MMDetection

Release 3.0.0rc0

MMDetection Authors

Sep 30, 2022

GET STARTED

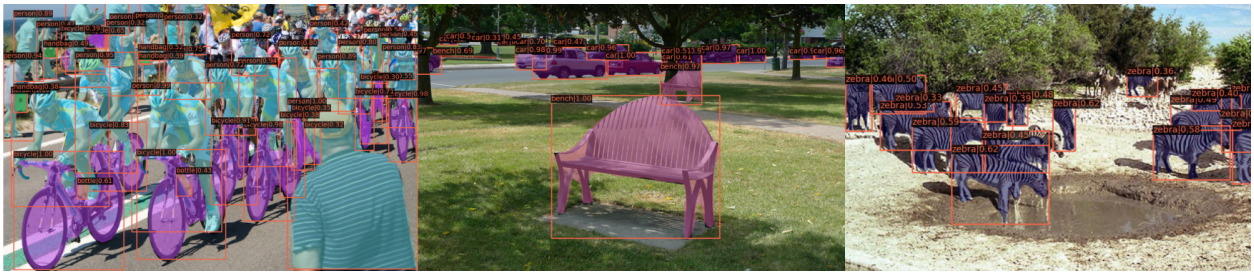
CHAPTER

ONE

OVERVIEW

This chapter introduces you to the framework of MMDetection, and provides links to detailed tutorials about MMDetection.

1.1 What is MMDetection



MMDetection is an object detection toolbox that contains a rich set of object detection, instance segmentation, and panoptic segmentation methods as well as related components and modules, and below is its whole framework:

MMDetection consists of 7 main parts, apis, structures, datasets, models, engine, evaluation and visualization.

- **apis** provides high-level APIs for model inference.
- **structures** provides data structures like bbox, mask, and DetDataSample.
- **datasets** supports various dataset for object detection, instance segmentation, and panoptic segmentation.
  - **transforms** contains a lot of useful data augmentation transforms .
  - **samplers** defines different data loader sampling strategy.
- **models** is the most vital part for detectors and contains different components of a detector.
  - **detectors** defines all of the detection model classes.
  - **data_preprocessors** is for preprocessing the input data of the model.
  - **backbones** contains various backbone networks
  - **necks** contains various neck components
  - **dense_heads** contains various detection heads that perform dense predictions.
  - **roi_heads** contains various detection heads that predict from RoIs.
  - **seg_heads** contains various segmentation heads
  - **losses** contains various loss functions

  - **task_modules** provides modules for detection tasks. E.g. assigners, samplers, box coders, and prior generators.
  - **layers** provides some basic neural network layers

- **engine** is a part for runtime components.
  - **runner** provides extensions for MMEngine's runner.
  - **schedulers** provides schedulers for adjusting optimization hyperparameters.
  - **optimizers** provides optimizers and optimizer wrappers.
  - **hooks** provides various hooks of the runner.
- **evaluation** provides different metrics for evaluating model performance.
- **visualization** is for visualizing detection results.

1.2 How to Use this Guide

Here is a detailed step-by-step guide to learn more about MMDetection:

1. For installation instructions, please see *get_started*.
2. Refer to the below tutorials for the basic usage of MMDetection.
   - Train and Test
   - Useful Tools
3. Refer to the below tutorials to dive deeper:
   - Basic Concepts
   - Component Customization

CHAPTER

TWO

GET STARTED

2.1 Prerequisites

In this section we demonstrate how to prepare an environment with PyTorch.

MMDetection works on Linux, Windows and macOS. It requires Python 3.6+, CUDA 9.2+ and PyTorch 1.6+.

Note: If you are experienced with PyTorch and have already installed it, just skip this part and jump to the *next section*. Otherwise, you can follow these steps for the preparation.

Step 0. Download and install Miniconda from the official website.

Step 1. Create a conda environment and activate it.

```
conda create --name openmmlab python=3.8 -y
conda activate openmmlab
```

Step 2. Install PyTorch following official instructions, e.g.

On GPU platforms:

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

On CPU platforms:

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

2.2 Installation

We recommend that users follow our best practices to install MMDetection. However, the whole process is highly customizable. See *Customize Installation* section for more information.

2.2.1 Best Practices

Step 0. Install MMEngine and MMCV using MIM.

```
pip install -U openmim
mim install mmengine
mim install "mmcv>=2.0.0rc1"
```

Note: In MMCV-v2.x, `mmcv-full` is rename to `mmcv`, if you want to install `mmcv` without CUDA ops, you can use `mim install "mmcv-lite>=2.0.0rc1"` to install the lite version.

Step 1. Install MMDetection.

Case a: If you develop and run mmdet directly, install it from source:

```
git clone https://github.com/open-mmlab/mmdetection.git -b 3.x
# "-b 3.x" means checkout to the `3.x` branch.
cd mmdetection
pip install -v -e .
# "-v" means verbose, or more output
# "-e" means installing a project in editable mode,
# thus any local modifications made to the code will take effect without reinstallation.
```

Case b: If you use mmdet as a dependency or third-party package, install it with MIM:

```
mim install "mmdet>=3.0.0rc0"
```

2.3 Verify the installation

To verify whether MMDetection is installed correctly, we provide some sample codes to run an inference demo.

Step 1. We need to download config and checkpoint files.

```
mim download mmdet --config yolov3_mobilenetv2_8xb24-320-300e_coco --dest .
```

The downloading will take several seconds or more, depending on your network environment. When it is done, you will find two files `yolov3_mobilenetv2_8xb24-320-300e_coco.py` and `yolov3_mobilenetv2_320_300e_coco_20210719_215349-d18dff72.pth` in your current folder.

Step 2. Verify the inference demo.

Option (a). If you install MMDetection from source, just run the following command.

```
python demo/image_demo.py demo/demo.jpg yolov3_mobilenetv2_8xb24-ms-416-300e_coco.py␣
↪yolov3_mobilenetv2_320_300e_coco_20210719_215349-d18dff72.pth --device cpu --out-file␣
↪result.jpg
```

You will see a new image `result.jpg` on your current folder, where bounding boxes are plotted on cars, benches, etc.

Option (b). If you install MMDetection with MIM, open you python interpreter and copy&paste the following codes.

```
from mmdet.apis import init_detector, inference_detector
from mmdet.utils import register_all_modules

register_all_modules()
```

(continues on next page)

(continued from previous page)

```
config_file = 'yolov3_mobilenetv2_8xb24-ms-416-300e_coco.py'
checkpoint_file = 'yolov3_mobilenetv2_320_300e_coco_20210719_215349-d18dff72.pth'
model = init_detector(config_file, checkpoint_file, device='cpu')  # or device='cuda:0'
inference_detector(model, 'demo/demo.jpg')
```

You will see a list of `DetDataSample`, and the predictions are in the `pred_instance`, indicating the detected bounding boxes, labels, and scores.

2.3.1 Customize Installation

CUDA versions

When installing PyTorch, you need to specify the version of CUDA. If you are not clear on which to choose, follow our recommendations:

- For Ampere-based NVIDIA GPUs, such as GeForce 30 series and NVIDIA A100, CUDA 11 is a must.
- For older NVIDIA GPUs, CUDA 11 is backward compatible, but CUDA 10.2 offers better compatibility and is more lightweight.

Please make sure the GPU driver satisfies the minimum version requirements. See this table for more information.

Note: Installing CUDA runtime libraries is enough if you follow our best practices, because no CUDA code will be compiled locally. However if you hope to compile MMCV from source or develop other CUDA operators, you need to install the complete CUDA toolkit from NVIDIA's website, and its version should match the CUDA version of PyTorch. i.e., the specified version of cudatoolkit in `conda install` command.

Install MMEngine without MIM

To install MMEngine with pip instead of MIM, please follow [MMEngine installation guides](https://mmengine.readthedocs.io/en/latest/get_started/installation.html).

For example, you can install MMEngine by the following command.

```
pip install mmengine
```

Install MMCV without MIM

MMCV contains C++ and CUDA extensions, thus depending on PyTorch in a complex way. MIM solves such dependencies automatically and makes the installation easier. However, it is not a must.

To install MMCV with pip instead of MIM, please follow MMCV installation guides. This requires manually specifying a find-url based on PyTorch version and its CUDA version.

For example, the following command install mmcv built for PyTorch 1.12.x and CUDA 11.6.

```
pip install "mmcv>=2.0.0rc1" -f https://download.openmmlab.com/mmcv/dist/cu116/torch1.12.
↪0/index.html
```

Install on CPU-only platforms

MMDetection can be built for CPU only environment. In CPU mode you can train (requires MMCV version >= 2.0.0rc1), test or inference a model.

However some functionalities are gone in this mode:

- Deformable Convolution
- Modulated Deformable Convolution
- ROI pooling
- Deformable ROI pooling
- CARAFE
- SyncBatchNorm
- CrissCrossAttention
- MaskedConv2d
- Temporal Interlace Shift
- nms_cuda
- sigmoid_focal_loss_cuda
- bbox_overlaps

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

Install on Google Colab

Google Colab usually has PyTorch installed, thus we only need to install MMEngine, MMCV, and MMDetection with the following commands.

Step 1. Install MMEngine and MMCV using MIM.

```
!pip3 install openmim
!mim install mmengine
!mim install mmcv>=2.0.0rc1,<2.1.0
```

Step 2. Install MMDetection from the source.

```
!git clone https://github.com/open-mmlab/mmdetection.git -b 3.x
%cd mmdetection
!pip install -e .
```

Step 3. Verification.

```
import mmdet
print(mmdet.__version__)
# Example output: 3.0.0rc0, or other version.
```

Note: Within Jupyter, the exclamation mark ! is used to call external executables and %cd is a magic command to change the current working directory of Python.

Using MMDetection with Docker

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

```
# build an image with PyTorch 1.6, CUDA 10.1
# If you prefer other versions, just modified the Dockerfile
docker build -t mmdetection docker/
```

Run it with

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

2.3.2 Trouble shooting

If you have some issues during the installation, please first view the *FAQ* page. You may open an issue on GitHub if no solution is found.

CHAPTER

THREE

TRAIN & TEST

MMDetection provides hundreds of pretrained 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:

3.1 Learn about Configs

MMDetection and other OpenMMLab repositories use MMEngine's config system. It has a modular and inheritance design, which is convenient to conduct various experiments.

3.1.1 Config file content

MMDetection uses a modular design, all modules with different functions can be configured through the config. Taking Mask R-CNN as an example, we will introduce each field in the config according to different function modules:

Model config

In mmdetection's config, we use `model` to setup detection algorithm components. In addition to neural network components such as `backbone`, `neck` etc, it also requires `data_preprocessor`, `train_cfg`, and `test_cfg`. `data_preprocessor` is responsible for processing a batch of data output by dataloader. `train_cfg`, and `test_cfg` in the model config are for training and testing hyperparameters of the components.

```
model = dict(
    type='MaskRCNN',  # The name of detector
    data_preprocessor=dict(  # The config of data preprocessor, usually includes image␣
↪normalization and padding
        type='DetDataPreprocessor',  # The type of the data preprocessor, refer to␣
↪https://mmdetection.readthedocs.io/en/dev-3.x/api.html#module-mmdet.models.data_
↪preprocessors
        mean=[123.675, 116.28, 103.53],  # Mean values used to pre-training the pre-
↪trained backbone models, ordered in R, G, B
        std=[58.395, 57.12, 57.375],  # Standard variance used to pre-training the pre-
↪trained backbone models, ordered in R, G, B
        bgr_to_rgb=True,  # whether to convert image from BGR to RGB
        pad_mask=True,  # whether to pad instance masks
        pad_size_divisor=32),  # The size of padded image should be divisible by ``pad_
↪size_divisor``
    backbone=dict(  # The config of backbone
```

(continues on next page)

(continued from previous page)

```
        type='ResNet',
        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 stage are frozen
        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/dev-3.x/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/dev-3.x/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/dev-3.x/mmdet/models/task_modules/prior_generators/anchor_generator.py for more␣
↪details
            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/
↪dev-3.x/mmdet/models/task_modules/coders/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.
```

(continues on next page)

(continued from previous page)

```
            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/dev-3.x/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',
        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/dev-3.x/mmdet/models/roi_heads/roi_extractors/single_level_roi_extractor.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://mmcv.readthedocs.io/
↪en/latest/api.html#mmcv.ops.RoIAlign 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/dev-3.x/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.
            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.
```

(continues on next page)

(continued from previous page)

```
        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/dev-3.x/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/dev-3.x/
↪mmdet/models/task_modules/assigners/max_iou_assigner.py 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
                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/
↪dev-3.x/mmdet/models/task_modules/samplers/random_sampler.py 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
```

(continues on next page)

(continued from previous page)

```
        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/dev-3.x/
↪mmdet/models/task_modules/assigners/max_iou_assigner.py 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/
↪dev-3.x/mmdet/models/task_modules/samplers/random_sampler.py 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.
                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
                ),
```

(continues on next page)

(continued from previous page)

```
            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 and evaluator config

Dataloaders are required for the training, validation, and testing of the runner. Dataset and data pipeline need to be set to build the dataloader. Due to the complexity of this part, we use intermediate variables to simplify the writing of dataloader configs.

```
dataset_type = 'CocoDataset'  # Dataset type, this will be used to define the dataset
data_root = 'data/coco/'  # Root path of data
file_client_args = dict(backend='disk')  # file client arguments

train_pipeline = [  # Training data processing pipeline
    dict(type='LoadImageFromFile', file_client_args=file_client_args),  # 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=True),  # Whether to convert the polygon mask to instance mask, set␣
↪False for acceleration and to save memory
    dict(
        type='Resize',  # Pipeline that resizes the images and their annotations
        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 flips the images and their␣
↪annotations
        prob=0.5),  # The probability to flip
    dict(type='PackDetInputs')  # Pipeline that formats the annotation data and decides␣
↪which keys in the data should be packed into data_samples
]
test_pipeline = [  # Testing data processing pipeline
    dict(type='LoadImageFromFile', file_client_args=file_client_args),  # First pipeline␣
↪to load images from file path
    dict(type='Resize', scale=(1333, 800), keep_ratio=True),  # Pipeline that resizes␣
↪the images
    dict(
        type='PackDetInputs',  # Pipeline that formats the annotation data and decides␣
↪which keys in the data should be packed into data_samples
        meta_keys=('img_id', 'img_path', 'ori_shape', 'img_shape',
                   'scale_factor'))
]
```

(continues on next page)

(continued from previous page)

```
train_dataloader = dict(   # Train dataloader config
    batch_size=2,  # Batch size of a single GPU
    num_workers=2,  # Worker to pre-fetch data for each single GPU
    persistent_workers=True,  # If ``True``, the dataloader will not shut down the worker␣
↪processes after an epoch end, which can accelerate training speed.
    sampler=dict(  # training data sampler
        type='DefaultSampler',  # DefaultSampler which supports both distributed and non-
↪distributed training. Refer to https://github.com/open-mmlab/mmengine/blob/main/
↪mmengine/dataset/sampler.py
        shuffle=True),  # randomly shuffle the training data in each epoch
    batch_sampler=dict(type='AspectRatioBatchSampler'),  # Batch sampler for grouping␣
↪images with similar aspect ratio into a same batch. It can reduce GPU memory cost.
    dataset=dict(  # Train dataset config
        type=dataset_type,
        data_root=data_root,
        ann_file='annotations/instances_train2017.json',  # Path of annotation file
        data_prefix=dict(img='train2017/'),  # Prefix of image path
        filter_cfg=dict(filter_empty_gt=True, min_size=32),  # Config of filtering␣
↪images and annotations
        pipeline=train_pipeline))
val_dataloader = dict(  # Validation dataloader config
    batch_size=1,  # Batch size of a single GPU. If batch-szie > 1, the extra padding␣
↪area may influence the performance.
    num_workers=2,  # Worker to pre-fetch data for each single GPU
    persistent_workers=True,  # If ``True``, the dataloader will not shut down the worker␣
↪processes after an epoch end, which can accelerate training speed.
    drop_last=False,  # Whether to drop the last incomplete batch, if the dataset size␣
↪is not divisible by the batch size
    sampler=dict(
        type='DefaultSampler',
        shuffle=False),  # not shuffle during validation and testing
    dataset=dict(
        type=dataset_type,
        data_root=data_root,
        ann_file='annotations/instances_val2017.json',
        data_prefix=dict(img='val2017/'),
        test_mode=True,  # Turn on test mode of the dataset to avoid filtering␣
↪annotations or images
        pipeline=test_pipeline))
test_dataloader = val_dataloader  # Testing dataloader config
```

Evaluators are used to compute the metrics of the trained model on the validation and testing datasets. The config of evaluators consists of one or a list of metric configs:

```
val_evaluator = dict(  # Validation evaluator config
    type='CocoMetric',  # The coco metric used to evaluate AR, AP, and mAP for detection␣
↪and instance segmentation
    ann_file=data_root + 'annotations/instances_val2017.json',  # Annotation file path
    metric=['bbox', 'segm'],  # Metrics to be evaluated, `bbox` for detection and `segm`␣
↪for instance segmentation
    format_only=False)
test_evaluator = val_evaluator  # Testing evaluator config
```

Since the test dataset has no annotation files, the test_dataloader and test_evaluator config in MMDetection are generally equal to the val's. If you want to save the detection results on the test dataset, you can write the config like this:

```
# inference on test dataset and
# format the output results for submission.
test_dataloader = dict(
    batch_size=1,
    num_workers=2,
    persistent_workers=True,
    drop_last=False,
    sampler=dict(type='DefaultSampler', shuffle=False),
    dataset=dict(
        type=dataset_type,
        data_root=data_root,
        ann_file=data_root + 'annotations/image_info_test-dev2017.json',
        data_prefix=dict(img='test2017/'),
        test_mode=True,
        pipeline=test_pipeline))
test_evaluator = dict(
    type='CocoMetric',
    ann_file=data_root + 'annotations/image_info_test-dev2017.json',
    metric=['bbox', 'segm'],  # Metrics to be evaluated
    format_only=True,  # Only format and save the results to coco json file
    outfile_prefix='./work_dirs/coco_detection/test')  # The prefix of output json files
```

Training and testing config

MMEngine's runner uses Loop to control the training, validation, and testing processes. Users can set the maximum training epochs and validation intervals with these fields.

```
train_cfg = dict(
    type='EpochBasedTrainLoop',  # The training loop type. Refer to https://github.com/
↪open-mmlab/mmengine/blob/main/mmengine/runner/loops.py
    max_epochs=12,  # Maximum training epochs
    val_interval=1)  # Validation intervals. Run validation every epoch.
val_cfg = dict(type='ValLoop')  # The validation loop type
test_cfg = dict(type='TestLoop')  # The testing loop type
```

Optimization config

`optim_wrapper` is the field to configure optimization related settings. The optimizer wrapper not only provides the functions of the optimizer, but also supports functions such as gradient clipping, mixed precision training, etc. Find more in optimizer wrapper tutorial.

```
optim_wrapper = dict(  # Optimizer wrapper config
    type='OptimWrapper',  # Optimizer wrapper type, switch to AmpOptimWrapper to enable␣
↪mixed precision training.
    optimizer=dict(  # Optimizer config. Support all kinds of optimizers in PyTorch.␣
↪Refer to https://pytorch.org/docs/stable/optim.html#algorithms
        type='SGD',  # Stochastic gradient descent optimizer
        lr=0.02,  # The base learning rate
```

(continues on next page)

(continued from previous page)

```
        momentum=0.9,  # Stochastic gradient descent with momentum
        weight_decay=0.0001),  # Weight decay of SGD
    clip_grad=None,  # Gradient clip option. Set None to disable gradient clip. Find␣
↪usage in https://mmengine.readthedocs.io/en/latest/tutorials/optimizer.html
    )
```

`param_scheduler` is a field that configures methods of adjusting optimization hyperparameters such as learning rate and momentum. Users can combine multiple schedulers to create a desired parameter adjustment strategy. Find more in parameter scheduler tutorial and parameter scheduler API documents

```
param_scheduler = [
    dict(
        type='LinearLR',  # Use linear policy to warmup learning rate
        start_factor=0.001, # The ratio of the starting learning rate used for warmup
        by_epoch=False,  # The warmup learning rate is updated by iteration
        begin=0,  # Start from the first iteration
        end=500),  # End the warmup at the 500th iteration
    dict(
        type='MultiStepLR',  # Use multi step learning rate policy during training
        by_epoch=True,  # The learning rate is updated by epoch
        begin=0,   # Start from the first epoch
        end=12,  # End at the 12th epoch
        milestones=[8, 11],  # Epochs to decay the learning rate
        gamma=0.1)  # The learning rate decay ratio
]
```

Hook config

Users can attach hooks to training, validation, and testing loops to insert some oprations during running. There are two different hook fields, one is `default_hooks` and the other is `custom_hooks`.

`default_hooks` is a dict of hook configs. `default_hooks` are the hooks must required at runtime. They have default priority which should not be modified. If not set, runner will use the default values. To disable a default hook, users can set its config to `None`.

