MMDetection

发布 *2.15.1*

MMDetection Authors

2021 年 10 月 14 日

开始你的第一步

1 依赖 **1**

2 安装流程 **3**
- 2.1 准备环境 . . . 3
- 2.2 安装 MMDetection . . . 4
- 2.3 只在 CPU 安装 . . . 5
- 2.4 另一种选择：Docker 镜像 . . . 6
- 2.5 从零开始设置脚本 . . . 6
- 2.6 使用多个 MMDetection 版本进行开发 . . . 7

3 验证 **9**

4 模型库 **11**
- 4.1 镜像地址 . . . 11

5 1: 使用已有模型在标准数据集上进行推理 **13**
- 5.1 使用现有模型进行推理 . . . 13
- 5.2 在标准数据集上测试现有模型 . . . 17
- 5.3 在标准数据集上训练预定义的模型 . . . 23

6 2: 在自定义数据集上进行训练 **27**
- 6.1 准备自定义数据集 . . . 27
- 6.2 准备配置文件 . . . 32
- 6.3 训练一个新的模型 . . . 33
- 6.4 测试以及推理 . . . 33

7 教程 1: 学习配置文件 **35**
- 7.1 通过脚本参数修改配置 . . . 35
- 7.2 配置文件结构 . . . 36

7.3 配置文件名称风格 . . . 36
7.4 弃用的 train_cfg/test_cfg . . . 37
7.5 Mask R-CNN 配置文件示例 . . . 37
7.6 常问问题 (FAQ) . . . 46

8 教程 2: 自定义数据集 51

9 教程 3: 自定义数据预处理流程 53

10 教程 4: 自定义模型 55

11 教程 5: 自定义训练配置 57

12 教程 6: 自定义损失函数 59

13 教程 7: 模型微调 61

14 教程 8: Pytorch 到 ONNX 的模型转换（实验性支持） 63

15 教程 9: ONNX 到 TensorRT 的模型转换（实验性支持） 65

16 日志分析 67

17 默认约定 69

18 MMDetection v2.x 兼容性说明 71
18.1 MMDetection v2.14.0 . . . 71
18.2 MMDetection v2.12.0 . . . 72
18.3 与 MMDetection v1.x 的兼容性 . . . 73
18.4 pycocotools 兼容性 . . . 75

19 常见问题解答 77
19.1 MMCV 安装相关 . . . 77
19.2 PyTorch/CUDA 环境相关 . . . 78
19.3 Training 相关 . . . 79
19.4 Evaluation 相关 . . . 80

20 English 81

21 简体中文 83

22 API Reference 85
22.1 mmdet.apis . . . 85
22.2 mmdet.core . . . 85
22.3 mmdet.datasets . . . 131
22.4 mmdet.models . . . 132
22.5 mmdet.utils . . . 175

23 Indices and tables 177

Python 模块索引 179

索引 181

CHAPTER 1

依赖

- Linux 和 macOS （Windows 理论上支持）
- Python 3.6+
- PyTorch 1.3+
- CUDA 9.2+ （如果基于 PyTorch 源码安装，也能够支持 CUDA 9.0）
- GCC 5+
- MMCV

MMDetection 和 MMCV 版本兼容性如下所示，需要安装正确的 MMCV 版本以避免安装出现问题。

** 注意：** 如果已经安装了 mmcv，首先需要使用 `pip uninstall mmcv` 卸载已安装的 mmcv，如果同时安装了 mmcv 和 mmcv-full，将会报 `ModuleNotFoundError` 错误。

CHAPTER 2

安装流程

2.1 准备环境

1. 使用 conda 新建虚拟环境，并进入该虚拟环境；

```
conda create -n open-mmlab python=3.7 -y
conda activate open-mmlab
```

2. 基于 PyTorch 官网安装 PyTorch 和 torchvision，例如：

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

注意：需要确保 CUDA 的编译版本和运行版本匹配。可以在 PyTorch 官网查看预编译包所支持的 CUDA 版本。

例 1 例如在 `/usr/local/cuda` 下安装了 CUDA 10.1，并想安装 PyTorch 1.5，则需要安装支持 CUDA 10.1 的预构建 PyTorch：

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

例 2 例如在 `/usr/local/cuda` 下安装了 CUDA 9.2，并想安装 PyTorch 1.3.1，则需要安装支持 CUDA 9.2 的预构建 PyTorch：

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

如果不是安装预构建的包，而是从源码中构建 PyTorch，则可以使用更多的 CUDA 版本，例如 CUDA 9.0。

2.2 安装 MMDetection

我们建议使用 MIM 来安装 MMDetection：

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

MIM 能够自动地安装 OpenMMLab 的项目以及对应的依赖包。

或者，可以手动安装 MMDetection：

1. 安装 mmcv-full，我们建议使用预构建包来安装：

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

   需要把命令行中的 {cu_version} 和 {torch_version} 替换成对应的版本。例如：在 CUDA 11 和 PyTorch 1.7.0 的环境下，可以使用下面命令安装最新版本的 MMCV：

   ```
   pip install mmcv-full -f https://download.openmmlab.com/mmcv/dist/cu110/torch1.7.
   ↪0/index.html
   ```

   请参考 MMCV 获取不同版本的 MMCV 所兼容的的不同的 PyTorch 和 CUDA 版本。同时，也可以通过以下命令行从源码编译 MMCV：

   ```
   git clone https://github.com/open-mmlab/mmcv.git
   cd mmcv
   MMCV_WITH_OPS=1 pip install -e .  # 安装好 mmcv-full
   cd ..
   ```

   或者，可以直接使用命令行安装：

   ```
   pip install mmcv-full
   ```

2. 安装 MMDetection：

   你可以直接通过如下命令从 pip 安装使用 mmdetection:

   ```
   pip install mmdet
   ```

   或者从 git 仓库编译源码

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

3. 安装额外的依赖以使用 Instaboost, 全景分割, 或者 LVIS 数据集

```
# 安装 instaboost 依赖
pip install instaboostfast
# 安装全景分割依赖
pip install git+https://github.com/cocodataset/panopticapi.git
# 安装 LVIS 数据集依赖
pip install git+https://github.com/lvis-dataset/lvis-api.git
```

注意：

(1) 按照上述说明，MMDetection 安装在 `dev` 模式下，因此在本地对代码做的任何修改都会生效，无需重新安装；

(2) 如果希望使用 `opencv-python-headless` 而不是 `opencv-python`，可以在安装 MMCV 之前安装；

(3) 一些安装依赖是可以选择的。例如只需要安装最低运行要求的版本，则可以使用 `pip install -v -e .` 命令。如果希望使用可选择的像 `albumentations` 和 `imagecorruptions` 这种依赖项，可以使用 `pip install -r requirements/optional.txt` 进行手动安装，或者在使用 `pip` 时指定所需的附加功能（例如 `pip install -v -e .[optional]`），支持附加功能的有效键值包括 `all`、`tests`、`build` 以及 `optional` 。

2.3 只在 CPU 安装

我们的代码能够建立在只使用 CPU 的环境（CUDA 不可用）。

在 CPU 模式下，可以运行 `demo/webcam_demo.py` 示例，然而以下功能将在 CPU 模式下不能使用：

- Deformable Convolution
- Modulated Deformable Convolution
- ROI pooling
- Deformable ROI pooling
- CARAFE: Content-Aware ReAssembly of FEatures
- SyncBatchNorm
- CrissCrossAttention: Criss-Cross Attention
- MaskedConv2d

- Temporal Interlace Shift
- nms_cuda
- sigmoid_focal_loss_cuda
- bbox_overlaps

因此，如果尝试使用包含上述操作的模型进行推理，将会报错。下表列出了由于依赖上述算子而无法在 CPU 上运行的相关模型：

注意：MMDetection 目前不支持使用 CPU 进行训练。

2.4 另一种选择：Docker 镜像

我们提供了 Dockerfile 来生成镜像，请确保 docker 的版本 >= 19.03。

```
# 基于 PyTorch 1.6, CUDA 10.1 生成镜像
docker build -t mmdetection docker/
```

运行命令：

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

2.5 从零开始设置脚本

假设当前已经成功安装 CUDA 10.1，这里提供了一个完整的基于 conda 安装 MMDetection 的脚本：

```
conda create -n open-mmlab python=3.7 -y
conda activate open-mmlab

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

# 安装最新版本的 mmcv
pip install mmcv-full -f https://download.openmmlab.com/mmcv/dist/cu101/torch1.6.0/
↪index.html

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

2.6 使用多个 MMDetection 版本进行开发

训练和测试的脚本已经在 PYTHONPATH 中进行了修改，以确保脚本使用当前目录中的 MMDetection。

要使环境中安装默认的 MMDetection 而不是当前正在在使用的，可以删除出现在相关脚本中的代码：

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

CHAPTER 3

验证

为了验证是否正确安装了 MMDetection 和所需的环境，我们可以运行示例的 Python 代码来初始化检测器并推理一个演示图像：

```
from mmdet.apis import init_detector, inference_detector

config_file = 'configs/faster_rcnn/faster_rcnn_r50_fpn_1x_coco.py'
# 从 model zoo 下载 checkpoint 并放在 `checkpoints/` 文件下
# 网址为: http://download.openmmlab.com/mmdetection/v2.0/faster_rcnn/faster_rcnn_r50_
↪fpn_1x_coco/faster_rcnn_r50_fpn_1x_coco_20200130-047c8118.pth
checkpoint_file = 'checkpoints/faster_rcnn_r50_fpn_1x_coco_20200130-047c8118.pth'
device = 'cuda:0'
# 初始化检测器
model = init_detector(config_file, checkpoint_file, device=device)
# 推理演示图像
inference_detector(model, 'demo/demo.jpg')
```

如果成功安装 MMDetection，则上面的代码可以完整地运行。

CHAPTER 4

模型库

4.1 镜像地址

CHAPTER 5

1: 使用已有模型在标准数据集上进行推理

MMDetection 在 Model Zoo 中提供了数以百计的检测模型，并支持多种标准数据集，包括 Pascal VOC，COCO，Cityscapes，LVIS 等。这份文档将会讲述如何使用这些模型和标准数据集来运行一些常见的任务，包括：

- 使用现有模型在给定图片上进行推理
- 在标准数据集上测试现有模型
- 在标准数据集上训练预定义的模型

5.1 使用现有模型进行推理

推理是指使用训练好的模型来检测图像上的目标。在 MMDetection 中，一个模型被定义为一个配置文件和对应的存储在 checkpoint 文件内的模型参数的集合。

首先，我们建议从 Faster RCNN 开始，其 配置 文件和 checkpoint 文件在此。我们建议将 checkpoint 文件下载到 `checkpoints` 文件夹内。

5.1.1 推理的高层编程接口

MMDetection 为在图片上推理提供了 Python 的高层编程接口。下面是建立模型和在图像或视频上进行推理的例子。

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

# 指定模型的配置文件和 checkpoint 文件路径
config_file = 'configs/faster_rcnn/faster_rcnn_r50_fpn_1x_coco.py'
checkpoint_file = 'checkpoints/faster_rcnn_r50_fpn_1x_coco_20200130-047c8118.pth'

# 根据配置文件和 checkpoint 文件构建模型
model = init_detector(config_file, checkpoint_file, device='cuda:0')

# 测试单张图片并展示结果
img = 'test.jpg'  # 或者 img = mmcv.imread(img)，这样图片仅会被读一次
result = inference_detector(model, img)
# 在一个新的窗口中将结果可视化
model.show_result(img, result)
# 或者将可视化结果保存为图片
model.show_result(img, result, out_file='result.jpg')

# 测试视频并展示结果
video = mmcv.VideoReader('video.mp4')
for frame in video:
    result = inference_detector(model, frame)
    model.show_result(frame, result, wait_time=1)
```

jupyter notebook 上的演示样例在 demo/inference_demo.ipynb 。

5.1.2 异步接口-支持 Python 3.7+

对于 Python 3.7+，MMDetection 也有异步接口。利用 CUDA 流，绑定 GPU 的推理代码不会阻塞 CPU，从而使得 CPU/GPU 在单线程应用中能达到更高的利用率。在推理流程中，不同数据样本的推理和不同模型的推理都能并发地运行。

您可以参考 `tests/async_benchmark.py` 来对比同步接口和异步接口的运行速度。

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

```

(下页继续)

(续上页)

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

    # 此队列用于并行推理多张图像
    streamqueue = asyncio.Queue()
    # 队列大小定义了并行的数量
    streamqueue_size = 3

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

    # 测试单张图片并展示结果
    img = 'test.jpg'  # or 或者 img = mmcv.imread(img)，这样图片仅会被读一次

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

    # 在一个新的窗口中将结果可视化
    model.show_result(img, result)
    # 或者将可视化结果保存为图片
    model.show_result(img, result, out_file='result.jpg')


asyncio.run(main())
```

5.1.3 演示样例

我们还提供了三个演示脚本，它们是使用高层编程接口实现的。源码在此 。

图片样例

这是在单张图片上进行推理的脚本，可以开启 `--async-test` 来进行异步推理。

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

(下页继续)

(续上页)

```
    [--score-thr ${SCORE_THR}] \
    [--async-test]
```

运行样例：

```
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
```

摄像头样例

这是使用摄像头实时图片的推理脚本。

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

运行样例：

```
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
```

视频样例

这是在视频样例上进行推理的脚本。

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

运行样例：

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

5.2 在标准数据集上测试现有模型

为了测试一个模型的精度，我们通常会在标准数据集上对其进行测试。MMDetection 支持多个公共数据集，包括 COCO ，Pascal VOC ，Cityscapes 等等。这一部分将会介绍如何在支持的数据集上测试现有模型。

5.2.1 数据集准备

一些公共数据集，比如 Pascal VOC 及其镜像数据集，或者 COCO 等数据集都可以从官方网站或者镜像网站获取。注意：在检测任务中，Pascal VOC 2012 是 Pascal VOC 2007 的无交集扩展，我们通常将两者一起使用。我们建议将数据集下载，然后解压到项目外部的某个文件夹内，然后通过符号链接的方式，将数据集根目录链接到 `$MMDETECTION/data` 文件夹下，格式如下所示。如果你的文件夹结构和下方不同的话，你需要在配置文件中改变对应的路径。

```
mmdetection
├── mmdet
├── tools
├── configs
├── data
│   ├── coco
│   │   ├── annotations
│   │   ├── train2017
│   │   ├── val2017
│   │   ├── test2017
│   ├── cityscapes
│   │   ├── annotations
│   │   ├── leftImg8bit
│   │   │   ├── train
│   │   │   ├── val
│   │   ├── gtFine
│   │   │   ├── train
│   │   │   ├── val
│   ├── VOCdevkit
│   │   ├── VOC2007
│   │   ├── VOC2012
```

有些模型需要额外的 COCO-stuff 数据集，比如 HTC，DetectoRS 和 SCNet，你可以下载并解压它们到 `coco` 文件夹下。文件夹会是如下结构：

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

Cityscape 数据集的标注格式需要转换，以与 COCO 数据集标注格式保持一致，使用 `tools/dataset_converters/cityscapes.py` 来完成转换：

```
pip install cityscapesscripts

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

5.2.2 测试现有模型

我们提供了测试脚本，能够测试一个现有模型在所有数据集（COCO，Pascal VOC，Cityscapes 等）上的性能。我们支持在如下环境下测试：

- 单 GPU 测试
- 单节点多 GPU 测试
- 多节点测试

根据以上测试环境，选择合适的脚本来执行测试过程。

```
# 单 GPU 测试
python tools/test.py \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    [--out ${RESULT_FILE}] \
    [--eval ${EVAL_METRICS}] \
    [--show]

# 单节点多 GPU 测试
bash tools/dist_test.sh \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    ${GPU_NUM} \
```

(下页继续)

(续上页)

```
    [--out ${RESULT_FILE}] \
    [--eval ${EVAL_METRICS}]
```

tools/dist_test.sh 也支持多节点测试，不过需要依赖 PyTorch 的 启动工具 。

可选参数:

- RESULT_FILE: 结果文件名称，需以.pkl 形式存储。如果没有声明，则不将结果存储到文件。
- EVAL_METRICS: 需要测试的度量指标。可选值是取决于数据集的，比如 proposal_fast，proposal，bbox，segm 是 COCO 数据集的可选值，mAP，recall 是 Pascal VOC 数据集的可选值。Cityscapes 数据集可以测试 cityscapes 和所有 COCO 数据集支持的度量指标。
- --show: 如果开启，检测结果将被绘制在图像上，以一个新窗口的形式展示。它只适用于单 GPU 的测试，是用于调试和可视化的。请确保使用此功能时，你的 GUI 可以在环境中打开。否则，你可能会遇到这么一个错误 cannot connect to X server。
- --show-dir: 如果指明，检测结果将会被绘制在图像上并保存到指定目录。它只适用于单 GPU 的测试，是用于调试和可视化的。即使你的环境中没有 GUI，这个选项也可使用。
- --show-score-thr: 如果指明，得分低于此阈值的检测结果将会被移除。
- --cfg-options: 如果指明，这里的键值对将会被合并到配置文件中。
- --eval-options: 如果指明，这里的键值对将会作为字典参数被传入 dataset.evalutation() 函数中，仅在测试阶段使用。

5.2.3 样例

假设你已经下载了 checkpoint 文件到 checkpoints/ 文件下了。

1. 测试 Faster R-CNN 并可视化其结果。按任意键继续下张图片的测试。配置文件和 checkpoint 文件 在此 。

```
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. 测试 Faster R-CNN，并为了之后的可视化保存绘制的图像。配置文件和 checkpoint 文件 在此 。

```
python tools/test.py \
    configs/faster_rcnn/faster_rcnn_r50_fpn_1x.py \
    checkpoints/faster_rcnn_r50_fpn_1x_coco_20200130-047c8118.pth \
    --show-dir faster_rcnn_r50_fpn_1x_results
```

3. 在 Pascal VOC 数据集上测试 Faster R-CNN，不保存测试结果，测试 mAP。配置文件和 checkpoint 文件 在此 。

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

4. 使用 8 块 GPU 测试 Mask R-CNN，测试 bbox 和 mAP 。配置文件和 checkpoint 文件 在此 。

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

5. 使用 8 块 GPU 测试 Mask R-CNN，测试**每类**的 bbox 和 mAP。配置文件和 checkpoint 文件 在此 。

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

6. 在 COCO test-dev 数据集上，使用 8 块 GPU 测试 Mask R-CNN，并生成 JSON 文件提交到官方评测服务器。配置文件和 checkpoint 文件 在此 。

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

这行命令生成两个 JSON 文件 mask_rcnn_test-dev_results.bbox.json 和 mask_rcnn_test-dev_results.segm.json。

1. 在 Cityscapes 数据集上，使用 8 块 GPU 测试 Mask R-CNN，生成 txt 和 png 文件，并上传到官方评测服务器。配置文件和 checkpoint 文件 在此 。

```
./tools/dist_test.sh \
    configs/cityscapes/mask_rcnn_r50_fpn_1x_cityscapes.py \
    checkpoints/mask_rcnn_r50_fpn_1x_cityscapes_20200227-afe51d5a.pth \
    8 \
    --format-only \
```

(下页继续)

(续上页)

```
    --options "txtfile_prefix=./mask_rcnn_cityscapes_test_results"
```

生成的 png 和 txt 文件在 `./mask_rcnn_cityscapes_test_results` 文件夹下。

5.2.4 不使用 Ground Truth 标注进行测试

MMDetection 支持在不使用 ground-truth 标注的情况下对模型进行测试，这需要用到 `CocoDataset`。如果你的数据集格式不是 COCO 格式的，请将其转化成 COCO 格式。比如，你的数据集格式是 VOC，你可以使用 `tools` 内的脚本直接将其转化成 COCO 格式。

```
# 单 GPU 测试
python tools/test.py \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    --format-only \
    --options ${JSONFILE_PREFIX} \
    [--show]

# 单节点多 GPU 测试
bash tools/dist_test.sh \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    ${GPU_NUM} \
    --format-only \
    --options ${JSONFILE_PREFIX} \
    [--show]
```

假设 model zoo 中的 checkpoint 文件被下载到了 `checkpoints/` 文件夹下，我们可以使用以下命令，用 8 块 GPU 在 COCO test-dev 数据集上测试 Mask R-CNN，并且生成 JSON 文件。

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

这行命令生成两个 JSON 文件 `mask_rcnn_test-dev_results.bbox.json` 和 `mask_rcnn_test-dev_results.segm.json`。

5.2.5 批量推理

MMDetection 在测试模式下，既支持单张图片的推理，也支持对图像进行批量推理。默认情况下，我们使用单张图片的测试，你可以通过修改测试数据配置文件中的 `samples_per_gpu` 来开启批量测试。开启批量推理的配置文件修改方法为：

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

或者你可以通过将 `--cfg-options` 设置为 `--cfg-options data.test.samples_per_gpu=2` 来开启它。

5.2.6 弃用 ImageToTensor

在测试模式下，弃用 `ImageToTensor` 流程，取而代之的是 `DefaultFormatBundle`。建议在你的测试数据流程的配置文件中手动替换它，如：

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

# （建议使用）手动将 ImageToTensor 替换为 DefaultFormatBundle
pipelines = [
    dict(type='LoadImageFromFile'),
    dict(
        type='MultiScaleFlipAug',
        img_scale=(1333, 800),
        flip=False,
        transforms=[
            dict(type='Resize', keep_ratio=True),
            dict(type='RandomFlip'),
            dict(type='Normalize', mean=[0, 0, 0], std=[1, 1, 1]),
```

(下页继续)

(续上页)

```
            dict(type='Pad', size_divisor=32),
            dict(type='DefaultFormatBundle'),
            dict(type='Collect', keys=['img']),
        ])
    ]
```

5.3 在标准数据集上训练预定义的模型

MMDetection 也为训练检测模型提供了开盖即食的工具。本节将展示在标准数据集（比如 COCO）上如何训练一个预定义的模型。

重要信息：在配置文件中的学习率是在 8 块 GPU，每块 GPU 有 2 张图像（批大小为 8*2=16）的情况下设置的。根据 线性扩展规则 ，如果你使用不同数目的 GPU 或者每块 GPU 上有不同数量的图片，你需要设置学习率以正比于批大小，比如，在 4 块 GPU 并且每张 GPU 上有 2 张图片的情况下，设置 `lr=0.01`；在 16 块 GPU 并且每张 GPU 上有 4 张图片的情况下, 设置 `lr=0.08`。

5.3.1 数据集

训练需要准备好数据集，细节请参考*数据集准备* 。

注意：目前，`configs/cityscapes` 文件夹下的配置文件都是使用 COCO 预训练权值进行初始化的。如果网络连接不可用或者速度很慢，你可以提前下载现存的模型。否则可能在训练的开始会有错误发生。

5.3.2 使用单 GPU 训练

我们提供了 `tools/train.py` 来开启在单张 GPU 上的训练任务。基本使用如下：

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

在训练期间，日志文件和 checkpoint 文件将会被保存在工作目录下，它需要通过配置文件中的 `work_dir` 或者 CLI 参数中的 `--work-dir` 来指定。

默认情况下，模型将在每轮训练之后在 validation 集上进行测试，测试的频率可以通过设置配置文件来指定：

```
# 每 12 轮迭代进行一次测试评估
evaluation = dict(interval=12)
```

这个工具接受以下参数：

- `--no-validate` (**不建议**): 在训练期间关闭测试.

- `--work-dir ${WORK_DIR}`: 覆盖工作目录.
- `--resume-from ${CHECKPOINT_FILE}`: 从某个 checkpoint 文件继续训练.
- `--options 'Key=value'`: 覆盖使用的配置文件中的其他设置.

注意：`resume-from` 和 `load-from` 的区别：

`resume-from` 既加载了模型的权重和优化器的状态，也会继承指定 checkpoint 的迭代次数，不会重新开始训练。`load-from` 则是只加载模型的权重，它的训练是从头开始的，经常被用于微调模型。

5.3.3 在多 GPU 上训练

我们提供了 `tools/dist_train.sh` 来开启在多 GPU 上的训练。基本使用如下：

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

可选参数和上一节所说的一致。

同时启动多个任务

如果你想在一台机器上启动多个任务的话，比如在一个有 8 块 GPU 的机器上启动 2 个需要 4 块 GPU 的任务，你需要给不同的训练任务指定不同的端口（默认为 29500）来避免冲突。

如果你使用 `dist_train.sh` 来启动训练任务，你可以使用命令来设置端口。

```
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
```

在多个节点上训练

MMDetection 是依赖 `torch.distributed` 包进行分布式训练的。因此，我们可以通过 PyTorch 的 启动工具 来进行基本地使用。

使用 Slurm 来管理任务

Slurm 是一个常见的计算集群调度系统。在 Slurm 管理的集群上，你可以使用 `slurm.sh` 来开启训练任务。它既支持单节点训练也支持多节点训练。

基本使用如下：

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

以下是在一个名称为 *dev* 的 Slurm 分区上，使用 16 块 GPU 来训练 Mask R-CNN 的例子，并且将 `work-dir` 设置在了某些共享文件系统下。

```
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
```

你可以查看 源码 来检查全部的参数和环境变量.

在使用 Slurm 时，端口需要以下方的某个方法之一来设置。

1. 通过 `--options` 来设置端口。我们非常建议用这种方法，因为它无需改变原始的配置文件。

   ```
   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. 修改配置文件来设置不同的交流端口。

   在 `config1.py` 中，设置:

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

   在 `config2.py` 中，设置:

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

   然后你可以使用 `config1.py` 和 `config2.py` 来启动两个任务了。

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

CHAPTER 6

2: 在自定义数据集上进行训练

通过本文档，你将会知道如何使用自定义数据集对预先定义好的模型进行推理，测试以及训练。我们使用 balloon dataset 作为例子来描述整个过程。

基本步骤如下：

1. 准备自定义数据集
2. 准备配置文件
3. 在自定义数据集上进行训练，测试和推理。

6.1 准备自定义数据集

MMDetection 一共支持三种形式应用新数据集：

1. 将数据集重新组织为 COCO 格式。
2. 将数据集重新组织为一个中间格式。
3. 实现一个新的数据集。

我们通常建议使用前面两种方法，因为它们通常来说比第三种方法要简单。

在本文档中，我们展示一个例子来说明如何将数据转化为 COCO 格式。

注意：MMDetection 现只支持对 COCO 格式的数据集进行 mask AP 的评测。

所以用户如果要进行实例分割，只能将数据转成 COCO 格式。

6.1.1 COCO 标注格式

用于实例分割的 COCO 数据集格式如下所示，其中的键（key）都是必要的，参考这里来获取更多细节。

```
{
    "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,
    "segmentation": RLE or [polygon],
    "area": float,
    "bbox": [x,y,width,height],
    "iscrowd": 0 or 1,
}

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

现在假设我们使用 balloon dataset。

下载了数据集之后，我们需要实现一个函数将标注格式转化为 COCO 格式。然后我们就可以使用已经实现的 `COCODataset` 类来加载数据并进行训练以及评测。

如果你浏览过新数据集，你会发现格式如下：

```
{'base64_img_data': '',
 'file_attributes': {},
 'filename': '34020010494_e5cb88e1c4_k.jpg',
 'fileref': '',
 'regions': {'0': {'region_attributes': {},
```

(下页继续)

(续上页)

```
  '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,
    1174,
    1170,
    1153,
    1127,
    1104,
    1061,
    1032,
    1020],
   'all_points_y': [963,
    899,
    841,
    787,
    738,
    700,
```

(下页继续)

(续上页)

```
      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}
```

标注文件时是 JSON 格式的，其中所有键（key）组成了一张图片的所有标注。

其中将 balloon dataset 转化为 COCO 格式的代码如下所示。

```
import os.path as osp

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

(下页继续)

(续上页)

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

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

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

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


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

    coco_format_json = dict(
        images=images,
        annotations=annotations,
```

(下页继续)

(续上页)

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

使用如上的函数，用户可以成功将标注文件转化为 JSON 格式，之后可以使用 CocoDataset 对模型进行训练和评测。

6.2 准备配置文件

第二步需要准备一个配置文件来成功加载数据集。假设我们想要用 balloon dataset 来训练配备了 FPN 的 Mask R-CNN ，如下是我们的配置文件。假设配置文件命名为 mask_rcnn_r50_caffe_fpn_mstrain-poly_1x_balloon.py, 相应保存路径为 configs/ballon/ ，配置文件内容如下所示。

```
# 这个新的配置文件继承自一个原始配置文件，只需要突出必要的修改部分即可
_base_ = 'mask_rcnn/mask_rcnn_r50_caffe_fpn_mstrain-poly_1x_coco.py'

# 我们需要对头中的类别数量进行修改来匹配数据集的标注
model = dict(
    roi_head=dict(
        bbox_head=dict(num_classes=1),
        mask_head=dict(num_classes=1)))

# 修改数据集相关设置
dataset_type = 'COCODataset'
classes = ('balloon',)
data = dict(
    train=dict(
        img_prefix='balloon/train/',
        classes=classes,
        ann_file='balloon/train/annotation_coco.json'),
    val=dict(
        img_prefix='balloon/val/',
        classes=classes,
        ann_file='balloon/val/annotation_coco.json'),
    test=dict(
        img_prefix='balloon/val/',
        classes=classes,
        ann_file='balloon/val/annotation_coco.json'))

