementation of score voting method works on each remaining boxes after NMS procedure.

Parameters

- **det_bboxes** (*Tensor*) – Remaining boxes after NMS procedure, with shape (k, 5), each dimension means (x1, y1, x2, y2, score).
- **det_labels** (*Tensor*) – The label of remaining boxes, with shape (k, 1),Labels are 0-based.
- **mlvl_bboxes** (*Tensor*) – All boxes before the NMS procedure, with shape (num_anchors,4).
- **mlvl_nms_scores** (*Tensor*) – The scores of all boxes which is used in the NMS procedure, with shape (num_anchors, num_class)

- **score_thr** (*float*) – The score threshold of bboxes.

Returns

Usually returns a tuple containing voting results.

- **det_bboxes_voted (Tensor): Remaining boxes after** score voting procedure, with shape (k, 5), each dimension means (x1, y1, x2, y2, score).
- **det_labels_voted (Tensor): Label of remaining bboxes** after voting, with shape (num_anchors,).

Return type tuple

class mmdet.models.dense_heads.**PISARetinaHead**(*num_classes*, *in_channels*, *stacked_convs=4*, *conv_cfg=None*, *norm_cfg=None*, *anchor_generator={'octave_base_scale': 4, 'ratios': [0.5, 1.0, 2.0], 'scales_per_octave': 3, 'strides': [8, 16, 32, 64, 128], 'type': 'AnchorGenerator'}*, *init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'retina_cls', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'}*, ***kwargs*)

PISA Retinanet Head.

The head owns the same structure with Retinanet Head, but differs in two aspects: 1. Importance-based Sample Reweighting Positive (ISR-P) is applied to

change the positive loss weights.

2. Classification-aware regression loss is adopted as a third loss.

loss(*cls_scores*, *bbox_preds*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)

Compute losses of the head.

Parameters

- **cls_scores** (*list[Tensor]*) – Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level with shape (N, num_anchors * 4, H, W)
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes of each image with shape (num_obj, 4).
- **gt_labels** (*list[Tensor]*) – Ground truth labels of each image with shape (num_obj, 4).
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*list[Tensor]*) – Ignored gt bboxes of each image. Default: None.

Returns

Loss dict, comprise classification loss, regression loss and carl loss.

Return type dict

class mmdet.models.dense_heads.**PISASSDHead**(*num_classes=80*, *in_channels=(512, 1024, 512, 256, 256, 256)*, *stacked_convs=0*, *feat_channels=256*, *use_depthwise=False*, *conv_cfg=None*, *norm_cfg=None*, *act_cfg=None*, *anchor_generator={'basesize_ratio_range': (0.1, 0.9), 'input_size': 300, 'ratios': ([2], [2, 3], [2, 3], [2, 3], [2], [2]), 'scale_major': False, 'strides': [8, 16, 32, 64, 100, 300], 'type': 'SSDAnchorGenerator'}*, *bbox_coder={'clip_border': True, 'target_means': [0.0, 0.0, 0.0, 0.0], 'target_stds': [1.0, 1.0, 1.0, 1.0], 'type': 'DeltaXYWHBBoxCoder'}*, *reg_decoded_bbox=False*, *train_cfg=None*, *test_cfg=None*, *init_cfg={'bias': 0, 'distribution': 'uniform', 'layer': 'Conv2d', 'type': 'Xavier'}*)

loss(*cls_scores*, *bbox_preds*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)
Compute losses of the head.

Parameters

- **cls_scores** (*list[Tensor]*) – Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level with shape (N, num_anchors * 4, H, W)
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes of each image with shape (num_obj, 4).
- **gt_labels** (*list[Tensor]*) – Ground truth labels of each image with shape (num_obj, 4).
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*list[Tensor]*) – Ignored gt bboxes of each image. Default: None.

Returns

Loss dict, comprise classification loss regression loss and carl loss.

Return type dict

class mmdet.models.dense_heads.**RPNHead**(*in_channels*, *init_cfg={'layer': 'Conv2d', 'std': 0.01, 'type': 'Normal'}*, *num_convs=1*, ***kwargs*)

RPN head.

Parameters

- **in_channels** (*int*) – Number of channels in the input feature map.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.
- **num_convs** (*int*) – Number of convolution layers in the head. Default 1.

forward_single(*x*)
Forward feature map of a single scale level.

loss(*cls_scores*, *bbox_preds*, *gt_bboxes*, *img_metas*, *gt_bboxes_ignore=None*)
Compute losses of the head.

Parameters

- **cls_scores** (*list[Tensor]*) – Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)

- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level with shape (N, num_anchors * 4, H, W)
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

onnx_export(*x*, *img_metas*)

Test without augmentation.

Parameters

- **x** (*tuple[Tensor]*) – Features from the upstream network, each is a 4D-tensor.
- **img_metas** (*list[dict]*) – Meta info of each image.

Returns dets of shape [N, num_det, 5].

Return type Tensor

class mmdet.models.dense_heads.**RepPointsHead**(*num_classes*, *in_channels*, *point_feat_channels=256*, *num_points=9*, *gradient_mul=0.1*, *point_strides=[8, 16, 32, 64, 128]*, *point_base_scale=4*, *loss_cls={'alpha': 0.25, 'gamma': 2.0, 'loss_weight': 1.0, 'type': 'FocalLoss', 'use_sigmoid': True}*, *loss_bbox_init={'beta': 0.1111111111111111, 'loss_weight': 0.5, 'type': 'SmoothL1Loss'}*, *loss_bbox_refine={'beta': 0.1111111111111111, 'loss_weight': 1.0, 'type': 'SmoothL1Loss'}*, *use_grid_points=False*, *center_init=True*, *transform_method='moment'*, *moment_mul=0.01*, *init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'reppoints_cls_out', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'}*, ***kwargs*)

RepPoint head.

Parameters

- **point_feat_channels** (*int*) – Number of channels of points features.
- **gradient_mul** (*float*) – The multiplier to gradients from points refinement and recognition.
- **point_strides** (*Iterable*) – points strides.
- **point_base_scale** (*int*) – bbox scale for assigning labels.
- **loss_cls** (*dict*) – Config of classification loss.
- **loss_bbox_init** (*dict*) – Config of initial points loss.
- **loss_bbox_refine** (*dict*) – Config of points loss in refinement.
- **use_grid_points** (*bool*) – If we use bounding box representation, the
- **is represented as grid points on the bounding box.** (*reppoints*) –

- **center_init** (*bool*) – Whether to use center point assignment.
- **transform_method** (*str*) – The methods to transform RepPoints to bbox.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

centers_to_bboxes(*point_list*)
Get bboxes according to center points.

Only used in `MaxIoUAssigner`.

forward(*feats*)
Forward features from the upstream network.

Parameters **feats** (*tuple[Tensor]*) – Features from the upstream network, each is a 4D-tensor.

Returns

Usually contain classification scores and bbox predictions.

cls_scores (list[Tensor]): Box scores for each scale level, each is a 4D-tensor, the channel number is num_points * num_classes.

bbox_preds (list[Tensor]): Box energies / deltas for each scale level, each is a 4D-tensor, the channel number is num_points * 4.

Return type tuple

forward_single(*x*)
Forward feature map of a single FPN level.

gen_grid_from_reg(*reg*, *previous_boxes*)
Base on the previous bboxes and regression values, we compute the regressed bboxes and generate the grids on the bboxes.

Parameters

- **reg** – the regression value to previous bboxes.
- **previous_boxes** – previous bboxes.

Returns generate grids on the regressed bboxes.

get_points(*featmap_sizes*, *img_metas*, *device*)
Get points according to feature map sizes.

Parameters

- **featmap_sizes** (*list[tuple]*) – Multi-level feature map sizes.
- **img_metas** (*list[dict]*) – Image meta info.

Returns points of each image, valid flags of each image

Return type tuple

get_targets(*proposals_list*, *valid_flag_list*, *gt_bboxes_list*, *img_metas*, *gt_bboxes_ignore_list=None*, *gt_labels_list=None*, *stage='init'*, *label_channels=1*, *unmap_outputs=True*)
Compute corresponding GT box and classification targets for proposals.

Parameters

- **proposals_list** (*list[list]*) – Multi level points/bboxes of each image.
- **valid_flag_list** (*list[list]*) – Multi level valid flags of each image.
- **gt_bboxes_list** (*list[Tensor]*) – Ground truth bboxes of each image.

- **img_metas** (*list[dict]*) – Meta info of each image.
- **gt_bboxes_ignore_list** (*list[Tensor]*) – Ground truth bboxes to be ignored.
- **gt_bboxes_list** – Ground truth labels of each box.
- **stage** (*str*) – *init* or *refine*. Generate target for init stage or refine stage
- **label_channels** (*int*) – Channel of label.
- **unmap_outputs** (*bool*) – Whether to map outputs back to the original set of anchors.

Returns

- labels_list (list[Tensor]): Labels of each level.
- label_weights_list (list[Tensor]): Label weights of each level. # noqa: E501
- bbox_gt_list (list[Tensor]): Ground truth bbox of each level.
- proposal_list (list[Tensor]): Proposals(points/bboxes) of each level. # noqa: E501
- proposal_weights_list (list[Tensor]): Proposal weights of each level. # noqa: E501
- num_total_pos (int): Number of positive samples in all images. # noqa: E501
- num_total_neg (int): Number of negative samples in all images. # noqa: E501

Return type tuple

loss(*cls_scores*, *pts_preds_init*, *pts_preds_refine*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)
Compute loss of the head.

Parameters

- **cls_scores** (*list[Tensor]*) – Box scores for each scale level, each is a 4D-tensor, the channel number is num_points * num_classes.
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level, each is a 4D-tensor, the channel number is num_points * 4.
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss.

offset_to_pts(*center_list*, *pred_list*)
Change from point offset to point coordinate.

points2bbox(*pts*, *y_first=True*)
Converting the points set into bounding box.

Parameters

- **pts** – the input points sets (fields), each points set (fields) is represented as 2n scalar.
- **y_first** – if y_first=True, the point set is represented as [y1, x1, y2, x2 … yn, xn], otherwise the point set is represented as [x1, y1, x2, y2 … xn, yn].

Returns each points set is converting to a bbox [x1, y1, x2, y2].

class mmdet.models.dense_heads.**RetinaHead**(*num_classes*, *in_channels*, *stacked_convs=4*, *conv_cfg=None*, *norm_cfg=None*, *anchor_generator={'octave_base_scale': 4, 'ratios': [0.5, 1.0, 2.0], 'scales_per_octave': 3, 'strides': [8, 16, 32, 64, 128], 'type': 'AnchorGenerator'}*, *init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'retina_cls', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'}*, ***kwargs*)

An anchor-based head used in RetinaNet.

The head contains two subnetworks. The first classifies anchor boxes and the second regresses deltas for the anchors.

Example

```
>>> import torch
>>> self = RetinaHead(11, 7)
>>> x = torch.rand(1, 7, 32, 32)
>>> cls_score, bbox_pred = self.forward_single(x)
>>> # Each anchor predicts a score for each class except background
>>> cls_per_anchor = cls_score.shape[1] / self.num_anchors
>>> box_per_anchor = bbox_pred.shape[1] / self.num_anchors
>>> assert cls_per_anchor == (self.num_classes)
>>> assert box_per_anchor == 4
```

forward_single(*x*)

Forward feature of a single scale level.

Parameters **x** (*Tensor*) – Features of a single scale level.

Returns

cls_score (Tensor): Cls scores for a single scale level the channels number is num_anchors * num_classes.

bbox_pred (Tensor): Box energies / deltas for a single scale level, the channels number is num_anchors * 4.

Return type tuple

class mmdet.models.dense_heads.**RetinaSepBNHead**(*num_classes*, *num_ins*, *in_channels*, *stacked_convs=4*, *conv_cfg=None*, *norm_cfg=None*, *init_cfg=None*, ***kwargs*)

“RetinaHead with separate BN.

In RetinaHead, conv/norm layers are shared across different FPN levels, while in RetinaSepBNHead, conv layers are shared across different FPN levels, but BN layers are separated.

forward(*feats*)

Forward features from the upstream network.

Parameters **feats** (*tuple[Tensor]*) – Features from the upstream network, each is a 4D-tensor.

Returns

Usually a tuple of classification scores and bbox prediction

cls_scores (list[Tensor]): Classification scores for all scale levels, each is a 4D-tensor, the channels number is num_anchors * num_classes.

bbox_preds (list[Tensor]): Box energies / deltas for all scale levels, each is a 4D-tensor, the channels number is num_anchors * 4.

Return type tuple

`init_weights()`
Initialize weights of the head.

`class mmdet.models.dense_heads.SABLRetinaHead(num_classes, in_channels, stacked_convs=4, feat_channels=256, approx_anchor_generator={'octave_base_scale': 4, 'ratios': [0.5, 1.0, 2.0], 'scales_per_octave': 3, 'strides': [8, 16, 32, 64, 128], 'type': 'AnchorGenerator'}, square_anchor_generator={'ratios': [1.0], 'scales': [4], 'strides': [8, 16, 32, 64, 128], 'type': 'AnchorGenerator'}, conv_cfg=None, norm_cfg=None, bbox_coder={'num_buckets': 14, 'scale_factor': 3.0, 'type': 'BucketingBBoxCoder'}, reg_decoded_bbox=False, train_cfg=None, test_cfg=None, loss_cls={'alpha': 0.25, 'gamma': 2.0, 'loss_weight': 1.0, 'type': 'FocalLoss', 'use_sigmoid': True}, loss_bbox_cls={'loss_weight': 1.5, 'type': 'CrossEntropyLoss', 'use_sigmoid': True}, loss_bbox_reg={'beta': 0.1111111111111111, 'loss_weight': 1.5, 'type': 'SmoothL1Loss'}, init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'retina_cls', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'})`

Side-Aware Boundary Localization (SABL) for RetinaNet.

The anchor generation, assigning and sampling in SABLRetinaHead are the same as GuidedAnchorHead for guided anchoring.

Please refer to https://arxiv.org/abs/1912.04260 for more details.

Parameters

- **num_classes** (*int*) – Number of classes.
- **in_channels** (*int*) – Number of channels in the input feature map.
- **stacked_convs** (*int*) – Number of Convs for classification and regression branches. Defaults to 4.
- **feat_channels** (*int*) – Number of hidden channels. Defaults to 256.
- **approx_anchor_generator** (*dict*) – Config dict for approx generator.
- **square_anchor_generator** (*dict*) – Config dict for square generator.
- **conv_cfg** (*dict*) – Config dict for ConvModule. Defaults to None.
- **norm_cfg** (*dict*) – Config dict for Norm Layer. Defaults to None.
- **bbox_coder** (*dict*) – Config dict for bbox coder.
- **reg_decoded_bbox** (*bool*) – If true, the regression loss would be applied directly on decoded bounding boxes, converting both the predicted boxes and regression targets to absolute coordinates format. Default False. It should be *True* when using *IoULoss*, *GIoULoss*, or *DIoULoss* in the bbox head.
- **train_cfg** (*dict*) – Training config of SABLRetinaHead.

- **test_cfg** (*dict*) – Testing config of SABLRetinaHead.
- **loss_cls** (*dict*) – Config of classification loss.
- **loss_bbox_cls** (*dict*) – Config of classification loss for bbox branch.
- **loss_bbox_reg** (*dict*) – Config of regression loss for bbox branch.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

forward(*feats*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

get_anchors(*featmap_sizes*, *img_metas*, *device='cuda'*)

Get squares according to feature map sizes and guided anchors.

Parameters

- **featmap_sizes** (*list[tuple]*) – Multi-level feature map sizes.
- **img_metas** (*list[dict]*) – Image meta info.
- **device** (*torch.device | str*) – device for returned tensors

Returns square approxs of each image

Return type tuple

get_bboxes(*cls_scores*, *bbox_preds*, *img_metas*, *cfg=None*, *rescale=False*)

Transform network outputs of a batch into bbox results.

Note: When score_factors is not None, the cls_scores are usually multiplied by it then obtain the real score used in NMS, such as CenterNess in FCOS, IoU branch in ATSS.

Parameters

- **cls_scores** (*list[Tensor]*) – Classification scores for all scale levels, each is a 4D-tensor, has shape (batch_size, num_priors * num_classes, H, W).
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for all scale levels, each is a 4D-tensor, has shape (batch_size, num_priors * 4, H, W).
- **score_factors** (*list[Tensor], Optional*) – Score factor for all scale level, each is a 4D-tensor, has shape (batch_size, num_priors * 1, H, W). Default None.
- **img_metas** (*list[dict], Optional*) – Image meta info. Default None.
- **cfg** (*mmcv.Config, Optional*) – Test / postprocessing configuration, if None, test_cfg would be used. Default None.
- **rescale** (*bool*) – If True, return boxes in original image space. Default False.
- **with_nms** (*bool*) – If True, do nms before return boxes. Default True.

Returns

Each item in result_list is 2-tuple. The first item is an (n, 5) tensor, where the first 4 columns are bounding box positions (tl_x, tl_y, br_x, br_y) and the 5-th column is a score

between 0 and 1. The second item is a (n,) tensor where each item is the predicted class label of the corresponding box.

Return type list[list[Tensor, Tensor]]

get_target(*approx_list*, *inside_flag_list*, *square_list*, *gt_bboxes_list*, *img_metas*, *gt_bboxes_ignore_list=None*, *gt_labels_list=None*, *label_channels=None*, *sampling=True*, *unmap_outputs=True*)

Compute bucketing targets. :param approx_list: Multi level approxs of each image. :type approx_list: list[list] :param inside_flag_list: Multi level inside flags of each

image.

Parameters

- **square_list** (*list[list]*) – Multi level squares of each image.
- **gt_bboxes_list** (*list[Tensor]*) – Ground truth bboxes of each image.
- **img_metas** (*list[dict]*) – Meta info of each image.
- **gt_bboxes_ignore_list** (*list[Tensor]*) – ignore list of gt bboxes.
- **gt_bboxes_list** – Gt bboxes of each image.
- **label_channels** (*int*) – Channel of label.
- **sampling** (*bool*) – Sample Anchors or not.
- **unmap_outputs** (*bool*) – unmap outputs or not.

Returns

Returns a tuple containing learning targets.

- labels_list (list[Tensor]): Labels of each level.
- label_weights_list (list[Tensor]): Label weights of each level.
- bbox_cls_targets_list (list[Tensor]): BBox cls targets of each level.
- bbox_cls_weights_list (list[Tensor]): BBox cls weights of each level.
- bbox_reg_targets_list (list[Tensor]): BBox reg targets of each level.
- bbox_reg_weights_list (list[Tensor]): BBox reg weights of each level.
- num_total_pos (int): Number of positive samples in all images.
- num_total_neg (int): Number of negative samples in all images.

Return type tuple

loss(*cls_scores*, *bbox_preds*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)

Compute losses of the head.

class mmdet.models.dense_heads.**SOLOHead**(*num_classes*, *in_channels*, *feat_channels=256*, *stacked_convs=4*, *strides=(4, 8, 16, 32, 64)*, *scale_ranges=((8, 32), (16, 64), (32, 128), (64, 256), (128, 512))*, *pos_scale=0.2*, *num_grids=[40, 36, 24, 16, 12]*, *cls_down_index=0*, *loss_mask=None*, *loss_cls=None*, *norm_cfg={'num_groups': 32, 'requires_grad': True, 'type': 'GN'}*, *train_cfg=None*, *test_cfg=None*, *init_cfg=[{'type': 'Normal', 'layer': 'Conv2d', 'std': 0.01}, {'type': 'Normal', 'std': 0.01, 'bias_prob': 0.01, 'override': {'name': 'conv_mask_list'}}, {'type': 'Normal', 'std': 0.01, 'bias_prob': 0.01, 'override': {'name': 'conv_cls'}}]*)

SOLO mask head used in `SOLO: Segmenting Objects by Locations.

<https://arxiv.org/abs/1912.04488>`_

Parameters

- **num_classes** (*int*) – Number of categories excluding the background category.
- **in_channels** (*int*) – Number of channels in the input feature map.
- **feat_channels** (*int*) – Number of hidden channels. Used in child classes. Default: 256.
- **stacked_convs** (*int*) – Number of stacking convs of the head. Default: 4.
- **strides** (*tuple*) – Downsample factor of each feature map.
- **scale_ranges** (*tuple[tuple[int, int]]*) – Area range of multiple level masks, in the format [(min1, max1), (min2, max2), ...]. A range of (16, 64) means the area range between (16, 64).
- **pos_scale** (*float*) – Constant scale factor to control the center region.
- **num_grids** (*list[int]*) – Divided image into a uniform grids, each feature map has a different grid value. The number of output channels is grid ** 2. Default: [40, 36, 24, 16, 12].
- **cls_down_index** (*int*) – The index of downsample operation in classification branch. Default: 0.
- **loss_mask** (*dict*) – Config of mask loss.
- **loss_cls** (*dict*) – Config of classification loss.
- **norm_cfg** (*dict*) – dictionary to construct and config norm layer. Default: norm_cfg=dict(type='GN', num_groups=32,

  requires_grad=True).
- **train_cfg** (*dict*) – Training config of head.
- **test_cfg** (*dict*) – Testing config of head.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

forward(*feats*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

get_results(*mlvl_mask_preds*, *mlvl_cls_scores*, *img_metas*, ***kwargs*)
Get multi-image mask results.

Parameters

- **mlvl_mask_preds** (*list[Tensor]*) – Multi-level mask prediction. Each element in the list has shape (batch_size, num_grids**2 ,h ,w).
- **mlvl_cls_scores** (*list[Tensor]*) – Multi-level scores. Each element in the list has shape (batch_size, num_classes, num_grids ,num_grids).
- **img_metas** (*list[dict]*) – Meta information of all images.

Returns

Processed results of multiple images.Each `InstanceData` usually contains following keys.

- scores (Tensor): Classification scores, has shape (num_instance,).
- labels (Tensor): Has shape (num_instances,).
- masks (Tensor): Processed mask results, has shape (num_instances, h, w).

Return type list[`InstanceData`]

loss(*mlvl_mask_preds*, *mlvl_cls_preds*, *gt_labels*, *gt_masks*, *img_metas*, *gt_bboxes=None*, ***kwargs*)
Calculate the loss of total batch.