```
default_hooks = dict(
    timer=dict(type='IterTimerHook'),
    logger=dict(type='LoggerHook', interval=50),
    param_scheduler=dict(type='ParamSchedulerHook'),
    checkpoint=dict(type='CheckpointHook', interval=1),
    sampler_seed=dict(type='DistSamplerSeedHook'),
    visualization=dict(type='DetVisualizationHook'))
```

`custom_hooks` is a list of hook configs. Users can develop their own hooks and insert them in this field.

```
custom_hooks = []
```

Runtime config

```
default_scope = 'mmdet'  # The default registry scope to find modules. Refer to https://
↪mmengine.readthedocs.io/en/latest/tutorials/registry.html

env_cfg = dict(
    cudnn_benchmark=False,  # Whether to enable cudnn benchmark
    mp_cfg=dict(  # Multi-processing config
        mp_start_method='fork',  # Use fork to start multi-processing threads. 'fork'␣
↪usually faster than 'spawn' but maybe unsafe. See discussion in https://github.com/
↪pytorch/pytorch/issues/1355
        opencv_num_threads=0),  # Disable opencv multi-threads to avoid system being␣
↪overloaded
    dist_cfg=dict(backend='nccl'),  # Distribution configs
)

vis_backends = [dict(type='LocalVisBackend')]  # Visualization backends. Refer to TODO:␣
↪visualization documents
visualizer = dict(
    type='DetLocalVisualizer', vis_backends=vis_backends, name='visualizer')
log_processor = dict(
    type='LogProcessor',  # Log processor to process runtime logs
    window_size=50,  # Smooth interval of log values
    by_epoch=True)  # Whether to format logs with epoch stype. Should be consistent with␣
↪train loop's type.

log_level = 'INFO'  # The level of logging.
load_from = None  # Load model checkpoint as a pre-trained model from a given path. This␣
↪will not resume training.
resume = False  # Whether to resume from the checkpoint defined in `load_from`. If `load_
↪from` is None, it will resume the latest checkpoint in the `work_dir`.
```

3.1.2 Iter-based config

MMEngine's Runner also provides an iter-based training loop except for epoch-based. To use iter-based training, users should modify the `train_cfg`, `param_scheduler`, `train_dataloader`, `default_hooks`, and `log_processor`. Here is an example of changing an epoch-based RetinaNet config to iter-based: `configs/retinanet/retinanet_r50_fpn_90k_coco.py`

```
# Iter-based training config
train_cfg = dict(
    _delete_=True,  # Ignore the base config setting (optional)
    type='IterBasedTrainLoop',  # Use iter-based training loop
    max_iters=90000,  # Maximum iterations
    val_interval=10000)  # Validation interval


# Change the scheduler to iter-based
param_scheduler = [
    dict(
        type='LinearLR', start_factor=0.001, by_epoch=False, begin=0, end=500),
    dict(
```

(continues on next page)

(continued from previous page)

```
        type='MultiStepLR',
        begin=0,
        end=90000,
        by_epoch=False,
        milestones=[60000, 80000],
        gamma=0.1)
]

# Switch to InfiniteSampler to avoid dataloader restart
train_dataloader = dict(sampler=dict(type='InfiniteSampler'))

# Change the checkpoint saving interval to iter-based
default_hooks = dict(checkpoint=dict(by_epoch=False, interval=10000))

# Change the log format to iter-based
log_processor = dict(by_epoch=False)
```

3.1.3 Config file inheritance

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 mmengine config tutorial for detailed documentation.

By setting the `_base_` field, we can set which files the current configuration file inherits from.

When `_base_` is a string of a file path, it means inherit the contents of one config file.

```
_base_ = './mask-rcnn_r50_fpn_1x_coco.py'
```

When `_base_` is a list of multiple file paths, it means inheriting multiple files.

```
_base_ = [
    '../_base_/models/mask-rcnn_r50_fpn.py',
    '../_base_/datasets/coco_instance.py',
    '../_base_/schedules/schedule_1x.py', '../_base_/default_runtime.py'
]
```

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.

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 mmengine config tutorial for simple illustration.

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

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

(continues on next page)

(continued from previous page)

```
                num_channels=(32, 64, 128, 256))),
        init_cfg=dict(type='Pretrained', checkpoint='open-mmlab://msra/hrnetv2_w32')),
    neck=dict(...))
```

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

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'

train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='LoadAnnotations', with_bbox=True, with_mask=True),
    dict(
        type='RandomResize', scale=[(1333, 640), (1333, 800)],
        keep_ratio=True),
    dict(type='RandomFlip', prob=0.5),
    dict(type='PackDetInputs')
]
test_pipeline = [
    dict(type='LoadImageFromFile', file_client_args=file_client_args),
    dict(type='Resize', scale=(1333, 800), keep_ratio=True),
    dict(
        type='PackDetInputs',
        meta_keys=('img_id', 'img_path', 'ori_shape', 'img_shape',
                   'scale_factor'))
]
train_dataloader = dict(dataset=dict(pipeline=train_pipeline))
val_dataloader = dict(dataset=dict(pipeline=test_pipeline))
test_dataloader = dict(dataset=dict(pipeline=test_pipeline))
```

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

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),
    ...)
```

Reuse variables in _base_ file

If the users want to reuse the variables in the base file, they can get a copy of the corresponding variable by using {{_base_.xxx}}. E.g:

```
_base_ = './mask-rcnn_r50_fpn_1x_coco.py'

a = {{_base_.model}} # Variable `a` is equal to the `model` defined in `_base_`
```

3.1.4 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 train_dataloader.dataset.pipeline is normally a list e.g. [dict(type='LoadImageFromFile'), ...]. If you want to change 'LoadImageFromFile' to 'LoadImageFromNDArray' in the pipeline, you may specify --cfg-options data.train.pipeline.0.type=LoadImageFromNDArray.

- Update values of list/tuples.

  If the value to be updated is a list or a tuple. For example, the config file normally sets model.data_preprocessor.mean=[123.675, 116.28, 103.53]. If you want to change the mean values, you may specify --cfg-options model.data_preprocessor.mean="[127,127,127]". 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.

3.1.5 Config name style

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

```
{algorithm name}_{model component names [component1]_[component2]_[...]}_{training␣
↪settings}_{training dataset information}_{testing dataset information}.py
```

The file name is divided to five parts. All parts and components are connected with _ and words of each part or component should be connected with -.

- {algorithm name}: The name of the algorithm. It can be a detector name such as faster-rcnn, mask-rcnn, etc. Or can be a semi-supervise or knowladge-distillation algorithm such as soft-teacher, lad. etc.
- {model component names}: Names of the components used in the algorithm such as backbone, neck, etc. For example, r50-caffe_fpn_gn-head means using caffe-style ResNet50, FPN and detection head with Group Norm in the algorithm.
- {training settings}: Information of training settings such as batch size, augmentations, loss trick, scheduler, and epochs/iterations. For example: 4xb4-mixup-giou-coslr-100e means using 8-gpus x 4-images-per-gpu, mixup augmentation, GIoU loss, cosine annealing learning rate, and train 100 epochs. Some abbreviations:

  - `{gpu x batch_per_gpu}`: GPUs and samples per GPU. `bN` indicates N batch size per GPU. E.g. `4xb4` is the short term of 4-gpus x 4-images-per-gpu. And `8xb2` is used by default if not mentioned.
  - `{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.

- `{training dataset information}`: Training dataset names like `coco`, `coco-panoptic`, `cityscapes`, `voc-0712`, `wider-face`.
- `{testing dataset information}` (optional): Testing dataset name for models trained on one dataset but tested on another. If not mentioned, it means the model was trained and tested on the same dataset type.

3.2 Inference with existing models

MMDetection provides hundreds of pretrained detection models in Model Zoo. This note will show how to inference, which means 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.

3.2.1 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.

```
import cv2
import mmcv
from mmcv.transforms import Compose
from mmengine.utils import track_iter_progress
from mmdet.registry import VISUALIZERS
from mmdet.utils import register_all_modules
from mmdet.apis import init_detector, inference_detector

# Register all modules in mmdet into the registries
register_all_modules()

# 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')

# Init visualizer
visualizer = VISUALIZERS.build(model.cfg.visualizer)
# The dataset_meta is loaded from the checkpoint and
# then pass to the model in init_detector
visualizer.dataset_meta = model.dataset_meta
# Ttest a single image and show the results
```

(continues on next page)

(continued from previous page)

```
img = 'test.jpg'  # or img = mmcv.imread(img), which will only load it once
result = inference_detector(model, img)

# Show the results
img = mmcv.imread(img)
img = mmcv.imconvert(img, 'bgr', 'rgb')


visualizer.add_datasample(
    'result',
    img,
    data_sample=result,
    draw_gt=False,
    show=True)

# Test a video and show the results
# Build test pipeline
model.cfg.test_dataloader.dataset.pipeline[0].type = 'LoadImageFromNDArray'
test_pipeline = Compose(model.cfg.test_dataloader.dataset.pipeline)

# Init visualizer
visualizer = VISUALIZERS.build(model.cfg.visualizer)
# The dataset_meta is loaded from the checkpoint and
# then pass to the model in init_detector
visualizer.dataset_meta = model.dataset_meta

# The interval of show (s), 0 is block
wait_time = 1

video_reader = mmcv.VideoReader('video.mp4')

for frame in track_iter_progress(video_reader):
    result = inference_detector(model, frame, test_pipeline=test_pipeline)
    visualizer.add_datasample(
        name='video',
        image=frame,
        data_sample=result,
        draw_gt=False,
        show=False)
    frame = visualizer.get_image()

    cv2.namedWindow('video', 0)
    mmcv.imshow(frame, 'video', wait_time)
```

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

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

3.2.2 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}] \
```

(continues on next page)

(continued from previous page)

```
    [--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
```

Video demo with GPU acceleration

This script performs inference on a video with GPU acceleration.

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

Examples:

```
python demo/video_gpuaccel_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 \
    --nvdecode --out result.mp4
```

3.3 Dataset Prepare

MMDetection supports multiple public datasets including COCO, Pascal VOC, CityScapes, and more.

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.

We provide a script to download datasets such as COCO , you can run `python tools/misc/download_dataset.py --dataset-name coco2017` to download COCO dataset.

For more usage please refer to dataset-download

```
mmdetection
├── mmdet
├── tools
├── configs
```

(continues on next page)

(continued from previous page)

```
├── data
│   ├── coco
│   │   ├── annotations
│   │   ├── train2017
│   │   ├── val2017
│   │   ├── test2017
│   ├── cityscapes
│   │   ├── annotations
│   │   ├── leftImg8bit
│   │   │   ├── train
│   │   │   ├── val
│   │   ├── gtFine
│   │   │   ├── 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 \
```

(continues on next page)

(continued from previous page)

```
    --nproc 8 \
    --out-dir ./data/cityscapes/annotations
```

3.4 Test existing models on standard datasets

To evaluate a model's accuracy, one usually tests the model on some standard datasets, please refer to *dataset prepare guide* to prepare the dataset.

This section will show how to test existing models on supported datasets.

3.4.1 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
- CPU
- 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}] \
    [--show]

# CPU: disable GPUs and run single-gpu testing script
export CUDA_VISIBLE_DEVICES=-1
python tools/test.py \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    [--out ${RESULT_FILE}] \
    [--show]

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

`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.

- `--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.
- `--work-dir`: If specified, detection results containing evaluation metrics will be saved to the specified directory.
- `--cfg-options`: If specified, the key-value pair optional cfg will be merged into config file

3.4.2 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). 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
```

4. Test Mask R-CNN with 8 GPUs, and evaluate. 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
```

5. Test Mask R-CNN with 8 GPUs, and evaluate the metric **class-wise**. 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 \
    --cfg-options test_evaluator.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.

   Replace the original test_evaluator and test_dataloader with test_evaluator and test_dataloader in the comment in config and run:

   ```
   ./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
   ```

   This command generates two JSON files `./work_dirs/coco_instance/test.bbox.json` and `./work_dirs/coco_instance/test.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.

   Replace the original test_evaluator and test_dataloader with test_evaluator and test_dataloader in the comment in config and run:

   ```
   ./tools/dist_test.sh \
       configs/cityscapes/mask-rcnn_r50_fpn_1x_cityscapes.py \
       checkpoints/mask_rcnn_r50_fpn_1x_cityscapes_20200227-afe51d5a.pth \
       8
   ```

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

3.4.3 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 need to replace the original test_evaluator and test_dataloader with test_evaluator and test_dataloader in the comment in config(find which dataset in 'configs/*base*/datasets' the current config corresponds to) and run:

```
# Single-gpu testing
python tools/test.py \
    ${CONFIG_FILE} \
```

(continues on next page)

(continued from previous page)

```
    ${CHECKPOINT_FILE} \
    [--show]

# CPU: disable GPUs and run single-gpu testing script
export CUDA_VISIBLE_DEVICES=-1
python tools/test.py \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    [--out ${RESULT_FILE}] \
    [--show]

# Multi-gpu testing
bash tools/dist_test.sh \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    ${GPU_NUM} \
    [--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
```

This command generates two JSON files `./work_dirs/coco_instance/test.bbox.json` and `./work_dirs/coco_instance/test.segm.json`.

3.4.4 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_dataloader=dict(...), val_dataloader=dict(...), test_
↪dataloader=dict(batch_size=2, ...))
```

Or you can set it through `--cfg-options` as `--cfg-options test_dataloader.batch_size=2`

3.5 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.

3.5.1 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.

3.5.2 Learning rate automatically scale

Important: The default learning rate in config files is for 8 GPUs and 2 sample per gpu (batch size = 8 * 2 = 16). And it had been set to `auto_scale_lr.base_batch_size` in `config/_base_/default_runtime.py`. Learning rate will be automatically scaled base on this value when the batch size is `16`. Meanwhile, in order not to affect other codebase which based on mmdet, the flag `auto_scale_lr.enable` is set to `False` by default.

If you want to enable this feature, you need to add argument `--auto-scale-lr`. And you need to check the config name which you want to use before you process the command, because the config name indicates the default batch size. By default, it is `8 x 2 = 16 batch size`, like `faster_rcnn_r50_caffe_fpn_90k_coco.py` or `pisa_faster_rcnn_x101_32x4d_fpn_1x_coco.py`. In other cases, you will see the config file name have `_NxM_` in dictating, like `cornernet_hourglass104_mstest_32x3_210e_coco.py` which batch size is `32 x 3 = 96`, or `scnet_x101_64x4d_fpn_8x1_20e_coco.py` which batch size is `8 x 1 = 8`.

Please remember to check the bottom of the specific config file you want to use, it will have `auto_scale_lr.base_batch_size` if the batch size is not `16`. If you can't find those values, check the config file which in `_base_=[xxx]` and you will find it. Please do not modify its values if you want to automatically scale the LR.

Learning rate automatically scale basic usage is as follows.

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

If you enabled this feature, the learning rate will be automatically scaled according to the number of GPUs of the machine and the batch size of training. See linear scaling rule for details. For example, If there are 4 GPUs and 2 pictures on each GPU, `lr = 0.01`, then if there are 16 GPUs and 4 pictures on each GPU, it will automatically scale to `lr = 0.08`.

If you don't want to use it, you need to calculate the learning rate according to the linear scaling rule manually then change `optimizer.lr` in specific config file.

3.5.3 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.
train_cfg = dict(val_interval=12)
```

This tool accepts several optional arguments, including:

- `--work-dir ${WORK_DIR}`: Override the working directory.
- `--auto-resume`: resume from the latest checkpoint in the work_dir automatically.
- `--cfg-options 'Key=value'`: Overrides other settings in the used config.

3.5.4 Training on CPU

The process of training on the CPU is consistent with single GPU training. We just need to disable GPUs before the training process.

```
export CUDA_VISIBLE_DEVICES=-1
```

And then run the script above.

Note:

We do not recommend users to use CPU for training because it is too slow. We support this feature to allow users to debug on machines without GPU for convenience.

3.5.5 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
```

3.5.6 Train with multiple machines

If you launch with multiple machines simply connected with ethernet, you can simply run following commands:

On the first machine:

```
NNODES=2 NODE_RANK=0 PORT=$MASTER_PORT MASTER_ADDR=$MASTER_ADDR sh tools/dist_train.sh
↪$CONFIG $GPUS
```

On the second machine:

```
NNODES=2 NODE_RANK=1 PORT=$MASTER_PORT MASTER_ADDR=$MASTER_ADDR sh tools/dist_train.sh
↪$CONFIG $GPUS
```

Usually it is slow if you do not have high speed networking like InfiniBand.

3.5.7 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}
   ```

(continues on next page)

(continued from previous page)

3.6 Train with customized datasets

In this part, you will know how to train predefined models with customized datasets and then test it. 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.

3.6.1 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 using 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: Datasets and metrics have been decoupled except CityScapes since MMDetection 3.0 . Therefore, uers can use any kind of evaluation metrics for any format of datasets during validation. For example: evaluate on COCO dataset with VOC metric, or evaluate on OpenImages dataset with both VOC and COCO metrics.

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,
    "file_name": str,
}

annotation = {
    "id": int,
    "image_id": int,
    "category_id": int,
```

(continues on next page)

(continued from previous page)

```
    "segmentation": RLE or [polygon],
    "area": float,
    "bbox": [x,y,width,height], # (x, y) are the coordinates of the upper left corner of␣
↪the bbox
    "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,
     1198,
     1207,
     1210,
     1190,
     1177,
     1172,
```

(continues on next page)

(continued from previous page)

```
    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'}}},
 '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
```

(continues on next page)

(continued from previous page)

```
import mmcv

from mmengine.fileio import dump, load
from mmengine.utils import track_iter_progress


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

    annotations = []
    images = []
    obj_count = 0
    for idx, v in enumerate(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))

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

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    convert_balloon_to_coco(ann_file='data/balloon/train/via_region_data.json',
                            out_file='data/balloon/train/annotation_coco.json',
                            image_prefix='data/balloon/train')
    convert_balloon_to_coco(ann_file='data/balloon/val/via_region_data.json',
                            out_file='data/balloon/val/annotation_coco.json',
                            image_prefix='data/balloon/val')
```

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 with `CocoMetric`.

3.6.2 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_ms-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_ms-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
data_root = 'data/balloon/'
metainfo = {
    'CLASSES': ('balloon', ),
    'PALETTE': [
        (220, 20, 60),
    ]
}
train_dataloader = dict(
    batch_size=1,
    dataset=dict(
        data_root=data_root,
        metainfo=metainfo,
        ann_file='train/annotation_coco.json',
        data_prefix=dict(img='train/')))
val_dataloader = dict(
    dataset=dict(
        data_root=data_root,
        metainfo=metainfo,
        ann_file='val/annotation_coco.json',
        data_prefix=dict(img='val/')))
test_dataloader = val_dataloader

# Modify metric related settings
val_evaluator = dict(ann_file=data_root + 'val/annotation_coco.json')
```

(continues on next page)

(continued from previous page)

```
test_evaluator = val_evaluator

# We can use the pre-trained Mask RCNN model to obtain higher performance
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'
```

3.6.3 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_ms-poly-1x_balloon.py
```

For more detailed usages, please refer to the *training guide*.

3.6.4 Test and inference

To test the trained model, you can simply run

```
python tools/test.py configs/balloon/mask-rcnn_r50-caffe_fpn_ms-poly-1x_balloon.py work_
↪dirs/mask-rcnn_r50-caffe_fpn_ms-poly-1x_balloon/epoch_12.pth
```

For more detailed usages, please refer to the *testing guide*.

3.7 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 `TranslateX` 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.

3.7.1 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
│   │   ├── VOC2007
│   │   ├── VOC2012
```

Or you can set your dataset root through

```
export MMDET_DATASETS=$data_root
```

We will replace dataset root with `$MMDET_DATASETS`, so you don't have to modify the corresponding path in config files.

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.

3.7.2 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.