# 我们可以使用预训练的 Mask R-CNN 来获取更好的性能
load_from = 'checkpoints/mask_rcnn_r50_caffe_fpn_mstrain-poly_3x_coco_bbox_mAP-0.408__
↪segm_mAP-0.37_20200504_163245-42aa3d00.pth'
```

6.3 训练一个新的模型

为了使用新的配置方法来对模型进行训练，你只需要运行如下命令。

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

参考情况 *1*来获取更多详细的使用方法。

6.4 测试以及推理

为了测试训练完毕的模型，你只需要运行如下命令。

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

参考情况 *1*来获取更多详细的使用方法。

CHAPTER 7

教程 1: 学习配置文件

我们在配置文件中支持了继承和模块化，这便于进行各种实验。如果需要检查配置文件，可以通过运行 `python tools/misc/print_config.py /PATH/TO/CONFIG` 来查看完整的配置。

7.1 通过脚本参数修改配置

当运行 `tools/train.py` 和 `tools/test.py` 时，可以通过 `--cfg-options` 来修改配置文件。

- 更新字典链中的配置

  可以按照原始配置文件中的 dict 键顺序地指定配置预选项。例如，使用 `--cfg-options model.backbone.norm_eval=False` 将模型主干网络中的所有 BN 模块都改为 `train` 模式。

- 更新配置列表中的键

  在配置文件里，一些字典型的配置被包含在列表中。例如，数据训练流程 `data.train.pipeline` 通常是一个列表，比如 `[dict(type='LoadImageFromFile'), ...]`。如果需要将 `'LoadImageFromFile'` 改成 `'LoadImageFromWebcam'`，需要写成下述形式：`--cfg-options data.train.pipeline.0.type=LoadImageFromWebcam`。

- 更新列表或元组的值

  如果要更新的值是列表或元组。例如，配置文件通常设置 `workflow=[('train', 1)]`，如果需要改变这个键，可以通过 `--cfg-options workflow="[(train,1),(val,1)]"` 来重新设置。需要注意，引号“是支持列表或元组数据类型所必需的，并且在指定值的引号内**不允许**有空格。

7.2 配置文件结构

在 config/_base_ 文件夹下有 4 个基本组件类型，分别是：数据集 (dataset)，模型 (model)，训练策略 (schedule) 和运行时的默认设置 (default runtime)。许多方法，例如 Faster R-CNN、Mask R-CNN、Cascade R-CNN、RPN、SSD 能够很容易地构建出来。由 _base_ 下的组件组成的配置，被我们称为 *原始配置 (primitive)*。

对于同一文件夹下的所有配置，推荐**只有一个**对应的**原始配置**文件。所有其他的配置文件都应该继承自这个**原始配置**文件。这样就能保证配置文件的最大继承深度为 3。

为了便于理解，我们建议贡献者继承现有方法。例如，如果在 Faster R-CNN 的基础上做了一些修改，用户首先可以通过指定 _base_ = ../faster_rcnn/faster_rcnn_r50_fpn_1x_coco.py 来继承基础的 Faster R-CNN 结构，然后修改配置文件中的必要参数以完成继承。

如果你在构建一个与任何现有方法不共享结构的全新方法，那么可以在 configs 文件夹下创建一个新的例如 xxx_rcnn 文件夹。更多细节请参考 MMCV 文档。

7.3 配置文件名称风格

我们遵循以下样式来命名配置文件。建议贡献者遵循相同的风格。

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

{xxx} 是被要求的文件 [yyy] 是可选的。

- {model}：模型种类，例如 faster_rcnn, mask_rcnn 等。
- [model setting]：特定的模型，例如 htc 中的 without_semantic，reppoints 中的 moment 等。
- {backbone}：主干网络种类例如 r50 (ResNet-50), x101 (ResNeXt-101) 等。
- {neck}：Neck 模型的种类包括 fpn, pafpn, nasfpn, c4 等。
- [norm_setting]：默认使用 bn (Batch Normalization)，其他指定可以有 gn (Group Normalization), syncbn (Synchronized Batch Normalization) 等。gn-head/gn-neck 表示 GN 仅应用于网络的 Head 或 Neck，gn-all 表示 GN 用于整个模型，例如主干网络、Neck 和 Head。
- [misc]：模型中各式各样的设置/插件，例如 dconv、gcb、attention、albu、mstrain 等。
- [gpu x batch_per_gpu]：GPU 数量和每个 GPU 的样本数，默认使用 8x2。
- {schedule}：训练方案，选项是 1x、2x、20e 等。1x 和 2x 分别代表 12 epoch 和 24 epoch，20e 在级联模型中使用，表示 20 epoch。对于 1x/2x，初始学习率在第 8/16 和第 11/22 epoch 衰减 10 倍；对于 20e ，初始学习率在第 16 和第 19 epoch 衰减 10 倍。
- {dataset}：数据集，例如 coco、cityscapes、voc_0712、wider_face 等。

7.4 弃用的 train_cfg/test_cfg

train_cfg 和 test_cfg 在配置文件中已弃用，请在模型配置中指定它们。原始配置结构如下：

```
# 已经弃用的形式
model = dict(
   type=...,
   ...
)
train_cfg=dict(...)
test_cfg=dict(...)
```

推荐的配置结构如下：

```
# 推荐的形式
model = dict(
   type=...,
   ...
   train_cfg=dict(...),
   test_cfg=dict(...),
)
```

7.5 Mask R-CNN 配置文件示例

为了帮助用户对 MMDetection 检测系统中的完整配置和模块有一个基本的了解，我们对使用 ResNet50 和 FPN 的 Mask R-CNN 的配置文件进行简要注释说明。更详细的用法和各个模块对应的替代方案，请参考 API 文档。

```
model = dict(
    type='MaskRCNN',  # 检测器(detector)名称
    backbone=dict(  # 主干网络的配置文件
        type='ResNet',  # 主干网络的类别，可用选项请参考 https://github.com/open-
↪mmlab/mmdetection/blob/master/mmdet/models/backbones/resnet.py#L308
        depth=50,  # 主干网络的深度，对于 ResNet 和 ResNext 通常设置为 50 或 101。
        num_stages=4,  # 主干网络状态(stages)的数目，这些状态产生的特征图作为后续的␣
↪head 的输入。
        out_indices=(0, 1, 2, 3),  # 每个状态产生的特征图输出的索引。
        frozen_stages=1,  # 第一个状态的权重被冻结
        norm_cfg=dict(  # 归一化层(norm layer)的配置项。
            type='BN',  # 归一化层的类别，通常是 BN 或 GN。
            requires_grad=True),  # 是否训练归一化里的 gamma 和 beta。
        norm_eval=True,  # 是否冻结 BN 里的统计项。
        style='pytorch',  # 主干网络的风格，'pytorch' 意思是步长为2的层为 3x3 卷积，
↪'caffe' 意思是步长为2的层为 1x1 卷积。
```

(下页继续)

(续上页)

```
        init_cfg=dict(type='Pretrained', checkpoint='torchvision://resnet50')),  # 加载通过 ImageNet 与训练的模型
    neck=dict(
        type='FPN',  # 检测器的 neck 是 FPN，我们同样支持 'NASFPN', 'PAFPN' 等，更多细节可以参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/models/necks/fpn.py#L10。
        in_channels=[256, 512, 1024, 2048],  # 输入通道数，这与主干网络的输出通道一致
        out_channels=256,  # 金字塔特征图每一层的输出通道
        num_outs=5),  # 输出的范围(scales)
    rpn_head=dict(
        type='RPNHead',  # RPN_head 的类型是 'RPNHead'，我们也支持 'GARPNHead' 等，更多细节可以参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/models/dense_heads/rpn_head.py#L12。
        in_channels=256,  # 每个输入特征图的输入通道，这与 neck 的输出通道一致。
        feat_channels=256,  # head 卷积层的特征通道。
        anchor_generator=dict(  # 锚点(Anchor)生成器的配置。
            type='AnchorGenerator',  # 大多是方法使用 AnchorGenerator 作为锚点生成器，SSD 检测器使用 `SSDAnchorGenerator`。更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/core/anchor/anchor_generator.py#L10。
            scales=[8],  # 锚点的基本比例，特征图某一位置的锚点面积为 scale * base_sizes
            ratios=[0.5, 1.0, 2.0],  # 高度和宽度之间的比率。
            strides=[4, 8, 16, 32, 64]),  # 锚生成器的步幅。这与 FPN 特征步幅一致。如果未设置 base_sizes，则当前步幅值将被视为 base_sizes。
        bbox_coder=dict(  # 在训练和测试期间对框进行编码和解码。
            type='DeltaXYWHBBoxCoder',  # 框编码器的类别，'DeltaXYWHBBoxCoder' 是最常用的，更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/core/bbox/coder/delta_xywh_bbox_coder.py#L9。
            target_means=[0.0, 0.0, 0.0, 0.0],  # 用于编码和解码框的目标均值
            target_stds=[1.0, 1.0, 1.0, 1.0]),  # 用于编码和解码框的标准方差
        loss_cls=dict(  # 分类分支的损失函数配置
            type='CrossEntropyLoss',  # 分类分支的损失类型，我们也支持 FocalLoss 等。
            use_sigmoid=True,  # RPN通常进行二分类，所以通常使用sigmoid函数。
            los_weight=1.0),  # 分类分支的损失权重。
        loss_bbox=dict(  # 回归分支的损失函数配置。
            type='L1Loss',  # 损失类型，我们还支持许多 IoU Losses 和 Smooth L1-loss 等，更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/models/losses/smooth_l1_loss.py#L56。
            loss_weight=1.0)),  # 回归分支的损失权重。
    roi_head=dict(  # RoIHead 封装了两步(two-stage)/级联(cascade)检测器的第二步。
        type='StandardRoIHead',  # RoI head 的类型，更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/models/roi_heads/standard_roi_head.py#L10。
        bbox_roi_extractor=dict(  # 用于 bbox 回归的 RoI 特征提取器。
```

(下页继续)

(续上页)

```
            type='SingleRoIExtractor',  # RoI 特征提取器的类型，大多数方法使用 SingleRoIExtractor，更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/models/roi_heads/roi_extractors/single_level.py#L10。
            roi_layer=dict(  # RoI 层的配置
                type='RoIAlign',  # RoI 层的类别，也支持 DeformRoIPoolingPack 和 ModulatedDeformRoIPoolingPack，更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/ops/roi_align/roi_align.py#L79。
                output_size=7,  # 特征图的输出大小。
                sampling_ratio=0),  # 提取 RoI 特征时的采样率。0 表示自适应比率。
            out_channels=256,  # 提取特征的输出通道。
            featmap_strides=[4, 8, 16, 32]),  # 多尺度特征图的步幅，应该与主干的架构保持一致。
        bbox_head=dict(  # RoIHead 中 box head 的配置.
            type='Shared2FCBBoxHead',  # bbox head 的类别，更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/mmdet/models/roi_heads/bbox_heads/convfc_bbox_head.py#L177。
            in_channels=256,  # bbox head 的输入通道。 这与 roi_extractor 中的 out_channels 一致。
            fc_out_channels=1024,  # FC 层的输出特征通道。
            roi_feat_size=7,  # 候选区域(Region of Interest)特征的大小。
            num_classes=80,  # 分类的类别数量。
            bbox_coder=dict(  # 第二阶段使用的框编码器。
                type='DeltaXYWHBBoxCoder',  # 框编码器的类别，大多数情况使用 'DeltaXYWHBBoxCoder'。
                target_means=[0.0, 0.0, 0.0, 0.0],  # 用于编码和解码框的均值
                target_stds=[0.1, 0.1, 0.2, 0.2]),  # 编码和解码的标准方差。因为框更准确，所以值更小，常规设置时 [0.1, 0.1, 0.2, 0.2]。
            reg_class_agnostic=False,  # 回归是否与类别无关。
            loss_cls=dict(  # 分类分支的损失函数配置
                type='CrossEntropyLoss',  # 分类分支的损失类型，我们也支持 FocalLoss 等。
                use_sigmoid=False,  # 是否使用 sigmoid。
                loss_weight=1.0),  # 分类分支的损失权重。
            loss_bbox=dict(  # 回归分支的损失函数配置。
                type='L1Loss',  # 损失类型，我们还支持许多 IoU Losses 和 Smooth L1-loss 等。
                loss_weight=1.0)),  # 回归分支的损失权重。
        mask_roi_extractor=dict(  # 用于 mask 生成的 RoI 特征提取器。
            type='SingleRoIExtractor',  # RoI 特征提取器的类型，大多数方法使用 SingleRoIExtractor。
            roi_layer=dict(  # 提取实例分割特征的 RoI 层配置
                type='RoIAlign',  # RoI 层的类型，也支持 DeformRoIPoolingPack 和 ModulatedDeformRoIPoolingPack。
```

(下页继续)

(续上页)

```
                output_size=14,  # 特征图的输出大小。
                sampling_ratio=0),  # 提取 RoI 特征时的采样率。
            out_channels=256,  # 提取特征的输出通道。
            featmap_strides=[4, 8, 16, 32]),  # 多尺度特征图的步幅。
        mask_head=dict(  # mask 预测 head 模型
            type='FCNMaskHead',  # mask head 的类型，更多细节请参考 https://github.
↪com/open-mmlab/mmdetection/blob/master/mmdet/models/roi_heads/mask_heads/fcn_mask_
↪head.py#L21。
            num_convs=4,  # mask head 中的卷积层数
            in_channels=256,  # 输入通道，应与 mask roi extractor 的输出通道一致。
            conv_out_channels=256,  # 卷积层的输出通道。
            num_classes=80,  # 要分割的类别数。
            loss_mask=dict(  # mask 分支的损失函数配置。
                type='CrossEntropyLoss',  # 用于分割的损失类型。
                use_mask=True,  # 是否只在正确的类中训练 mask。
                loss_weight=1.0))))  # mask 分支的损失权重.
    train_cfg = dict(  # rpn 和 rcnn 训练超参数的配置
        rpn=dict(  # rpn 的训练配置
            assigner=dict(  # 分配器(assigner)的配置
                type='MaxIoUAssigner',  # 分配器的类型，MaxIoUAssigner␣
↪用于许多常见的检测器，更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/
↪master/mmdet/core/bbox/assigners/max_iou_assigner.py#L10。
                pos_iou_thr=0.7,  # IoU >= 0.7(阈值) 被视为正样本。
                neg_iou_thr=0.3,  # IoU < 0.3(阈值) 被视为负样本。
                min_pos_iou=0.3,  # 将框作为正样本的最小 IoU 阈值。
                match_low_quality=True,  # 是否匹配低质量的框(更多细节见 API 文档).
                ignore_iof_thr=-1),  # 忽略 bbox 的 IoF 阈值。
            sampler=dict(  # 正/负采样器(sampler)的配置
                type='RandomSampler',  # 采样器类型，还支持 PseudoSampler␣
↪和其他采样器，更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/
↪mmdet/core/bbox/samplers/random_sampler.py#L8。
                num=256,  # 样本数量。
                pos_fraction=0.5,  # 正样本占总样本的比例。
                neg_pos_ub=-1,  # 基于正样本数量的负样本上限。
                add_gt_as_proposals=False),  # 采样后是否添加 GT 作为 proposal。
            allowed_border=-1,  # 填充有效锚点后允许的边框。
            pos_weight=-1,  # 训练期间正样本的权重。
            debug=False),  # 是否设置调试(debug)模式
        rpn_proposal=dict(  # 在训练期间生成 proposals 的配置
            nms_across_levels=False,  # 是否对跨层的 box 做 NMS。仅适用于 `GARPNHead`␣
↪, naive rpn 不支持 nms cross levels。
            nms_pre=2000,  # NMS 前的 box 数
            nms_post=1000,  # NMS 要保留的 box 的数量，只在 GARPNHHead 中起作用。
```

(下页继续)

(续上页)

```
            max_per_img=1000,  # NMS 后要保留的 box 数量。
            nms=dict( # NMS 的配置
                type='nms',  # NMS 的类别
                iou_threshold=0.7 # NMS 的阈值
                ),
            min_bbox_size=0),  # 允许的最小 box 尺寸
        rcnn=dict(  # roi head 的配置。
            assigner=dict(  # 第二阶段分配器的配置，这与 rpn 中的不同
                type='MaxIoUAssigner',  # 分配器的类型，MaxIoUAssigner 目前用于所有
↪roi_heads。更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/
↪mmdet/core/bbox/assigners/max_iou_assigner.py#L10。
                pos_iou_thr=0.5,  # IoU >= 0.5(阈值)被认为是正样本。
                neg_iou_thr=0.5,  # IoU < 0.5(阈值)被认为是负样本。
                min_pos_iou=0.5,  # 将 box 作为正样本的最小 IoU 阈值
                match_low_quality=False,  # 是否匹配低质量下的
↪box(有关更多详细信息，请参阅 API 文档)。
                ignore_iof_thr=-1),  # 忽略 bbox 的 IoF 阈值
            sampler=dict(
                type='RandomSampler',  #采样器的类型，还支持 PseudoSampler
↪和其他采样器，更多细节请参考 https://github.com/open-mmlab/mmdetection/blob/master/
↪mmdet/core/bbox/samplers/random_sampler.py#L8。
                num=512,  # 样本数量
                pos_fraction=0.25,  # 正样本占总样本的比例。.
                neg_pos_ub=-1,  # 基于正样本数量的负样本上限。.
                add_gt_as_proposals=True
            ),  # 采样后是否添加 GT 作为 proposal。
            mask_size=28,  # mask 的大小
            pos_weight=-1,  # 训练期间正样本的权重。
            debug=False))  # 是否设置调试模式。
    test_cfg = dict(  # 用于测试 rnn 和 rnn 超参数的配置
        rpn=dict(  # 测试阶段生成 proposals 的配置
            nms_across_levels=False,  # 是否对跨层的 box 做
↪NMS。仅适用于`GARPNHead`，naive rpn 不支持做 NMS cross levels。
            nms_pre=1000,  # NMS 前的 box 数
            nms_post=1000,  # NMS 要保留的 box 的数量，只在`GARPNHHead`中起作用。
            max_per_img=1000,  # NMS 后要保留的 box 数量
            nms=dict( # NMS 的配置
                type='nms',  # NMS 的类型
                iou_threshold=0.7 # NMS 阈值
                ),
            min_bbox_size=0),  # box 允许的最小尺寸
        rcnn=dict(  # roi heads 的配置
            score_thr=0.05,  # bbox 的分数阈值
```

(下页继续)

(续上页)

```
            nms=dict(  # 第二步的 NMS 配置
                type='nms',  # NMS 的类型
                iou_thr=0.5),  # NMS 的阈值
            max_per_img=100,  # 每张图像的最大检测次数
            mask_thr_binary=0.5))  # mask 预处的阈值
dataset_type = 'CocoDataset'  # 数据集类型，这将被用来定义数据集。
data_root = 'data/coco/'  # 数据的根路径。
img_norm_cfg = dict(  #图像归一化配置，用来归一化输入的图像。
    mean=[123.675, 116.28, 103.53],  # 预训练里用于预训练主干网络模型的平均值。
    std=[58.395, 57.12, 57.375],  # 预训练里用于预训练主干网络模型的标准差。
    to_rgb=True
)  #  预训练里用于预训练主干网络的图像的通道顺序。
train_pipeline = [  # 训练流程
    dict(type='LoadImageFromFile'),  # 第 1 个流程，从文件路径里加载图像。
    dict(
        type='LoadAnnotations',  # 第 2 个流程，对于当前图像，加载它的注释信息。
        with_bbox=True,  # 是否使用标注框(bounding box)， 目标检测需要设置为 True。
        with_mask=True,  # 是否使用 instance mask，实例分割需要设置为 True。
        poly2mask=False),  # 是否将 polygon mask 转化为 instance mask，设置为 False
↪以加速和节省内存。
    dict(
        type='Resize',  # 变化图像和其注释大小的数据增广的流程。
        img_scale=(1333, 800),  # 图像的最大规模。
        keep_ratio=True
    ),  # 是否保持图像的长宽比。
    dict(
        type='RandomFlip',  #  翻转图像和其注释大小的数据增广的流程。
        flip_ratio=0.5),  # 翻转图像的概率。
    dict(
        type='Normalize',  # 归一化当前图像的数据增广的流程。
        mean=[123.675, 116.28, 103.53],  # 这些键与 img_norm_cfg 一致，因为 img_norm_
↪cfg 被
        std=[58.395, 57.12, 57.375],      # 用作参数。
        to_rgb=True),
    dict(
        type='Pad',  # 填充当前图像到指定大小的数据增广的流程。
        size_divisor=32),  # 填充图像可以被当前值整除。
    dict(type='DefaultFormatBundle'),  # 流程里收集数据的默认格式捆。
    dict(
        type='Collect',  # 决定数据中哪些键应该传递给检测器的流程
        keys=['img', 'gt_bboxes', 'gt_labels', 'gt_masks'])
]
test_pipeline = [
```

(下页继续)

(续上页)

```
    dict(type='LoadImageFromFile'),  # 第 1 个流程，从文件路径里加载图像。
    dict(
        type='MultiScaleFlipAug',  # 封装测试时数据增广(test time augmentations)。
        img_scale=(1333, 800),  #␣
↪决定测试时可改变图像的最大规模。用于改变图像大小的流程。
        flip=False,  # 测试时是否翻转图像。
        transforms=[
            dict(type='Resize',  # 使用改变图像大小的数据增广。
                 keep_ratio=True),  #␣
↪是否保持宽和高的比例，这里的图像比例设置将覆盖上面的图像规模大小的设置。
            dict(type='RandomFlip'),  # 考虑到 RandomFlip 已经被添加到流程里，当␣
↪flip=False 时它将不被使用。
            dict(
                type='Normalize',  #  归一化配置项，值来自 img_norm_cfg。
                mean=[123.675, 116.28, 103.53],
                std=[58.395, 57.12, 57.375],
                to_rgb=True),
            dict(
                type='Pad',  # 将配置传递给可被 32 整除的图像。
                size_divisor=32),
            dict(
                type='ImageToTensor',  # 将图像转为张量
                keys=['img']),
            dict(
                type='Collect',  # 收集测试时必须的键的收集流程。
                keys=['img'])
        ])
]
data = dict(
    samples_per_gpu=2,  # 单个 GPU 的 Batch size
    workers_per_gpu=2,  # 单个 GPU 分配的数据加载线程数
    train=dict(  # 训练数据集配置
        type='CocoDataset',  # 数据集的类别，更多细节请参考 https://github.com/open-
↪mmlab/mmdetection/blob/master/mmdet/datasets/coco.py#L19。
        ann_file='data/coco/annotations/instances_train2017.json',  # 注释文件路径
        img_prefix='data/coco/train2017/',  # 图片路径前缀
        pipeline=[  # 流程，这是由之前创建的 train_pipeline 传递的。
            dict(type='LoadImageFromFile'),
            dict(
                type='LoadAnnotations',
                with_bbox=True,
                with_mask=True,
                poly2mask=False),
```

(下页继续)

(续上页)

```
            dict(type='Resize', img_scale=(1333, 800), keep_ratio=True),
            dict(type='RandomFlip', flip_ratio=0.5),
            dict(
                type='Normalize',
                mean=[123.675, 116.28, 103.53],
                std=[58.395, 57.12, 57.375],
                to_rgb=True),
            dict(type='Pad', size_divisor=32),
            dict(type='DefaultFormatBundle'),
            dict(
                type='Collect',
                keys=['img', 'gt_bboxes', 'gt_labels', 'gt_masks'])
        ]),
    val=dict(  # 验证数据集的配置
        type='CocoDataset',
        ann_file='data/coco/annotations/instances_val2017.json',
        img_prefix='data/coco/val2017/',
        pipeline=[  # 由之前创建的 test_pipeline 传递的流程。
            dict(type='LoadImageFromFile'),
            dict(
                type='MultiScaleFlipAug',
                img_scale=(1333, 800),
                flip=False,
                transforms=[
                    dict(type='Resize', keep_ratio=True),
                    dict(type='RandomFlip'),
                    dict(
                        type='Normalize',
                        mean=[123.675, 116.28, 103.53],
                        std=[58.395, 57.12, 57.375],
                        to_rgb=True),
                    dict(type='Pad', size_divisor=32),
                    dict(type='ImageToTensor', keys=['img']),
                    dict(type='Collect', keys=['img'])
                ])
        ]),
    test=dict(  # 测试数据集配置，修改测试开发/测试(test-dev/test)提交的 ann_file
        type='CocoDataset',
        ann_file='data/coco/annotations/instances_val2017.json',
        img_prefix='data/coco/val2017/',
        pipeline=[  # 由之前创建的 test_pipeline 传递的流程。
            dict(type='LoadImageFromFile'),
            dict(
```

(下页继续)

(续上页)

```
                type='MultiScaleFlipAug',
                img_scale=(1333, 800),
                flip=False,
                transforms=[
                    dict(type='Resize', keep_ratio=True),
                    dict(type='RandomFlip'),
                    dict(
                        type='Normalize',
                        mean=[123.675, 116.28, 103.53],
                        std=[58.395, 57.12, 57.375],
                        to_rgb=True),
                    dict(type='Pad', size_divisor=32),
                    dict(type='ImageToTensor', keys=['img']),
                    dict(type='Collect', keys=['img'])
                ])
        ],
        samples_per_gpu=2  # 单个 GPU 测试时的 Batch size
        ))
evaluation = dict(  # evaluation hook 的配置，更多细节请参考 https://github.com/open-
↪mmlab/mmdetection/blob/master/mmdet/core/evaluation/eval_hooks.py#L7。
    interval=1,  # 验证的间隔。
    metric=['bbox', 'segm'])  # 验证期间使用的指标。
optimizer = dict(  # 用于构建优化器的配置文件。支持 PyTorch␣
↪中的所有优化器，同时它们的参数与 PyTorch 里的优化器参数一致。
    type='SGD',  # 优化器种类，更多细节可参考 https://github.com/open-mmlab/
↪mmdetection/blob/master/mmdet/core/optimizer/default_constructor.py#L13。
    lr=0.02,  # 优化器的学习率，参数的使用细节请参照对应的 PyTorch 文档。
    momentum=0.9,  # 动量(Momentum)
    weight_decay=0.0001)  # SGD 的衰减权重(weight decay)。
optimizer_config = dict(  # optimizer hook 的配置文件，执行细节请参考 https://github.
↪com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/optimizer.py#L8。
    grad_clip=None)  # 大多数方法不使用梯度限制(grad_clip)。
lr_config = dict(  # 学习率调整配置，用于注册 LrUpdater hook。
    policy='step',  # 调度流程(scheduler)的策略，也支持 CosineAnnealing, Cyclic,␣
↪等。请从 https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/lr_
↪updater.py#L9 参考 LrUpdater 的细节。
    warmup='linear',  # 预热(warmup)策略，也支持 `exp` 和 `constant`。
    warmup_iters=500,  # 预热的迭代次数
    warmup_ratio=
    0.001,  # 用于热身的起始学习率的比率
    step=[8, 11])  # 衰减学习率的起止回合数
runner = dict(
    type='EpochBasedRunner',  # 将使用的 runner 的类别 (例如 IterBasedRunner 或␣
↪EpochBasedRunner)。
```

(下页继续)

(续上页)

```
    max_epochs=12) # runner 总回合数， 对于 IterBasedRunner 使用 `max_iters`
checkpoint_config = dict(  # Checkpoint hook 的配置文件。执行时请参考 https://github.
↪com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/checkpoint.py。
    interval=1)  # 保存的间隔是 1。
log_config = dict(  # register logger hook 的配置文件。
    interval=50,  # 打印日志的间隔
    hooks=[
        # dict(type='TensorboardLoggerHook')  # 同样支持 Tensorboard 日志
        dict(type='TextLoggerHook')
    ])  # 用于记录训练过程的记录器(logger)。
dist_params = dict(backend='nccl')  # 用于设置分布式训练的参数，端口也同样可被设置。
log_level = 'INFO'  # 日志的级别。
load_from = None  # 从一个给定路径里加载模型作为预训练模型，它并不会消耗训练时间。
resume_from = None  #␣
↪从给定路径里恢复检查点(checkpoints)，训练模式将从检查点保存的轮次开始恢复训练。
workflow = [('train', 1)]  # runner 的工作流程，[('train', 1)]␣
↪表示只有一个工作流且工作流仅执行一次。根据 total_epochs 工作流训练 12个回合。
work_dir = 'work_dir'  # 用于保存当前实验的模型检查点和日志的目录文件地址。
```

7.6 常问问题 (FAQ)

7.6.1 忽略基础配置文件里的部分内容

有时，您也许会设置 `_delete_=True` 去忽略基础配置文件里的一些域内容。您也许可以参照 mmcv 来获得一些简单的指导。

在 MMDetection 里，例如为了改变 Mask R-CNN 的主干网络的某些内容：

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

(下页继续)

(续上页)

```
    rpn_head=dict(...),
    roi_head=dict(...))
```

基础配置的 Mask R-CNN 使用 ResNet-50，在需要将主干网络改成 HRNet 的时候，因为 HRNet 和 ResNet 中有不同的字段，需要使用 _delete_=True 将新的键去替换 backbone 域内所有老的键。

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

7.6.2 使用配置文件里的中间变量

配置文件里会使用一些中间变量，例如数据集里的 `train_pipeline/test_pipeline`。我们在定义新的 `train_pipeline/test_pipeline` 之后，需要将它们传递到 `data` 里。例如，我们想在训练或测试时，改变 Mask R-CNN 的多尺度策略 (multi scale strategy)，`train_pipeline/test_pipeline` 是我们想要修改的中间变量。

```
_base_ = './mask_rcnn_r50_fpn_1x_coco.py'
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='LoadAnnotations', with_bbox=True, with_mask=True),
    dict(
        type='Resize',
        img_scale=[(1333, 640), (1333, 672), (1333, 704), (1333, 736),
                   (1333, 768), (1333, 800)],
        multiscale_mode="value",
        keep_ratio=True),
    dict(type='RandomFlip', flip_ratio=0.5),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='Pad', size_divisor=32),
    dict(type='DefaultFormatBundle'),
    dict(type='Collect', keys=['img', 'gt_bboxes', 'gt_labels', 'gt_masks']),
]
test_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(
        type='MultiScaleFlipAug',
        img_scale=(1333, 800),
        flip=False,
        transforms=[
            dict(type='Resize', keep_ratio=True),
            dict(type='RandomFlip'),
            dict(type='Normalize', **img_norm_cfg),
            dict(type='Pad', size_divisor=32),
            dict(type='ImageToTensor', keys=['img']),
            dict(type='Collect', keys=['img']),
        ])
]
data = dict(
    train=dict(pipeline=train_pipeline),
    val=dict(pipeline=test_pipeline),
    test=dict(pipeline=test_pipeline))
```

我们首先定义新的 train_pipeline/test_pipeline 然后传递到 data 里。

同样的，如果我们想从 SyncBN 切换到 BN 或者 MMSyncBN，我们需要修改配置文件里的每一个 norm_cfg。

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

CHAPTER 8

教程 2: 自定义数据集

CHAPTER 9

教程 3: 自定义数据预处理流程

CHAPTER 10

教程 4: 自定义模型

CHAPTER 11

教程 5: 自定义训练配置

CHAPTER 12

教程 6: 自定义损失函数

CHAPTER 13

教程 7: 模型微调

CHAPTER 14

教程 8: Pytorch 到 ONNX 的模型转换（实验性支持）

CHAPTER 15

教程 9: ONNX 到 TensorRT 的模型转换（实验性支持）

CHAPTER 16

日志分析

CHAPTER 17

默认约定

CHAPTER 18

MMDetection v2.x 兼容性说明

18.1 MMDetection v2.14.0

18.1.1 MMCV 版本

为了修复 EvalHook 优先级过低的问题，MMCV v1.3.8 中所有 hook 的优先级都重新进行了调整，因此 MMDetection v2.14.0 需要依赖最新的 MMCV v1.3.8 版本。相关信息请参考PR #1120 ，相关问题请参考#5343 。

18.1.2 SSD 兼容性

在 v2.14.0 中，为了使 SSD 能够被更灵活地使用，PR #5291 重构了 SSD 的 backbone、neck 和 head。用户可以使用 tools/model_converters/upgrade_ssd_version.py 转换旧版本训练的模型。