Parameters

- **mlvl_mask_preds** (*list[Tensor]*) – Multi-level mask prediction. Each element in the list has shape (batch_size, num_grids**2 ,h ,w).
- **mlvl_cls_preds** (*list[Tensor]*) – Multi-level scores. Each element in the list has shape (batch_size, num_classes, num_grids ,num_grids).
- **gt_labels** (*list[Tensor]*) – Labels of multiple images.
- **gt_masks** (*list[Tensor]*) – Ground truth masks of multiple images. Each has shape (num_instances, h, w).
- **img_metas** (*list[dict]*) – Meta information of multiple images.
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes of multiple images. Default: None.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

resize_feats(*feats*)
Downsample the first feat and upsample last feat in feats.

class mmdet.models.dense_heads.**SSDHead**(*num_classes=80*, *in_channels=(512, 1024, 512, 256, 256, 256)*, *stacked_convs=0*, *feat_channels=256*, *use_depthwise=False*, *conv_cfg=None*, *norm_cfg=None*, *act_cfg=None*, *anchor_generator={'basesize_ratio_range': (0.1, 0.9), 'input_size': 300, 'ratios': ([2], [2, 3], [2, 3], [2, 3], [2], [2]), 'scale_major': False, 'strides': [8, 16, 32, 64, 100, 300], 'type': 'SSDAnchorGenerator'}*, *bbox_coder={'clip_border': True, 'target_means': [0.0, 0.0, 0.0, 0.0], 'target_stds': [1.0, 1.0, 1.0, 1.0], 'type': 'DeltaXYWHBBoxCoder'}*, *reg_decoded_bbox=False*, *train_cfg=None*, *test_cfg=None*, *init_cfg={'bias': 0, 'distribution': 'uniform', 'layer': 'Conv2d', 'type': 'Xavier'}*)
SSD head used in https://arxiv.org/abs/1512.02325.

Parameters

- **num_classes** (*int*) – Number of categories excluding the background category.
- **in_channels** (*int*) – Number of channels in the input feature map.
- **stacked_convs** (*int*) – Number of conv layers in cls and reg tower. Default: 0.
- **feat_channels** (*int*) – Number of hidden channels when stacked_convs > 0. Default: 256.
- **use_depthwise** (*bool*) – Whether to use DepthwiseSeparableConv. Default: False.
- **conv_cfg** (*dict*) – Dictionary to construct and config conv layer. Default: None.
- **norm_cfg** (*dict*) – Dictionary to construct and config norm layer. Default: None.
- **act_cfg** (*dict*) – Dictionary to construct and config activation layer. Default: None.
- **anchor_generator** (*dict*) – Config dict for anchor generator
- **bbox_coder** (*dict*) – Config of bounding box coder.
- **reg_decoded_bbox** (*bool*) – If true, the regression loss would be applied directly on decoded bounding boxes, converting both the predicted boxes and regression targets to absolute coordinates format. Default False. It should be *True* when using *IoULoss*, *GIoULoss*, or *DIoULoss* in the bbox head.
- **train_cfg** (*dict*) – Training config of anchor head.
- **test_cfg** (*dict*) – Testing config of anchor head.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

forward(*feats*)

Forward features from the upstream network.

Parameters **feats** (*tuple[Tensor]*) – Features from the upstream network, each is a 4D-tensor.

Returns

cls_scores (list[Tensor]): Classification scores for all scale levels, each is a 4D-tensor, the channels number is num_anchors * num_classes.

bbox_preds (list[Tensor]): Box energies / deltas for all scale levels, each is a 4D-tensor, the channels number is num_anchors * 4.

Return type tuple

loss(*cls_scores*, *bbox_preds*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)

Compute losses of the head.

Parameters

- **cls_scores** (*list[Tensor]*) – Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level with shape (N, num_anchors * 4, H, W)
- **gt_bboxes** (*list[Tensor]*) – each item are the truth boxes for each image in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box

- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

loss_single(*cls_score*, *bbox_pred*, *anchor*, *labels*, *label_weights*, *bbox_targets*, *bbox_weights*, *num_total_samples*)

Compute loss of a single image.

Parameters

- **cls_score** (*Tensor*) – Box scores for eachimage Has shape (num_total_anchors, num_classes).
- **bbox_pred** (*Tensor*) – Box energies / deltas for each image level with shape (num_total_anchors, 4).
- **anchors** (*Tensor*) – Box reference for each scale level with shape (num_total_anchors, 4).
- **labels** (*Tensor*) – Labels of each anchors with shape (num_total_anchors,).
- **label_weights** (*Tensor*) – Label weights of each anchor with shape (num_total_anchors,)
- **bbox_targets** (*Tensor*) – BBox regression targets of each anchor weight shape (num_total_anchors, 4).
- **bbox_weights** (*Tensor*) – BBox regression loss weights of each anchor with shape (num_total_anchors, 4).
- **num_total_samples** (*int*) – If sampling, num total samples equal to the number of total anchors; Otherwise, it is the number of positive anchors.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

property num_anchors

Returns: list[int]: Number of base_anchors on each point of each level.

class mmdet.models.dense_heads.**StageCascadeRPNHead**(*in_channels*, *anchor_generator={'ratios': [1.0], 'scales': [8], 'strides': [4, 8, 16, 32, 64], 'type': 'AnchorGenerator'}*, *adapt_cfg={'dilation': 3, 'type': 'dilation'}*, *bridged_feature=False*, *with_cls=True*, *sampling=True*, *init_cfg=None*, ***kwargs*)

Stage of CascadeRPNHead.

Parameters

- **in_channels** (*int*) – Number of channels in the input feature map.
- **anchor_generator** (*dict*) – anchor generator config.
- **adapt_cfg** (*dict*) – adaptation config.
- **bridged_feature** (*bool, optional*) – whether update rpn feature. Default: False.
- **with_cls** (*bool, optional*) – whether use classification branch. Default: True.

- **sampling** (*bool, optional*) – whether use sampling. Default: True.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

anchor_offset(*anchor_list*, *anchor_strides*, *featmap_sizes*)
Get offset for deformable conv based on anchor shape NOTE: currently support deformable kernel_size=3 and dilation=1

Parameters

- **anchor_list** (*list[list[tensor]]*) – [NI, NLVL, NA, 4] list of multi-level anchors
- **anchor_strides** (*list[int]*) – anchor stride of each level

Returns

[NLVL, NA, 2, 18]: offset of DeformConv kernel.

Return type offset_list (list[tensor])

forward(*feats*, *offset_list=None*)
Forward function.

forward_single(*x*, *offset*)
Forward function of single scale.

get_bboxes(*anchor_list*, *cls_scores*, *bbox_preds*, *img_metas*, *cfg*, *rescale=False*)
Get proposal predict.

Parameters

- **anchor_list** (*list[list]*) – Multi level anchors of each image.
- **cls_scores** (*list[Tensor]*) – Classification scores for all scale levels, each is a 4D-tensor, has shape (batch_size, num_priors * num_classes, H, W).
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for all scale levels, each is a 4D-tensor, has shape (batch_size, num_priors * 4, H, W).
- **img_metas** (*list[dict], Optional*) – Image meta info. Default None.
- **cfg** (*mmcv.Config, Optional*) – Test / postprocessing configuration, if None, test_cfg would be used.
- **rescale** (*bool*) – If True, return boxes in original image space. Default: False.

Returns

Labeled boxes in shape (n, 5), where the first 4 columns are bounding box positions (tl_x, tl_y, br_x, br_y) and the 5-th column is a score between 0 and 1.

Return type Tensor

get_targets(*anchor_list*, *valid_flag_list*, *gt_bboxes*, *img_metas*, *featmap_sizes*, *gt_bboxes_ignore=None*, *label_channels=1*)
Compute regression and classification targets for anchors.

Parameters

- **anchor_list** (*list[list]*) – Multi level anchors of each image.
- **valid_flag_list** (*list[list]*) – Multi level valid flags of each image.
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes of each image.
- **img_metas** (*list[dict]*) – Meta info of each image.
- **featmap_sizes** (*list[Tensor]*) – Feature mapsize each level

- **gt_bboxes_ignore** (*list[Tensor]*) – Ignore bboxes of each images
- **label_channels** (*int*) – Channel of label.

Returns cls_reg_targets (tuple)

loss(*anchor_list*, *valid_flag_list*, *cls_scores*, *bbox_preds*, *gt_bboxes*, *img_metas*, *gt_bboxes_ignore=None*)
Compute losses of the head.

Parameters

- **anchor_list** (*list[list]*) – Multi level anchors of each image.
- **cls_scores** (*list[Tensor]*) – Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level with shape (N, num_anchors * 4, H, W)
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss. Default: None

Returns A dictionary of loss components.

Return type dict[str, Tensor]

loss_single(*cls_score*, *bbox_pred*, *anchors*, *labels*, *label_weights*, *bbox_targets*, *bbox_weights*, *num_total_samples*)
Loss function on single scale.

refine_bboxes(*anchor_list*, *bbox_preds*, *img_metas*)
Refine bboxes through stages.

region_targets(*anchor_list*, *valid_flag_list*, *gt_bboxes_list*, *img_metas*, *featmap_sizes*, *gt_bboxes_ignore_list=None*, *gt_labels_list=None*, *label_channels=1*, *unmap_outputs=True*)
See *StageCascadeRPNHead.get_targets()*.

class mmdet.models.dense_heads.**VFNetHead**(*num_classes*, *in_channels*, *regress_ranges=((- 1, 64), (64, 128), (128, 256), (256, 512), (512, 100000000.0))*, *center_sampling=False*, *center_sample_radius=1.5*, *sync_num_pos=True*, *gradient_mul=0.1*, *bbox_norm_type='reg_denom'*, *loss_cls_fl={'alpha': 0.25, 'gamma': 2.0, 'loss_weight': 1.0, 'type': 'FocalLoss', 'use_sigmoid': True}*, *use_vfl=True*, *loss_cls={'alpha': 0.75, 'gamma': 2.0, 'iou_weighted': True, 'loss_weight': 1.0, 'type': 'VarifocalLoss', 'use_sigmoid': True}*, *loss_bbox={'loss_weight': 1.5, 'type': 'GIoULoss'}*, *loss_bbox_refine={'loss_weight': 2.0, 'type': 'GIoULoss'}*, *norm_cfg={'num_groups': 32, 'requires_grad': True, 'type': 'GN'}*, *use_atss=True*, *reg_decoded_bbox=True*, *anchor_generator={'center_offset': 0.0, 'octave_base_scale': 8, 'ratios': [1.0], 'scales_per_octave': 1, 'strides': [8, 16, 32, 64, 128], 'type': 'AnchorGenerator'}*, *init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'vfnet_cls', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'}*, ***kwargs*)

Head of **`VarifocalNet (VFNet): An IoU-aware Dense Object Detector.<https://arxiv.org/abs/2008.13367>`_**.

The VFNet predicts IoU-aware classification scores which mix the object presence confidence and object localization accuracy as the detection score. It is built on the FCOS architecture and uses ATSS for defining positive/negative training examples. The VFNet is trained with Varifocal Loss and empolys star-shaped deformable convolution to extract features for a bbox.

Parameters

- **num_classes** (*int*) – Number of categories excluding the background category.
- **in_channels** (*int*) – Number of channels in the input feature map.
- **regress_ranges** (*tuple[tuple[int, int]]*) – Regress range of multiple level points.
- **center_sampling** (*bool*) – If true, use center sampling. Default: False.
- **center_sample_radius** (*float*) – Radius of center sampling. Default: 1.5.
- **sync_num_pos** (*bool*) – If true, synchronize the number of positive examples across GPUs. Default: True
- **gradient_mul** (*float*) – The multiplier to gradients from bbox refinement and recognition. Default: 0.1.
- **bbox_norm_type** (*str*) – The bbox normalization type, ‘reg_denom’ or ‘stride’. Default: reg_denom
- **loss_cls_fl** (*dict*) – Config of focal loss.
- **use_vfl** (*bool*) – If true, use varifocal loss for training. Default: True.
- **loss_cls** (*dict*) – Config of varifocal loss.
- **loss_bbox** (*dict*) – Config of localization loss, GIoU Loss.
- **loss_bbox** – Config of localization refinement loss, GIoU Loss.
- **norm_cfg** (*dict*) – dictionary to construct and config norm layer. Default: norm_cfg=dict(type=’GN’, num_groups=32, requires_grad=True).
- **use_atss** (*bool*) – If true, use ATSS to define positive/negative examples. Default: True.

- **anchor_generator** (*dict*) – Config of anchor generator for ATSS.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

Example

```
>>> self = VFNetHead(11, 7)
>>> feats = [torch.rand(1, 7, s, s) for s in [4, 8, 16, 32, 64]]
>>> cls_score, bbox_pred, bbox_pred_refine= self.forward(feats)
>>> assert len(cls_score) == len(self.scales)
```

forward(*feats*)

Forward features from the upstream network.

Parameters **feats** (*tuple[Tensor]*) – Features from the upstream network, each is a 4D-tensor.

Returns

cls_scores (list[Tensor]): Box iou-aware scores for each scale level, each is a 4D-tensor, the channel number is num_points * num_classes.

bbox_preds (list[Tensor]): Box offsets for each scale level, each is a 4D-tensor, the channel number is num_points * 4.

bbox_preds_refine (list[Tensor]): Refined Box offsets for each scale level, each is a 4D-tensor, the channel number is num_points * 4.

Return type tuple

forward_single(*x*, *scale*, *scale_refine*, *stride*, *reg_denom*)

Forward features of a single scale level.

Parameters

- **x** (*Tensor*) – FPN feature maps of the specified stride.
- **(** (*scale_refine*) – obj: *mmcv.cnn.Scale*): Learnable scale module to resize the bbox prediction.
- **(** – obj: *mmcv.cnn.Scale*): Learnable scale module to resize the refined bbox prediction.
- **stride** (*int*) – The corresponding stride for feature maps, used to normalize the bbox prediction when bbox_norm_type = 'stride'.
- **reg_denom** (*int*) – The corresponding regression range for feature maps, only used to normalize the bbox prediction when bbox_norm_type = 'reg_denom'.

Returns

iou-aware cls scores for each box, bbox predictions and refined bbox predictions of input feature maps.

Return type tuple

get_anchors(*featmap_sizes*, *img_metas*, *device='cuda'*)

Get anchors according to feature map sizes.

Parameters

- **featmap_sizes** (*list[tuple]*) – Multi-level feature map sizes.
- **img_metas** (*list[dict]*) – Image meta info.

- **device** (*torch.device | str*) – Device for returned tensors

Returns anchor_list (list[Tensor]): Anchors of each image. valid_flag_list (list[Tensor]): Valid flags of each image.

Return type tuple

get_atss_targets(*cls_scores*, *mlvl_points*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)
A wrapper for computing ATSS targets for points in multiple images.

Parameters

- **cls_scores** (*list[Tensor]*) – Box iou-aware scores for each scale level with shape (N, num_points * num_classes, H, W).
- **mlvl_points** (*list[Tensor]*) – Points of each fpn level, each has shape (num_points, 2).
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes of each image, each has shape (num_gt, 4).
- **gt_labels** (*list[Tensor]*) – Ground truth labels of each box, each has shape (num_gt,).
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | Tensor*) – Ground truth bboxes to be ignored, shape (num_ignored_gts, 4). Default: None.

Returns

labels_list (list[Tensor]): Labels of each level. label_weights (Tensor): Label weights of all levels. bbox_targets_list (list[Tensor]): Regression targets of each

level, (l, t, r, b).

bbox_weights (Tensor): Bbox weights of all levels.

Return type tuple

get_fcos_targets(*points*, *gt_bboxes_list*, *gt_labels_list*)
Compute FCOS regression and classification targets for points in multiple images.

Parameters

- **points** (*list[Tensor]*) – Points of each fpn level, each has shape (num_points, 2).
- **gt_bboxes_list** (*list[Tensor]*) – Ground truth bboxes of each image, each has shape (num_gt, 4).
- **gt_labels_list** (*list[Tensor]*) – Ground truth labels of each box, each has shape (num_gt,).

Returns labels (list[Tensor]): Labels of each level. label_weights: None, to be compatible with ATSS targets. bbox_targets (list[Tensor]): BBox targets of each level. bbox_weights: None, to be compatible with ATSS targets.

Return type tuple

get_targets(*cls_scores*, *mlvl_points*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore*)
A wrapper for computing ATSS and FCOS targets for points in multiple images.

Parameters

- **cls_scores** (*list[Tensor]*) – Box iou-aware scores for each scale level with shape (N, num_points * num_classes, H, W).

- **mlvl_points** (*list[Tensor]*) – Points of each fpn level, each has shape (num_points, 2).
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes of each image, each has shape (num_gt, 4).
- **gt_labels** (*list[Tensor]*) – Ground truth labels of each box, each has shape (num_gt,).
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | Tensor*) – Ground truth bboxes to be ignored, shape (num_ignored_gts, 4).

Returns

labels_list (list[Tensor]): Labels of each level. label_weights (Tensor/None): Label weights of all levels. bbox_targets_list (list[Tensor]): Regression targets of each

level, (l, t, r, b).

bbox_weights (Tensor/None): Bbox weights of all levels.

Return type tuple

loss(*cls_scores*, *bbox_preds*, *bbox_preds_refine*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)

Compute loss of the head.

Parameters

- **cls_scores** (*list[Tensor]*) – Box iou-aware scores for each scale level, each is a 4D-tensor, the channel number is num_points * num_classes.
- **bbox_preds** (*list[Tensor]*) – Box offsets for each scale level, each is a 4D-tensor, the channel number is num_points * 4.
- **bbox_preds_refine** (*list[Tensor]*) – Refined Box offsets for each scale level, each is a 4D-tensor, the channel number is num_points * 4.
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss. Default: None.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

property num_anchors

Returns: int: Number of anchors on each point of feature map.

star_dcn_offset(*bbox_pred*, *gradient_mul*, *stride*)

Compute the star deformable conv offsets.

Parameters

- **bbox_pred** (*Tensor*) – Predicted bbox distance offsets (l, r, t, b).
- **gradient_mul** (*float*) – Gradient multiplier.

- **stride** (*int*) – The corresponding stride for feature maps, used to project the bbox onto the feature map.

Returns The offsets for deformable convolution.

Return type dcn_offsets (Tensor)

transform_bbox_targets(*decoded_bboxes*, *mlvl_points*, *num_imgs*)

Transform bbox_targets (x1, y1, x2, y2) into (l, t, r, b) format.

Parameters

- **decoded_bboxes** (*list[Tensor]*) – Regression targets of each level, in the form of (x1, y1, x2, y2).
- **mlvl_points** (*list[Tensor]*) – Points of each fpn level, each has shape (num_points, 2).
- **num_imgs** (*int*) – the number of images in a batch.

Returns

Regression targets of each level in the form of (l, t, r, b).

Return type bbox_targets (list[Tensor])

class mmdet.models.dense_heads.**YOLACTHead**(*num_classes*, *in_channels*, *anchor_generator={'octave_base_scale': 3, 'ratios': [0.5, 1.0, 2.0], 'scales_per_octave': 1, 'strides': [8, 16, 32, 64, 128], 'type': 'AnchorGenerator'}*, *loss_cls={'loss_weight': 1.0, 'reduction': 'none', 'type': 'CrossEntropyLoss', 'use_sigmoid': False}*, *loss_bbox={'beta': 1.0, 'loss_weight': 1.5, 'type': 'SmoothL1Loss'}*, *num_head_convs=1*, *num_protos=32*, *use_ohem=True*, *conv_cfg=None*, *norm_cfg=None*, *init_cfg={'bias': 0, 'distribution': 'uniform', 'layer': 'Conv2d', 'type': 'Xavier'}*, ***kwargs*)

YOLACT box head used in https://arxiv.org/abs/1904.02689.

Note that YOLACT head is a light version of RetinaNet head. Four differences are described as follows:

1. YOLACT box head has three-times fewer anchors.
2. YOLACT box head shares the convs for box and cls branches.
3. YOLACT box head uses OHEM instead of Focal loss.
4. YOLACT box head predicts a set of mask coefficients for each box.

Parameters

- **num_classes** (*int*) – Number of categories excluding the background category.
- **in_channels** (*int*) – Number of channels in the input feature map.
- **anchor_generator** (*dict*) – Config dict for anchor generator
- **loss_cls** (*dict*) – Config of classification loss.
- **loss_bbox** (*dict*) – Config of localization loss.
- **num_head_convs** (*int*) – Number of the conv layers shared by box and cls branches.
- **num_protos** (*int*) – Number of the mask coefficients.
- **use_ohem** (*bool*) – If true, `loss_single_OHEM` will be used for cls loss calculation. If false, `loss_single` will be used.

- **conv_cfg** (*dict*) – Dictionary to construct and config conv layer.
- **norm_cfg** (*dict*) – Dictionary to construct and config norm layer.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

forward_single(*x*)

Forward feature of a single scale level.

Parameters **x** (*Tensor*) – Features of a single scale level.

Returns cls_score (Tensor): Cls scores for a single scale level the channels number is num_anchors * num_classes. bbox_pred (Tensor): Box energies / deltas for a single scale level, the channels number is num_anchors * 4. coeff_pred (Tensor): Mask coefficients for a single scale level, the channels number is num_anchors * num_protos.

Return type tuple

get_bboxes(*cls_scores*, *bbox_preds*, *coeff_preds*, *img_metas*, *cfg=None*, *rescale=False*)

"Similar to func:AnchorHead.get_bboxes, but additionally processes coeff_preds.

Parameters

- **cls_scores** (*list[Tensor]*) – Box scores for each scale level with shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level with shape (N, num_anchors * 4, H, W)
- **coeff_preds** (*list[Tensor]*) – Mask coefficients for each scale level with shape (N, num_anchors * num_protos, H, W)
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **cfg** (*mmcv.Config | None*) – Test / postprocessing configuration, if None, test_cfg would be used
- **rescale** (*bool*) – If True, return boxes in original image space. Default: False.

Returns

Each item in result_list is a 3-tuple. The first item is an (n, 5) tensor, where the first 4 columns are bounding box positions (tl_x, tl_y, br_x, br_y) and the 5-th column is a score between 0 and 1. The second item is an (n,) tensor where each item is the predicted class label of the corresponding box. The third item is an (n, num_protos) tensor where each item is the predicted mask coefficients of instance inside the corresponding box.

Return type list[tuple[Tensor, Tensor, Tensor]]

loss(*cls_scores*, *bbox_preds*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)

A combination of the func:AnchorHead.loss and func:SSDHead.loss.

When `self.use_ohem == True`, it functions like `SSDHead.loss`, otherwise, it follows `AnchorHead.loss`. Besides, it additionally returns `sampling_results`.