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

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

```
import torch.nn as nn
from mmdet.registry import MODELS


@MODELS.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
```

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'],
    allow_failed_imports=False)
```

to the config file and avoid modifying the original code.

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 customizing 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.

3.7.3 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`.
    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,
```

(continues on next page)

(continued from previous page)

```
                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',
                    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))))
```

(continues on next page)

(continued from previous page)

```
# over-write `train_pipeline` for new added `AutoAugment` training setting
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='TranslateX',
                 level=5,
                 prob=0.5,
                 img_fill_val=(124, 116, 104))
            ],
        ]),
    dict(
        type='RandomResize',
        scale=[(2048, 800), (2048, 1024)],
        keep_ratio=True),
    dict(type='RandomFlip', prob=0.5),
    dict(type='PackDetInputs'),
]

# set batch_size per gpu, and set new training pipeline
train_dataloader = dict(
    batch_size=1,
    num_workers=3,
    # over-write `pipeline` with new training pipeline setting
    dataset=dict(pipeline=train_pipeline))

# Set optimizer
optim_wrapper = dict(
    type='OptimWrapper',
    optimizer=dict(type='SGD', lr=0.01, momentum=0.9, weight_decay=0.0001))

# Set customized learning policy
param_scheduler = [
    dict(
        type='LinearLR', start_factor=0.001, by_epoch=False, begin=0, end=500),
    dict(
        type='MultiStepLR',
        begin=0,
        end=10,
        by_epoch=True,
        milestones=[8],
```

(continues on next page)

(continued from previous page)

```
        gamma=0.1)
]

# train, val, test loop config
train_cfg = dict(max_epochs=10, val_interval=1)

# 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'
```

3.7.4 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 *training guide*.

3.7.5 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/epoch_10.
↪pth
```

For more detailed usages, please refer to the *testing guide*.

3.8 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 *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.

3.8.1 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 logger settings, the new config needs to inherit `_base_/default_runtime.py`. For training schedules, the new config can to inherit `_base_/schedules/schedule_1x.py`. These 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',
    '../_base_/schedules/schedule_1x.py'
]
```

3.8.2 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(
    roi_head=dict(
        bbox_head=dict(
            type='Shared2FCBBoxHead',
            in_channels=256,
            fc_out_channels=1024,
            roi_feat_size=7,
            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))))
```

3.8.3 Modify dataset

The users may also need to prepare the dataset and write the configs about dataset, refer to *Customize Datasets* for more detail. MMDetection V3.0 already supports VOC, WIDERFACE, COCO, LIVS, OpenImages, DeepFashion and Cityscapes Dataset.

3.8.4 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
optim_wrapper = dict(optimizer=dict(lr=0.01))

# learning rate
param_scheduler = [
    dict(
        type='LinearLR', start_factor=0.001, by_epoch=False, begin=0, end=500),
    dict(
        type='MultiStepLR',
        begin=0,
        end=8,
        by_epoch=True,
        milestones=[7],
        gamma=0.1)
]

# max_epochs
train_cfg = dict(max_epochs=8)

# log config
default_hooks = dict(logger=dict(interval=100)),
```

3.8.5 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
```

3.9 Test Results Submission

3.9.1 Panoptic segmentation test results submission

The following sections introduce how to produce the prediction results of panoptic segmentation models on the COCO test-dev set and submit the predictions to COCO evaluation server.

Prerequisites

- Download COCO test dataset images, testing image info, and panoptic train/val annotations, then unzip them, put 'test2017' to `data/coco/`, put json files and annotation files to `data/coco/annotations/`.

```
# suppose data/coco/ does not exist
mkdir -pv data/coco/

# download test2017
wget -P data/coco/ http://images.cocodataset.org/zips/test2017.zip
wget -P data/coco/ http://images.cocodataset.org/annotations/image_info_test2017.zip
wget -P data/coco/ http://images.cocodataset.org/annotations/panoptic_annotations_
↪trainval2017.zip

# unzip them
unzip data/coco/test2017.zip -d data/coco/
unzip data/coco/image_info_test2017.zip -d data/coco/
unzip data/coco/panoptic_annotations_trainval2017.zip -d data/coco/

# remove zip files (optional)
rm -rf data/coco/test2017.zip data/coco/image_info_test2017.zip data/coco/panoptic_
↪annotations_trainval2017.zip
```

- Run the following code to update category information in testing image info. Since the attribute `isthing` is missing in category information of 'image_info_test-dev2017.json', we need to update it with the category information in 'panoptic_val2017.json'.

```
python tools/misc/gen_coco_panoptic_test_info.py data/coco/annotations
```

After completing the above preparations, your directory structure of `data` should be like this:

```
data
`-- coco
    |-- annotations
    |   |-- image_info_test-dev2017.json
    |   |-- image_info_test2017.json
    |   |-- panoptic_image_info_test-dev2017.json
    |   |-- panoptic_train2017.json
    |   |-- panoptic_train2017.zip
    |   |-- panoptic_val2017.json
    |   `-- panoptic_val2017.zip
    `-- test2017
```

Inference on coco test-dev

To do inference on coco test-dev, we should update the setting of `test_dataloder` and `test_evaluator` first. There two ways to do this: 1. update them in config file; 2. update them in command line.

Update them in config file

The relevant settings are provided at the end of `configs/_base_/datasets/coco_panoptic.py`, as below.

```
test_dataloader = dict(
    batch_size=1,
    num_workers=1,
    persistent_workers=True,
    drop_last=False,
    sampler=dict(type='DefaultSampler', shuffle=False),
    dataset=dict(
        type=dataset_type,
        data_root=data_root,
        ann_file='annotations/panoptic_image_info_test-dev2017.json',
        data_prefix=dict(img='test2017/'),
        test_mode=True,
        pipeline=test_pipeline))
test_evaluator = dict(
    type='CocoPanopticMetric',
    format_only=True,
    ann_file=data_root + 'annotations/panoptic_image_info_test-dev2017.json',
    outfile_prefix='./work_dirs/coco_panoptic/test')
```

Any of the following way can be used to update the setting for inference on coco test-dev set.

Case 1: Directly uncomment the setting in `configs/_base_/datasets/coco_panoptic.py`.

Case 2: Copy the following setting to the config file you used now.

```
test_dataloader = dict(
    dataset=dict(
        ann_file='annotations/panoptic_image_info_test-dev2017.json',
        data_prefix=dict(img='test2017/', _delete_=True)))
test_evaluator = dict(
    format_only=True,
    ann_file=data_root + 'annotations/panoptic_image_info_test-dev2017.json',
    outfile_prefix='./work_dirs/coco_panoptic/test')
```

Then infer on coco test-dev et by the following command.

```
python tools/test.py \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE}
```

Update them in command line

The command for update of the related settings and inference on coco test-dev are as below.

```
# test with single gpu
CUDA_VISIBLE_DEVICES=0 python tools/test.py \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    --cfg-options \
    test_dataloader.dataset.ann_file=annotations/panoptic_image_info_test-dev2017.json \
    test_dataloader.dataset.data_prefix.img=test2017 \
    test_dataloader.dataset.data_prefix._delete_=True \
    test_evaluator.format_only=True \
    test_evaluator.ann_file=data/coco/annotations/panoptic_image_info_test-dev2017.json \
    test_evaluator.outfile_prefix=${WORK_DIR}/results

# test with four gpus
CUDA_VISIBLE_DEVICES=0,1,3,4 bash tools/dist_test.sh \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    8 \  # eights gpus
    --cfg-options \
    test_dataloader.dataset.ann_file=annotations/panoptic_image_info_test-dev2017.json \
    test_dataloader.dataset.data_prefix.img=test2017 \
    test_dataloader.dataset.data_prefix._delete_=True \
    test_evaluator.format_only=True \
    test_evaluator.ann_file=data/coco/annotations/panoptic_image_info_test-dev2017.json \
    test_evaluator.outfile_prefix=${WORK_DIR}/results

# test with slurm
GPUS=8 tools/slurm_test.sh \
    ${Partition} \
    ${JOB_NAME} \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    --cfg-options \
    test_dataloader.dataset.ann_file=annotations/panoptic_image_info_test-dev2017.json \
    test_dataloader.dataset.data_prefix.img=test2017 \
    test_dataloader.dataset.data_prefix._delete_=True \
    test_evaluator.format_only=True \
    test_evaluator.ann_file=data/coco/annotations/panoptic_image_info_test-dev2017.json \
    test_evaluator.outfile_prefix=${WORK_DIR}/results
```

Example

Suppose we perform inference on `test2017` using pretrained MaskFormer with ResNet-50 backbone.

```
# test with single gpu
CUDA_VISIBLE_DEVICES=0 python tools/test.py \
    configs/maskformer/maskformer_r50_mstrain_16x1_75e_coco.py \
    checkpoints/maskformer_r50_mstrain_16x1_75e_coco_20220221_141956-bc2699cb.pth \
    --cfg-options \
    test_dataloader.dataset.ann_file=annotations/panoptic_image_info_test-dev2017.json \
    test_dataloader.dataset.data_prefix.img=test2017 \
```

(continues on next page)

(continued from previous page)

```
    test_dataloader.dataset.data_prefix._delete_=True \
    test_evaluator.format_only=True \
    test_evaluator.ann_file=data/coco/annotations/panoptic_image_info_test-dev2017.json \
    test_evaluator.outfile_prefix=work_dirs/maskformer/results
```

Rename files and zip results

After inference, the panoptic segmentation results (a json file and a directory where the masks are stored) will be in `WORK_DIR`. We should rename them according to the naming convention described on COCO's Website. Finally, we need to compress the json and the directory where the masks are stored into a zip file, and rename the zip file according to the naming convention. Note that the zip file should **directly** contains the above two files.

The commands to rename files and zip results:

```
# In WORK_DIR, we have panoptic segmentation results: 'panoptic' and 'results.panoptic.json
↪'.
cd ${WORK_DIR}

# replace '[algorithm_name]' with the name of algorithm you used.
mv ./panoptic ./panoptic_test-dev2017_[algorithm_name]_results
mv ./results.panoptic.json ./panoptic_test-dev2017_[algorithm_name]_results.json
zip panoptic_test-dev2017_[algorithm_name]_results.zip -ur panoptic_test-dev2017_
↪[algorithm_name]_results panoptic_test-dev2017_[algorithm_name]_results.json
```

3.10 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()

3.10.1 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 basic layers 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.

3.10.2 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,
                     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(
```

(continues on next page)

(continued from previous page)

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

3.10.3 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)
```

2. 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)
```

(continues on next page)

(continued from previous page)

```
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))
```

3. Initialize model with the pretrained model

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

More details can refer to the documentation in MMEngine

3.11 Use a single stage detector as RPN

Region proposal network (RPN) is a submodule in Faster R-CNN, which generates proposals for the second stage of Faster R-CNN. Most two-stage detectors in MMDetection use `RPNHead` to generate proposals as RPN. However, any single-stage detector can serve as an RPN since their bounding box predictions can also be regarded as region proposals and thus be refined in the R-CNN. Therefore, MMDetection v3.0 supports that.

To illustrate the whole process, here we give an example of how to use an anchor-free single-stage model FCOS as an RPN in Faster R-CNN.

The outline of this tutorial is as below:

1. Use `FCOSHead` as an `RPNHead` in Faster R-CNN
2. Evaluate proposals
3. Train the customized Faster R-CNN with pre-trained FCOS

3.11.1 Use `FCOSHead` as an `RPNHead` in Faster R-CNN

To set `FCOSHead` as an `RPNHead` in Faster R-CNN, we should create a new config file named `configs/faster_rcnn/faster-rcnn_r50_fpn_fcos-rpn_1x_coco.py`, and replace with the setting of `rpn_head` with the setting of `bbox_head` in `configs/fcos/fcos_r50-caffe_fpn_gn-head_1x_coco.py`. Besides, we still use the neck setting of FCOS with strides of `[8, 16, 32, 64, 128]`, and update `featmap_strides` of `bbox_roi_extractor` to `[8, 16, 32, 64, 128]`. To avoid loss goes NAN, we apply warmup during the first 1000 iterations instead of the first 500 iterations, which means that the lr increases more slowly. The config is as follows:

```
_base_ = [
    '../_base_/models/faster-rcnn_r50_fpn.py',
    '../_base_/datasets/coco_detection.py',
    '../_base_/schedules/schedule_1x.py', '../_base_/default_runtime.py'
]

model = dict(
    # copied from configs/fcos/fcos_r50-caffe_fpn_gn-head_1x_coco.py
    neck=dict(
        start_level=1,
        add_extra_convs='on_output',  # use P5
        relu_before_extra_convs=True),
    rpn_head=dict(
        _delete_=True,  # ignore the unused old settings
        type='FCOSHead',
        num_classes=1,  # num_classes = 1 for rpn, if num_classes > 1, it will be set to␣
↪1 in TwoStageDetector automatically
        in_channels=256,
        stacked_convs=4,
        feat_channels=256,
        strides=[8, 16, 32, 64, 128],
        loss_cls=dict(
            type='FocalLoss',
            use_sigmoid=True,
            gamma=2.0,
            alpha=0.25,
            loss_weight=1.0),
        loss_bbox=dict(type='IoULoss', loss_weight=1.0),
        loss_centerness=dict(
            type='CrossEntropyLoss', use_sigmoid=True, loss_weight=1.0)),
    roi_head=dict(  # update featmap_strides due to the strides in neck
        bbox_roi_extractor=dict(featmap_strides=[8, 16, 32, 64, 128])))

# learning rate
param_scheduler = [
    dict(
        type='LinearLR', start_factor=0.001, by_epoch=False, begin=0,
        end=1000),  # Slowly increase lr, otherwise loss becomes NAN
    dict(
        type='MultiStepLR',
        begin=0,
        end=12,
        by_epoch=True,
        milestones=[8, 11],
        gamma=0.1)
]
```

Then, we could use the following command to train our customized model. For more training commands, please refer to here.

```
# training with 8 GPUS
bash tools/dist_train.sh configs/faster_rcnn/faster-rcnn_r50_fpn_fcos-rpn_1x_coco.py \
    8 \
```

(continues on next page)

(continued from previous page)

```
    --work-dir ./work_dirs/faster-rcnn_r50_fpn_fcos-rpn_1x_coco
```

3.11.2 Evaluate proposals

The quality of proposals is of great importance to the performance of detector, therefore, we also provide a way to evaluate proposals. Same as above, create a new config file named `configs/rpn/fcos-rpn_r50_fpn_1x_coco.py`, and replace with setting of `rpn_head` with the setting of `bbox_head` in `configs/fcos/fcos_r50-caffe_fpn_gn-head_1x_coco.py`.

```
_base_ = [
    '../_base_/models/rpn_r50_fpn.py', '../_base_/datasets/coco_detection.py',
    '../_base_/schedules/schedule_1x.py', '../_base_/default_runtime.py'
]

val_evaluator = dict(metric='proposal_fast')
test_evaluator = val_evaluator

model = dict(
    # copied from configs/fcos/fcos_r50-caffe_fpn_gn-head_1x_coco.py
    neck=dict(
        start_level=1,
        add_extra_convs='on_output',  # use P5
        relu_before_extra_convs=True),
    rpn_head=dict(
        _delete_=True,  # ignore the unused old settings
        type='FCOSHead',
        num_classes=1,  # num_classes = 1 for rpn, if num_classes > 1, it will be set to␣
↪1 in RPN automatically
        in_channels=256,
        stacked_convs=4,
        feat_channels=256,
        strides=[8, 16, 32, 64, 128],
        loss_cls=dict(
            type='FocalLoss',
            use_sigmoid=True,
            gamma=2.0,
            alpha=0.25,
            loss_weight=1.0),
        loss_bbox=dict(type='IoULoss', loss_weight=1.0),
        loss_centerness=dict(
            type='CrossEntropyLoss', use_sigmoid=True, loss_weight=1.0)))
```

Suppose we have the checkpoint `./work_dirs/faster-rcnn_r50_fpn_fcos-rpn_1x_coco/epoch_12.pth` after training, then we can evaluate the quality of proposals with the following command.

```
# testing with 8 GPUs
bash tools/dist_test.sh \
    configs/rpn/fcos-rpn_r50_fpn_1x_coco.py \
    ./work_dirs/faster-rcnn_r50_fpn_fcos-rpn_1x_coco/epoch_12.pth \
    8
```

3.11.3 Train the customized Faster R-CNN with pre-trained FCOS

Pre-training not only speeds up convergence of training, but also improves the performance of the detector. Therefore, here we give an example to illustrate how to do use a pre-trained FCOS as an RPN to accelerate training and improve the accuracy. Suppose we want to use `FCOSHead` as an rpn head in Faster R-CNN and train with the pre-trained `fcos_r50-caffe_fpn_gn-head_1x_coco`. The content of config file named `configs/faster_rcnn/faster-rcnn_r50-caffe_fpn_fcos-rpn_1x_coco.py` is as the following. Note that `fcos_r50-caffe_fpn_gn-head_1x_coco` uses a caffe version of ResNet50, the pixel mean and std in `data_preprocessor` thus need to be updated.

```
_base_ = [
    '../_base_/models/faster-rcnn_r50_fpn.py',
    '../_base_/datasets/coco_detection.py',
    '../_base_/schedules/schedule_1x.py', '../_base_/default_runtime.py'
]

model = dict(
    data_preprocessor=dict(
        mean=[103.530, 116.280, 123.675],
        std=[1.0, 1.0, 1.0],
        bgr_to_rgb=False),
    backbone=dict(
        norm_cfg=dict(type='BN', requires_grad=False),
        style='caffe',
        init_cfg=None),  # the checkpoint in ``load_from`` contains the weights of␣
↪backbone
    neck=dict(
        start_level=1,
        add_extra_convs='on_output',  # use P5
        relu_before_extra_convs=True),
    rpn_head=dict(
        _delete_=True,  # ignore the unused old settings
        type='FCOSHead',
        num_classes=1,  # num_classes = 1 for rpn, if num_classes > 1, it will be set to␣
↪1 in TwoStageDetector automatically
        in_channels=256,
        stacked_convs=4,
        feat_channels=256,
        strides=[8, 16, 32, 64, 128],
        loss_cls=dict(
            type='FocalLoss',
            use_sigmoid=True,
            gamma=2.0,
            alpha=0.25,
            loss_weight=1.0),
        loss_bbox=dict(type='IoULoss', loss_weight=1.0),
        loss_centerness=dict(
            type='CrossEntropyLoss', use_sigmoid=True, loss_weight=1.0)),
    roi_head=dict(  # update featmap_strides due to the strides in neck
        bbox_roi_extractor=dict(featmap_strides=[8, 16, 32, 64, 128])))

load_from = 'https://download.openmmlab.com/mmdetection/v2.0/fcos/fcos_r50_caffe_fpn_gn-
↪head_1x_coco/fcos_r50_caffe_fpn_gn-head_1x_coco-821213aa.pth'
```

The command for training is as below.

```
bash tools/dist_train.sh \
    configs/faster_rcnn/faster-rcnn_r50-caffe_fpn_fcos-rpn_1x_coco.py \
    8 \
    --work-dir ./work_dirs/faster-rcnn_r50-caffe_fpn_fcos-rpn_1x_coco
```

3.12 Semi-supervised Object Detection

Semi-supervised object detection uses both labeled data and unlabeled data for training. It not only reduces the annotation burden for training high-performance object detectors but also further improves the object detector by using a large number of unlabeled data.