```
python tools/model_converters/upgrade_ssd_version.py ${OLD_MODEL_PATH} ${NEW_MODEL_
↪PATH}
```

- OLD_MODEL_PATH：旧版 SSD 模型的路径。
- NEW_MODEL_PATH：保存转换后模型权重的路径。

18.2 MMDetection v2.12.0

在 v2.12.0 到 v2.18.0（或以上）版本的这段时间，为了提升通用性和便捷性，MMDetection 正在进行大规模重构。在升级到 v2.12.0 后 MMDetection 不可避免地带来了一些 BC Breaking，包括 MMCV 的版本依赖、模型初始化方式、模型 registry 和 mask AP 的评估。

18.2.1 MMCV 版本

MMDetection v2.12.0 依赖 MMCV v1.3.3 中新增加的功能，包括：使用 `BaseModule` 统一参数初始化，模型 registry，以及Deformable DETR 中的 `MultiScaleDeformableAttn` CUDA 算子。注意，尽管 MMCV v1.3.2 已经包含了 MMDet 所需的功能，但是存在一些已知的问题。我们建议用户跳过 MMCV v1.3.2 使用 v1.3.3 版本。

18.2.2 统一模型初始化

为了统一 OpenMMLab 项目中的参数初始化方式，MMCV 新增加了 `BaseModule` 类，使用 `init_cfg` 参数对模块进行统一且灵活的初始化配置管理。现在用户需要在训练脚本中显式调用 `model.init_weights()` 来初始化模型（例如 这行代码 ，在这之前则是在 detector 中进行处理的。**下游项目必须相应地更新模型初始化方式才能使用 MMDetection v2.12.0**。请参阅 PR #4750 了解详情。

18.2.3 统一模型 registry

为了能够使用在其他 OpenMMLab 项目中实现的 backbone，MMDetection v2.12.0 继承了在 MMCV (#760) 中创建的模型 registry。这样，只要 OpenMMLab 项目实现了某个 backbone，并且该项目也使用 MMCV 中的 registry，那么用户只需修改配置即可在 MMDetection 中使用该 backbone，不再需要将代码复制到 MMDetection 中。更多详细信息，请参阅 PR #5059 。

18.2.4 Mask AP 评估

在 PR #4898 和 v2.12.0 之前，对小、中、大目标的 mask AP 的评估是基于其边界框区域而不是真正的 mask 区域。这导致 `APs` 和 `APm` 变得更高但 `APl` 变得更低，但是不会影响整体的 mask AP。PR #4898 删除了 mask AP 计算中的 `bbox` ，改为使用 mask 区域。新的计算方式不会影响整体的 mask AP 评估，与 Detectron2一致。

18.3 与 MMDetection v1.x 的兼容性

MMDetection v2.0 经过了大规模重构并解决了许多遗留问题。MMDetection v2.0 不兼容 v1.x 版本，在这两个版本中使用相同的模型权重运行推理会产生不同的结果。因此，MMDetection v2.0 重新对所有模型进行了 benchmark，并在 model zoo 中提供了新模型的权重和训练记录。

新旧版本的主要的区别有四方面：坐标系、代码库约定、训练超参和模块设计。

18.3.1 坐标系

新坐标系与 Detectron2 一致，将最左上角的像素的中心视为坐标原点 (0, 0) 而不是最左上角像素的左上角。因此 COCO 边界框和分割标注中的坐标被解析为范围 `[0, width]` 和 `[0, height]` 中的坐标。这个修改影响了所有与 bbox 及像素选择相关的计算，变得更加自然且更加准确。

- 在新坐标系中，左上角和右下角为 (x1, y1) (x2, y2) 的框的宽度及高度计算公式为 `width = x2 - x1` 和 `height = y2 - y1`。在 MMDetection v1.x 和之前的版本中，高度和宽度都多了 `+ 1` 的操作。本次修改包括三部分：
  1. box 回归中的检测框变换以及编码/解码。
  2. IoU 计算。这会影响 ground truth 和检测框之间的匹配以及 NMS 。但对兼容性的影响可以忽略不计。
  3. Box 的角点坐标为浮点型，不再取整。这能使得检测结果更为准确，也使得检测框和 RoI 的最小尺寸不再为 1，但影响很小。
- Anchor 的中心与特征图的网格点对齐，类型变为 float。在 MMDetection v1.x 和之前的版本中，anchors 是 `int` 类型且没有居中对齐。这会影响 RPN 中的 Anchor 生成和所有基于 Anchor 的方法。
- ROIAlign 更好地与图像坐标系对齐。新的实现来自 Detectron2 。当 RoI 用于提取 RoI 特征时，与 MMDetection v1.x 相比默认情况下相差半个像素。能够通过设置 `aligned=False` 而不是 `aligned=True` 来维持旧版本的设置。
- Mask 的裁剪和粘贴更准确。
  1. 我们使用新的 RoIAlign 来提取 mask 目标。在 MMDetection v1.x 中，bounding box 在提取 mask 目标之前被取整，裁剪过程是 numpy 实现的。而在新版本中，裁剪的边界框不经过取整直接输入 RoIAlign。此实现大大加快了训练速度（每次迭代约加速 0.1 秒，1x schedule 训练 Mask R50 时加速约 2 小时）并且理论上会更准确。
  2. 在 MMDetection v2.0 中，修改后的 `paste_mask()` 函数应该比之前版本更准确。此更改参考了 Detectron2 中的修改，可以将 COCO 上的 mask AP 提高约 0.5%。

18.3.2 代码库约定

- MMDetection v2.0 更改了类别标签的顺序，减少了回归和 mask 分支里的无用参数并使得顺序更加自然（没有 +1 和 -1）。这会影响模型的所有分类层，使其输出的类别标签顺序发生改变。回归分支和 mask head 的最后一层不再为 K 个类别保留 K+1 个通道，类别顺序与分类分支一致。
  - 在 MMDetection v2.0 中，标签 “K” 表示背景，标签 [0, K-1] 对应于 K = num_categories 个对象类别。
  - 在 MMDetection v1.x 及之前的版本中，标签 “0” 表示背景，标签 [1, K] 对应 K 个类别。
  - **注意**：softmax RPN 的类顺序在 version<=2.4.0 中仍然和 1.x 中的一样，而 sigmoid RPN 不受影响。从 MMDetection v2.5.0 开始，所有 head 中的类顺序是统一的。
- 不使用 R-CNN 中的低质量匹配。在 MMDetection v1.x 和之前的版本中，`max_iou_assigner` 会在 RPN 和 R-CNN 训练时给每个 ground truth 匹配低质量框。我们发现这会导致最佳的 GT 框不会被分配给某些边界框，因此，在 MMDetection v2.0 的 R-CNN 训练中默认不允许低质量匹配。这有时可能会稍微改善 box AP（约为 0.1%）。
- 单独的宽高比例系数。在 MMDetection v1.x 和以前的版本中，`keep_ratio=True` 时比例系数是单个浮点数，这并不准确，因为宽度和高度的比例系数会有一定的差异。MMDetection v2.0 对宽度和高度使用单独的比例系数，对 AP 的提升约为 0.1%。
- 修改了 config 文件名称的规范。由于 model zoo 中模型不断增多，MMDetection v2.0 采用新的命名规则：

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

  其中 (`misc`) 包括 DCN 和 GCBlock 等。更多详细信息在 配置文件说明文档 中说明
- MMDetection v2.0 使用新的 ResNet Caffe backbone 来减少加载预训练模型时的警告。新 backbone 中的大部分权重与以前的相同，但没有 `conv.bias`，且它们使用不同的 `img_norm_cfg`。因此，新的 backbone 不会报 `unexpected keys` 的警告。

18.3.3 训练超参

训练超参的调整不会影响模型的兼容性，但会略微提高性能。主要有：

- 通过设置 `nms_post=1000` 和 `max_num=1000`，将 nms 之后的 proposal 数量从 2000 更改为 1000。使 mask AP 和 bbox AP 提高了约 0.2%。
- Mask R-CNN、Faster R-CNN 和 RetinaNet 的默认回归损失从 smooth L1 损失更改为 L1 损失，使得 box AP 整体上都有所提升（约 0.6%）。但是，将 L1-loss 用在 Cascade R-CNN 和 HTC 等其他方法上并不能提高性能，因此我们保留这些方法的原始设置。
- 为简单起见，RoIAlign 层的 `sampling_ratio` 设置为 0。略微提升了 AP（约 0.2% 绝对值）。
- 为了提升训练速度，默认设置在训练过程中不再使用梯度裁剪。大多数模型的性能不会受到影响。对于某些模型（例如 RepPoints），我们依旧使用梯度裁剪来稳定训练过程从而获得更好的性能。

- 因为不再默认使用梯度裁剪，默认 warmup 比率从 1/3 更改为 0.001，以使模型训练预热更加平缓。不过我们重新进行基准测试时发现这种影响可以忽略不计。

18.3.4 将模型从 v1.x 升级至 v2.0

用户可以使用脚本 `tools/model_converters/upgrade_model_version.py` 来将 MMDetection 1.x 训练的模型转换为 MMDetection v2.0。转换后的模型可以在 MMDetection v2.0 中运行，但性能略有下降（小于 1% AP）。详细信息可以在 `configs/legacy` 中找到。

18.4 pycocotools 兼容性

`mmpycocotools` 是 OpenMMLab 维护的 `pycocotools` 的复刻版，适用于 MMDetection 和 Detectron2。在 PR #4939 之前，由于 `pycocotools` 和 `mmpycocotool` 具有相同的包名，如果用户已经安装了 `pyccocotools`（在相同环境下先安装了 Detectron2 ），那么 MMDetection 的安装过程会跳过安装 `mmpycocotool`。导致 MMDetection 缺少 `mmpycocotools` 而报错。但如果在 Detectron2 之前安装 MMDetection，则可以在相同的环境下工作。PR #4939 弃用 mmpycocotools，使用官方 pycocotools。在 PR #4939 之后，用户能够在相同环境下安装 MMDetection 和 Detectron2，不再需要关注安装顺序。

CHAPTER 19

常见问题解答

我们在这里列出了使用时的一些常见问题及其相应的解决方案。如果您发现有一些问题被遗漏，请随时提 PR 丰富这个列表。如果您无法在此获得帮助，请使用 issue 模板创建问题，但是请在模板中填写所有必填信息，这有助于我们更快定位问题。

19.1 MMCV 安装相关

- MMCV 与 MMDetection 的兼容问题: “ConvWS is already registered in conv layer”；“AssertionError: MMCV==xxx is used but incompatible. Please install mmcv>=xxx, <=xxx.”

  请按 安装说明 为你的 MMDetection 安装正确版本的 MMCV 。

- “No module named ‘mmcv.ops’”；“No module named ‘mmcv._ext’”.

  原因是安装了 `mmcv` 而不是 `mmcv-full`。

  1. `pip uninstall mmcv` 卸载安装的 `mmcv`
  2. 安装 `mmcv-full` 根据 安装说明。

19.2 PyTorch/CUDA 环境相关

- “RTX 30 series card fails when building MMCV or MMDet”

  1. 临时解决方案为使用命令 `MMCV_WITH_OPS=1 MMCV_CUDA_ARGS='-gencode=arch=compute_80,code=sm_80' pip install -e .` 进行编译。常见报错信息为 `nvcc fatal : Unsupported gpu architecture 'compute_86'` 意思是你的编译器不支持 sm_86 架构 (包括英伟达 30 系列的显卡) 的优化，至 CUDA toolkit 11.0 依旧未支持. 这个命令是通过增加宏 `MMCV_CUDA_ARGS='-gencode=arch=compute_80,code=sm_80` 让 nvcc 编译器为英伟达 30 系列显卡进行 `sm_80` 的优化，虽然这有可能会无法发挥出显卡所有性能。
  2. 有开发者已经在 pytorch/pytorch#47585 更新了 PyTorch 默认的编译 flag，但是我们对此并没有进行测试。

- “invalid device function” or “no kernel image is available for execution”.

  1. 检查您正常安装了 CUDA runtime (一般在`/usr/local/`)，或者使用 `nvcc --version` 检查本地版本，有时安装 PyTorch 会顺带安装一个 CUDA runtime，并且实际优先使用 conda 环境中的版本，你可以使用 `conda list cudatoolkit` 查看其版本。
  2. 编译 extention 的 CUDA Toolkit 版本与运行时的 CUDA Toolkit 版本是否相符，
     - 如果您从源码自己编译的，使用 `python mmdet/utils/collect_env.py` 检查编译编译 extention 的 CUDA Toolkit 版本，然后使用 `conda list cudatoolkit` 检查当前 conda 环境是否有 CUDA Toolkit，若有检查版本是否匹配，如不匹配，更换 conda 环境的 CUDA Toolkit，或者使用匹配的 CUDA Toolkit 中的 nvcc 编译即可，如环境中无 CUDA Toolkit，可以使用 `nvcc -V`。

       等命令查看当前使用的 CUDA runtime。
     - 如果您是通过 pip 下载的预编译好的版本，请确保与当前 CUDA runtime 一致。
  3. 运行 `python mmdet/utils/collect_env.py` 检查是否为正确的 GPU 架构编译的 PyTorch，torchvision，与 MMCV。你或许需要设置 `TORCH_CUDA_ARCH_LIST` 来重新安装 MMCV，可以参考 GPU 架构表, 例如，运行 `TORCH_CUDA_ARCH_LIST=7.0 pip install mmcv-full` 为 Volta GPU 编译 MMCV。这种架构不匹配的问题一般会出现在使用一些旧型号的 GPU 时候出现，例如，Tesla K80。

- “undefined symbol” or “cannot open xxx.so”.

  1. 如果这些 symbol 属于 CUDA/C++ (如 libcudart.so 或者 GLIBCXX)，使用 `python mmdet/utils/collect_env.py` 检查 CUDA/GCC runtime 与编译 MMCV 的 CUDA 版本是否相同。
  2. 如果这些 symbols 属于 PyTorch，(例如, symbols containing caffe, aten, and TH), 检查当前 Pytorch 版本是否与编译 MMCV 的版本一致。
  3. 运行 `python mmdet/utils/collect_env.py` 检查 PyTorch，torchvision，MMCV 等的编译环境与运行环境一致。

- setuptools.sandbox.UnpickleableException: DistutilsSetupError(" each element of 'ext_modules' option must be an Extension instance or 2-tuple")

  1. 如果你在使用 miniconda 而不是 anaconda，检查是否正确的安装了 Cython 如 #3379.
  2. 检查环境中的 `setuptools`, `Cython`, and `PyTorch` 相互之间版本是否匹配。

- "Segmentation fault".

  1. 检查 GCC 的版本，通常是因为 PyTorch 版本与 GCC 版本不匹配（例如 GCC < 4.9），我们推荐用户使用 GCC 5.4，我们也不推荐使用 GCC 5.5，因为有反馈 GCC 5.5 会导致 "segmentation fault" 并且切换到 GCC 5.4 就可以解决问题。
  2. 检查是否正确安装了 CUDA 版本的 PyTorch 。

     ```
     python -c 'import torch; print(torch.cuda.is_available())'
     ```

     是否返回 True。
  3. 如果 `torch` 的安装是正确的，检查是否正确编译了 MMCV。

     ```
     python -c 'import mmcv; import mmcv.ops'
     ```
  4. 如果 MMCV 与 PyTorch 都被正确安装了，则使用 `ipdb`, `pdb` 设置断点，直接查找哪一部分的代码导致了 `segmentation fault`。

19.3 Training 相关

- "Loss goes Nan"

  1. 检查数据的标注是否正常，长或宽为 0 的框可能会导致回归 loss 变为 nan，一些小尺寸（宽度或高度小于 1）的框在数据增强（例如，instaboost）后也会导致此问题。因此，可以检查标注并过滤掉那些特别小甚至面积为 0 的框，并关闭一些可能会导致 0 面积框出现数据增强。
  2. 降低学习率：由于某些原因，例如 batch size 大小的变化，导致当前学习率可能太大。您可以降低为可以稳定训练模型的值。
  3. 延长 warm up 的时间：一些模型在训练初始时对学习率很敏感，您可以把 `warmup_iters` 从 500 更改为 1000 或 2000。
  4. 添加 gradient clipping: 一些模型需要梯度裁剪来稳定训练过程。默认的 `grad_clip` 是 `None`, 你可以在 config 设置 `optimizer_config=dict(_delete_=True, grad_clip=dict(max_norm=35, norm_type=2))` 如果你的 config 没有继承任何包含 `optimizer_config=dict(grad_clip=None)`, 你可以直接设置 `optimizer_config=dict(grad_clip=dict(max_norm=35, norm_type=2))`.

- ' GPU out of memory"

1. 存在大量 ground truth boxes 或者大量 anchor 的场景，可能在 assigner 会 OOM。您可以在 assigner 的配置中设置 `gpu_assign_thr=N`，这样当超过 N 个 GT boxes 时，assigner 会通过 CPU 计算 IOU。
2. 在 backbone 中设置 `with_cp=True`。这使用 PyTorch 中的 `sublinear strategy` 来降低 backbone 占用的 GPU 显存。
3. 使用 `config/fp16` 中的示例尝试混合精度训练。`loss_scale` 可能需要针对不同模型进行调整。

- "RuntimeError: Expected to have finished reduction in the prior iteration before starting a new one"
  1. 这个错误出现在存在参数没有在 forward 中使用，容易在 DDP 中运行不同分支时发生。
  2. 你可以在 config 设置 `find_unused_parameters = True`，或者手动查找哪些参数没有用到。

19.4 Evaluation 相关

- 使用 COCO Dataset 的测评接口时, 测评结果中 AP 或者 AR = -1
  1. 根据 COCO 数据集的定义，一张图像中的中等物体与小物体面积的阈值分别为 9216（96*96）与 1024（32*32）。
  2. 如果在某个区间没有检测框 AP 与 AR 认定为 -1.

CHAPTER 20

English

CHAPTER 21

简体中文

CHAPTER 22

API Reference

22.1 mmdet.apis

22.2 mmdet.core

22.2.1 anchor

class mmdet.core.anchor.**AnchorGenerator**(*strides*, *ratios*, *scales=None*, *base_sizes=None*, *scale_major=True*, *octave_base_scale=None*, *scales_per_octave=None*, *centers=None*, *center_offset=0.0*)

Standard anchor generator for 2D anchor-based detectors.

参数

- **strides** (*list[int] | list[tuple[int, int]]*) –Strides of anchors in multiple feature levels in order (w, h).
- **ratios** (*list[float]*) –The list of ratios between the height and width of anchors in a single level.
- **scales** (*list[int] | None*) –Anchor scales for anchors in a single level. It cannot be set at the same time if *octave_base_scale* and *scales_per_octave* are set.
- **base_sizes** (*list[int] | None*) –The basic sizes of anchors in multiple levels. If None is given, strides will be used as base_sizes. (If strides are non square, the shortest stride

is taken.)

- **scale_major** (*bool*) –Whether to multiply scales first when generating base anchors. If true, the anchors in the same row will have the same scales. By default it is True in V2.0
- **octave_base_scale** (*int*) –The base scale of octave.
- **scales_per_octave** (*int*) –Number of scales for each octave. *octave_base_scale* and *scales_per_octave* are usually used in retinanet and the *scales* should be None when they are set.
- **centers** (*list[tuple[float, float]] | None*) –The centers of the anchor relative to the feature grid center in multiple feature levels. By default it is set to be None and not used. If a list of tuple of float is given, they will be used to shift the centers of anchors.
- **center_offset** (*float*) –The offset of center in proportion to anchors' width and height. By default it is 0 in V2.0.

实际案例

```
>>> from mmdet.core import AnchorGenerator
>>> self = AnchorGenerator([16], [1.], [1.], [9])
>>> all_anchors = self.grid_anchors([(2, 2)], device='cpu')
>>> print(all_anchors)
[tensor([[-4.5000, -4.5000,  4.5000,  4.5000],
        [11.5000, -4.5000, 20.5000,  4.5000],
        [-4.5000, 11.5000,  4.5000, 20.5000],
        [11.5000, 11.5000, 20.5000, 20.5000]])]
>>> self = AnchorGenerator([16, 32], [1.], [1.], [9, 18])
>>> all_anchors = self.grid_anchors([(2, 2), (1, 1)], device='cpu')
>>> print(all_anchors)
[tensor([[-4.5000, -4.5000,  4.5000,  4.5000],
        [11.5000, -4.5000, 20.5000,  4.5000],
        [-4.5000, 11.5000,  4.5000, 20.5000],
        [11.5000, 11.5000, 20.5000, 20.5000]]),         tensor([[-9., -9., 9., 9.
↪]])]
```

gen_base_anchors()

Generate base anchors.

返回 Base anchors of a feature grid in multiple feature levels.

返回类型 list(torch.Tensor)

gen_single_level_base_anchors(*base_size*, *scales*, *ratios*, *center=None*)

Generate base anchors of a single level.

参数

- **base_size** (*int | float*) –Basic size of an anchor.
- **scales** (*torch.Tensor*) –Scales of the anchor.
- **ratios** (*torch.Tensor*) –The ratio between between the height and width of anchors in a single level.
- **center** (*tuple[float], optional*) –The center of the base anchor related to a single feature grid. Defaults to None.

返回 Anchors in a single-level feature maps.

返回类型 torch.Tensor

grid_anchors(*featmap_sizes*, *device='cuda'*)

Generate grid anchors in multiple feature levels.

参数

- **featmap_sizes** (*list[tuple]*) –List of feature map sizes in multiple feature levels.
- **device** (*str*) –Device where the anchors will be put on.

返回 Anchors in multiple feature levels. The sizes of each tensor should be [N, 4], where N = width * height * num_base_anchors, width and height are the sizes of the corresponding feature level, num_base_anchors is the number of anchors for that level.

返回类型 list[torch.Tensor]

grid_priors(*featmap_sizes*, *device='cuda'*)

Generate grid anchors in multiple feature levels.

参数

- **featmap_sizes** (*list[tuple]*) –List of feature map sizes in multiple feature levels.
- **device** (*str*) –The device where the anchors will be put on.

返回 Anchors in multiple feature levels. The sizes of each tensor should be [N, 4], where N = width * height * num_base_anchors, width and height are the sizes of the corresponding feature level, num_base_anchors is the number of anchors for that level.

返回类型 list[torch.Tensor]

property num_base_anchors

total number of base anchors in a feature grid

Type list[int]

property num_base_priors

The number of priors (anchors) at a point on the feature grid

Type list[int]

property num_levels

number of feature levels that the generator will be applied

Type int

single_level_grid_anchors(*base_anchors*, *featmap_size*, *stride=(16, 16)*, *device='cuda'*)

Generate grid anchors of a single level.

注解： This function is usually called by method `self.grid_anchors`.

参数

- **base_anchors** (*torch.Tensor*) –The base anchors of a feature grid.
- **featmap_size** (*tuple[int]*) –Size of the feature maps.
- **stride** (*tuple[int], optional*) –Stride of the feature map in order (w, h). Defaults to (16, 16).
- **device** (*str, optional*) –Device the tensor will be put on. Defaults to 'cuda'.

返回 Anchors in the overall feature maps.

返回类型 torch.Tensor

single_level_grid_priors(*featmap_size*, *level_idx*, *device='cuda'*)

Generate grid anchors of a single level.

注解： This function is usually called by method `self.grid_priors`.

参数

- **featmap_size** (*tuple[int]*) –Size of the feature maps.
- **level_idx** (*int*) –The index of corresponding feature map level.
- **device** (*str, optional*) –The device the tensor will be put on. Defaults to 'cuda'.

返回 Anchors in the overall feature maps.

返回类型 torch.Tensor

single_level_valid_flags(*featmap_size*, *valid_size*, *num_base_anchors*, *device='cuda'*)

Generate the valid flags of anchor in a single feature map.

参数

- **featmap_size** (*tuple[int]*) –The size of feature maps, arrange as (h, w).

- **valid_size** (*tuple[int]*) –The valid size of the feature maps.
- **num_base_anchors** (*int*) –The number of base anchors.
- **device** (*str, optional*) –Device where the flags will be put on. Defaults to 'cuda'.

返回 The valid flags of each anchor in a single level feature map.

返回类型 torch.Tensor

sparse_priors(*prior_idxs*, *featmap_size*, *level_idx*, *dtype=torch.float32*, *device='cuda'*)

Generate sparse anchors according to the `prior_idxs`.

参数

- **prior_idxs** (*Tensor*) –The index of corresponding anchors in the feature map.
- **featmap_size** (*tuple[int]*) –feature map size arrange as (h, w).
- **level_idx** (*int*) –The level index of corresponding feature map.
- **(obj** (*device*) –*torch.dtype*): Date type of points.Defaults to `torch.float32`.
- **(obj** –*torch.device*): The device where the points is located.

返回

Anchor with shape (N, 4), N should be equal to the length of `prior_idxs`.

返回类型 Tensor

valid_flags(*featmap_sizes*, *pad_shape*, *device='cuda'*)

Generate valid flags of anchors in multiple feature levels.

参数

- **featmap_sizes** (*list(tuple)*) –List of feature map sizes in multiple feature levels.
- **pad_shape** (*tuple*) –The padded shape of the image.
- **device** (*str*) –Device where the anchors will be put on.

返回 Valid flags of anchors in multiple levels.

返回类型 list(torch.Tensor)

class mmdet.core.anchor.**LegacyAnchorGenerator**(*strides*, *ratios*, *scales=None*, *base_sizes=None*, *scale_major=True*, *octave_base_scale=None*, *scales_per_octave=None*, *centers=None*, *center_offset=0.0*)

Legacy anchor generator used in MMDetection V1.x.

注解: Difference to the V2.0 anchor generator:

1. The center offset of V1.x anchors are set to be 0.5 rather than 0.
2. The width/height are minused by 1 when calculating the anchors' centers and corners to meet the V1.x coordinate system.
3. The anchors' corners are quantized.

参数

- **strides** (*list[int] | list[tuple[int]]*) –Strides of anchors in multiple feature levels.
- **ratios** (*list[float]*) –The list of ratios between the height and width of anchors in a single level.
- **scales** (*list[int] | None*) –Anchor scales for anchors in a single level. It cannot be set at the same time if *octave_base_scale* and *scales_per_octave* are set.
- **base_sizes** (*list[int]*) –The basic sizes of anchors in multiple levels. If None is given, strides will be used to generate base_sizes.
- **scale_major** (*bool*) –Whether to multiply scales first when generating base anchors. If true, the anchors in the same row will have the same scales. By default it is True in V2.0
- **octave_base_scale** (*int*) –The base scale of octave.
- **scales_per_octave** (*int*) –Number of scales for each octave. *octave_base_scale* and *scales_per_octave* are usually used in retinanet and the *scales* should be None when they are set.
- **centers** (*list[tuple[float, float]] | None*) –The centers of the anchor relative to the feature grid center in multiple feature levels. By default it is set to be None and not used. It a list of float is given, this list will be used to shift the centers of anchors.
- **center_offset** (*float*) –The offset of center in propotion to anchors' width and height. By default it is 0.5 in V2.0 but it should be 0.5 in v1.x models.

实际案例

```
>>> from mmdet.core import LegacyAnchorGenerator
>>> self = LegacyAnchorGenerator(
>>>     [16], [1.], [1.], [9], center_offset=0.5)
>>> all_anchors = self.grid_anchors(((2, 2),), device='cpu')
>>> print(all_anchors)
[tensor([[ 0.,  0.,  8.,  8.],
        [16.,  0., 24.,  8.],
```

(下页继续)

(续上页)

```
        [ 0., 16.,  8., 24.],
        [16., 16., 24., 24.]])]
```

gen_single_level_base_anchors(*base_size*, *scales*, *ratios*, *center=None*)

Generate base anchors of a single level.

注解： The width/height of anchors are minused by 1 when calculating the centers and corners to meet the V1.x coordinate system.

参数

- **base_size** (*int | float*) –Basic size of an anchor.
- **scales** (*torch.Tensor*) –Scales of the anchor.
- **ratios** (*torch.Tensor*) –The ratio between between the height. and width of anchors in a single level.
- **center** (*tuple[float], optional*) –The center of the base anchor related to a single feature grid. Defaults to None.

返回 Anchors in a single-level feature map.

返回类型 torch.Tensor

class mmdet.core.anchor.**MlvlPointGenerator**(*strides*, *offset=0.5*)

Standard points generator for multi-level (Mlvl) feature maps in 2D points-based detectors.

参数

- **strides** (*list[int] | list[tuple[int, int]]*) –Strides of anchors in multiple feature levels in order (w, h).
- **offset** (*float*) –The offset of points, the value is normalized with corresponding stride. Defaults to 0.5.

grid_priors(*featmap_sizes*, *device='cuda'*, *with_stride=False*)

Generate grid points of multiple feature levels.

参数

- **featmap_sizes** (*list[tuple]*) –List of feature map sizes in multiple feature levels, each size arrange as as (h, w).
- **device** (*str*) –The device where the anchors will be put on.
- **with_stride** (*bool*) –Whether to concatenate the stride to the last dimension of points.

返回 Points of multiple feature levels. The sizes of each tensor should be (N, 2) when with stride is `False`, where N = width * height, width and height are the sizes of the corresponding feature level, and the last dimension 2 represent (coord_x, coord_y), otherwise the shape should be (N, 4), and the last dimension 4 represent (coord_x, coord_y, stride_w, stride_h).

返回类型 list[torch.Tensor]

property num_base_priors

The number of priors (points) at a point on the feature grid

Type list[int]

property num_levels

number of feature levels that the generator will be applied

Type int

single_level_grid_priors(*featmap_size*, *level_idx*, *device='cuda'*, *with_stride=False*)

Generate grid Points of a single level.

注解: This function is usually called by method `self.grid_priors`.

参数

- **featmap_size** (*tuple[int]*) –Size of the feature maps, arrange as (h, w).
- **level_idx** (*int*) –The index of corresponding feature map level.
- **device** (*str, optional*) –The device the tensor will be put on. Defaults to 'cuda'.
- **with_stride** (*bool*) –Concatenate the stride to the last dimension of points.

返回 Points of single feature levels. The shape of tensor should be (N, 2) when with stride is `False`, where N = width * height, width and height are the sizes of the corresponding feature level, and the last dimension 2 represent (coord_x, coord_y), otherwise the shape should be (N, 4), and the last dimension 4 represent (coord_x, coord_y, stride_w, stride_h).

返回类型 Tensor

single_level_valid_flags(*featmap_size*, *valid_size*, *device='cuda'*)

Generate the valid flags of points of a single feature map.

参数

- **featmap_size** (*tuple[int]*) –The size of feature maps, arrange as as (h, w).
- **valid_size** (*tuple[int]*) –The valid size of the feature maps. The size arrange as as (h, w).

- **device** (*str, optional*) –The device where the flags will be put on. Defaults to 'cuda'.

返回 The valid flags of each points in a single level feature map.

返回类型 torch.Tensor

sparse_priors(*prior_idxs*, *featmap_size*, *level_idx*, *dtype=torch.float32*, *device='cuda'*)

Generate sparse points according to the `prior_idxs`.

参数

- **prior_idxs** (*Tensor*) –The index of corresponding anchors in the feature map.
- **featmap_size** (*tuple[int]*) –feature map size arrange as (w, h).
- **level_idx** (*int*) –The level index of corresponding feature map.
- **(obj** (*device*) –*torch.dtype*): Date type of points. Defaults to `torch.float32`.
- **(obj** –*torch.device*): The device where the points is located.

返回 Anchor with shape (N, 2), N should be equal to the length of `prior_idxs`. And last dimension 2 represent (coord_x, coord_y).

返回类型 Tensor

valid_flags(*featmap_sizes*, *pad_shape*, *device='cuda'*)

Generate valid flags of points of multiple feature levels.

参数

- **featmap_sizes** (*list(tuple)*) –List of feature map sizes in multiple feature levels, each size arrange as as (h, w).
- **pad_shape** (*tuple(int)*) –The padded shape of the image, arrange as (h, w).
- **device** (*str*) –The device where the anchors will be put on.

返回 Valid flags of points of multiple levels.

返回类型 list(torch.Tensor)

class mmdet.core.anchor.**YOLOAnchorGenerator**(*strides*, *base_sizes*)

Anchor generator for YOLO.

参数

- **strides** (*list[int] | list[tuple[int, int]]*) –Strides of anchors in multiple feature levels.
- **base_sizes** (*list[list[tuple[int, int]]]*) –The basic sizes of anchors in multiple levels.

gen_base_anchors()

Generate base anchors.

返回 Base anchors of a feature grid in multiple feature levels.

返回类型 list(torch.Tensor)

gen_single_level_base_anchors(*base_sizes_per_level*, *center=None*)

Generate base anchors of a single level.

参数

- **base_sizes_per_level** (*list[tuple[int, int]]*) –Basic sizes of anchors.
- **center** (*tuple[float], optional*) –The center of the base anchor related to a single feature grid. Defaults to None.

返回 Anchors in a single-level feature maps.

返回类型 torch.Tensor

property num_levels

number of feature levels that the generator will be applied

Type int

responsible_flags(*featmap_sizes*, *gt_bboxes*, *device='cuda'*)

Generate responsible anchor flags of grid cells in multiple scales.

参数

- **featmap_sizes** (*list(tuple)*) –List of feature map sizes in multiple feature levels.
- **gt_bboxes** (*Tensor*) –Ground truth boxes, shape (n, 4).
- **device** (*str*) –Device where the anchors will be put on.

返回 responsible flags of anchors in multiple level

返回类型 list(torch.Tensor)

single_level_responsible_flags(*featmap_size*, *gt_bboxes*, *stride*, *num_base_anchors*, *device='cuda'*)

Generate the responsible flags of anchor in a single feature map.

参数

- **featmap_size** (*tuple[int]*) –The size of feature maps.
- **gt_bboxes** (*Tensor*) –Ground truth boxes, shape (n, 4).
- **stride** (*tuple(int)*) –stride of current level
- **num_base_anchors** (*int*) –The number of base anchors.
- **device** (*str, optional*) –Device where the flags will be put on. Defaults to 'cuda'.

返回 The valid flags of each anchor in a single level feature map.

返回类型 torch.Tensor

mmdet.core.anchor.**anchor_inside_flags**(*flat_anchors*, *valid_flags*, *img_shape*, *allowed_border=0*)

Check whether the anchors are inside the border.

参数

- **flat_anchors** (*torch.Tensor*) –Flatten anchors, shape (n, 4).
- **valid_flags** (*torch.Tensor*) –An existing valid flags of anchors.
- **img_shape** (*tuple(int)*) –Shape of current image.
- **allowed_border** (*int, optional*) –The border to allow the valid anchor. Defaults to 0.

返回 Flags indicating whether the anchors are inside a valid range.

返回类型 torch.Tensor

mmdet.core.anchor.**calc_region**(*bbox*, *ratio*, *featmap_size=None*)

Calculate a proportional bbox region.

The bbox center are fixed and the new h' and w' is h * ratio and w * ratio.

参数

- **bbox** (*Tensor*) –Bboxes to calculate regions, shape (n, 4).
- **ratio** (*float*) –Ratio of the output region.
- **featmap_size** (*tuple*) –Feature map size used for clipping the boundary.

返回 x1, y1, x2, y2

返回类型 tuple

mmdet.core.anchor.**images_to_levels**(*target*, *num_levels*)

Convert targets by image to targets by feature level.

[target_img0, target_img1] -> [target_level0, target_level1, ···]

22.2.2 bbox

class mmdet.core.bbox.**AssignResult**(*num_gts*, *gt_inds*, *max_overlaps*, *labels=None*)

Stores assignments between predicted and truth boxes.

num_gts

the number of truth boxes considered when computing this assignment

Type int

gt_inds
for each predicted box indicates the 1-based index of the assigned truth box. 0 means unassigned and -1 means ignore.

Type LongTensor

max_overlaps
the iou between the predicted box and its assigned truth box.

Type FloatTensor

labels
If specified, for each predicted box indicates the category label of the assigned truth box.

Type None | LongTensor

示例

