MMDetection

发行版本 *2.28.1*

MMDetection Authors

2023 年 02 月 01 日

开始你的第一步

CHAPTER 1

依赖

- Linux 和 macOS （Windows 理论上支持）
- Python 3.7 +
- 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 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。您可以参考下一节中的分步安装说明。

```
conda create -n openmmlab python=3.7 pytorch==1.6.0 cudatoolkit=10.1 torchvision -c␣
↪pytorch -y
conda activate openmmlab
pip install openmim
mim install mmcv-full
git clone https://github.com/open-mmlab/mmdetection.git
cd mmdetection
pip install -r requirements/build.txt
pip install -v -e .
```

2.2 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。

2.3 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
```

PyTorch 在 1.x.0 和 1.x.1 之间通常是兼容的，故 mmcv-full 只提供 1.x.0 的编译包。如果你的 PyTorch 版本是 1.x.1，你可以放心地安装在 1.x.0 版本编译的 mmcv-full。

```
# 我们可以忽略 PyTorch 的小版本号
pip install mmcv-full -f https://download.openmmlab.com/mmcv/dist/cu110/torch1.7/
↪index.html
```

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
# 安装 albumentations 依赖
pip install -r requirements/albu.txt
```

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` 。

(4) 如果希望使用 `albumentations`，我们建议使用 `pip install -r requirements/albu.txt` 或者 `pip install -U albumentations --no-binary qudida,albumentations` 进行安装。如果简单地使用 `pip install albumentations>=0.3.2` 进行安装，则会同时安装 `opencv-python-headless`（即便已经安装了 `opencv-python` 也会再次安装）。我们建议在安装 `albumentations` 后检查环境，以确保没有同时安装 `opencv-python` 和 `opencv-python-headless`，因为同时安装可能会导致一些问题。更多细节请参考官方文档。

2.4 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，可以进行模型训练（需要 MMCV 版本 >= 1.4.4)、测试或者推理，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训练/测试/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：

2.5 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
```

2.6 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

从 MMDetection V2.0 起，我们只通过阿里云维护模型库。V1.x 版本的模型已经弃用。

4.2 共同设置

- 所有模型都是在 `coco_2017_train` 上训练，在 `coco_2017_val` 上测试。
- 我们使用分布式训练。
- 所有 pytorch-style 的 ImageNet 预训练主干网络来自 PyTorch 的模型库，caffe-style 的预训练主干网络来自 detectron2 最新开源的模型。
- 为了与其他代码库公平比较，文档中所写的 GPU 内存是 8 个 GPU 的 `torch.cuda.max_memory_allocated()` 的最大值，此值通常小于 nvidia-smi 显示的值。
- 我们以网络 forward 和后处理的时间加和作为推理时间，不包含数据加载时间。所有结果通过 benchmark.py 脚本计算所得。该脚本会计算推理 2000 张图像的平均时间。

4.3 ImageNet 预训练模型

通过 ImageNet 分类任务预训练的主干网络进行初始化是很常见的操作。所有预训练模型的链接都可以在 open_mmlab 中找到。根据 `img_norm_cfg` 和原始权重，我们可以将所有 ImageNet 预训练模型分为以下几种情况：

- TorchVision：torchvision 模型权重，包含 ResNet50, ResNet101。`img_norm_cfg` 为 `dict(mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)`。
- Pycls：pycls 模型权重，包含 RegNetX。`img_norm_cfg` 为 `dict( mean=[103.530, 116.280, 123.675], std=[57.375, 57.12, 58.395], to_rgb=False)`。
- MSRA styles：MSRA 模型权重，包含 ResNet50_Caffe，ResNet101_Caffe。`img_norm_cfg` 为 `dict( mean=[103.530, 116.280, 123.675], std=[1.0, 1.0, 1.0], to_rgb=False)`。
- Caffe2 styles：现阶段只包含 ResNext101_32x8d。`img_norm_cfg` 为 `dict(mean=[103.530, 116.280, 123.675], std=[57.375, 57.120, 58.395], to_rgb=False)`。
- Other styles: SSD 的 `img_norm_cfg` 为 `dict(mean=[123.675, 116.28, 103.53], std=[1, 1, 1], to_rgb=True)`，YOLOv3 的 `img_norm_cfg` 为 `dict(mean=[0, 0, 0], std=[255., 255., 255.], to_rgb=True)`。

MMdetection 常用到的主干网络细节如下表所示：

4.4 Baselines

4.4.1 RPN

请参考 RPN。

4.4.2 Faster R-CNN

请参考 Faster R-CNN。

4.4.3 Mask R-CNN

请参考 Mask R-CNN。

4.4.4 Fast R-CNN (使用提前计算的 proposals)

请参考 Fast R-CNN。

4.4.5 RetinaNet

请参考 RetinaNet。

4.4.6 Cascade R-CNN and Cascade Mask R-CNN

请参考 Cascade R-CNN。

4.4.7 Hybrid Task Cascade (HTC)

请参考 HTC。

4.4.8 SSD

请参考 SSD。

4.4.9 Group Normalization (GN)

请参考 Group Normalization。

4.4.10 Weight Standardization

请参考 Weight Standardization。

4.4.11 Deformable Convolution v2

请参考 Deformable Convolutional Networks。

4.4.12 CARAFE: Content-Aware ReAssembly of FEatures

请参考 CARAFE。

4.4.13 Instaboost

请参考 Instaboost。

4.4.14 Libra R-CNN

请参考 Libra R-CNN。

4.4.15 Guided Anchoring

请参考 Guided Anchoring。

4.4.16 FCOS

请参考 FCOS。

4.4.17 FoveaBox

请参考 FoveaBox。

4.4.18 RepPoints

请参考 RepPoints。

4.4.19 FreeAnchor

请参考 FreeAnchor。

4.4.20 Grid R-CNN (plus)

请参考 Grid R-CNN。

4.4.21 GHM

请参考 GHM。

4.4.22 GCNet

请参考 GCNet。

4.4.23 HRNet

请参考 HRNet。

4.4.24 Mask Scoring R-CNN

请参考 Mask Scoring R-CNN。

4.4.25 Train from Scratch

请参考 Rethinking ImageNet Pre-training。

4.4.26 NAS-FPN

请参考 NAS-FPN。

4.4.27 ATSS

请参考 ATSS。

4.4.28 FSAF

请参考 FSAF。

4.4.29 RegNetX

请参考 RegNet。

4.4.30 Res2Net

请参考 Res2Net。

4.4.31 GRoIE

请参考 GRoIE。

4.4.32 Dynamic R-CNN

请参考 Dynamic R-CNN。

4.4.33 PointRend

请参考 PointRend。

4.4.34 DetectoRS

请参考 DetectoRS。

4.4.35 Generalized Focal Loss

请参考 Generalized Focal Loss。

4.4.36 CornerNet

请参考 CornerNet。

4.4.37 YOLOv3

请参考 YOLOv3。

4.4.38 PAA

请参考 PAA。

4.4.39 SABL

请参考 SABL。

4.4.40 CentripetalNet

请参考 CentripetalNet。

4.4.41 ResNeSt

请参考 ResNeSt。

4.4.42 DETR

请参考 DETR。

4.4.43 Deformable DETR

请参考 Deformable DETR。

4.4.44 AutoAssign

请参考 AutoAssign。

4.4.45 YOLOF

请参考 YOLOF。

4.4.46 Seesaw Loss

请参考 Seesaw Loss。

4.4.47 CenterNet

请参考 CenterNet。

4.4.48 YOLOX

请参考 YOLOX。

4.4.49 PVT

请参考 PVT。

4.4.50 SOLO

请参考 SOLO。

4.4.51 QueryInst

请参考 QueryInst。

4.4.52 RF-Next

请参考 RF-Next.

4.4.53 Other datasets

我们还在 PASCAL VOC，Cityscapes 和 WIDER FACE 上对一些方法进行了基准测试。

4.4.54 Pre-trained Models

我们还通过多尺度训练和更长的训练策略来训练用 ResNet-50 和 RegNetX-3.2G 作为主干网络的 Faster R-CNN 和 Mask R-CNN。这些模型可以作为下游任务的预训练模型。

4.5 速度基准

4.5.1 训练速度基准

我们提供 analyze_logs.py 来得到训练中每一次迭代的平均时间。示例请参考 Log Analysis。

我们与其他流行框架的 Mask R-CNN 训练速度进行比较（数据是从 detectron2 复制而来）。在 mmdetection 中，我们使用 mask_rcnn_r50_caffe_fpn_poly_1x_coco_v1.py 进行基准测试。它与 detectron2 的 mask_rcnn_R_50_FPN_noaug_1x.yaml 设置完全一样。同时，我们还提供了模型权重和训练 log 作为参考。为了跳过 GPU 预热时间，吞吐量按照 100-500 次迭代之间的平均吞吐量来计算。

4.5.2 推理时间基准

我们提供 benchmark.py 对推理时间进行基准测试。此脚本将推理 2000 张图片并计算忽略前 5 次推理的平均推理时间。可以通过设置 `LOG-INTERVAL` 来改变 log 输出间隔（默认为 50）。

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

模型库中，所有模型在基准测量推理时间时都没设置 `fuse-conv-bn`, 此设置可以使推理时间更短。

4.6 与 Detectron2 对比

我们在速度和精度方面对 mmdetection 和 Detectron2 进行对比。对比所使用的 detectron2 的 commit id 为 185c27e(30/4/2020)。为了公平对比，我们所有的实验都在同一机器下进行。

4.6.1 硬件

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

4.6.2 软件环境

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

4.6.3 精度

4.6.4 训练速度

训练速度使用 s/iter 来度量。结果越低越好。

4.6.5 推理速度

推理速度通过单张 GPU 下的 fps(img/s) 来度量，越高越好。为了与 Detectron2 保持一致，我们所写的推理时间除去了数据加载时间。对于 Mask RCNN，我们去除了后处理中 RLE 编码的时间。我们在括号中给出了官方给出的速度。由于硬件差异，官方给出的速度会比我们所测试得到的速度快一些。

4.6.6 训练内存

CHAPTER 5

中文解读文案汇总

5.1 1 官方解读文案

5.1.1 1.1 框架解读

- **轻松掌握 MMDetection 整体构建流程 (一)**
- **轻松掌握 MMDetection 整体构建流程 (二)**
- **轻松掌握 MMDetection 中 Head 流程**

5.1.2 1.2 算法解读

- **轻松掌握 MMDetection 中常用算法 (一)：RetinaNet 及配置详解**
- **轻松掌握 MMDetection 中常用算法 (二)：Faster R-CNN|Mask R-CNN**
- 轻松掌握 MMDetection 中常用算法 (三)：FCOS
- 轻松掌握 MMDetection 中常用算法 (四)：ATSS
- 轻松掌握 MMDetection 中常用算法 (五)：Cascade R-CNN
- 轻松掌握 MMDetection 中常用算法 (六)：YOLOF
- 轻松掌握 MMDetection 中常用算法 (七)：CenterNet
- 轻松掌握 MMDetection 中常用算法 (八)：YOLACT

- 轻松掌握 MMDetection 中常用算法 (九)：AutoAssign
- YOLOX 在 MMDetection 中复现全流程解析
- 喂喂喂！你可以减重了！小模型 - MMDetection 新增 SSDLite 、MobileNetV2YOLOV3 两大经典算法

5.1.3 1.3 工具解读

- OpenMMLab 中混合精度训练 AMP 的正确打开方式
- 小白都能看懂！手把手教你使用混淆矩阵分析目标检测
- MMDetection 图像缩放 Resize 详细说明 OpenMMLab
- 拿什么拯救我的 4G 显卡
- MMDet 居然能用 MMCls 的 Backbone？论配置文件的打开方式

5.1.4 1.4 知乎问答

- COCO 数据集上 1x 模式下为什么不采用多尺度训练?
- MMDetection 中 SOTA 论文源码中将训练过程中 BN 层的 eval 打开?
- 基于 PyTorch 的 MMDetection 中训练的随机性来自何处?
- 单阶段、双阶段、anchor-based、anchor-free 这四者之间有什么联系吗?
- 目标检测的深度学习方法，有推荐的书籍或资料吗?
- 大佬们，刚入学研究生，想入门目标检测，有什么学习路线可以入门的?
- 目标检测领域还有什么可以做的?
- 如何看待 Transformer 在 CV 上的应用前景，未来有可能替代 CNN 吗?
- MMDetection 如何学习源码?
- 如何具体上手实现目标检测呢?

5.1.5 1.5 其他

- **不得不知的 MMDetection 学习路线 (个人经验版)**
- OpenMMLab 社区专访之 YOLOX 复现篇

5.2 2 社区解读文案

- 手把手带你实现经典检测网络 Mask R-CNN 的推理

CHAPTER 6

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

6.1 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。

6.1.1 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 。

6.1.2 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
```