A typical procedure to train a semi-supervised object detector is as below:

- Prepare and split dataset
- Configure multi-branch pipeline
- Configure semi-supervised dataloader
- Configure semi-supervised model
- Configure MeanTeacherHook
- Configure TeacherStudentValLoop

3.12.1 Prepare and split dataset

We provide a dataset download script, which downloads the coco2017 dataset by default and decompresses it automatically.

```
python tools/misc/download_dataset.py
```

The decompressed dataset directory structure is as below:

```
mmdetection
├── data
│   ├── coco
│   │   ├── annotations
│   │   │   ├── image_info_unlabeled2017.json
│   │   │   ├── instances_train2017.json
│   │   │   └── instances_val2017.json
│   │   ├── test2017
│   │   ├── train2017
│   │   ├── unlabeled2017
│   │   ├── val2017
```

There are two common experimental settings for semi-supervised object detection on the coco2017 dataset:

(1) Split `train2017` according to a fixed percentage (1%, 2%, 5% and 10%) as a labeled dataset, and the rest of `train2017` as an unlabeled dataset. Because the different splits of `train2017` as labeled datasets will cause significant fluctuation on the accuracy of the semi-supervised detectors, five-fold cross-validation is used in practice to evaluate the algorithm. We provide the dataset split script:

```
python tools/misc/split_coco.py
```

By default, the script will split `train2017` according to the labeled data ratio 1%, 2%, 5% and 10%, and each split will be randomly repeated 5 times for cross-validation. The generated semi-supervised annotation file name format is as below:

- the name format of labeled dataset: `instances_train2017.{fold}@{percent}.json`
- the name format of unlabeled dataset: `instances_train2017.{fold}@{percent}-unlabeled.json`

Here, `fold` is used for cross-validation, and `percent` represents the ratio of labeled data. The directory structure of the divided dataset is as below:

```
mmdetection
├── data
│   ├── coco
│   │   ├── annotations
│   │   │   ├── image_info_unlabeled2017.json
│   │   │   ├── instances_train2017.json
│   │   │   ├── instances_val2017.json
│   │   ├── semi_anns
│   │   │   ├── instances_train2017.1@1.json
│   │   │   ├── instances_train2017.1@1-unlabeled.json
│   │   │   ├── instances_train2017.1@2.json
│   │   │   ├── instances_train2017.1@2-unlabeled.json
│   │   │   ├── instances_train2017.1@5.json
│   │   │   ├── instances_train2017.1@5-unlabeled.json
│   │   │   ├── instances_train2017.1@10.json
│   │   │   ├── instances_train2017.1@10-unlabeled.json
│   │   │   ├── instances_train2017.2@1.json
│   │   │   ├── instances_train2017.2@1-unlabeled.json
│   │   ├── test2017
│   │   ├── train2017
│   │   ├── unlabeled2017
│   │   ├── val2017
```

(2) Use `train2017` as the labeled dataset and `unlabeled2017` as the unlabeled dataset. Since `image_info_unlabeled2017.json` does not contain `categories` information, the `CocoDataset` cannot be initialized, so you need to write the `categories` of `instances_train2017.json` into `image_info_unlabeled2017.json` and save it as `instances_unlabeled2017.json`, the relevant script is as below:

```
from mmengine.fileio import load, dump

anns_train = load('instances_train2017.json')
anns_unlabeled = load('image_info_unlabeled2017.json')
anns_unlabeled['categories'] = anns_train['categories']
dump(anns_unlabeled, 'instances_unlabeled2017.json')
```

The processed dataset directory is as below:

```
mmdetection
├── data
│   ├── coco
│   │   ├── annotations
│   │   │   ├── image_info_unlabeled2017.json
```

(continues on next page)

(continued from previous page)

```
│   │   │   ├── instances_train2017.json
│   │   │   ├── instances_unlabeled2017.json
│   │   │   ├── instances_val2017.json
│   │   ├── test2017
│   │   ├── train2017
│   │   ├── unlabeled2017
│   │   ├── val2017
```

3.12.2 Configure multi-branch pipeline

There are two main approaches to semi-supervised learning, consistency regularization and pseudo label. Consistency regularization often requires some careful design, while pseudo label have a simpler form and are easier to extend to downstream tasks. We adopt a teacher-student joint training semi-supervised object detection framework based on pseudo label, so labeled data and unlabeled data need to configure different data pipeline:

(1) Pipeline for labeled data

```python
# pipeline used to augment labeled data,
# which will be sent to student model for supervised training.
sup_pipeline = [
    dict(type='LoadImageFromFile', file_client_args=file_client_args),
    dict(type='LoadAnnotations', with_bbox=True),
    dict(type='RandomResize', scale=scale, keep_ratio=True),
    dict(type='RandomFlip', prob=0.5),
    dict(type='RandAugment', aug_space=color_space, aug_num=1),
    dict(type='FilterAnnotations', min_gt_bbox_wh=(1e-2, 1e-2)),
    dict(type='MultiBranch', sup=dict(type='PackDetInputs'))
]
```

(2) Pipeline for unlabeled data

```python
# pipeline used to augment unlabeled data weakly,
# which will be sent to teacher model for predicting pseudo instances.
weak_pipeline = [
    dict(type='RandomResize', scale=scale, keep_ratio=True),
    dict(type='RandomFlip', prob=0.5),
    dict(
        type='PackDetInputs',
        meta_keys=('img_id', 'img_path', 'ori_shape', 'img_shape',
                   'scale_factor', 'flip', 'flip_direction',
                   'homography_matrix')),
]

# pipeline used to augment unlabeled data strongly,
# which will be sent to student model for unsupervised training.
strong_pipeline = [
    dict(type='RandomResize', scale=scale, keep_ratio=True),
    dict(type='RandomFlip', prob=0.5),
    dict(
        type='RandomOrder',
        transforms=[
```

(continues on next page)

(continued from previous page)

```
            dict(type='RandAugment', aug_space=color_space, aug_num=1),
            dict(type='RandAugment', aug_space=geometric, aug_num=1),
        ]),
    dict(type='RandomErasing', n_patches=(1, 5), ratio=(0, 0.2)),
    dict(type='FilterAnnotations', min_gt_bbox_wh=(1e-2, 1e-2)),
    dict(
        type='PackDetInputs',
        meta_keys=('img_id', 'img_path', 'ori_shape', 'img_shape',
                   'scale_factor', 'flip', 'flip_direction',
                   'homography_matrix')),
]

# pipeline used to augment unlabeled data into different views
unsup_pipeline = [
    dict(type='LoadImageFromFile', file_client_args=file_client_args),
    dict(type='LoadEmptyAnnotations'),
    dict(
        type='MultiBranch',
        unsup_teacher=weak_pipeline,
        unsup_student=strong_pipeline,
    )
]
```

3.12.3 Configure semi-supervised dataloader

(1) Build a semi-supervised dataset. Use `ConcatDataset` to concatenate labeled and unlabeled datasets.

```
labeled_dataset = dict(
    type=dataset_type,
    data_root=data_root,
    ann_file='annotations/instances_train2017.json',
    data_prefix=dict(img='train2017/'),
    filter_cfg=dict(filter_empty_gt=True, min_size=32),
    pipeline=sup_pipeline)

unlabeled_dataset = dict(
    type=dataset_type,
    data_root=data_root,
    ann_file='annotations/instances_unlabeled2017.json',
    data_prefix=dict(img='unlabeled2017/'),
    filter_cfg=dict(filter_empty_gt=False),
    pipeline=unsup_pipeline)

train_dataloader = dict(
    batch_size=batch_size,
    num_workers=num_workers,
    persistent_workers=True,
    sampler=dict(
        type='GroupMultiSourceSampler',
        batch_size=batch_size,
        source_ratio=[1, 4]),
```

(continues on next page)

(continued from previous page)

```
    dataset=dict(
        type='ConcatDataset', datasets=[labeled_dataset, unlabeled_dataset]))
```

(2) Use multi-source dataset sampler. Use `GroupMultiSourceSampler` to sample data form batches from `labeled_dataset` and `labeled_dataset`, `source_ratio` controls the proportion of labeled data and unlabeled data in the batch. `GroupMultiSourceSampler` also ensures that the images in the same batch have similar aspect ratios. If you don't need to guarantee the aspect ratio of the images in the batch, you can use `MultiSourceSampler`. The sampling diagram of `GroupMultiSourceSampler` is as below:

`sup=1000` indicates that the scale of the labeled dataset is 1000, `sup_h=200` indicates that the scale of the images with an aspect ratio greater than or equal to 1 in the labeled dataset is 200, and `sup_w=800` indicates that the scale of the images with an aspect ratio less than 1 in the labeled dataset is 800, `unsup=9000` indicates that the scale of the unlabeled dataset is 9000, `unsup_h=1800` indicates that the scale of the images with an aspect ratio greater than or equal to 1 in the unlabeled dataset is 1800, and `unsup_w=7200` indicates the scale of the images with an aspect ratio less than 1 in the unlabeled dataset is 7200. `GroupMultiSourceSampler` randomly selects a group according to the overall aspect ratio distribution of the images in the labeled dataset and the unlabeled dataset, and then sample data to form batches from the two datasets according to `source_ratio`, so labeled datasets and unlabeled datasets have different repetitions.

3.12.4 Configure semi-supervised model

We choose `Faster R-CNN` as `detector` for semi-supervised training. Take the semi-supervised object detection algorithm `SoftTeacher` as an example, the model configuration can be inherited from `_base_/models/faster-rcnn_r50_fpn.py`, replacing the backbone network of the detector with `caffe` style. Note that unlike the supervised training configs, `Faster R-CNN` as `detector` is an attribute of `model`, not `model` . In addition, `data_preprocessor` needs to be set to `MultiBranchDataPreprocessor`, which is used to pad and normalize images from different pipelines. Finally, parameters required for semi-supervised training and testing can be configured via `semi_train_cfg` and `semi_test_cfg`.

```
_base_ = [
    '../_base_/models/faster-rcnn_r50_fpn.py', '../_base_/default_runtime.py',
    '../_base_/datasets/semi_coco_detection.py'
]

detector = _base_.model
detector.data_preprocessor = dict(
    type='DetDataPreprocessor',
    mean=[103.530, 116.280, 123.675],
    std=[1.0, 1.0, 1.0],
    bgr_to_rgb=False,
    pad_size_divisor=32)
detector.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=False),
    norm_eval=True,
    style='caffe',
    init_cfg=dict(
        type='Pretrained',
```

(continues on next page)

(continued from previous page)

```
        checkpoint='open-mmlab://detectron2/resnet50_caffe'))

model = dict(
    _delete_=True,
    type='SoftTeacher',
    detector=detector,
    data_preprocessor=dict(
        type='MultiBranchDataPreprocessor',
        data_preprocessor=detector.data_preprocessor),
    semi_train_cfg=dict(
        freeze_teacher=True,
        sup_weight=1.0,
        unsup_weight=4.0,
        pseudo_label_initial_score_thr=0.5,
        rpn_pseudo_thr=0.9,
        cls_pseudo_thr=0.9,
        reg_pseudo_thr=0.02,
        jitter_times=10,
        jitter_scale=0.06,
        min_pseudo_bbox_wh=(1e-2, 1e-2)),
    semi_test_cfg=dict(predict_on='teacher'))
```

In addition, we also support semi-supervised training for other detection models, such as `RetinaNet` and `Cascade R-CNN`. Since `SoftTeacher` only supports `Faster R-CNN`, it needs to be replaced with `SemiBaseDetector`, example is as below:

```
_base_ = [
    '../_base_/models/retinanet_r50_fpn.py', '../_base_/default_runtime.py',
    '../_base_/datasets/semi_coco_detection.py'
]

detector = _base_.model

model = dict(
    _delete_=True,
    type='SemiBaseDetector',
    detector=detector,
    data_preprocessor=dict(
        type='MultiBranchDataPreprocessor',
        data_preprocessor=detector.data_preprocessor),
    semi_train_cfg=dict(
        freeze_teacher=True,
        sup_weight=1.0,
        unsup_weight=1.0,
        cls_pseudo_thr=0.9,
        min_pseudo_bbox_wh=(1e-2, 1e-2)),
    semi_test_cfg=dict(predict_on='teacher'))
```

Following the semi-supervised training configuration of `SoftTeacher`, change `batch_size` to 2 and `source_ratio` to `[1, 1]`, the experimental results of supervised and semi-supervised training of `RetinaNet`, `Faster R-CNN`, `Cascade R-CNN` and `SoftTeacher` on the 10% coco `train2017` are as below:

3.12.5 Configure MeanTeacherHook

Usually, the teacher model is updated by Exponential Moving Average (EMA) the student model, and then the teacher model is optimized with the optimization of the student model, which can be achieved by configuring `custom_hooks`:

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

3.12.6 Configure TeacherStudentValLoop

Since there are two models in the teacher-student joint training framework, we can replace `ValLoop` with `TeacherStudentValLoop` to test the accuracy of both models during the training process.

```
val_cfg = dict(type='TeacherStudentValLoop')
```

CHAPTER

FOUR

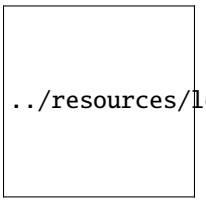
USEFUL TOOLS

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

4.1 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 ${KEYS}] [--eval-interval
↪${EVALUATION_INTERVAL}] [--title ${TITLE}] [--legend ${LEGEND}] [--backend ${BACKEND}]␣
↪[--style ${STYLE}] [--out ${OUT_FILE}]
```



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
```

4.2 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
```

2. 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
```

3. 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
```

4.3 Visualization

4.3.1 Visualize Datasets

`tools/analysis_tools/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}]
```

4.3.2 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.

4.3.3 Visualize Predictions

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

4.4 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 modify the test_evaluator to save the results bbox by:

1. Find which dataset in 'configs/base/datasets' the current config corresponds to.

2. Replace the original test_evaluator and test_dataloader with test_evaluator and test_dataloader in the comment in dateset config.
3. Use the following command to get the results bbox and segmentation json file.

```
python tools/test.py \
       configs/mask_rcnn/mask-rcnn_r50_fpn_1x_coco.py \
       checkpoint/mask_rcnn_r50_fpn_1x_coco_20200205-d4b0c5d6.pth \
```

1. Get COCO bbox error results per category , save analyze result images to the directory(In config the default directory is './work_dirs/coco_instance/test')

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

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

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

4.5 Model Serving

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

4.5.1 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.

4.5.2 2. Build `mmdet-serve` docker image

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

4.5.3 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

4.5.4 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
```

(continues on next page)

(continued from previous page)

```
    ],
    "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_8xb8-320-273e_coco.py \
checkpoint/yolov3_d53_320_273e_coco-421362b6.pth \
yolov3
```

4.6 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.

4.7 Model conversion

4.7.1 MMDetection model to ONNX

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

4.7.2 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]
```

4.7.3 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]
```

4.7.4 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]
```

4.7.5 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`.

4.8 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}]
```

4.9 Dataset Download

tools/misc/download_dataset.py supports downloading datasets such as COCO, VOC, and LVIS.

```
python tools/misc/download_dataset.py --dataset-name coco2017
python tools/misc/download_dataset.py --dataset-name voc2007
python tools/misc/download_dataset.py --dataset-name lvis
```

4.10 Benchmark

4.10.1 Robust Detection Benchmark

tools/analysis_tools/test_robustness.py andtools/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*.

4.10.2 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
```

4.11 Miscellaneous

4.11.1 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 ...]}]
```

4.11.2 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...]}]
```

4.12 Hyper-parameter Optimization

4.12.1 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_8xb8-320-273e_
↪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.
```

(continues on next page)

(continued from previous page)

```
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
```

4.13 Confusion Matrix

A confusion matrix is a summary of prediction results.

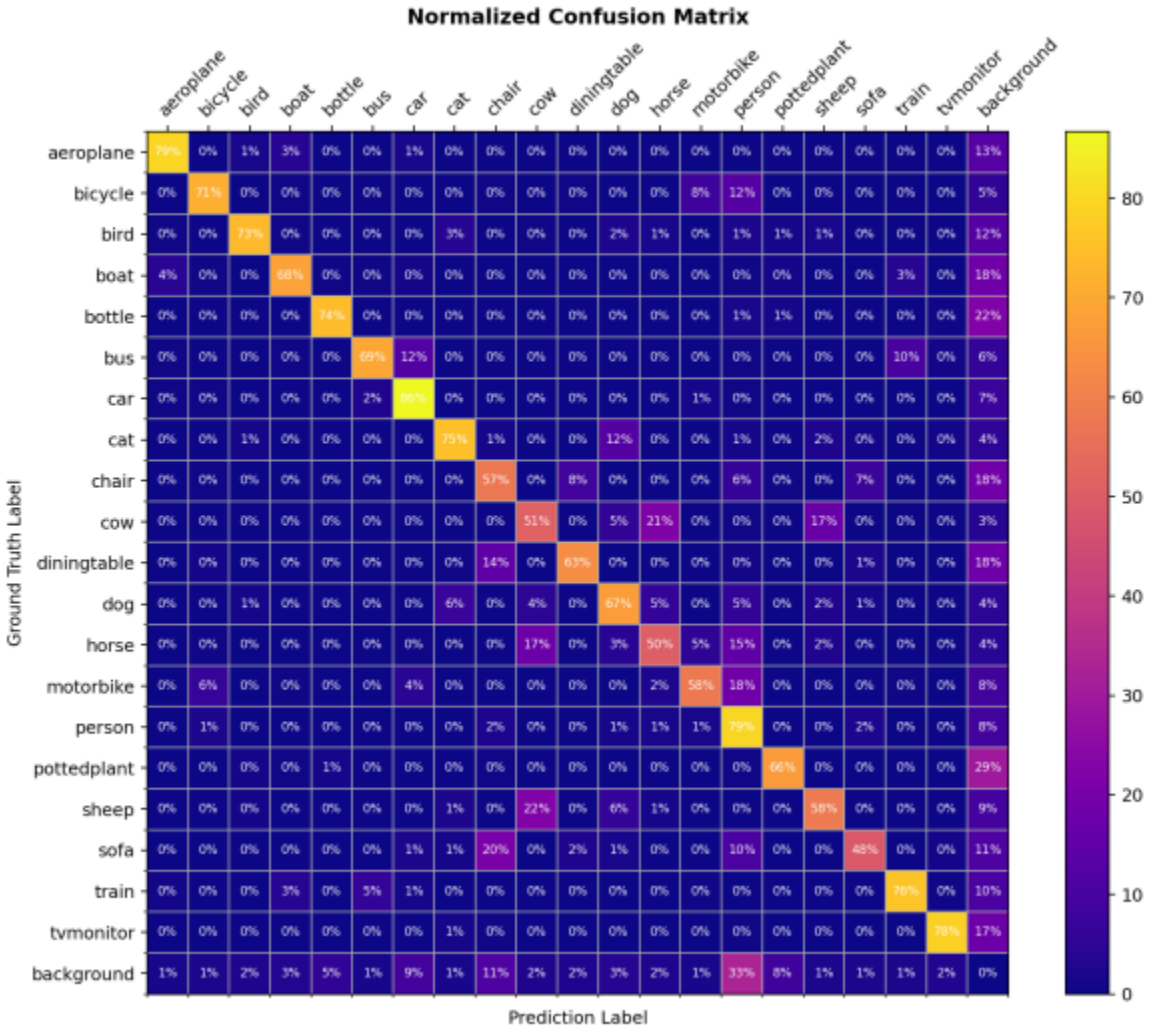
`tools/analysis_tools/confusion_matrix.py` can analyze the prediction results and plot a confusion matrix table.

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 confusion matrix like this:



4.14 Useful Hooks

MMDetection and MMEngine provide users with various useful hooks including log hooks, `NumClassCheckHook`, etc. This tutorial introduces the functionalities and usages of hooks implemented in MMDetection. For using hooks in MMEngine, please read the API documentation in MMEngine.

4.14.1 CheckInvalidLossHook

4.14.2 NumClassCheckHook

4.14.3 MemoryProfilerHook

Memory profiler hook records memory information including virtual memory, swap memory, and the memory of the current process. This hook helps grasp the memory usage of the system and discover potential memory leak bugs. To use this hook, users should install `memory_profiler` and `psutil` by `pip install memory_profiler psutil` first.

Usage

To use this hook, users should add the following code to the config file.

```
custom_hooks = [
    dict(type='MemoryProfilerHook', interval=50)
]
```

Result

During training, you can see the messages in the log recorded by `MemoryProfilerHook` as below. The system has 250 GB (246360 MB + 9407 MB) of memory and 8 GB (5740 MB + 2452 MB) of swap memory in total. Currently 9407 MB (4.4%) of memory and 5740 MB (29.9%) of swap memory were consumed. And the current training process consumed 5434 MB of memory.

```
2022-04-21 08:49:56,881 - mmengine - INFO - Memory information available_memory: 246360␣
↪MB, used_memory: 9407 MB, memory_utilization: 4.4 %, available_swap_memory: 5740 MB,␣
↪used_swap_memory: 2452 MB, swap_memory_utilization: 29.9 %, current_process_memory:␣
↪5434 MB
```

4.14.4 SetEpochInfoHook

4.14.5 SyncNormHook

4.14.6 SyncRandomSizeHook

4.14.7 YOLOXLrUpdaterHook

4.14.8 YOLOXModeSwitchHook

4.14.9 How to implement a custom hook

In general, there are 20 points where hooks can be inserted from the beginning to the end of model training. The users can implement custom hooks and insert them at different points in the process of training to do what they want.

- global points: `before_run`, `after_run`
- points in training: `before_train`, `before_train_epoch`, `before_train_iter`, `after_train_iter`, `after_train_epoch`, `after_train`
- points in validation: `before_val`, `before_val_epoch`, `before_val_iter`, `after_val_iter`, `after_val_epoch`, `after_val`
- points at testing: `before_test`, `before_test_epoch`, `before_test_iter`, `after_test_iter`, `after_test_epoch`, `after_test`
- other points: `before_save_checkpoint`, `after_save_checkpoint`

For example, users can implement a hook to check loss and terminate training when loss goes NaN. To achieve that, there are three steps to go:

1. Implement a new hook that inherits the `Hook` class in MMEngine, and implement `after_train_iter` method which checks whether loss goes NaN after every `n` training iterations.

2. The implemented hook should be registered in `HOOKS` by `@HOOKS.register_module()` as shown in the code below.
3. Add `custom_hooks = [dict(type='MemoryProfilerHook', interval=50)]` in the config file.