Parameters

- **cls_scores** (*list[Tensor]*) – Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level with shape (N, num_anchors * 4, H, W)

- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – Class indices corresponding to each box
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | list[Tensor]*) – Specify which bounding boxes can be ignored when computing the loss. Default: None

Returns dict[str, Tensor]: A dictionary of loss components. List[:obj:SamplingResult]: Sampler results for each image.

Return type tuple

loss_single_OHEM(*cls_score*, *bbox_pred*, *anchors*, *labels*, *label_weights*, *bbox_targets*, *bbox_weights*, *num_total_samples*)

"See func:SSDHead.loss.

class mmdet.models.dense_heads.**YOLACTProtonet**(*num_classes*, *in_channels=256*, *proto_channels=(256, 256, 256, None, 256, 32)*, *proto_kernel_sizes=(3, 3, 3, -2, 3, 1)*, *include_last_relu=True*, *num_protos=32*, *loss_mask_weight=1.0*, *max_masks_to_train=100*, *init_cfg={'distribution': 'uniform', 'override': {'name': 'protonet'}, 'type': 'Xavier'}*)

YOLACT mask head used in https://arxiv.org/abs/1904.02689.

This head outputs the mask prototypes for YOLACT.

Parameters

- **in_channels** (*int*) – Number of channels in the input feature map.
- **proto_channels** (*tuple[int]*) – Output channels of protonet convs.
- **proto_kernel_sizes** (*tuple[int]*) – Kernel sizes of protonet convs.
- **include_last_relu** (*Bool*) – If keep the last relu of protonet.
- **num_protos** (*int*) – Number of prototypes.
- **num_classes** (*int*) – Number of categories excluding the background category.
- **loss_mask_weight** (*float*) – Reweight the mask loss by this factor.
- **max_masks_to_train** (*int*) – Maximum number of masks to train for each image.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

crop(*masks*, *boxes*, *padding=1*)

Crop predicted masks by zeroing out everything not in the predicted bbox.

Parameters

- **masks** (*Tensor*) – shape [H, W, N].
- **boxes** (*Tensor*) – bbox coords in relative point form with shape [N, 4].

Returns The cropped masks.

Return type Tensor

forward(*x*, *coeff_pred*, *bboxes*, *img_meta*, *sampling_results=None*)

Forward feature from the upstream network to get prototypes and linearly combine the prototypes, using masks coefficients, into instance masks. Finally, crop the instance masks with given bboxes.

Parameters

- **x** (*Tensor*) – Feature from the upstream network, which is a 4D-tensor.
- **coeff_pred** (*list[Tensor]*) – Mask coefficients for each scale level with shape (N, num_anchors * num_protos, H, W).
- **bboxes** (*list[Tensor]*) – Box used for cropping with shape (N, num_anchors * 4, H, W). During training, they are ground truth boxes. During testing, they are predicted boxes.
- **img_meta** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **sampling_results** (List[:obj:SamplingResult]) – Sampler results for each image.

Returns Predicted instance segmentation masks.

Return type list[Tensor]

get_seg_masks(*mask_pred*, *label_pred*, *img_meta*, *rescale*)
Resize, binarize, and format the instance mask predictions.

Parameters

- **mask_pred** (*Tensor*) – shape (N, H, W).
- **label_pred** (*Tensor*) – shape (N,).
- **img_meta** (*dict*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **rescale** (*bool*) – If rescale is False, then returned masks will fit the scale of imgs[0].

Returns Mask predictions grouped by their predicted classes.

Return type list[ndarray]

get_targets(*mask_pred*, *gt_masks*, *pos_assigned_gt_inds*)
Compute instance segmentation targets for each image.

Parameters

- **mask_pred** (*Tensor*) – Predicted prototypes with shape (num_classes, H, W).
- **gt_masks** (*Tensor*) – Ground truth masks for each image with the same shape of the input image.
- **pos_assigned_gt_inds** (*Tensor*) – GT indices of the corresponding positive samples.

Returns

Instance segmentation targets with shape (num_instances, H, W).

Return type Tensor

loss(*mask_pred*, *gt_masks*, *gt_bboxes*, *img_meta*, *sampling_results*)
Compute loss of the head.

Parameters

- **mask_pred** (*list[Tensor]*) – Predicted prototypes with shape (num_classes, H, W).
- **gt_masks** (*list[Tensor]*) – Ground truth masks for each image with the same shape of the input image.
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.

- **img_meta** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **sampling_results** (List[:obj:SamplingResult]) – Sampler results for each image.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

sanitize_coordinates(*x1*, *x2*, *img_size*, *padding=0*, *cast=True*)

Sanitizes the input coordinates so that x1 < x2, x1 != x2, x1 >= 0, and x2 <= image_size. Also converts from relative to absolute coordinates and casts the results to long tensors.

Warning: this does things in-place behind the scenes so copy if necessary.

Parameters

- **_x1** (*Tensor*) – shape (N,).
- **_x2** (*Tensor*) – shape (N,).
- **img_size** (*int*) – Size of the input image.
- **padding** (*int*) – x1 >= padding, x2 <= image_size-padding.
- **cast** (*bool*) – If cast is false, the result won't be cast to longs.

Returns x1 (Tensor): Sanitized _x1. x2 (Tensor): Sanitized _x2.

Return type tuple

simple_test(*feats*, *det_bboxes*, *det_labels*, *det_coeffs*, *img_metas*, *rescale=False*)

Test function without test-time augmentation.

Parameters

- **feats** (*tuple[torch.Tensor]*) – Multi-level features from the upstream network, each is a 4D-tensor.
- **det_bboxes** (*list[Tensor]*) – BBox results of each image. each element is (n, 5) tensor, where 5 represent (tl_x, tl_y, br_x, br_y, score) and the score between 0 and 1.
- **det_labels** (*list[Tensor]*) – BBox results of each image. each element is (n,) tensor, each element represents the class label of the corresponding box.
- **det_coeffs** (*list[Tensor]*) – BBox coefficient of each image. each element is (n, m) tensor, m is vector length.
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **rescale** (*bool, optional*) – Whether to rescale the results. Defaults to False.

Returns

encoded masks. The c-th item in the outer list corresponds to the c-th class. Given the c-th outer list, the i-th item in that inner list is the mask for the i-th box with class label c.

Return type list[list]

class mmdet.models.dense_heads.**YOLACTSegmHead**(*num_classes*, *in_channels=256*, *loss_segm={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, *init_cfg={'distribution': 'uniform', 'override': {'name': 'segm_conv'}, 'type': 'Xavier'}*)

YOLACT segmentation head used in https://arxiv.org/abs/1904.02689.

Apply a semantic segmentation loss on feature space using layers that are only evaluated during training to increase performance with no speed penalty.

Parameters

- **in_channels** (*int*) – Number of channels in the input feature map.
- **num_classes** (*int*) – Number of categories excluding the background category.
- **loss_segm** (*dict*) – Config of semantic segmentation loss.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

forward(*x*)

Forward feature from the upstream network.

Parameters **x** (*Tensor*) – Feature from the upstream network, which is a 4D-tensor.

Returns

Predicted semantic segmentation map with shape (N, num_classes, H, W).

Return type Tensor

get_targets(*segm_pred*, *gt_masks*, *gt_labels*)

Compute semantic segmentation targets for each image.

Parameters

- **segm_pred** (*Tensor*) – Predicted semantic segmentation map with shape (num_classes, H, W).
- **gt_masks** (*Tensor*) – Ground truth masks for each image with the same shape of the input image.
- **gt_labels** (*Tensor*) – Class indices corresponding to each box.

Returns

Semantic segmentation targets with shape (num_classes, H, W).

Return type Tensor

loss(*segm_pred*, *gt_masks*, *gt_labels*)

Compute loss of the head.

Parameters

- **segm_pred** (*list[Tensor]*) – Predicted semantic segmentation map with shape (N, num_classes, H, W).
- **gt_masks** (*list[Tensor]*) – Ground truth masks for each image with the same shape of the input image.
- **gt_labels** (*list[Tensor]*) – Class indices corresponding to each box.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

simple_test(*feats*, *img_metas*, *rescale=False*)
Test function without test-time augmentation.

class mmdet.models.dense_heads.**YOLOFHead**(*num_classes*, *in_channels*, *num_cls_convs=2*, *num_reg_convs=4*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, ***kwargs*)
YOLOFHead Paper link: https://arxiv.org/abs/2103.09460.

Parameters

- **num_classes** (*int*) – The number of object classes (w/o background)
- **in_channels** (*List[int]*) – The number of input channels per scale.
- **cls_num_convs** (*int*) – The number of convolutions of cls branch. Default 2.
- **reg_num_convs** (*int*) – The number of convolutions of reg branch. Default 4.
- **norm_cfg** (*dict*) – Dictionary to construct and config norm layer.

forward_single(*feature*)
Forward feature of a single scale level.

Parameters **x** (*Tensor*) – Features of a single scale level.

Returns cls_score (Tensor): Cls scores for a single scale level the channels number is num_base_priors * num_classes. bbox_pred (Tensor): Box energies / deltas for a single scale level, the channels number is num_base_priors * 4.

Return type tuple

get_targets(*cls_scores_list*, *bbox_preds_list*, *anchor_list*, *valid_flag_list*, *gt_bboxes_list*, *img_metas*, *gt_bboxes_ignore_list=None*, *gt_labels_list=None*, *label_channels=1*, *unmap_outputs=True*)
Compute regression and classification targets for anchors in multiple images.

Parameters

- **cls_scores_list** (*list[Tensor]*) – each image. each is a 4D-tensor, the shape is (h * w, num_anchors * num_classes).
- **bbox_preds_list** (*list[Tensor]*) – each is a 4D-tensor, the shape is (h * w, num_anchors * 4).
- **anchor_list** (*list[Tensor]*) – Anchors of each image. Each element of is a tensor of shape (h * w * num_anchors, 4).
- **valid_flag_list** (*list[Tensor]*) – Valid flags of each image. Each element of is a tensor of shape (h * w * num_anchors,)
- **gt_bboxes_list** (*list[Tensor]*) – Ground truth bboxes of each image.
- **img_metas** (*list[dict]*) – Meta info of each image.
- **gt_bboxes_ignore_list** (*list[Tensor]*) – Ground truth bboxes to be ignored.
- **gt_labels_list** (*list[Tensor]*) – Ground truth labels of each box.
- **label_channels** (*int*) – Channel of label.
- **unmap_outputs** (*bool*) – Whether to map outputs back to the original set of anchors.

Returns

Usually returns a tuple containing learning targets.

- batch_labels (Tensor): Label of all images. Each element of is a tensor of shape (batch, h * w * num_anchors)

- batch_label_weights (Tensor): Label weights of all images of is a tensor of shape (batch, h * w * num_anchors)
- num_total_pos (int): Number of positive samples in all images.
- num_total_neg (int): Number of negative samples in all images.

additional_returns: This function enables user-defined returns from *self._get_targets_single*. These returns are currently refined to properties at each feature map (i.e. having HxW dimension). The results will be concatenated after the end

Return type tuple

`init_weights`()
Initialize the weights.

`loss`(*cls_scores*, *bbox_preds*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)
Compute losses of the head.

Parameters

- **cls_scores** (*list[Tensor]*) – Box scores for each scale level Has shape (batch, num_anchors * num_classes, h, w)
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level with shape (batch, num_anchors * 4, h, w)
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss. Default: None

Returns A dictionary of loss components.

Return type dict[str, Tensor]

`class mmdet.models.dense_heads.`**YOLOV3Head**(*num_classes*, *in_channels*, *out_channels=(1024, 512, 256)*, *anchor_generator={'base_sizes': [[(116, 90), (156, 198), (373, 326)], [(30, 61), (62, 45), (59, 119)], [(10, 13), (16, 30), (33, 23)]], 'strides': [32, 16, 8], 'type': 'YOLOAnchorGenerator'}*, *bbox_coder={'type': 'YOLOBBoxCoder'}*, *featmap_strides=[32, 16, 8]*, *one_hot_smoother=0.0*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *act_cfg={'negative_slope': 0.1, 'type': 'LeakyReLU'}*, *loss_cls={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, *loss_conf={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, *loss_xy={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, *loss_wh={'loss_weight': 1.0, 'type': 'MSELoss'}*, *train_cfg=None*, *test_cfg=None*, *init_cfg={'override': {'name': 'convs_pred'}, 'std': 0.01, 'type': 'Normal'}*)
YOLOV3Head Paper link: https://arxiv.org/abs/1804.02767.

Parameters

- **num_classes** (*int*) – The number of object classes (w/o background)
- **in_channels** (*List[int]*) – Number of input channels per scale.
- **out_channels** (*List[int]*) – The number of output channels per scale before the final 1x1 layer. Default: (1024, 512, 256).
- **anchor_generator** (*dict*) – Config dict for anchor generator
- **bbox_coder** (*dict*) – Config of bounding box coder.
- **featmap_strides** (*List[int]*) – The stride of each scale. Should be in descending order. Default: (32, 16, 8).
- **one_hot_smoother** (*float*) – Set a non-zero value to enable label-smooth Default: 0.
- **conv_cfg** (*dict*) – Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict*) – Dictionary to construct and config norm layer. Default: dict(type='BN', requires_grad=True)
- **act_cfg** (*dict*) – Config dict for activation layer. Default: dict(type='LeakyReLU', negative_slope=0.1).
- **loss_cls** (*dict*) – Config of classification loss.
- **loss_conf** (*dict*) – Config of confidence loss.
- **loss_xy** (*dict*) – Config of xy coordinate loss.
- **loss_wh** (*dict*) – Config of wh coordinate loss.
- **train_cfg** (*dict*) – Training config of YOLOV3 head. Default: None.
- **test_cfg** (*dict*) – Testing config of YOLOV3 head. Default: None.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

aug_test(*feats*, *img_metas*, *rescale=False*)

Test function with test time augmentation.

Parameters

- **feats** (*list[Tensor]*) – the outer list indicates test-time augmentations and inner Tensor should have a shape NxCxHxW, which contains features for all images in the batch.
- **img_metas** (*list[list[dict]]*) – the outer list indicates test-time augs (multiscale, flip, etc.) and the inner list indicates images in a batch. each dict has image information.
- **rescale** (*bool, optional*) – Whether to rescale the results. Defaults to False.

Returns bbox results of each class

Return type list[ndarray]

forward(*feats*)

Forward features from the upstream network.

Parameters **feats** (*tuple[Tensor]*) – Features from the upstream network, each is a 4D-tensor.

Returns

A tuple of multi-level predication map, each is a 4D-tensor of shape (batch_size, 5+num_classes, height, width).

Return type tuple[Tensor]

get_bboxes(*pred_maps*, *img_metas*, *cfg=None*, *rescale=False*, *with_nms=True*)
Transform network output for a batch into bbox predictions. It has been accelerated since PR #5991.

Parameters

- **pred_maps** (*list[Tensor]*) – Raw predictions for a batch of images.
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **cfg** (*mmcv.Config | None*) – Test / postprocessing configuration, if None, test_cfg would be used. Default: None.
- **rescale** (*bool*) – If True, return boxes in original image space. Default: False.
- **with_nms** (*bool*) – If True, do nms before return boxes. Default: True.

Returns

Each item in result_list is 2-tuple. The first item is an (n, 5) tensor, where 5 represent (tl_x, tl_y, br_x, br_y, score) and the score between 0 and 1. The shape of the second tensor in the tuple is (n,), and each element represents the class label of the corresponding box.

Return type list[tuple[Tensor, Tensor]]

get_targets(*anchor_list*, *responsible_flag_list*, *gt_bboxes_list*, *gt_labels_list*)
Compute target maps for anchors in multiple images.

Parameters

- **anchor_list** (*list[list[Tensor]]*) – Multi level anchors of each image. The outer list indicates images, and the inner list corresponds to feature levels of the image. Each element of the inner list is a tensor of shape (num_total_anchors, 4).
- **responsible_flag_list** (*list[list[Tensor]]*) – Multi level responsible flags of each image. Each element is a tensor of shape (num_total_anchors,)
- **gt_bboxes_list** (*list[Tensor]*) – Ground truth bboxes of each image.
- **gt_labels_list** (*list[Tensor]*) – Ground truth labels of each box.

Returns

Usually returns a tuple containing learning targets.

- target_map_list (list[Tensor]): Target map of each level.
- neg_map_list (list[Tensor]): Negative map of each level.

Return type tuple

init_weights()
Initialize the weights.

loss(*pred_maps*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)
Compute loss of the head.

Parameters

- **pred_maps** (*list[Tensor]*) – Prediction map for each scale level, shape (N, num_anchors * num_attrib, H, W)
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.

- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss.

Returns A dictionary of loss components.

Return type dict[str, Tensor]

loss_single(*pred_map*, *target_map*, *neg_map*)
Compute loss of a single image from a batch.

Parameters

- **pred_map** (*Tensor*) – Raw predictions for a single level.
- **target_map** (*Tensor*) – The Ground-Truth target for a single level.
- **neg_map** (*Tensor*) – The negative masks for a single level.

Returns loss_cls (Tensor): Classification loss. loss_conf (Tensor): Confidence loss. loss_xy (Tensor): Regression loss of x, y coordinate. loss_wh (Tensor): Regression loss of w, h coordinate.

Return type tuple

property num_anchors
Returns: int: Number of anchors on each point of feature map.

property num_attrib
number of attributes in pred_map, bboxes (4) + objectness (1) + num_classes

Type int

onnx_export(*pred_maps*, *img_metas*, *with_nms=True*)
Transform network output for a batch into bbox predictions.

Parameters

- **cls_scores** (*list[Tensor]*) – Box scores for each scale level with shape (N, num_points * num_classes, H, W).
- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level with shape (N, num_points * 4, H, W).
- **score_factors** (*list[Tensor]*) – score_factors for each s cale level with shape (N, num_points * 1, H, W). Default: None.
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc. Default: None.
- **with_nms** (*bool*) – Whether apply nms to the bboxes. Default: True.

Returns When *with_nms* is True, it is tuple[Tensor, Tensor], first tensor bboxes with shape [N, num_det, 5], 5 arrange as (x1, y1, x2, y2, score) and second element is class labels of shape [N, num_det]. When *with_nms* is False, first tensor is bboxes with shape [N, num_det, 4], second tensor is raw score has shape [N, num_det, num_classes].

Return type tuple[Tensor, Tensor] | list[tuple]

class mmdet.models.dense_heads.**YOLOXHead**(*num_classes*, *in_channels*, *feat_channels=256*, *stacked_convs=2*, *strides=[8, 16, 32]*, *use_depthwise=False*, *dcn_on_last_conv=False*, *conv_bias='auto'*, *conv_cfg=None*, *norm_cfg={'eps': 0.001, 'momentum': 0.03, 'type': 'BN'}*, *act_cfg={'type': 'Swish'}*, *loss_cls={'loss_weight': 1.0, 'reduction': 'sum', 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, *loss_bbox={'eps': 1e-16, 'loss_weight': 5.0, 'mode': 'square', 'reduction': 'sum', 'type': 'IoULoss'}*, *loss_obj={'loss_weight': 1.0, 'reduction': 'sum', 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, *loss_l1={'loss_weight': 1.0, 'reduction': 'sum', 'type': 'L1Loss'}*, *train_cfg=None*, *test_cfg=None*, *init_cfg={'a': 2.23606797749979, 'distribution': 'uniform', 'layer': 'Conv2d', 'mode': 'fan_in', 'nonlinearity': 'leaky_relu', 'type': 'Kaiming'}*)

YOLOXHead head used in YOLOX.

Parameters

- **num_classes** (*int*) – Number of categories excluding the background category.
- **in_channels** (*int*) – Number of channels in the input feature map.
- **feat_channels** (*int*) – Number of hidden channels in stacking convs. Default: 256
- **stacked_convs** (*int*) – Number of stacking convs of the head. Default: 2.
- **strides** (*tuple*) – Downsample factor of each feature map.
- **use_depthwise** (*bool*) – Whether to depthwise separable convolution in blocks. Default: False
- **dcn_on_last_conv** (*bool*) – If true, use dcn in the last layer of towers. Default: False.
- **conv_bias** (*bool | str*) – If specified as *auto*, it will be decided by the norm_cfg. Bias of conv will be set as True if *norm_cfg* is None, otherwise False. Default: "auto".
- **conv_cfg** (*dict*) – Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: None.
- **act_cfg** (*dict*) – Config dict for activation layer. Default: None.
- **loss_cls** (*dict*) – Config of classification loss.
- **loss_bbox** (*dict*) – Config of localization loss.
- **loss_obj** (*dict*) – Config of objectness loss.
- **loss_l1** (*dict*) – Config of L1 loss.
- **train_cfg** (*dict*) – Training config of anchor head.
- **test_cfg** (*dict*) – Testing config of anchor head.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

forward(*feats*)

Forward features from the upstream network.

Parameters **feats** (*tuple[Tensor]*) – Features from the upstream network, each is a 4D-tensor.

Returns

A tuple of multi-level predication map, each is a 4D-tensor of shape (batch_size, 5+num_classes, height, width).

Return type tuple[Tensor]

forward_single(*x*, *cls_convs*, *reg_convs*, *conv_cls*, *conv_reg*, *conv_obj*)
Forward feature of a single scale level.

get_bboxes(*cls_scores*, *bbox_preds*, *objectnesses*, *img_metas=None*, *cfg=None*, *rescale=False*, *with_nms=True*)
Transform network outputs of a batch into bbox results. :param cls_scores: Classification scores for all

scale levels, each is a 4D-tensor, has shape (batch_size, num_priors * num_classes, H, W).

Parameters

- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for all scale levels, each is a 4D-tensor, has shape (batch_size, num_priors * 4, H, W).
- **objectnesses** (*list[Tensor], Optional*) – Score factor for all scale level, each is a 4D-tensor, has shape (batch_size, 1, H, W).
- **img_metas** (*list[dict], Optional*) – Image meta info. Default None.
- **cfg** (*mmcv.Config, Optional*) – Test / postprocessing configuration, if None, test_cfg would be used. Default None.
- **rescale** (*bool*) – If True, return boxes in original image space. Default False.
- **with_nms** (*bool*) – If True, do nms before return boxes. Default True.

Returns

Each item in result_list is 2-tuple. The first item is an (n, 5) tensor, where the first 4 columns are bounding box positions (tl_x, tl_y, br_x, br_y) and the 5-th column is a score between 0 and 1. The second item is a (n,) tensor where each item is the predicted class label of the corresponding box.