(续下页)

(接上页)

```
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())
```

6.1.3 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}] \
```

(续下页)

(接上页)

```
    [--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
```

视频样例，显卡加速版本

这是在视频样例上进行推理的脚本，使用显卡加速。

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

运行样例：

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

6.2 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。

6.2.1 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。我们提供了下载 COCO 等数据集的脚本，你可以运行 `python tools/misc/download_dataset.py --dataset-name coco2017` 下载 COCO 数据集。

```
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
```

PanopticFPN 等全景分割模型需要额外的 COCO Panoptic 数据集，你可以下载并解压它们到 `coco/annotations` 文件夹下。文件夹会是如下结构：

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

(续下页)

(接上页)

```
|   |   |   ├── panoptic_val2017
|   |   ├── train2017
|   |   ├── val2017
|   |   ├── test2017
```

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
```

6.2.2 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
- CPU 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]

# CPU 测试：禁用 GPU 并运行单 GPU 测试脚本
export CUDA_VISIBLE_DEVICES=-1
python tools/test.py \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    [--out ${RESULT_FILE}] \
    [--eval ${EVAL_METRICS}] \
```

(续下页)

(接上页)

```
    [--show]

# 单节点多 GPU 测试
bash tools/dist_test.sh \
    ${CONFIG_FILE} \
    ${CHECKPOINT_FILE} \
    ${GPU_NUM} \
    [--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.evaluation()` 函数中，仅在测试阶段使用。

6.2.3 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_coco.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。

7. 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 \
    --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。

6.2.4 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。如果你的数据集格式是 VOC 或者 Cityscapes，你可以使用 tools/dataset_converters 内的脚本直接将其转化成 COCO 格式。如果是其他格式，可以使用 images2coco 脚本 进行转换。

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

参数：

- `IMG_PATH`: 图片根路径。
- `CLASSES`: 类列表文本文件名。文本中每一行存储一个类别。
- `OUT`: 输出 json 文件名。默认保存目录和 `IMG_PATH` 在同一级。
- `exclude-extensions`: 待排除的文件后缀名。

在转换完成后，使用如下命令进行测试

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

# CPU 测试：禁用 GPU 并运行单 GPU 测试脚本
```

(续下页)

(接上页)

```
export CUDA_VISIBLE_DEVICES=-1
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} \
    --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`。

6.2.5 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。

6.2.6 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]),
            dict(type='Pad', size_divisor=32),
            dict(type='DefaultFormatBundle'),
            dict(type='Collect', keys=['img']),
        ])
    ]
```

6.3 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。

6.3.1 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。

6.3.2 学习率自动缩放

注意：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。其已经设置在 config/_base_/default_runtime.py 中的 auto_scale_lr.base_batch_size。当配置文件的批次大小为 16 时，学习率会基于该值进行自动缩放。同时，为了不影响其他基于 mmdet 的 codebase，启用自动缩放标志 auto_scale_lr.enable 默认设置为 False。

如果要启用此功能，需在命令添加参数 --auto-scale-lr。并且在启动命令之前，请检查下即将使用的配置文件的名称，因为配置名称指示默认的批处理大小。在默认情况下，批次大小是 8 x 2 = 16，例如：faster_rcnn_r50_caffe_fpn_90k_coco.py 或者 pisa_faster_rcnn_x101_32x4d_fpn_1x_coco.py；若不是默认批次，你可以在配置文件看到像 _NxM_ 字样的，例如：cornernet_hourglass104_mstest_32x3_210e_coco.py 的批次大小是 32 x 3 = 96, 或者 scnet_x101_64x4d_fpn_8x1_20e_coco.py 的批次大小是 8 x 1 = 8。

请记住：如果使用不是默认批次大小为 16 的配置文件，请检查配置文件中的底部，会有 auto_scale_lr.base_batch_size。如果找不到，可以在其继承的 _base_=[xxx] 文件中找到。另外，如果想使用自动缩放学习率的功能，请不要修改这些值。

学习率自动缩放基本用法如下：

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

执行命令之后，会根据机器的 GPU 数量和训练的批次大小对学习率进行自动缩放，缩放方式详见 线性扩展规则，比如：在 4 块 GPU 并且每张 GPU 上有 2 张图片的情况下 lr=0.01，那么在 16 块 GPU 并且每张 GPU 上有 4 张图片的情况下, LR 会自动缩放至 lr=0.08。

如果不启用该功能，则需要根据 线性扩展规则 来手动计算并修改配置文件里面 optimizer.lr 的值。

6.3.3 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。

6.3.4 使用 CPU 训练

使用 CPU 训练的流程和使用单 GPU 训练的流程一致，我们仅需要在训练流程开始前禁用 GPU。

```
export CUDA_VISIBLE_DEVICES=-1
```

之后运行单 GPU 训练脚本即可。

注意：

我们不推荐用户使用 CPU 进行训练，这太过缓慢。我们支持这个功能是为了方便用户在没有 GPU 的机器上进行调试。

6.3.5 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]
```

可选参数和单 GPU 训练的可选参数一致。

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
```

6.3.6 使用多台机器训练

如果您想使用由 ethernet 连接起来的多台机器，您可以使用以下命令:

在第一台机器上:

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

在第二台机器上:

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

但是，如果您不使用高速网路连接这几台机器的话，训练将会非常慢。

6.3.7 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 7

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。

7.1 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。

7.1.1 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': {},
```

(续下页)

(接上页)

```
  '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,
```

(续下页)

(接上页)

```
    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
import mmcv

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

(续下页)

(接上页)

```
    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,
```

(续下页)

(接上页)

```
        annotations=annotations,
        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。

7.2 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/balloon/`，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__
```

(续下页)

(接上页)

```
↪segm_mAP-0.37_20200504_163245-42aa3d00.pth'
```

7.3 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。

7.4 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 8

3: 在标准数据集上训练自定义模型

在本文中，你将知道如何在标准数据集上训练、测试和推理自定义模型。我们将在 cityscapes 数据集上以自定义 Cascade Mask R-CNN R50 模型为例演示整个过程，为了方便说明，我们将 neck 模块中的 `FPN` 替换为 `AugFPN`，并且在训练中的自动增强类中增加 `Rotate` 或 `Translate`。

基本步骤如下所示：

1. 准备标准数据集
2. 准备你的自定义模型
3. 准备配置文件
4. 在标准数据集上对模型进行训练、测试和推理

8.1 准备标准数据集

在本文中，我们使用 cityscapes 标准数据集为例进行说明。

推荐将数据集根路径采用符号链接方式链接到 `$MMDETECTION/data`。

如果你的文件结构不同，你可能需要在配置文件中进行相应的路径更改。标准的文件组织格式如下所示：

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

(续下页)

(接上页)

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

你也可以通过如下方式设定数据集根路径

```
export MMDET_DATASETS=$data_root
```

我们将会使用环境便变量 `$MMDET_DATASETS` 作为数据集的根目录，因此你无需再修改相应配置文件的路径信息。

你需要使用脚本 `tools/dataset_converters/cityscapes.py` 将 cityscapes 标注转化为 coco 标注格式。

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

目前在 `cityscapes` 文件夹中的配置文件所对应模型是采用 COCO 预训练权重进行初始化的。

如果你的网络不可用或者比较慢，建议你先手动下载对应的预训练权重，否则可能在训练开始时候出现错误。

8.2 准备你的自定义模型

第二步是准备你的自定义模型或者训练相关配置。假设你想在已有的 Cascade Mask R-CNN R50 检测模型基础上，新增一个新的 neck 模块 `AugFPN` 去代替默认的 `FPN`，以下是具体实现：

8.2.1 1 定义新的 neck (例如 AugFPN)

首先创建新文件 mmdet/models/necks/augfpn.py.

```
from ..builder import NECKS

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

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

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

8.2.2 2 导入模块

你可以采用两种方式导入模块，第一种是在 mmdet/models/necks/__init__.py 中添加如下内容

```
from .augfpn import AugFPN
```

第二种是增加如下代码到对应配置中，这种方式的好处是不需要改动代码

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

8.2.3 3 修改配置

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

关于自定义模型其余相关细节例如实现新的骨架网络，头部网络、损失函数，以及运行时训练配置例如定义新的优化器、使用梯度裁剪、定制训练调度策略和钩子等，请参考文档自定义模型 和自定义运行时训练配置。

8.3 准备配置文件

第三步是准备训练配置所需要的配置文件。假设你打算基于 cityscapes 数据集，在 Cascade Mask R-CNN R50 中新增 AugFPN 模块，同时增加 Rotate 或者 Translate 数据增强策略，假设你的配置文件位于 configs/cityscapes/ 目录下，并且取名为 cascade_mask_rcnn_r50_augfpn_autoaug_10e_cityscapes.py，则配置信息如下：

```
# 继承 base 配置，然后进行针对性修改
_base_ = [
    '../_base_/models/cascade_mask_rcnn_r50_fpn.py',
    '../_base_/datasets/cityscapes_instance.py', '../_base_/default_runtime.py'
]

model = dict(
    # 设置为 None，表示不加载 ImageNet 预训练权重，
    # 后续可以设置 `load_from` 参数用来加载 COCO 预训练权重
    backbone=dict(init_cfg=None),
    pretrained=None,
    # 使用新增的 `AugFPN` 模块代替默认的 `FPN`
    neck=dict(
        type='AugFPN',
        in_channels=[256, 512, 1024, 2048],
        out_channels=256,
        num_outs=5),
    # 我们也需要将 num_classes 从 80 修改为 8 来匹配 cityscapes 数据集标注
    # 这个修改包括 `bbox_head` 和 `mask_head`.
    roi_head=dict(
        bbox_head=[
            dict(
                type='Shared2FCBBoxHead',
                in_channels=256,
                fc_out_channels=1024,
                roi_feat_size=7,
                # 将 COCO 类别修改为 cityscapes 类别
                num_classes=8,
                bbox_coder=dict(
                    type='DeltaXYWHBBoxCoder',
                    target_means=[0., 0., 0., 0.],
                    target_stds=[0.1, 0.1, 0.2, 0.2]),
                reg_class_agnostic=True,
```

(续下页)

(接上页)

```
                loss_cls=dict(
                    type='CrossEntropyLoss',
                    use_sigmoid=False,
                    loss_weight=1.0),
                loss_bbox=dict(type='SmoothL1Loss', beta=1.0,
                               loss_weight=1.0)),
            dict(
                type='Shared2FCBBoxHead',
                in_channels=256,
                fc_out_channels=1024,
                roi_feat_size=7,
                # 将 COCO 类别修改为 cityscapes 类别
                num_classes=8,
                bbox_coder=dict(
                    type='DeltaXYWHBBoxCoder',
                    target_means=[0., 0., 0., 0.],
                    target_stds=[0.05, 0.05, 0.1, 0.1]),
                reg_class_agnostic=True,
                loss_cls=dict(
                    type='CrossEntropyLoss',
                    use_sigmoid=False,
                    loss_weight=1.0),
                loss_bbox=dict(type='SmoothL1Loss', beta=1.0,
                               loss_weight=1.0)),
            dict(
                type='Shared2FCBBoxHead',
                in_channels=256,
                fc_out_channels=1024,
                roi_feat_size=7,
                # 将 COCO 类别修改为 cityscapes 类别
                num_classes=8,
                bbox_coder=dict(
                    type='DeltaXYWHBBoxCoder',
                    target_means=[0., 0., 0., 0.],
                    target_stds=[0.033, 0.033, 0.067, 0.067]),
                reg_class_agnostic=True,
                loss_cls=dict(
                    type='CrossEntropyLoss',
                    use_sigmoid=False,
                    loss_weight=1.0),
                loss_bbox=dict(type='SmoothL1Loss', beta=1.0, loss_weight=1.0))
        ],
        mask_head=dict(
```

(续下页)