```
>>> # An assign result between 4 predicted boxes and 9 true boxes
>>> # where only two boxes were assigned.
>>> num_gts = 9
>>> max_overlaps = torch.LongTensor([0, .5, .9, 0])
>>> gt_inds = torch.LongTensor([-1, 1, 2, 0])
>>> labels = torch.LongTensor([0, 3, 4, 0])
>>> self = AssignResult(num_gts, gt_inds, max_overlaps, labels)
>>> print(str(self))  # xdoctest: +IGNORE_WANT
<AssignResult(num_gts=9, gt_inds.shape=(4,), max_overlaps.shape=(4,),
              labels.shape=(4,))>
>>> # Force addition of gt labels (when adding gt as proposals)
>>> new_labels = torch.LongTensor([3, 4, 5])
>>> self.add_gt_(new_labels)
>>> print(str(self))  # xdoctest: +IGNORE_WANT
<AssignResult(num_gts=9, gt_inds.shape=(7,), max_overlaps.shape=(7,),
              labels.shape=(7,))>
```

add_gt_(*gt_labels*)
Add ground truth as assigned results.

参数 **gt_labels** (*torch.Tensor*) –Labels of gt boxes

get_extra_property(*key*)
Get user-defined property.

property info
a dictionary of info about the object

Type dict

property num_preds

the number of predictions in this assignment

Type int

classmethod random(***kwargs*)

Create random AssignResult for tests or debugging.

参数

- **num_preds** –number of predicted boxes
- **num_gts** –number of true boxes
- **p_ignore** (*float*) –probability of a predicted box assigned to an ignored truth
- **p_assigned** (*float*) –probability of a predicted box not being assigned
- **p_use_label** (*float | bool*) –with labels or not
- **rng** (*None | int | numpy.random.RandomState*) –seed or state

返回 Randomly generated assign results.

返回类型 *AssignResult*

示例

```
>>> from mmdet.core.bbox.assigners.assign_result import *  # NOQA
>>> self = AssignResult.random()
>>> print(self.info)
```

set_extra_property(*key*, *value*)

Set user-defined new property.

class mmdet.core.bbox.**BaseAssigner**

Base assigner that assigns boxes to ground truth boxes.

abstract assign(*bboxes*, *gt_bboxes*, *gt_bboxes_ignore=None*, *gt_labels=None*)

Assign boxes to either a ground truth boxes or a negative boxes.

class mmdet.core.bbox.**BaseBBoxCoder**(***kwargs*)

Base bounding box coder.

abstract decode(*bboxes*, *bboxes_pred*)

Decode the predicted bboxes according to prediction and base boxes.

abstract encode(*bboxes*, *gt_bboxes*)

Encode deltas between bboxes and ground truth boxes.

class mmdet.core.bbox.**BaseSampler**(*num*, *pos_fraction*, *neg_pos_ub=-1*, *add_gt_as_proposals=True*, ***kwargs*)

Base class of samplers.

sample(*assign_result*, *bboxes*, *gt_bboxes*, *gt_labels=None*, ***kwargs*)

Sample positive and negative bboxes.

This is a simple implementation of bbox sampling given candidates, assigning results and ground truth bboxes.

参数

- **assign_result** (*AssignResult*) –Bbox assigning results.
- **bboxes** (*Tensor*) –Boxes to be sampled from.
- **gt_bboxes** (*Tensor*) –Ground truth bboxes.
- **gt_labels** (*Tensor, optional*) –Class labels of ground truth bboxes.

返回 Sampling result.

返回类型 *SamplingResult*

示例

```
>>> from mmdet.core.bbox import RandomSampler
>>> from mmdet.core.bbox import AssignResult
>>> from mmdet.core.bbox.demodata import ensure_rng, random_boxes
>>> rng = ensure_rng(None)
>>> assign_result = AssignResult.random(rng=rng)
>>> bboxes = random_boxes(assign_result.num_preds, rng=rng)
>>> gt_bboxes = random_boxes(assign_result.num_gts, rng=rng)
>>> gt_labels = None
>>> self = RandomSampler(num=32, pos_fraction=0.5, neg_pos_ub=-1,
>>>                      add_gt_as_proposals=False)
>>> self = self.sample(assign_result, bboxes, gt_bboxes, gt_labels)
```

class mmdet.core.bbox.**BboxOverlaps2D**(*scale=1.0*, *dtype=None*)

2D Overlaps (e.g. IoUs, GIoUs) Calculator.

class mmdet.core.bbox.**CenterRegionAssigner**(*pos_scale*, *neg_scale*, *min_pos_iof=0.01*, *ignore_gt_scale=0.5*, *foreground_dominate=False*, *iou_calculator={'type': 'BboxOverlaps2D'}*)

Assign pixels at the center region of a bbox as positive.

Each proposals will be assigned with *-1*, *0*, or a positive integer indicating the ground truth index. - -1: negative samples - semi-positive numbers: positive sample, index (0-based) of assigned gt

参数

- **pos_scale** (*float*) –Threshold within which pixels are labelled as positive.
- **neg_scale** (*float*) –Threshold above which pixels are labelled as positive.
- **min_pos_iof** (*float*) –Minimum iof of a pixel with a gt to be labelled as positive. Default: 1e-2
- **ignore_gt_scale** (*float*) –Threshold within which the pixels are ignored when the gt is labelled as shadowed. Default: 0.5
- **foreground_dominate** (*bool*) –If True, the bbox will be assigned as positive when a gt's kernel region overlaps with another's shadowed (ignored) region, otherwise it is set as ignored. Default to False.

assign(*bboxes*, *gt_bboxes*, *gt_bboxes_ignore=None*, *gt_labels=None*)

Assign gt to bboxes.

This method assigns gts to every bbox (proposal/anchor), each bbox will be assigned with -1, or a semi-positive number. -1 means negative sample, semi-positive number is the index (0-based) of assigned gt.

参数

- **bboxes** (*Tensor*) –Bounding boxes to be assigned, shape(n, 4).
- **gt_bboxes** (*Tensor*) –Groundtruth boxes, shape (k, 4).
- **gt_bboxes_ignore** (*tensor, optional*) –Ground truth bboxes that are labelled as *ignored*, e.g., crowd boxes in COCO.
- **gt_labels** (*tensor, optional*) –Label of gt_bboxes, shape (num_gts,).

返回 The assigned result. Note that shadowed_labels of shape (N, 2) is also added as an *assign_result* attribute. *shadowed_labels* is a tensor composed of N pairs of anchor_ind, class_label], where N is the number of anchors that lie in the outer region of a gt, anchor_ind is the shadowed anchor index and class_label is the shadowed class label.

返回类型 *AssignResult*

示例

```
>>> self = CenterRegionAssigner(0.2, 0.2)
>>> bboxes = torch.Tensor([[0, 0, 10, 10], [10, 10, 20, 20]])
>>> gt_bboxes = torch.Tensor([[0, 0, 10, 10]])
>>> assign_result = self.assign(bboxes, gt_bboxes)
>>> expected_gt_inds = torch.LongTensor([1, 0])
>>> assert torch.all(assign_result.gt_inds == expected_gt_inds)
```

assign_one_hot_gt_indices(*is_bbox_in_gt_core*, *is_bbox_in_gt_shadow*, *gt_priority=None*)

Assign only one gt index to each prior box.

Gts with large gt_priority are more likely to be assigned.

参数

- **is_bbox_in_gt_core** (*Tensor*) –Bool tensor indicating the bbox center is in the core area of a gt (e.g. 0-0.2). Shape: (num_prior, num_gt).
- **is_bbox_in_gt_shadow** (*Tensor*) –Bool tensor indicating the bbox center is in the shadowed area of a gt (e.g. 0.2-0.5). Shape: (num_prior, num_gt).
- **gt_priority** (*Tensor*) –Priorities of gts. The gt with a higher priority is more likely to be assigned to the bbox when the bbox match with multiple gts. Shape: (num_gt,).

返回

Returns (assigned_gt_inds, shadowed_gt_inds).

- assigned_gt_inds: The assigned gt index of each prior bbox (i.e. index from 1 to num_gts). Shape: (num_prior,).
- shadowed_gt_inds: shadowed gt indices. It is a tensor of shape (num_ignore, 2) with first column being the shadowed prior bbox indices and the second column the shadowed gt indices (1-based).

返回类型 tuple

get_gt_priorities(*gt_bboxes*)

Get gt priorities according to their areas.

Smaller gt has higher priority.

参数 **gt_bboxes** (*Tensor*) –Ground truth boxes, shape (k, 4).

返回 The priority of gts so that gts with larger priority is more likely to be assigned. Shape (k,)

返回类型 Tensor

class mmdet.core.bbox.**CombinedSampler**(*pos_sampler*, *neg_sampler*, ***kwargs*)

A sampler that combines positive sampler and negative sampler.

class mmdet.core.bbox.**DeltaXYWHBBoxCoder**(*target_means=(0.0, 0.0, 0.0, 0.0)*, *target_stds=(1.0, 1.0, 1.0, 1.0)*, *clip_border=True*, *add_ctr_clamp=False*, *ctr_clamp=32*)

Delta XYWH BBox coder.

Following the practice in R-CNN, this coder encodes bbox (x1, y1, x2, y2) into delta (dx, dy, dw, dh) and decodes delta (dx, dy, dw, dh) back to original bbox (x1, y1, x2, y2).

参数

- **target_means** (*Sequence[float]*) –Denormalizing means of target for delta coordinates

- **target_stds** (*Sequence[float]*) –Denormalizing standard deviation of target for delta coordinates
- **clip_border** (*bool, optional*) –Whether clip the objects outside the border of the image. Defaults to True.
- **add_ctr_clamp** (*bool*) –Whether to add center clamp, when added, the predicted box is clamped is its center is too far away from the original anchor's center. Only used by YOLOF. Default False.
- **ctr_clamp** (*int*) –the maximum pixel shift to clamp. Only used by YOLOF. Default 32.

decode(*bboxes*, *pred_bboxes*, *max_shape=None*, *wh_ratio_clip=0.016*)

Apply transformation *pred_bboxes* to *boxes*.

参数

- **bboxes** (*torch.Tensor*) –Basic boxes. Shape (B, N, 4) or (N, 4)
- **pred_bboxes** (*Tensor*) –Encoded offsets with respect to each roi. Has shape (B, N, num_classes * 4) or (B, N, 4) or (N, num_classes * 4) or (N, 4). Note N = num_anchors * W * H when rois is a grid of anchors.Offset encoding follows[1].
- **Sequence[** (*max_shape (Sequence[int] or torch.Tensor or*) –Sequence[int]],optional): Maximum bounds for boxes, specifies (H, W, C) or (H, W). If bboxes shape is (B, N, 4), then the max_shape should be a Sequence[Sequence[int]] and the length of max_shape should also be B.
- **wh_ratio_clip** (*float, optional*) –The allowed ratio between width and height.

返回 Decoded boxes.

返回类型 torch.Tensor

encode(*bboxes*, *gt_bboxes*)

Get box regression transformation deltas that can be used to transform the `bboxes` into the `gt_bboxes`.

参数

- **bboxes** (*torch.Tensor*) –Source boxes, e.g., object proposals.
- **gt_bboxes** (*torch.Tensor*) –Target of the transformation, e.g., ground-truth boxes.

返回 Box transformation deltas

返回类型 torch.Tensor

class mmdet.core.bbox.**InstanceBalancedPosSampler**(*num*, *pos_fraction*, *neg_pos_ub=-1*, *add_gt_as_proposals=True*, ***kwargs*)

Instance balanced sampler that samples equal number of positive samples for each instance.

[1] https://gitlab.kitware.com/computer-vision/kwimage/-/blob/928cae35ca8/kwimage/structs/polygon.py#L379 # noqa: E501

class mmdet.core.bbox.**IoUBalancedNegSampler**(*num*, *pos_fraction*, *floor_thr=- 1*, *floor_fraction=0*, *num_bins=3*, ***kwargs*)

IoU Balanced Sampling.

arXiv: https://arxiv.org/pdf/1904.02701.pdf (CVPR 2019)

Sampling proposals according to their IoU. *floor_fraction* of needed RoIs are sampled from proposals whose IoU are lower than *floor_thr* randomly. The others are sampled from proposals whose IoU are higher than *floor_thr*. These proposals are sampled from some bins evenly, which are split by *num_bins* via IoU evenly.

参数

- **num** (*int*) –number of proposals.
- **pos_fraction** (*float*) –fraction of positive proposals.
- **floor_thr** (*float*) –threshold (minimum) IoU for IoU balanced sampling, set to -1 if all using IoU balanced sampling.
- **floor_fraction** (*float*) –sampling fraction of proposals under floor_thr.
- **num_bins** (*int*) –number of bins in IoU balanced sampling.

sample_via_interval(*max_overlaps*, *full_set*, *num_expected*)

Sample according to the iou interval.

参数

- **max_overlaps** (*torch.Tensor*) –IoU between bounding boxes and ground truth boxes.
- **full_set** (*set(int)*) –A full set of indices of boxes。
- **num_expected** (*int*) –Number of expected samples。

返回 Indices of samples

返回类型 np.ndarray

class mmdet.core.bbox.**MaxIoUAssigner**(*pos_iou_thr*, *neg_iou_thr*, *min_pos_iou=0.0*, *gt_max_assign_all=True*, *ignore_iof_thr=- 1*, *ignore_wrt_candidates=True*, *match_low_quality=True*, *gpu_assign_thr=- 1*, *iou_calculator={'type': 'BboxOverlaps2D'}*)

Assign a corresponding gt bbox or background to each bbox.

Each proposals will be assigned with *-1*, or a semi-positive integer indicating the ground truth index.

- -1: negative sample, no assigned gt
- semi-positive integer: positive sample, index (0-based) of assigned gt

参数

- **pos_iou_thr** (*float*) –IoU threshold for positive bboxes.
- **neg_iou_thr** (*float or tuple*) –IoU threshold for negative bboxes.
- **min_pos_iou** (*float*) –Minimum iou for a bbox to be considered as a positive bbox. Positive samples can have smaller IoU than pos_iou_thr due to the 4th step (assign max IoU sample to each gt).
- **gt_max_assign_all** (*bool*) –Whether to assign all bboxes with the same highest overlap with some gt to that gt.
- **ignore_iof_thr** (*float*) –IoF threshold for ignoring bboxes (if *gt_bboxes_ignore* is specified). Negative values mean not ignoring any bboxes.
- **ignore_wrt_candidates** (*bool*) –Whether to compute the iof between *bboxes* and *gt_bboxes_ignore*, or the contrary.
- **match_low_quality** (*bool*) –Whether to allow low quality matches. This is usually allowed for RPN and single stage detectors, but not allowed in the second stage. Details are demonstrated in Step 4.
- **gpu_assign_thr** (*int*) –The upper bound of the number of GT for GPU assign. When the number of gt is above this threshold, will assign on CPU device. Negative values mean not assign on CPU.

assign(*bboxes*, *gt_bboxes*, *gt_bboxes_ignore=None*, *gt_labels=None*)

Assign gt to bboxes.

This method assign a gt bbox to every bbox (proposal/anchor), each bbox will be assigned with -1, or a semi-positive number. -1 means negative sample, semi-positive number is the index (0-based) of assigned gt. The assignment is done in following steps, the order matters.

1. assign every bbox to the background
2. assign proposals whose iou with all gts < neg_iou_thr to 0
3. for each bbox, if the iou with its nearest gt >= pos_iou_thr, assign it to that bbox
4. for each gt bbox, assign its nearest proposals (may be more than one) to itself

参数

- **bboxes** (*Tensor*) –Bounding boxes to be assigned, shape(n, 4).
- **gt_bboxes** (*Tensor*) –Groundtruth boxes, shape (k, 4).
- **gt_bboxes_ignore** (*Tensor, optional*) –Ground truth bboxes that are labelled as *ignored*, e.g., crowd boxes in COCO.
- **gt_labels** (*Tensor, optional*) –Label of gt_bboxes, shape (k,).

返回 The assign result.

返回类型 *AssignResult*

示例

```
>>> self = MaxIoUAssigner(0.5, 0.5)
>>> bboxes = torch.Tensor([[0, 0, 10, 10], [10, 10, 20, 20]])
>>> gt_bboxes = torch.Tensor([[0, 0, 10, 9]])
>>> assign_result = self.assign(bboxes, gt_bboxes)
>>> expected_gt_inds = torch.LongTensor([1, 0])
>>> assert torch.all(assign_result.gt_inds == expected_gt_inds)
```

assign_wrt_overlaps(*overlaps*, *gt_labels=None*)

Assign w.r.t. the overlaps of bboxes with gts.

参数

- **overlaps** (*Tensor*) –Overlaps between k gt_bboxes and n bboxes, shape(k, n).
- **gt_labels** (*Tensor, optional*) –Labels of k gt_bboxes, shape (k,).

返回 The assign result.

返回类型 *AssignResult*

class mmdet.core.bbox.**OHEMSampler**(*num*, *pos_fraction*, *context*, *neg_pos_ub=-1*, *add_gt_as_proposals=True*, ***kwargs*)

Online Hard Example Mining Sampler described in Training Region-based Object Detectors with Online Hard Example Mining.

class mmdet.core.bbox.**PseudoBBoxCoder**(***kwargs*)

Pseudo bounding box coder.

decode(*bboxes*, *pred_bboxes*)

torch.Tensor: return the given `pred_bboxes`

encode(*bboxes*, *gt_bboxes*)

torch.Tensor: return the given `bboxes`

class mmdet.core.bbox.**PseudoSampler**(***kwargs*)

A pseudo sampler that does not do sampling actually.

sample(*assign_result*, *bboxes*, *gt_bboxes*, ***kwargs*)

Directly returns the positive and negative indices of samples.

参数

- **assign_result** (*AssignResult*) –Assigned results
- **bboxes** (*torch.Tensor*) –Bounding boxes
- **gt_bboxes** (*torch.Tensor*) –Ground truth boxes

返回 sampler results

返回类型 *SamplingResult*

class mmdet.core.bbox.**RandomSampler**(*num*, *pos_fraction*, *neg_pos_ub=-1*, *add_gt_as_proposals=True*, ***kwargs*)

Random sampler.

参数

- **num** (*int*) –Number of samples
- **pos_fraction** (*float*) –Fraction of positive samples
- **neg_pos_up** (*int, optional*) –Upper bound number of negative and positive samples. Defaults to -1.
- **add_gt_as_proposals** (*bool, optional*) –Whether to add ground truth boxes as proposals. Defaults to True.

random_choice(*gallery*, *num*)

Random select some elements from the gallery.

If *gallery* is a Tensor, the returned indices will be a Tensor; If *gallery* is a ndarray or list, the returned indices will be a ndarray.

参数

- **gallery** (*Tensor | ndarray | list*) –indices pool.
- **num** (*int*) –expected sample num.

返回 sampled indices.

返回类型 Tensor or ndarray

class mmdet.core.bbox.**RegionAssigner**(*center_ratio=0.2*, *ignore_ratio=0.5*)

Assign a corresponding gt bbox or background to each bbox.

Each proposals will be assigned with *-1*, *0*, or a positive integer indicating the ground truth index.

- -1: don’t care
- 0: negative sample, no assigned gt
- positive integer: positive sample, index (1-based) of assigned gt

参数

- **center_ratio** –ratio of the region in the center of the bbox to define positive sample.
- **ignore_ratio** –ratio of the region to define ignore samples.

assign(*mlvl_anchors*, *mlvl_valid_flags*, *gt_bboxes*, *img_meta*, *featmap_sizes*, *anchor_scale*, *anchor_strides*, *gt_bboxes_ignore=None*, *gt_labels=None*, *allowed_border=0*)

Assign gt to anchors.

This method assign a gt bbox to every bbox (proposal/anchor), each bbox will be assigned with -1, 0, or a positive number. -1 means don't care, 0 means negative sample, positive number is the index (1-based) of assigned gt.

The assignment is done in following steps, and the order matters.

1. Assign every anchor to 0 (negative)
2. (For each gt_bboxes) Compute ignore flags based on ignore_region then assign -1 to anchors w.r.t. ignore flags
3. (For each gt_bboxes) Compute pos flags based on center_region then assign gt_bboxes to anchors w.r.t. pos flags
4. (For each gt_bboxes) Compute ignore flags based on adjacent anchor level then assign -1 to anchors w.r.t. ignore flags
5. Assign anchor outside of image to -1

参数

- **mlvl_anchors** (*list[Tensor]*) –Multi level anchors.
- **mlvl_valid_flags** (*list[Tensor]*) –Multi level valid flags.
- **gt_bboxes** (*Tensor*) –Ground truth bboxes of image
- **img_meta** (*dict*) –Meta info of image.
- **featmap_sizes** (*list[Tensor]*) –Feature mapsize each level
- **anchor_scale** (*int*) –Scale of the anchor.
- **anchor_strides** (*list[int]*) –Stride of the anchor.
- **gt_bboxes** –Groundtruth boxes, shape (k, 4).
- **gt_bboxes_ignore** (*Tensor, optional*) –Ground truth bboxes that are labelled as *ignored*, e.g., crowd boxes in COCO.
- **gt_labels** (*Tensor, optional*) –Label of gt_bboxes, shape (k,).
- **allowed_border** (*int, optional*) –The border to allow the valid anchor. Defaults to 0.

返回 The assign result.

返回类型 *AssignResult*

class mmdet.core.bbox.**SamplingResult**(*pos_inds*, *neg_inds*, *bboxes*, *gt_bboxes*, *assign_result*, *gt_flags*)

Bbox sampling result.

示例

```
>>> # xdoctest: +IGNORE_WANT
>>> from mmdet.core.bbox.samplers.sampling_result import *  # NOQA
>>> self = SamplingResult.random(rng=10)
>>> print(f'self = {self}')
self = <SamplingResult({
    'neg_bboxes': torch.Size([12, 4]),
    'neg_inds': tensor([ 0,  1,  2,  4,  5,  6,  7,  8,  9, 10, 11, 12]),
    'num_gts': 4,
    'pos_assigned_gt_inds': tensor([], dtype=torch.int64),
    'pos_bboxes': torch.Size([0, 4]),
    'pos_inds': tensor([], dtype=torch.int64),
    'pos_is_gt': tensor([], dtype=torch.uint8)
})>
```

property bboxes

concatenated positive and negative boxes

Type torch.Tensor

property info

Returns a dictionary of info about the object.

classmethod random(*rng=None*, ***kwargs*)

参数

- **rng** (*None | int | numpy.random.RandomState*) –seed or state.
- **kwargs** (*keyword arguments*) –
  - num_preds: number of predicted boxes
  - num_gts: number of true boxes
  - p_ignore (float): probability of a predicted box assigned to an ignored truth.
  - p_assigned (float): probability of a predicted box not being assigned.
  - p_use_label (float | bool): with labels or not.

返回 Randomly generated sampling result.

返回类型 *SamplingResult*

示例

```
>>> from mmdet.core.bbox.samplers.sampling_result import *  # NOQA
>>> self = SamplingResult.random()
>>> print(self.__dict__)
```

to(*device*)

Change the device of the data inplace.

示例

```
>>> self = SamplingResult.random()
>>> print(f'self = {self.to(None)}')
>>> # xdoctest: +REQUIRES(--gpu)
>>> print(f'self = {self.to(0)}')
```

class mmdet.core.bbox.**ScoreHLRSampler**(*num*, *pos_fraction*, *context*, *neg_pos_ub=- 1*, *add_gt_as_proposals=True*, *k=0.5*, *bias=0*, *score_thr=0.05*, *iou_thr=0.5*, ***kwargs*)

Importance-based Sample Reweighting (ISR_N), described in Prime Sample Attention in Object Detection.

Score hierarchical local rank (HLR) differentiates with RandomSampler in negative part. It firstly computes Score-HLR in a two-step way, then linearly maps score hlr to the loss weights.

参数

- **num** (*int*) –Total number of sampled RoIs.
- **pos_fraction** (*float*) –Fraction of positive samples.
- **context** (BaseRoIHead) –RoI head that the sampler belongs to.
- **neg_pos_ub** (*int*) –Upper bound of the ratio of num negative to num positive, -1 means no upper bound.
- **add_gt_as_proposals** (*bool*) –Whether to add ground truth as proposals.
- **k** (*float*) –Power of the non-linear mapping.
- **bias** (*float*) –Shift of the non-linear mapping.
- **score_thr** (*float*) –Minimum score that a negative sample is to be considered as valid bbox.

static random_choice(*gallery*, *num*)

Randomly select some elements from the gallery.

If *gallery* is a Tensor, the returned indices will be a Tensor; If *gallery* is a ndarray or list, the returned indices will be a ndarray.

参数

- **gallery** (*Tensor | ndarray | list*) –indices pool.
- **num** (*int*) –expected sample num.

返回 sampled indices.

返回类型 Tensor or ndarray

sample(*assign_result*, *bboxes*, *gt_bboxes*, *gt_labels=None*, *img_meta=None*, ***kwargs*)

Sample positive and negative bboxes.

This is a simple implementation of bbox sampling given candidates, assigning results and ground truth bboxes.

参数

- **assign_result** (*AssignResult*) –Bbox assigning results.
- **bboxes** (*Tensor*) –Boxes to be sampled from.
- **gt_bboxes** (*Tensor*) –Ground truth bboxes.
- **gt_labels** (*Tensor, optional*) –Class labels of ground truth bboxes.

返回

Sampling result and negative label weights.

返回类型 tuple[*SamplingResult*, Tensor]

class mmdet.core.bbox.**TBLRBBoxCoder**(*normalizer=4.0*, *clip_border=True*)

TBLR BBox coder.

Following the practice in FSAF, this coder encodes gt bboxes (x1, y1, x2, y2) into (top, bottom, left, right) and decode it back to the original.

参数

- **normalizer** (*list | float*) –Normalization factor to be divided with when coding the coordinates. If it is a list, it should have length of 4 indicating normalization factor in tblr dims. Otherwise it is a unified float factor for all dims. Default: 4.0
- **clip_border** (*bool, optional*) –Whether clip the objects outside the border of the image. Defaults to True.

decode(*bboxes*, *pred_bboxes*, *max_shape=None*)

Apply transformation *pred_bboxes* to *boxes*.

参数

- **bboxes** (*torch.Tensor*) –Basic boxes.Shape (B, N, 4) or (N, 4)
- **pred_bboxes** (*torch.Tensor*) –Encoded boxes with shape (B, N, 4) or (N, 4)

- **Sequence[** (*max_shape (Sequence[int] or torch.Tensor or*) –Sequence[int]],optional): Maximum bounds for boxes, specifies (H, W, C) or (H, W). If bboxes shape is (B, N, 4), then the max_shape should be a Sequence[Sequence[int]] and the length of max_shape should also be B.

返回 Decoded boxes.

返回类型 torch.Tensor

encode(*bboxes*, *gt_bboxes*)

Get box regression transformation deltas that can be used to transform the `bboxes` into the `gt_bboxes` in the (top, left, bottom, right) order.

参数

- **bboxes** (*torch.Tensor*) –source boxes, e.g., object proposals.
- **gt_bboxes** (*torch.Tensor*) –target of the transformation, e.g., ground truth boxes.

返回 Box transformation deltas

返回类型 torch.Tensor

mmdet.core.bbox.**bbox2distance**(*points*, *bbox*, *max_dis=None*, *eps=0.1*)

Decode bounding box based on distances.

参数

- **points** (*Tensor*) –Shape (n, 2), [x, y].
- **bbox** (*Tensor*) –Shape (n, 4), "xyxy" format
- **max_dis** (*float*) –Upper bound of the distance.
- **eps** (*float*) –a small value to ensure target < max_dis, instead <=

返回 Decoded distances.

返回类型 Tensor

mmdet.core.bbox.**bbox2result**(*bboxes*, *labels*, *num_classes*)

Convert detection results to a list of numpy arrays.

参数

- **bboxes** (*torch.Tensor | np.ndarray*) –shape (n, 5)
- **labels** (*torch.Tensor | np.ndarray*) –shape (n,)
- **num_classes** (*int*) –class number, including background class

返回 bbox results of each class

返回类型 list(ndarray)

mmdet.core.bbox.**bbox2roi**(*bbox_list*)

Convert a list of bboxes to roi format.

参数 **bbox_list** (*list[Tensor]*) –a list of bboxes corresponding to a batch of images.

返回 shape (n, 5), [batch_ind, x1, y1, x2, y2]

返回类型 Tensor

mmdet.core.bbox.**bbox_cxcywh_to_xyxy**(*bbox*)

Convert bbox coordinates from (cx, cy, w, h) to (x1, y1, x2, y2).

参数 **bbox** (*Tensor*) –Shape (n, 4) for bboxes.

返回 Converted bboxes.

返回类型 Tensor

mmdet.core.bbox.**bbox_flip**(*bboxes*, *img_shape*, *direction='horizontal'*)

Flip bboxes horizontally or vertically.

参数

- **bboxes** (*Tensor*) –Shape (…, 4*k)
- **img_shape** (*tuple*) –Image shape.
- **direction** (*str*) –Flip direction, options are "horizontal", "vertical", "diagonal". Default: "horizontal"

返回 Flipped bboxes.

返回类型 Tensor

mmdet.core.bbox.**bbox_mapping**(*bboxes*, *img_shape*, *scale_factor*, *flip*, *flip_direction='horizontal'*)

Map bboxes from the original image scale to testing scale.

mmdet.core.bbox.**bbox_mapping_back**(*bboxes*, *img_shape*, *scale_factor*, *flip*, *flip_direction='horizontal'*)

Map bboxes from testing scale to original image scale.

mmdet.core.bbox.**bbox_overlaps**(*bboxes1*, *bboxes2*, *mode='iou'*, *is_aligned=False*, *eps=1e-06*)

Calculate overlap between two set of bboxes.

FP16 Contributed by https://github.com/open-mmlab/mmdetection/pull/4889 .. note:

```
Assume bboxes1 is M x 4, bboxes2 is N x 4, when mode is 'iou',
there are some new generated variable when calculating IOU
using bbox_overlaps function:

1) is_aligned is False
    area1: M x 1
    area2: N x 1
    lt: M x N x 2
```

(下页继续)

(续上页)

```
    rb: M x N x 2
    wh: M x N x 2
    overlap: M x N x 1
    union: M x N x 1
    ious: M x N x 1

    Total memory:
        S = (9 x N x M + N + M) * 4 Byte,

    When using FP16, we can reduce:
        R = (9 x N x M + N + M) * 4 / 2 Byte
        R large than (N + M) * 4 * 2 is always true when N and M >= 1.
        Obviously, N + M <= N * M < 3 * N * M, when N >=2 and M >=2,
                   N + 1 < 3 * N, when N or M is 1.