Return type list[list[Tensor, Tensor]]

init_weights()
Initialize the weights.

loss(*cls_scores*, *bbox_preds*, *objectnesses*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)
Compute loss of the head. :param cls_scores: Box scores for each scale level,

each is a 4D-tensor, the channel number is num_priors * num_classes.

Parameters

- **bbox_preds** (*list[Tensor]*) – Box energies / deltas for each scale level, each is a 4D-tensor, the channel number is num_priors * 4.
- **objectnesses** (*list[Tensor], Optional*) – Score factor for all scale level, each is a 4D-tensor, has shape (batch_size, 1, H, W).
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **img_metas** (*list[dict]*) – Meta information of each image, e.g., image size, scaling factor, etc.

- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss.

40.5 roi_heads

class mmdet.models.roi_heads.**BBoxHead**(*with_avg_pool=False*, *with_cls=True*, *with_reg=True*, *roi_feat_size=7*, *in_channels=256*, *num_classes=80*, *bbox_coder={'clip_border': True, 'target_means': [0.0, 0.0, 0.0, 0.0], 'target_stds': [0.1, 0.1, 0.2, 0.2], 'type': 'DeltaXYWHBBoxCoder'}*, *reg_class_agnostic=False*, *reg_decoded_bbox=False*, *reg_predictor_cfg={'type': 'Linear'}*, *cls_predictor_cfg={'type': 'Linear'}*, *loss_cls={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_sigmoid': False}*, *loss_bbox={'beta': 1.0, 'loss_weight': 1.0, 'type': 'SmoothL1Loss'}*, *init_cfg=None*)

Simplest RoI head, with only two fc layers for classification and regression respectively.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

get_bboxes(*rois*, *cls_score*, *bbox_pred*, *img_shape*, *scale_factor*, *rescale=False*, *cfg=None*)

Transform network output for a batch into bbox predictions.

Parameters

- **rois** (*Tensor*) – Boxes to be transformed. Has shape (num_boxes, 5). last dimension 5 arrange as (batch_index, x1, y1, x2, y2).
- **cls_score** (*Tensor*) – Box scores, has shape (num_boxes, num_classes + 1).
- **bbox_pred** (*Tensor, optional*) – Box energies / deltas. has shape (num_boxes, num_classes * 4).
- **img_shape** (*Sequence[int], optional*) – Maximum bounds for boxes, specifies (H, W, C) or (H, W).
- **scale_factor** (*ndarray*) – Scale factor of the image arrange as (w_scale, h_scale, w_scale, h_scale).
- **rescale** (*bool*) – If True, return boxes in original image space. Default: False.
- **(obj** (*cfg*) – *ConfigDict*): *test_cfg* of Bbox Head. Default: None

Returns First tensor is *det_bboxes*, has the shape (num_boxes, 5) and last dimension 5 represent (tl_x, tl_y, br_x, br_y, score). Second tensor is the labels with shape (num_boxes,).

Return type tuple[Tensor, Tensor]

get_targets(*sampling_results*, *gt_bboxes*, *gt_labels*, *rcnn_train_cfg*, *concat=True*)

Calculate the ground truth for all samples in a batch according to the sampling_results.

Almost the same as the implementation in bbox_head, we passed additional parameters pos_inds_list and neg_inds_list to *_get_target_single* function.

Parameters

- **(List[obj** (*sampling_results*) – SamplingResults]): Assign results of all images in a batch after sampling.
- **gt_bboxes** (*list[Tensor]*) – Gt_bboxes of all images in a batch, each tensor has shape (num_gt, 4), the last dimension 4 represents [tl_x, tl_y, br_x, br_y].
- **gt_labels** (*list[Tensor]*) – Gt_labels of all images in a batch, each tensor has shape (num_gt,).
- **(obj** (*rcnn_train_cfg*) – ConfigDict): *train_cfg* of RCNN.
- **concat** (*bool*) – Whether to concatenate the results of all the images in a single batch.

Returns

Ground truth for proposals in a single image. Containing the following list of Tensors:

- labels (list[Tensor],Tensor): Gt_labels for all proposals in a batch, each tensor in list has shape (num_proposals,) when *concat=False*, otherwise just a single tensor has shape (num_all_proposals,).
- label_weights (list[Tensor]): Labels_weights for all proposals in a batch, each tensor in list has shape (num_proposals,) when *concat=False*, otherwise just a single tensor has shape (num_all_proposals,).
- bbox_targets (list[Tensor],Tensor): Regression target for all proposals in a batch, each tensor in list has shape (num_proposals, 4) when *concat=False*, otherwise just a single tensor has shape (num_all_proposals, 4), the last dimension 4 represents [tl_x, tl_y, br_x, br_y].
- bbox_weights (list[tensor],Tensor): Regression weights for all proposals in a batch, each tensor in list has shape (num_proposals, 4) when *concat=False*, otherwise just a single tensor has shape (num_all_proposals, 4).

Return type Tuple[Tensor]

onnx_export(*rois*, *cls_score*, *bbox_pred*, *img_shape*, *cfg=None*, ***kwargs*)
Transform network output for a batch into bbox predictions.

Parameters

- **rois** (*Tensor*) – Boxes to be transformed. Has shape (B, num_boxes, 5)
- **cls_score** (*Tensor*) – Box scores. has shape (B, num_boxes, num_classes + 1), 1 represent the background.
- **bbox_pred** (*Tensor, optional*) – Box energies / deltas for, has shape (B, num_boxes, num_classes * 4) when.
- **img_shape** (*torch.Tensor*) – Shape of image.
- **(obj** (*cfg*) – *ConfigDict*): *test_cfg* of Bbox Head. Default: None

Returns

dets of shape [N, num_det, 5] and class labels of shape [N, num_det].

Return type tuple[Tensor, Tensor]

refine_bboxes(*rois*, *labels*, *bbox_preds*, *pos_is_gts*, *img_metas*)
Refine bboxes during training.

Parameters

- **rois** (*Tensor*) – Shape (n*bs, 5), where n is image number per GPU, and bs is the sampled RoIs per image. The first column is the image id and the next 4 columns are x1, y1, x2, y2.
- **labels** (*Tensor*) – Shape (n*bs,).
- **bbox_preds** (*Tensor*) – Shape (n*bs, 4) or (n*bs, 4*#class).
- **pos_is_gts** (*list[Tensor]*) – Flags indicating if each positive bbox is a gt bbox.
- **img_metas** (*list[dict]*) – Meta info of each image.

Returns Refined bboxes of each image in a mini-batch.

Return type list[Tensor]

Example

```
>>> # xdoctest: +REQUIRES(module:kwarray)
>>> import kwarray
>>> import numpy as np
>>> from mmdet.core.bbox.demodata import random_boxes
>>> self = BBoxHead(reg_class_agnostic=True)
>>> n_roi = 2
>>> n_img = 4
>>> scale = 512
>>> rng = np.random.RandomState(0)
>>> img_metas = [{'img_shape': (scale, scale)}
...              for _ in range(n_img)]
>>> # Create rois in the expected format
>>> roi_boxes = random_boxes(n_roi, scale=scale, rng=rng)
>>> img_ids = torch.randint(0, n_img, (n_roi,))
>>> img_ids = img_ids.float()
>>> rois = torch.cat([img_ids[:, None], roi_boxes], dim=1)
>>> # Create other args
>>> labels = torch.randint(0, 2, (n_roi,)).long()
>>> bbox_preds = random_boxes(n_roi, scale=scale, rng=rng)
>>> # For each image, pretend random positive boxes are gts
>>> is_label_pos = (labels.numpy() > 0).astype(np.int)
>>> lbl_per_img = kwarray.group_items(is_label_pos,
...                                   img_ids.numpy())
>>> pos_per_img = [sum(lbl_per_img.get(gid, []))
...                for gid in range(n_img)]
>>> pos_is_gts = [
>>>     torch.randint(0, 2, (npos,)).byte().sort(
>>>         descending=True)[0]
>>>     for npos in pos_per_img
>>> ]
>>> bboxes_list = self.refine_bboxes(rois, labels, bbox_preds,
>>>                    pos_is_gts, img_metas)
>>> print(bboxes_list)
```

regress_by_class(*rois*, *label*, *bbox_pred*, *img_meta*)

Regress the bbox for the predicted class. Used in Cascade R-CNN.

Parameters

- **rois** (*Tensor*) – Rois from *rpn_head* or last stage *bbox_head*, has shape (num_proposals, 4) or (num_proposals, 5).
- **label** (*Tensor*) – Only used when *self.reg_class_agnostic* is False, has shape (num_proposals,).
- **bbox_pred** (*Tensor*) – Regression prediction of current stage *bbox_head*. When *self.reg_class_agnostic* is False, it has shape (n, num_classes * 4), otherwise it has shape (n, 4).
- **img_meta** (*dict*) – Image meta info.

Returns Regressed bboxes, the same shape as input rois.

Return type Tensor

class mmdet.models.roi_heads.**BaseRoIExtractor**(*roi_layer*, *out_channels*, *featmap_strides*, *init_cfg=None*)

Base class for RoI extractor.

Parameters

- **roi_layer** (*dict*) – Specify RoI layer type and arguments.
- **out_channels** (*int*) – Output channels of RoI layers.
- **featmap_strides** (*int*) – Strides of input feature maps.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

build_roi_layers(*layer_cfg*, *featmap_strides*)

Build RoI operator to extract feature from each level feature map.

Parameters

- **layer_cfg** (*dict*) – Dictionary to construct and config RoI layer operation. Options are modules under mmcv/ops such as RoIAlign.
- **featmap_strides** (*List[int]*) – The stride of input feature map w.r.t to the original image size, which would be used to scale RoI coordinate (original image coordinate system) to feature coordinate system.

Returns

The RoI extractor modules for each level feature map.

Return type nn.ModuleList

abstract forward(*feats*, *rois*, *roi_scale_factor=None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

property num_inputs

Number of input feature maps.

Type int

roi_rescale(*rois*, *scale_factor*)

Scale RoI coordinates by scale factor.

Parameters

- **rois** (*torch.Tensor*) – RoI (Region of Interest), shape (n, 5)
- **scale_factor** (*float*) – Scale factor that RoI will be multiplied by.

Returns Scaled RoI.

Return type torch.Tensor

class mmdet.models.roi_heads.**BaseRoIHead**(*bbox_roi_extractor=None*, *bbox_head=None*, *mask_roi_extractor=None*, *mask_head=None*, *shared_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

Base class for RoIHeads.

async async_simple_test(*x*, *proposal_list*, *img_metas*, *proposals=None*, *rescale=False*, ***kwargs*)
Asynchronized test function.

aug_test(*x*, *proposal_list*, *img_metas*, *rescale=False*, ***kwargs*)
Test with augmentations.

If rescale is False, then returned bboxes and masks will fit the scale of imgs[0].

abstract forward_train(*x*, *img_meta*, *proposal_list*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*, *gt_masks=None*, ***kwargs*)
Forward function during training.

abstract init_assigner_sampler()
Initialize assigner and sampler.

abstract init_bbox_head()
Initialize bbox_head

abstract init_mask_head()
Initialize mask_head

simple_test(*x*, *proposal_list*, *img_meta*, *proposals=None*, *rescale=False*, ***kwargs*)
Test without augmentation.

property with_bbox
whether the RoI head contains a *bbox_head*

Type bool

property with_mask
whether the RoI head contains a *mask_head*

Type bool

property with_shared_head
whether the RoI head contains a *shared_head*

Type bool

class mmdet.models.roi_heads.**CascadeRoIHead**(*num_stages*, *stage_loss_weights*, *bbox_roi_extractor=None*, *bbox_head=None*, *mask_roi_extractor=None*, *mask_head=None*, *shared_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

Cascade roi head including one bbox head and one mask head.

https://arxiv.org/abs/1712.00726

aug_test(*features*, *proposal_list*, *img_metas*, *rescale=False*)
Test with augmentations.

If rescale is False, then returned bboxes and masks will fit the scale of imgs[0].

forward_dummy(*x*, *proposals*)
Dummy forward function.

forward_train(*x*, *img_metas*, *proposal_list*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*, *gt_masks=None*)

Parameters

- **x** (*list[Tensor]*) – list of multi-level img features.
- **img_metas** (*list[dict]*) – list of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmdet/datasets/pipelines/formatting.py:Collect*.
- **proposals** (*list[Tensors]*) – list of region proposals.
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss.
- **gt_masks** (*None | Tensor*) – true segmentation masks for each box used if the architecture supports a segmentation task.

Returns a dictionary of loss components

Return type dict[str, Tensor]

init_assigner_sampler()
Initialize assigner and sampler for each stage.

init_bbox_head(*bbox_roi_extractor*, *bbox_head*)
Initialize box head and box roi extractor.

Parameters

- **bbox_roi_extractor** (*dict*) – Config of box roi extractor.
- **bbox_head** (*dict*) – Config of box in box head.

init_mask_head(*mask_roi_extractor*, *mask_head*)
Initialize mask head and mask roi extractor.

Parameters

- **mask_roi_extractor** (*dict*) – Config of mask roi extractor.
- **mask_head** (*dict*) – Config of mask in mask head.

simple_test(*x*, *proposal_list*, *img_metas*, *rescale=False*)
Test without augmentation.

Parameters

- **x** (*tuple[Tensor]*) – Features from upstream network. Each has shape (batch_size, c, h, w).

- **proposal_list** (*list(Tensor)*) – Proposals from rpn head. Each has shape (num_proposals, 5), last dimension 5 represent (x1, y1, x2, y2, score).
- **img_metas** (*list[dict]*) – Meta information of images.
- **rescale** (*bool*) – Whether to rescale the results to the original image. Default: True.

Returns When no mask branch, it is bbox results of each image and classes with type *list[list[np.ndarray]]*. The outer list corresponds to each image. The inner list corresponds to each class. When the model has mask branch, it contains bbox results and mask results. The outer list corresponds to each image, and first element of tuple is bbox results, second element is mask results.

Return type list[list[np.ndarray]] or list[tuple]

class mmdet.models.roi_heads.**CoarseMaskHead**(*num_convs=0*, *num_fcs=2*, *fc_out_channels=1024*, *downsample_factor=2*, *init_cfg={'override': [{'name': 'fcs'}, {'type': 'Constant', 'val': 0.001, 'name': 'fc_logits'}], 'type': 'Xavier'}*, **arg*, ***kwarg*)

Coarse mask head used in PointRend.

Compared with standard `FCNMaskHead`, `CoarseMaskHead` will downsample the input feature map instead of upsample it.

Parameters

- **num_convs** (*int*) – Number of conv layers in the head. Default: 0.
- **num_fcs** (*int*) – Number of fc layers in the head. Default: 2.
- **fc_out_channels** (*int*) – Number of output channels of fc layer. Default: 1024.
- **downsample_factor** (*int*) – The factor that feature map is downsampled by. Default: 2.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

init_weights()

Initialize the weights.

class mmdet.models.roi_heads.**ConvFCBBoxHead**(*num_shared_convs=0*, *num_shared_fcs=0*, *num_cls_convs=0*, *num_cls_fcs=0*, *num_reg_convs=0*, *num_reg_fcs=0*, *conv_out_channels=256*, *fc_out_channels=1024*, *conv_cfg=None*, *norm_cfg=None*, *init_cfg=None*, **args*, ***kwargs*)

More general bbox head, with shared conv and fc layers and two optional separated branches.

```
                            /-> cls convs -> cls fcs -> cls
shared convs -> shared fcs
                            \-> reg convs -> reg fcs -> reg
```

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.roi_heads.**DIIHead**(*num_classes=80*, *num_ffn_fcs=2*, *num_heads=8*, *num_cls_fcs=1*, *num_reg_fcs=3*, *feedforward_channels=2048*, *in_channels=256*, *dropout=0.0*, *ffn_act_cfg={'inplace': True, 'type': 'ReLU'}*, *dynamic_conv_cfg={'act_cfg': {'inplace': True, 'type': 'ReLU'}, 'feat_channels': 64, 'in_channels': 256, 'input_feat_shape': 7, 'norm_cfg': {'type': 'LN'}, 'out_channels': 256, 'type': 'DynamicConv'}*, *loss_iou={'loss_weight': 2.0, 'type': 'GIoULoss'}*, *init_cfg=None*, ***kwargs*)

Dynamic Instance Interactive Head for Sparse R-CNN: End-to-End Object Detection with Learnable Proposals

Parameters

- **num_classes** (*int*) – Number of class in dataset. Defaults to 80.
- **num_ffn_fcs** (*int*) – The number of fully-connected layers in FFNs. Defaults to 2.
- **num_heads** (*int*) – The hidden dimension of FFNs. Defaults to 8.
- **num_cls_fcs** (*int*) – The number of fully-connected layers in classification subnet. Defaults to 1.
- **num_reg_fcs** (*int*) – The number of fully-connected layers in regression subnet. Defaults to 3.
- **feedforward_channels** (*int*) – The hidden dimension of FFNs. Defaults to 2048
- **in_channels** (*int*) – Hidden_channels of MultiheadAttention. Defaults to 256.
- **dropout** (*float*) – Probability of drop the channel. Defaults to 0.0
- **ffn_act_cfg** (*dict*) – The activation config for FFNs.
- **dynamic_conv_cfg** (*dict*) – The convolution config for DynamicConv.
- **loss_iou** (*dict*) – The config for iou or giou loss.

forward(*roi_feat*, *proposal_feat*)

Forward function of Dynamic Instance Interactive Head.

Parameters

- **roi_feat** (*Tensor*) – Roi-pooling features with shape (batch_size*num_proposals, feature_dimensions, pooling_h , pooling_w).
- **proposal_feat** – Intermediate feature get from diihead in last stage, has shape (batch_size, num_proposals, feature_dimensions)

get_targets(*sampling_results*, *gt_bboxes*, *gt_labels*, *rcnn_train_cfg*, *concat=True*)

Calculate the ground truth for all samples in a batch according to the sampling_results.

Almost the same as the implementation in bbox_head, we passed additional parameters pos_inds_list and neg_inds_list to *_get_target_single* function.

Parameters

- **(List[obj** (*sampling_results*) – SamplingResults]): Assign results of all images in a batch after sampling.

- **gt_bboxes** (*list[Tensor]*) – Gt_bboxes of all images in a batch, each tensor has shape (num_gt, 4), the last dimension 4 represents [tl_x, tl_y, br_x, br_y].
- **gt_labels** (*list[Tensor]*) – Gt_labels of all images in a batch, each tensor has shape (num_gt,).
- **(obj** (*rcnn_train_cfg*) – *ConfigDict*): *train_cfg* of RCNN.
- **concat** (*bool*) – Whether to concatenate the results of all the images in a single batch.

Returns

Ground truth for proposals in a single image. Containing the following list of Tensors:

- labels (list[Tensor],Tensor): Gt_labels for all proposals in a batch, each tensor in list has shape (num_proposals,) when *concat=False*, otherwise just a single tensor has shape (num_all_proposals,).
- label_weights (list[Tensor]): Labels_weights for all proposals in a batch, each tensor in list has shape (num_proposals,) when *concat=False*, otherwise just a single tensor has shape (num_all_proposals,).
- bbox_targets (list[Tensor],Tensor): Regression target for all proposals in a batch, each tensor in list has shape (num_proposals, 4) when *concat=False*, otherwise just a single tensor has shape (num_all_proposals, 4), the last dimension 4 represents [tl_x, tl_y, br_x, br_y].
- bbox_weights (list[tensor],Tensor): Regression weights for all proposals in a batch, each tensor in list has shape (num_proposals, 4) when *concat=False*, otherwise just a single tensor has shape (num_all_proposals, 4).

Return type Tuple[Tensor]

init_weights()
Use xavier initialization for all weight parameter and set classification head bias as a specific value when use focal loss.

loss(*cls_score*, *bbox_pred*, *labels*, *label_weights*, *bbox_targets*, *bbox_weights*, *imgs_whwh=None*, *reduction_override=None*, ***kwargs*)
"Loss function of DIIHead, get loss of all images.

Parameters

- **cls_score** (*Tensor*) – Classification prediction results of all class, has shape (batch_size * num_proposals_single_image, num_classes)
- **bbox_pred** (*Tensor*) – Regression prediction results, has shape (batch_size * num_proposals_single_image, 4), the last dimension 4 represents [tl_x, tl_y, br_x, br_y].
- **labels** (*Tensor*) – Label of each proposals, has shape (batch_size * num_proposals_single_image
- **label_weights** (*Tensor*) – Classification loss weight of each proposals, has shape (batch_size * num_proposals_single_image
- **bbox_targets** (*Tensor*) – Regression targets of each proposals, has shape (batch_size * num_proposals_single_image, 4), the last dimension 4 represents [tl_x, tl_y, br_x, br_y].
- **bbox_weights** (*Tensor*) – Regression loss weight of each proposals's coordinate, has shape (batch_size * num_proposals_single_image, 4),
- **imgs_whwh** (*Tensor*) – imgs_whwh (Tensor): Tensor with shape (batch_size, num_proposals, 4), the last dimension means [img_width,img_height, img_width, img_height].

- **reduction_override** (*str, optional*) – The reduction method used to override the original reduction method of the loss. Options are "none", "mean" and "sum". Defaults to None,
- **Returns** – dict[str, Tensor]: Dictionary of loss components

class mmdet.models.roi_heads.**DoubleConvFCBBoxHead**(*num_convs=0*, *num_fcs=0*, *conv_out_channels=1024*, *fc_out_channels=1024*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *init_cfg={'override': [{'type': 'Normal', 'name': 'fc_cls', 'std': 0.01}, {'type': 'Normal', 'name': 'fc_reg', 'std': 0.001}, {'type': 'Xavier', 'name': 'fc_branch', 'distribution': 'uniform'}], 'type': 'Normal'}*, ***kwargs*)

Bbox head used in Double-Head R-CNN

```
                                /-> cls
             /-> shared convs ->
                                \-> reg
roi features
                                /-> cls
             \-> shared fc    ->
                                \-> reg
```

forward(*x_cls*, *x_reg*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.roi_heads.**DoubleHeadRoIHead**(*reg_roi_scale_factor*, ***kwargs*)

RoI head for Double Head RCNN.

https://arxiv.org/abs/1904.06493

class mmdet.models.roi_heads.**DynamicRoIHead**(***kwargs*)

RoI head for Dynamic R-CNN.

forward_train(*x*, *img_metas*, *proposal_list*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*, *gt_masks=None*)

Forward function for training.