(接上页)

```
            type='FCNMaskHead',
            num_convs=4,
            in_channels=256,
            conv_out_channels=256,
            # 将 COCO 类别修改为 cityscapes 类别
            num_classes=8,
            loss_mask=dict(
                type='CrossEntropyLoss', use_mask=True, loss_weight=1.0))))

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

# 设置每张显卡的批处理大小，同时设置新的训练 pipeline
```

(续下页)

(接上页)

```
data = dict(
    samples_per_gpu=1,
    workers_per_gpu=3,
    # 用新的训练 pipeline 配置覆写 pipeline
    train=dict(dataset=dict(pipeline=train_pipeline)))

# 设置优化器
optimizer = dict(type='SGD', lr=0.01, momentum=0.9, weight_decay=0.0001)
optimizer_config = dict(grad_clip=None)
# 设置定制的学习率策略
lr_config = dict(
    policy='step',
    warmup='linear',
    warmup_iters=500,
    warmup_ratio=0.001,
    step=[8])
runner = dict(type='EpochBasedRunner', max_epochs=10)

# 我们采用 COCO 预训练过的 Cascade Mask R-CNN R50␣
↪模型权重作为初始化权重，可以得到更加稳定的性能
load_from = 'http://download.openmmlab.com/mmdetection/v2.0/cascade_rcnn/cascade_mask_
↪rcnn_r50_fpn_1x_coco/cascade_mask_rcnn_r50_fpn_1x_coco_20200203-9d4dcb24.pth'
```

8.4 训练新模型

为了能够使用新增配置来训练模型，你可以运行如下命令：

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

如果想了解更多用法，可以参考*例子 1*。

8.5 测试和推理

为了能够测试训练好的模型，你可以运行如下命令：

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

如果想了解更多用法，可以参考*例子 1*。

CHAPTER 9

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。

9.1 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。

9.2 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。

9.3 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。

9.4 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(...),
)
```

9.5 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，
```

(续下页)

(接上页)

```
↪'caffe' 意思是步长为2的层为 1x1 卷积。
        init_cfg=dict(type='Pretrained', checkpoint='torchvision://resnet50')),  #␣
↪加载通过 ImageNet 预训练的模型
    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。
```

(续下页)

(接上页)

```
        bbox_roi_extractor=dict(  # 用于 bbox 回归的 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，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
```

(续下页)

(接上页)

```
↪ModulatedDeformRoIPoolingPack。
                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
```

(续下页)

(接上页)

```
            nms_post=1000,  # NMS 要保留的 box 的数量，只在 GARPNHead 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 rpn 和 rcnn 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`GARPNHead`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
```

(续下页)

(接上页)

```
            score_thr=0.05,  # bbox 的分数阈值
            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'])
```

(续下页)

(接上页)

```
]
test_pipeline = [
    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,
```

(续下页)

(接上页)

```
            with_mask=True,
            poly2mask=False),
        dict(type='Resize', img_scale=(1333, 800), keep_ratio=True),
        dict(type='RandomFlip', flip_ratio=0.5),
        dict(
            type='Normalize',
            mean=[123.675, 116.28, 103.53],
            std=[58.395, 57.12, 57.375],
            to_rgb=True),
        dict(type='Pad', size_divisor=32),
        dict(type='DefaultFormatBundle'),
        dict(
            type='Collect',
            keys=['img', 'gt_bboxes', 'gt_labels', 'gt_masks'])
    ]),
val=dict(  # 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。
```

(续下页)

(接上页)

```
            dict(type='LoadImageFromFile'),
            dict(
                type='MultiScaleFlipAug',
                img_scale=(1333, 800),
                flip=False,
                transforms=[
                    dict(type='Resize', keep_ratio=True),
                    dict(type='RandomFlip'),
                    dict(
                        type='Normalize',
                        mean=[123.675, 116.28, 103.53],
                        std=[58.395, 57.12, 57.375],
                        to_rgb=True),
                    dict(type='Pad', size_divisor=32),
                    dict(type='ImageToTensor', keys=['img']),
                    dict(type='Collect', keys=['img'])
                ])
        ],
        samples_per_gpu=2  # 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
```

(续下页)

(接上页)

```
runner = dict(
    type='EpochBasedRunner',  # 将使用的 runner 的类别 (例如 IterBasedRunner 或 EpochBasedRunner)。
    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=[ # 训练期间执行的钩子
        dict(type='TextLoggerHook', by_epoch=False),
        dict(type='TensorboardLoggerHook', by_epoch=False),
        dict(type='MMDetWandbHook', by_epoch=False, # 还支持 Wandb 记录器，它需要安装 `wandb`。
             init_kwargs={'entity': "OpenMMLab", # 用于登录wandb的实体
                          'project': "MMDet", # WandB中的项目名称
                          'config': cfg_dict}), # 检查 https://docs.wandb.ai/ref/python/init 以获取更多初始化参数
    ])  # 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。
```

9.6 常问问题 (FAQ)

9.6.1 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(
```

(续下页)

(接上页)

```
        type='ResNet',
        depth=50,
        num_stages=4,
        out_indices=(0, 1, 2, 3),
        frozen_stages=1,
        norm_cfg=dict(type='BN', requires_grad=True),
        norm_eval=True,
        style='pytorch'),
    neck=dict(...),
    rpn_head=dict(...),
    roi_head=dict(...))
```

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,
```

(续下页)

(接上页)

```
                block='BASIC',
                num_blocks=(4, 4, 4, 4),
                num_channels=(32, 64, 128, 256)))),
    neck=dict(...))
```

9.6.2 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']),
```

(续下页)

(接上页)

```
            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 10

教程 2: 自定义数据集

10.1 支持新的数据格式

为了支持新的数据格式，可以选择将数据转换成现成的格式（COCO 或者 PASCAL）或将其转换成中间格式。当然也可以选择以离线的形式（在训练之前使用脚本转换）或者在线的形式（实现一个新的 dataset 在训练中进行转换）来转换数据。

在 MMDetection 中，建议将数据转换成 COCO 格式并以离线的方式进行，因此在完成数据转换后只需修改配置文件中的标注数据的路径和类别即可。

10.1.1 将新的数据格式转换为现有的数据格式

最简单的方法就是将你的数据集转换成现有的数据格式（COCO 或者 PASCAL VOC）

COCO 格式的 json 标注文件有如下必要的字段：

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

(续下页)

(接上页)

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

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

在 json 文件中有三个必要的键：

- `images`: 包含多个图片以及它们的信息的数组，例如 `file_name`、`height`、`width` 和 `id`。
- `annotations`: 包含多个实例标注信息的数组。
- `categories`: 包含多个类别名字和 ID 的数组。

在数据预处理之后，使用现有的数据格式来训练自定义的新数据集有如下两步（以 COCO 为例）：

1. 为自定义数据集修改配置文件。
2. 检查自定义数据集的标注。

这里我们举一个例子来展示上面的两个步骤，这个例子使用包括 5 个类别的 COCO 格式的数据集来训练一个现有的 Cascade Mask R-CNN R50-FPN 检测器

1. 为自定义数据集修改配置文件

配置文件的修改涉及两个方面：

1. data 部分。需要在 data.train、data.val 和 data.test 中添加 classes。
2. model 部分中的 num_classes。需要将默认值（COCO 数据集中为 80）修改为自定义数据集中的类别数。

configs/my_custom_config.py 内容如下：

```
# 新的配置来自基础的配置以更好地说明需要修改的地方
_base_ = './cascade_mask_rcnn_r50_fpn_1x_coco.py'

# 1. 数据集设定
dataset_type = 'CocoDataset'
classes = ('a', 'b', 'c', 'd', 'e')
data = dict(
    samples_per_gpu=2,
    workers_per_gpu=2,
    train=dict(
        type=dataset_type,
        # 将类别名字添加至 `classes` 字段中
        classes=classes,
        ann_file='path/to/your/train/annotation_data',
        img_prefix='path/to/your/train/image_data'),
    val=dict(
        type=dataset_type,
        # 将类别名字添加至 `classes` 字段中
        classes=classes,
        ann_file='path/to/your/val/annotation_data',
        img_prefix='path/to/your/val/image_data'),
    test=dict(
        type=dataset_type,
        # 将类别名字添加至 `classes` 字段中
        classes=classes,
        ann_file='path/to/your/test/annotation_data',
        img_prefix='path/to/your/test/image_data'))

# 2. 模型设置

# 将所有的 `num_classes` 默认值修改为5（原来为80）
model = dict(
    roi_head=dict(
        bbox_head=[
```

(续下页)

(接上页)

```
            dict(
                type='Shared2FCBBoxHead',
                # 将所有的 `num_classes` 默认值修改为 5（原来为 80）
                num_classes=5),
            dict(
                type='Shared2FCBBoxHead',
                # 将所有的 `num_classes` 默认值修改为 5（原来为 80）
                num_classes=5),
            dict(
                type='Shared2FCBBoxHead',
                # 将所有的 `num_classes` 默认值修改为 5（原来为 80）
                num_classes=5)],
    # 将所有的 `num_classes` 默认值修改为 5（原来为 80）
    mask_head=dict(num_classes=5)))
```

2. 检查自定义数据集的标注

假设你自己的数据集是 COCO 格式，那么需要保证数据的标注没有问题：

1. 标注文件中 `categories` 的长度要与配置中的 `classes` 元组长度相匹配，它们都表示有几类。（如例子中有 5 个类别）
2. 配置文件中 `classes` 字段应与标注文件里 `categories` 下的 `name` 有相同的元素且顺序一致。MMDetection 会自动将 `categories` 中不连续的 `id` 映射成连续的索引，因此 `categories` 下的 `name` 的字符串顺序会影响标签的索引。同时，配置文件中的 `classes` 的字符串顺序也会影响到预测框可视化时的标签。
3. `annotations` 中的 `category_id` 必须是有效的值。比如所有 `category_id` 的值都应该属于 `categories` 中的 `id`。

下面是一个有效标注的例子：

```
'annotations': [
    {
        'segmentation': [[192.81,
            247.09,
            ...
            219.03,
            249.06]],  #如果有 mask 标签。
        'area': 1035.749,
        'iscrowd': 0,
        'image_id': 1268,
        'bbox': [192.81, 224.8, 74.73, 33.43],
```

(续下页)

(接上页)

```
        'category_id': 16,
        'id': 42986
    },
    ...
],

# MMDetection 会自动将 `categories` 中不连续的 `id` 映射成连续的索引。
'categories': [
    {'id': 1, 'name': 'a'}, {'id': 3, 'name': 'b'}, {'id': 4, 'name': 'c'}, {'id': 16,
↪ 'name': 'd'}, {'id': 17, 'name': 'e'},
 ]
```

我们使用这种方式来支持 CityScapes 数据集。脚本在cityscapes.py 并且我们提供了微调的configs.

注意

1. 对于实例分割数据集, **MMDetection 目前只支持评估 COCO 格式的 mask AP**.
2. 推荐训练之前进行离线转换，这样就可以继续使用 `CocoDataset` 且只需修改标注文件的路径以及训练的种类。

10.1.2 调整新的数据格式为中间格式

如果不想将标注格式转换为 COCO 或者 PASCAL 格式也是可行的。实际上，我们定义了一种简单的标注格式并且与所有现有的数据格式兼容，也能进行离线或者在线转换。

数据集的标注是包含多个字典 (dict) 的列表，每个字典 (dict) 都与一张图片对应。测试时需要用到 `filename` (相对路径)、`width` 和 `height` 三个字段；训练时则额外需要 `ann`。`ann` 也是至少包含了两个字段的字典: `bboxes` 和 `labels`，它们都是 numpy array。有些数据集可能会提供如：crowd/difficult/ignored bboxes 标注，那么我们使用 `bboxes_ignore` 以及 `labels_ignore` 来包含它们。

下面给出一个例子。

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

(续下页)

(接上页)

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

有两种方法处理自定义数据。

- 在线转换（online conversion）

  可以新写一个继承自 `CustomDataset` 的 Dataset 类，并重写 `load_annotations(self, ann_file)` 以及 `get_ann_info(self, idx)` 这两个方法，正如CocoDataset与VOCDataset.