    Given M = 40 (ground truth), N = 400000 (three anchor boxes
    in per grid, FPN, R-CNNs),
        R = 275 MB (one times)

    A special case (dense detection), M = 512 (ground truth),
        R = 3516 MB = 3.43 GB

    When the batch size is B, reduce:
        B x R

    Therefore, CUDA memory runs out frequently.

    Experiments on GeForce RTX 2080Ti (11019 MiB):

    |   dtype   |   M   |   N   |   Use    |   Real   |   Ideal   |
    |:----:|:----:|:----:|:----:|:----:|:----:|
    |   FP32   |   512 | 400000 | 8020 MiB |   --   |   --   |
    |   FP16   |   512 | 400000 |   4504 MiB | 3516 MiB | 3516 MiB |
    |   FP32   |   40 | 400000 |   1540 MiB |   --   |   --   |
    |   FP16   |   40 | 400000 |   1264 MiB |   276MiB   | 275 MiB |

2) is_aligned is True
    area1: N x 1
    area2: N x 1
    lt: N x 2
    rb: N x 2
    wh: N x 2
    overlap: N x 1
```

(下页继续)

(续上页)

```
    union: N x 1
    ious: N x 1

    Total memory:
        S = 11 x N * 4 Byte

    When using FP16, we can reduce:
        R = 11 x N * 4 / 2 Byte

So do the 'giou' (large than 'iou').

Time-wise, FP16 is generally faster than FP32.

When gpu_assign_thr is not -1, it takes more time on cpu
but not reduce memory.
There, we can reduce half the memory and keep the speed.
```

If `is_aligned` is `False`, then calculate the overlaps between each bbox of bboxes1 and bboxes2, otherwise the overlaps between each aligned pair of bboxes1 and bboxes2.

参数

- **bboxes1** (*Tensor*) –shape (B, m, 4) in <x1, y1, x2, y2> format or empty.
- **bboxes2** (*Tensor*) –shape (B, n, 4) in <x1, y1, x2, y2> format or empty. B indicates the batch dim, in shape (B1, B2, ···, Bn). If `is_aligned` is `True`, then m and n must be equal.
- **mode** (*str*) –"iou" (intersection over union), "iof" (intersection over foreground) or "giou" (generalized intersection over union). Default "iou".
- **is_aligned** (*bool, optional*) –If True, then m and n must be equal. Default False.
- **eps** (*float, optional*) –A value added to the denominator for numerical stability. Default 1e-6.

返回 shape (m, n) if `is_aligned` is False else shape (m,)

返回类型 Tensor

示例

```
>>> bboxes1 = torch.FloatTensor([
>>>     [0, 0, 10, 10],
>>>     [10, 10, 20, 20],
>>>     [32, 32, 38, 42],
>>> ])
```

(下页继续)

(续上页)

```
>>> bboxes2 = torch.FloatTensor([
>>>     [0, 0, 10, 20],
>>>     [0, 10, 10, 19],
>>>     [10, 10, 20, 20],
>>> ])
>>> overlaps = bbox_overlaps(bboxes1, bboxes2)
>>> assert overlaps.shape == (3, 3)
>>> overlaps = bbox_overlaps(bboxes1, bboxes2, is_aligned=True)
>>> assert overlaps.shape == (3, )
```

示例

```
>>> empty = torch.empty(0, 4)
>>> nonempty = torch.FloatTensor([[0, 0, 10, 9]])
>>> assert tuple(bbox_overlaps(empty, nonempty).shape) == (0, 1)
>>> assert tuple(bbox_overlaps(nonempty, empty).shape) == (1, 0)
>>> assert tuple(bbox_overlaps(empty, empty).shape) == (0, 0)
```

mmdet.core.bbox.**bbox_rescale**(*bboxes*, *scale_factor=1.0*)

Rescale bounding box w.r.t. scale_factor.

参数

- **bboxes** (*Tensor*) –Shape (n, 4) for bboxes or (n, 5) for rois
- **scale_factor** (*float*) –rescale factor

返回 Rescaled bboxes.

返回类型 Tensor

mmdet.core.bbox.**bbox_xyxy_to_cxcywh**(*bbox*)

Convert bbox coordinates from (x1, y1, x2, y2) to (cx, cy, w, h).

参数 **bbox** (*Tensor*) –Shape (n, 4) for bboxes.

返回 Converted bboxes.

返回类型 Tensor

mmdet.core.bbox.**build_assigner**(*cfg*, ***default_args*)

Builder of box assigner.

mmdet.core.bbox.**build_bbox_coder**(*cfg*, ***default_args*)

Builder of box coder.

mmdet.core.bbox.**build_sampler**(*cfg*, ***default_args*)

Builder of box sampler.

mmdet.core.bbox.**distance2bbox**(*points*, *distance*, *max_shape=None*)

Decode distance prediction to bounding box.

参数

- **points** (*Tensor*) –Shape (B, N, 2) or (N, 2).
- **distance** (*Tensor*) –Distance from the given point to 4 boundaries (left, top, right, bottom). Shape (B, N, 4) or (N, 4)
- **Sequence[** (*max_shape (Sequence[int] or torch.Tensor or*) –Sequence[int]],optional): Maximum bounds for boxes, specifies (H, W, C) or (H, W). If priors shape is (B, N, 4), then the max_shape should be a Sequence[Sequence[int]] and the length of max_shape should also be B.

返回 Boxes with shape (N, 4) or (B, N, 4)

返回类型 Tensor

mmdet.core.bbox.**roi2bbox**(*rois*)

Convert rois to bounding box format.

参数 **rois** (*torch.Tensor*) –RoIs with the shape (n, 5) where the first column indicates batch id of each RoI.

返回 Converted boxes of corresponding rois.

返回类型 list[torch.Tensor]

22.2.3 export

22.2.4 mask

class mmdet.core.mask.**BaseInstanceMasks**

Base class for instance masks.

abstract property areas

areas of each instance.

Type ndarray

abstract crop(*bbox*)

Crop each mask by the given bbox.

参数 **bbox** (*ndarray*) –Bbox in format [x1, y1, x2, y2], shape (4,).

返回 The cropped masks.

返回类型 *BaseInstanceMasks*

abstract crop_and_resize(*bboxes*, *out_shape*, *inds*, *device*, *interpolation='bilinear'*, *binarize=True*)

Crop and resize masks by the given bboxes.

This function is mainly used in mask targets computation. It firstly align mask to bboxes by assigned_inds, then crop mask by the assigned bbox and resize to the size of (mask_h, mask_w)

参数

- **bboxes** (*Tensor*) –Bboxes in format [x1, y1, x2, y2], shape (N, 4)
- **out_shape** (*tuple[int]*) –Target (h, w) of resized mask
- **inds** (*ndarray*) –Indexes to assign masks to each bbox, shape (N,) and values should be between [0, num_masks - 1].
- **device** (*str*) –Device of bboxes
- **interpolation** (*str*) –See *mmcv.imresize*
- **binarize** (*bool*) –if True fractional values are rounded to 0 or 1 after the resize operation. if False and unsupported an error will be raised. Defaults to True.

返回 the cropped and resized masks.

返回类型 *BaseInstanceMasks*

abstract expand(*expanded_h*, *expanded_w*, *top*, *left*)

see `Expand`.

abstract flip(*flip_direction='horizontal'*)

Flip masks alone the given direction.

参数 **flip_direction** (*str*) –Either ‘horizontal’ or ‘vertical’.

返回 The flipped masks.

返回类型 *BaseInstanceMasks*

abstract pad(*out_shape*, *pad_val*)

Pad masks to the given size of (h, w).

参数

- **out_shape** (*tuple[int]*) –Target (h, w) of padded mask.
- **pad_val** (*int*) –The padded value.

返回 The padded masks.

返回类型 *BaseInstanceMasks*

abstract rescale(*scale*, *interpolation='nearest'*)

Rescale masks as large as possible while keeping the aspect ratio. For details can refer to *mmcv.imrescale*.

参数

- **scale** (*tuple[int]*) –The maximum size (h, w) of rescaled mask.
- **interpolation** (*str*) –Same as `mmcv.imrescale()`.

返回 The rescaled masks.

返回类型 *BaseInstanceMasks*

abstract resize(*out_shape*, *interpolation='nearest'*)

Resize masks to the given out_shape.

参数

- **out_shape** –Target (h, w) of resized mask.
- **interpolation** (*str*) –See `mmcv.imresize()`.

返回 The resized masks.

返回类型 *BaseInstanceMasks*

abstract rotate(*out_shape*, *angle*, *center=None*, *scale=1.0*, *fill_val=0*)

Rotate the masks.

参数

- **out_shape** (*tuple[int]*) –Shape for output mask, format (h, w).
- **angle** (*int | float*) –Rotation angle in degrees. Positive values mean counter-clockwise rotation.
- **center** (*tuple[float], optional*) –Center point (w, h) of the rotation in source image. If not specified, the center of the image will be used.
- **scale** (*int | float*) –Isotropic scale factor.
- **fill_val** (*int | float*) –Border value. Default 0 for masks.

返回 Rotated masks.

shear(*out_shape*, *magnitude*, *direction='horizontal'*, *border_value=0*, *interpolation='bilinear'*)

Shear the masks.

参数

- **out_shape** (*tuple[int]*) –Shape for output mask, format (h, w).
- **magnitude** (*int | float*) –The magnitude used for shear.
- **direction** (*str*) –The shear direction, either "horizontal" or "vertical".
- **border_value** (*int | tuple[int]*) –Value used in case of a constant border. Default 0.
- **interpolation** (*str*) –Same as in `mmcv.imshear()`.

返回 Sheared masks.

返回类型 ndarray

abstract to_ndarray()

Convert masks to the format of ndarray.

返回 Converted masks in the format of ndarray.

返回类型 ndarray

abstract to_tensor(*dtype*, *device*)

Convert masks to the format of Tensor.

参数

- **dtype** (*str*) –Dtype of converted mask.
- **device** (*torch.device*) –Device of converted masks.

返回 Converted masks in the format of Tensor.

返回类型 Tensor

abstract translate(*out_shape*, *offset*, *direction='horizontal'*, *fill_val=0*, *interpolation='bilinear'*)

Translate the masks.

参数

- **out_shape** (*tuple[int]*) –Shape for output mask, format (h, w).
- **offset** (*int | float*) –The offset for translate.
- **direction** (*str*) –The translate direction, either "horizontal" or "vertical".
- **fill_val** (*int | float*) –Border value. Default 0.
- **interpolation** (*str*) –Same as mmcv.imtranslate().

返回 Translated masks.

class mmdet.core.mask.**BitmapMasks**(*masks*, *height*, *width*)

This class represents masks in the form of bitmaps.

参数

- **masks** (*ndarray*) –ndarray of masks in shape (N, H, W), where N is the number of objects.
- **height** (*int*) –height of masks
- **width** (*int*) –width of masks

示例

```
>>> from mmdet.core.mask.structures import *  # NOQA
>>> num_masks, H, W = 3, 32, 32
>>> rng = np.random.RandomState(0)
>>> masks = (rng.rand(num_masks, H, W) > 0.1).astype(np.int)
>>> self = BitmapMasks(masks, height=H, width=W)
```

```
>>> # demo crop_and_resize
>>> num_boxes = 5
>>> bboxes = np.array([[0, 0, 30, 10.0]] * num_boxes)
>>> out_shape = (14, 14)
>>> inds = torch.randint(0, len(self), size=(num_boxes,))
>>> device = 'cpu'
>>> interpolation = 'bilinear'
>>> new = self.crop_and_resize(
...     bboxes, out_shape, inds, device, interpolation)
>>> assert len(new) == num_boxes
>>> assert new.height, new.width == out_shape
```

property areas

See *BaseInstanceMasks.areas*.

crop(*bbox*)

See *BaseInstanceMasks.crop()*.

crop_and_resize(*bboxes*, *out_shape*, *inds*, *device='cpu'*, *interpolation='bilinear'*, *binarize=True*)

See *BaseInstanceMasks.crop_and_resize()*.

expand(*expanded_h*, *expanded_w*, *top*, *left*)

See *BaseInstanceMasks.expand()*.

flip(*flip_direction='horizontal'*)

See *BaseInstanceMasks.flip()*.

pad(*out_shape*, *pad_val=0*)

See *BaseInstanceMasks.pad()*.

classmethod random(*num_masks=3*, *height=32*, *width=32*, *dtype=<class 'numpy.uint8'>*, *rng=None*)

Generate random bitmap masks for demo / testing purposes.

示例

```
>>> from mmdet.core.mask.structures import BitmapMasks
>>> self = BitmapMasks.random()
>>> print('self = {}'.format(self))
self = BitmapMasks(num_masks=3, height=32, width=32)
```

rescale(*scale*, *interpolation='nearest'*)

See *BaseInstanceMasks.rescale()*.

resize(*out_shape*, *interpolation='nearest'*)

See *BaseInstanceMasks.resize()*.

rotate(*out_shape*, *angle*, *center=None*, *scale=1.0*, *fill_val=0*)

Rotate the BitmapMasks.

参数

- **out_shape** (*tuple[int]*) –Shape for output mask, format (h, w).
- **angle** (*int | float*) –Rotation angle in degrees. Positive values mean counter-clockwise rotation.
- **center** (*tuple[float], optional*) –Center point (w, h) of the rotation in source image. If not specified, the center of the image will be used.
- **scale** (*int | float*) –Isotropic scale factor.
- **fill_val** (*int | float*) –Border value. Default 0 for masks.

返回 Rotated BitmapMasks.

返回类型 *BitmapMasks*

shear(*out_shape*, *magnitude*, *direction='horizontal'*, *border_value=0*, *interpolation='bilinear'*)

Shear the BitmapMasks.

参数

- **out_shape** (*tuple[int]*) –Shape for output mask, format (h, w).
- **magnitude** (*int | float*) –The magnitude used for shear.
- **direction** (*str*) –The shear direction, either "horizontal" or "vertical".
- **border_value** (*int | tuple[int]*) –Value used in case of a constant border.
- **interpolation** (*str*) –Same as in `mmcv.imshear()`.

返回 The sheared masks.

返回类型 *BitmapMasks*

to_ndarray()

See *BaseInstanceMasks.to_ndarray()*.

to_tensor(*dtype*, *device*)

See *BaseInstanceMasks.to_tensor()*.

translate(*out_shape*, *offset*, *direction='horizontal'*, *fill_val=0*, *interpolation='bilinear'*)

Translate the BitmapMasks.

参数

- **out_shape** (*tuple[int]*) –Shape for output mask, format (h, w).
- **offset** (*int | float*) –The offset for translate.
- **direction** (*str*) –The translate direction, either "horizontal" or "vertical".
- **fill_val** (*int | float*) –Border value. Default 0 for masks.
- **interpolation** (*str*) –Same as `mmcv.imtranslate()`.

返回 Translated BitmapMasks.

返回类型 *BitmapMasks*

示例

```
>>> from mmdet.core.mask.structures import BitmapMasks
>>> self = BitmapMasks.random(dtype=np.uint8)
>>> out_shape = (32, 32)
>>> offset = 4
>>> direction = 'horizontal'
>>> fill_val = 0
>>> interpolation = 'bilinear'
>>> # Note, There seem to be issues when:
>>> # * out_shape is different than self's shape
>>> # * the mask dtype is not supported by cv2.AffineWarp
>>> new = self.translate(out_shape, offset, direction, fill_val,
>>>                      interpolation)
>>> assert len(new) == len(self)
>>> assert new.height, new.width == out_shape
```

class mmdet.core.mask.**PolygonMasks**(*masks*, *height*, *width*)

This class represents masks in the form of polygons.

Polygons is a list of three levels. The first level of the list corresponds to objects, the second level to the polys that compose the object, the third level to the poly coordinates

参数

- **masks** (*list[list[ndarray]]*) –The first level of the list corresponds to objects, the second level to the polys that compose the object, the third level to the poly coordinates
- **height** (*int*) –height of masks
- **width** (*int*) –width of masks

示例

```
>>> from mmdet.core.mask.structures import *  # NOQA
>>> masks = [
>>>     [ np.array([0, 0, 10, 0, 10, 10., 0, 10, 0, 0]) ]
>>> ]
>>> height, width = 16, 16
>>> self = PolygonMasks(masks, height, width)
```

```
>>> # demo translate
>>> new = self.translate((16, 16), 4., direction='horizontal')
>>> assert np.all(new.masks[0][0][1::2] == masks[0][0][1::2])
>>> assert np.all(new.masks[0][0][0::2] == masks[0][0][0::2] + 4)
```

```
>>> # demo crop_and_resize
>>> num_boxes = 3
>>> bboxes = np.array([[0, 0, 30, 10.0]] * num_boxes)
>>> out_shape = (16, 16)
>>> inds = torch.randint(0, len(self), size=(num_boxes,))
>>> device = 'cpu'
>>> interpolation = 'bilinear'
>>> new = self.crop_and_resize(
...     bboxes, out_shape, inds, device, interpolation)
>>> assert len(new) == num_boxes
>>> assert new.height, new.width == out_shape
```

property areas

Compute areas of masks.

This func is modified from detectron2. The function only works with Polygons using the shoelace formula.

返回 areas of each instance

返回类型 ndarray

crop(*bbox*)

see *BaseInstanceMasks.crop()*

crop_and_resize(*bboxes*, *out_shape*, *inds*, *device='cpu'*, *interpolation='bilinear'*, *binarize=True*)

see *BaseInstanceMasks.crop_and_resize()*

expand(**args*, ***kwargs*)

TODO: Add expand for polygon

flip(*flip_direction='horizontal'*)

see *BaseInstanceMasks.flip()*

pad(*out_shape*, *pad_val=0*)

padding has no effect on polygons‘

classmethod random(*num_masks=3*, *height=32*, *width=32*, *n_verts=5*, *dtype=<class 'numpy.float32'>*, *rng=None*)

Generate random polygon masks for demo / testing purposes.

Adapted from[Page 101, 1]

引用

示例

```
>>> from mmdet.core.mask.structures import PolygonMasks
>>> self = PolygonMasks.random()
>>> print('self = {}'.format(self))
```

rescale(*scale*, *interpolation=None*)

see *BaseInstanceMasks.rescale()*

resize(*out_shape*, *interpolation=None*)

see *BaseInstanceMasks.resize()*

rotate(*out_shape*, *angle*, *center=None*, *scale=1.0*, *fill_val=0*)

See *BaseInstanceMasks.rotate()*.

shear(*out_shape*, *magnitude*, *direction='horizontal'*, *border_value=0*, *interpolation='bilinear'*)

See *BaseInstanceMasks.shear()*.

to_bitmap()

convert polygon masks to bitmap masks.

to_ndarray()

Convert masks to the format of ndarray.

to_tensor(*dtype*, *device*)

See *BaseInstanceMasks.to_tensor()*.

translate(*out_shape*, *offset*, *direction='horizontal'*, *fill_val=None*, *interpolation=None*)

Translate the PolygonMasks.

示例

```
>>> self = PolygonMasks.random(dtype=np.int)
>>> out_shape = (self.height, self.width)
>>> new = self.translate(out_shape, 4., direction='horizontal')
>>> assert np.all(new.masks[0][0][1::2] == self.masks[0][0][1::2])
>>> assert np.all(new.masks[0][0][0::2] == self.masks[0][0][0::2] + 4)  #␣
↪noqa: E501
```

mmdet.core.mask.**encode_mask_results**(*mask_results*)

Encode bitmap mask to RLE code.

参数 **mask_results** (*list | tuple[list]*) –bitmap mask results. In mask scoring rcnn, mask_results is a tuple of (segm_results, segm_cls_score).

返回 RLE encoded mask.

返回类型 list | tuple

mmdet.core.mask.**mask_target**(*pos_proposals_list*, *pos_assigned_gt_inds_list*, *gt_masks_list*, *cfg*)

Compute mask target for positive proposals in multiple images.

参数

- **pos_proposals_list** (*list[Tensor]*) –Positive proposals in multiple images.
- **pos_assigned_gt_inds_list** (*list[Tensor]*) –Assigned GT indices for each positive proposals.
- **gt_masks_list** (list[*BaseInstanceMasks*]) –Ground truth masks of each image.
- **cfg** (*dict*) –Config dict that specifies the mask size.

返回 Mask target of each image.

返回类型 list[Tensor]

示例

```
>>> import mmcv
>>> import mmdet
>>> from mmdet.core.mask import BitmapMasks
>>> from mmdet.core.mask.mask_target import *
>>> H, W = 17, 18
>>> cfg = mmcv.Config({'mask_size': (13, 14)})
>>> rng = np.random.RandomState(0)
>>> # Positive proposals (tl_x, tl_y, br_x, br_y) for each image
>>> pos_proposals_list = [
```

(下页继续)

(续上页)

```
>>>     torch.Tensor([
>>>         [ 7.2425,  5.5929, 13.9414, 14.9541],
>>>         [ 7.3241,  3.6170, 16.3850, 15.3102],
>>>     ]),
>>>     torch.Tensor([
>>>         [ 4.8448, 6.4010, 7.0314, 9.7681],
>>>         [ 5.9790, 2.6989, 7.4416, 4.8580],
>>>         [ 0.0000, 0.0000, 0.1398, 9.8232],
>>>     ]),
>>> ]
>>> # Corresponding class index for each proposal for each image
>>> pos_assigned_gt_inds_list = [
>>>     torch.LongTensor([7, 0]),
>>>     torch.LongTensor([5, 4, 1]),
>>> ]
>>> # Ground truth mask for each true object for each image
>>> gt_masks_list = [
>>>     BitmapMasks(rng.rand(8, H, W), height=H, width=W),
>>>     BitmapMasks(rng.rand(6, H, W), height=H, width=W),
>>> ]
>>> mask_targets = mask_target(
>>>     pos_proposals_list, pos_assigned_gt_inds_list,
>>>     gt_masks_list, cfg)
>>> assert mask_targets.shape == (5,) + cfg['mask_size']
```

mmdet.core.mask.**split_combined_polys**(*polys*, *poly_lens*, *polys_per_mask*)

Split the combined 1-D polys into masks.

A mask is represented as a list of polys, and a poly is represented as a 1-D array. In dataset, all masks are concatenated into a single 1-D tensor. Here we need to split the tensor into original representations.

参数

- **polys** (*list*) –a list (length = image num) of 1-D tensors
- **poly_lens** (*list*) –a list (length = image num) of poly length
- **polys_per_mask** (*list*) –a list (length = image num) of poly number of each mask

返回 a list (length = image num) of list (length = mask num) of list (length = poly num) of numpy array.

返回类型 list

22.2.5 evaluation

class mmdet.core.evaluation.**DistEvalHook**(*dataloader*, *start=None*, *interval=1*, *by_epoch=True*, *save_best=None*, *rule=None*, *test_fn=None*, *greater_keys=None*, *less_keys=None*, *broadcast_bn_buffer=True*, *tmpdir=None*, *gpu_collect=False*, ***eval_kwargs*)

class mmdet.core.evaluation.**EvalHook**(*dataloader*, *start=None*, *interval=1*, *by_epoch=True*, *save_best=None*, *rule=None*, *test_fn=None*, *greater_keys=None*, *less_keys=None*, ***eval_kwargs*)

mmdet.core.evaluation.**average_precision**(*recalls*, *precisions*, *mode='area'*)

Calculate average precision (for single or multiple scales).

参数

- **recalls** (*ndarray*) –shape (num_scales, num_dets) or (num_dets,)
- **precisions** (*ndarray*) –shape (num_scales, num_dets) or (num_dets,)
- **mode** (*str*) –'area' or '11points', 'area' means calculating the area under precision-recall curve, '11points' means calculating the average precision of recalls at [0, 0.1, ···, 1]

返回 calculated average precision

返回类型 float or ndarray

mmdet.core.evaluation.**eval_map**(*det_results*, *annotations*, *scale_ranges=None*, *iou_thr=0.5*, *dataset=None*, *logger=None*, *tpfp_fn=None*, *nproc=4*)

Evaluate mAP of a dataset.

参数

- **det_results** (*list[list]*) –[[cls1_det, cls2_det, ···], ···]. The outer list indicates images, and the inner list indicates per-class detected bboxes.
- **annotations** (*list[dict]*) –Ground truth annotations where each item of the list indicates an image. Keys of annotations are:
  - *bboxes*: numpy array of shape (n, 4)
  - *labels*: numpy array of shape (n,)
  - *bboxes_ignore* (optional): numpy array of shape (k, 4)
  - *labels_ignore* (optional): numpy array of shape (k,)
- **scale_ranges** (*list[tuple] | None*) –Range of scales to be evaluated, in the format [(min1, max1), (min2, max2), ···]. A range of (32, 64) means the area range between (32**2, 64**2). Default: None.
- **iou_thr** (*float*) –IoU threshold to be considered as matched. Default: 0.5.

- **dataset** (*list[str] | str | None*) –Dataset name or dataset classes, there are minor differences in metrics for different datsets, e.g. “voc07”, “imagenet_det”, etc. Default: None.
- **logger** (*logging.Logger | str | None*) –The way to print the mAP summary. See *mmcv.utils.print_log()* for details. Default: None.
- **tpfp_fn** (*callable | None*) –The function used to determine true/ false positives. If None, `tpfp_default()` is used as default unless dataset is ‘det’ or ‘vid’ (`tpfp_imagenet()` in this case). If it is given as a function, then this function is used to evaluate tp & fp. Default None.
- **nproc** (*int*) –Processes used for computing TP and FP. Default: 4.

返回 (mAP, [dict, dict, ···])

返回类型 tuple

mmdet.core.evaluation.**eval_recalls**(*gts*, *proposals*, *proposal_nums=None*, *iou_thrs=0.5*, *logger=None*)

Calculate recalls.

参数

- **gts** (*list[ndarray]*) –a list of arrays of shape (n, 4)
- **proposals** (*list[ndarray]*) –a list of arrays of shape (k, 4) or (k, 5)
- **proposal_nums** (*int | Sequence[int]*) –Top N proposals to be evaluated.
- **iou_thrs** (*float | Sequence[float]*) –IoU thresholds. Default: 0.5.
- **logger** (*logging.Logger | str | None*) –The way to print the recall summary. See *mmcv.utils.print_log()* for details. Default: None.

返回 recalls of different ious and proposal nums

返回类型 ndarray

mmdet.core.evaluation.**get_classes**(*dataset*)

Get class names of a dataset.

mmdet.core.evaluation.**plot_iou_recall**(*recalls*, *iou_thrs*)

Plot IoU-Recalls curve.

参数

- **recalls** (*ndarray or list*) –shape (k,)
- **iou_thrs** (*ndarray or list*) –same shape as *recalls*

mmdet.core.evaluation.**plot_num_recall**(*recalls*, *proposal_nums*)

Plot Proposal_num-Recalls curve.

参数

- **recalls** (*ndarray or list*) –shape (k,)
- **proposal_nums** (*ndarray or list*) –same shape as *recalls*

mmdet.core.evaluation.**print_map_summary**(*mean_ap*, *results*, *dataset=None*, *scale_ranges=None*, *logger=None*)

Print mAP and results of each class.

A table will be printed to show the gts/dets/recall/AP of each class and the mAP.

参数

- **mean_ap** (*float*) –Calculated from *eval_map()*.
- **results** (*list[dict]*) –Calculated from *eval_map()*.
- **dataset** (*list[str] | str | None*) –Dataset name or dataset classes.
- **scale_ranges** (*list[tuple] | None*) –Range of scales to be evaluated.
- **logger** (*logging.Logger | str | None*) –The way to print the mAP summary. See *mmcv.utils.print_log()* for details. Default: None.

mmdet.core.evaluation.**print_recall_summary**(*recalls*, *proposal_nums*, *iou_thrs*, *row_idxs=None*, *col_idxs=None*, *logger=None*)

Print recalls in a table.

参数

- **recalls** (*ndarray*) –calculated from *bbox_recalls*
- **proposal_nums** (*ndarray or list*) –top N proposals
- **iou_thrs** (*ndarray or list*) –iou thresholds
- **row_idxs** (*ndarray*) –which rows(proposal nums) to print
- **col_idxs** (*ndarray*) –which cols(iou thresholds) to print
- **logger** (*logging.Logger | str | None*) –The way to print the recall summary. See *mmcv.utils.print_log()* for details. Default: None.

22.2.6 post_processing

mmdet.core.post_processing.**fast_nms**(*multi_bboxes*, *multi_scores*, *multi_coeffs*, *score_thr*, *iou_thr*, *top_k*, *max_num=- 1*)

Fast NMS in YOLACT.

Fast NMS allows already-removed detections to suppress other detections so that every instance can be decided to be kept or discarded in parallel, which is not possible in traditional NMS. This relaxation allows us to implement Fast NMS entirely in standard GPU-accelerated matrix operations.

参数

- **multi_bboxes** (*Tensor*) –shape (n, #class*4) or (n, 4)
- **multi_scores** (*Tensor*) –shape (n, #class+1), where the last column contains scores of the background class, but this will be ignored.
- **multi_coeffs** (*Tensor*) –shape (n, #class*coeffs_dim).
- **score_thr** (*float*) –bbox threshold, bboxes with scores lower than it will not be considered.
- **iou_thr** (*float*) –IoU threshold to be considered as conflicted.
- **top_k** (*int*) –if there are more than top_k bboxes before NMS, only top top_k will be kept.
- **max_num** (*int*) –if there are more than max_num bboxes after NMS, only top max_num will be kept. If -1, keep all the bboxes. Default: -1.

返回

(dets, labels, coefficients), tensors of shape (k, 5), (k, 1), and (k, coeffs_dim). Dets are boxes with scores. Labels are 0-based.

返回类型 tuple

mmdet.core.post_processing.**merge_aug_bboxes**(*aug_bboxes*, *aug_scores*, *img_metas*, *rcnn_test_cfg*)

Merge augmented detection bboxes and scores.

参数

- **aug_bboxes** (*list[Tensor]*) –shape (n, 4*#class)
- **aug_scores** (*list[Tensor] or None*) –shape (n, #class)
- **img_shapes** (*list[Tensor]*) –shape (3,).
- **rcnn_test_cfg** (*dict*) –rcnn test config.

返回 (bboxes, scores)

返回类型 tuple

mmdet.core.post_processing.**merge_aug_masks**(*aug_masks*, *img_metas*, *rcnn_test_cfg*, *weights=None*)

Merge augmented mask prediction.

参数

- **aug_masks** (*list[ndarray]*) –shape (n, #class, h, w)
- **img_shapes** (*list[ndarray]*) –shape (3,).
- **rcnn_test_cfg** (*dict*) –rcnn test config.

返回 (bboxes, scores)

返回类型 tuple

mmdet.core.post_processing.**merge_aug_proposals**(*aug_proposals*, *img_metas*, *cfg*)

Merge augmented proposals (multiscale, flip, etc.)

参数

- **aug_proposals** (*list[Tensor]*) –proposals from different testing schemes, shape (n, 5). Note that they are not rescaled to the original image size.
- **img_metas** (*list[dict]*) –list of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmdet/datasets/pipelines/formatting.py:Collect*.
- **cfg** (*dict*) –rpn test config.

返回 shape (n, 4), proposals corresponding to original image scale.

返回类型 Tensor

mmdet.core.post_processing.**merge_aug_scores**(*aug_scores*)

Merge augmented bbox scores.

mmdet.core.post_processing.**multiclass_nms**(*multi_bboxes*, *multi_scores*, *score_thr*, *nms_cfg*, *max_num=- 1*, *score_factors=None*, *return_inds=False*)

NMS for multi-class bboxes.

参数

- **multi_bboxes** (*Tensor*) –shape (n, #class*4) or (n, 4)
- **multi_scores** (*Tensor*) –shape (n, #class), where the last column contains scores of the background class, but this will be ignored.
- **score_thr** (*float*) –bbox threshold, bboxes with scores lower than it will not be considered.
- **nms_thr** (*float*) –NMS IoU threshold
- **max_num** (*int, optional*) –if there are more than max_num bboxes after NMS, only top max_num will be kept. Default to -1.
- **score_factors** (*Tensor, optional*) –The factors multiplied to scores before applying NMS. Default to None.
- **return_inds** (*bool, optional*) –Whether return the indices of kept bboxes. Default to False.

返回

(dets, labels, indices (optional)), tensors of shape (k, 5), (k), and (k). Dets are boxes with scores. Labels are 0-based.

返回类型 tuple

22.2.7 utils

22.3 mmdet.datasets

22.3.1 datasets

22.3.2 pipelines

22.3.3 samplers

class mmdet.datasets.samplers.**DistributedGroupSampler**(*dataset*, *samples_per_gpu=1*, *num_replicas=None*, *rank=None*, *seed=0*)

Sampler that restricts data loading to a subset of the dataset.

It is especially useful in conjunction with `torch.nn.parallel.DistributedDataParallel`. In such case, each process can pass a DistributedSampler instance as a DataLoader sampler, and load a subset of the original dataset that is exclusive to it.

注解： Dataset is assumed to be of constant size.

参数

- **dataset** –Dataset used for sampling.
- **num_replicas** (*optional*) –Number of processes participating in distributed training.
- **rank** (*optional*) –Rank of the current process within num_replicas.
- **seed** (*int, optional*) –random seed used to shuffle the sampler if `shuffle=True`. This number should be identical across all processes in the distributed group. Default: 0.

class mmdet.datasets.samplers.**DistributedSampler**(*dataset*, *num_replicas=None*, *rank=None*, *shuffle=True*, *seed=0*)

class mmdet.datasets.samplers.**GroupSampler**(*dataset*, *samples_per_gpu=1*)

22.3.4 api_wrappers

22.4 mmdet.models

22.4.1 detectors

22.4.2 backbones

class mmdet.models.backbones.**CSPDarknet**(*arch='P5'*, *deepen_factor=1.0*, *widen_factor=1.0*, *out_indices=(2, 3, 4)*, *frozen_stages=- 1*, *use_depthwise=False*, *arch_ovewrite=None*, *spp_kernal_sizes=(5, 9, 13)*, *conv_cfg=None*, *norm_cfg={'eps': 0.001, 'momentum': 0.03, 'type': 'BN'}*, *act_cfg={'type': 'Swish'}*, *norm_eval=False*, *init_cfg={'a': 2.23606797749979, 'distribution': 'uniform', 'layer': 'Conv2d', 'mode': 'fan_in', 'nonlinearity': 'leaky_relu', 'type': 'Kaiming'}*)

CSP-Darknet backbone used in YOLOv5 and YOLOX.

参数

- **arch** (*str*) –Architechture of CSP-Darknet, from {P5, P6}. Default: P5.
- **deepen_factor** (*float*) –Depth multiplier, multiply number of channels in each layer by this amount. Default: 1.0.
- **widen_factor** (*float*) –Width multiplier, multiply number of blocks in CSP layer by this amount. Default: 1.0.
- **out_indices** (*Sequence[int]*) –Output from which stages. Default: (2, 3, 4).
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Default: -1.
- **use_depthwise** (*bool*) –Whether to use depthwise separable convolution. Default: False.
- **arch_ovewrite** (*list*) –Overwrite default arch settings. Default: None.
- **spp_kernal_sizes** –(tuple[int]): Sequential of kernel sizes of SPP layers. Default: (5, 9, 13).
- **conv_cfg** (*dict*) –Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict*) –Dictionary to construct and config norm layer. Default: dict(type='BN', requires_grad=True).
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type='LeakyReLU', negative_slope=0.1).

- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None.

示例