Parameters

- **x** (*list[Tensor]*) – list of multi-level img features.
- **img_metas** (*list[dict]*) – list of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmdet/datasets/pipelines/formatting.py:Collect*.
- **proposals** (*list[Tensors]*) – list of region proposals.
- **gt_bboxes** (*list[Tensor]*) – each item are the truth boxes for each image in [tl_x, tl_y, br_x, br_y] format.

- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss.
- **gt_masks** (*None | Tensor*) – true segmentation masks for each box used if the architecture supports a segmentation task.

Returns a dictionary of loss components

Return type dict[str, Tensor]

update_hyperparameters()
Update hyperparameters like IoU thresholds for assigner and beta for SmoothL1 loss based on the training statistics.

Returns the updated `iou_thr` and `beta`.

Return type tuple[float]

class mmdet.models.roi_heads.**FCNMaskHead**(*num_convs=4*, *roi_feat_size=14*, *in_channels=256*, *conv_kernel_size=3*, *conv_out_channels=256*, *num_classes=80*, *class_agnostic=False*, *upsample_cfg={'scale_factor': 2, 'type': 'deconv'}*, *conv_cfg=None*, *norm_cfg=None*, *predictor_cfg={'type': 'Conv'}*, *loss_mask={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_mask': True}*, *init_cfg=None*)

forward(*x*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

get_seg_masks(*mask_pred*, *det_bboxes*, *det_labels*, *rcnn_test_cfg*, *ori_shape*, *scale_factor*, *rescale*)
Get segmentation masks from mask_pred and bboxes.

Parameters

- **mask_pred** (*Tensor or ndarray*) – shape (n, #class, h, w). For single-scale testing, mask_pred is the direct output of model, whose type is Tensor, while for multi-scale testing, it will be converted to numpy array outside of this method.
- **det_bboxes** (*Tensor*) – shape (n, 4/5)
- **det_labels** (*Tensor*) – shape (n,)
- **rcnn_test_cfg** (*dict*) – rcnn testing config
- **ori_shape** (*Tuple*) – original image height and width, shape (2,)
- **scale_factor** (*ndarray | Tensor*) – If `rescale is True`, box coordinates are divided by this scale factor to fit `ori_shape`.
- **rescale** (*bool*) – If True, the resulting masks will be rescaled to `ori_shape`.

Returns

encoded masks. The c-th item in the outer list corresponds to the c-th class. Given the c-th outer list, the i-th item in that inner list is the mask for the i-th box with class label c.

Return type list[list]

Example

```
>>> import mmcv
>>> from mmdet.models.roi_heads.mask_heads.fcn_mask_head import *  # NOQA
>>> N = 7  # N = number of extracted ROIs
>>> C, H, W = 11, 32, 32
>>> # Create example instance of FCN Mask Head.
>>> self = FCNMaskHead(num_classes=C, num_convs=0)
>>> inputs = torch.rand(N, self.in_channels, H, W)
>>> mask_pred = self.forward(inputs)
>>> # Each input is associated with some bounding box
>>> det_bboxes = torch.Tensor([[1, 1, 42, 42 ]] * N)
>>> det_labels = torch.randint(0, C, size=(N,))
>>> rcnn_test_cfg = mmcv.Config({'mask_thr_binary': 0, })
>>> ori_shape = (H * 4, W * 4)
>>> scale_factor = torch.FloatTensor((1, 1))
>>> rescale = False
>>> # Encoded masks are a list for each category.
>>> encoded_masks = self.get_seg_masks(
>>>     mask_pred, det_bboxes, det_labels, rcnn_test_cfg, ori_shape,
>>>     scale_factor, rescale
>>> )
>>> assert len(encoded_masks) == C
>>> assert sum(list(map(len, encoded_masks))) == N
```

init_weights()
Initialize the weights.

loss(*mask_pred*, *mask_targets*, *labels*)

Example

```
>>> from mmdet.models.roi_heads.mask_heads.fcn_mask_head import *  # NOQA
>>> N = 7  # N = number of extracted ROIs
>>> C, H, W = 11, 32, 32
>>> # Create example instance of FCN Mask Head.
>>> # There are lots of variations depending on the configuration
>>> self = FCNMaskHead(num_classes=C, num_convs=1)
>>> inputs = torch.rand(N, self.in_channels, H, W)
>>> mask_pred = self.forward(inputs)
>>> sf = self.scale_factor
>>> labels = torch.randint(0, C, size=(N,))
>>> # With the default properties the mask targets should indicate
>>> # a (potentially soft) single-class label
>>> mask_targets = torch.rand(N, H * sf, W * sf)
```

(continues on next page)

(continued from previous page)

```
>>> loss = self.loss(mask_pred, mask_targets, labels)
>>> print('loss = {!r}'.format(loss))
```

onnx_export(*mask_pred*, *det_bboxes*, *det_labels*, *rcnn_test_cfg*, *ori_shape*, ***kwargs*)

Get segmentation masks from mask_pred and bboxes.

Parameters

- **mask_pred** (*Tensor*) – shape (n, #class, h, w).
- **det_bboxes** (*Tensor*) – shape (n, 4/5)
- **det_labels** (*Tensor*) – shape (n,)
- **rcnn_test_cfg** (*dict*) – rcnn testing config
- **ori_shape** (*Tuple*) – original image height and width, shape (2,)

Returns a mask of shape (N, img_h, img_w).

Return type Tensor

class mmdet.models.roi_heads.**FeatureRelayHead**(*in_channels=1024*, *out_conv_channels=256*, *roi_feat_size=7*, *scale_factor=2*, *init_cfg={'layer': 'Linear', 'type': 'Kaiming'}*)

Feature Relay Head used in SCNet.

Parameters

- **in_channels** (*int, optional*) – number of input channels. Default: 256.
- **conv_out_channels** (*int, optional*) – number of output channels before classification layer. Default: 256.
- **roi_feat_size** (*int, optional*) – roi feat size at box head. Default: 7.
- **scale_factor** (*int, optional*) – scale factor to match roi feat size at mask head. Default: 2.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

forward(*x*)

Forward function.

class mmdet.models.roi_heads.**FusedSemanticHead**(*num_ins*, *fusion_level*, *num_convs=4*, *in_channels=256*, *conv_out_channels=256*, *num_classes=183*, *conv_cfg=None*, *norm_cfg=None*, *ignore_label=None*, *loss_weight=None*, *loss_seg={'ignore_index': 255, 'loss_weight': 0.2, 'type': 'CrossEntropyLoss'}*, *init_cfg={'override': {'name': 'conv_logits'}, 'type': 'Kaiming'}*)

Multi-level fused semantic segmentation head.

```
in_1 -> 1x1 conv ---
                    |
in_2 -> 1x1 conv -- |
                   ||
in_3 -> 1x1 conv - ||
                  |||                  /-> 1x1 conv (mask prediction)
in_4 -> 1x1 conv -----> 3x3 convs (*4)
```

(continues on next page)

(continued from previous page)

```
                    |                   \-> 1x1 conv (feature)
in_5 -> 1x1 conv ---
```

forward(*feats*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.roi_heads.**GenericRoIExtractor**(*aggregation='sum'*, *pre_cfg=None*, *post_cfg=None*, ***kwargs*)

Extract RoI features from all level feature maps levels.

This is the implementation of A novel Region of Interest Extraction Layer for Instance Segmentation.

Parameters

- **aggregation** (*str*) – The method to aggregate multiple feature maps. Options are 'sum', 'concat'. Default: 'sum'.
- **pre_cfg** (*dict | None*) – Specify pre-processing modules. Default: None.
- **post_cfg** (*dict | None*) – Specify post-processing modules. Default: None.
- **kwargs** (*keyword arguments*) – Arguments that are the same as *BaseRoIExtractor*.

forward(*feats*, *rois*, *roi_scale_factor=None*)

Forward function.

class mmdet.models.roi_heads.**GlobalContextHead**(*num_convs=4*, *in_channels=256*, *conv_out_channels=256*, *num_classes=80*, *loss_weight=1.0*, *conv_cfg=None*, *norm_cfg=None*, *conv_to_res=False*, *init_cfg={'override': {'name': 'fc'}, 'std': 0.01, 'type': 'Normal'}*)

Global context head used in SCNet.

Parameters

- **num_convs** (*int, optional*) – number of convolutional layer in GlbCtxHead. Default: 4.
- **in_channels** (*int, optional*) – number of input channels. Default: 256.
- **conv_out_channels** (*int, optional*) – number of output channels before classification layer. Default: 256.
- **num_classes** (*int, optional*) – number of classes. Default: 80.
- **loss_weight** (*float, optional*) – global context loss weight. Default: 1.
- **conv_cfg** (*dict, optional*) – config to init conv layer. Default: None.
- **norm_cfg** (*dict, optional*) – config to init norm layer. Default: None.
- **conv_to_res** (*bool, optional*) – if True, 2 convs will be grouped into 1 *SimplifiedBasicBlock* using a skip connection. Default: False.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

forward(*feats*)
Forward function.

loss(*pred*, *labels*)
Loss function.

class mmdet.models.roi_heads.**GridHead**(*grid_points=9*, *num_convs=8*, *roi_feat_size=14*, *in_channels=256*, *conv_kernel_size=3*, *point_feat_channels=64*, *deconv_kernel_size=4*, *class_agnostic=False*, *loss_grid={'loss_weight': 15, 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, *conv_cfg=None*, *norm_cfg={'num_groups': 36, 'type': 'GN'}*, *init_cfg=[{'type': 'Kaiming', 'layer': ['Conv2d', 'Linear']}, {'type': 'Normal', 'layer': 'ConvTranspose2d', 'std': 0.001, 'override': {'type': 'Normal', 'name': 'deconv2', 'std': 0.001, 'bias': - 4.59511985013459}}]*)

calc_sub_regions()
Compute point specific representation regions.

See Grid R-CNN Plus (https://arxiv.org/abs/1906.05688) for details.

forward(*x*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.roi_heads.**GridRoIHead**(*grid_roi_extractor*, *grid_head*, ***kwargs*)
Grid roi head for Grid R-CNN.

https://arxiv.org/abs/1811.12030

forward_dummy(*x*, *proposals*)
Dummy forward function.

simple_test(*x*, *proposal_list*, *img_metas*, *proposals=None*, *rescale=False*)
Test without augmentation.

class mmdet.models.roi_heads.**HTCMaskHead**(*with_conv_res=True*, **args*, ***kwargs*)

forward(*x*, *res_feat=None*, *return_logits=True*, *return_feat=True*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.roi_heads.**HybridTaskCascadeRoIHead**(*num_stages*, *stage_loss_weights*, *semantic_roi_extractor=None*, *semantic_head=None*, *semantic_fusion=('bbox', 'mask')*, *interleaved=True*, *mask_info_flow=True*, ***kwargs*)

Hybrid task cascade roi head including one bbox head and one mask head.

https://arxiv.org/abs/1901.07518

aug_test(*img_feats*, *proposal_list*, *img_metas*, *rescale=False*)

Test with augmentations.

If rescale is False, then returned bboxes and masks will fit the scale of imgs[0].

forward_dummy(*x*, *proposals*)

Dummy forward function.

forward_train(*x*, *img_metas*, *proposal_list*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*, *gt_masks=None*, *gt_semantic_seg=None*)

Parameters

- **x** (*list[Tensor]*) – list of multi-level img features.
- **img_metas** (*list[dict]*) – list of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmdet/datasets/pipelines/formatting.py:Collect*.
- **proposal_list** (*list[Tensors]*) – list of region proposals.
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **gt_bboxes_ignore** (*None, list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss.
- **gt_masks** (*None, Tensor*) – true segmentation masks for each box used if the architecture supports a segmentation task.
- **gt_semantic_seg** (*None, list[Tensor]*) – semantic segmentation masks used if the architecture supports semantic segmentation task.

Returns a dictionary of loss components

Return type dict[str, Tensor]

simple_test(*x*, *proposal_list*, *img_metas*, *rescale=False*)

Test without augmentation.

Parameters

- **x** (*tuple[Tensor]*) – Features from upstream network. Each has shape (batch_size, c, h, w).
- **proposal_list** (*list(Tensor)*) – Proposals from rpn head. Each has shape (num_proposals, 5), last dimension 5 represent (x1, y1, x2, y2, score).
- **img_metas** (*list[dict]*) – Meta information of images.
- **rescale** (*bool*) – Whether to rescale the results to the original image. Default: True.

Returns When no mask branch, it is bbox results of each image and classes with type *list[list[np.ndarray]]*. The outer list corresponds to each image. The inner list corresponds to each class. When the model has mask branch, it contains bbox results and mask results. The outer list corresponds to each image, and first element of tuple is bbox results, second element is mask results.

Return type list[list[np.ndarray]] or list[tuple]

property with_semantic
whether the head has semantic head

Type bool

class mmdet.models.roi_heads.**MaskIoUHead**(*num_convs=4*, *num_fcs=2*, *roi_feat_size=14*, *in_channels=256*, *conv_out_channels=256*, *fc_out_channels=1024*, *num_classes=80*, *loss_iou={'loss_weight': 0.5, 'type': 'MSELoss'}*, *init_cfg=[{'type': 'Kaiming', 'override': {'name': 'convs'}}, {'type': 'Caffe2Xavier', 'override': {'name': 'fcs'}}, {'type': 'Normal', 'std': 0.01, 'override': {'name': 'fc_mask_iou'}}]*)

Mask IoU Head.

This head predicts the IoU of predicted masks and corresponding gt masks.

forward(*mask_feat*, *mask_pred*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

get_mask_scores(*mask_iou_pred*, *det_bboxes*, *det_labels*)
Get the mask scores.

mask_score = bbox_score * mask_iou

get_targets(*sampling_results*, *gt_masks*, *mask_pred*, *mask_targets*, *rcnn_train_cfg*)
Compute target of mask IoU.

Mask IoU target is the IoU of the predicted mask (inside a bbox) and the gt mask of corresponding gt mask (the whole instance). The intersection area is computed inside the bbox, and the gt mask area is computed with two steps, firstly we compute the gt area inside the bbox, then divide it by the area ratio of gt area inside the bbox and the gt area of the whole instance.

Parameters

- **sampling_results** (list[`SamplingResult`]) – sampling results.
- **gt_masks** (*`BitmapMask` | `PolygonMask`*) – Gt masks (the whole instance) of each image, with the same shape of the input image.
- **mask_pred** (*`Tensor`*) – Predicted masks of each positive proposal, shape (num_pos, h, w).
- **mask_targets** (*`Tensor`*) – Gt mask of each positive proposal, binary map of the shape (num_pos, h, w).
- **rcnn_train_cfg** (*`dict`*) – Training config for R-CNN part.

Returns mask iou target (length == num positive).

Return type Tensor

class mmdet.models.roi_heads.**MaskPointHead**(*num_classes*, *num_fcs=3*, *in_channels=256*, *fc_channels=256*, *class_agnostic=False*, *coarse_pred_each_layer=True*, *conv_cfg={'type': 'Conv1d'}*, *norm_cfg=None*, *act_cfg={'type': 'ReLU'}*, *loss_point={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_mask': True}*, *init_cfg={'override': {'name': 'fc_logits'}, 'std': 0.001, 'type': 'Normal'}*)

A mask point head use in PointRend.

MaskPointHead use shared multi-layer perceptron (equivalent to nn.Conv1d) to predict the logit of input points. The fine-grained feature and coarse feature will be concatenate together for predication.

Parameters

- **num_fcs** (*int*) – Number of fc layers in the head. Default: 3.
- **in_channels** (*int*) – Number of input channels. Default: 256.
- **fc_channels** (*int*) – Number of fc channels. Default: 256.
- **num_classes** (*int*) – Number of classes for logits. Default: 80.
- **class_agnostic** (*bool*) – Whether use class agnostic classification. If so, the output channels of logits will be 1. Default: False.
- **coarse_pred_each_layer** (*bool*) – Whether concatenate coarse feature with the output of each fc layer. Default: True.
- **conv_cfg** (*dict | None*) – Dictionary to construct and config conv layer. Default: dict(type='Conv1d'))
- **norm_cfg** (*dict | None*) – Dictionary to construct and config norm layer. Default: None.
- **loss_point** (*dict*) – Dictionary to construct and config loss layer of point head. Default: dict(type='CrossEntropyLoss', use_mask=True, loss_weight=1.0).
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

forward(*fine_grained_feats*, *coarse_feats*)

Classify each point base on fine grained and coarse feats.

Parameters

- **fine_grained_feats** (*Tensor*) – Fine grained feature sampled from FPN, shape (num_rois, in_channels, num_points).
- **coarse_feats** (*Tensor*) – Coarse feature sampled from CoarseMaskHead, shape (num_rois, num_classes, num_points).

Returns

Point classification results, shape (num_rois, num_class, num_points).

Return type Tensor

get_roi_rel_points_test(*mask_pred*, *pred_label*, *cfg*)

Get num_points most uncertain points during test.

Parameters

- **mask_pred** (*Tensor*) – A tensor of shape (num_rois, num_classes, mask_height, mask_width) for class-specific or class-agnostic prediction.
- **pred_label** (*list*) – The predication class for each instance.

- **cfg** (*dict*) – Testing config of point head.

Returns

A tensor of shape (num_rois, num_points) that contains indices from [0, mask_height x mask_width) of the most uncertain points.

point_coords (Tensor): A tensor of shape (num_rois, num_points, 2) that contains [0, 1] x [0, 1] normalized coordinates of the most uncertain points from the [mask_height, mask_width] grid .

Return type point_indices (Tensor)

get_roi_rel_points_train(*mask_pred*, *labels*, *cfg*)

Get num_points most uncertain points with random points during train.

Sample points in [0, 1] x [0, 1] coordinate space based on their uncertainty. The uncertainties are calculated for each point using '_get_uncertainty()' function that takes point's logit prediction as input.

Parameters

- **mask_pred** (*Tensor*) – A tensor of shape (num_rois, num_classes, mask_height, mask_width) for class-specific or class-agnostic prediction.
- **labels** (*list*) – The ground truth class for each instance.
- **cfg** (*dict*) – Training config of point head.

Returns

A tensor of shape (num_rois, num_points, 2) that contains the coordinates sampled points.

Return type point_coords (Tensor)

get_targets(*rois*, *rel_roi_points*, *sampling_results*, *gt_masks*, *cfg*)

Get training targets of MaskPointHead for all images.

Parameters

- **rois** (*Tensor*) – Region of Interest, shape (num_rois, 5).
- **rel_roi_points** – Points coordinates relative to RoI, shape (num_rois, num_points, 2).
- **sampling_results** (SamplingResult) – Sampling result after sampling and assignment.
- **gt_masks** (*Tensor*) – Ground truth segmentation masks of corresponding boxes, shape (num_rois, height, width).
- **cfg** (*dict*) – Training cfg.

Returns Point target, shape (num_rois, num_points).

Return type Tensor

loss(*point_pred*, *point_targets*, *labels*)

Calculate loss for MaskPointHead.

Parameters

- **point_pred** (*Tensor*) – Point predication result, shape (num_rois, num_classes, num_points).
- **point_targets** (*Tensor*) – Point targets, shape (num_roi, num_points).
- **labels** (*Tensor*) – Class label of corresponding boxes, shape (num_rois,)

Returns a dictionary of point loss components

Return type dict[str, Tensor]

class mmdet.models.roi_heads.**MaskScoringRoIHead**(*mask_iou_head*, ***kwargs*)
Mask Scoring RoIHead for Mask Scoring RCNN.

https://arxiv.org/abs/1903.00241

simple_test_mask(*x*, *img_metas*, *det_bboxes*, *det_labels*, *rescale=False*)
Obtain mask prediction without augmentation.

class mmdet.models.roi_heads.**PISARoIHead**(*bbox_roi_extractor=None*, *bbox_head=None*, *mask_roi_extractor=None*, *mask_head=None*, *shared_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)
The RoI head for Prime Sample Attention in Object Detection.

forward_train(*x*, *img_metas*, *proposal_list*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*, *gt_masks=None*)
Forward function for training.

Parameters

- **x** (*list[Tensor]*) – List of multi-level img features.
- **img_metas** (*list[dict]*) – List of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmdet/datasets/pipelines/formatting.py:Collect*.
- **proposals** (*list[Tensors]*) – List of region proposals.
- **gt_bboxes** (*list[Tensor]*) – Each item are the truth boxes for each image in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – Class indices corresponding to each box
- **gt_bboxes_ignore** (*list[Tensor], optional*) – Specify which bounding boxes can be ignored when computing the loss.
- **gt_masks** (*None | Tensor*) – True segmentation masks for each box used if the architecture supports a segmentation task.

Returns a dictionary of loss components

Return type dict[str, Tensor]

class mmdet.models.roi_heads.**PointRendRoIHead**(*point_head*, **args*, ***kwargs*)
PointRend.

aug_test_mask(*feats*, *img_metas*, *det_bboxes*, *det_labels*)
Test for mask head with test time augmentation.

init_point_head(*point_head*)
Initialize `point_head`

mask_onnx_export(*x*, *img_metas*, *det_bboxes*, *det_labels*, ***kwargs*)
Export mask branch to onnx which supports batch inference.

Parameters

- **x** (*tuple[Tensor]*) – Feature maps of all scale level.
- **img_metas** (*list[dict]*) – Image meta info.

- **det_bboxes** (*Tensor*) – Bboxes and corresponding scores. has shape [N, num_bboxes, 5].
- **det_labels** (*Tensor*) – class labels of shape [N, num_bboxes].

Returns

The segmentation results of shape [N, num_bboxes, image_height, image_width].

Return type Tensor

simple_test_mask(*x*, *img_metas*, *det_bboxes*, *det_labels*, *rescale=False*)

Obtain mask prediction without augmentation.

class mmdet.models.roi_heads.**ResLayer**(*depth*, *stage=3*, *stride=2*, *dilation=1*, *style='pytorch'*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *norm_eval=True*, *with_cp=False*, *dcn=None*, *pretrained=None*, *init_cfg=None*)

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

train(*mode=True*)

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. `Dropout`, `BatchNorm`, etc.

Parameters **mode** (*bool*) – whether to set training mode (`True`) or evaluation mode (`False`). Default: `True`.