- 离线转换（offline conversion）

  可以将标注格式转换为上述的任意格式并将其保存为 pickle 或者 json 文件，例如pascal_voc.py。然后使用 `CustomDataset`。

10.1.3 自定义数据集的例子：

假设文本文件中表示的是一种全新的标注格式。边界框的标注信息保存在 `annotation.txt` 中，内容如下：

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

我们可以在 `mmdet/datasets/my_dataset.py` 中创建一个新的 dataset 用以加载数据。

```
import mmcv
import numpy as np

from .builder import DATASETS
from .custom import CustomDataset


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

(续下页)

(接上页)

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

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

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

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

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

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

        return data_infos

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

配置文件中，可以使用 `MyDataset` 进行如下修改

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

(续下页)

(接上页)

```
    pipeline=train_pipeline
)
```

10.2 使用 dataset 包装器自定义数据集

MMDetection 也支持非常多的数据集包装器（wrapper）来混合数据集或在训练时修改数据集的分布。最近 MMDetection 支持如下三种数据集包装：

- `RepeatDataset`：将整个数据集简单地重复。
- `ClassBalancedDataset`：以类别均衡的方式重复数据集。
- `ConcatDataset`：合并数据集。

10.2.1 重复数据集（Repeat dataset）

使用 `RepeatDataset` 包装器来重复数据集。例如，假设原始数据集为 `Dataset_A`，重复它过后，其配置如下：

```
dataset_A_train = dict(
        type='RepeatDataset',
        times=N,
        dataset=dict(  # Dataset_A 的原始配置信息
            type='Dataset_A',
            ...
            pipeline=train_pipeline
        )
    )
```

10.2.2 类别均衡数据集（Class balanced dataset）

使用 `ClassBalancedDataset` 作为包装器在类别的出现的频率上重复数据集。数据集需要实例化 `self.get_cat_ids(idx)` 函数以支持 `ClassBalancedDataset`。比如，以 `oversample_thr=1e-3` 来重复数据集 `Dataset_A`，其配置如下：

```
dataset_A_train = dict(
        type='ClassBalancedDataset',
        oversample_thr=1e-3,
        dataset=dict(  # Dataset_A 的原始配置信息
            type='Dataset_A',
            ...
```

(续下页)

(接上页)

```
            pipeline=train_pipeline
        )
    )
```

更多细节请参考源码。

10.2.3 合并数据集（Concatenate dataset）

合并数据集有三种方法：

1. 如果要合并的数据集类型一致但有多个的标注文件，那么可以使用如下配置将其合并。

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

   如果合并的数据集适用于测试或者评估，那么这种方式支持每个数据集分开进行评估。如果想要将合并的数据集作为整体用于评估，那么可以像如下一样设置 `separate_eval=False`。

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

2. 如果想要合并的是不同数据集，那么可以使用如下配置。

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

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

   只需设置 `separate_eval=False`，用户就可以将所有的数据集作为一个整体来评估。

注意

1. 在做评估时，`separate_eval=False` 选项是假设数据集使用了 `self.data_infos`。因此 COCO 数据集不支持此项操作，因为 COCO 数据集在做评估时并不是所有都依赖 `self.data_infos`。组合不同类型的数据集并将其作为一个整体来评估，这种做法没有得到测试，也不建议这样做。
2. 因为不支持评估 `ClassBalancedDataset` 和 `RepeatDataset`，所以也不支持评估它们的组合。

一个更复杂的例子则是分别将 `Dataset_A` 和 `Dataset_B` 重复 N 和 M 次，然后进行如下合并。

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

10.3 修改数据集的类别

根据现有数据集的类型，我们可以修改它们的类别名称来训练其标注的子集。例如，如果只想训练当前数据集中的三个类别，那么就可以修改数据集的类别元组。数据集就会自动屏蔽掉其他类别的真实框。

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

MMDetection V2.0 也支持从文件中读取类别名称，这种方式在实际应用中很常见。假设存在文件 `classes.txt`，其包含了如下的类别名称。

```
person
bicycle
car
```

用户可以将类别设置成文件路径，数据集就会自动将其加载并转换成一个列表。

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

注意

- 在 MMDetection v2.5.0 之前，如果类别为集合时数据集将自动过滤掉不包含 GT 的图片，且没办法通过修改配置将其关闭。这是一种不可取的行为而且会引起混淆，因为当类别不是集合时数据集只有在 `filter_empty_gt=True` 以及 `test_mode=False` 的情况下才会过滤掉不包含 GT 的图片。在 MMDetection v2.5.0 之后，我们将图片的过滤以及类别的修改进行解耦，如，数据集只有在 `filter_empty_gt=True` 和 `test_mode=False` 的情况下才会过滤掉不包含 GT 的图片，无论类别是否为集合。设置类别只会影响用于训练的标注类别，用户可以自行决定是否过滤不包含 GT 的图片。
- 因为中间格式只有框的标签并不包含类别的名字，所以使用 `CustomDataset` 时用户不能通过修改配置来过滤不含 GT 的图片。但是可以通过离线的方式来解决。
- 当设置数据集中的 `classes` 时，记得修改 `num_classes`。从 v2.9.0 (PR#4508) 之后，我们实现了NumClassCheckHook来检查类别数是否一致。
- 我们在未来将会重构设置数据集类别以及数据集过滤的特性，使其更加地方便用户使用。

CHAPTER 11

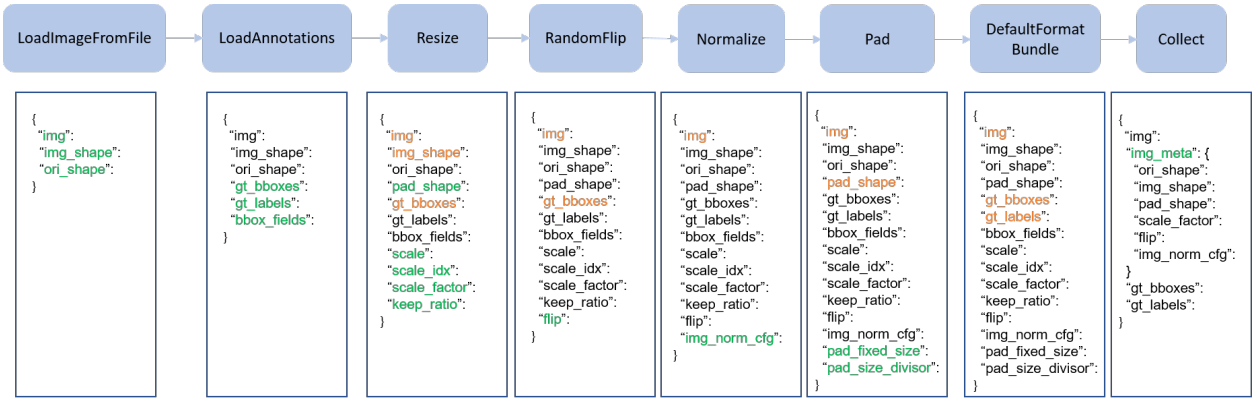
教程 3: 自定义数据预处理流程

11.1 数据流程的设计

按照惯例，我们使用 `Dataset` 和 `DataLoader` 进行多进程的数据加载。`Dataset` 返回字典类型的数据，数据内容为模型 `forward` 方法的各个参数。由于在目标检测中，输入的图像数据具有不同的大小，我们在 `MMCV` 里引入一个新的 `DataContainer` 类去收集和分发不同大小的输入数据。更多细节请参考这里。

数据的准备流程和数据集是解耦的。通常一个数据集定义了如何处理标注数据（annotations）信息，而一个数据流程定义了准备一个数据字典的所有步骤。一个流程包括一系列的操作，每个操作都把一个字典作为输入，然后再输出一个新的字典给下一个变换操作。

我们在下图展示了一个经典的数据处理流程。蓝色块是数据处理操作，随着数据流程的处理，每个操作都可以在结果字典中加入新的键（标记为绿色）或更新现有的键（标记为橙色）。



这些操作可以分为数据加载（data loading）、预处理（pre-processing）、格式变化（formatting）和测试时数据

增强（test-time augmentation）。

下面的例子是 Faster R-CNN 的一个流程：

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

对于每个操作，我们列出它添加、更新、移除的相关字典域 (dict fields)：

11.1.1 数据加载 Data loading

`LoadImageFromFile`

- 增加：img, img_shape, ori_shape

`LoadAnnotations`

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

`LoadProposals`

- 增加：proposals

11.1.2 预处理 Pre-processing

`Resize`

- 增加：scale, scale_idx, pad_shape, scale_factor, keep_ratio
- 更新：img, img_shape, *bbox_fields, *mask_fields, *seg_fields

`RandomFlip`

- 增加：flip
- 更新：img, *bbox_fields, *mask_fields, *seg_fields

`Pad`

- 增加：pad_fixed_size, pad_size_divisor
- 更新：img, pad_shape, *mask_fields, *seg_fields

`RandomCrop`

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

`Normalize`

- 增加：img_norm_cfg
- 更新：img

`SegRescale`

- 更新：gt_semantic_seg

`PhotoMetricDistortion`

- 更新：img

`Expand`

- 更新：img, gt_bboxes

`MinIoURandomCrop`

- 更新：img, gt_bboxes, gt_labels

`Corrupt`

- 更新：img

11.1.3 格式 Formatting

`ToTensor`

- 更新：由 `keys` 指定

`ImageToTensor`

- 更新：由 `keys` 指定

`Transpose`

- 更新：由 `keys` 指定

`ToDataContainer`

- 更新：由 `keys` 指定

`DefaultFormatBundle`

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

`Collect`

- 增加：img_metas（img_metas 的键（key）被 `meta_keys` 指定)
- 移除：除了 `keys` 指定的键（key）之外的所有其他的键（key）

11.1.4 测试时数据增强 Test time augmentation

`MultiScaleFlipAug`

11.2 拓展和使用自定义的流程

1. 在任意文件里写一个新的流程，例如在 `my_pipeline.py`，它以一个字典作为输入并且输出一个字典：

```
import random
from mmdet.datasets import PIPELINES


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

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

(续下页)

(接上页)

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

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

2. 在配置文件里调用并使用你写的数据处理流程，需要确保你的训练脚本能够正确导入新增模块：

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

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

3. 可视化数据增强处理流程的结果

   如果想要可视化数据增强处理流程的结果，可以使用 `tools/misc/browse_dataset.py` 直观地浏览检测数据集（图像和标注信息），或将图像保存到指定目录。使用方法请参考日志分析

CHAPTER 12

教程 4: 自定义模型

我们简单地把模型的各个组件分为五类：

- 主干网络 (backbone)：通常是一个用来提取特征图 (feature map) 的全卷积网络 (FCN network)，例如：ResNet, MobileNet。
- Neck：主干网络和 Head 之间的连接部分，例如：FPN, PAFPN。
- Head：用于具体任务的组件，例如：边界框预测和掩码预测。
- 区域提取器 (roi extractor)：从特征图中提取 RoI 特征，例如：RoI Align。
- 损失 (loss)：在 Head 组件中用于计算损失的部分，例如：FocalLoss, L1Loss, GHMLoss.