```python
from typing import Optional

import torch
from mmengine.hooks import Hook
from mmengine.runner import Runner

from mmdet.registry import HOOKS


@HOOKS.register_module()
class CheckInvalidLossHook(Hook):
    """Check invalid loss hook.

    This hook will regularly check whether the loss is valid
    during training.

    Args:
        interval (int): Checking interval (every k iterations).
            Default: 50.
    """

    def __init__(self, interval: int = 50) -> None:
        self.interval = interval

    def after_train_iter(self,
                         runner: Runner,
                         batch_idx: int,
                         data_batch: Optional[dict] = None,
                         outputs: Optional[dict] = None) -> None:
        """Regularly check whether the loss is valid every n iterations.

        Args:
            runner (:obj:`Runner`): The runner of the training process.
            batch_idx (int): The index of the current batch in the train loop.
            data_batch (dict, Optional): Data from dataloader.
                Defaults to None.
            outputs (dict, Optional): Outputs from model. Defaults to None.
        """
        if self.every_n_train_iters(runner, self.interval):
            assert torch.isfinite(outputs['loss']), \
                runner.logger.info('loss become infinite or NaN!')
```

Please read customize_runtime for more about implementing a custom hook.

4.15 Visualization

4.16 Corruption Benchmarking

4.16.1 Introduction

We provide tools to test object detection and instance segmentation models on the image corruption benchmark defined in Benchmarking Robustness in Object Detection: Autonomous Driving when Winter is Coming. This page provides basic tutorials how to use the benchmark.

```
@article{michaelis2019winter,
  title={Benchmarking Robustness in Object Detection:
    Autonomous Driving when Winter is Coming},
  author={Michaelis, Claudio and Mitzkus, Benjamin and
    Geirhos, Robert and Rusak, Evgenia and
    Bringmann, Oliver and Ecker, Alexander S. and
    Bethge, Matthias and Brendel, Wieland},
  journal={arXiv:1907.07484},
  year={2019}
}
```

4.16.2 About the benchmark

To submit results to the benchmark please visit the benchmark homepage

The benchmark is modelled after the imagenet-c benchmark which was originally published in Benchmarking Neural Network Robustness to Common Corruptions and Perturbations (ICLR 2019) by Dan Hendrycks and Thomas Dietterich.

The image corruption functions are included in this library but can be installed separately using:

```
pip install imagecorruptions
```

Compared to imagenet-c a few changes had to be made to handle images of arbitrary size and greyscale images. We also modified the 'motion blur' and 'snow' corruptions to remove dependency from a linux specific library, which would have to be installed separately otherwise. For details please refer to the imagecorruptions repository.

4.16.3 Inference with pretrained models

We provide a testing script to evaluate a models performance on any combination of the corruptions provided in the benchmark.

Test a dataset

- [x] single GPU testing
- [ ] multiple GPU testing
- [ ] visualize detection results

You can use the following commands to test a models performance under the 15 corruptions used in the benchmark.

```
# single-gpu testing
python tools/analysis_tools/test_robustness.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--out $
↪{RESULT_FILE}] [--eval ${EVAL_METRICS}]
```

Alternatively different group of corruptions can be selected.

```
# noise
python tools/analysis_tools/test_robustness.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--out $
↪{RESULT_FILE}] [--eval ${EVAL_METRICS}] --corruptions noise

# blur
python tools/analysis_tools/test_robustness.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--out $
↪{RESULT_FILE}] [--eval ${EVAL_METRICS}] --corruptions blur

# wetaher
python tools/analysis_tools/test_robustness.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--out $
↪{RESULT_FILE}] [--eval ${EVAL_METRICS}] --corruptions weather

# digital
python tools/analysis_tools/test_robustness.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--out $
↪{RESULT_FILE}] [--eval ${EVAL_METRICS}] --corruptions digital
```

Or a costom set of corruptions e.g.:

```
# gaussian noise, zoom blur and snow
python tools/analysis_tools/test_robustness.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--out $
↪{RESULT_FILE}] [--eval ${EVAL_METRICS}] --corruptions gaussian_noise zoom_blur snow
```

Finally the corruption severities to evaluate can be chosen. Severity 0 corresponds to clean data and the effect increases from 1 to 5.

```
# severity 1
python tools/analysis_tools/test_robustness.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--out $
↪{RESULT_FILE}] [--eval ${EVAL_METRICS}] --severities 1

# severities 0,2,4
python tools/analysis_tools/test_robustness.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--out $
↪{RESULT_FILE}] [--eval ${EVAL_METRICS}] --severities 0 2 4
```

4.16.4 Results for modelzoo models

The results on COCO 2017val are shown in the below table.

Results may vary slightly due to the stochastic application of the corruptions.

CHAPTER

FIVE

BASIC CONCEPTS

5.1 Data Flow

5.2 Structures

5.3 Models

5.4 Datasets

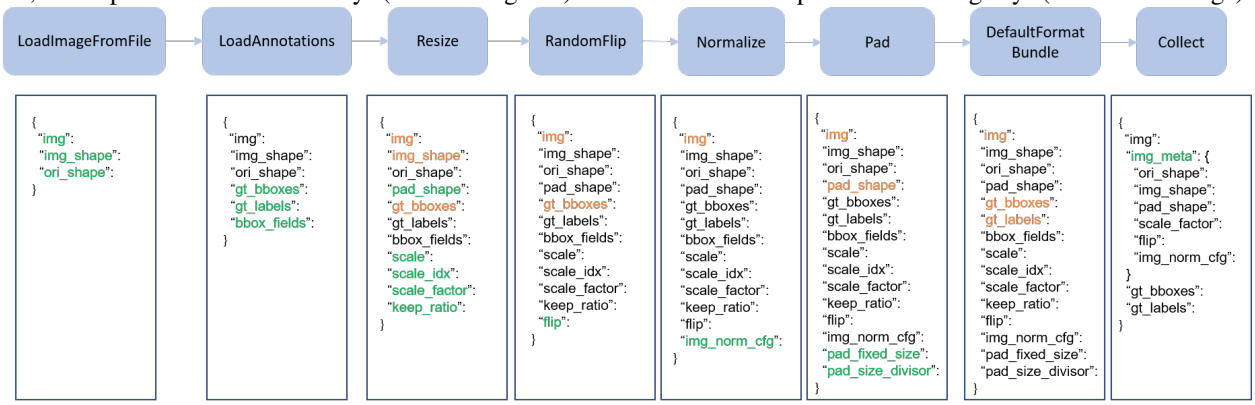
5.5 Data Transforms

5.5.1 Design of Data transforms pipeline

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.

The data transforms pipeline and the dataset is decomposed. Usually a dataset defines how to process the annotations and a data transforms pipeline defines all the steps to prepare a data dict. A pipeline consists of a sequence of data transforms. 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).



Here is a pipeline example for Faster R-CNN.

```
train_pipeline = [  # Training data processing 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
    dict(
        type='Resize',  # Pipeline that resize the images and their annotations
        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
        prob=0.5),  # The probability to flip
    dict(type='PackDetInputs')  # Pipeline that formats the annotation data and decides␣
↪which keys in the data should be packed into data_samples
]
test_pipeline = [  # Testing data processing pipeline
    dict(type='LoadImageFromFile', file_client_args=file_client_args),  # First pipeline␣
↪to load images from file path
    dict(type='Resize', scale=(1333, 800), keep_ratio=True),  # Pipeline that resize the␣
↪images
    dict(
        type='PackDetInputs',  # Pipeline that formats the annotation data and decides␣
↪which keys in the data should be packed into data_samples
        meta_keys=('img_id', 'img_path', 'ori_shape', 'img_shape',
                   'scale_factor'))
]
```

5.6 Evaluation

5.7 Engine

5.8 Conventions

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

5.8.1 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(...)
```

(continues on next page)

(continued from previous page)

```
        # 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()`.

5.8.2 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)
        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.

5.8.3 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

SIX

COMPONENT CUSTOMIZATION

6.1 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.

6.1.1 Develop new components

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 mmdet.registry import MODELS


@MODELS.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),
    ...
```

Add new necks

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

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

```
import torch.nn as nn

from mmdet.registry import MODELS

@MODELS.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'],
    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)
```

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 typing import Tuple

import torch.nn as nn
from mmcv.cnn import ConvModule
from mmengine.model import BaseModule, ModuleList
from torch import Tensor

from mmdet.models.backbones.resnet import Bottleneck
from mmdet.registry import MODELS
from mmdet.utils import ConfigType, MultiConfig, OptConfigType, OptMultiConfig
from .bbox_head import BBoxHead

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

    .. code-block:: none

                                         /-> cls
                     /-> shared convs ->
                                         \-> reg
```

(continues on next page)

(continued from previous page)

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

    def __init__(self,
                 num_convs: int = 0,
                 num_fcs: int = 0,
                 conv_out_channels: int = 1024,
                 fc_out_channels: int = 1024,
                 conv_cfg: OptConfigType = None,
                 norm_cfg: ConfigType = dict(type='BN'),
                 init_cfg: MultiConfig = dict(
                     type='Normal',
                     override=[
                         dict(type='Normal', name='fc_cls', std=0.01),
                         dict(type='Normal', name='fc_reg', std=0.001),
                         dict(
                             type='Xavier',
                             name='fc_branch',
                             distribution='uniform')
                     ]),
                 **kwargs) -> None:
        kwargs.setdefault('with_avg_pool', True)
        super().__init__(init_cfg=init_cfg, **kwargs)

    def forward(self, x_cls: Tensor, x_reg: Tensor) -> Tuple[Tensor]:
```

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.

```
from typing import List, Optional, Tuple

import torch
from torch import Tensor

from mmdet.registry import MODELS, TASK_UTILS
from mmdet.structures import DetDataSample
from mmdet.structures.bbox import bbox2roi
from mmdet.utils import ConfigType, InstanceList
from ..task_modules.samplers import SamplingResult
from ..utils import empty_instances, unpack_gt_instances
from .base_roi_head import BaseRoIHead


@MODELS.register_module()
class StandardRoIHead(BaseRoIHead):
    """Simplest base roi head including one bbox head and one mask head."""

    def init_assigner_sampler(self) -> None:
```

(continues on next page)

(continued from previous page)

```
    def init_bbox_head(self, bbox_roi_extractor: ConfigType,
                       bbox_head: ConfigType) -> None:

    def init_mask_head(self, mask_roi_extractor: ConfigType,
                       mask_head: ConfigType) -> None:

    def forward(self, x: Tuple[Tensor],
                rpn_results_list: InstanceList) -> tuple:

    def loss(self, x: Tuple[Tensor], rpn_results_list: InstanceList,
             batch_data_samples: List[DetDataSample]) -> dict:

    def _bbox_forward(self, x: Tuple[Tensor], rois: Tensor) -> dict:

    def bbox_loss(self, x: Tuple[Tensor],
                  sampling_results: List[SamplingResult]) -> dict:

    def mask_loss(self, x: Tuple[Tensor],
                  sampling_results: List[SamplingResult], bbox_feats: Tensor,
                  batch_gt_instances: InstanceList) -> dict:

    def _mask_forward(self,
                      x: Tuple[Tensor],
                      rois: Tensor = None,
                      pos_inds: Optional[Tensor] = None,
                      bbox_feats: Optional[Tensor] = None) -> dict:

    def predict_bbox(self,
                     x: Tuple[Tensor],
                     batch_img_metas: List[dict],
                     rpn_results_list: InstanceList,
                     rcnn_test_cfg: ConfigType,
                     rescale: bool = False) -> InstanceList:

    def predict_mask(self,
                     x: Tuple[Tensor],
                     batch_img_metas: List[dict],
                     results_list: InstanceList,
                     rescale: bool = False) -> InstanceList:
```

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 typing import Tuple

from torch import Tensor

from mmdet.registry import MODELS
from .standard_roi_head import StandardRoIHead
```

(continues on next page)

(continued from previous page)

```python
@MODELS.register_module()
class DoubleHeadRoIHead(StandardRoIHead):
    """RoI head for `Double Head RCNN <https://arxiv.org/abs/1904.06493>`_.

    Args:
        reg_roi_scale_factor (float): The scale factor to extend the rois
            used to extract the regression features.
    """

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

    def _bbox_forward(self, x: Tuple[Tensor], rois: Tensor) -> dict:
        """Box head forward function used in both training and testing.

        Args:
            x (tuple[Tensor]): List of multi-level img features.
            rois (Tensor): RoIs with the shape (n, 5) where the first
                column indicates batch id of each RoI.

        Returns:
             dict[str, Tensor]: Usually returns a dictionary with keys:

                - `cls_score` (Tensor): Classification scores.
                - `bbox_pred` (Tensor): Box energies / deltas.
                - `bbox_feats` (Tensor): Extract bbox RoI features.
        """
        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.roi_heads.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.

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 mmdet.registry import MODELS
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)
```

(continues on next page)

(continued from previous page)

```
    return loss

@MODELS.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))
```

6.2 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.

6.2.1 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 five 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**.

6.2.2 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.

6.2.3 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)
```

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)
```

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')
```

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)
```

6.2.4 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):

    ...
```

(continues on next page)

(continued from previous page)

```
    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):
```

6.3 Customize Datasets

6.3.1 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.

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, and it is in polygon XY point␣
↪coordinate format, you need to ensure that at least 3 point coordinates are included.␣
↪Otherwise, it is an invalid polygon.
        'area': 1035.749,
        'iscrowd': 0,
        'image_id': 1268,
```

(continues on next page)

(continued from previous page)

```
        'bbox': [192.81, 224.8, 74.73, 33.43],
        'category_id': 16,
        'id': 42986
    },
    ...
],

'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 `metainfo=dict(CLASSES=classes)` fields in `train_dataloader.dataset`, `val_dataloader.dataset` and `test_dataloader.dataset` and `classes` must be a tuple type.
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_root='path/to/your/'

train_dataloader = dict(
    batch_size=2,
    num_workers=2,
    dataset=dict(
        type=dataset_type,
```

(continues on next page)

(continued from previous page)

```
        # explicitly add your class names to the field `metainfo`
        metainfo=dict(CLASSES=classes),
        data_root=data_root,
        ann_file='train/annotation_data',
        data_prefix=dict(img='train/image_data')
        )
    )

val_dataloader = dict(
    batch_size=1,
    num_workers=2,
    dataset=dict(
        type=dataset_type,
        test_mode=True,
        # explicitly add your class names to the field `metainfo`
        metainfo=dict(CLASSES=classes),
        data_root=data_root,
        ann_file='val/annotation_data',
        data_prefix=dict(img='val/image_data')
        )
    )

test_dataloader = dict(
    batch_size=1,
    num_workers=2,
    dataset=dict(
        type=dataset_type,
        test_mode=True,
        # explicitly add your class names to the field `metainfo`
        metainfo=dict(CLASSES=classes),
        data_root=data_root,
        ann_file='test/annotation_data',
        data_prefix=dict(img='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',
```

(continues on next page)

(continued from previous page)

```
                # 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,
            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.

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 in MMEninge's BaseDataset and all existing datasets are processed to be compatible with it, either online or offline.

The annotation of the dataset must be in `json` or `yaml`, `yml` or `pickle`, `pkl` format; the dictionary stored in the annotation file must contain two fields `metainfo` and `data_list`. The `metainfo` is a dictionary, which contains the metadata of the dataset, such as class information; `data_list` is a list, each element in the list is a dictionary, the dictionary defines the raw data of one image, and each raw data contains a or several training/testing samples.

Here is an example.

```
{
    'metainfo':
        {
            'classes': ('person', 'bicycle', 'car', 'motorcycle'),
            ...
        },
    'data_list':
        [
            {
                "img_path": "xxx/xxx_1.jpg",
                "height": 604,
                "width": 640,
                "instances":
                [
                  {
                    "bbox": [0, 0, 10, 20],
                    "bbox_label": 1,
                    "ignore_flag": 0
                  },
                  {
                    "bbox": [10, 10, 110, 120],
                    "bbox_label": 2,
                    "ignore_flag": 0
                  }
                ]
              },
            {
                "img_path": "xxx/xxx_2.jpg",
                "height": 320,
                "width": 460,
                "instances":
                [
                  {
                    "bbox": [10, 0, 20, 20],
                    "bbox_label": 3,
                    "ignore_flag": 1,
                  }
                ]
```

(continues on next page)

(continued from previous page)

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

Some datasets may provide annotations like crowd/difficult/ignored bboxes, we use `ignore_flag`to cover them.

After obtaining the above standard data annotation format, you can directly use BaseDetDataset of MMDetection in the configuration , without conversion.

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 mmengine

from mmdet.base_det_dataset import BaseDetDataset
from mmdet.registry import DATASETS


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

    METAINFO = {
        'CLASSES': ('person', 'bicycle', 'car', 'motorcycle'),
        'PALETTE': [(220, 20, 60), (119, 11, 32), (0, 0, 142), (0, 0, 230)]
    }

    def load_data_list(self, ann_file):
        ann_list = mmengine.list_from_file(ann_file)

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

(continues on next page)

(continued from previous page)

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

            instances = []
            for anns in ann_list[i + 4:i + 4 + bbox_number]:
                instance = {}
                instance['bbox'] = [float(ann) for ann in anns.split(' ')[:4]]
                instance['bbox_label']=int(anns[4])
                                instances.append(instance)

            data_infos.append(
                dict(
                    img_path=ann_list[i + 1],
                    img_id=i,
                    width=width,
                    height=height,
                    instances=instances
                ))

        return data_infos
```

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
)
```

6.3.2 Customize datasets by dataset wrappers

MMEngine 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.

For detailed usage, see MMEngine Dataset Wrapper.

6.3.3 Modify Dataset Classes

With existing dataset types, we can modify the metainfo 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')
train_dataloader = dict(
    dataset=dict(
        metainfo=dict(CLASSES=classes))
    )
val_dataloader = dict(
    dataset=dict(
        metainfo=dict(CLASSES=classes))
    )
test_dataloader = dict(
    dataset=dict(
        metainfo=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_cfg=dict(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.
- When directly using `BaseDataset` in MMEngine or `BaseDetDataset` in MMDetection, users cannot filter images without GT by modifying the configuration, but it can be solved in an offline way.
- 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).

6.3.4 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': [
    {
```

(continues on next page)

(continued from previous page)

```
        'filename': '000000001268.jpg',
        'image_id': 1268,
        'segments_info': [
            {
                '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` must be set to the path of the panoptic annotation images.

```
dataset_type = 'CocoPanopticDataset'
data_root='path/to/your/'

train_dataloader = dict(
    dataset=dict(
        type=dataset_type,
        data_root=data_root,
        data_prefix=dict(
            img='train/image_data/', seg='train/panoptic/image_annotation_data/')
    )
)
val_dataloader = dict(
    dataset=dict(
        type=dataset_type,
        data_root=data_root,
        data_prefix=dict(
            img='val/image_data/', seg='val/panoptic/image_annotation_data/')
    )
)
test_dataloader = dict(
    dataset=dict(
        type=dataset_type,
        data_root=data_root,
        data_prefix=dict(
            img='test/image_data/', seg='test/panoptic/image_annotation_data/')
    )
)
```

6.4 Customize Data Pipelines

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

```python
import random
from mmcv.transforms import BaseTransform
from mmdet.registry import TRANSFORMS


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

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

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

    def transform(self, results):
        if random.random() > self.prob:
            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.

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

train_pipeline = [
    dict(type='LoadImageFromFile', file_client_args=file_client_args),
    dict(type='LoadAnnotations', with_bbox=True),
    dict(type='Resize', scale=(1333, 800), keep_ratio=True),
    dict(type='RandomFlip', prob=0.5),
    dict(type='MyTransform', prob=0.2),
    dict(type='PackDetInputs')
]
```

3. Visualize the output of your transforms pipeline

   To visualize the output of your transforms 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 *visualization documentation*

6.5 Customize Runtime Settings

6.5.1 Customize optimization settings

Optimization related configuration is now all managed by `optim_wrapper` which usually has three fields: `optimizer`, `paramwise_cfg`, `clip_grad`, refer to OptimWrapper for more detail. See the example below, where `Adamw` is used as an `optimizer`, the learning rate of the backbone is reduced by a factor of 10, and gradient clipping is added.

```
optim_wrapper = dict(
    type='OptimWrapper',
    # optimizer
    optimizer=dict(
        type='AdamW',
        lr=0.0001,
        weight_decay=0.05,
        eps=1e-8,
        betas=(0.9, 0.999)),

    # Parameter-level learning rate and weight decay settings
    paramwise_cfg=dict(
        custom_keys={
            'backbone': dict(lr_mult=0.1, decay_mult=1.0),
        },
        norm_decay_mult=0.0),

    # gradient clipping
    clip_grad=dict(max_norm=0.01, norm_type=2))
```

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 in `optim_wrapper` 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.

```
optim_wrapper = dict(
    type='OptimWrapper',
    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 `optimizer`. The users can directly set arguments following the API doc of PyTorch.

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/engine/optimizers`. And then implement the new optimizer in a file, e.g., in `mmdet/engine/optimizers/my_optimizer.py`:

```
from mmdet.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/engine/optimizers/__init__.py` to import it.