Returns self

Return type Module

class mmdet.models.roi_heads.**SABLHead**(*num_classes*, *cls_in_channels=256*, *reg_in_channels=256*, *roi_feat_size=7*, *reg_feat_up_ratio=2*, *reg_pre_kernel=3*, *reg_post_kernel=3*, *reg_pre_num=2*, *reg_post_num=1*, *cls_out_channels=1024*, *reg_offset_out_channels=256*, *reg_cls_out_channels=256*, *num_cls_fcs=1*, *num_reg_fcs=0*, *reg_class_agnostic=True*, *norm_cfg=None*, *bbox_coder={'num_buckets': 14, 'scale_factor': 1.7, 'type': 'BucketingBBoxCoder'}*, *loss_cls={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_sigmoid': False}*, *loss_bbox_cls={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, *loss_bbox_reg={'beta': 0.1, 'loss_weight': 1.0, 'type': 'SmoothL1Loss'}*, *init_cfg=None*)

Side-Aware Boundary Localization (SABL) for RoI-Head.

Side-Aware features are extracted by conv layers with an attention mechanism. Boundary Localization with Bucketing and Bucketing Guided Rescoring are implemented in BucketingBBoxCoder.

Please refer to https://arxiv.org/abs/1912.04260 for more details.

Parameters

- **cls_in_channels** (*int*) – Input channels of cls RoI feature. Defaults to 256.
- **reg_in_channels** (*int*) – Input channels of reg RoI feature. Defaults to 256.
- **roi_feat_size** (*int*) – Size of RoI features. Defaults to 7.
- **reg_feat_up_ratio** (*int*) – Upsample ratio of reg features. Defaults to 2.
- **reg_pre_kernel** (*int*) – Kernel of 2D conv layers before attention pooling. Defaults to 3.
- **reg_post_kernel** (*int*) – Kernel of 1D conv layers after attention pooling. Defaults to 3.
- **reg_pre_num** (*int*) – Number of pre convs. Defaults to 2.
- **reg_post_num** (*int*) – Number of post convs. Defaults to 1.
- **num_classes** (*int*) – Number of classes in dataset. Defaults to 80.
- **cls_out_channels** (*int*) – Hidden channels in cls fcs. Defaults to 1024.
- **reg_offset_out_channels** (*int*) – Hidden and output channel of reg offset branch. Defaults to 256.
- **reg_cls_out_channels** (*int*) – Hidden and output channel of reg cls branch. Defaults to 256.
- **num_cls_fcs** (*int*) – Number of fcs for cls branch. Defaults to 1.
- **num_reg_fcs** (*int*) – Number of fcs for reg branch.. Defaults to 0.
- **reg_class_agnostic** (*bool*) – Class agnostic regression or not. Defaults to True.
- **norm_cfg** (*dict*) – Config of norm layers. Defaults to None.
- **bbox_coder** (*dict*) – Config of bbox coder. Defaults 'BucketingBBoxCoder'.
- **loss_cls** (*dict*) – Config of classification loss.
- **loss_bbox_cls** (*dict*) – Config of classification loss for bbox branch.
- **loss_bbox_reg** (*dict*) – Config of regression loss for bbox branch.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

attention_pool(*reg_x*)
Extract direction-specific features fx and fy with attention methanism.

bbox_pred_split(*bbox_pred*, *num_proposals_per_img*)
Split batch bbox prediction back to each image.

forward(*x*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

refine_bboxes(*rois*, *labels*, *bbox_preds*, *pos_is_gts*, *img_metas*)
Refine bboxes during training.

Parameters

- **rois** (*Tensor*) – Shape (n*bs, 5), where n is image number per GPU, and bs is the sampled RoIs per image.

- **labels** (*Tensor*) – Shape (n*bs,).
- **bbox_preds** (*list[Tensor]*) – Shape [(n*bs, num_buckets*2), (n*bs, num_buckets*2)].
- **pos_is_gts** (*list[Tensor]*) – Flags indicating if each positive bbox is a gt bbox.
- **img_metas** (*list[dict]*) – Meta info of each image.

Returns Refined bboxes of each image in a mini-batch.

Return type list[Tensor]

reg_pred(*x*, *offset_fcs*, *cls_fcs*)
Predict bucketing estimation (cls_pred) and fine regression (offset pred) with side-aware features.

regress_by_class(*rois*, *label*, *bbox_pred*, *img_meta*)
Regress the bbox for the predicted class. Used in Cascade R-CNN.

Parameters

- **rois** (*Tensor*) – shape (n, 4) or (n, 5)
- **label** (*Tensor*) – shape (n,)
- **bbox_pred** (*list[Tensor]*) – shape [(n, num_buckets *2), (n, num_buckets *2)]
- **img_meta** (*dict*) – Image meta info.

Returns Regressed bboxes, the same shape as input rois.

Return type Tensor

side_aware_feature_extractor(*reg_x*)
Refine and extract side-aware features without split them.

side_aware_split(*feat*)
Split side-aware features aligned with orders of bucketing targets.

class mmdet.models.roi_heads.**SCNetBBoxHead**(*num_shared_convs=0*, *num_shared_fcs=0*, *num_cls_convs=0*, *num_cls_fcs=0*, *num_reg_convs=0*, *num_reg_fcs=0*, *conv_out_channels=256*, *fc_out_channels=1024*, *conv_cfg=None*, *norm_cfg=None*, *init_cfg=None*, **args*, ***kwargs*)
BBox head for SCNet.

This inherits ConvFCBBoxHead with modified forward() function, allow us to get intermediate shared feature.

forward(*x*, *return_shared_feat=False*)
Forward function.

Parameters

- **x** (*Tensor*) – input features
- **return_shared_feat** (*bool*) – If True, return cls-reg-shared feature.

Returns

contain cls_score and bbox_pred, if return_shared_feat is True, append x_shared to the returned tuple.

Return type out (tuple[Tensor])

class mmdet.models.roi_heads.**SCNetMaskHead**(*conv_to_res=True*, ***kwargs*)
Mask head for SCNet.

Parameters **conv_to_res** (*bool, optional*) – if True, change the conv layers to SimplifiedBasicBlock.

class mmdet.models.roi_heads.**SCNetRoIHead**(*num_stages*, *stage_loss_weights*, *semantic_roi_extractor=None*, *semantic_head=None*, *feat_relay_head=None*, *glbctx_head=None*, ***kwargs*)

RoIHead for SCNet.

Parameters

- **num_stages** (*int*) – number of cascade stages.
- **stage_loss_weights** (*list*) – loss weight of cascade stages.
- **semantic_roi_extractor** (*dict*) – config to init semantic roi extractor.
- **semantic_head** (*dict*) – config to init semantic head.
- **feat_relay_head** (*dict*) – config to init feature_relay_head.
- **glbctx_head** (*dict*) – config to init global context head.

aug_test(*img_feats*, *proposal_list*, *img_metas*, *rescale=False*)

Test with augmentations.

If rescale is False, then returned bboxes and masks will fit the scale of imgs[0].

forward_train(*x*, *img_metas*, *proposal_list*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*, *gt_masks=None*, *gt_semantic_seg=None*)

Parameters

- **x** (*list[Tensor]*) – list of multi-level img features.
- **img_metas** (*list[dict]*) – list of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmdet/datasets/pipelines/formatting.py:Collect*.
- **proposal_list** (*list[Tensors]*) – list of region proposals.
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **gt_bboxes_ignore** (*None, list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss.
- **gt_masks** (*None, Tensor*) – true segmentation masks for each box used if the architecture supports a segmentation task.
- **gt_semantic_seg** (*None, list[Tensor]*) – semantic segmentation masks used if the architecture supports semantic segmentation task.

Returns a dictionary of loss components

Return type dict[str, Tensor]

init_mask_head(*mask_roi_extractor*, *mask_head*)

Initialize mask_head

simple_test(*x*, *proposal_list*, *img_metas*, *rescale=False*)

Test without augmentation.

Parameters

- **x** (*tuple[Tensor]*) – Features from upstream network. Each has shape (batch_size, c, h, w).
- **proposal_list** (*list(Tensor)*) – Proposals from rpn head. Each has shape (num_proposals, 5), last dimension 5 represent (x1, y1, x2, y2, score).
- **img_metas** (*list[dict]*) – Meta information of images.
- **rescale** (*bool*) – Whether to rescale the results to the original image. Default: True.

Returns When no mask branch, it is bbox results of each image and classes with type *list[list[np.ndarray]]*. The outer list corresponds to each image. The inner list corresponds to each class. When the model has mask branch, it contains bbox results and mask results. The outer list corresponds to each image, and first element of tuple is bbox results, second element is mask results.

Return type list[list[np.ndarray]] or list[tuple]

property with_feat_relay
whether the head has feature relay head

Type bool

property with_glbctx
whether the head has global context head

Type bool

property with_semantic
whether the head has semantic head

Type bool

class mmdet.models.roi_heads.**SCNetSemanticHead**(*conv_to_res=True*, ***kwargs*)
Mask head for SCNet.

Parameters **conv_to_res** (*bool, optional*) – if True, change the conv layers to SimplifiedBasicBlock.

class mmdet.models.roi_heads.**Shared2FCBBoxHead**(*fc_out_channels=1024*, **args*, ***kwargs*)

class mmdet.models.roi_heads.**Shared4Conv1FCBBoxHead**(*fc_out_channels=1024*, **args*, ***kwargs*)

class mmdet.models.roi_heads.**SingleRoIExtractor**(*roi_layer*, *out_channels*, *featmap_strides*, *finest_scale=56*, *init_cfg=None*)
Extract RoI features from a single level feature map.

If there are multiple input feature levels, each RoI is mapped to a level according to its scale. The mapping rule is proposed in FPN.

Parameters

- **roi_layer** (*dict*) – Specify RoI layer type and arguments.
- **out_channels** (*int*) – Output channels of RoI layers.
- **featmap_strides** (*List[int]*) – Strides of input feature maps.
- **finest_scale** (*int*) – Scale threshold of mapping to level 0. Default: 56.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

forward(*feats*, *rois*, *roi_scale_factor=None*)
Forward function.

map_roi_levels(*rois*, *num_levels*)
Map rois to corresponding feature levels by scales.

- scale < finest_scale * 2: level 0
- finest_scale * 2 <= scale < finest_scale * 4: level 1
- finest_scale * 4 <= scale < finest_scale * 8: level 2
- scale >= finest_scale * 8: level 3

Parameters

- **rois** (*Tensor*) – Input RoIs, shape (k, 5).
- **num_levels** (*int*) – Total level number.

Returns Level index (0-based) of each RoI, shape (k,)

Return type Tensor

class mmdet.models.roi_heads.**SparseRoIHead**(*num_stages=6*, *stage_loss_weights=(1, 1, 1, 1, 1, 1)*, *proposal_feature_channel=256*, *bbox_roi_extractor={'featmap_strides': [4, 8, 16, 32], 'out_channels': 256, 'roi_layer': {'output_size': 7, 'sampling_ratio': 2, 'type': 'RoIAlign'}, 'type': 'SingleRoIExtractor'}*, *mask_roi_extractor=None*, *bbox_head={'dropout': 0.0, 'feedforward_channels': 2048, 'ffn_act_cfg': {'inplace': True, 'type': 'ReLU'}, 'hidden_channels': 256, 'num_classes': 80, 'num_cls_fcs': 1, 'num_fcs': 2, 'num_heads': 8, 'num_reg_fcs': 3, 'roi_feat_size': 7, 'type': 'DIIHead'}*, *mask_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

The RoIHead for Sparse R-CNN: End-to-End Object Detection with Learnable Proposals and Instances as Queries

Parameters

- **num_stages** (*int*) – Number of stage whole iterative process. Defaults to 6.
- **stage_loss_weights** (*Tuple[float]*) – The loss weight of each stage. By default all stages have the same weight 1.
- **bbox_roi_extractor** (*dict*) – Config of box roi extractor.
- **mask_roi_extractor** (*dict*) – Config of mask roi extractor.
- **bbox_head** (*dict*) – Config of box head.
- **mask_head** (*dict*) – Config of mask head.
- **train_cfg** (*dict, optional*) – Configuration information in train stage. Defaults to None.
- **test_cfg** (*dict, optional*) – Configuration information in test stage. Defaults to None.
- **pretrained** (*str, optional*) – model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

aug_test(*features*, *proposal_list*, *img_metas*, *rescale=False*)
Test with augmentations.

If rescale is False, then returned bboxes and masks will fit the scale of imgs[0].

forward_dummy(*x*, *proposal_boxes*, *proposal_features*, *img_metas*)
Dummy forward function when do the flops computing.

forward_train(*x*, *proposal_boxes*, *proposal_features*, *img_metas*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*, *imgs_whwh=None*, *gt_masks=None*)
Forward function in training stage.

Parameters

- **x** (*list[Tensor]*) – list of multi-level img features.
- **proposals** (*Tensor*) – Decoded proposal bboxes, has shape (batch_size, num_proposals, 4)
- **proposal_features** (*Tensor*) – Expanded proposal features, has shape (batch_size, num_proposals, proposal_feature_channel)
- **img_metas** (*list[dict]*) – list of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmdet/datasets/pipelines/formatting.py:Collect*.
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss.
- **imgs_whwh** (*Tensor*) – Tensor with shape (batch_size, 4), the dimension means [img_width,img_height, img_width, img_height].
- **gt_masks** (*None | Tensor*) – true segmentation masks for each box used if the architecture supports a segmentation task.

Returns a dictionary of loss components of all stage.

Return type dict[str, Tensor]

simple_test(*x*, *proposal_boxes*, *proposal_features*, *img_metas*, *imgs_whwh*, *rescale=False*)
Test without augmentation.

Parameters

- **x** (*list[Tensor]*) – list of multi-level img features.
- **proposal_boxes** (*Tensor*) – Decoded proposal bboxes, has shape (batch_size, num_proposals, 4)
- **proposal_features** (*Tensor*) – Expanded proposal features, has shape (batch_size, num_proposals, proposal_feature_channel)
- **img_metas** (*dict*) – meta information of images.
- **imgs_whwh** (*Tensor*) – Tensor with shape (batch_size, 4), the dimension means [img_width,img_height, img_width, img_height].
- **rescale** (*bool*) – If True, return boxes in original image space. Defaults to False.

Returns When no mask branch, it is bbox results of each image and classes with type *list[list[np.ndarray]]*. The outer list corresponds to each image. The inner list corresponds to each class. When the model has a mask branch, it is a list[tuple] that contains bbox results

and mask results. The outer list corresponds to each image, and first element of tuple is bbox results, second element is mask results.

Return type list[list[np.ndarray]] or list[tuple]

class mmdet.models.roi_heads.**StandardRoIHead**(*bbox_roi_extractor=None*, *bbox_head=None*, *mask_roi_extractor=None*, *mask_head=None*, *shared_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

Simplest base roi head including one bbox head and one mask head.

async async_simple_test(*x*, *proposal_list*, *img_metas*, *proposals=None*, *rescale=False*)

Async test without augmentation.

aug_test(*x*, *proposal_list*, *img_metas*, *rescale=False*)

Test with augmentations.

If rescale is False, then returned bboxes and masks will fit the scale of imgs[0].

bbox_onnx_export(*x*, *img_metas*, *proposals*, *rcnn_test_cfg*, ***kwargs*)

Export bbox branch to onnx which supports batch inference.

Parameters

- **x** (*tuple[Tensor]*) – Feature maps of all scale level.
- **img_metas** (*list[dict]*) – Image meta info.
- **proposals** (*Tensor*) – Region proposals with batch dimension, has shape [N, num_bboxes, 5].
- **(obj** (*rcnn_test_cfg*) – *ConfigDict*): *test_cfg* of R-CNN.

Returns

bboxes of shape [N, num_bboxes, 5] and class labels of shape [N, num_bboxes].

Return type tuple[Tensor, Tensor]

forward_dummy(*x*, *proposals*)

Dummy forward function.

forward_train(*x*, *img_metas*, *proposal_list*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*, *gt_masks=None*, ***kwargs*)

Parameters

- **x** (*list[Tensor]*) – list of multi-level img features.
- **img_metas** (*list[dict]*) – list of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmdet/datasets/pipelines/formatting.py:Collect*.
- **proposals** (*list[Tensors]*) – list of region proposals.
- **gt_bboxes** (*list[Tensor]*) – Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) – class indices corresponding to each box
- **gt_bboxes_ignore** (*None | list[Tensor]*) – specify which bounding boxes can be ignored when computing the loss.

- **gt_masks** (*None | Tensor*) – true segmentation masks for each box used if the architecture supports a segmentation task.

Returns a dictionary of loss components

Return type dict[str, Tensor]

init_assigner_sampler()
Initialize assigner and sampler.

init_bbox_head(*bbox_roi_extractor*, *bbox_head*)
Initialize bbox_head

init_mask_head(*mask_roi_extractor*, *mask_head*)
Initialize mask_head

mask_onnx_export(*x*, *img_metas*, *det_bboxes*, *det_labels*, ***kwargs*)
Export mask branch to onnx which supports batch inference.

Parameters

- **x** (*tuple[Tensor]*) – Feature maps of all scale level.
- **img_metas** (*list[dict]*) – Image meta info.
- **det_bboxes** (*Tensor*) – Bboxes and corresponding scores. has shape [N, num_bboxes, 5].
- **det_labels** (*Tensor*) – class labels of shape [N, num_bboxes].

Returns

The segmentation results of shape [N, num_bboxes, image_height, image_width].

Return type Tensor

onnx_export(*x*, *proposals*, *img_metas*, *rescale=False*)
Test without augmentation.

simple_test(*x*, *proposal_list*, *img_metas*, *proposals=None*, *rescale=False*)
Test without augmentation.

Parameters

- **x** (*tuple[Tensor]*) – Features from upstream network. Each has shape (batch_size, c, h, w).
- **proposal_list** (*list(Tensor)*) – Proposals from rpn head. Each has shape (num_proposals, 5), last dimension 5 represent (x1, y1, x2, y2, score).
- **img_metas** (*list[dict]*) – Meta information of images.
- **rescale** (*bool*) – Whether to rescale the results to the original image. Default: True.

Returns When no mask branch, it is bbox results of each image and classes with type *list[list[np.ndarray]]*. The outer list corresponds to each image. The inner list corresponds to each class. When the model has mask branch, it contains bbox results and mask results. The outer list corresponds to each image, and first element of tuple is bbox results, second element is mask results.

Return type list[list[np.ndarray]] or list[tuple]

class mmdet.models.roi_heads.**TridentRoIHead**(*num_branch*, *test_branch_idx*, ***kwargs*)
Trident roi head.

Parameters

- **num_branch** (*int*) – Number of branches in TridentNet.
- **test_branch_idx** (*int*) – In inference, all 3 branches will be used if *test_branch_idx==-1*, otherwise only branch with index *test_branch_idx* will be used.

aug_test_bboxes(*feats*, *img_metas*, *proposal_list*, *rcnn_test_cfg*)
Test det bboxes with test time augmentation.

merge_trident_bboxes(*trident_det_bboxes*, *trident_det_labels*)
Merge bbox predictions of each branch.

simple_test(*x*, *proposal_list*, *img_metas*, *proposals=None*, *rescale=False*)
Test without augmentation as follows:

1. Compute prediction bbox and label per branch.
2. Merge predictions of each branch according to scores of bboxes, i.e., bboxes with higher score are kept to give top-k prediction.

40.6 losses

class mmdet.models.losses.**Accuracy**(*topk=(1)*, *thresh=None*)

forward(*pred*, *target*)
Forward function to calculate accuracy.

Parameters

- **pred** (*torch.Tensor*) – Prediction of models.
- **target** (*torch.Tensor*) – Target for each prediction.

Returns The accuracies under different topk criterions.

Return type tuple[float]

class mmdet.models.losses.**AssociativeEmbeddingLoss**(*pull_weight=0.25*, *push_weight=0.25*)
Associative Embedding Loss.

More details can be found in Associative Embedding and CornerNet . Code is modified from kp_utils.py # noqa: E501

Parameters

- **pull_weight** (*float*) – Loss weight for corners from same object.
- **push_weight** (*float*) – Loss weight for corners from different object.

forward(*pred*, *target*, *match*)
Forward function.

class mmdet.models.losses.**BalancedL1Loss**(*alpha=0.5*, *gamma=1.5*, *beta=1.0*, *reduction='mean'*, *loss_weight=1.0*)
Balanced L1 Loss.

arXiv: https://arxiv.org/pdf/1904.02701.pdf (CVPR 2019)

Parameters

- **alpha** (*float*) – The denominator `alpha` in the balanced L1 loss. Defaults to 0.5.
- **gamma** (*float*) – The `gamma` in the balanced L1 loss. Defaults to 1.5.

- **beta** (*float, optional*) – The loss is a piecewise function of prediction and target. beta serves as a threshold for the difference between the prediction and target. Defaults to 1.0.
- **reduction** (*str, optional*) – The method that reduces the loss to a scalar. Options are "none", "mean" and "sum".
- **loss_weight** (*float, optional*) – The weight of the loss. Defaults to 1.0

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*, ***kwargs*)
Forward function of loss.

Parameters

- **pred** (*torch.Tensor*) – The prediction with shape (N, 4).
- **target** (*torch.Tensor*) – The learning target of the prediction with shape (N, 4).
- **weight** (*torch.Tensor, optional*) – Sample-wise loss weight with shape (N,).
- **avg_factor** (*int, optional*) – Average factor that is used to average the loss. Defaults to None.
- **reduction_override** (*str, optional*) – The reduction method used to override the original reduction method of the loss. Options are "none", "mean" and "sum".

Returns The calculated loss

Return type torch.Tensor

class mmdet.models.losses.**BoundedIoULoss**(*beta=0.2*, *eps=0.001*, *reduction='mean'*, *loss_weight=1.0*)

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*, ***kwargs*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.losses.**CIoULoss**(*eps=1e-06*, *reduction='mean'*, *loss_weight=1.0*)

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*, ***kwargs*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.losses.**CrossEntropyLoss**(*use_sigmoid=False*, *use_mask=False*, *reduction='mean'*, *class_weight=None*, *ignore_index=None*, *loss_weight=1.0*)

forward(*cls_score*, *label*, *weight=None*, *avg_factor=None*, *reduction_override=None*, *ignore_index=None*, ***kwargs*)
Forward function.

Parameters

- **cls_score** (*torch.Tensor*) – The prediction.
- **label** (*torch.Tensor*) – The learning label of the prediction.
- **weight** (*torch.Tensor, optional*) – Sample-wise loss weight.
- **avg_factor** (*int, optional*) – Average factor that is used to average the loss. Defaults to None.
- **reduction_override** (*str, optional*) – The method used to reduce the loss. Options are "none", "mean" and "sum".
- **ignore_index** (*int | None*) – The label index to be ignored. If not None, it will override the default value. Default: None.