12.1 开发新的组件

12.1.1 添加一个新的主干网络

这里，我们以 MobileNet 为例来展示如何开发新组件。

1. 定义一个新的主干网络（以 MobileNet 为例）

新建一个文件 mmdet/models/backbones/mobilenet.py

```
import torch.nn as nn

from ..builder import BACKBONES


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

    def __init__(self, arg1, arg2):
        pass

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

2. 导入该模块

你可以添加下述代码到 mmdet/models/backbones/__init__.py

```
from .mobilenet import MobileNet
```

或添加：

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

到配置文件以避免原始代码被修改。

3. 在你的配置文件中使用该主干网络

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

12.1.2 添加新的 Neck

1. 定义一个 Neck（以 PAFPN 为例）

新建一个文件 mmdet/models/necks/pafpn.py

```
from ..builder import NECKS

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

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

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

2. 导入该模块

你可以添加下述代码到 mmdet/models/necks/__init__.py

```
from .pafpn import PAFPN
```

或添加：

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

到配置文件以避免原始代码被修改。

3. 修改配置文件

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

12.1.3 添加新的 Head

我们以 Double Head R-CNN 为例来展示如何添加一个新的 Head。

首先，添加一个新的 bbox head 到 `mmdet/models/roi_heads/bbox_heads/double_bbox_head.py`。Double Head R-CNN 在目标检测上实现了一个新的 bbox head。为了实现 bbox head，我们需要使用如下的新模块中三个函数。

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

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

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

    def __init__(self,
                 num_convs=0,
                 num_fcs=0,
                 conv_out_channels=1024,
                 fc_out_channels=1024,
                 conv_cfg=None,
                 norm_cfg=dict(type='BN'),
                 **kwargs):
        kwargs.setdefault('with_avg_pool', True)
        super(DoubleConvFCBBoxHead, self).__init__(**kwargs)
```

(续下页)

(接上页)

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

然后，如有必要，实现一个新的 bbox head。我们打算从 StandardRoIHead 来继承新的 DoubleHeadRoIHead。我们可以发现 StandardRoIHead 已经实现了下述函数。

```
import torch

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


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

    def init_assigner_sampler(self):

    def init_bbox_head(self, bbox_roi_extractor, bbox_head):

    def init_mask_head(self, mask_roi_extractor, mask_head):


    def forward_dummy(self, x, proposals):


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

    def _bbox_forward(self, x, rois):

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

(续下页)

(接上页)

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

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


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

Double Head 的修改主要在 bbox_forward 的逻辑中，且它从 `StandardRoIHead` 中继承了其他逻辑。在 `mmdet/models/roi_heads/double_roi_head.py` 中，我们用下述代码实现新的 bbox head：

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


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

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

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

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

(续下页)

(接上页)

```
        cls_score, bbox_pred = self.bbox_head(bbox_cls_feats, bbox_reg_feats)

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

最终，用户需要把该模块添加到 mmdet/models/bbox_heads/__init__.py 和 mmdet/models/roi_heads/__init__.py 以使相关的注册表可以找到并加载他们。

或者，用户可以添加：

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

到配置文件并实现相同的目的。

Double Head R-CNN 的配置文件如下：

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

从 MMDetection 2.0 版本起，配置系统支持继承配置以使用户可以专注于修改。Double Head R-CNN 主要使用了一个新的 DoubleHeadRoIHead 和一个新的 `DoubleConvFCBBoxHead`，参数需要根据每个模块的 `__init__` 函数来设置。

12.1.4 添加新的损失

假设你想添加一个新的损失 `MyLoss` 用于边界框回归。为了添加一个新的损失函数，用户需要在 `mmdet/models/losses/my_loss.py` 中实现。装饰器 `weighted_loss` 可以使损失每个部分加权。

```
import torch
import torch.nn as nn

from ..builder import LOSSES
from .utils import weighted_loss

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

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

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

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

然后，用户需要把它加到 `mmdet/models/losses/__init__.py`。

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

或者，你可以添加：

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

到配置文件来实现相同的目的。

如使用，请修改 `loss_xxx` 字段。因为 MyLoss 是用于回归的，你需要在 Head 中修改 `loss_xxx` 字段。

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

CHAPTER 13

教程 5: 自定义训练配置

CHAPTER 14

教程 6: 自定义损失函数

MMDetection 为用户提供了不同的损失函数。但是默认的配置可能无法适应不同的数据和模型，所以用户可能会希望修改某一个损失函数来适应新的情况。

本教程首先详细的解释计算损失的过程然后给出一些关于如何修改每一个步骤的指导。对损失的修改可以被分为微调和加权。

14.1 一个损失的计算过程

给定输入（包括预测和目标，以及权重），损失函数会把输入的张量映射到最后的损失标量。映射过程可以分为下面五个步骤：

1. 设置采样方法为对正负样本进行采样。
2. 通过损失核函数获取**元素**或者**样本**损失。
3. 通过权重张量来给损失**逐元素**权重。
4. 把损失张量归纳为一个**标量**。
5. 用一个**张量**给当前损失一个权重。

14.2 设置采样方法（步骤 1）

对于一些损失函数，需要采样策略来避免正负样本之间的不平衡。

例如，在 RPN head 中使用 `CrossEntropyLoss` 时，我们需要在 `train_cfg` 中设置 `RandomSampler`

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

对于其他一些具有正负样本平衡机制的损失，例如 Focal Loss、GHMC 和 QualityFocalLoss，不再需要进行采样。

14.3 微调损失

微调一个损失主要与步骤 2，4，5 有关，大部分的修改可以在配置文件中指定。这里我们用 Focal Loss (FL) 作为例子。下面的代码分别是构建 FL 的方法和它的配置文件，他们是一一对应的。

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

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

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

14.3.1 微调超参数（步骤 2）

gamma 和 beta 是 Focal Loss 中的两个超参数。如果我们想把 gamma 的值设为 1.5，把 alpha 的值设为 0.5，我们可以在配置文件中按照如下指定：

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

14.3.2 微调归纳方式（步骤 4）

Focal Loss 默认的归纳方式是 mean。如果我们想把归纳方式从 mean 改成 sum，我们可以在配置文件中按照如下指定：

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

14.3.3 微调损失权重（步骤 5）

这里的损失权重是一个标量，他用来控制多任务学习中不同损失的重要程度，例如，分类损失和回归损失。如果我们想把分类损失的权重设为 0.5，我们可以在配置文件中如下指定：

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

14.4 加权损失（步骤 3）

加权损失就是我们逐元素修改损失权重。更具体来说，我们给损失张量乘以一个与他有相同形状的权重张量。所以，损失中不同的元素可以被赋予不同的比例，所以这里叫做逐元素。损失的权重在不同模型中变化很大，而且与上下文相关，但是总的来说主要有两种损失权重：分类损失的 `label_weights` 和边界框的 `bbox_weights`。你可以在相应的头中的 `get_target` 方法中找到他们。这里我们使用 ATSSHead 作为一个例子。它继承了 AnchorHead，但是我们重写它的 `get_targets` 方法来产生不同的 `label_weights` 和 `bbox_weights`。

```
class ATSSHead(AnchorHead):

    ...

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

CHAPTER 15

教程 7: 模型微调

在 COCO 数据集上预训练的检测器可以作为其他数据集（例如 CityScapes 和 KITTI 数据集）优质的预训练模型。本教程将指导用户如何把*ModelZoo* 中提供的模型用于其他数据集中并使得当前所训练的模型获得更好性能。

以下是在新数据集中微调模型需要的两个步骤。

- 按*教程 2：自定义数据集的方法* 中的方法对新数据集添加支持中的方法对新数据集添加支持
- 按照本教程中所讨论方法，修改配置信息

接下来将会以 Cityscapes Dataset 上的微调过程作为例子，具体讲述用户需要在配置中修改的五个部分。

15.1 继承基础配置

为了减轻编写整个配置的负担并减少漏洞的数量，MMDetection V2.0 支持从多个现有配置中继承配置信息。微调 MaskRCNN 模型的时候，新的配置信息需要使用从 `_base_/models/mask_rcnn_r50_fpn.py` 中继承的配置信息来构建模型的基本结构。当使用 Cityscapes 数据集时，新的配置信息可以简便地从 `_base_/datasets/cityscapes_instance.py` 中继承。对于训练过程的运行设置部分，新配置需要从 `_base_/default_runtime.py` 中继承。这些配置文件 `configs` 的目录下，用户可以选择全部内容的重新编写而不是使用继承方法。

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

15.2 Head 的修改

接下来新的配置还需要根据新数据集的类别数量对 Head 进行修改。只需要对 roi_head 中的 num_classes 进行修改。修改后除了最后的预测模型的 Head 之外，预训练模型的权重的大部分都会被重新使用。

```
model = dict(
    pretrained=None,
    roi_head=dict(
        bbox_head=dict(
            type='Shared2FCBBoxHead',
            in_channels=256,
            fc_out_channels=1024,
            roi_feat_size=7,
            num_classes=8,
            bbox_coder=dict(
                type='DeltaXYWHBBoxCoder',
                target_means=[0., 0., 0., 0.],
                target_stds=[0.1, 0.1, 0.2, 0.2]),
            reg_class_agnostic=False,
            loss_cls=dict(
                type='CrossEntropyLoss', use_sigmoid=False, loss_weight=1.0),
            loss_bbox=dict(type='SmoothL1Loss', beta=1.0, loss_weight=1.0)),
        mask_head=dict(
            type='FCNMaskHead',
            num_convs=4,
            in_channels=256,
            conv_out_channels=256,
            num_classes=8,
            loss_mask=dict(
                type='CrossEntropyLoss', use_mask=True, loss_weight=1.0))))
```

15.3 数据集的修改

用户可能还需要准备数据集并编写有关数据集的配置。目前 MMDetection V2.0 的配置文件已经支持 VOC、WIDER FACE、COCO 和 Cityscapes Dataset 的数据集信息。

15.4 训练策略的修改

微调超参数与默认的训练策略不同。它通常需要更小的学习率和更少的训练回合。

```
# 优化器
# batch size 为 8 时的 lr 配置
optimizer = dict(type='SGD', lr=0.01, momentum=0.9, weight_decay=0.0001)
optimizer_config = dict(grad_clip=None)
# 学习策略
lr_config = dict(
    policy='step',
    warmup='linear',
    warmup_iters=500,
    warmup_ratio=0.001,
    step=[7])
# lr_config 中的 max_epochs 和 step 需要针对自定义数据集进行专门调整
runner = dict(max_epochs=8)
log_config = dict(interval=100)
```

15.5 使用预训练模型

如果要使用预训练模型时，可以在 load_from 中查阅新的配置信息，用户需要在训练开始之前下载好需要的模型权重，从而避免在训练过程中浪费了宝贵时间。

```
load_from = 'https://download.openmmlab.com/mmdetection/v2.0/mask_rcnn/mask_rcnn_r50_
↪caffe_fpn_mstrain-poly_3x_coco/mask_rcnn_r50_caffe_fpn_mstrain-poly_3x_coco_bbox_
↪mAP-0.408__segm_mAP-0.37_20200504_163245-42aa3d00.pth'  # noqa
```

CHAPTER 16

教程 8: Pytorch 到 ONNX 的模型转换（实验性支持）

16.1 尝试使用新的 MMDeploy 來部署你的模型

CHAPTER 17

教程 9: ONNX 到 TensorRT 的模型转换（实验性支持）

17.1 尝试使用新的 MMDeploy 来部署你的模型

17.2 如何将模型从 ONNX 转换为 TensorRT

17.2.1 先决条件

1. 请参考 get_started.md 从源码安装 MMCV 和 MMDetection。
2. 请参考 ONNXRuntime in mmcv 和 TensorRT plugin in mmcv 安装支持 ONNXRuntime 自定义操作和 TensorRT 插件的 `mmcv-full`。

3. 使用工具 pytorch2onnx 将模型从 PyTorch 转换为 ONNX。

17.2.2 用法

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

所有参数的说明:

- `config`: 模型配置文件的路径。
- `model`: ONNX 模型文件的路径。
- `--trt-file`: 输出 TensorRT 引擎文件的路径。如果未指定，它将被设置为 `tmp.trt`。
- `--input-img`: 用于追踪和转换的输入图像的路径。默认情况下，它将设置为 `demo/demo.jpg`。
- `--shape`: 模型输入的高度和宽度。如果未指定，它将设置为 `400 600`。
- `--min-shape`: 模型输入的最小高度和宽度。如果未指定，它将被设置为与 `--shape` 相同。
- `--max-shape`: 模型输入的最大高度和宽度。如果未指定，它将被设置为与 `--shape` 相同。
- `--workspace-size`: 构建 TensorRT 引擎所需的 GPU 工作空间大小（以 GiB 为单位）。如果未指定，它将设置为 `1` GiB。
- `--show`: 确定是否显示模型的输出。如果未指定，它将设置为 `False`。
- `--verify`: 确定是否在 ONNXRuntime 和 TensorRT 之间验证模型的正确性。如果未指定，它将设置为 `False`。
- `--verbose`: 确定是否打印日志消息。它对调试很有用。如果未指定，它将设置为 `False`。

例子:

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

(续下页)

(接上页)