```
>>> from mmdet.models import CSPDarknet
>>> import torch
>>> self = CSPDarknet(depth=53)
>>> self.eval()
>>> inputs = torch.rand(1, 3, 416, 416)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
...
(1, 256, 52, 52)
(1, 512, 26, 26)
(1, 1024, 13, 13)
```

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

注解: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

train(*mode=True*)

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. `Dropout`, `BatchNorm`, etc.

参数 **mode** (*bool*) –whether to set training mode (`True`) or evaluation mode (`False`). Default: `True`.

返回 self

返回类型 Module

class mmdet.models.backbones.**Darknet**(*depth=53*, *out_indices=(3, 4, 5)*, *frozen_stages=-1*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *act_cfg={'negative_slope': 0.1, 'type': 'LeakyReLU'}*, *norm_eval=True*, *pretrained=None*, *init_cfg=None*)

Darknet backbone.

参数

- **depth** (*int*) –Depth of Darknet. Currently only support 53.
- **out_indices** (*Sequence[int]*) –Output from which stages.
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Default: -1.
- **conv_cfg** (*dict*) –Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict*) –Dictionary to construct and config norm layer. Default: dict(type='BN', requires_grad=True)
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type='LeakyReLU', negative_slope=0.1).
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only.
- **pretrained** (*str, optional*) –model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None

示例

```
>>> from mmdet.models import Darknet
>>> import torch
>>> self = Darknet(depth=53)
>>> self.eval()
>>> inputs = torch.rand(1, 3, 416, 416)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
...
(1, 256, 52, 52)
(1, 512, 26, 26)
(1, 1024, 13, 13)
```

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

注解: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

static make_conv_res_block(*in_channels*, *out_channels*, *res_repeat*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *act_cfg={'negative_slope': 0.1, 'type': 'LeakyReLU'}*)

In Darknet backbone, ConvLayer is usually followed by ResBlock. This function will make that. The Conv layers always have 3x3 filters with stride=2. The number of the filters in Conv layer is the same as the out channels of the ResBlock.

参数

- **in_channels** (*int*) –The number of input channels.
- **out_channels** (*int*) –The number of output channels.
- **res_repeat** (*int*) –The number of ResBlocks.
- **conv_cfg** (*dict*) –Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict*) –Dictionary to construct and config norm layer. Default: dict(type='BN', requires_grad=True)
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type='LeakyReLU', negative_slope=0.1).

train(*mode=True*)

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. `Dropout`, `BatchNorm`, etc.

参数 **mode** (*bool*) –whether to set training mode (`True`) or evaluation mode (`False`). Default: `True`.

返回 self

返回类型 Module

class mmdet.models.backbones.**DetectoRS_ResNeXt**(*groups=1*, *base_width=4*, ***kwargs*)

ResNeXt backbone for DetectoRS.

参数

- **groups** (*int*) –The number of groups in ResNeXt.
- **base_width** (*int*) –The base width of ResNeXt.

make_res_layer(***kwargs*)

Pack all blocks in a stage into a `ResLayer` for DetectoRS.

class mmdet.models.backbones.**DetectoRS_ResNet**(*sac=None*, *stage_with_sac=(False, False, False, False)*, *rfp_inplanes=None*, *output_img=False*, *pretrained=None*, *init_cfg=None*, ***kwargs*)

ResNet backbone for DetectoRS.

参数

- **sac** (*dict, optional*) –Dictionary to construct SAC (Switchable Atrous Convolution). Default: None.
- **stage_with_sac** (*list*) –Which stage to use sac. Default: (False, False, False, False).
- **rfp_inplanes** (*int, optional*) –The number of channels from RFP. Default: None. If specified, an additional conv layer will be added for `rfp_feat`. Otherwise, the structure is the same as base class.
- **output_img** (*bool*) –If `True`, the input image will be inserted into the starting position of output. Default: False.

forward(*x*)

Forward function.

init_weights()

Initialize the weights.

make_res_layer(***kwargs*)

Pack all blocks in a stage into a `ResLayer` for DetectoRS.

rfp_forward(*x*, *rfp_feats*)

Forward function for RFP.

class mmdet.models.backbones.**HRNet**(*extra*, *in_channels=3*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *norm_eval=True*, *with_cp=False*, *zero_init_residual=False*, *pretrained=None*, *init_cfg=None*)

HRNet backbone.

High-Resolution Representations for Labeling Pixels and Regions arXiv: https://arxiv.org/abs/1904.04514

参数

- **extra** (*dict*) –detailed configuration for each stage of HRNet.
- **in_channels** (*int*) –Number of input image channels. Default: 3.
- **conv_cfg** (*dict*) –dictionary to construct and config conv layer.
- **norm_cfg** (*dict*) –dictionary to construct and config norm layer.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only.

- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed.
- **zero_init_residual** (*bool*) –whether to use zero init for last norm layer in resblocks to let them behave as identity.
- **pretrained** (*str, optional*) –model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None

示例

```
>>> from mmdet.models import HRNet
>>> import torch
>>> extra = dict(
>>>     stage1=dict(
>>>         num_modules=1,
>>>         num_branches=1,
>>>         block='BOTTLENECK',
>>>         num_blocks=(4, ),
>>>         num_channels=(64, )),
>>>     stage2=dict(
>>>         num_modules=1,
>>>         num_branches=2,
>>>         block='BASIC',
>>>         num_blocks=(4, 4),
>>>         num_channels=(32, 64)),
>>>     stage3=dict(
>>>         num_modules=4,
>>>         num_branches=3,
>>>         block='BASIC',
>>>         num_blocks=(4, 4, 4),
>>>         num_channels=(32, 64, 128)),
>>>     stage4=dict(
>>>         num_modules=3,
>>>         num_branches=4,
>>>         block='BASIC',
>>>         num_blocks=(4, 4, 4, 4),
>>>         num_channels=(32, 64, 128, 256)))
>>> self = HRNet(extra, in_channels=1)
>>> self.eval()
>>> inputs = torch.rand(1, 1, 32, 32)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
```

(下页继续)

(续上页)

```
...     print(tuple(level_out.shape))
(1, 32, 8, 8)
(1, 64, 4, 4)
(1, 128, 2, 2)
(1, 256, 1, 1)
```

forward(*x*)

Forward function.

property norm1

the normalization layer named "norm1"

Type nn.Module

property norm2

the normalization layer named "norm2"

Type nn.Module

train(*mode=True*)

Convert the model into training mode will keeping the normalization layer freezed.

class mmdet.models.backbones.**HourglassNet**(*downsample_times=5*, *num_stacks=2*, *stage_channels=(256, 256, 384, 384, 384, 512)*, *stage_blocks=(2, 2, 2, 2, 2, 4)*, *feat_channel=256*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *pretrained=None*, *init_cfg=None*)

HourglassNet backbone.

Stacked Hourglass Networks for Human Pose Estimation. More details can be found in the paper .

参数

- **downsample_times** (*int*) –Downsample times in a HourglassModule.
- **num_stacks** (*int*) –Number of HourglassModule modules stacked, 1 for Hourglass-52, 2 for Hourglass-104.
- **stage_channels** (*list[int]*) –Feature channel of each sub-module in a HourglassModule.
- **stage_blocks** (*list[int]*) –Number of sub-modules stacked in a HourglassModule.
- **feat_channel** (*int*) –Feature channel of conv after a HourglassModule.
- **norm_cfg** (*dict*) –Dictionary to construct and config norm layer.
- **pretrained** (*str, optional*) –model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None

示例

```
>>> from mmdet.models import HourglassNet
>>> import torch
>>> self = HourglassNet()
>>> self.eval()
>>> inputs = torch.rand(1, 3, 511, 511)
>>> level_outputs = self.forward(inputs)
>>> for level_output in level_outputs:
...     print(tuple(level_output.shape))
(1, 256, 128, 128)
(1, 256, 128, 128)
```

forward(*x*)

Forward function.

init_weights()

Init module weights.

class mmdet.models.backbones.**MobileNetV2**(*widen_factor=1.0*, *out_indices=(1, 2, 4, 7)*, *frozen_stages=-1*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU6'}*, *norm_eval=False*, *with_cp=False*, *pretrained=None*, *init_cfg=None*)

MobileNetV2 backbone.

参数

- **widen_factor** (*float*) –Width multiplier, multiply number of channels in each layer by this amount. Default: 1.0.
- **out_indices** (*Sequence[int], optional*) –Output from which stages. Default: (1, 2, 4, 7).
- **frozen_stages** (*int*) –Stages to be frozen (all param fixed). Default: -1, which means not freezing any parameters.
- **conv_cfg** (*dict, optional*) –Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type='BN').
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type='ReLU6').
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **pretrained** (*str, optional*) –model pretrained path. Default: None

- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None

forward(*x*)

Forward function.

make_layer(*out_channels*, *num_blocks*, *stride*, *expand_ratio*)

Stack InvertedResidual blocks to build a layer for MobileNetV2.

参数

- **out_channels** (*int*) –out_channels of block.
- **num_blocks** (*int*) –number of blocks.
- **stride** (*int*) –stride of the first block. Default: 1
- **expand_ratio** (*int*) –Expand the number of channels of the hidden layer in InvertedResidual by this ratio. Default: 6.

train(*mode=True*)

Convert the model into training mode while keep normalization layer frozen.

class mmdet.models.backbones.**RegNet**(*arch*, *in_channels=3*, *stem_channels=32*, *base_channels=32*, *strides=(2, 2, 2, 2)*, *dilations=(1, 1, 1, 1)*, *out_indices=(0, 1, 2, 3)*, *style='pytorch'*, *deep_stem=False*, *avg_down=False*, *frozen_stages=-1*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *norm_eval=True*, *dcn=None*, *stage_with_dcn=(False, False, False, False)*, *plugins=None*, *with_cp=False*, *zero_init_residual=True*, *pretrained=None*, *init_cfg=None*)

RegNet backbone.

More details can be found in paper .

参数

- **arch** (*dict*) –The parameter of RegNets.
  - w0 (int): initial width
  - wa (float): slope of width
  - wm (float): quantization parameter to quantize the width
  - depth (int): depth of the backbone
  - group_w (int): width of group
  - bot_mul (float): bottleneck ratio, i.e. expansion of bottleneck.
- **strides** (*Sequence[int]*) –Strides of the first block of each stage.
- **base_channels** (*int*) –Base channels after stem layer.

- **in_channels** (*int*) –Number of input image channels. Default: 3.
- **dilations** (*Sequence[int]*) –Dilation of each stage.
- **out_indices** (*Sequence[int]*) –Output from which stages.
- **style** (*str*) –*pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer.
- **frozen_stages** (*int*) –Stages to be frozen (all param fixed). -1 means not freezing any parameters.
- **norm_cfg** (*dict*) –dictionary to construct and config norm layer.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed.
- **zero_init_residual** (*bool*) –whether to use zero init for last norm layer in resblocks to let them behave as identity.
- **pretrained** (*str, optional*) –model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None

示例

```
>>> from mmdet.models import RegNet
>>> import torch
>>> self = RegNet(
        arch=dict(
            w0=88,
            wa=26.31,
            wm=2.25,
            group_w=48,
            depth=25,
            bot_mul=1.0))
>>> self.eval()
>>> inputs = torch.rand(1, 3, 32, 32)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 96, 8, 8)
(1, 192, 4, 4)
```

(下页继续)

(续上页)

```
(1, 432, 2, 2)
(1, 1008, 1, 1)
```

adjust_width_group(*widths*, *bottleneck_ratio*, *groups*)

Adjusts the compatibility of widths and groups.

参数

- **widths** (*list[int]*) –Width of each stage.
- **bottleneck_ratio** (*float*) –Bottleneck ratio.
- **groups** (*int*) –number of groups in each stage

返回 The adjusted widths and groups of each stage.

返回类型 tuple(list)

forward(*x*)

Forward function.

generate_regnet(*initial_width*, *width_slope*, *width_parameter*, *depth*, *divisor=8*)

Generates per block width from RegNet parameters.

参数

- **initial_width** (*[int]*) –Initial width of the backbone
- **width_slope** (*[float]*) –Slope of the quantized linear function
- **width_parameter** (*[int]*) –Parameter used to quantize the width.
- **depth** (*[int]*) –Depth of the backbone.
- **divisor** (*int, optional*) –The divisor of channels. Defaults to 8.

返回 return a list of widths of each stage and the number of stages

返回类型 list, int

get_stages_from_blocks(*widths*)

Gets widths/stage_blocks of network at each stage.

参数 **widths** (*list[int]*) –Width in each stage.

返回 width and depth of each stage

返回类型 tuple(list)

static quantize_float(*number*, *divisor*)

Converts a float to closest non-zero int divisible by divisor.

参数

- **number** (*int*) –Original number to be quantized.

- **divisor** (*int*) –Divisor used to quantize the number.

返回 quantized number that is divisible by devisor.

返回类型 int

class mmdet.models.backbones.**Res2Net**(*scales=4*, *base_width=26*, *style='pytorch'*, *deep_stem=True*, *avg_down=True*, *pretrained=None*, *init_cfg=None*, ***kwargs*)

Res2Net backbone.

参数

- **scales** (*int*) –Scales used in Res2Net. Default: 4
- **base_width** (*int*) –Basic width of each scale. Default: 26
- **depth** (*int*) –Depth of res2net, from {50, 101, 152}.
- **in_channels** (*int*) –Number of input image channels. Default: 3.
- **num_stages** (*int*) –Res2net stages. Default: 4.
- **strides** (*Sequence[int]*) –Strides of the first block of each stage.
- **dilations** (*Sequence[int]*) –Dilation of each stage.
- **out_indices** (*Sequence[int]*) –Output from which stages.
- **style** (*str*) –*pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer.
- **deep_stem** (*bool*) –Replace 7x7 conv in input stem with 3 3x3 conv
- **avg_down** (*bool*) –Use AvgPool instead of stride conv when downsampling in the bottle2neck.
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters.
- **norm_cfg** (*dict*) –Dictionary to construct and config norm layer.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only.
- **plugins** (*list[dict]*) –List of plugins for stages, each dict contains:
  - cfg (dict, required): Cfg dict to build plugin.
  - position (str, required): Position inside block to insert plugin, options are 'after_conv1', 'after_conv2', 'after_conv3'.
  - stages (tuple[bool], optional): Stages to apply plugin, length should be same as 'num_stages'.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed.

- **zero_init_residual** (*bool*) –Whether to use zero init for last norm layer in resblocks to let them behave as identity.
- **pretrained** (*str, optional*) –model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None

示例

```
>>> from mmdet.models import Res2Net
>>> import torch
>>> self = Res2Net(depth=50, scales=4, base_width=26)
>>> self.eval()
>>> inputs = torch.rand(1, 3, 32, 32)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 256, 8, 8)
(1, 512, 4, 4)
(1, 1024, 2, 2)
(1, 2048, 1, 1)
```

make_res_layer(***kwargs*)

Pack all blocks in a stage into a `ResLayer`.

class mmdet.models.backbones.**ResNeSt**(*groups=1*, *base_width=4*, *radix=2*, *reduction_factor=4*, *avg_down_stride=True*, ***kwargs*)

ResNeSt backbone.

参数

- **groups** (*int*) –Number of groups of Bottleneck. Default: 1
- **base_width** (*int*) –Base width of Bottleneck. Default: 4
- **radix** (*int*) –Radix of SplitAttentionConv2d. Default: 2
- **reduction_factor** (*int*) –Reduction factor of inter_channels in SplitAttentionConv2d. Default: 4.
- **avg_down_stride** (*bool*) –Whether to use average pool for stride in Bottleneck. Default: True.
- **kwargs** (*dict*) –Keyword arguments for ResNet.

make_res_layer(***kwargs*)

Pack all blocks in a stage into a `ResLayer`.

class mmdet.models.backbones.**ResNeXt**(*groups=1*, *base_width=4*, ***kwargs*)

ResNeXt backbone.

参数

- **depth** (*int*) –Depth of resnet, from {18, 34, 50, 101, 152}.
- **in_channels** (*int*) –Number of input image channels. Default: 3.
- **num_stages** (*int*) –Resnet stages. Default: 4.
- **groups** (*int*) –Group of resnext.
- **base_width** (*int*) –Base width of resnext.
- **strides** (*Sequence[int]*) –Strides of the first block of each stage.
- **dilations** (*Sequence[int]*) –Dilation of each stage.
- **out_indices** (*Sequence[int]*) –Output from which stages.
- **style** (*str*) –*pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer.
- **frozen_stages** (*int*) –Stages to be frozen (all param fixed). -1 means not freezing any parameters.
- **norm_cfg** (*dict*) –dictionary to construct and config norm layer.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed.
- **zero_init_residual** (*bool*) –whether to use zero init for last norm layer in resblocks to let them behave as identity.

make_res_layer(***kwargs*)

Pack all blocks in a stage into a ResLayer

class mmdet.models.backbones.**ResNet**(*depth*, *in_channels=3*, *stem_channels=None*, *base_channels=64*, *num_stages=4*, *strides=(1, 2, 2, 2)*, *dilations=(1, 1, 1, 1)*, *out_indices=(0, 1, 2, 3)*, *style='pytorch'*, *deep_stem=False*, *avg_down=False*, *frozen_stages=- 1*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *norm_eval=True*, *dcn=None*, *stage_with_dcn=(False, False, False, False)*, *plugins=None*, *with_cp=False*, *zero_init_residual=True*, *pretrained=None*, *init_cfg=None*)

ResNet backbone.

参数

- **depth** (*int*) –Depth of resnet, from {18, 34, 50, 101, 152}.
- **stem_channels** (*int | None*) –Number of stem channels. If not specified, it will be the same as *base_channels*. Default: None.
- **base_channels** (*int*) –Number of base channels of res layer. Default: 64.
- **in_channels** (*int*) –Number of input image channels. Default: 3.
- **num_stages** (*int*) –Resnet stages. Default: 4.
- **strides** (*Sequence[int]*) –Strides of the first block of each stage.
- **dilations** (*Sequence[int]*) –Dilation of each stage.
- **out_indices** (*Sequence[int]*) –Output from which stages.
- **style** (*str*) –*pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer.
- **deep_stem** (*bool*) –Replace 7x7 conv in input stem with 3 3x3 conv
- **avg_down** (*bool*) –Use AvgPool instead of stride conv when downsampling in the bottleneck.
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters.
- **norm_cfg** (*dict*) –Dictionary to construct and config norm layer.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only.
- **plugins** (*list[dict]*) –List of plugins for stages, each dict contains:
  - cfg (dict, required): Cfg dict to build plugin.
  - position (str, required): Position inside block to insert plugin, options are 'after_conv1', 'after_conv2', 'after_conv3'.
  - stages (tuple[bool], optional): Stages to apply plugin, length should be same as 'num_stages'.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed.
- **zero_init_residual** (*bool*) –Whether to use zero init for last norm layer in resblocks to let them behave as identity.
- **pretrained** (*str, optional*) –model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None

示例

```
>>> from mmdet.models import ResNet
>>> import torch
>>> self = ResNet(depth=18)
>>> self.eval()
>>> inputs = torch.rand(1, 3, 32, 32)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 64, 8, 8)
(1, 128, 4, 4)
(1, 256, 2, 2)
(1, 512, 1, 1)
```

forward(*x*)

Forward function.

make_res_layer(***kwargs*)

Pack all blocks in a stage into a `ResLayer`.

make_stage_plugins(*plugins*, *stage_idx*)

Make plugins for ResNet `stage_idx` th stage.

Currently we support to insert `context_block`, `empirical_attention_block`, `nonlocal_block` into the backbone like ResNet/ResNeXt. They could be inserted after conv1/conv2/conv3 of Bottleneck.

An example of plugins format could be:

实际案例

```
>>> plugins=[
...     dict(cfg=dict(type='xxx', arg1='xxx'),
...          stages=(False, True, True, True),
...          position='after_conv2'),
...     dict(cfg=dict(type='yyy'),
...          stages=(True, True, True, True),
...          position='after_conv3'),
...     dict(cfg=dict(type='zzz', postfix='1'),
...          stages=(True, True, True, True),
...          position='after_conv3'),
...     dict(cfg=dict(type='zzz', postfix='2'),
...          stages=(True, True, True, True),
...          position='after_conv3')
```

(下页继续)

(续上页)

```
... ]
>>> self = ResNet(depth=18)
>>> stage_plugins = self.make_stage_plugins(plugins, 0)
>>> assert len(stage_plugins) == 3
```

Suppose `stage_idx=0`, the structure of blocks in the stage would be:

```
conv1-> conv2->conv3->yyy->zzz1->zzz2
```

Suppose ‘stage_idx=1’, the structure of blocks in the stage would be:

```
conv1-> conv2->xxx->conv3->yyy->zzz1->zzz2
```

If stages is missing, the plugin would be applied to all stages.

参数

- **plugins** (*list[dict]*) –List of plugins cfg to build. The postfix is required if multiple same type plugins are inserted.
- **stage_idx** (*int*) –Index of stage to build

返回 Plugins for current stage

返回类型 list[dict]

property norm1

the normalization layer named “norm1”

Type nn.Module

train(*mode=True*)

Convert the model into training mode while keep normalization layer freezed.

class mmdet.models.backbones.**ResNetV1d**(***kwargs*)

ResNetV1d variant described in Bag of Tricks.

Compared with default ResNet(ResNetV1b), ResNetV1d replaces the 7x7 conv in the input stem with three 3x3 convs. And in the downsampling block, a 2x2 avg_pool with stride 2 is added before conv, whose stride is changed to 1.

class mmdet.models.backbones.**SSDVGG**(*depth*, *with_last_pool=False*, *ceil_mode=True*, *out_indices=(3, 4)*, *out_feature_indices=(22, 34)*, *pretrained=None*, *init_cfg=None*, *input_size=None*, *l2_norm_scale=None*)

VGG Backbone network for single-shot-detection.

参数

- **depth** (*int*) –Depth of vgg, from {11, 13, 16, 19}.
- **with_last_pool** (*bool*) –Whether to add a pooling layer at the last of the model

- **ceil_mode** (*bool*) –When True, will use *ceil* instead of *floor* to compute the output shape.
- **out_indices** (*Sequence[int]*) –Output from which stages.
- **out_feature_indices** (*Sequence[int]*) –Output from which feature map.
- **pretrained** (*str, optional*) –model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None
- **input_size** (*int, optional*) –Deprecated argumment. Width and height of input, from {300, 512}.
- **l2_norm_scale** (*float, optional*) –Deprecated argumment. L2 normalization layer init scale.

示例

```
>>> self = SSDVGG(input_size=300, depth=11)
>>> self.eval()
>>> inputs = torch.rand(1, 3, 300, 300)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 1024, 19, 19)
(1, 512, 10, 10)
(1, 256, 5, 5)
(1, 256, 3, 3)
(1, 256, 1, 1)
```

forward(*x*)

Forward function.

init_weights(*pretrained=None*)

Initialize the weights.

class mmdet.models.backbones.**TridentResNet**(*depth*, *num_branch*, *test_branch_idx*, *trident_dilations*, ***kwargs*)

The stem layer, stage 1 and stage 2 in Trident ResNet are identical to ResNet, while in stage 3, Trident BottleBlock is utilized to replace the normal BottleBlock to yield trident output. Different branch shares the convolution weight but uses different dilations to achieve multi-scale output.

/ stage3(b0) x - stem - stage1 - stage2 - stage3(b1) - output stage3(b2) /

参数

- **depth** (*int*) –Depth of resnet, from {50, 101, 152}.

- **num_branch** (*int*) –Number of branches in TridentNet.
- **test_branch_idx** (*int*) –In inference, all 3 branches will be used if *test_branch_idx==-1*, otherwise only branch with index *test_branch_idx* will be used.
- **trident_dilations** (*tuple[int]*) –Dilations of different trident branch. len(trident_dilations) should be equal to num_branch.

22.4.3 necks

class mmdet.models.necks.**BFP**(*Balanced Feature Pyramids*)

BFP takes multi-level features as inputs and gather them into a single one, then refine the gathered feature and scatter the refined results to multi-level features. This module is used in Libra R-CNN (CVPR 2019), see the paper Libra R-CNN: Towards Balanced Learning for Object Detection for details.

参数

- **in_channels** (*int*) –Number of input channels (feature maps of all levels should have the same channels).
- **num_levels** (*int*) –Number of input feature levels.
- **conv_cfg** (*dict*) –The config dict for convolution layers.
- **norm_cfg** (*dict*) –The config dict for normalization layers.
- **refine_level** (*int*) –Index of integration and refine level of BSF in multi-level features from bottom to top.
- **refine_type** (*str*) –Type of the refine op, currently support [None, 'conv', 'non_local'].
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

forward(*inputs*)

Forward function.

class mmdet.models.necks.**CTResNetNeck**(*in_channel*, *num_deconv_filters*, *num_deconv_kernels*, *use_dcn=True*, *init_cfg=None*)

The neck used in CenterNet for object classification and box regression.

参数

- **in_channel** (*int*) –Number of input channels.
- **num_deconv_filters** (*tuple[int]*) –Number of filters per stage.
- **num_deconv_kernels** (*tuple[int]*) –Number of kernels per stage.
- **use_dcn** (*bool*) –If True, use DCNv2. Default: True.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

forward(*inputs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

注解: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

init_weights()

Initialize the weights.

class mmdet.models.necks.**ChannelMapper**(*in_channels*, *out_channels*, *kernel_size=3*, *conv_cfg=None*, *norm_cfg=None*, *act_cfg={'type': 'ReLU'}*, *num_outs=None*, *init_cfg={'distribution': 'uniform', 'layer': 'Conv2d', 'type': 'Xavier'}*)

Channel Mapper to reduce/increase channels of backbone features.

This is used to reduce/increase channels of backbone features.

参数

- **in_channels** (*List[int]*) –Number of input channels per scale.
- **out_channels** (*int*) –Number of output channels (used at each scale).
- **kernel_size** (*int, optional*) –kernel_size for reducing channels (used at each scale). Default: 3.
- **conv_cfg** (*dict, optional*) –Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict, optional*) –Config dict for normalization layer. Default: None.
- **act_cfg** (*dict, optional*) –Config dict for activation layer in ConvModule. Default: dict(type='ReLU').
- **num_outs** (*int, optional*) –Number of output feature maps. There would be extra_convs when num_outs larger than the length of in_channels.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

示例