Returns The calculated loss.

Return type torch.Tensor

class mmdet.models.losses.**DIoULoss**(*eps=1e-06*, *reduction='mean'*, *loss_weight=1.0*)

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*, ***kwargs*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.losses.**DiceLoss**(*use_sigmoid=True*, *activate=True*, *reduction='mean'*, *loss_weight=1.0*, *eps=0.001*)

forward(*pred*, *target*, *weight=None*, *reduction_override=None*, *avg_factor=None*)
Forward function.

Parameters

- **pred** (*torch.Tensor*) – The prediction, has a shape (n, *).
- **target** (*torch.Tensor*) – The label of the prediction, shape (n, *), same shape of pred.
- **weight** (*torch.Tensor, optional*) – The weight of loss for each prediction, has a shape (n,). Defaults to None.
- **avg_factor** (*int, optional*) – Average factor that is used to average the loss. Defaults to None.
- **reduction_override** (*str, optional*) – The reduction method used to override the original reduction method of the loss. Options are "none", "mean" and "sum".

Returns The calculated loss

Return type torch.Tensor

class mmdet.models.losses.**DistributionFocalLoss**(*reduction='mean'*, *loss_weight=1.0*)
Distribution Focal Loss (DFL) is a variant of Generalized Focal Loss: Learning Qualified and Distributed Bounding Boxes for Dense Object Detection.

Parameters

- **reduction** (*str*) – Options are *'none'*, *'mean'* and *'sum'*.
- **loss_weight** (*float*) – Loss weight of current loss.

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*)
Forward function.

Parameters

- **pred** (*torch.Tensor*) – Predicted general distribution of bounding boxes (before softmax) with shape (N, n+1), n is the max value of the integral set *{0, ..., n}* in paper.
- **target** (*torch.Tensor*) – Target distance label for bounding boxes with shape (N,).
- **weight** (*torch.Tensor, optional*) – The weight of loss for each prediction. Defaults to None.
- **avg_factor** (*int, optional*) – Average factor that is used to average the loss. Defaults to None.
- **reduction_override** (*str, optional*) – The reduction method used to override the original reduction method of the loss. Defaults to None.

class mmdet.models.losses.**FocalLoss**(*use_sigmoid=True*, *gamma=2.0*, *alpha=0.25*, *reduction='mean'*, *loss_weight=1.0*)

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*)
Forward function.

Parameters

- **pred** (*torch.Tensor*) – The prediction.
- **target** (*torch.Tensor*) – The learning label of the prediction.
- **weight** (*torch.Tensor, optional*) – The weight of loss for each prediction. Defaults to None.
- **avg_factor** (*int, optional*) – Average factor that is used to average the loss. Defaults to None.
- **reduction_override** (*str, optional*) – The reduction method used to override the original reduction method of the loss. Options are "none", "mean" and "sum".

Returns The calculated loss

Return type torch.Tensor

class mmdet.models.losses.**GHMC**(*bins=10*, *momentum=0*, *use_sigmoid=True*, *loss_weight=1.0*, *reduction='mean'*)
GHM Classification Loss.

Details of the theorem can be viewed in the paper Gradient Harmonized Single-stage Detector.

Parameters

- **bins** (*int*) – Number of the unit regions for distribution calculation.
- **momentum** (*float*) – The parameter for moving average.
- **use_sigmoid** (*bool*) – Can only be true for BCE based loss now.
- **loss_weight** (*float*) – The weight of the total GHM-C loss.
- **reduction** (*str*) – Options are "none", "mean" and "sum". Defaults to "mean"

forward(*pred*, *target*, *label_weight*, *reduction_override=None*, ***kwargs*)
Calculate the GHM-C loss.

Parameters

- **pred** (*float tensor of size [batch_num, class_num]*) – The direct prediction of classification fc layer.
- **target** (*float tensor of size [batch_num, class_num]*) – Binary class target for each sample.
- **label_weight** (*float tensor of size [batch_num, class_num]*) – the value is 1 if the sample is valid and 0 if ignored.
- **reduction_override** (*str, optional*) – The reduction method used to override the original reduction method of the loss. Defaults to None.

Returns The gradient harmonized loss.

class mmdet.models.losses.**GHMR**(*mu=0.02*, *bins=10*, *momentum=0*, *loss_weight=1.0*, *reduction='mean'*)
GHM Regression Loss.

Details of the theorem can be viewed in the paper Gradient Harmonized Single-stage Detector.

Parameters

- **mu** (*float*) – The parameter for the Authentic Smooth L1 loss.
- **bins** (*int*) – Number of the unit regions for distribution calculation.
- **momentum** (*float*) – The parameter for moving average.
- **loss_weight** (*float*) – The weight of the total GHM-R loss.
- **reduction** (*str*) – Options are “none”, “mean” and “sum”. Defaults to “mean”

forward(*pred*, *target*, *label_weight*, *avg_factor=None*, *reduction_override=None*)
Calculate the GHM-R loss.

Parameters

- **pred** (*float tensor of size [batch_num, 4 (* class_num)]*) – The prediction of box regression layer. Channel number can be 4 or 4 * class_num depending on whether it is class-agnostic.
- **target** (*float tensor of size [batch_num, 4 (* class_num)]*) – The target regression values with the same size of pred.
- **label_weight** (*float tensor of size [batch_num, 4 (* class_num)]*) – The weight of each sample, 0 if ignored.
- **reduction_override** (*str, optional*) – The reduction method used to override the original reduction method of the loss. Defaults to None.

Returns The gradient harmonized loss.

class mmdet.models.losses.**GIoULoss**(*eps=1e-06*, *reduction='mean'*, *loss_weight=1.0*)

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*, ***kwargs*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

class mmdet.models.losses.**GaussianFocalLoss**(*alpha=2.0*, *gamma=4.0*, *reduction='mean'*, *loss_weight=1.0*)

GaussianFocalLoss is a variant of focal loss.

More details can be found in the paper Code is modified from kp_utils.py # noqa: E501 Please notice that the target in GaussianFocalLoss is a gaussian heatmap, not 0/1 binary target.

Parameters

- **alpha** (*float*) – Power of prediction.
- **gamma** (*float*) – Power of target for negative samples.
- **reduction** (*str*) – Options are "none", "mean" and "sum".
- **loss_weight** (*float*) – Loss weight of current loss.

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*)

Forward function.

Parameters

- **pred** (*torch.Tensor*) – The prediction.
- **target** (*torch.Tensor*) – The learning target of the prediction in gaussian distribution.
- **weight** (*torch.Tensor, optional*) – The weight of loss for each prediction. Defaults to None.
- **avg_factor** (*int, optional*) – Average factor that is used to average the loss. Defaults to None.
- **reduction_override** (*str, optional*) – The reduction method used to override the original reduction method of the loss. Defaults to None.

class mmdet.models.losses.**IoULoss**(*linear=False*, *eps=1e-06*, *reduction='mean'*, *loss_weight=1.0*, *mode='log'*)

IoULoss.

Computing the IoU loss between a set of predicted bboxes and target bboxes.

Parameters

- **linear** (*bool*) – If True, use linear scale of loss else determined by mode. Default: False.
- **eps** (*float*) – Eps to avoid log(0).
- **reduction** (*str*) – Options are "none", "mean" and "sum".
- **loss_weight** (*float*) – Weight of loss.
- **mode** (*str*) – Loss scaling mode, including "linear", "square", and "log". Default: 'log'

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*, ***kwargs*)

Forward function.

Parameters

- **pred** (*torch.Tensor*) – The prediction.
- **target** (*torch.Tensor*) – The learning target of the prediction.

- **weight** (*torch.Tensor, optional*) – The weight of loss for each prediction. Defaults to None.
- **avg_factor** (*int, optional*) – Average factor that is used to average the loss. Defaults to None.
- **reduction_override** (*str, optional*) – The reduction method used to override the original reduction method of the loss. Defaults to None. Options are "none", "mean" and "sum".

class mmdet.models.losses.**KnowledgeDistillationKLDivLoss**(*reduction='mean'*, *loss_weight=1.0*, *T=10*)

Loss function for knowledge distilling using KL divergence.

Parameters

- **reduction** (*str*) – Options are *'none'*, *'mean'* and *'sum'*.
- **loss_weight** (*float*) – Loss weight of current loss.
- **T** (*int*) – Temperature for distillation.

forward(*pred*, *soft_label*, *weight=None*, *avg_factor=None*, *reduction_override=None*)

Forward function.

Parameters

- **pred** (*Tensor*) – Predicted logits with shape (N, n + 1).
- **soft_label** (*Tensor*) – Target logits with shape (N, N + 1).
- **weight** (*torch.Tensor, optional*) – The weight of loss for each prediction. Defaults to None.
- **avg_factor** (*int, optional*) – Average factor that is used to average the loss. Defaults to None.
- **reduction_override** (*str, optional*) – The reduction method used to override the original reduction method of the loss. Defaults to None.

class mmdet.models.losses.**L1Loss**(*reduction='mean'*, *loss_weight=1.0*)

L1 loss.

Parameters

- **reduction** (*str, optional*) – The method to reduce the loss. Options are "none", "mean" and "sum".
- **loss_weight** (*float, optional*) – The weight of loss.

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*)

Forward function.

Parameters

- **pred** (*torch.Tensor*) – The prediction.
- **target** (*torch.Tensor*) – The learning target of the prediction.
- **weight** (*torch.Tensor, optional*) – The weight of loss for each prediction. Defaults to None.
- **avg_factor** (*int, optional*) – Average factor that is used to average the loss. Defaults to None.

- **reduction_override** (*str, optional*) – The reduction method used to override the original reduction method of the loss. Defaults to None.

class mmdet.models.losses.**MSELoss**(*reduction='mean'*, *loss_weight=1.0*)

MSELoss.

Parameters

- **reduction** (*str, optional*) – The method that reduces the loss to a scalar. Options are "none", "mean" and "sum".
- **loss_weight** (*float, optional*) – The weight of the loss. Defaults to 1.0

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*)

Forward function of loss.

Parameters

- **pred** (*torch.Tensor*) – The prediction.
- **target** (*torch.Tensor*) – The learning target of the prediction.
- **weight** (*torch.Tensor, optional*) – Weight of the loss for each prediction. Defaults to None.
- **avg_factor** (*int, optional*) – Average factor that is used to average the loss. Defaults to None.
- **reduction_override** (*str, optional*) – The reduction method used to override the original reduction method of the loss. Defaults to None.

Returns The calculated loss

Return type torch.Tensor

class mmdet.models.losses.**QualityFocalLoss**(*use_sigmoid=True*, *beta=2.0*, *reduction='mean'*, *loss_weight=1.0*)

Quality Focal Loss (QFL) is a variant of Generalized Focal Loss: Learning Qualified and Distributed Bounding Boxes for Dense Object Detection.

Parameters

- **use_sigmoid** (*bool*) – Whether sigmoid operation is conducted in QFL. Defaults to True.
- **beta** (*float*) – The beta parameter for calculating the modulating factor. Defaults to 2.0.
- **reduction** (*str*) – Options are "none", "mean" and "sum".
- **loss_weight** (*float*) – Loss weight of current loss.

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*)

Forward function.

Parameters

- **pred** (*torch.Tensor*) – Predicted joint representation of classification and quality (IoU) estimation with shape (N, C), C is the number of classes.
- **target** (*tuple([torch.Tensor])*) – Target category label with shape (N,) and target quality label with shape (N,).
- **weight** (*torch.Tensor, optional*) – The weight of loss for each prediction. Defaults to None.
- **avg_factor** (*int, optional*) – Average factor that is used to average the loss. Defaults to None.

- **reduction_override** (*str, optional*) – The reduction method used to override the original reduction method of the loss. Defaults to None.

class mmdet.models.losses.**SeesawLoss**(*use_sigmoid=False*, *p=0.8*, *q=2.0*, *num_classes=1203*, *eps=0.01*, *reduction='mean'*, *loss_weight=1.0*, *return_dict=True*)

Seesaw Loss for Long-Tailed Instance Segmentation (CVPR 2021) arXiv: https://arxiv.org/abs/2008.10032

Parameters

- **use_sigmoid** (*bool, optional*) – Whether the prediction uses sigmoid of softmax. Only False is supported.
- **p** (*float, optional*) – The p in the mitigation factor. Defaults to 0.8.
- **q** (*float, optional*) – The q in the compenstation factor. Defaults to 2.0.
- **num_classes** (*int, optional*) – The number of classes. Default to 1203 for LVIS v1 dataset.
- **eps** (*float, optional*) – The minimal value of divisor to smooth the computation of compensation factor
- **reduction** (*str, optional*) – The method that reduces the loss to a scalar. Options are "none", "mean" and "sum".
- **loss_weight** (*float, optional*) – The weight of the loss. Defaults to 1.0
- **return_dict** (*bool, optional*) – Whether return the losses as a dict. Default to True.

forward(*cls_score*, *labels*, *label_weights=None*, *avg_factor=None*, *reduction_override=None*)

Forward function.

Parameters

- **cls_score** (*torch.Tensor*) – The prediction with shape (N, C + 2).
- **labels** (*torch.Tensor*) – The learning label of the prediction.
- **label_weights** (*torch.Tensor, optional*) – Sample-wise loss weight.
- **avg_factor** (*int, optional*) – Average factor that is used to average the loss. Defaults to None.
- **reduction** (*str, optional*) – The method used to reduce the loss. Options are "none", "mean" and "sum".

Returns if return_dict == False: The calculated loss | if return_dict == True: The dict of calculated losses for objectness and classes, respectively.

Return type torch.Tensor | Dict [str, torch.Tensor]

get_accuracy(*cls_score*, *labels*)

Get custom accuracy w.r.t. cls_score and labels.

Parameters

- **cls_score** (*torch.Tensor*) – The prediction with shape (N, C + 2).
- **labels** (*torch.Tensor*) – The learning label of the prediction.

Returns

The accuracy for objectness and classes, respectively.

Return type Dict [str, torch.Tensor]

get_activation(*cls_score*)
Get custom activation of cls_score.

Parameters **cls_score** (*torch.Tensor*) – The prediction with shape (N, C + 2).

Returns

The custom activation of cls_score with shape (N, C + 1).

Return type torch.Tensor

get_cls_channels(*num_classes*)
Get custom classification channels.

Parameters **num_classes** (*int*) – The number of classes.

Returns The custom classification channels.

Return type int

class mmdet.models.losses.**SmoothL1Loss**(*beta=1.0*, *reduction='mean'*, *loss_weight=1.0*)
Smooth L1 loss.

Parameters

- **beta** (*float, optional*) – The threshold in the piecewise function. Defaults to 1.0.
- **reduction** (*str, optional*) – The method to reduce the loss. Options are "none", "mean" and "sum". Defaults to "mean".
- **loss_weight** (*float, optional*) – The weight of loss.

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*, ***kwargs*)
Forward function.

Parameters

- **pred** (*torch.Tensor*) – The prediction.
- **target** (*torch.Tensor*) – The learning target of the prediction.
- **weight** (*torch.Tensor, optional*) – The weight of loss for each prediction. Defaults to None.
- **avg_factor** (*int, optional*) – Average factor that is used to average the loss. Defaults to None.
- **reduction_override** (*str, optional*) – The reduction method used to override the original reduction method of the loss. Defaults to None.

class mmdet.models.losses.**VarifocalLoss**(*use_sigmoid=True*, *alpha=0.75*, *gamma=2.0*, *iou_weighted=True*, *reduction='mean'*, *loss_weight=1.0*)

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*)
Forward function.

Parameters

- **pred** (*torch.Tensor*) – The prediction.
- **target** (*torch.Tensor*) – The learning target of the prediction.
- **weight** (*torch.Tensor, optional*) – The weight of loss for each prediction. Defaults to None.

- **avg_factor** (*int, optional*) – Average factor that is used to average the loss. Defaults to None.
- **reduction_override** (*str, optional*) – The reduction method used to override the original reduction method of the loss. Options are "none", "mean" and "sum".

Returns The calculated loss

Return type torch.Tensor

mmdet.models.losses.**binary_cross_entropy**(*pred*, *label*, *weight=None*, *reduction='mean'*, *avg_factor=None*, *class_weight=None*, *ignore_index=- 100*)

Calculate the binary CrossEntropy loss.

Parameters

- **pred** (*torch.Tensor*) – The prediction with shape (N, 1).
- **label** (*torch.Tensor*) – The learning label of the prediction.
- **weight** (*torch.Tensor, optional*) – Sample-wise loss weight.
- **reduction** (*str, optional*) – The method used to reduce the loss. Options are "none", "mean" and "sum".
- **avg_factor** (*int, optional*) – Average factor that is used to average the loss. Defaults to None.
- **class_weight** (*list[float], optional*) – The weight for each class.
- **ignore_index** (*int | None*) – The label index to be ignored. If None, it will be set to default value. Default: -100.

Returns The calculated loss.

Return type torch.Tensor

mmdet.models.losses.**cross_entropy**(*pred*, *label*, *weight=None*, *reduction='mean'*, *avg_factor=None*, *class_weight=None*, *ignore_index=- 100*)

Calculate the CrossEntropy loss.

Parameters

- **pred** (*torch.Tensor*) – The prediction with shape (N, C), C is the number of classes.
- **label** (*torch.Tensor*) – The learning label of the prediction.
- **weight** (*torch.Tensor, optional*) – Sample-wise loss weight.
- **reduction** (*str, optional*) – The method used to reduce the loss.
- **avg_factor** (*int, optional*) – Average factor that is used to average the loss. Defaults to None.
- **class_weight** (*list[float], optional*) – The weight for each class.
- **ignore_index** (*int | None*) – The label index to be ignored. If None, it will be set to default value. Default: -100.

Returns The calculated loss

Return type torch.Tensor

mmdet.models.losses.**mask_cross_entropy**(*pred*, *target*, *label*, *reduction='mean'*, *avg_factor=None*, *class_weight=None*, *ignore_index=None*)

Calculate the CrossEntropy loss for masks.

Parameters

- **pred** (*torch.Tensor*) – The prediction with shape (N, C, *), C is the number of classes. The trailing * indicates arbitrary shape.
- **target** (*torch.Tensor*) – The learning label of the prediction.
- **label** (*torch.Tensor*) – `label` indicates the class label of the mask corresponding object. This will be used to select the mask in the of the class which the object belongs to when the mask prediction if not class-agnostic.
- **reduction** (*str, optional*) – The method used to reduce the loss. Options are "none", "mean" and "sum".
- **avg_factor** (*int, optional*) – Average factor that is used to average the loss. Defaults to None.
- **class_weight** (*list[float], optional*) – The weight for each class.
- **ignore_index** (*None*) – Placeholder, to be consistent with other loss. Default: None.

Returns The calculated loss

Return type torch.Tensor

Example

```
>>> N, C = 3, 11
>>> H, W = 2, 2
>>> pred = torch.randn(N, C, H, W) * 1000
>>> target = torch.rand(N, H, W)
>>> label = torch.randint(0, C, size=(N,))
>>> reduction = 'mean'
>>> avg_factor = None
>>> class_weights = None
>>> loss = mask_cross_entropy(pred, target, label, reduction,
>>>                           avg_factor, class_weights)
>>> assert loss.shape == (1,)
```

mmdet.models.losses.**mse_loss**(*pred*, *target*)
Warpper of mse loss.

mmdet.models.losses.**reduce_loss**(*loss*, *reduction*)
Reduce loss as specified.

Parameters

- **loss** (*Tensor*) – Elementwise loss tensor.
- **reduction** (*str*) – Options are "none", "mean" and "sum".

Returns Reduced loss tensor.

Return type Tensor

mmdet.models.losses.**sigmoid_focal_loss**(*pred*, *target*, *weight=None*, *gamma=2.0*, *alpha=0.25*, *reduction='mean'*, *avg_factor=None*)
A warpper of cuda version Focal Loss.

Parameters

- **pred** (*torch.Tensor*) – The prediction with shape (N, C), C is the number of classes.

- **target** (*torch.Tensor*) – The learning label of the prediction.
- **weight** (*torch.Tensor, optional*) – Sample-wise loss weight.
- **gamma** (*float, optional*) – The gamma for calculating the modulating factor. Defaults to 2.0.
- **alpha** (*float, optional*) – A balanced form for Focal Loss. Defaults to 0.25.
- **reduction** (*str, optional*) – The method used to reduce the loss into a scalar. Defaults to 'mean'. Options are "none", "mean" and "sum".
- **avg_factor** (*int, optional*) – Average factor that is used to average the loss. Defaults to None.

mmdet.models.losses.**weighted_loss**(*loss_func*)

Create a weighted version of a given loss function.

To use this decorator, the loss function must have the signature like *loss_func(pred, target, **kwargs)*. The function only needs to compute element-wise loss without any reduction. This decorator will add weight and reduction arguments to the function. The decorated function will have the signature like *loss_func(pred, target, weight=None, reduction='mean', avg_factor=None, **kwargs)*.

Example

```
>>> import torch
>>> @weighted_loss
>>> def l1_loss(pred, target):
>>>     return (pred - target).abs()
```

```
>>> pred = torch.Tensor([0, 2, 3])
>>> target = torch.Tensor([1, 1, 1])
>>> weight = torch.Tensor([1, 0, 1])
```

```
>>> l1_loss(pred, target)
tensor(1.3333)
>>> l1_loss(pred, target, weight)
tensor(1.)
>>> l1_loss(pred, target, reduction='none')
tensor([1., 1., 2.])
>>> l1_loss(pred, target, weight, avg_factor=2)
tensor(1.5000)
```

40.7 utils

class mmdet.models.utils.**AdaptiveAvgPool2d**(*output_size: Union[int, None, Tuple[Optional[int], ...]]*)

Handle empty batch dimension to AdaptiveAvgPool2d.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while

the latter silently ignores them.

---

class mmdet.models.utils.**CSPLayer**(*in_channels*, *out_channels*, *expand_ratio=0.5*, *num_blocks=1*, *add_identity=True*, *use_depthwise=False*, *conv_cfg=None*, *norm_cfg={'eps': 0.001, 'momentum': 0.03, 'type': 'BN'}*, *act_cfg={'type': 'Swish'}*, *init_cfg=None*)

Cross Stage Partial Layer.