```
    --shape 400 600 \
    --show \
    --verify \
```

17.3 如何评估导出的模型

我们准备了一个工具 `tools/deplopyment/test.py` 来评估 TensorRT 模型。

请参阅以下链接以获取更多信息。

- 如何评估导出的模型
- 结果和模型

17.4 支持转换为 TensorRT 的模型列表

下表列出了确定可转换为 TensorRT 的模型。

注意:

- *以上所有模型通过 Pytorch==1.6.0, onnx==1.7.0 与 TensorRT-7.2.1.6.Ubuntu-16.04.x86_64-gnu.cuda-10.2.cudnn8.0 测试*

17.5 提醒

- 如果您在上面列出的模型中遇到任何问题，请创建 issue，我们会尽快处理。对于未包含在列表中的模型，由于资源有限，我们可能无法在此提供太多帮助。请尝试深入挖掘并自行调试。
- 由于此功能是实验性的，并且可能会快速更改，因此请始终尝试使用最新的 `mmcv` 和 `mmdetecion`。

17.6 常见问题

- 空

CHAPTER 18

教程 10: 权重初始化

在训练过程中，适当的初始化策略有利于加快训练速度或获得更高的性能。MMCV 提供了一些常用的初始化模块的方法，如 `nn.Conv2d`。MMdetection 中的模型初始化主要使用 `init_cfg`。用🅴可以通过以下两个步骤来初始化模型：

1. 在 `model_cfg` 中为模型或其组件定义 `init_cfg`，但子组件的 `init_cfg` 优先级更高，会覆盖父模块的 `init_cfg` 。
2. 像往常一样构建模型，然后显式调用 `model.init_weights()` 方法，此时模型参数将会被按照配置文件写法进行初始化。

MMdetection 初始化工作流的高层 API 调用流程是：

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

18.1 描述

它的数据类型是 dict 或者 list[dict]，包含了下列键值:

- `type` (str)，包含 `INTIALIZERS` 中的初始化器名称，后面跟着初始化器的参数。
- `layer`（str 或 list[str]），包含 Pytorch 或 MMCV 中基本层的名称，以及将被初始化的可学习参数，例如 `'Conv2d'`, `'DeformConv2d'`。
- `override` (dict 或 list[dict])，包含不继承自 `BaseModule` 且其初始化配置与 `layer` 键中的其他层不同的子模块。`type` 中定义的初始化器将适用于 `layer` 中定义的所有层，因此如果子模块不

是 BaseModule 的派生类但可以与 layer 中的层相同的方式初始化，则不需要使用 override。override 包含了：

- type 后跟初始化器的参数；
- name 用以指示将被初始化的子模块。

18.2 初始化参数

从 mmcv.runner.BaseModule 或 mmdet.models 继承一个新模型。这里我们用 FooModel 来举个例子。

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

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

- 直接在代码中使用 init_cfg 初始化模型

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

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

- 在 mmcv.Sequential 或 mmcv.ModuleList 代码中直接使用 init_cfg 初始化模型

```
from mmcv.runner import BaseModule, ModuleList

class FooModel(BaseModule)
        def __init__(self,
                        arg1,
                        arg2,
                        init_cfg=None):
```

(续下页)

(接上页)

```
                super(FooModel, self).__init__(init_cfg)
            ...
            self.conv1 = ModuleList(init_cfg=XXX)
```

- 使用配置文件中的 init_cfg 初始化模型

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

18.3 init_cfg 的使用

1. 用 layer 键初始化模型

   如果我们只定义了 layer, 它只会在 layer 键中初始化网络层。

   注意: layer 键对应的值是 Pytorch 的带有 weights 和 bias 属性的类名（因此不支持 MultiheadAttention 层）。

- 定义用于初始化具有相同配置的模块的 layer 键。

```
init_cfg = dict(type='Constant', layer=['Conv1d', 'Conv2d', 'Linear'], val=1)
# 用相同的配置初始化整个模块
```

- 定义用于初始化具有不同配置的层的 layer 键。

```
init_cfg = [dict(type='Constant', layer='Conv1d', val=1),
            dict(type='Constant', layer='Conv2d', val=2),
            dict(type='Constant', layer='Linear', val=3)]
# nn.Conv1d 将被初始化为 dict(type='Constant', val=1)
# nn.Conv2d 将被初始化为 dict(type='Constant', val=2)
# nn.Linear 将被初始化为 dict(type='Constant', val=3)
```

2. 使用 override 键初始化模型

- 当使用属性名初始化某些特定部分时，我们可以使用 override 键，override 中的值将忽略 init_cfg 中的值。

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

init_cfg = dict(type='Constant',
                layer=['Conv1d','Conv2d'], val=1, bias=2,
                override=dict(type='Constant', name='reg', val=3, bias=4))
# self.feat and self.cls 将被初始化为 dict(type='Constant', val=1, bias=2)
# 叫 'reg' 的模块将被初始化为 dict(type='Constant', val=3, bias=4)
```

- 如果 init_cfg 中的 `layer` 为 None，则只会初始化 override 中有 name 的子模块，而 override 中的 type 和其他参数可以省略。

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

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

# self.feat and self.cls 将被 Pytorch 初始化
# 叫 'reg' 的模块将被 dict(type='Constant', val=1, bias=2) 初始化
```

- 如果我们不定义 `layer` 或 `override` 键，它不会初始化任何东西。
- 无效的使用

```
# override 没有 name 键的话是无效的
init_cfg = dict(type='Constant', layer=['Conv1d','Conv2d'], val=1, bias=2,
                    override=dict(type='Constant', val=3, bias=4))

# override 有 name 键和其他参数但是没有 type 键也是无效的
init_cfg = dict(type='Constant', layer=['Conv1d','Conv2d'], val=1, bias=2,
                override=dict(name='reg', val=3, bias=4))
```

3. 使用预训练模型初始化模型

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

更多细节可以参考 MMCV 的文档和 MMCV PR #780

CHAPTER 19

教程 11: How to xxx

本教程收集了任何如何使用 MMDetection 进行 xxx 的答案。如果您遇到有关如何做的问题及答案，请随时更新此文档!

19.1 使用 MMClassification 的骨干网络

MMDet、MMCls、MMSeg 中的模型注册表都继承自 MMCV 中的根注册表，允许这些存储库直接使用彼此已经实现的模块。因此用户可以在 MMDetection 中使用来自 MMClassification 的骨干网络，而无需实现 MMClassification 中已经存在的网络。

19.1.1 使用在 MMClassification 中实现的骨干网络

假设想将 `MobileNetV3-small` 作为 `RetinaNet` 的骨干网络，则配置文件如下。

```
_base_ = [
    '../_base_/models/retinanet_r50_fpn.py',
    '../_base_/datasets/coco_detection.py',
    '../_base_/schedules/schedule_1x.py', '../_base_/default_runtime.py'
]
# please install mmcls>=0.20.0
# import mmcls.models to trigger register_module in mmcls
custom_imports = dict(imports=['mmcls.models'], allow_failed_imports=False)
pretrained = 'https://download.openmmlab.com/mmclassification/v0/mobilenet_v3/convert/
```

(续下页)

(接上页)

```
↪mobilenet_v3_small-8427ecf0.pth'
model = dict(
    backbone=dict(
        _delete_=True, # 将 _base_ 中关于 backbone 的字段删除
        type='mmcls.MobileNetV3', # 使用 mmcls 中的 MobileNetV3
        arch='small',
        out_indices=(3, 8, 11), # 修改 out_indices
        init_cfg=dict(
            type='Pretrained',
            checkpoint=pretrained,
            prefix='backbone.')), # MMCls 中骨干网络的预训练权重含义 prefix='backbone.
↪', 为了正常加载权重，需要把这个 prefix 去掉。
    # 修改 in_channels
    neck=dict(in_channels=[24, 48, 96], start_level=0))
```

19.1.2 通过 MMClassification 使用 TIMM 中实现的骨干网络

由于 MMClassification 提供了 PyTorch Image Models (timm) 骨干网络的封装，用户也可以通过 MMClassification 直接使用 timm 中的骨干网络。假设想将 EfficientNet-B1 作为 RetinaNet 的骨干网络，则配置文件如下。

```
# https://github.com/open-mmlab/mmdetection/blob/master/configs/timm_example/
↪retinanet_timm_efficientnet_b1_fpn_1x_coco.py
_base_ = [
    '../_base_/models/retinanet_r50_fpn.py',
    '../_base_/datasets/coco_detection.py',
    '../_base_/schedules/schedule_1x.py', '../_base_/default_runtime.py'
]

# please install mmcls>=0.20.0
# import mmcls.models to trigger register_module in mmcls
custom_imports = dict(imports=['mmcls.models'], allow_failed_imports=False)
model = dict(
    backbone=dict(
        _delete_=True, # 将 _base_ 中关于 backbone 的字段删除
        type='mmcls.TIMMBackbone', # 使用 mmcls 中 timm 骨干网络
        model_name='efficientnet_b1',
        features_only=True,
        pretrained=True,
        out_indices=(1, 2, 3, 4)), # 修改 out_indices
    neck=dict(in_channels=[24, 40, 112, 320])) # 修改 in_channels
```

(续下页)

(接上页)

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

type='mmcls.TIMMBackbone' 表示在 MMDetection 中使用 MMClassification 中的 TIMMBackbone 类，并且使用的模型为 EfficientNet-B1，其中 mmcls 表示 MMClassification 库，而 TIMMBackbone 表示 MMClassification 中实现的 TIMMBackbone 包装器。

关于层次注册器的具体原理可以参考 MMCV 文档，关于如何使用 MMClassification 中的其他 backbone，可以参考 MMClassification 文档。

19.2 使用马赛克数据增强

如果你想在训练中使用 Mosaic，那么请确保你同时使用 MultiImageMixDataset。以 Faster R-CNN 算法为例，你可以通过如下做法实现：

```
# 直接打开 configs/faster_rcnn/faster_rcnn_r50_fpn_1x_coco.py ,增添如下字段
data_root = 'data/coco/'
dataset_type = 'CocoDataset'
img_scale=(1333, 800)
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='Mosaic', img_scale=img_scale, pad_val=114.0),
    dict(
        type='RandomAffine',
        scaling_ratio_range=(0.1, 2),
        border=(-img_scale[0] // 2, -img_scale[1] // 2)), #␣
↪图像经过马赛克处理后会放大4倍，所以我们使用仿射变换来恢复图像的大小。
    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'])
]

train_dataset = dict(
    _delete_ = True, # 删除不必要的设置
    type='MultiImageMixDataset',
    dataset=dict(
        type=dataset_type,
        ann_file=data_root + 'annotations/instances_train2017.json',
        img_prefix=data_root + 'train2017/',
```

(续下页)

(接上页)

```
        pipeline=[
            dict(type='LoadImageFromFile'),
            dict(type='LoadAnnotations', with_bbox=True)
        ],
        filter_empty_gt=False,
    ),
    pipeline=train_pipeline
    )

data = dict(
    train=train_dataset
    )
```

19.3 在配置文件中冻结骨干网络后在训练中解冻骨干网络

如果你在配置文件中已经冻结了骨干网络并希望在几个训练周期后解冻它，你可以通过 hook 来实现这个功能。以用 ResNet 为骨干网络的 Faster R-CNN 为例，你可以冻结一个骨干网络的一个层并在配置文件中添加如下 `custom_hooks`:

```
_base_ = [
    '../_base_/models/faster_rcnn_r50_fpn.py',
    '../_base_/datasets/coco_detection.py',
    '../_base_/schedules/schedule_1x.py', '../_base_/default_runtime.py'
]
model = dict(
    # freeze one stage of the backbone network.
    backbone=dict(frozen_stages=1),
)
custom_hooks = [dict(type="UnfreezeBackboneEpochBasedHook", unfreeze_epoch=1)]
```

同时在 `mmdet/core/hook/unfreeze_backbone_epoch_based_hook.py` 当中书写 `UnfreezeBackboneEpochBasedHook` 类

```
from mmcv.parallel import is_module_wrapper
from mmcv.runner.hooks import HOOKS, Hook


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

    Args:
```

(续下页)

(接上页)

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

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

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

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

19.4 获得新的骨干网络的通道数

如果你想获得一个新骨干网络的通道数，你可以单独构建这个骨干网络并输入一个伪造的图片来获取每一个阶段的输出。

以 ResNet 为例：

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

(续下页)

(接上页)

```
inputs = torch.rand(1, 3, 32, 32)
level_outputs = self.forward(inputs)
for level_out in level_outputs:
    print(tuple(level_out.shape))
```

以上脚本的输出为:

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

用户可以通过将脚本中的 `ResNet(depth=18)` 替换为自己的骨干网络配置来得到新的骨干网络的通道数。

CHAPTER 20

日志分析

CHAPTER 21

默认约定

如果你想把 MMDetection 修改为自己的项目，请遵循下面的约定。

21.1 损失

在 MMDetection 中，`model(**data)` 的返回值是一个字典，包含着所有的损失和评价指标，他们将会由 `model(**data)` 返回。

例如，在 bbox head 中，

```
class BBoxHead(nn.Module):
    ...
    def loss(self, ...):
        losses = dict()
        # 分类损失
        losses['loss_cls'] = self.loss_cls(...)
        # 分类准确率
        losses['acc'] = accuracy(...)
        # 边界框损失
        losses['loss_bbox'] = self.loss_bbox(...)
        return losses
```

`'bbox_head.loss()'` 在模型 forward 阶段会被调用。返回的字典中包含了 `'loss_bbox','loss_cls','acc'`。只有 `'loss_bbox'`, `'loss_cls'` 会被用于反向传播，`'acc'` 只会被作为评价指标来监控训练过程。

我们默认，只有那些键的名称中包含 'loss' 的值会被用于反向传播。这个行为可以通过修改 BaseDetector.train_step() 来改变。

21.2 空 proposals

在 MMDetection 中，我们为两阶段方法中空 proposals 的情况增加了特殊处理和单元测试。我们同时需要处理整个 batch 和单一图片中空 proposals 的情况。例如，在 CascadeRoIHead 中，

```
# 简单的测试
...