```
>>> import torch
>>> in_channels = [2, 3, 5, 7]
>>> scales = [340, 170, 84, 43]
>>> inputs = [torch.rand(1, c, s, s)
...           for c, s in zip(in_channels, scales)]
>>> self = ChannelMapper(in_channels, 11, 3).eval()
>>> outputs = self.forward(inputs)
>>> for i in range(len(outputs)):
...     print(f'outputs[{i}].shape = {outputs[i].shape}')
outputs[0].shape = torch.Size([1, 11, 340, 340])
outputs[1].shape = torch.Size([1, 11, 170, 170])
outputs[2].shape = torch.Size([1, 11, 84, 84])
outputs[3].shape = torch.Size([1, 11, 43, 43])
```

forward(*inputs*)

Forward function.

class mmdet.models.necks.**DilatedEncoder**(*in_channels*, *out_channels*, *block_mid_channels*, *num_residual_blocks*)

Dilated Encoder for YOLOF <https://arxiv.org/abs/2103.09460>‘.

This module contains two types of components:

- **the original FPN lateral convolution layer and fpn convolution layer,** which are 1x1 conv + 3x3 conv
- the dilated residual block

参数

- **in_channels** (*int*) –The number of input channels.
- **out_channels** (*int*) –The number of output channels.
- **block_mid_channels** (*int*) –The number of middle block output channels
- **num_residual_blocks** (*int*) –The number of residual blocks.

forward(*feature*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

注解: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while

the latter silently ignores them.

---

class mmdet.models.necks.**FPG**(*in_channels*, *out_channels*, *num_outs*, *stack_times*, *paths*, *inter_channels=None*, *same_down_trans=None*, *same_up_trans={'kernel_size': 3, 'padding': 1, 'stride': 2, 'type': 'conv'}*, *across_lateral_trans={'kernel_size': 1, 'type': 'conv'}*, *across_down_trans={'kernel_size': 3, 'type': 'conv'}*, *across_up_trans=None*, *across_skip_trans={'type': 'identity'}*, *output_trans={'kernel_size': 3, 'type': 'last_conv'}*, *start_level=0*, *end_level=- 1*, *add_extra_convs=False*, *norm_cfg=None*, *skip_inds=None*, *init_cfg=[{'type': 'Caffe2Xavier', 'layer': 'Conv2d'}, {'type': 'Constant', 'layer': ['_BatchNorm', '_InstanceNorm', 'GroupNorm', 'LayerNorm'], 'val': 1.0}]*)

FPG.

Implementation of Feature Pyramid Grids (FPG). This implementation only gives the basic structure stated in the paper. But users can implement different type of transitions to fully explore the the potential power of the structure of FPG.

参数

- **in_channels** (*int*) –Number of input channels (feature maps of all levels should have the same channels).
- **out_channels** (*int*) –Number of output channels (used at each scale)
- **num_outs** (*int*) –Number of output scales.
- **stack_times** (*int*) –The number of times the pyramid architecture will be stacked.
- **paths** (*list[str]*) –Specify the path order of each stack level. Each element in the list should be either ‘bu’ (bottom-up) or ‘td’ (top-down).
- **inter_channels** (*int*) –Number of inter channels.
- **same_up_trans** (*dict*) –Transition that goes down at the same stage.
- **same_down_trans** (*dict*) –Transition that goes up at the same stage.
- **across_lateral_trans** (*dict*) –Across-pathway same-stage
- **across_down_trans** (*dict*) –Across-pathway bottom-up connection.
- **across_up_trans** (*dict*) –Across-pathway top-down connection.
- **across_skip_trans** (*dict*) –Across-pathway skip connection.
- **output_trans** (*dict*) –Transition that trans the output of the last stage.
- **start_level** (*int*) –Index of the start input backbone level used to build the feature pyramid. Default: 0.

- **end_level** (*int*) –Index of the end input backbone level (exclusive) to build the feature pyramid. Default: -1, which means the last level.
- **add_extra_convs** (*bool*) –It decides whether to add conv layers on top of the original feature maps. Default to False. If True, its actual mode is specified by *extra_convs_on_inputs*.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: None.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

forward(*inputs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

注解: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.necks.**FPN**(*in_channels*, *out_channels*, *num_outs*, *start_level=0*, *end_level=- 1*, *add_extra_convs=False*, *relu_before_extra_convs=False*, *no_norm_on_lateral=False*, *conv_cfg=None*, *norm_cfg=None*, *act_cfg=None*, *upsample_cfg={'mode': 'nearest'}*, *init_cfg={'distribution': 'uniform', 'layer': 'Conv2d', 'type': 'Xavier'}*)

Feature Pyramid Network.

This is an implementation of paper Feature Pyramid Networks for Object Detection.

参数

- **in_channels** (*List[int]*) –Number of input channels per scale.
- **out_channels** (*int*) –Number of output channels (used at each scale)
- **num_outs** (*int*) –Number of output scales.
- **start_level** (*int*) –Index of the start input backbone level used to build the feature pyramid. Default: 0.
- **end_level** (*int*) –Index of the end input backbone level (exclusive) to build the feature pyramid. Default: -1, which means the last level.
- **add_extra_convs** (*bool | str*) –If bool, it decides whether to add conv layers on top of the original feature maps. Default to False. If True, it is equivalent to *add_extra_convs='on_input'*. If str, it specifies the source feature map of the extra convs. Only the following options are allowed
  - 'on_input': Last feat map of neck inputs (i.e. backbone feature).
  - 'on_lateral': Last feature map after lateral convs.

  - ’ on_output’ : The last output feature map after fpn convs.

- **relu_before_extra_convs** (*bool*) –Whether to apply relu before the extra conv. Default: False.
- **no_norm_on_lateral** (*bool*) –Whether to apply norm on lateral. Default: False.
- **conv_cfg** (*dict*) –Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: None.
- **act_cfg** (*str*) –Config dict for activation layer in ConvModule. Default: None.
- **upsample_cfg** (*dict*) –Config dict for interpolate layer. Default: *dict(mode=’ nearest’)*
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

示例