  The newly defined module should be imported in `mmdet/engine/optimizers/__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.engine.optimizers.my_optimizer'], allow_failed_
↪imports=False)
```

The module `mmdet.engine.optimizers.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.engine.optimizers.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 in `optim_wrapper` field of config files. In the configs, the optimizers are defined by the field `optimizer` like the following:

```
optim_wrapper = dict(
    type='OptimWrapper',
    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

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

Customize optimizer wrapper 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 wrapper constructor.

```
from mmengine.optim import DefaultOptiWrapperConstructor

from mmdet.registry import OPTIM_WRAPPER_CONSTRUCTORS
from .my_optimizer import MyOptimizer


@OPTIM_WRAPPER_CONSTRUCTORS.register_module()
class MyOptimizerWrapperConstructor(DefaultOptimWrapperConstructor):

    def __init__(self,
                 optim_wrapper_cfg: dict,
                 paramwise_cfg: Optional[dict] = None):

    def __call__(self, model: nn.Module) -> OptimWrapper:

        return optim_wrapper
```

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

Additional settings

Tricks not implemented by the optimizer should be implemented through optimizer wrapper 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:

  ```
  optim_wrapper = dict(
      _delete_=True, clip_grad=dict(max_norm=35, norm_type=2))
  ```

  If your config inherits the base config which already sets the `optim_wrapper`, 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 CosineAnnealingLR and CosineAnnealingMomentum.

  ```
  param_scheduler = [
      # learning rate scheduler
      # During the first 8 epochs, learning rate increases from 0 to lr * 10
      # during the next 12 epochs, learning rate decreases from lr * 10 to lr * 1e-4
      dict(
          type='CosineAnnealingLR',
          T_max=8,
  ```

(continues on next page)

(continued from previous page)

```
        eta_min=lr * 10,
        begin=0,
        end=8,
        by_epoch=True,
        convert_to_iter_based=True),
    dict(
        type='CosineAnnealingLR',
        T_max=12,
        eta_min=lr * 1e-4,
        begin=8,
        end=20,
        by_epoch=True,
        convert_to_iter_based=True),
    # momentum scheduler
    # During the first 8 epochs, momentum increases from 0 to 0.85 / 0.95
    # during the next 12 epochs, momentum increases from 0.85 / 0.95 to 1
    dict(
        type='CosineAnnealingMomentum',
        T_max=8,
        eta_min=0.85 / 0.95,
        begin=0,
        end=8,
        by_epoch=True,
        convert_to_iter_based=True),
    dict(
        type='CosineAnnealingMomentum',
        T_max=12,
        eta_min=1,
        begin=8,
        end=20,
        by_epoch=True,
        convert_to_iter_based=True)
]
```

6.5.2 Customize training schedules

By default we use step learning rate with 1x schedule, this calls MultiStepLR in MMEngine. We support many other learning rate schedule here, such as `CosineAnnealingLR` and `PolyLR` schedule. Here are some examples

- Poly schedule:

```
param_scheduler = [
    dict(
        type='PolyLR',
        power=0.9,
        eta_min=1e-4,
        begin=0,
        end=8,
        by_epoch=True)]
```

- ConsineAnnealing schedule:

```
param_scheduler = [
    dict(
        type='CosineAnnealingLR',
        T_max=8,
        eta_min=lr * 1e-5,
        begin=0,
        end=8,
        by_epoch=True)]
```

6.5.3 Customize train loop

By default, EpochBasedTrainLoop is used in train_cfg and validation is done after every train epoch, as follows.

```
train_cfg = dict(type='EpochBasedTrainLoop', max_epochs=12, val_begin=1, val_interval=1)
```

Actually, both IterBasedTrainLoop and EpochBasedTrainLoop support dynamical interval, see the following example.

```
# Before 365001th iteration, we do evaluation every 5000 iterations.
# After 365000th iteration, we do evaluation every 368750 iteraions,
# which means that we do evaluation at the end of training.

interval = 5000
max_iters = 368750
dynamic_intervals = [(max_iters // interval * interval + 1, max_iters)]
train_cfg = dict(
    type='IterBasedTrainLoop',
    max_iters=max_iters,
    val_interval=interval,
    dynamic_intervals=dynamic_intervals)
```

6.5.4 Customize hooks

Customize self-implemented hooks

1. Implement a new hook

MMEngine provides many useful hooks, but there are some occasions when the users might need to implement a new hook. MMDetection supports customized hooks in training in v3.0 . 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. Here we give an example of creating a new hook in mmdet and using it in training.

```
from mmengine.hooks import Hook
from mmdet.registry import HOOKS


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

(continues on next page)

(continued from previous page)

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

    def before_run(self, runner) -> None:

    def after_run(self, runner) -> None:

    def before_train(self, runner) -> None:

    def after_train(self, runner) -> None:

    def before_train_epoch(self, runner) -> None:

    def after_train_epoch(self, runner) -> None:

    def before_train_iter(self,
                          runner,
                          batch_idx: int,
                          data_batch: DATA_BATCH = None) -> None:

    def after_train_iter(self,
                         runner,
                         batch_idx: int,
                         data_batch: DATA_BATCH = None,
                         outputs: Optional[dict] = None) -> None:
```

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_train`, `after_train` , `before_train_epoch`, `after_train_epoch`, `before_train_iter`, and `after_train_iter`. There are more points where hooks can be inserted, refer to base hook class for more detail.

2. Register the new hook

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

- Modify `mmdet/engine/hooks/__init__.py` to import it.

  The newly defined module should be imported in `mmdet/engine/hooks/__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.engine.hooks.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.

Use hooks implemented in MMDetection

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

Example: `NumClassCheckHook`

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

We set it in default_runtime.py.

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

Modify default runtime hooks

There are some common hooks that are registered through `default_hooks`, they are

- `IterTimerHook`: A hook that logs 'data_time' for loading data and 'time' for a model train step.
- `LoggerHook`: A hook that Collect logs from different components of `Runner` and write them to terminal, JSON file, tensorboard and wandb .etc.
- `ParamSchedulerHook`: A hook to update some hyper-parameters in optimizer, e.g., learning rate and momentum.
- `CheckpointHook`: A hook that saves checkpoints periodically.
- `DistSamplerSeedHook`: A hook that sets the seed for sampler and batch_sampler.
- `DetVisualizationHook`: A hook used to visualize validation and testing process prediction results.

`IterTimerHook`, `ParamSchedulerHook` and `DistSamplerSeedHook` are simple and no need to be modified usually, so here we reveals how what we can do with `LoggerHook`, `CheckpointHook` and `DetVisualizationHook`.

CheckpointHook

Except saving checkpoints periodically, CheckpointHook provides other options such as max_keep_ckpts, save_optimizer and etc. The users could set max_keep_ckpts to only save small number of checkpoints or decide whether to store state dict of optimizer by save_optimizer. More details of the arguments are here

```
default_hooks = dict(
    checkpoint=dict(
        type='CheckpointHook',
        interval=1,
        max_keep_ckpts=3,
        save_optimizer=True))
```

LoggerHook

The LoggerHook enables to set intervals. And the detail usages can be found in the docstring.

```
default_hooks = dict(logger=dict(type='LoggerHook', interval=50))
```

DetVisualizationHook

DetVisualizationHook use DetLocalVisualizer to visualize prediction results, and DetLocalVisualizer current supports different backends, e.g., TensorboardVisBackend and WandbVisBackend (see docstring for more detail). The users could add multi backbends to do visualization, as follows.

```
default_hooks = dict(
    visualization=dict(type='DetVisualizationHook', draw=True))

vis_backends = [dict(type='LocalVisBackend'),
                dict(type='TensorboardVisBackend')]
visualizer = dict(
    type='DetLocalVisualizer', vis_backends=vis_backends, name='visualizer')
```

CHAPTER

SEVEN

HOW TO

This tutorial collects answers to any `How to xxx with MMDetection`. Feel free to update this doc if you meet new questions about `How to` and find the answers!

7.1 Use backbone network through MMClassification

The model registry in MMDet, MMCls, MMSeg all inherit from the root registry in MMEngine. This allows these repositories to directly use the modules already implemented by each other. Therefore, users can use backbone networks from MMClassification in MMDetection without implementing a network that already exists in MMClassification.

7.1.1 Use backbone network implemented in MMClassification

Suppose you want to use `MobileNetV3-small` as the backbone network of `RetinaNet`, the example config is as the following.

```
_base_ = [
    '../_base_/models/retinanet_r50_fpn.py',
    '../_base_/datasets/coco_detection.py',
    '../_base_/schedules/schedule_1x.py', '../_base_/default_runtime.py'
]
# please install mmcls>=1.0.0rc0
# import mmcls.models to trigger register_module in mmcls
custom_imports = dict(imports=['mmcls.models'], allow_failed_imports=False)
pretrained = 'https://download.openmmlab.com/mmclassification/v0/mobilenet_v3/convert/
↪mobilenet_v3_small-8427ecf0.pth'
model = dict(
    backbone=dict(
        _delete_=True, # Delete the backbone field in _base_
        type='mmcls.MobileNetV3', # Using MobileNetV3 from mmcls
        arch='small',
        out_indices=(3, 8, 11), # Modify out_indices
        init_cfg=dict(
            type='Pretrained',
            checkpoint=pretrained,
            prefix='backbone.')), # The pre-trained weights of backbone network in MMCls␣
↪have prefix='backbone.'. The prefix in the keys will be removed so that these weights␣
↪can be normally loaded.
    # Modify in_channels
    neck=dict(in_channels=[24, 48, 96], start_level=0))
```

7.1.2 Use backbone network in TIMM through MMClassification

MMClassification also provides a wrapper for the PyTorch Image Models (timm) backbone network, users can directly use the backbone network in timm through MMClassification. Suppose you want to use EfficientNet-B1 as the backbone network of RetinaNet, the example config is as the following.

```
# https://github.com/open-mmlab/mmdetection/blob/dev-3.x/configs/timm_example/retinanet_
↪timm-efficientnet-b1_fpn_1x_coco.py

_base_ = [
    '../_base_/models/retinanet_r50_fpn.py',
    '../_base_/datasets/coco_detection.py',
    '../_base_/schedules/schedule_1x.py', '../_base_/default_runtime.py'
]

# please install mmcls>=1.0.0rc0
# import mmcls.models to trigger register_module in mmcls
custom_imports = dict(imports=['mmcls.models'], allow_failed_imports=False)
model = dict(
    backbone=dict(
        _delete_=True, # Delete the backbone field in _base_
        type='mmcls.TIMMBackbone', # Using timm from mmcls
        model_name='efficientnet_b1',
        features_only=True,
        pretrained=True,
        out_indices=(1, 2, 3, 4)), # Modify out_indices
    neck=dict(in_channels=[24, 40, 112, 320])) # Modify in_channels

optimizer = dict(type='SGD', lr=0.01, momentum=0.9, weight_decay=0.0001)
```

`type='mmcls.TIMMBackbone'` means use the `TIMMBackbone` class from MMClassification in MMDetection, and the model used is `EfficientNet-B1`, where `mmcls` means the MMClassification repo and `TIMMBackbone` means the TIMMBackbone wrapper implemented in MMClassification.

For the principle of the Hierarchy Registry, please refer to the MMEngine document. For how to use other backbones in MMClassification, you can refer to the MMClassification document.

7.2 Use Mosaic augmentation

If you want to use `Mosaic` in training, please make sure that you use `MultiImageMixDataset` at the same time. Taking the 'Faster R-CNN' algorithm as an example, you should modify the values of `train_pipeline` and `train_dataset` in the config as below:

```
# Open configs/faster_rcnn/faster-rcnn_r50_fpn_1x_coco.py directly and add the following␣
↪fields
data_root = 'data/coco/'
dataset_type = 'CocoDataset'
img_scale=(1333, 800)

train_pipeline = [
    dict(type='Mosaic', img_scale=img_scale, pad_val=114.0),
    dict(
```

(continues on next page)

(continued from previous page)

```
        type='RandomAffine',
        scaling_ratio_range=(0.1, 2),
        border=(-img_scale[0] // 2, -img_scale[1] // 2)), # The image will be enlarged␣
↪by 4 times after Mosaic processing,so we use affine transformation to restore the␣
↪image size.
    dict(type='RandomFlip', prob=0.5),
    dict(type='PackDetInputs')
]

train_dataset = dict(
    _delete_ = True, # remove unnecessary Settings
    type='MultiImageMixDataset',
    dataset=dict(
        type=dataset_type,
        ann_file=data_root + 'annotations/instances_train2017.json',
        img_prefix=data_root + 'train2017/',
        pipeline=[
            dict(type='LoadImageFromFile'),
            dict(type='LoadAnnotations', with_bbox=True)
        ],
        filter_empty_gt=False,
    ),
    pipeline=train_pipeline
    )

data = dict(
    train=train_dataset
    )
```

7.3 Unfreeze backbone network after freezing the backbone in the config

If you have freezed the backbone network in the config and want to unfreeze it after some epoches, you can write a hook function to do it. Taking the Faster R-CNN with the resnet backbone as an example, you can freeze one stage of the backbone network and add a `custom_hooks` in the config as below:

```
_base_ = [
    '../_base_/models/faster-rcnn_r50_fpn.py',
    '../_base_/datasets/coco_detection.py',
    '../_base_/schedules/schedule_1x.py', '../_base_/default_runtime.py'
]
model = dict(
    # freeze one stage of the backbone network.
    backbone=dict(frozen_stages=1),
)
custom_hooks = [dict(type="UnfreezeBackboneEpochBasedHook", unfreeze_epoch=1)]
```

Meanwhile write the hook class `UnfreezeBackboneEpochBasedHook` in `mmdet/core/hook/unfreeze_backbone_epoch_based_hook.py`

```
from mmengine.model import is_model_wrapper
from mmengine.hooks import Hook
from mmdet.registry import HOOKS


@HOOKS.register_module()
class UnfreezeBackboneEpochBasedHook(Hook):
    """Unfreeze backbone network Hook.

    Args:
        unfreeze_epoch (int): The epoch unfreezing the backbone network.
    """

    def __init__(self, unfreeze_epoch=1):
        self.unfreeze_epoch = unfreeze_epoch

    def before_train_epoch(self, runner):
        # Unfreeze the backbone network.
        # Only valid for resnet.
        if runner.epoch == self.unfreeze_epoch:
            model = runner.model
            if is_model_wrapper(model):
                model = model.module
            backbone = model.backbone
            if backbone.frozen_stages >= 0:
                if backbone.deep_stem:
                    backbone.stem.train()
                    for param in backbone.stem.parameters():
                        param.requires_grad = True
                else:
                    backbone.norm1.train()
                    for m in [backbone.conv1, backbone.norm1]:
                        for param in m.parameters():
                            param.requires_grad = True

            for i in range(1, backbone.frozen_stages + 1):
                m = getattr(backbone, f'layer{i}')
                m.train()
                for param in m.parameters():
                    param.requires_grad = True
```

7.4 Get the channels of a new backbone

If you want to get the channels of a new backbone, you can build this backbone alone and input a pseudo image to get each stage output.

Take `ResNet` as an example:

```
from mmdet.models import ResNet
import torch
self = ResNet(depth=18)
```

(continues on next page)

(continued from previous page)

```
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))
```

Output of the above script is as below:

```
(1, 64, 8, 8)
(1, 128, 4, 4)
(1, 256, 2, 2)
(1, 512, 1, 1)
```

Users can get the channels of the new backbone by Replacing the `ResNet(depth=18)` in this script with their customized backbone.

CHAPTER

EIGHT

MIGRATION

CHAPTER

NINE

MMDET.APIS

CHAPTER

TEN

MMDET.DATASETS

10.1 datasets

10.2 api_wrappers

10.3 samplers

10.4 transforms

CHAPTER

ELEVEN

MMDET.ENGINE

11.1 hooks

11.2 optimizers

11.3 runner

11.4 schedulers

CHAPTER

TWELVE

MMDET.EVALUATION

12.1 functional

12.2 metrics

CHAPTER

THIRTEEN

MMDET.MODELS

CHAPTER

FOURTEEN

MMDET.STRUCTURES

14.1 bbox

14.2 mask

CHAPTER

FIFTEEN

MMDET.TESTING

CHAPTER

SIXTEEN

MMDET.VISULIZATION

CHAPTER

SEVENTEEN

MMDET.UTILS

CHAPTER

EIGHTEEN

BENCHMARK AND MODEL ZOO

18.1 Mirror sites

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

18.2 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.

18.3 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 :

18.4 Baselines

18.4.1 RPN

Please refer to RPN for details.

18.4.2 Faster R-CNN

Please refer to Faster R-CNN for details.

18.4.3 Mask R-CNN

Please refer to Mask R-CNN for details.

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

Please refer to Fast R-CNN for details.

18.4.5 RetinaNet

Please refer to RetinaNet for details.

18.4.6 Cascade R-CNN and Cascade Mask R-CNN

Please refer to Cascade R-CNN for details.

18.4.7 Hybrid Task Cascade (HTC)

Please refer to HTC for details.

18.4.8 SSD

Please refer to SSD for details.

18.4.9 Group Normalization (GN)

Please refer to Group Normalization for details.

18.4.10 Weight Standardization

Please refer to Weight Standardization for details.

18.4.11 Deformable Convolution v2

Please refer to Deformable Convolutional Networks for details.

18.4.12 CARAFE: Content-Aware ReAssembly of FEatures

Please refer to CARAFE for details.

18.4.13 Instaboost

Please refer to Instaboost for details.

18.4.14 Libra R-CNN

Please refer to Libra R-CNN for details.

18.4.15 Guided Anchoring

Please refer to Guided Anchoring for details.

18.4.16 FCOS

Please refer to FCOS for details.

18.4.17 FoveaBox

Please refer to FoveaBox for details.

18.4.18 RepPoints

Please refer to RepPoints for details.

18.4.19 FreeAnchor

Please refer to FreeAnchor for details.

18.4.20 Grid R-CNN (plus)

Please refer to Grid R-CNN for details.

18.4.21 GHM

Please refer to GHM for details.

18.4.22 GCNet

Please refer to GCNet for details.

18.4.23 HRNet

Please refer to HRNet for details.

18.4.24 Mask Scoring R-CNN

Please refer to Mask Scoring R-CNN for details.

18.4.25 Train from Scratch

Please refer to Rethinking ImageNet Pre-training for details.

18.4.26 NAS-FPN

Please refer to NAS-FPN for details.

18.4.27 ATSS

Please refer to ATSS for details.

18.4.28 FSAF

Please refer to FSAF for details.

18.4.29 RegNetX

Please refer to RegNet for details.

18.4.30 Res2Net

Please refer to Res2Net for details.

18.4.31 GRoIE

Please refer to GRoIE for details.

18.4.32 Dynamic R-CNN

Please refer to Dynamic R-CNN for details.

18.4.33 PointRend

Please refer to PointRend for details.

18.4.34 DetectoRS

Please refer to DetectoRS for details.

18.4.35 Generalized Focal Loss

Please refer to Generalized Focal Loss for details.

18.4.36 CornerNet

Please refer to CornerNet for details.

18.4.37 YOLOv3

Please refer to YOLOv3 for details.