# 在整个 batch中 都没有 proposals
if rois.shape[0] == 0:
    bbox_results = [[
        np.zeros((0, 5), dtype=np.float32)
        for _ in range(self.bbox_head[-1].num_classes)
    ]] * num_imgs
    if self.with_mask:
        mask_classes = self.mask_head[-1].num_classes
        segm_results = [[[] for _ in range(mask_classes)]
                        for _ in range(num_imgs)]
        results = list(zip(bbox_results, segm_results))
    else:
        results = bbox_results
    return results
...

# 在单张图片中没有 proposals
for i in range(self.num_stages):
    ...
    if i < self.num_stages - 1:
          for j in range(num_imgs):
                    # 处理空 proposals
                    if rois[j].shape[0] > 0:
                        bbox_label = cls_score[j][:, :-1].argmax(dim=1)
                        refine_roi = self.bbox_head[i].regress_by_class(
                             rois[j], bbox_label[j], bbox_pred[j], img_metas[j])
                        refine_roi_list.append(refine_roi)
```

如果你有自定义的 RoIHead, 你可以参考上面的方法来处理空 proposals 的情况。

21.3 全景分割数据集

在 MMDetection 中，我们支持了 COCO 全景分割数据集 `CocoPanopticDataset`。对于它的实现，我们在这里声明一些默认约定。

1. 在 mmdet<=2.16.0 时，语义分割标注中的前景和背景标签范围与 MMDetection 中的默认规定有所不同。标签 `0` 代表 `VOID` 标签。从 mmdet=2.17.0 开始，为了和框的类别标注保持一致，语义分割标注的类别标签也改为从 `0` 开始，标签 `255` 代表 `VOID` 类。为了达成这一目标，我们在流程 `Pad` 里支持了设置 `seg` 的填充值的功能。
2. 在评估中，全景分割结果必须是一个与原图大小相同的图。结果图中每个像素的值有如此形式: `instance_id * INSTANCE_OFFSET + category_id`。

CHAPTER 22

MMDetection v2.x 兼容性说明

22.1 MMDetection 2.25.0

为了加入 Mask2Former 实例分割模型，对 Mask2Former 的配置文件进行了重命名 PR #7571：

```
'mask2former_xxx_coco.py' 代表全景分割的配置文件
```

```
'mask2former_xxx_coco.py' 代表实例分割的配置文件
'mask2former_xxx_coco-panoptic.py' 代表全景分割的配置文件
```

22.2 MMDetection 2.21.0

为了支持 CPU 训练，MMCV 中进行批处理的 scatter 的代码逻辑已经被修改。我们推荐使用 MMCV v1.4.4 或更高版本，更多信息请参考 MMCV PR #1621.

22.3 MMDetection 2.18.1

22.3.1 MMCV compatibility

为了修复 BaseTransformerLayer 中的权重引用问题, MultiheadAttention 中 batch first 的逻辑有所改变。我们推荐使用 MMCV v1.3.17 或更高版本。更多信息请参考 MMCV PR #1418 。

22.4 MMDetection 2.18.0

22.4.1 DIIHead 兼容性

为了支持 QueryInst，在 DIIHead 的返回元组中加入了 attn_feats。

22.5 MMDetection v2.14.0

22.5.1 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 。

22.5.2 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。

22.6 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。

22.6.1 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。

22.6.2 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。

22.6.3 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 。

22.6.4 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。

22.7 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。

22.7.1 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%。

22.7.2 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。

22.7.3 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。

22.7.4 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。

22.8 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 23

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。

23.1 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。

23.2 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 extension 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 extension 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`。

23.3 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。
4. 你也可以尝试使用 `AvoidCUDAOOM` 来避免该问题。首先它将尝试调用 `torch.cuda.empty_cache()`。如果失败，将会尝试把输入类型转换到 FP16。如果仍然失败，将会把输入从 GPUs 转换到 CPUs 进行计算。这里提供了两个使用的例子：

```
from mmdet.utils import AvoidCUDAOOM

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

你也可也使用 `AvoidCUDAOOM` 作为装饰器让代码遇到 OOM 的时候继续运行：

```
from mmdet.utils import AvoidCUDAOOM

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

- “RuntimeError: Expected to have finished reduction in the prior iteration before starting a new one”
  1. 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` 进行训练 (会降低训练速度)。
  3. 你也可以通过在 config 中的 `optimizer_config` 里设置 `detect_anomalous_params=True` 查找哪些参数没有用到，但是需要 MMCV 的版本 >= 1.4.1。
- 训练中保存最好模型

  可以通过配置 `evaluation = dict(save_best='auto')` 开启。在 auto 参数情况下会根据返回的验证结果中的第一个 key 作为选择最优模型的依据，你也可以直接设置评估结果中的 key 来手动设置，例如 `evaluation = dict(save_best='mAP')`。
- 在 Resume 训练中使用 `ExpMomentumEMAHook`

  如果在训练中使用了 `ExpMomentumEMAHook`，那么 resume 时候不能仅仅通过命令行参数 `--resume-from` 或 `--cfg-options resume_from` 实现恢复模型参数功能例如 `python tools/train.py configs/yolox/yolox_s_8x8_300e_coco.py --resume-from ./work_dir/yolox_s_8x8_300e_coco/epoch_x.pth`。以 `yolox_s` 算法为例，由于 `ExpMomentumEMAHook` 需要重新加载权重，你可以通过如下做法实现：

```
# 直接打开 configs/yolox/yolox_s_8x8_300e_coco.py 修改所有 resume_from 字段
resume_from=./work_dir/yolox_s_8x8_300e_coco/epoch_x.pth
custom_hooks=[...
    dict(
        type='ExpMomentumEMAHook',
        resume_from=./work_dir/yolox_s_8x8_300e_coco/epoch_x.pth,
        momentum=0.0001,
        priority=49)
    ]
```

23.4 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.

23.5 Model 相关

- **ResNet style 参数说明**

  ResNet style 可选参数允许 `pytorch` 和 `caffe`，其差别在于 Bottleneck 模块。Bottleneck 是 `1x1-3x3-1x1` 堆叠结构，在 `caffe` 模式模式下 stride=2 参数放置在第一个 `1x1` 卷积处，而 `pyorch` 模式下 stride=2 放在第二个 `3x3` 卷积处。一个简单示例如下：

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

- **ResNeXt 参数说明**

  ResNeXt 来自论文 `Aggregated Residual Transformations for Deep Neural Networks`. 其引入分组卷积，并且通过变量基数来控制组的数量达到精度和复杂度的平衡，其有两个超参 `baseWidth` 和 `cardinality` 来控制内部 Bottleneck 模块的基本宽度和分组数参数。以 MMDetection 中配置名为 `mask_rcnn_x101_64x4d_fpn_mstrain-poly_3x_coco.py` 为例，其中 `mask_rcnn` 代表算法采用 Mask R-CNN，`x101` 代表骨架网络采用 ResNeXt-101，`64x4d` 代表 Bottleneck 一共分成 64 组，每组的基本宽度是 4。

- **骨架网络 eval 模式说明**

  因为检测模型通常比较大且输入图片分辨率很高，这会导致检测模型的 batch 很小，通常是 2，这会使得 BatchNorm 在训练过程计算的统计量方差非常大，不如主干网络预训练时得到的统计量稳定，因此在训练是一般都会使用 `norm_eval=True` 模式，直接使用预训练主干网络中的 BatchNorm 统计量，少数使用大 batch 的算法是 `norm_eval=False` 模式，例如 NASFPN。对于没有 ImageNet 预训练的骨架网络，如果 batch 比较小，可以考虑使用 `SyncBN`。

CHAPTER 24

English

CHAPTER 25

简体中文

CHAPTER 26

mmdet.apis

CHAPTER 27

mmdet.core

27.1 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.

示例

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

gen_base_anchors()

Generate base anchors.

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*, *dtype=torch.float32*, *device='cuda'*)

Generate grid anchors in multiple feature levels.

参数

- **featmap_sizes** (*list[tuple]*) –List of feature map sizes in multiple feature levels.
- **dtype** (`torch.dtype`) –Dtype of priors. Default: torch.float32.
- **device** (*str*) –The device where the anchors will be put on.

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.

备注： 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*, *dtype=torch.float32*, *device='cuda'*)

Generate grid anchors of a single level.

备注： 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.

- **(obj** (*dtype*) –*torch.dtype*): Date type of points.Defaults to `torch.float32`.
- **device** (*str, optional*) –The device the tensor will be put on. Defaults to 'cuda'.

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.

备注： 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 proportion to anchors' width and height. By default it is 0.5 in V2.0 but it should be 0.5 in v1.x models.

示例

```
>>> 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.],
        [ 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.

备注： 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*, *dtype=torch.float32*, *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).
- **dtype** (dtype) –Dtype of priors. Default: torch.float32.
- **device** (*str*) –The device where the anchors will be put on.
- **with_stride** (*bool*) –Whether to concatenate the stride to the last dimension of points.

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*, *dtype=torch.float32*, *device='cuda'*, *with_stride=False*)

Generate grid Points of a single level.

备注：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.
- **dtype** (dtype) –Dtype of priors. Default: torch.float32.
- **device** (*str, optional*) –The device the tensor will be put on. Defaults to 'cuda'.
- **with_stride** (*bool*) –Concatenate the stride to the last dimension of points.

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, ···]

27.2 bbox

27.3 export

27.4 mask

27.5 evaluation

27.6 post_processing

27.7 utils

CHAPTER 28

mmdet.datasets

28.1 datasets

28.2 pipelines

28.3 samplers

28.4 api_wrappers

CHAPTER 29

mmdet.models

29.1 detectors

29.2 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*) –Architecture of CSP-Darknet, from {P5, P6}. Default: P5.
- **deepen_factor** (*float*) –Depth multiplier, multiply number of blocks in CSP layer by this amount. Default: 1.0.
- **widen_factor** (*float*) –Width multiplier, multiply number of channels in each 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.

备注： 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.

备注: 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.**EfficientNet**(*arch='b0'*, *drop_path_rate=0.0*, *out_indices=(6,)*, *frozen_stages=0*, *conv_cfg={'type': 'Conv2dAdaptivePadding'}*, *norm_cfg={'eps': 0.001, 'type': 'BN'}*, *act_cfg={'type': 'Swish'}*, *norm_eval=False*, *with_cp=False*, *init_cfg=[{'type': 'Kaiming', 'layer': 'Conv2d'}, {'type': 'Constant', 'layer': ['_BatchNorm', 'GroupNorm'], 'val': 1}]*)

EfficientNet backbone.

参数

- **arch** (*str*) –Architecture of efficientnet. Defaults to b0.
- **out_indices** (*Sequence[int]*) –Output from which stages. Defaults to (6,).
- **frozen_stages** (*int*) –Stages to be frozen (all param fixed). Defaults to 0, which means not freezing any parameters.
- **conv_cfg** (*dict*) –Config dict for convolution layer. Defaults to None, which means using conv2d.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Defaults to dict(type='BN').
- **act_cfg** (*dict*) –Config dict for activation layer. Defaults to dict(type='Swish').
- **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. Defaults to False.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Defaults to False.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

HRNet backbone.