Parameters

- **in_channels** (*int*) – The input channels of the CSP layer.
- **out_channels** (*int*) – The output channels of the CSP layer.
- **expand_ratio** (*float*) – Ratio to adjust the number of channels of the hidden layer. Default: 0.5
- **num_blocks** (*int*) – Number of blocks. Default: 1
- **add_identity** (*bool*) – Whether to add identity in blocks. Default: True
- **use_depthwise** (*bool*) – Whether to depthwise separable convolution in blocks. Default: False
- **conv_cfg** (*dict, optional*) – Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: dict(type='BN')
- **act_cfg** (*dict*) – Config dict for activation layer. Default: dict(type='Swish')

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.utils.**ConvUpsample**(*in_channels*, *inner_channels*, *num_layers=1*, *num_upsample=None*, *conv_cfg=None*, *norm_cfg=None*, *init_cfg=None*, ***kwargs*)

ConvUpsample performs 2x upsampling after Conv.

There are several *ConvModule* layers. In the first few layers, upsampling will be applied after each layer of convolution. The number of upsampling must be no more than the number of ConvModule layers.

Parameters

- **in_channels** (*int*) – Number of channels in the input feature map.
- **inner_channels** (*int*) – Number of channels produced by the convolution.
- **num_layers** (*int*) – Number of convolution layers.
- **num_upsample** (*int | optional*) – Number of upsampling layer. Must be no more than num_layers. Upsampling will be applied after the first `num_upsample` layers of convolution. Default: `num_layers`.
- **conv_cfg** (*dict*) – Config dict for convolution layer. Default: None, which means using conv2d.

- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: None.
- **init_cfg** (*dict*) – Config dict for initialization. Default: None.
- **kwargs** (*key word augments*) – Other augments used in ConvModule.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.utils.**DetrTransformerDecoder**(**args*, *post_norm_cfg={'type': 'LN'}*, *return_intermediate=False*, ***kwargs*)

Implements the decoder in DETR transformer.

Parameters

- **return_intermediate** (*bool*) – Whether to return intermediate outputs.
- **post_norm_cfg** (*dict*) – Config of last normalization layer. Default *LN*.

forward(*query*, **args*, ***kwargs*)

Forward function for *TransformerDecoder*.

Parameters **query** (*Tensor*) – Input query with shape *(num_query, bs, embed_dims)*.

Returns

Results with shape [1, num_query, bs, embed_dims] when return_intermediate is *False*, otherwise it has shape [num_layers, num_query, bs, embed_dims].

Return type Tensor

class mmdet.models.utils.**DetrTransformerDecoderLayer**(*attn_cfgs*, *feedforward_channels*, *ffn_dropout=0.0*, *operation_order=None*, *act_cfg={'inplace': True, 'type': 'ReLU'}*, *norm_cfg={'type': 'LN'}*, *ffn_num_fcs=2*, ***kwargs*)

Implements decoder layer in DETR transformer.

Parameters

- **attn_cfgs** (list[*mmcv.ConfigDict*] | list[dict] | dict)) – Configs for self_attention or cross_attention, the order should be consistent with it in *operation_order*. If it is a dict, it would be expand to the number of attention in *operation_order*.
- **feedforward_channels** (*int*) – The hidden dimension for FFNs.
- **ffn_dropout** (*float*) – Probability of an element to be zeroed in ffn. Default 0.0.
- **operation_order** (*tuple[str]*) – The execution order of operation in transformer. Such as ('self_attn', 'norm', 'ffn', 'norm'). DefaultNone
- **act_cfg** (*dict*) – The activation config for FFNs. Default: *LN*
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: *LN*.
- **ffn_num_fcs** (*int*) – The number of fully-connected layers in FFNs. Default2.

class mmdet.models.utils.**DynamicConv**(*in_channels=256*, *feat_channels=64*, *out_channels=None*, *input_feat_shape=7*, *with_proj=True*, *act_cfg={'inplace': True, 'type': 'ReLU'}*, *norm_cfg={'type': 'LN'}*, *init_cfg=None*)

Implements Dynamic Convolution.

This module generate parameters for each sample and use bmm to implement 1*1 convolution. Code is modified from the official github repo .

Parameters

- **in_channels** (*int*) – The input feature channel. Defaults to 256.
- **feat_channels** (*int*) – The inner feature channel. Defaults to 64.
- **out_channels** (*int, optional*) – The output feature channel. When not specified, it will be set to *in_channels* by default
- **input_feat_shape** (*int*) – The shape of input feature. Defaults to 7.
- **with_proj** (*bool*) – Project two-dimentional feature to one-dimentional feature. Default to True.
- **act_cfg** (*dict*) – The activation config for DynamicConv.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default layer normalization.
- **(obj** (*init_cfg*) – *mmcv.ConfigDict*): The Config for initialization. Default: None.

forward(*param_feature*, *input_feature*)

Forward function for *DynamicConv*.

Parameters

- **param_feature** (*Tensor*) – The feature can be used to generate the parameter, has shape (num_all_proposals, in_channels).
- **input_feature** (*Tensor*) – Feature that interact with parameters, has shape (num_all_proposals, in_channels, H, W).

Returns The output feature has shape (num_all_proposals, out_channels).

Return type Tensor

class mmdet.models.utils.**InvertedResidual**(*in_channels*, *out_channels*, *mid_channels*, *kernel_size=3*, *stride=1*, *se_cfg=None*, *with_expand_conv=True*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *with_cp=False*, *init_cfg=None*)

Inverted Residual Block.

Parameters

- **in_channels** (*int*) – The input channels of this Module.
- **out_channels** (*int*) – The output channels of this Module.
- **mid_channels** (*int*) – The input channels of the depthwise convolution.
- **kernel_size** (*int*) – The kernel size of the depthwise convolution. Default: 3.
- **stride** (*int*) – The stride of the depthwise convolution. Default: 1.
- **se_cfg** (*dict*) – Config dict for se layer. Default: None, which means no se layer.
- **with_expand_conv** (*bool*) – Use expand conv or not. If set False, mid_channels must be the same with in_channels. Default: True.

- **conv_cfg** (*dict*) – Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) – Config dict for normalization layer. Default: dict(type='BN').
- **act_cfg** (*dict*) – Config dict for activation layer. Default: dict(type='ReLU').
- **with_cp** (*bool*) – Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

Returns The output tensor.

Return type Tensor

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.utils.**LearnedPositionalEncoding**(*num_feats*, *row_num_embed=50*, *col_num_embed=50*, *init_cfg={'layer': 'Embedding', 'type': 'Uniform'}*)

Position embedding with learnable embedding weights.

Parameters

- **num_feats** (*int*) – The feature dimension for each position along x-axis or y-axis. The final returned dimension for each position is 2 times of this value.
- **row_num_embed** (*int, optional*) – The dictionary size of row embeddings. Default 50.
- **col_num_embed** (*int, optional*) – The dictionary size of col embeddings. Default 50.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict.

forward(*mask*)

Forward function for *LearnedPositionalEncoding*.

Parameters **mask** (*Tensor*) – ByteTensor mask. Non-zero values representing ignored positions, while zero values means valid positions for this image. Shape [bs, h, w].

Returns

Returned position embedding with shape [bs, num_feats*2, h, w].

Return type pos (Tensor)

class mmdet.models.utils.**NormedConv2d**(**args*, *tempearture=20*, *power=1.0*, *eps=1e-06*, *norm_over_kernel=False*, ***kwargs*)

Normalized Conv2d Layer.

Parameters

- **tempeature** (*float, optional*) – Tempeature term. Default to 20.
- **power** (*int, optional*) – Power term. Default to 1.0.

- **eps** (*float, optional*) – The minimal value of divisor to keep numerical stability. Default to 1e-6.
- **norm_over_kernel** (*bool, optional*) – Normalize over kernel. Default to False.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.utils.**NormedLinear**(**args*, *tempearture=20*, *power=1.0*, *eps=1e-06*, ***kwargs*)

Normalized Linear Layer.

Parameters

- **tempeature** (*float, optional*) – Tempeature term. Default to 20.
- **power** (*int, optional*) – Power term. Default to 1.0.
- **eps** (*float, optional*) – The minimal value of divisor to keep numerical stability. Default to 1e-6.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.utils.**PatchEmbed**(*in_channels=3*, *embed_dims=768*, *conv_type='Conv2d'*, *kernel_size=16*, *stride=16*, *padding='corner'*, *dilation=1*, *bias=True*, *norm_cfg=None*, *input_size=None*, *init_cfg=None*)

Image to Patch Embedding.

We use a conv layer to implement PatchEmbed.

Parameters

- **in_channels** (*int*) – The num of input channels. Default: 3
- **embed_dims** (*int*) – The dimensions of embedding. Default: 768
- **conv_type** (*str*) – The config dict for embedding conv layer type selection. Default: "Conv2d.
- **kernel_size** (*int*) – The kernel_size of embedding conv. Default: 16.
- **stride** (*int*) – The slide stride of embedding conv. Default: None (Would be set as *kernel_size*).
- **padding** (*int | tuple | string*) – The padding length of embedding conv. When it is a string, it means the mode of adaptive padding, support "same" and "corner" now. Default: "corner".
- **dilation** (*int*) – The dilation rate of embedding conv. Default: 1.

- **bias** (*bool*) – Bias of embed conv. Default: True.
- **norm_cfg** (*dict, optional*) – Config dict for normalization layer. Default: None.
- **input_size** (*int | tuple | None*) – The size of input, which will be used to calculate the out size. Only work when *dynamic_size* is False. Default: None.
- **init_cfg** (*mmcv.ConfigDict*, optional) – The Config for initialization. Default: None.

forward(*x*)

Parameters **x** (*Tensor*) – Has shape (B, C, H, W). In most case, C is 3.

Returns

Contains merged results and its spatial shape.

- x (Tensor): Has shape (B, out_h * out_w, embed_dims)
- **out_size (tuple[int]): Spatial shape of x, arrange as** (out_h, out_w).

Return type tuple

class mmdet.models.utils.**ResLayer**(*block*, *inplanes*, *planes*, *num_blocks*, *stride=1*, *avg_down=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *downsample_first=True*, ***kwargs*)

ResLayer to build ResNet style backbone.

Parameters

- **block** (*nn.Module*) – block used to build ResLayer.
- **inplanes** (*int*) – inplanes of block.
- **planes** (*int*) – planes of block.
- **num_blocks** (*int*) – number of blocks.
- **stride** (*int*) – stride of the first block. Default: 1
- **avg_down** (*bool*) – Use AvgPool instead of stride conv when downsampling in the bottleneck. Default: False
- **conv_cfg** (*dict*) – dictionary to construct and config conv layer. Default: None
- **norm_cfg** (*dict*) – dictionary to construct and config norm layer. Default: dict(type='BN')
- **downsample_first** (*bool*) – Downsample at the first block or last block. False for Hourglass, True for ResNet. Default: True

class mmdet.models.utils.**SELayer**(*channels*, *ratio=16*, *conv_cfg=None*, *act_cfg=({'type': 'ReLU'}, {'type': 'Sigmoid'})*, *init_cfg=None*)

Squeeze-and-Excitation Module.

Parameters

- **channels** (*int*) – The input (and output) channels of the SE layer.
- **ratio** (*int*) – Squeeze ratio in SELayer, the intermediate channel will be `int(channels/ratio)`. Default: 16.
- **conv_cfg** (*None or dict*) – Config dict for convolution layer. Default: None, which means using conv2d.

- **act_cfg** (*dict or Sequence[dict]*) – Config dict for activation layer. If act_cfg is a dict, two activation layers will be configurated by this dict. If act_cfg is a sequence of dicts, the first activation layer will be configurated by the first dict and the second activation layer will be configurated by the second dict. Default: (dict(type='ReLU'), dict(type='Sigmoid'))
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.utils.**SimplifiedBasicBlock**(*inplanes*, *planes*, *stride=1*, *dilation=1*, *downsample=None*, *style='pytorch'*, *with_cp=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *dcn=None*, *plugins=None*, *init_fg=None*)

Simplified version of original basic residual block. This is used in SCNet.

- Norm layer is now optional
- Last ReLU in forward function is removed

forward(*x*)

Forward function.

property norm1

normalization layer after the first convolution layer

Type nn.Module

property norm2

normalization layer after the second convolution layer

Type nn.Module

class mmdet.models.utils.**SinePositionalEncoding**(*num_feats*, *temperature=10000*, *normalize=False*, *scale=6.283185307179586*, *eps=1e-06*, *offset=0.0*, *init_cfg=None*)

Position encoding with sine and cosine functions.

See End-to-End Object Detection with Transformers for details.

Parameters

- **num_feats** (*int*) – The feature dimension for each position along x-axis or y-axis. Note the final returned dimension for each position is 2 times of this value.
- **temperature** (*int, optional*) – The temperature used for scaling the position embedding. Defaults to 10000.
- **normalize** (*bool, optional*) – Whether to normalize the position embedding. Defaults to False.
- **scale** (*float, optional*) – A scale factor that scales the position embedding. The scale will be used only when *normalize* is True. Defaults to 2*pi.
- **eps** (*float, optional*) – A value added to the denominator for numerical stability. Defaults to 1e-6.

- **offset** (*float*) – offset add to embed when do the normalization. Defaults to 0.
- **init_cfg** (*dict or list[dict], optional*) – Initialization config dict. Default: None

forward(*mask*)
Forward function for *SinePositionalEncoding*.

Parameters **mask** (*Tensor*) – ByteTensor mask. Non-zero values representing ignored positions, while zero values means valid positions for this image. Shape [bs, h, w].

Returns

Returned position embedding with shape [bs, num_feats*2, h, w].

Return type pos (Tensor)

class mmdet.models.utils.**Transformer**(*encoder=None*, *decoder=None*, *init_cfg=None*)
Implements the DETR transformer.

Following the official DETR implementation, this module copy-paste from torch.nn.Transformer with modifications:

- positional encodings are passed in MultiheadAttention
- extra LN at the end of encoder is removed
- decoder returns a stack of activations from all decoding layers

See paper: End-to-End Object Detection with Transformers for details.

Parameters

- **encoder** (*mmcv.ConfigDict* | Dict) – Config of TransformerEncoder. Defaults to None.
- **decoder** ((*mmcv.ConfigDict* | Dict)) – Config of TransformerDecoder. Defaults to None
- **(obj** (*init_cfg*) – *mmcv.ConfigDict*): The Config for initialization. Defaults to None.

forward(*x*, *mask*, *query_embed*, *pos_embed*)
Forward function for *Transformer*.

Parameters

- **x** (*Tensor*) – Input query with shape [bs, c, h, w] where c = embed_dims.
- **mask** (*Tensor*) – The key_padding_mask used for encoder and decoder, with shape [bs, h, w].
- **query_embed** (*Tensor*) – The query embedding for decoder, with shape [num_query, c].
- **pos_embed** (*Tensor*) – The positional encoding for encoder and decoder, with the same shape as *x*.

Returns

results of decoder containing the following tensor.

- **out_dec: Output from decoder. If return_intermediate_dec is True output has shape [num_dec_layers, bs,** num_query, embed_dims], else has shape [1, bs, num_query, embed_dims].
- memory: Output results from encoder, with shape [bs, embed_dims, h, w].

Return type tuple[Tensor]

init_weights()
Initialize the weights.

mmdet.models.utils.**adaptive_avg_pool2d**(*input*, *output_size*)
Handle empty batch dimension to adaptive_avg_pool2d.

Parameters

- **input** (*tensor*) – 4D tensor.
- **output_size** (*int, tuple[int,int]*) – the target output size.

mmdet.models.utils.**build_linear_layer**(*cfg*, **args*, ***kwargs*)
Build linear layer. :param cfg: The linear layer config, which should contain:

- type (str): Layer type.
- layer args: Args needed to instantiate an linear layer.

Parameters

- **args** (*argument list*) – Arguments passed to the *__init__* method of the corresponding linear layer.
- **kwargs** (*keyword arguments*) – Keyword arguments passed to the *__init__* method of the corresponding linear layer.

Returns Created linear layer.

Return type nn.Module

mmdet.models.utils.**build_transformer**(*cfg*, *default_args=None*)
Builder for Transformer.

mmdet.models.utils.**gaussian_radius**(*det_size*, *min_overlap*)
Generate 2D gaussian radius.

This function is modified from the official github repo.

Given `min_overlap`, radius could computed by a quadratic equation according to Vieta's formulas.

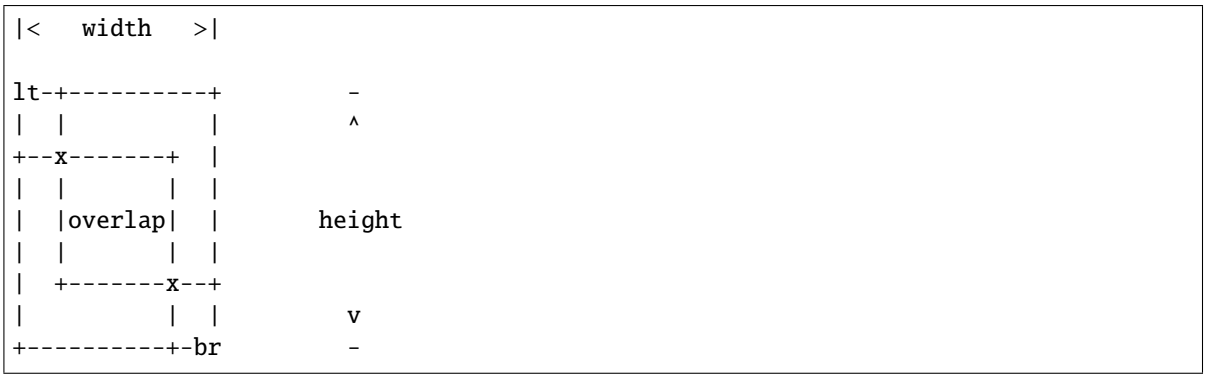
There are 3 cases for computing gaussian radius, details are following:

- Explanation of figure: `lt` and `br` indicates the left-top and bottom-right corner of ground truth box. `x` indicates the generated corner at the limited position when `radius=r`.
- Case1: one corner is inside the gt box and the other is outside.

To ensure IoU of generated box and gt box is larger than `min_overlap`:

$$\frac{(w-r)*(h-r)}{w*h+(w+h)r-r^2} \geq iou \quad \Rightarrow \quad r^2-(w+h)r+\frac{1-iou}{1+iou}*w*h \geq 0$$
$$a=1, \quad b=-(w+h), \quad c=\frac{1-iou}{1+iou}*w*hr \leq \frac{-b-\sqrt{b^2-4*a*c}}{2*a}$$

- Case2: both two corners are inside the gt box.



To ensure IoU of generated box and gt box is larger than `min_overlap`:

$$\frac{(w-2*r)*(h-2*r)}{w*h} \geq iou \quad \Rightarrow \quad 4r^2-2(w+h)r+(1-iou)*w*h \geq 0$$
$$a=4, \quad b=-2(w+h), \quad c=(1-iou)*w*hr \leq \frac{-b-\sqrt{b^2-4*a*c}}{2*a}$$

- Case3: both two corners are outside the gt box.

To ensure IoU of generated box and gt box is larger than `min_overlap`:

$$\frac{w*h}{(w+2*r)*(h+2*r)} \geq iou \quad \Rightarrow \quad 4*iou*r^2+2*iou*(w+h)r+(iou-1)*w*h \leq 0$$
$$a=4*iou, \quad b=2*iou*(w+h), \quad c=(iou-1)*w*h$$
$$r \leq \frac{-b+\sqrt{b^2-4*a*c}}{2*a}$$

Parameters

- **det_size** (*list[int]*) – Shape of object.

- **min_overlap** (*float*) – Min IoU with ground truth for boxes generated by keypoints inside the gaussian kernel.

Returns Radius of gaussian kernel.

Return type radius (int)

mmdet.models.utils.**gen_gaussian_target**(*heatmap*, *center*, *radius*, *k=1*)
Generate 2D gaussian heatmap.

Parameters

- **heatmap** (*Tensor*) – Input heatmap, the gaussian kernel will cover on it and maintain the max value.
- **center** (*list[int]*) – Coord of gaussian kernel's center.
- **radius** (*int*) – Radius of gaussian kernel.
- **k** (*int*) – Coefficient of gaussian kernel. Default: 1.

Returns Updated heatmap covered by gaussian kernel.

Return type out_heatmap (Tensor)

mmdet.models.utils.**interpolate_as**(*source*, *target*, *mode='bilinear'*, *align_corners=False*)
Interpolate the *source* to the shape of the *target*.

The *source* must be a Tensor, but the *target* can be a Tensor or a np.ndarray with the shape (..., target_h, target_w).

Parameters

- **source** (*Tensor*) – A 3D/4D Tensor with the shape (N, H, W) or (N, C, H, W).
- **target** (*Tensor | np.ndarray*) – The interpolation target with the shape (..., target_h, target_w).
- **mode** (*str*) – Algorithm used for interpolation. The options are the same as those in F.interpolate(). Default: `'bilinear'`.
- **align_corners** (*bool*) – The same as the argument in F.interpolate().

Returns The interpolated source Tensor.

Return type Tensor

mmdet.models.utils.**make_divisible**(*value*, *divisor*, *min_value=None*, *min_ratio=0.9*)
Make divisible function.

This function rounds the channel number to the nearest value that can be divisible by the divisor. It is taken from the original tf repo. It ensures that all layers have a channel number that is divisible by divisor. It can be seen here: https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet/mobilenet.py # noqa

Parameters

- **value** (*int*) – The original channel number.
- **divisor** (*int*) – The divisor to fully divide the channel number.
- **min_value** (*int*) – The minimum value of the output channel. Default: None, means that the minimum value equal to the divisor.
- **min_ratio** (*float*) – The minimum ratio of the rounded channel number to the original channel number. Default: 0.9.

Returns The modified output channel number.

Return type int

mmdet.models.utils.**nchw_to_nlc**(*x*)
Flatten [N, C, H, W] shape tensor to [N, L, C] shape tensor.

Parameters **x** (*Tensor*) – The input tensor of shape [N, C, H, W] before conversion.

Returns The output tensor of shape [N, L, C] after conversion.

Return type Tensor

mmdet.models.utils.**nlc_to_nchw**(*x*, *hw_shape*)
Convert [N, L, C] shape tensor to [N, C, H, W] shape tensor.

Parameters

- **x** (*Tensor*) – The input tensor of shape [N, L, C] before conversion.
- **hw_shape** (*Sequence[int]*) – The height and width of output feature map.

Returns The output tensor of shape [N, C, H, W] after conversion.

Return type Tensor

CHAPTER

FORTYONE

MMDET.UTILS

CHAPTER

FORTYTWO

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

m

INDEX

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y