```
>>> import torch
>>> in_channels = [2, 3, 5, 7]
>>> scales = [340, 170, 84, 43]
>>> inputs = [torch.rand(1, c, s, s)
...           for c, s in zip(in_channels, scales)]
>>> self = FPN(in_channels, 11, len(in_channels)).eval()
>>> outputs = self.forward(inputs)
>>> for i in range(len(outputs)):
...     print(f'outputs[{i}].shape = {outputs[i].shape}')
outputs[0].shape = torch.Size([1, 11, 340, 340])
outputs[1].shape = torch.Size([1, 11, 170, 170])
outputs[2].shape = torch.Size([1, 11, 84, 84])
outputs[3].shape = torch.Size([1, 11, 43, 43])
```

forward(*inputs*)

Forward function.

class mmdet.models.necks.**FPN_CARAFE**(*in_channels*, *out_channels*, *num_outs*, *start_level=0*, *end_level=-1*, *norm_cfg=None*, *act_cfg=None*, *order=('conv', 'norm', 'act')*, *upsample_cfg={'encoder_dilation': 1, 'encoder_kernel': 3, 'type': 'carafe', 'up_group': 1, 'up_kernel': 5}*, *init_cfg=None*)

FPN_CARAFE is a more flexible implementation of FPN. It allows more choice for upsample methods during the top-down pathway.

It can reproduce the performance of ICCV 2019 paper CARAFE: Content-Aware ReAssembly of FEatures Please refer to https://arxiv.org/abs/1905.02188 for more details.

参数

- **in_channels** (*list[int]*) –Number of channels for each input feature map.
- **out_channels** (*int*) –Output channels of feature pyramids.
- **num_outs** (*int*) –Number of output stages.
- **start_level** (*int*) –Start level of feature pyramids. (Default: 0)
- **end_level** (*int*) –End level of feature pyramids. (Default: -1 indicates the last level).
- **norm_cfg** (*dict*) –Dictionary to construct and config norm layer.
- **activate** (*str*) –Type of activation function in ConvModule (Default: None indicates w/o activation).
- **order** (*dict*) –Order of components in ConvModule.
- **upsample** (*str*) –Type of upsample layer.
- **upsample_cfg** (*dict*) –Dictionary to construct and config upsample layer.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None

forward(*inputs*)

Forward function.

init_weights()

Initialize the weights of module.

slice_as(*src*, *dst*)

Slice `src` as `dst`

注解： `src` should have the same or larger size than `dst`.

参数

- **src** (*torch.Tensor*) –Tensors to be sliced.
- **dst** (*torch.Tensor*) –`src` will be sliced to have the same size as `dst`.

返回 Sliced tensor.

返回类型 torch.Tensor

tensor_add(*a*, *b*)

Add tensors `a` and `b` that might have different sizes.

class mmdet.models.necks.**HRFPN**(*High Resolution Feature Pyramids*)

paper: High-Resolution Representations for Labeling Pixels and Regions.

参数

- **in_channels** (*list*) –number of channels for each branch.
- **out_channels** (*int*) –output channels of feature pyramids.
- **num_outs** (*int*) –number of output stages.
- **pooling_type** (*str*) –pooling for generating feature pyramids from {MAX, AVG}.
- **conv_cfg** (*dict*) –dictionary to construct and config conv layer.
- **norm_cfg** (*dict*) –dictionary to construct and config norm layer.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed.
- **stride** (*int*) –stride of 3x3 convolutional layers
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

forward(*inputs*)

Forward function.

class mmdet.models.necks.**NASFCOS_FPN**(*in_channels*, *out_channels*, *num_outs*, *start_level=1*, *end_level=- 1*, *add_extra_convs=False*, *conv_cfg=None*, *norm_cfg=None*, *init_cfg=None*)

FPN structure in NASFPN.

Implementation of paper NAS-FCOS: Fast Neural Architecture Search for Object Detection

参数

- **in_channels** (*List[int]*) –Number of input channels per scale.
- **out_channels** (*int*) –Number of output channels (used at each scale)
- **num_outs** (*int*) –Number of output scales.
- **start_level** (*int*) –Index of the start input backbone level used to build the feature pyramid. Default: 0.
- **end_level** (*int*) –Index of the end input backbone level (exclusive) to build the feature pyramid. Default: -1, which means the last level.
- **add_extra_convs** (*bool*) –It decides whether to add conv layers on top of the original feature maps. Default to False. If True, its actual mode is specified by *extra_convs_on_inputs*.
- **conv_cfg** (*dict*) –dictionary to construct and config conv layer.
- **norm_cfg** (*dict*) –dictionary to construct and config norm layer.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None

forward(*inputs*)

Forward function.

init_weights()

Initialize the weights of module.

class mmdet.models.necks.**NASFPN**(*in_channels*, *out_channels*, *num_outs*, *stack_times*, *start_level=0*, *end_level=- 1*, *add_extra_convs=False*, *norm_cfg=None*, *init_cfg={'layer': 'Conv2d', 'type': 'Caffe2Xavier'}*)

NAS-FPN.

Implementation of NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection

参数

- **in_channels** (*List[int]*) –Number of input channels per scale.
- **out_channels** (*int*) –Number of output channels (used at each scale)
- **num_outs** (*int*) –Number of output scales.
- **stack_times** (*int*) –The number of times the pyramid architecture will be stacked.
- **start_level** (*int*) –Index of the start input backbone level used to build the feature pyramid. Default: 0.
- **end_level** (*int*) –Index of the end input backbone level (exclusive) to build the feature pyramid. Default: -1, which means the last level.
- **add_extra_convs** (*bool*) –It decides whether to add conv layers on top of the original feature maps. Default to False. If True, its actual mode is specified by *extra_convs_on_inputs*.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

forward(*inputs*)

Forward function.

class mmdet.models.necks.**PAFPN**(*in_channels*, *out_channels*, *num_outs*, *start_level=0*, *end_level=- 1*, *add_extra_convs=False*, *relu_before_extra_convs=False*, *no_norm_on_lateral=False*, *conv_cfg=None*, *norm_cfg=None*, *act_cfg=None*, *init_cfg={'distribution': 'uniform', 'layer': 'Conv2d', 'type': 'Xavier'}*)

Path Aggregation Network for Instance Segmentation.

This is an implementation of the PAFPN in Path Aggregation Network.

参数

- **in_channels** (*List[int]*) –Number of input channels per scale.
- **out_channels** (*int*) –Number of output channels (used at each scale)
- **num_outs** (*int*) –Number of output scales.
- **start_level** (*int*) –Index of the start input backbone level used to build the feature pyramid. Default: 0.

- **end_level** (*int*) –Index of the end input backbone level (exclusive) to build the feature pyramid. Default: -1, which means the last level.
- **add_extra_convs** (*bool | str*) –If bool, it decides whether to add conv layers on top of the original feature maps. Default to False. If True, it is equivalent to *add_extra_convs='on_input'*. If str, it specifies the source feature map of the extra convs. Only the following options are allowed
  - 'on_input': Last feat map of neck inputs (i.e. backbone feature).
  - 'on_lateral': Last feature map after lateral convs.
  - 'on_output': The last output feature map after fpn convs.
- **relu_before_extra_convs** (*bool*) –Whether to apply relu before the extra conv. Default: False.
- **no_norm_on_lateral** (*bool*) –Whether to apply norm on lateral. Default: False.
- **conv_cfg** (*dict*) –Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: None.
- **act_cfg** (*str*) –Config dict for activation layer in ConvModule. Default: None.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

forward(*inputs*)

Forward function.

class mmdet.models.necks.**RFP**(*Recursive Feature Pyramid*)

This is an implementation of RFP in DetectoRS. Different from standard FPN, the input of RFP should be multi level features along with origin input image of backbone.

参数

- **rfp_steps** (*int*) –Number of unrolled steps of RFP.
- **rfp_backbone** (*dict*) –Configuration of the backbone for RFP.
- **aspp_out_channels** (*int*) –Number of output channels of ASPP module.
- **aspp_dilations** (*tuple[int]*) –Dilation rates of four branches. Default: (1, 3, 6, 1)
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None

forward(*inputs*)

Forward function.

init_weights()

Initialize the weights.

class mmdet.models.necks.**SSDNeck**(*in_channels*, *out_channels*, *level_strides*, *level_paddings*, *l2_norm_scale=20.0*, *last_kernel_size=3*, *use_depthwise=False*, *conv_cfg=None*, *norm_cfg=None*, *act_cfg={'type': 'ReLU'}*, *init_cfg=[{'type': 'Xavier', 'distribution': 'uniform', 'layer': 'Conv2d'}, {'type': 'Constant', 'val': 1, 'layer': 'BatchNorm2d'}]*)

Extra layers of SSD backbone to generate multi-scale feature maps.

参数

- **in_channels** (*Sequence[int]*) –Number of input channels per scale.
- **out_channels** (*Sequence[int]*) –Number of output channels per scale.
- **level_strides** (*Sequence[int]*) –Stride of 3x3 conv per level.
- **level_paddings** (*Sequence[int]*) –Padding size of 3x3 conv per level.
- **l2_norm_scale** (*float|None*) –L2 normalization layer init scale. If None, not use L2 normalization on the first input feature.
- **last_kernel_size** (*int*) –Kernel size of the last conv layer. Default: 3.
- **use_depthwise** (*bool*) –Whether to use DepthwiseSeparableConv. Default: False.
- **conv_cfg** (*dict*) –Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict*) –Dictionary to construct and config norm layer. Default: None.
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type=' ReLU').
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

forward(*inputs*)

Forward function.

class mmdet.models.necks.**YOLOV3Neck**(*num_scales*, *in_channels*, *out_channels*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *act_cfg={'negative_slope': 0.1, 'type': 'LeakyReLU'}*, *init_cfg=None*)

The neck of YOLOV3.

It can be treated as a simplified version of FPN. It will take the result from Darknet backbone and do some upsampling and concatenation. It will finally output the detection result.

注解:

The input feats should be from top to bottom. i.e., from high-lvl to low-lvl

But YOLOV3Neck will process them in reversed order. i.e., from bottom (high-lvl) to top (low-lvl)

参数

- **num_scales** (*int*) –The number of scales / stages.
- **in_channels** (*List[int]*) –The number of input channels per scale.
- **out_channels** (*List[int]*) –The number of output channels per scale.
- **conv_cfg** (*dict, optional*) –Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict, optional*) –Dictionary to construct and config norm layer. Default: dict(type='BN', requires_grad=True)
- **act_cfg** (*dict, optional*) –Config dict for activation layer. Default: dict(type='LeakyReLU', negative_slope=0.1).
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None

forward(*feats*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

注解: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.necks.**YOLOXPAFPN**(*in_channels*, *out_channels*, *num_csp_blocks=3*, *use_depthwise=False*, *upsample_cfg={'mode': 'nearest', 'scale_factor': 2}*, *conv_cfg=None*, *norm_cfg={'eps': 0.001, 'momentum': 0.03, 'type': 'BN'}*, *act_cfg={'type': 'Swish'}*, *init_cfg={'a': 2.23606797749979, 'distribution': 'uniform', 'layer': 'Conv2d', 'mode': 'fan_in', 'nonlinearity': 'leaky_relu', 'type': 'Kaiming'}*)

Path Aggregation Network used in YOLOX.

参数

- **in_channels** (*List[int]*) –Number of input channels per scale.
- **out_channels** (*int*) –Number of output channels (used at each scale)
- **num_csp_blocks** (*int*) –Number of bottlenecks in CSPLayer. Default: 3
- **use_depthwise** (*bool*) –Whether to depthwise separable convolution in blocks. Default: False
- **upsample_cfg** (*dict*) –Config dict for interpolate layer. Default: *dict(scale_factor=2, mode='nearest')*

- **conv_cfg** (*dict, optional*) –Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type='BN')
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type='Swish')
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None.

forward(*inputs*)

参数 **inputs** (*tuple[Tensor]*) –input features.

返回 YOLOXPAFPN features.

返回类型 tuple[Tensor]

22.4.4 dense_heads

22.4.5 roi_heads

22.4.6 losses

22.4.7 utils

class mmdet.models.utils.**CSPLayer**(*in_channels*, *out_channels*, *expand_ratio=0.5*, *num_blocks=1*, *add_identity=True*, *use_depthwise=False*, *conv_cfg=None*, *norm_cfg={'eps': 0.001, 'momentum': 0.03, 'type': 'BN'}*, *act_cfg={'type': 'Swish'}*, *init_cfg=None*)

Cross Stage Partial Layer.

参数

- **in_channels** (*int*) –The input channels of the CSP layer.
- **out_channels** (*int*) –The output channels of the CSP layer.
- **expand_ratio** (*float*) –Ratio to adjust the number of channels of the hidden layer. Default: 0.5
- **num_blocks** (*int*) –Number of blocks. Default: 1
- **add_identity** (*bool*) –Whether to add identity in blocks. Default: True
- **use_depthwise** (*bool*) –Whether to depthwise separable convolution in blocks. Default: False
- **conv_cfg** (*dict, optional*) –Config dict for convolution layer. Default: None, which means using conv2d.

- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type='BN')
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type='Swish')

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

注解: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.utils.**ConvUpsample**(*in_channels*, *inner_channels*, *num_layers=1*, *num_upsample=None*, *conv_cfg=None*, *norm_cfg=None*, *init_cfg=None*, ***kwargs*)

ConvUpsample performs 2x upsampling after Conv.

There are several *ConvModule* layers. In the first few layers, upsampling will be applied after each layer of convolution. The number of upsampling must be no more than the number of ConvModule layers.

参数

- **in_channels** (*int*) –Number of channels in the input feature map.
- **inner_channels** (*int*) –Number of channels produced by the convolution.
- **num_layers** (*int*) –Number of convolution layers.
- **num_upsample** (*int | optional*) –Number of upsampling layer. Must be no more than num_layers. Upsampling will be applied after the first `num_upsample` layers of convolution. Default: `num_layers`.
- **conv_cfg** (*dict*) –Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: None.
- **init_cfg** (*dict*) –Config dict for initialization. Default: None.
- **kwargs** (*key word augments*) –Other augments used in ConvModule.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

注解: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while

the latter silently ignores them.

---

class mmdet.models.utils.**DetrTransformerDecoder**(**args*, *post_norm_cfg={'type': 'LN'}*, *return_intermediate=False*, ***kwargs*)

Implements the decoder in DETR transformer.

参数

- **return_intermediate** (*bool*) –Whether to return intermediate outputs.
- **post_norm_cfg** (*dict*) –Config of last normalization layer. Default：*LN*.

forward(*query*, **args*, ***kwargs*)

Forward function for *TransformerDecoder*.

参数 **query** (*Tensor*) –Input query with shape *(num_query, bs, embed_dims)*.

返回

Results with shape [1, num_query, bs, embed_dims] when return_intermediate is *False*, otherwise it has shape [num_layers, num_query, bs, embed_dims].

返回类型 Tensor

class mmdet.models.utils.**DetrTransformerDecoderLayer**(*attn_cfgs*, *feedforward_channels*, *ffn_dropout=0.0*, *operation_order=None*, *act_cfg={'inplace': True, 'type': 'ReLU'}*, *norm_cfg={'type': 'LN'}*, *ffn_num_fcs=2*, ***kwargs*)

Implements decoder layer in DETR transformer.

参数

- **attn_cfgs** (list[*mmcv.ConfigDict*] | list[dict] | dict) –Configs for self_attention or cross_attention, the order should be consistent with it in *operation_order*. If it is a dict, it would be expand to the number of attention in *operation_order*.
- **feedforward_channels** (*int*) –The hidden dimension for FFNs.
- **ffn_dropout** (*float*) –Probability of an element to be zeroed in ffn. Default 0.0.
- **operation_order** (*tuple[str]*) –The execution order of operation in transformer. Such as (‘self_attn’, ‘norm’, ‘ffn’, ‘norm’). Default：None
- **act_cfg** (*dict*) –The activation config for FFNs. Default: *LN*
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: *LN*.
- **ffn_num_fcs** (*int*) –The number of fully-connected layers in FFNs. Default：2.

class mmdet.models.utils.**DynamicConv**(*in_channels=256*, *feat_channels=64*, *out_channels=None*, *input_feat_shape=7*, *act_cfg={'inplace': True, 'type': 'ReLU'}*, *norm_cfg={'type': 'LN'}*, *init_cfg=None*)

Implements Dynamic Convolution.

This module generate parameters for each sample and use bmm to implement 1*1 convolution. Code is modified from the official github repo .

参数

- **in_channels** (*int*) –The input feature channel. Defaults to 256.
- **feat_channels** (*int*) –The inner feature channel. Defaults to 64.
- **out_channels** (*int, optional*) –The output feature channel. When not specified, it will be set to *in_channels* by default
- **input_feat_shape** (*int*) –The shape of input feature. Defaults to 7.
- **act_cfg** (*dict*) –The activation config for DynamicConv.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default layer normalization.
- **(obj** (*init_cfg*) –*mmcv.ConfigDict*): The Config for initialization. Default: None.

forward(*param_feature*, *input_feature*)

Forward function for *DynamicConv*.

参数

- **param_feature** (*Tensor*) –The feature can be used to generate the parameter, has shape (num_all_proposals, in_channels).
- **input_feature** (*Tensor*) –Feature that interact with parameters, has shape (num_all_proposals, in_channels, H, W).

返回 The output feature has shape (num_all_proposals, out_channels).

返回类型 Tensor

class mmdet.models.utils.**InvertedResidual**(*in_channels*, *out_channels*, *mid_channels*, *kernel_size=3*, *stride=1*, *se_cfg=None*, *with_expand_conv=True*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *with_cp=False*, *init_cfg=None*)

Inverted Residual Block.

参数

- **in_channels** (*int*) –The input channels of this Module.
- **out_channels** (*int*) –The output channels of this Module.
- **mid_channels** (*int*) –The input channels of the depthwise convolution.

- **kernel_size** (*int*) –The kernal size of the depthwise convolution. Default: 3.
- **stride** (*int*) –The stride of the depthwise convolution. Default: 1.
- **se_cfg** (*dict*) –Config dict for se layer. Defaul: None, which means no se layer.
- **with_expand_conv** (*bool*) –Use expand conv or not. If set False, mid_channels must be the same with in_channels. Default: True.
- **conv_cfg** (*dict*) –Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type='BN').
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type='ReLU').
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None

返回 The output tensor.

返回类型 Tensor

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

注解: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.utils.**LearnedPositionalEncoding**(*num_feats*, *row_num_embed=50*, *col_num_embed=50*, *init_cfg={'layer': 'Embedding', 'type': 'Uniform'}*)

Position embedding with learnable embedding weights.

参数

- **num_feats** (*int*) –The feature dimension for each position along x-axis or y-axis. The final returned dimension for each position is 2 times of this value.
- **row_num_embed** (*int, optional*) –The dictionary size of row embeddings. Default 50.
- **col_num_embed** (*int, optional*) –The dictionary size of col embeddings. Default 50.

- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

forward(*mask*)

Forward function for *LearnedPositionalEncoding*.

参数 **mask** (*Tensor*) –ByteTensor mask. Non-zero values representing ignored positions, while zero values means valid positions for this image. Shape [bs, h, w].

返回

Returned position embedding with shape [bs, num_feats*2, h, w].

返回类型 pos (Tensor)

class mmdet.models.utils.**NormedConv2d**(**args*, *tempearture=20*, *power=1.0*, *eps=1e-06*, *norm_over_kernel=False*, ***kwargs*)

Normalized Conv2d Layer.

参数

- **tempeature** (*float, optional*) –Tempeature term. Default to 20.
- **power** (*int, optional*) –Power term. Default to 1.0.
- **eps** (*float, optional*) –The minimal value of divisor to keep numerical stability. Default to 1e-6.
- **norm_over_kernel** (*bool, optional*) –Normalize over kernel. Default to False.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

注解: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.utils.**NormedLinear**(**args*, *tempearture=20*, *power=1.0*, *eps=1e-06*, ***kwargs*)

Normalized Linear Layer.

参数

- **tempeature** (*float, optional*) –Tempeature term. Default to 20.
- **power** (*int, optional*) –Power term. Default to 1.0.
- **eps** (*float, optional*) –The minimal value of divisor to keep numerical stability. Default to 1e-6.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

注解: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.utils.**ResLayer**(*block*, *inplanes*, *planes*, *num_blocks*, *stride=1*, *avg_down=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *downsample_first=True*, ***kwargs*)

ResLayer to build ResNet style backbone.

参数

- **block** (*nn.Module*) –block used to build ResLayer.
- **inplanes** (*int*) –inplanes of block.
- **planes** (*int*) –planes of block.
- **num_blocks** (*int*) –number of blocks.
- **stride** (*int*) –stride of the first block. Default: 1
- **avg_down** (*bool*) –Use AvgPool instead of stride conv when downsampling in the bottleneck. Default: False
- **conv_cfg** (*dict*) –dictionary to construct and config conv layer. Default: None
- **norm_cfg** (*dict*) –dictionary to construct and config norm layer. Default: dict(type='BN')
- **downsample_first** (*bool*) –Downsample at the first block or last block. False for Hourglass, True for ResNet. Default: True

class mmdet.models.utils.**SELayer**(*channels*, *ratio=16*, *conv_cfg=None*, *act_cfg=({'type': 'ReLU'}, {'type': 'Sigmoid'})*, *init_cfg=None*)

Squeeze-and-Excitation Module.

参数

- **channels** (*int*) –The input (and output) channels of the SE layer.
- **ratio** (*int*) –Squeeze ratio in SELayer, the intermediate channel will be `int(channels/ratio)`. Default: 16.
- **conv_cfg** (*None or dict*) –Config dict for convolution layer. Default: None, which means using conv2d.
- **act_cfg** (*dict or Sequence[dict]*) –Config dict for activation layer. If act_cfg is a dict, two activation layers will be configurated by this dict. If act_cfg is a sequence of dicts,

the first activation layer will be configurated by the first dict and the second activation layer will be configurated by the second dict. Default: (dict(type='ReLU'), dict(type='Sigmoid'))

- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

注解: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

class mmdet.models.utils.**SimplifiedBasicBlock**(*inplanes*, *planes*, *stride=1*, *dilation=1*, *downsample=None*, *style='pytorch'*, *with_cp=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *dcn=None*, *plugins=None*, *init_fg=None*)

Simplified version of original basic residual block. This is used in SCNet.

- Norm layer is now optional
- Last ReLU in forward function is removed

forward(*x*)

Forward function.

property norm1

normalization layer after the first convolution layer

Type nn.Module

property norm2

normalization layer after the second convolution layer

Type nn.Module

class mmdet.models.utils.**SinePositionalEncoding**(*num_feats*, *temperature=10000*, *normalize=False*, *scale=6.283185307179586*, *eps=1e-06*, *offset=0.0*, *init_cfg=None*)

Position encoding with sine and cosine functions.

See End-to-End Object Detection with Transformers for details.

参数

- **num_feats** (*int*) –The feature dimension for each position along x-axis or y-axis. Note the final returned dimension for each position is 2 times of this value.
- **temperature** (*int, optional*) –The temperature used for scaling the position embedding. Defaults to 10000.
- **normalize** (*bool, optional*) –Whether to normalize the position embedding. Defaults to False.
- **scale** (*float, optional*) –A scale factor that scales the position embedding. The scale will be used only when *normalize* is True. Defaults to 2*pi.
- **eps** (*float, optional*) –A value added to the denominator for numerical stability. Defaults to 1e-6.
- **offset** (*float*) –offset add to embed when do the normalization. Defaults to 0.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None

forward(*mask*)

Forward function for *SinePositionalEncoding*.

参数 **mask** (*Tensor*) –ByteTensor mask. Non-zero values representing ignored positions, while zero values means valid positions for this image. Shape [bs, h, w].

返回

Returned position embedding with shape [bs, num_feats*2, h, w].

返回类型 pos (Tensor)

class mmdet.models.utils.**Transformer**(*encoder=None*, *decoder=None*, *init_cfg=None*)

Implements the DETR transformer.

Following the official DETR implementation, this module copy-paste from torch.nn.Transformer with modifications:

- positional encodings are passed in MultiheadAttention
- extra LN at the end of encoder is removed
- decoder returns a stack of activations from all decoding layers

See paper: End-to-End Object Detection with Transformers for details.

参数

- **encoder** (*mmcv.ConfigDict* | Dict) –Config of TransformerEncoder. Defaults to None.
- **decoder** ((*mmcv.ConfigDict* | Dict)) –Config of TransformerDecoder. Defaults to None
- **(obj** (*init_cfg*) –*mmcv.ConfigDict*): The Config for initialization. Defaults to None.

forward(*x*, *mask*, *query_embed*, *pos_embed*)

Forward function for *Transformer*.

参数

- **x** (*Tensor*) –Input query with shape [bs, c, h, w] where c = embed_dims.
- **mask** (*Tensor*) –The key_padding_mask used for encoder and decoder, with shape [bs, h, w].
- **query_embed** (*Tensor*) –The query embedding for decoder, with shape [num_query, c].
- **pos_embed** (*Tensor*) –The positional encoding for encoder and decoder, with the same shape as *x*.

返回

results of decoder containing the following tensor.

- **out_dec: Output from decoder. If return_intermediate_dec is True output has shape [num_dec_layers, bs,** num_query, embed_dims], else has shape [1, bs, num_query, embed_dims].
- memory: Output results from encoder, with shape [bs, embed_dims, h, w].

返回类型 tuple[Tensor]

init_weights()

Initialize the weights.

mmdet.models.utils.**build_linear_layer**(*cfg*, **args*, ***kwargs*)

Build linear layer. :param cfg: The linear layer config, which should contain:

- type (str): Layer type.
- layer args: Args needed to instantiate an linear layer.

参数

- **args** (*argument list*) –Arguments passed to the *__init__* method of the corresponding linear layer.
- **kwargs** (*keyword arguments*) –Keyword arguments passed to the *__init__* method of the corresponding linear layer.

返回 Created linear layer.

返回类型 nn.Module

mmdet.models.utils.**build_transformer**(*cfg*, *default_args=None*)

Builder for Transformer.

mmdet.models.utils.**gaussian_radius**(*det_size*, *min_overlap*)

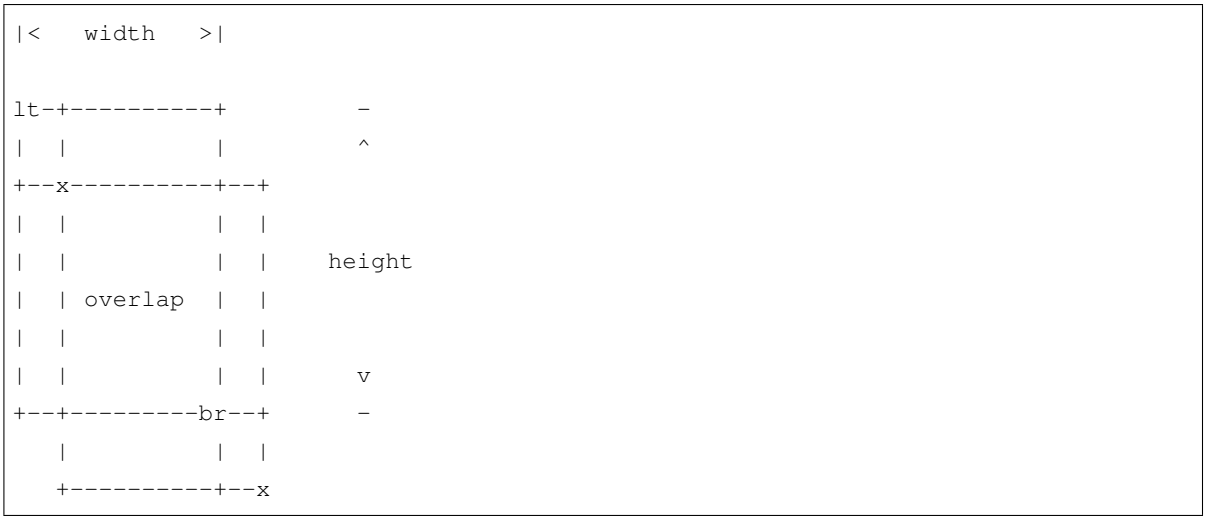
Generate 2D gaussian radius.

This function is modified from the official github repo.

Given `min_overlap`, radius could computed by a quadratic equation according to Vieta' s formulas.

There are 3 cases for computing gaussian radius, details are following:

- Explanation of figure: `lt` and `br` indicates the left-top and bottom-right corner of ground truth box. `x` indicates the generated corner at the limited position when `radius=r`.
- Case1: one corner is inside the gt box and the other is outside.



To ensure IoU of generated box and gt box is larger than `min_overlap`:

$$\frac{(w-r)*(h-r)}{w*h+(w+h)r-r^2} \geq iou \quad \Rightarrow \quad r^2-(w+h)r+\frac{1-iou}{1+iou}*w*h \geq 0$$

$$a=1, \quad b=-(w+h), \quad c=\frac{1-iou}{1+iou}*w*hr \leq \frac{-b-\sqrt{b^2-4*a*c}}{2*a}$$

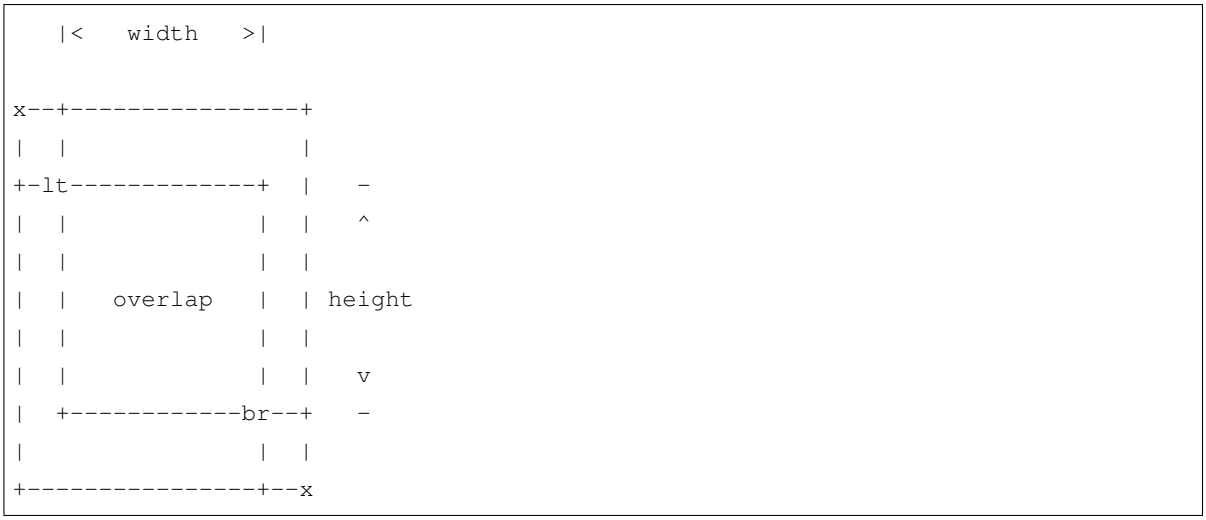
- Case2: both two corners are inside the gt box.

To ensure IoU of generated box and gt box is larger than `min_overlap`:

$$\frac{(w-2*r)*(h-2*r)}{w*h} \geq iou \quad \Rightarrow \quad 4r^2 - 2(w+h)r + (1-iou)*w*h \geq 0$$

$$a = 4, \quad b = -2(w+h), \quad c = (1-iou)*w*hr \leq \frac{-b-\sqrt{b^2-4*a*c}}{2*a}$$

- Case3: both two corners are outside the gt box.



To ensure IoU of generated box and gt box is larger than `min_overlap`:

$$\frac{w*h}{(w+2*r)*(h+2*r)} \geq iou \quad \Rightarrow \quad 4*iou*r^2 + 2*iou*(w+h)r + (iou-1)*w*h \leq 0$$

$$a = 4*iou, \quad b = 2*iou*(w+h), \quad c = (iou-1)*w*h$$

$$r \leq \frac{-b+\sqrt{b^2-4*a*c}}{2*a}$$

参数

- **det_size** (*list[int]*) –Shape of object.
- **min_overlap** (*float*) –Min IoU with ground truth for boxes generated by keypoints inside the gaussian kernel.

返回 Radius of gaussian kernel.

返回类型 radius (int)

mmdet.models.utils.**gen_gaussian_target**(*heatmap*, *center*, *radius*, *k=1*)

Generate 2D gaussian heatmap.

参数

- **heatmap** (*Tensor*) –Input heatmap, the gaussian kernel will cover on it and maintain the max value.
- **center** (*list[int]*) –Coord of gaussian kernel’s center.

- **radius** (*int*) –Radius of gaussian kernel.
- **k** (*int*) –Coefficient of gaussian kernel. Default: 1.

返回 Updated heatmap covered by gaussian kernel.

返回类型 out_heatmap (Tensor)

mmdet.models.utils.**interpolate_as**(*source*, *target*, *mode='bilinear'*, *align_corners=False*)

Interpolate the source to the shape of the target.

Interpolate the source to the shape of target. The input must be a Tensor, but the target can be a Tensor or a np.ndarray with the shape (…, target_h, target_w).

参数

- **source** (*Tensor*) –A 3D/4D Tensor with the shape (N, H, W) or (N, C, H, W).
- **target** (*Tensor | np.ndarray*) –The interpolation target with the shape (…, target_h, target_w).
- **mode** (*str*) –Algorithm used for interpolation. The options are the same as those in F.interpolate(). Default: `'bilinear'`.
- **align_corners** (*bool*) –The same as the argument in F.interpolate().

返回 The interpolated source Tensor.

返回类型 Tensor

mmdet.models.utils.**make_divisible**(*value*, *divisor*, *min_value=None*, *min_ratio=0.9*)

Make divisible function.

This function rounds the channel number to the nearest value that can be divisible by the divisor. It is taken from the original tf repo. It ensures that all layers have a channel number that is divisible by divisor. It can be seen here: https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet/mobilenet.py # noqa

参数

- **value** (*int*) –The original channel number.
- **divisor** (*int*) –The divisor to fully divide the channel number.
- **min_value** (*int*) –The minimum value of the output channel. Default: None, means that the minimum value equal to the divisor.
- **min_ratio** (*float*) –The minimum ratio of the rounded channel number to the original channel number. Default: 0.9.

返回 The modified output channel number.

返回类型 int

22.5 mmdet.utils

CHAPTER 23

Indices and tables

- genindex
- search

Python 模块索引

m

mmdet.core.anchor, 85
mmdet.core.bbox, 95
mmdet.core.evaluation, 126
mmdet.core.mask, 115
mmdet.core.post_processing, 128
mmdet.datasets.samplers, 131
mmdet.models.backbones, 132
mmdet.models.necks, 150
mmdet.models.utils, 162

索引

A

add_gt_() (*mmdet.core.bbox.AssignResult* 方法), 96
adjust_width_group() (*mmdet.models.backbones.RegNet* 方法), 142
anchor_inside_flags() (在 *mmdet.core.anchor* 模块中), 95
AnchorGenerator (*mmdet.core.anchor* 中的类), 85
areas (*mmdet.core.mask.BaseInstanceMasks property*), 115
areas (*mmdet.core.mask.BitmapMasks property*), 119
areas (*mmdet.core.mask.PolygonMasks property*), 122
assign() (*mmdet.core.bbox.BaseAssigner* 方法), 97
assign() (*mmdet.core.bbox.CenterRegionAssigner* 方法), 99
assign() (*mmdet.core.bbox.MaxIoUAssigner* 方法), 103
assign() (*mmdet.core.bbox.RegionAssigner* 方法), 105
assign_one_hot_gt_indices() (*mmdet.core.bbox.CenterRegionAssigner* 方法), 99
assign_wrt_overlaps() (*mmdet.core.bbox.MaxIoUAssigner* 方法), 104
AssignResult (*mmdet.core.bbox* 中的类), 95
average_precision() (在 *mmdet.core.evaluation* 模块中), 126

B

BaseAssigner (*mmdet.core.bbox* 中的类), 97
BaseBBoxCoder (*mmdet.core.bbox* 中的类), 97
BaseInstanceMasks (*mmdet.core.mask* 中的类), 115
BaseSampler (*mmdet.core.bbox* 中的类), 97
bbox2distance() (在 *mmdet.core.bbox* 模块中), 110
bbox2result() (在 *mmdet.core.bbox* 模块中), 110
bbox2roi() (在 *mmdet.core.bbox* 模块中), 110
bbox_cxcywh_to_xyxy() (在 *mmdet.core.bbox* 模块中), 111
bbox_flip() (在 *mmdet.core.bbox* 模块中), 111
bbox_mapping() (在 *mmdet.core.bbox* 模块中), 111
bbox_mapping_back() (在 *mmdet.core.bbox* 模块中), 111
bbox_overlaps() (在 *mmdet.core.bbox* 模块中), 111
bbox_rescale() (在 *mmdet.core.bbox* 模块中), 114
bbox_xyxy_to_cxcywh() (在 *mmdet.core.bbox* 模块中), 114
bboxes (*mmdet.core.bbox.SamplingResult property*), 107
BboxOverlaps2D (*mmdet.core.bbox* 中的类), 98
BFP (*mmdet.models.necks* 中的类), 150
BitmapMasks (*mmdet.core.mask* 中的类), 118
build_assigner() (在 *mmdet.core.bbox* 模块中), 114
build_bbox_coder() (在 *mmdet.core.bbox* 模块中), 114
build_linear_layer() (在 *mmdet.models.utils* 模块中), 171
build_sampler() (在 *mmdet.core.bbox* 模块中), 114
build_transformer() (在 *mmdet.models.utils* 模块中), 171

C

calc_region() (在 *mmdet.core.anchor* 模块中), 95
CenterRegionAssigner (*mmdet.core.bbox* 中的类), 98
ChannelMapper (*mmdet.models.necks* 中的类), 151
CombinedSampler (*mmdet.core.bbox* 中的类), 100
ConvUpsample (*mmdet.models.utils* 中的类), 163
crop() (*mmdet.core.mask.BaseInstanceMasks* 方法), 115
crop() (*mmdet.core.mask.BitmapMasks* 方法), 119
crop() (*mmdet.core.mask.PolygonMasks* 方法), 122
crop_and_resize() (*mmdet.core.mask.BaseInstanceMasks* 方法), 115
crop_and_resize() (*mmdet.core.mask.BitmapMasks* 方法), 119
crop_and_resize() (*mmdet.core.mask.PolygonMasks* 方法), 122
CSPDarknet (*mmdet.models.backbones* 中的类), 132
CSPLayer (*mmdet.models.utils* 中的类), 162
CTResNetNeck (*mmdet.models.necks* 中的类), 150

D

Darknet (*mmdet.models.backbones* 中的类), 133
decode() (*mmdet.core.bbox.BaseBBoxCoder* 方法), 97
decode() (*mmdet.core.bbox.DeltaXYWHBBoxCoder* 方法), 101
decode() (*mmdet.core.bbox.PseudoBBoxCoder* 方法), 104
decode() (*mmdet.core.bbox.TBLRBBoxCoder* 方法), 109
DeltaXYWHBBoxCoder (*mmdet.core.bbox* 中的类), 100
DetectoRS_ResNet (*mmdet.models.backbones* 中的类), 136
DetectoRS_ResNeXt (*mmdet.models.backbones* 中的类), 135
DetrTransformerDecoder (*mmdet.models.utils* 中的类), 164
DetrTransformerDecoderLayer (*mmdet.models.utils* 中的类), 164
DilatedEncoder (*mmdet.models.necks* 中的类), 152
distance2bbox() (在 *mmdet.core.bbox* 模块中), 114
DistEvalHook (*mmdet.core.evaluation* 中的类), 126
DistributedGroupSampler (*mmdet.datasets.samplers* 中的类), 131
DistributedSampler (*mmdet.datasets.samplers* 中的类), 131
DynamicConv (*mmdet.models.utils* 中的类), 164

E

encode() (*mmdet.core.bbox.BaseBBoxCoder* 方法), 97
encode() (*mmdet.core.bbox.DeltaXYWHBBoxCoder* 方法), 101
encode() (*mmdet.core.bbox.PseudoBBoxCoder* 方法), 104
encode() (*mmdet.core.bbox.TBLRBBoxCoder* 方法), 110
encode_mask_results() (在 *mmdet.core.mask* 模块中), 124
eval_map() (在 *mmdet.core.evaluation* 模块中), 126
eval_recalls() (在 *mmdet.core.evaluation* 模块中), 127
EvalHook (*mmdet.core.evaluation* 中的类), 126
expand() (*mmdet.core.mask.BaseInstanceMasks* 方法), 116
expand() (*mmdet.core.mask.BitmapMasks* 方法), 119
expand() (*mmdet.core.mask.PolygonMasks* 方法), 122

F

fast_nms() (在 *mmdet.core.post_processing* 模块中), 128
flip() (*mmdet.core.mask.BaseInstanceMasks* 方法), 116
flip() (*mmdet.core.mask.BitmapMasks* 方法), 119
flip() (*mmdet.core.mask.PolygonMasks* 方法), 123
forward() (*mmdet.models.backbones.CSPDarknet* 方法), 133
forward() (*mmdet.models.backbones.Darknet* 方法), 134
forward() (*mmdet.models.backbones.DetectoRS_ResNet* 方法), 136
forward() (*mmdet.models.backbones.HourglassNet* 方法), 139

forward() (*mmdet.models.backbones.HRNet 方法*), 138
forward() (*mmdet.models.backbones.MobileNetV2 方法*), 140
forward() (*mmdet.models.backbones.RegNet 方法*), 142
forward() (*mmdet.models.backbones.ResNet 方法*), 147
forward() (*mmdet.models.backbones.SSDVGG 方法*), 149
forward() (*mmdet.models.necks.BFP 方法*), 150
forward() (*mmdet.models.necks.ChannelMapper 方法*), 152
forward() (*mmdet.models.necks.CTResNetNeck 方法*), 150
forward() (*mmdet.models.necks.DilatedEncoder 方法*), 152
forward() (*mmdet.models.necks.FPG 方法*), 154
forward() (*mmdet.models.necks.FPN 方法*), 155
forward() (*mmdet.models.necks.FPN_CARAFE 方法*), 156
forward() (*mmdet.models.necks.HRFPN 方法*), 157
forward() (*mmdet.models.necks.NASFCOS_FPN 方法*), 157
forward() (*mmdet.models.necks.NASFPN 方法*), 158
forward() (*mmdet.models.necks.PAFPN 方法*), 159
forward() (*mmdet.models.necks.RFP 方法*), 159
forward() (*mmdet.models.necks.SSDNeck 方法*), 160
forward() (*mmdet.models.necks.YOLOV3Neck 方法*), 161
forward() (*mmdet.models.necks.YOLOXPAFPN 方法*), 162
forward() (*mmdet.models.utils.ConvUpsample 方法*), 163
forward() (*mmdet.models.utils.CSPLayer 方法*), 163
forward() (*mmdet.models.utils.DetrTransformerDecoder 方法*), 164
forward() (*mmdet.models.utils.DynamicConv 方法*), 165
forward() (*mmdet.models.utils.InvertedResidual 方法*), 166
forward() (*mmdet.models.utils.LearnedPositionalEncoding 方法*), 167
forward() (*mmdet.models.utils.NormedConv2d 方法*), 167
forward() (*mmdet.models.utils.NormedLinear 方法*), 167
forward() (*mmdet.models.utils.SELayer 方法*), 169
forward() (*mmdet.models.utils.SimplifiedBasicBlock 方法*), 169
forward() (*mmdet.models.utils.SinePositionalEncoding 方法*), 170
forward() (*mmdet.models.utils.Transformer 方法*), 170
FPG (*mmdet.models.necks 中的类*), 153
FPN (*mmdet.models.necks 中的类*), 154
FPN_CARAFE (*mmdet.models.necks 中的类*), 155

G

gaussian_radius() (*在 mmdet.models.utils 模块中*), 171
gen_base_anchors() (*mmdet.core.anchor.AnchorGenerator 方法*), 86
gen_base_anchors() (*mmdet.core.anchor.YOLOAnchorGenerator 方法*), 93
gen_gaussian_target() (*在 mmdet.models.utils 模块中*), 173
gen_single_level_base_anchors() (*mmdet.core.anchor.AnchorGenerator 方法*), 86
gen_single_level_base_anchors() (*mmdet.core.anchor.LegacyAnchorGenerator 方法*), 91
gen_single_level_base_anchors() (*mmdet.core.anchor.YOLOAnchorGenerator 方法*), 94
generate_regnet() (*mmdet.models.backbones.RegNet 方法*), 142
get_classes() (*在 mmdet.core.evaluation 模块中*), 127
get_extra_property() (*mmdet.core.bbox.AssignResult 方法*), 96
get_gt_priorities()

(*mmdet.core.bbox.CenterRegionAssigner* 方法), 100
get_stages_from_blocks() (*mmdet.models.backbones.RegNet* 方法), 142
grid_anchors() (*mmdet.core.anchor.AnchorGenerator* 方法), 87
grid_priors() (*mmdet.core.anchor.AnchorGenerator* 方法), 87
grid_priors() (*mmdet.core.anchor.MlvlPointGenerator* 方法), 91
GroupSampler (*mmdet.datasets.samplers* 中的类), 131
gt_inds (*mmdet.core.bbox.AssignResult* 属性), 95

H

HourglassNet (*mmdet.models.backbones* 中的类), 138
HRFPN (*mmdet.models.necks* 中的类), 156
HRNet (*mmdet.models.backbones* 中的类), 136

I

images_to_levels() (在 *mmdet.core.anchor* 模块中), 95
info (*mmdet.core.bbox.AssignResult property*), 96
info (*mmdet.core.bbox.SamplingResult property*), 107
init_weights() (*mmdet.models.backbones.DetectoRS_ResNet* 方法), 136
init_weights() (*mmdet.models.backbones.HourglassNet* 方法), 139
init_weights() (*mmdet.models.backbones.SSDVGG* 方法), 149
init_weights() (*mmdet.models.necks.CTResNetNeck* 方法), 151
init_weights() (*mmdet.models.necks.FPN_CARAFE* 方法), 156
init_weights() (*mmdet.models.necks.NASFCOS_FPN* 方法), 157
init_weights() (*mmdet.models.necks.RFP* 方法), 159
init_weights() (*mmdet.models.utils.Transformer* 方法), 171
InstanceBalancedPosSampler (*mmdet.core.bbox* 中的类), 101
interpolate_as() (在 *mmdet.models.utils* 模块中), 174
InvertedResidual (*mmdet.models.utils* 中的类), 165
IoUBalancedNegSampler (*mmdet.core.bbox* 中的类), 101

L

labels (*mmdet.core.bbox.AssignResult* 属性), 96
LearnedPositionalEncoding (*mmdet.models.utils* 中的类), 166
LegacyAnchorGenerator (*mmdet.core.anchor* 中的类), 89

M

make_conv_res_block() (*mmdet.models.backbones.Darknet* 静态方法), 135
make_divisible() (在 *mmdet.models.utils* 模块中), 174
make_layer() (*mmdet.models.backbones.MobileNetV2* 方法), 140
make_res_layer() (*mmdet.models.backbones.DetectoRS_ResNet* 方法), 136
make_res_layer() (*mmdet.models.backbones.DetectoRS_ResNeXt* 方法), 135
make_res_layer() (*mmdet.models.backbones.Res2Net* 方法), 144
make_res_layer() (*mmdet.models.backbones.ResNeSt* 方法), 144
make_res_layer() (*mmdet.models.backbones.ResNet* 方法), 147
make_res_layer() (*mmdet.models.backbones.ResNeXt* 方法), 145
make_stage_plugins() (*mmdet.models.backbones.ResNet* 方法), 147
mask_target() (在 *mmdet.core.mask* 模块中), 124
max_overlaps (*mmdet.core.bbox.AssignResult* 属性), 96
MaxIoUAssigner (*mmdet.core.bbox* 中的类), 102
merge_aug_bboxes() (在 *mmdet.core.post_processing* 模块中), 129

merge_aug_masks() (在 mmdet.core.post_processing 模块中), 129
merge_aug_proposals() (在 mmdet.core.post_processing 模块中), 129
merge_aug_scores() (在 mmdet.core.post_processing 模块中), 130
MlvlPointGenerator (mmdet.core.anchor 中的类), 91
mmdet.core.anchor 模块, 85
mmdet.core.bbox 模块, 95
mmdet.core.evaluation 模块, 126
mmdet.core.mask 模块, 115
mmdet.core.post_processing 模块, 128
mmdet.datasets.samplers 模块, 131
mmdet.models.backbones 模块, 132
mmdet.models.necks 模块, 150
mmdet.models.utils 模块, 162
MobileNetV2 (mmdet.models.backbones 中的类), 139
multiclass_nms() (在 mmdet.core.post_processing 模块中), 130

N

NASFCOS_FPN (mmdet.models.necks 中的类), 157
NASFPN (mmdet.models.necks 中的类), 158
norm1 (mmdet.models.backbones.HRNet property), 138
norm1 (mmdet.models.backbones.ResNet property), 148
norm1 (mmdet.models.utils.SimplifiedBasicBlock property), 169
norm2 (mmdet.models.backbones.HRNet property), 138
norm2 (mmdet.models.utils.SimplifiedBasicBlock property), 169
NormedConv2d (mmdet.models.utils 中的类), 167
NormedLinear (mmdet.models.utils 中的类), 167
num_base_anchors (mmdet.core.anchor.AnchorGenerator property), 87
num_base_priors (mmdet.core.anchor.AnchorGenerator property), 87
num_base_priors (mmdet.core.anchor.MlvlPointGenerator property), 92
num_gts (mmdet.core.bbox.AssignResult 属性), 95
num_levels (mmdet.core.anchor.AnchorGenerator property), 87
num_levels (mmdet.core.anchor.MlvlPointGenerator property), 92
num_levels (mmdet.core.anchor.YOLOAnchorGenerator property), 94
num_preds (mmdet.core.bbox.AssignResult property), 96

O

OHEMSampler (mmdet.core.bbox 中的类), 104

P

pad() (mmdet.core.mask.BaseInstanceMasks 方法), 116
pad() (mmdet.core.mask.BitmapMasks 方法), 119
pad() (mmdet.core.mask.PolygonMasks 方法), 123
PAFPN (mmdet.models.necks 中的类), 158
plot_iou_recall() (在 mmdet.core.evaluation 模块中), 127
plot_num_recall() (在 mmdet.core.evaluation 模块中), 127
PolygonMasks (mmdet.core.mask 中的类), 121
print_map_summary() (在 mmdet.core.evaluation 模块中), 128
print_recall_summary() (在 mmdet.core.evaluation 模块中), 128
PseudoBBoxCoder (mmdet.core.bbox 中的类), 104
PseudoSampler (mmdet.core.bbox 中的类), 104

Q

quantize_float() (mmdet.models.backbones.RegNet 静态方法), 142

R

random() (mmdet.core.bbox.AssignResult 类方法), 97
random() (mmdet.core.bbox.SamplingResult 类方法), 107

random() (*mmdet.core.mask.BitmapMasks* 类方法), 119
random() (*mmdet.core.mask.PolygonMasks* 类方法), 123
random_choice() (*mmdet.core.bbox.RandomSampler* 方法), 105
random_choice() (*mmdet.core.bbox.ScoreHLRSampler* 静态方法), 108
RandomSampler (*mmdet.core.bbox* 中的类), 105
RegionAssigner (*mmdet.core.bbox* 中的类), 105
RegNet (*mmdet.models.backbones* 中的类), 140
Res2Net (*mmdet.models.backbones* 中的类), 143
rescale() (*mmdet.core.mask.BaseInstanceMasks* 方法), 116
rescale() (*mmdet.core.mask.BitmapMasks* 方法), 120
rescale() (*mmdet.core.mask.PolygonMasks* 方法), 123
resize() (*mmdet.core.mask.BaseInstanceMasks* 方法), 117
resize() (*mmdet.core.mask.BitmapMasks* 方法), 120
resize() (*mmdet.core.mask.PolygonMasks* 方法), 123
ResLayer (*mmdet.models.utils* 中的类), 168
ResNeSt (*mmdet.models.backbones* 中的类), 144
ResNet (*mmdet.models.backbones* 中的类), 145
ResNetV1d (*mmdet.models.backbones* 中的类), 148
ResNeXt (*mmdet.models.backbones* 中的类), 144
responsible_flags() (*mmdet.core.anchor.YOLOAnchorGenerator* 方法), 94
RFP (*mmdet.models.necks* 中的类), 159
rfp_forward() (*mmdet.models.backbones.DetectoRS_ResNet* 方法), 136
roi2bbox() (在 *mmdet.core.bbox* 模块中), 115
rotate() (*mmdet.core.mask.BaseInstanceMasks* 方法), 117
rotate() (*mmdet.core.mask.BitmapMasks* 方法), 120
rotate() (*mmdet.core.mask.PolygonMasks* 方法), 123

S

sample() (*mmdet.core.bbox.BaseSampler* 方法), 98
sample() (*mmdet.core.bbox.PseudoSampler* 方法), 104
sample() (*mmdet.core.bbox.ScoreHLRSampler* 方法), 109
sample_via_interval() (*mmdet.core.bbox.IoUBalancedNegSampler* 方法), 102
SamplingResult (*mmdet.core.bbox* 中的类), 106
ScoreHLRSampler (*mmdet.core.bbox* 中的类), 108
SELayer (*mmdet.models.utils* 中的类), 168
set_extra_property() (*mmdet.core.bbox.AssignResult* 方法), 97
shear() (*mmdet.core.mask.BaseInstanceMasks* 方法), 117
shear() (*mmdet.core.mask.BitmapMasks* 方法), 120
shear() (*mmdet.core.mask.PolygonMasks* 方法), 123
SimplifiedBasicBlock (*mmdet.models.utils* 中的类), 169
SinePositionalEncoding (*mmdet.models.utils* 中的类), 169
single_level_grid_anchors() (*mmdet.core.anchor.AnchorGenerator* 方法), 88
single_level_grid_priors() (*mmdet.core.anchor.AnchorGenerator* 方法), 88
single_level_grid_priors() (*mmdet.core.anchor.MlvlPointGenerator* 方法), 92
single_level_responsible_flags() (*mmdet.core.anchor.YOLOAnchorGenerator* 方法), 94
single_level_valid_flags() (*mmdet.core.anchor.AnchorGenerator* 方法), 88
single_level_valid_flags() (*mmdet.core.anchor.MlvlPointGenerator* 方法), 92
slice_as() (*mmdet.models.necks.FPN_CARAFE* 方法), 156
sparse_priors() (*mmdet.core.anchor.AnchorGenerator* 方法), 89
sparse_priors() (*mmdet.core.anchor.MlvlPointGenerator* 方法), 93
split_combined_polys() (在 *mmdet.core.mask* 模块中), 125
SSDNeck (*mmdet.models.necks* 中的类), 159

SSDVGG (*mmdet.models.backbones* 中的类), 148

T

TBLRBBoxCoder (*mmdet.core.bbox* 中的类), 109
tensor_add() (*mmdet.models.necks.FPN_CARAFE* 方法), 156
to() (*mmdet.core.bbox.SamplingResult* 方法), 108
to_bitmap() (*mmdet.core.mask.PolygonMasks* 方法), 123
to_ndarray() (*mmdet.core.mask.BaseInstanceMasks* 方法), 117
to_ndarray() (*mmdet.core.mask.BitmapMasks* 方法), 120
to_ndarray() (*mmdet.core.mask.PolygonMasks* 方法), 123
to_tensor() (*mmdet.core.mask.BaseInstanceMasks* 方法), 118
to_tensor() (*mmdet.core.mask.BitmapMasks* 方法), 121
to_tensor() (*mmdet.core.mask.PolygonMasks* 方法), 123
train() (*mmdet.models.backbones.CSPDarknet* 方法), 133
train() (*mmdet.models.backbones.Darknet* 方法), 135
train() (*mmdet.models.backbones.HRNet* 方法), 138
train() (*mmdet.models.backbones.MobileNetV2* 方法), 140
train() (*mmdet.models.backbones.ResNet* 方法), 148
Transformer (*mmdet.models.utils* 中的类), 170
translate() (*mmdet.core.mask.BaseInstanceMasks* 方法), 118
translate() (*mmdet.core.mask.BitmapMasks* 方法), 121
translate() (*mmdet.core.mask.PolygonMasks* 方法), 123
TridentResNet (*mmdet.models.backbones* 中的类), 149

V

valid_flags() (*mmdet.core.anchor.AnchorGenerator* 方法), 89
valid_flags() (*mmdet.core.anchor.MlvlPointGenerator* 方法), 93

Y

YOLOAnchorGenerator (*mmdet.core.anchor* 中的类), 93
YOLOV3Neck (*mmdet.models.necks* 中的类), 160
YOLOXPAFPN (*mmdet.models.necks* 中的类), 161

模块

mmdet.core.anchor, 85
mmdet.core.bbox, 95
mmdet.core.evaluation, 126
mmdet.core.mask, 115
mmdet.core.post_processing, 128
mmdet.datasets.samplers, 131
mmdet.models.backbones, 132
mmdet.models.necks, 150
mmdet.models.utils, 162