18.4.38 PAA

Please refer to PAA for details.

18.4.39 SABL

Please refer to SABL for details.

18.4.40 CentripetalNet

Please refer to CentripetalNet for details.

18.4.41 ResNeSt

Please refer to ResNeSt for details.

18.4.42 DETR

Please refer to DETR for details.

18.4.43 Deformable DETR

Please refer to Deformable DETR for details.

18.4.44 AutoAssign

Please refer to AutoAssign for details.

18.4.45 YOLOF

Please refer to YOLOF for details.

18.4.46 Seesaw Loss

Please refer to Seesaw Loss for details.

18.4.47 CenterNet

Please refer to CenterNet for details.

18.4.48 YOLOX

Please refer to YOLOX for details.

18.4.49 PVT

Please refer to PVT for details.

18.4.50 SOLO

Please refer to SOLO for details.

18.4.51 QueryInst

Please refer to QueryInst for details.

18.4.52 PanopticFPN

Please refer to PanopticFPN for details.

18.4.53 MaskFormer

Please refer to MaskFormer for details.

18.4.54 DyHead

Please refer to DyHead for details.

18.4.55 Mask2Former

Please refer to Mask2Former for details.

18.4.56 Efficientnet

Please refer to Efficientnet for details.

18.4.57 Other datasets

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

18.4.58 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.

18.5 Speed benchmark

18.5.1 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.

18.5.2 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 tools/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.

18.6 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.

18.6.1 Hardware

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

18.6.2 Software environment

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

18.6.3 Performance

18.6.4 Training Speed

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

18.6.5 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.

18.6.6 Training memory

CHAPTER

NINETEEN

CONTRIBUTION

CHAPTER

TWENTY

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.

20.1 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.

20.2 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]
- 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

TWENTYONE

CHANGELOG OF V3.X

21.1 v3.0.0rc0 (31/8/2022)

We are excited to announce the release of MMDetection 3.0.0rc0. MMDet 3.0.0rc0 is the first version of MMDetection 3.x, a part of the OpenMMLab 2.0 projects. Built upon the new training engine, MMDet 3.x unifies the interfaces of the dataset, models, evaluation, and visualization with faster training and testing speed. It also provides a general semi-supervised object detection framework and strong baselines.

21.1.1 Highlights

1. **New engine**. MMDet 3.x is based on MMEngine, which provides a universal and powerful runner that allows more flexible customizations and significantly simplifies the entry points of high-level interfaces.
2. **Unified interfaces**. As a part of the OpenMMLab 2.0 projects, MMDet 3.x unifies and refactors the interfaces and internal logic of training, testing, datasets, models, evaluation, and visualization. All the OpenMMLab 2.0 projects share the same design in those interfaces and logic to allow the emergence of multi-task/modality algorithms.
3. **Faster speed**. We optimize the training and inference speed for common models and configurations, achieving a faster or similar speed than Detection2. Model details of benchmark will be updated in this note.
4. **General semi-supervised object detection**. Benefitting from the unified interfaces, we support a general semi-supervised learning framework that works with all the object detectors supported in MMDet 3.x. Please refer to *semi-supervised object detection* for details.
5. **Strong baselines**. We release strong baselines of many popular models to enable fair comparisons among state-of-the-art models.
6. **New features and algorithms**:
   - Enable all the single-stage detectors to serve as region proposal networks
   - SoftTeacher
   - the updated CenterNet
7. **More documentation and tutorials**. We add a bunch of documentation and tutorials to help users get started more smoothly. Read it here.

21.1.2 Breaking Changes

MMDet 3.x has undergone significant changes for better design, higher efficiency, more flexibility, and more unified interfaces. Besides the changes in API, we briefly list the major breaking changes in this section. We will update the *migration guide* to provide complete details and migration instructions. Users can also refer to the API doc for more details.

Dependencies

- MMDet 3.x runs on PyTorch>=1.6. We have deprecated the support of PyTorch 1.5 to embrace mixed precision training and other new features since PyTorch 1.6. Some models can still run on PyTorch 1.5, but the full functionality of MMDet 3.x is not guaranteed.
- MMDet 3.x relies on MMEngine to run. MMEngine is a new foundational library for training deep learning models of OpenMMLab and is the core dependency of OpenMMLab 2.0 projects. The dependencies of file IO and training are migrated from MMCV 1.x to MMEngine.
- MMDet 3.x relies on MMCV>=2.0.0rc0. Although MMCV no longer maintains the training functionalities since 2.0.0rc0, MMDet 3.x relies on the data transforms, CUDA operators, and image processing interfaces in MMCV. Note that the package `mmcv` is the version that provides pre-built CUDA operators and `mmcv-lite` does not since MMCV 2.0.0rc0, while `mmcv-full` has been deprecated since 2.0.0rc0.

Training and testing

- MMDet 3.x uses Runner in MMEngine rather than that in MMCV. The new Runner implements and unifies the building logic of the dataset, model, evaluation, and visualizer. Therefore, MMDet 3.x no longer maintains the building logic of those modules in `mmdet.train.apis` and `tools/train.py`. Those codes have been migrated into MMEngine. Please refer to the migration guide of Runner in MMEngine for more details.
- The Runner in MMEngine also supports testing and validation. The testing scripts are also simplified, which has similar logic to that in training scripts to build the runner.
- The execution points of hooks in the new Runner have been enriched to allow more flexible customization. Please refer to the migration guide of Hook in MMEngine for more details.
- Learning rate and momentum schedules have been migrated from Hook to Parameter Scheduler in MMEngine. Please refer to the migration guide of Parameter Scheduler in MMEngine for more details.

Configs

- The Runner in MMEngine uses a different config structure to ease the understanding of the components in the runner. Users can read the *config example of MMDet 3.x* or refer to the migration guide in MMEngine for migration details.
- The file names of configs and models are also refactored to follow the new rules unified across OpenMMLab 2.0 projects. The names of checkpoints are not updated for now as there is no BC-breaking of model weights between MMDet 3.x and 2.x. We will progressively replace all the model weights with those trained in MMDet 3.x. Please refer to the *user guides of config* for more details.

Dataset

The Dataset classes implemented in MMDet 3.x all inherit from the `BaseDetDataset`, which inherits from the Base-Dataset in MMEngine. In addition to the changes in interfaces, there are several changes in Dataset in MMDet 3.x.

- All the datasets support serializing the internal data list to reduce the memory when multiple workers are built for data loading.
- The internal data structure in the dataset is changed to be self-contained (without losing information like class names in MMDet 2.x) while keeping simplicity.
- The evaluation functionality of each dataset has been removed from the dataset so that some specific evaluation metrics like COCO AP can be used to evaluate the prediction on other datasets.

Data Transforms

The data transforms in MMDet 3.x all inherits from `BaseTransform` in MMCV>=2.0.0rc0, which defines a new convention in OpenMMLab 2.0 projects. Besides the interface changes, there are several changes listed below:

- The functionality of some data transforms (e.g., `Resize`) are decomposed into several transforms to simplify and clarify the usages.
- The format of data dict processed by each data transform is changed according to the new data structure of dataset.
- Some inefficient data transforms (e.g., normalization and padding) are moved into data preprocessor of model to improve data loading and training speed.
- The same data transforms in different OpenMMLab 2.0 libraries have the same augmentation implementation and the logic given the same arguments, i.e., `Resize` in MMDet 3.x and MMSeg 1.x will resize the image in the exact same manner given the same arguments.

Model

The models in MMDet 3.x all inherit from `BaseModel` in MMEngine, which defines a new convention of models in OpenMMLab 2.0 projects. Users can refer to the tutorial of the model in MMengine for more details. Accordingly, there are several changes as the following:

- The model interfaces, including the input and output formats, are significantly simplified and unified following the new convention in MMDet 3.x. Specifically, all the input data in training and testing are packed into `inputs` and `data_samples`, where `inputs` contains model inputs like a list of image tensors, and `data_samples` contains other information of the current data sample such as ground truths, region proposals, and model predictions. In this way, different tasks in MMDet 3.x can share the same input arguments, which makes the models more general and suitable for multi-task learning and some flexible training paradigms like semi-supervised learning.
- The model has a data preprocessor module, which is used to pre-process the input data of the model. In MMDet 3.x, the data preprocessor usually does the necessary steps to form the input images into a batch, such as padding. It can also serve as a place for some special data augmentations or more efficient data transformations like normalization.
- The internal logic of the model has been changed. In MMdet 2.x, model uses `forward_train`, `forward_test`, `simple_test`, and `aug_test` to deal with different model forward logics. In MMDet 3.x and OpenMMLab 2.0, the forward function has three modes: 'loss', 'predict', and 'tensor' for training, inference, and tracing or other purposes, respectively. The forward function calls `self.loss`, `self.predict`, and `self._forward` given the modes 'loss', 'predict', and 'tensor', respectively.

Evaluation

The evaluation in MMDet 2.x strictly binds with the dataset. In contrast, MMDet 3.x decomposes the evaluation from dataset so that all the detection datasets can evaluate with COCO AP and other metrics implemented in MMDet 3.x. MMDet 3.x mainly implements corresponding metrics for each dataset, which are manipulated by Evaluator to complete the evaluation. Users can build an evaluator in MMDet 3.x to conduct offline evaluation, i.e., evaluate predictions that may not produce in MMDet 3.x with the dataset as long as the dataset and the prediction follow the dataset conventions. More details can be found in the tutorial in mmengine.

Visualization

The functions of visualization in MMDet 2.x are removed. Instead, in OpenMMLab 2.0 projects, we use Visualizer to visualize data. MMDet 3.x implements `DetLocalVisualizer` to allow visualization of ground truths, model predictions, feature maps, etc., at any place. It also supports sending the visualization data to any external visualization backends such as Tensorboard.

21.1.3 Improvements

- Optimized training and testing speed of FCOS, RetinaNet, Faster R-CNN, Mask R-CNN, and Cascade R-CNN. The training speed of those models with some common training strategies is also optimized, including those with synchronized batch normalization and mixed precision training.
- Support mixed precision training of all the models. However, some models may get undesirable performance due to some numerical issues. We will update the documentation and list the results (accuracy of failure) of mixed precision training.
- Release strong baselines of some popular object detectors. Their accuracy and pre-trained checkpoints will be released.

21.1.4 Bug Fixes

- DeepFashion dataset: the config and results have been updated.

21.1.5 New Features

1. Support a general semi-supervised learning framework that works with all the object detectors supported in MMDet 3.x. Please refer to *semi-supervised object detection* for details.
2. Enable all the single-stage detectors to serve as region proposal networks. We give *an example of using FCOS as RPN*.
3. Support a semi-supervised object detection algorithm: SoftTeacher.
4. Support the updated CenterNet.
5. Support data structures `HorizontalBoxes` and `BaseBoxes` to encapsulate different kinds of bounding boxes. We are migrating to use data structures of boxes to replace the use of pure tensor boxes. This will unify the usages of different kinds of bounding boxes in MMDet 3.x and MMRotate 1.x to simplify the implementation and reduce redundant codes.

21.1.6 Planned changes

We list several planned changes of MMDet 3.0.0rc0 so that the community could more comprehensively know the progress of MMDet 3.x. Feel free to create a PR, issue, or discussion if you are interested, have any suggestions and feedback, or want to participate.

1. Test-time augmentation: which is supported in MMDet 2.x, is not implemented in this version due to the limited time slot. We will support it in the following releases with a new and simplified design.
2. Inference interfaces: unified inference interfaces will be supported in the future to ease the use of released models.
3. Interfaces of useful tools that can be used in Jupyter Notebook or Colab: more useful tools that are implemented in the `tools` directory will have their python interfaces so that they can be used in Jupyter Notebook, Colab, and downstream libraries.
4. Documentation: we will add more design docs, tutorials, and migration guidance so that the community can deep dive into our new design, participate the future development, and smoothly migrate downstream libraries to MMDet 3.x.
5. Wandb visualization: MMDet 2.x supports data visualization since v2.25.0, which has not been migrated to MMDet 3.x for now. Since WandB provides strong visualization and experiment management capabilities, a `DetWandBVisualizer` and maybe a hook are planned to fully migrate those functionalities from MMDet 2.x.
6. Full support of WiderFace dataset (#8508) and Fast R-CNN: we are verifying their functionalities and will fix related issues soon.
7. Migrate DETR-series algorithms (#8655, #8533) and YOLOv3 on IPU (#8552) from MMDet 2.x.

21.1.7 Contributors

A total of 11 developers contributed to this release. Thanks @shuxp, @wanghonglie, @Czm369, @BIGWangYuDong, @zytx121, @jbwang1997, @chhluo, @jshilong, @RangiLyu, @hhaAndroid, @ZwwWayne

CHAPTER

TWENTYTWO

CHANGELOG V2.X

22.1 v2.25.0 (31/5/2022)

22.1.1 Highlights

- Support dedicated `WandbLogger` hook
- Support ConvNeXt, DDOD, SOLOv2
- Support Mask2Former for instance segmentation
- Rename config files of Mask2Former

22.1.2 Backwards incompatible changes

- Rename config files of Mask2Former (#7571)
  - `mask2former_xxx_coco.py` represents config files for **panoptic segmentation**.
  - `mask2former_xxx_coco.py` represents config files for **instance segmentation**.
  - `mask2former_xxx_coco-panoptic.py` represents config files for **panoptic segmentation**.

22.1.3 New Features

- Support ConvNeXt (#7281)
- Support DDOD (#7279)
- Support SOLOv2 (#7441)
- Support Mask2Former for instance segmentation (#7571, #8032)

22.1.4 Bug Fixes

- Enable YOLOX training on different devices (#7912)
- Fix the log plot error when evaluation with `interval != 1` (#7784)
- Fix RuntimeError of HTC (#8083)

22.1.5 Improvements

- Support dedicated `WandbLogger` hook (#7459)

  Users can set

```
cfg.log_config.hooks = [
  dict(type='MMDetWandbHook',
       init_kwargs={'project': 'MMDetection-tutorial'},
       interval=10,
       log_checkpoint=True,
       log_checkpoint_metadata=True,
       num_eval_images=10)]
```

  in the config to use `MMDetWandbHook`. Example can be found in this colab tutorial

- Add `AvoidOOM` to avoid OOM (#7434, #8091)

  Try to use `AvoidCUDAOOM` to avoid GPU out of memory. It will first retry after calling `torch.cuda.empty_cache()`. If it still fails, it will then retry by converting the type of inputs to FP16 format. If it still fails, it will try to copy inputs from GPUs to CPUs to continue computing. Try AvoidOOM in code to make the code continue to run when GPU memory runs out:

```
from mmdet.utils import AvoidCUDAOOM

output = AvoidCUDAOOM.retry_if_cuda_oom(some_function)(input1, input2)
```

  Users can also try `AvoidCUDAOOM` as a decorator to make the code continue to run when GPU memory runs out:

```
from mmdet.utils import AvoidCUDAOOM

@AvoidCUDAOOM.retry_if_cuda_oom
def function(*args, **kwargs):
    ...
    return xxx
```

- Support reading `gpu_collect` from `cfg.evaluation.gpu_collect` (#7672)
- Speedup the Video Inference by Accelerating data-loading Stage (#7832)
- Support replacing the `${key}` with the value of `cfg.key` (#7492)
- Accelerate result analysis in `analyze_result.py`. The evaluation time is speedup by 10 ~ 15 times and only tasks 10 ~ 15 minutes now. (#7891)
- Support to set `block_dilations` in `DilatedEncoder` (#7812)
- Support panoptic segmentation result analysis (#7922)
- Release DyHead with Swin-Large backbone (#7733)
- Documentations updating and adding
  - Fix wrong default type of `act_cfg` in `SwinTransformer` (#7794)
  - Fix text errors in the tutorials (#7959)
  - Rewrite the installation guide (#7897)
  - Useful hooks (#7810)
  - Fix heading anchor in documentation (#8006)

- Replace `markdownlint` with `mdformat` for avoiding installing ruby (#8009)

22.1.6 Contributors

A total of 20 developers contributed to this release.

Thanks @ZwwWayne, @DarthThomas, @solyaH, @LutingWang, @chenxinfeng4, @Czm369, @Chenastron, @chhluo, @austinmw, @Shanyaliux @hellock, @Y-M-Y, @jbwang1997, @hhaAndroid, @Irvingao, @zhanggefan, @BIGWangYuDong, @Keiku, @PeterVennerstrom, @ayulockin

22.2 v2.24.0 (26/4/2022)

22.2.1 Highlights

- Support Simple Copy-Paste is a Strong Data Augmentation Method for Instance Segmentation
- Support automatically scaling LR according to GPU number and samples per GPU
- Support Class Aware Sampler that improves performance on OpenImages Dataset

22.2.2 New Features

- Support Simple Copy-Paste is a Strong Data Augmentation Method for Instance Segmentation, see example configs (#7501)
- Support Class Aware Sampler, users can set

```
data=dict(train_dataloader=dict(class_aware_sampler=dict(num_sample_class=1)))
```

  in the config to use `ClassAwareSampler`. Examples can be found in the configs of OpenImages Dataset. (#7436)

- Support automatically scaling LR according to GPU number and samples per GPU. (#7482) In each config, there is a corresponding config of auto-scaling LR as below,

```
auto_scale_lr = dict(enable=True, base_batch_size=N)
```

  where `N` is the batch size used for the current learning rate in the config (also equals to `samples_per_gpu` * gpu number to train this config). By default, we set `enable=False` so that the original usages will not be affected. Users can set `enable=True` in each config or add `--auto-scale-lr` after the command line to enable this feature and should check the correctness of `base_batch_size` in customized configs.

- Support setting dataloader arguments in config and add functions to handle config compatibility. (#7668) The comparison between the old and new usages is as below.

```
data = dict(
    samples_per_gpu=64, workers_per_gpu=4,
    train=dict(type='xxx', ...),
    val=dict(type='xxx', samples_per_gpu=4, ...),
    test=dict(type='xxx', ...),
)
```

```
# A recommended config that is clear
data = dict(
    train=dict(type='xxx', ...),
    val=dict(type='xxx', ...),
    test=dict(type='xxx', ...),
    # Use different batch size during inference.
    train_dataloader=dict(samples_per_gpu=64, workers_per_gpu=4),
    val_dataloader=dict(samples_per_gpu=8, workers_per_gpu=2),
    test_dataloader=dict(samples_per_gpu=8, workers_per_gpu=2),
)

# Old style still works but allows to set more arguments about data loaders
data = dict(
    samples_per_gpu=64,  # only works for train_dataloader
    workers_per_gpu=4,  # only works for train_dataloader
    train=dict(type='xxx', ...),
    val=dict(type='xxx', ...),
    test=dict(type='xxx', ...),
    # Use different batch size during inference.
    val_dataloader=dict(samples_per_gpu=8, workers_per_gpu=2),
    test_dataloader=dict(samples_per_gpu=8, workers_per_gpu=2),
)
```

- Support memory profile hook. Users can use it to monitor the memory usages during training as below (#7560)

```
custom_hooks = [
    dict(type='MemoryProfilerHook', interval=50)
]
```

- Support to run on PyTorch with MLU chip (#7578)
- Support re-spliting data batch with tag (#7641)
- Support the `DiceCost` used by K-Net in `MaskHungarianAssigner` (#7716)
- Support splitting COCO data for Semi-supervised object detection (#7431)
- Support Pathlib for Config.fromfile (#7685)
- Support to use file client in OpenImages dataset (#7433)
- Add a probability parameter to Mosaic transformation (#7371)
- Support specifying interpolation mode in `Resize` pipeline (#7585)

22.2.3 Bug Fixes

- Avoid invalid bbox after deform_sampling (#7567)
- Fix the issue that argument color_theme does not take effect when exporting confusion matrix (#7701)
- Fix the `end_level` in Necks, which should be the index of the end input backbone level (#7502)
- Fix the bug that `mix_results` may be None in `MultiImageMixDataset` (#7530)
- Fix the bug in ResNet plugin when two plugins are used (#7797)

22.2.4 Improvements

- Enhance `load_json_logs` of analyze_logs.py for resumed training logs (#7732)
- Add argument `out_file` in image_demo.py (#7676)
- Allow mixed precision training with `SimOTAAssigner` (#7516)
- Updated INF to 100000.0 to be the same as that in the official YOLOX (#7778)
- Add documentations of:
  - how to get channels of a new backbone (#7642)
  - how to unfreeze the backbone network (#7570)
  - how to train fast_rcnn model (#7549)
  - proposals in Deformable DETR (#7690)
  - from-scratch install script in get_started.md (#7575)
- Release pre-trained models of
  - Mask2Former (#7595, #7709)
  - RetinaNet with ResNet-18 and release models (#7387)
  - RetinaNet with EfficientNet backbone (#7646)

22.2.5 Contributors

A total of 27 developers contributed to this release. Thanks @jovialio, @zhangsanfeng2022, @HarryZJ, @jamiechoi1995, @nestiank, @PeterH0323, @RangeKing, @Y-M-Y, @mattcasey02, @weiji14, @Yulvgit, @xiefeifeihu, @FANG-MING, @meng976537406, @nijkah, @sudz123, @CCODING04, @SheffieldCao, @Czm369, @BIGWangYuDong, @zytx121, @jbwang1997, @chhluo, @jshilong, @RangiLyu, @hhaAndroid, @ZwwWayne

22.3 v2.23.0 (28/3/2022)

22.3.1 Highlights

- Support Mask2Former: Masked-attention Mask Transformer for Universal Image Segmentation
- Support EfficientNet: EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks
- Support setting data root through environment variable `MMDET_DATASETS`, users don't have to modify the corresponding path in config files anymore.
- Find a good recipe for fine-tuning high precision ResNet backbone pre-trained by Torchvision.

22.3.2 New Features

- Support Mask2Former(#6938)(#7466)(#7471)
- Support EfficientNet (#7514)
- Support setting data root through environment variable `MMDET_DATASETS`, users don't have to modify the corresponding path in config files anymore. (#7386)
- Support setting different seeds to different ranks (#7432)
- Update the `dist_train.sh` so that the script can be used to support launching multi-node training on machines without slurm (#7415)
- Find a good recipe for fine-tuning high precision ResNet backbone pre-trained by Torchvision (#7489)

22.3.3 Bug Fixes

- Fix bug in VOC unit test which removes the data directory (#7270)
- Adjust the order of `get_classes` and `FileClient` (#7276)
- Force the inputs of `get_bboxes` in yolox_head to float32 (#7324)
- Fix misplaced arguments in LoadPanopticAnnotations (#7388)
- Fix reduction=mean in CELoss. (#7449)
- Update unit test of CrossEntropyCost (#7537)
- Fix memory leaking in panpotic segmentation evaluation (#7538)
- Fix the bug of shape broadcast in YOLOv3 (#7551)

22.3.4 Improvements

- Add Chinese version of onnx2tensorrt.md (#7219)
- Update colab tutorials (#7310)
- Update information about Localization Distillation (#7350)
- Add Chinese version of `finetune.md` (#7178)
- Update YOLOX log for non square input (#7235)
- Add `nproc` in `coco_panoptic.py` for panoptic quality computing (#7315)
- Allow to set channel_order in LoadImageFromFile (#7258)
- Take point sample related functions out of mask_point_head (#7353)
- Add instance evaluation for coco_panoptic (#7313)
- Enhance the robustness of analyze_logs.py (#7407)
- Supplementary notes of sync_random_seed (#7440)
- Update docstring of cross entropy loss (#7472)
- Update pascal voc result (#7503)
- We create How-to documentation to record any questions about How to xxx. In this version, we added
  - How to use Mosaic augmentation (#7507)

  - How to use backbone in mmcls (#7438)
  - How to produce and submit the prediction results of panoptic segmentation models on COCO test-dev set (#7430))

22.3.5 Contributors

A total of 27 developers contributed to this release. Thanks @ZwwWayne, @haofanwang, @shinya7y, @chhluo, @yangrisheng, @triple-Mu, @jbwang1997, @HikariTJU, @imflash217, @274869388, @zytx121, @matrixgame2018, @jamiechoi1995, @BIGWangYuDong, @JingweiZhang12, @Xiangxu-0103, @hhaAndroid, @jshilong, @osbm, @ceroytres, @bunge-bedstraw-herb, @Youth-Got, @daavoo, @jiangyitong, @RangiLyu, @CCODING04, @yarkable

22.4 v2.22.0 (24/2/2022)

22.4.1 Highlights

- Support MaskFormer: Per-Pixel Classification is Not All You Need for Semantic Segmentation (#7212)
- Support DyHead: Dynamic Head: Unifying Object Detection Heads with Attentions (#6823)
- Release a good recipe of using ResNet in object detectors pre-trained by ResNet Strikes Back, which consistently brings about 3~4 mAP improvements over RetinaNet, Faster/Mask/Cascade Mask R-CNN (#7001)
- Support Open Images Dataset (#6331)
- Support TIMM backbone: PyTorch Image Models (#7020)

22.4.2 New Features

- Support MaskFormer (#7212)
- Support DyHead (#6823)
- Support ResNet Strikes Back (#7001)
- Support OpenImages Dataset (#6331)
- Support TIMM backbone (#7020)
- Support visualization for Panoptic Segmentation (#7041)

22.4.3 Breaking Changes

In order to support the visualization for Panoptic Segmentation, the `num_classes` can not be `None` when using the `get_palette` function to determine whether to use the panoptic palette.

22.4.4 Bug Fixes

- Fix bug for the best checkpoints can not be saved when the `key_score` is None (#7101)
- Fix MixUp transform filter boxes failing case (#7080)
- Add missing properties in SABLHead (#7091)
- Fix bug when NaNs exist in confusion matrix (#7147)
- Fix PALETTE AttributeError in downstream task (#7230)

22.4.5 Improvements

- Speed up SimOTA matching (#7098)
- Add Chinese translation of `docs_zh-CN/tutorials/init_cfg.md` (#7188)

22.4.6 Contributors

A total of 20 developers contributed to this release. Thanks @ZwwWayne, @hhaAndroid, @RangiLyu, @Aron-Lin, @BIGWangYuDong, @jbwang1997, @zytx121, @chhluo, @shinya7y, @LuooChen, @dvansa, @siatwang-min, @del-zhenwu, @vikashranjan26, @haofanwang, @jamiechoi1995, @HJoonKwon, @yarkable, @zhijian-liu, @RangeKing

22.5 v2.21.0 (8/2/2022)

22.6 Breaking Changes

To standardize the contents in config READMEs and meta files of OpenMMLab projects, the READMEs and meta files in each config directory have been significantly changed. The template will be released in the future, for now, you can refer to the examples of README for algorithm, dataset and backbone. To align with the standard, the configs in dcn are put into to two directories named `dcn` and `dcnv2`.

22.6.1 New Features

- Allow to customize colors of different classes during visualization (#6716)
- Support CPU training (#7016)
- Add download script of COCO, LVIS, and VOC dataset (#7015)

22.6.2 Bug Fixes

- Fix weight conversion issue of RetinaNet with Swin-S (#6973)
- Update `__repr__` of `Compose` (#6951)
- Fix BadZipFile Error when build docker (#6966)
- Fix bug in non-distributed multi-gpu training/testing (#7019)
- Fix bbox clamp in PyTorch 1.10 (#7074)
- Relax the requirement of PALETTE in dataset wrappers (#7085)
- Keep the same weights before reassign in the PAA head (#7032)
- Update code demo in doc (#7092)

22.6.3 Improvements

- Speed-up training by allow to set variables of multi-processing (#6974, #7036)
- Add links of Chinese tutorials in readme (#6897)
- Disable cv2 multiprocessing by default for acceleration (#6867)
- Deprecate the support for "python setup.py test" (#6998)
- Re-organize metafiles and config readmes (#7051)
- Fix None grad problem during training TOOD by adding `SigmoidGeometricMean` (#7090)

22.6.4 Contributors

A total of 26 developers contributed to this release. Thanks @del-zhenwu, @zimoqingfeng, @srishilesh, @imyhxy, @jenhaoyang, @jliu-ac, @kimnamu, @ShengliLiu, @garvan2021, @ciusji, @DIYer22, @kimnamu, @q3394101, @zhouzaida, @gaotongxiao, @topsy404, @AntoAndGar, @jbwang1997, @nijkah, @ZwwWayne, @Czm369, @jshilong, @RangiLyu, @BIGWangYuDong, @hhaAndroid, @AronLin

22.7 v2.20.0 (27/12/2021)

22.7.1 New Features

- Support TOOD: Task-aligned One-stage Object Detection (ICCV 2021 Oral) (#6746)
- Support resuming from the latest checkpoint automatically (#6727)

22.7.2 Bug Fixes

- Fix wrong bbox `loss_weight` of the PAA head (#6744)
- Fix the padding value of `gt_semantic_seg` in batch collating (#6837)
- Fix test error of lvis when using `classwise` (#6845)
- Avoid BC-breaking of `get_local_path` (#6719)
- Fix bug in `sync_norm_hook` when the BN layer does not exist (#6852)
- Use pycocotools directly no matter what platform it is (#6838)

22.7.3 Improvements

- Add unit test for SimOTA with no valid bbox (#6770)
- Use precommit to check readme (#6802)
- Support selecting GPU-ids in non-distributed testing time (#6781)

22.7.4 Contributors

A total of 16 developers contributed to this release. Thanks @ZwwWayne, @Czm369, @jshilong, @RangiLyu, @BIG-WangYuDong, @hhaAndroid, @jamiechoi1995, @AronLin, @Keiku, @gkagkos, @fcakyon, @www516717402, @vansin, @zactodd, @kimnamu, @jenhaoyang

22.8 v2.19.1 (14/12/2021)

22.8.1 New Features

- Release YOLOX COCO pretrained models (#6698)

22.8.2 Bug Fixes

- Fix DCN initialization in DenseHead (#6625)
- Fix initialization of ConvFCHead (#6624)
- Fix PseudoSampler in RCNN (#6622)
- Fix weight initialization in Swin and PVT (#6663)
- Fix dtype bug in BaseDenseHead (#6767)
- Fix SimOTA with no valid bbox (#6733)

22.8.3 Improvements

- Add an example of combining swin and one-stage models (#6621)
- Add `get_ann_info` to dataset_wrappers (#6526)
- Support keeping image ratio in the multi-scale training of YOLOX (#6732)
- Support `bbox_clip_border` for the augmentations of YOLOX (#6730)

22.8.4 Documents

- Update metafile (#6717)
- Add mmhuman3d in readme (#6699)
- Update FAQ docs (#6587)
- Add doc for `detect_anomalous_params` (#6697)

22.8.5 Contributors

A total of 11 developers contributed to this release. Thanks @ZwwWayne, @LJoson, @Czm369, @jshilong, @ZCMax, @RangiLyu, @BIGWangYuDong, @hhaAndroid, @zhaoxin111, @GT9505, @shinya7y

22.9 v2.19.0 (29/11/2021)

22.9.1 Highlights

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

22.9.2 New Features

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

22.9.3 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)

22.9.4 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)

22.9.5 Documents

- Update README.md (#6567)

22.9.6 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

22.10 v2.18.1 (15/11/2021)

22.10.1 Highlights

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

22.10.2 New Features

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

22.10.3 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)

22.10.4 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)

22.10.5 Documents

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

22.10.6 Contributors

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

22.11 v2.18.0 (27/10/2021)

22.11.1 Highlights

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

22.11.2 New Features

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

22.11.3 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)

22.11.4 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)

22.11.5 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)

22.11.6 Contributors

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

22.12 v2.17.0 (28/9/2021)

22.12.1 Highlights

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

22.12.2 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)

22.12.3 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)

22.12.4 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)

22.12.5 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

22.13 v2.16.0 (30/8/2021)

22.13.1 Highlights

- Support Panoptic FPN and Swin Transformer

22.13.2 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)

22.13.3 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)

22.13.4 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)

22.13.5 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

22.14 v2.15.1 (11/8/2021)

22.14.1 Highlights

- Support YOLOX

22.14.2 New Features

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

22.14.3 Bug Fixes

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

22.14.4 Improvements

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

22.14.5 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

22.15 v2.15.0 (02/8/2021)

22.15.1 Highlights

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

22.15.2 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)

22.15.3 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)

22.15.4 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)

22.15.5 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

22.16 v2.14.0 (29/6/2021)

22.16.1 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

22.16.2 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)

22.16.3 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)

22.16.4 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)

22.17 v2.13.0 (01/6/2021)

22.17.1 Highlights

- Support new methods: CenterNet, Seesaw Loss, MobileNetV2

22.17.2 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)

22.17.3 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)

22.17.4 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)

22.18 v2.12.0 (01/5/2021)

22.18.1 Highlights

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

22.18.2 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.

22.18.3 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)

22.18.4 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)

22.18.5 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)

22.19 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)

22.20 v2.10.0 (01/03/2021)

22.20.1 Highlights

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

22.20.2 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)

22.20.3 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)

22.20.4 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)

22.21 v2.9.0 (01/02/2021)

22.21.1 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

22.21.2 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)

22.21.3 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)

22.21.4 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)

22.22 v2.8.0 (04/01/2021)

22.22.1 Highlights

- Support new methods: Cascade RPN, TridentNet

22.22.2 New Features

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

22.22.3 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)

22.22.4 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)

22.23 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.

22.23.1 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)

22.23.2 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)

22.23.3 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)

22.24 v2.6.0 (1/11/2020)

- Support new method: VarifocalNet.
- Refactored documentation with more tutorials.

22.24.1 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)

22.24.2 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)

22.24.3 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)

22.25 v2.5.0 (5/10/2020)

22.25.1 Highlights

- Support new methods: YOLACT, CentripetalNet.
- Add more documentations for easier and more clear usage.

22.25.2 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.

22.25.3 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)

22.25.4 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)

22.25.5 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)

22.26 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)

22.27 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)

22.28 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)

22.29 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)

22.30 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)

22.31 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)

22.32 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)

22.33 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)

22.34 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.

22.35 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.

22.36 v0.6rc0(06/02/2019)

- Migrate to PyTorch 1.0.

22.37 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.

22.38 v0.5.6 (17/01/2019)

- Add support for Group Normalization.
- Unify RPNHead and single stage heads (RetinaHead, SSDHead) with AnchorHead.

22.39 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.

22.40 v0.5.4 (27/11/2018)

- Add SingleStageDetector and RetinaNet.

22.41 v0.5.3 (26/11/2018)

- Add Cascade R-CNN and Cascade Mask R-CNN.
- Add support for Soft-NMS in config files.

22.42 v0.5.2 (21/10/2018)

- Add support for custom datasets.
- Add a script to convert PASCAL VOC annotations to the expected format.

22.43 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

TWENTYTHREE

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.

23.1 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."

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

Note:

1. If you want to install mmdet-v2.x, the compatible MMDetection and MMCV versions table can be found at here. Please choose the correct version of MMCV to avoid installation issues.
2. In MMCV-v2.x, `mmcv-full` is rename to `mmcv`, if you want to install `mmcv` without CUDA ops, you can install `mmcv-lite`.

- "No module named 'mmcv.ops'"; "No module named 'mmcv._ext'".

  1. Uninstall existing `mmcv-lite` in the environment using `pip uninstall mmcv-lite`.
  2. Install `mmcv` following the installation instruction.

- Using Albumentations

  If you would like to use `albumentations`, we suggest using `pip install -r requirements/albu.txt` or `pip install -U albumentations --no-binary qudida,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`). Please refer to the official documentation for details.

- ModuleNotFoundError is raised when using some algorithms

  Some extra dependencies are required for Instaboost, Panoptic Segmentation, LVIS dataset, etc. Please note the error message and install corresponding packages, e.g.,

```
# 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
```

23.2 Coding

- Do I need to reinstall mmdet after some code modifications

  If you follow the best practice and install mmdet with `pip install -e .`, any local modifications made to the code will take effect without reinstallation.

- How to develop with multiple MMDetection versions

  You can have multiple folders like mmdet-3.0, mmdet-3.1. When you run the train or test script, it will adopt the mmdet package in the current folder.

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

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

23.3 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` 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.

23.4 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 `optim_wrapper=dict(clip_grad=dict(max_norm=35, norm_type=2))` in your config file.

- "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.
4. Try to use `AvoidCUDAOOM` to avoid GPU out of memory. It will first retry after calling `torch.cuda.empty_cache()`. If it still fails, it will then retry by converting the type of inputs to FP16 format. If it still fails, it will try to copy inputs from GPUs to CPUs to continue computing. Try AvoidOOM in you code to make the code continue to run when GPU memory runs out:

```
from mmdet.utils import AvoidCUDAOOM

output = AvoidCUDAOOM.retry_if_cuda_oom(some_function)(input1, input2)
```

You can also try `AvoidCUDAOOM` as a decorator to make the code continue to run when GPU memory runs out:

```
from mmdet.utils import AvoidCUDAOOM

@AvoidCUDAOOM.retry_if_cuda_oom
def function(*args, **kwargs):
    ...
    return xxx
```

- "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, but this will slow down the training speed.
  3. You can set `detect_anomalous_params = True` in the config or `model_wrapper_cfg = dict(type='MMDistributedDataParallel', detect_anomalous_params=True)` (More details please refer to MMEngine) to get the name of those unused parameters. Note `detect_anomalous_params = True` will slow down the training speed, so it is recommended for debugging only.
- Save the best model

  It can be turned on by configuring `default_hooks = dict(checkpoint=dict(type='CheckpointHook', interval=1, save_best='auto'),`. In the case of the `auto` parameter, the first key in the returned evaluation result will be used as the basis for selecting the best model. You can also directly set the key in the evaluation result to manually set it, for example, `save_best='mAP'`.

23.5 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.

23.6 Model

- `style` in ResNet

  The `style` parameter in ResNet allows either `pytorch` or `caffe` style. It indicates the difference in the Bottleneck module. Bottleneck is a stacking structure of `1x1-3x3-1x1` convolutional layers. In the case of `caffe` mode, the convolution layer with `stride=2` is the first `1x1` convolution, while in `pyorch` mode, it is the second `3x3` convolution has `stride=2`. A sample code is as below:

```
if self.style == 'pytorch':
      self.conv1_stride = 1
      self.conv2_stride = stride
else:
      self.conv1_stride = stride
      self.conv2_stride = 1
```

- ResNeXt parameter description

  ResNeXt comes from the paper `Aggregated Residual Transformations for Deep Neural Networks`. It introduces group and uses "cardinality" to control the number of groups to achieve a balance between accuracy and complexity. It controls the basic width and grouping parameters of the internal Bottleneck module through two hyperparameters `baseWidth` and `cardinality`. An example configuration name in MMDetection is `mask_rcnn_x101_64x4d_fpn_mstrain-poly_3x_coco.py`, where `mask_rcnn` represents the algorithm using Mask R-CNN, `x101` represents the backbone network using ResNeXt-101, and `64x4d` represents that the bottleneck block has 64 group and each group has basic width of 4.

- `norm_eval` in backbone

  Since the detection model is usually large and the input image resolution is high, this will result in a small batch of the detection model, which will make the variance of the statistics calculated by BatchNorm during the training process very large and not as stable as the statistics obtained during the pre-training of the backbone network . Therefore, the `norm_eval=True` mode is generally used in training, and the BatchNorm statistics in the pre-trained backbone network are directly used. The few algorithms that use large batches are the `norm_eval=False` mode, such as NASFPN. For the backbone network without ImageNet pre-training and the batch is relatively small, you can consider using `SyncBN`.

CHAPTER

TWENTYFOUR

COMPATIBILITY OF MMDETECTION 2.X

24.1 MMDetection 2.25.0

In order to support Mask2Former for instance segmentation, the original config files of Mask2Former for panpotic segmentation need to be renamed PR #7571.

```
'mask2former_xxx_coco.py' represents config files for **panoptic segmentation**.
```

```
'mask2former_xxx_coco.py' represents config files for **instance segmentation**.
'mask2former_xxx_coco-panoptic.py' represents config files for **panoptic segmentation**.
```

24.2 MMDetection 2.21.0

In order to support CPU training, the logic of scatter in batch collating has been changed. We recommend to use MMCV v1.4.4 or higher. For more details, please refer to MMCV PR #1621.

24.3 MMDetection 2.18.1

24.3.1 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.

24.4 MMDetection 2.18.0

24.4.1 DIIHead compatibility

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

24.5 MMDetection 2.14.0

24.5.1 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.

24.5.2 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.

24.6 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.

24.6.1 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.

24.6.2 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.

24.6.3 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.

24.6.4 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.

24.7 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.

24.7.1 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.

24.7.2 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.

24.7.3 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.

24.7.4 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`.

24.8 pycocotools compatibility

`mmpycocotools` is the OpenMMlab's fork 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 `pycocotools` (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

TWENTYFIVE

ENGLISH

CHAPTER

TWENTYSIX

CHAPTER

TWENTYSEVEN

INDICES AND TABLES

- genindex
- search