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

参数

- **extra** (*dict*) –Detailed configuration for each stage of HRNet. There must be 4 stages, the configuration for each stage must have 5 keys:
  - num_modules(int): The number of HRModule in this stage.
  - num_branches(int): The number of branches in the HRModule.
  - block(str): The type of convolution block.
  - **num_blocks(tuple): The number of blocks in each branch.**
    The length must be equal to num_branches.
  - **num_channels(tuple): The number of channels in each branch.**
    The length must be equal to num_branches.
- **in_channels** (*int*) –Number of input image channels. Default: 3.
- **conv_cfg** (*dict*) –Dictionary to construct and config conv layer.
- **norm_cfg** (*dict*) –Dictionary to construct and config norm layer.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: True.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.

- **zero_init_residual** (*bool*) –Whether to use zero init for last norm layer in resblocks to let them behave as identity. Default: False.
- **multiscale_output** (*bool*) –Whether to output multi-level features produced by multiple branches. If False, only the first level feature will be output. Default: True.
- **pretrained** (*str, optional*) –Model pretrained path. Default: None.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None.

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:
```

(续下页)

(接上页)

```
...     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.**PyramidVisionTransformer**(*pretrain_img_size=224*, *in_channels=3*, *embed_dims=64*, *num_stages=4*, *num_layers=[3, 4, 6, 3]*, *num_heads=[1, 2, 5, 8]*, *patch_sizes=[4, 2, 2, 2]*, *strides=[4, 2, 2, 2]*, *paddings=[0, 0, 0, 0]*, *sr_ratios=[8, 4, 2, 1]*, *out_indices=(0, 1, 2, 3)*, *mlp_ratios=[8, 8, 4, 4]*, *qkv_bias=True*, *drop_rate=0.0*, *attn_drop_rate=0.0*, *drop_path_rate=0.1*, *use_abs_pos_embed=True*, *norm_after_stage=False*, *use_conv_ffn=False*, *act_cfg={'type': 'GELU'}*, *norm_cfg={'eps': 1e-06, 'type': 'LN'}*, *pretrained=None*, *convert_weights=True*, *init_cfg=None*)

Pyramid Vision Transformer (PVT)

Implementation of Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction without Convolutions.

参数

- **pretrain_img_size** (*int | tuple[int]*) –The size of input image when pretrain. Defaults: 224.
- **in_channels** (*int*) –Number of input channels. Default: 3.
- **embed_dims** (*int*) –Embedding dimension. Default: 64.
- **num_stags** (*int*) –The num of stages. Default: 4.
- **num_layers** (*Sequence[int]*) –The layer number of each transformer encode layer. Default: [3, 4, 6, 3].
- **num_heads** (*Sequence[int]*) –The attention heads of each transformer encode layer. Default: [1, 2, 5, 8].
- **patch_sizes** (*Sequence[int]*) –The patch_size of each patch embedding. Default: [4, 2, 2, 2].
- **strides** (*Sequence[int]*) –The stride of each patch embedding. Default: [4, 2, 2, 2].
- **paddings** (*Sequence[int]*) –The padding of each patch embedding. Default: [0, 0, 0, 0].

- **sr_ratios** (*Sequence[int]*) –The spatial reduction rate of each transformer encode layer. Default: [8, 4, 2, 1].
- **out_indices** (*Sequence[int] | int*) –Output from which stages. Default: (0, 1, 2, 3).
- **mlp_ratios** (*Sequence[int]*) –The ratio of the mlp hidden dim to the embedding dim of each transformer encode layer. Default: [8, 8, 4, 4].
- **qkv_bias** (*bool*) –Enable bias for qkv if True. Default: True.
- **drop_rate** (*float*) –Probability of an element to be zeroed. Default 0.0.
- **attn_drop_rate** (*float*) –The drop out rate for attention layer. Default 0.0.
- **drop_path_rate** (*float*) –stochastic depth rate. Default 0.1.
- **use_abs_pos_embed** (*bool*) –If True, add absolute position embedding to the patch embedding. Defaults: True.
- **use_conv_ffn** (*bool*) –If True, use Convolutional FFN to replace FFN. Default: False.
- **act_cfg** (*dict*) –The activation config for FFNs. Default: dict(type='GELU').
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type='LN').
- **pretrained** (*str, optional*) –model pretrained path. Default: None.
- **convert_weights** (*bool*) –The flag indicates whether the pre-trained model is from the original repo. We may need to convert some keys to make it compatible. Default: True.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

init_weights()

Initialize the weights.

class mmdet.models.backbones.**PyramidVisionTransformerV2**(**kwargs*)

Implementation of PVTv2: Improved Baselines with Pyramid Vision Transformer.

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

RegNet backbone.

More details can be found in paper .

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)
(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:

示例

```
>>> 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),
```

(续下页)

(接上页)

```
...           position='after_conv3')
... ]
>>> self = ResNet(depth=18)
>>> stage_plugins = self.make_stage_plugins(plugins, 0)
>>> assert len(stage_plugins) == 3
```

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

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

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

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

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

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.**SwinTransformer**(*pretrain_img_size=224*, *in_channels=3*, *embed_dims=96*, *patch_size=4*, *window_size=7*, *mlp_ratio=4*, *depths=(2, 2, 6, 2)*, *num_heads=(3, 6, 12, 24)*, *strides=(4, 2, 2, 2)*, *out_indices=(0, 1, 2, 3)*, *qkv_bias=True*, *qk_scale=None*, *patch_norm=True*, *drop_rate=0.0*, *attn_drop_rate=0.0*, *drop_path_rate=0.1*, *use_abs_pos_embed=False*, *act_cfg={'type': 'GELU'}*, *norm_cfg={'type': 'LN'}*, *with_cp=False*, *pretrained=None*, *convert_weights=False*, *frozen_stages=-1*, *init_cfg=None*)

Swin Transformer A PyTorch implement of : *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows* -

https://arxiv.org/abs/2103.14030

Inspiration from https://github.com/microsoft/Swin-Transformer

参数

- **pretrain_img_size** (*int | tuple[int]*) –The size of input image when pretrain. Defaults: 224.
- **in_channels** (*int*) –The num of input channels. Defaults: 3.
- **embed_dims** (*int*) –The feature dimension. Default: 96.
- **patch_size** (*int | tuple[int]*) –Patch size. Default: 4.
- **window_size** (*int*) –Window size. Default: 7.
- **mlp_ratio** (*int | float*) –Ratio of mlp hidden dim to embedding dim. Default: 4.
- **depths** (*tuple[int]*) –Depths of each Swin Transformer stage. Default: (2, 2, 6, 2).
- **num_heads** (*tuple[int]*) –Parallel attention heads of each Swin Transformer stage. Default: (3, 6, 12, 24).
- **strides** (*tuple[int]*) –The patch merging or patch embedding stride of each Swin Transformer stage. (In swin, we set kernel size equal to stride.) Default: (4, 2, 2, 2).
- **out_indices** (*tuple[int]*) –Output from which stages. Default: (0, 1, 2, 3).
- **qkv_bias** (*bool, optional*) –If True, add a learnable bias to query, key, value. Default: True
- **qk_scale** (*float | None, optional*) –Override default qk scale of head_dim ** -0.5 if set. Default: None.
- **patch_norm** (*bool*) –If add a norm layer for patch embed and patch merging. Default: True.

- **drop_rate** (*float*) –Dropout rate. Defaults: 0.
- **attn_drop_rate** (*float*) –Attention dropout rate. Default: 0.
- **drop_path_rate** (*float*) –Stochastic depth rate. Defaults: 0.1.
- **use_abs_pos_embed** (*bool*) –If True, add absolute position embedding to the patch embedding. Defaults: False.
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type='GELU').
- **norm_cfg** (*dict*) –Config dict for normalization layer at output of backone. Defaults: dict(type='LN').
- **with_cp** (*bool, optional*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **pretrained** (*str, optional*) –model pretrained path. Default: None.
- **convert_weights** (*bool*) –The flag indicates whether the pre-trained model is from the original repo. We may need to convert some keys to make it compatible. Default: False.
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). Default: -1 (-1 means not freezing any parameters).
- **init_cfg** (*dict, optional*) –The Config for initialization. Defaults to None.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

init_weights()

Initialize the weights.

train(*mode=True*)

Convert the model into training mode while keep layers freezed.

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

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

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

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.

29.3 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.

备注: 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*, *block_dilations*)

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.
- **block_dilations** (*list*) –The list of residual blocks dilation.

forward(*feature*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

备注: 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.**DyHead**(*in_channels*, *out_channels*, *num_blocks=6*, *zero_init_offset=True*, *init_cfg=None*)

DyHead neck consisting of multiple DyHead Blocks.

See Dynamic Head: Unifying Object Detection Heads with Attentions for details.

参数

- **in_channels** (*int*) –Number of input channels.
- **out_channels** (*int*) –Number of output channels.
- **num_blocks** (*int, optional*) –Number of DyHead Blocks. Default: 6.
- **zero_init_offset** (*bool, optional*) –Whether to use zero init for *spatial_conv_offset*. Default: True.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None.

forward(*inputs*)

Forward function.

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.

备注: 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** (*dict*) –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()
```

(续下页)

(接上页)

```
>>> 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`

备注： `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.

备注:

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.

备注： 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]

29.4 dense_heads

29.5 roi_heads

29.6 losses

29.7 utils

class mmdet.models.utils.**AdaptiveAvgPool2d**(*output_size: Union[int, None, Tuple[Optional[int], ...]]*)

Handle empty batch dimension to AdaptiveAvgPool2d.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

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

Cross Stage Partial Layer.

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.

备注: 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.

备注： 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.**DyReLU**(*channels*, *ratio=4*, *conv_cfg=None*, *act_cfg=({'type': 'ReLU'}, {'type': 'HSigmoid', 'bias': 3.0, 'divisor': 6.0})*, *init_cfg=None*)

Dynamic ReLU (DyReLU) module.

See Dynamic ReLU for details. Current implementation is specialized for task-aware attention in DyHead. HSigmoid arguments in default act_cfg follow DyHead official code. https://github.com/microsoft/DynamicHead/blob/master/dyhead/dyrelu.py

参数

- **channels** (*int*) –The input (and output) channels of DyReLU module.
- **ratio** (*int*) –Squeeze ratio in Squeeze-and-Excitation-like module, the intermediate channel will be `int(channels/ratio)`. Default: 4.
- **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='HSigmoid', bias=3.0, divisor=6.0))
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None

forward(*x*)

Forward function.

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

Implements Dynamic Convolution.

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

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.
- **with_proj** (*bool*) –Project two-dimentional feature to one-dimentional feature. Default to True.
- **act_cfg** (*dict*) –The activation config for DynamicConv.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default layer normalization.
- **(obj** (*init_cfg*) –*mmcv.ConfigDict*): The Config for initialization. Default: None.

forward(*param_feature*, *input_feature*)

Forward function for *DynamicConv*.

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'}*, *drop_path_rate=0.0*, *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 kernel size of the depthwise convolution. Default: 3.
- **stride** (*int*) –The stride of the depthwise convolution. Default: 1.
- **se_cfg** (*dict*) –Config dict for se layer. Default: None, which means no se layer.

- **with_expand_conv** (*bool*) –Use expand conv or not. If set False, mid_channels must be the same with in_channels. Default: True.
- **conv_cfg** (*dict*) –Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type='BN').
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type='ReLU').
- **drop_path_rate** (*float*) –stochastic depth rate. Defaults to 0.
- **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.

备注: 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.

备注: 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.

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

---

class mmdet.models.utils.**PatchEmbed**(*in_channels=3*, *embed_dims=768*, *conv_type='Conv2d'*, *kernel_size=16*, *stride=16*, *padding='corner'*, *dilation=1*, *bias=True*, *norm_cfg=None*, *input_size=None*, *init_cfg=None*)

Image to Patch Embedding.

We use a conv layer to implement PatchEmbed.

参数

- **in_channels** (*int*) –The num of input channels. Default: 3
- **embed_dims** (*int*) –The dimensions of embedding. Default: 768
- **conv_type** (*str*) –The config dict for embedding conv layer type selection. Default: "Conv2d.
- **kernel_size** (*int*) –The kernel_size of embedding conv. Default: 16.
- **stride** (*int*) –The slide stride of embedding conv. Default: None (Would be set as *kernel_size*).
- **padding** (*int | tuple | string*) –The padding length of embedding conv. When it is a string, it means the mode of adaptive padding, support "same" and "corner" now. Default: "corner".
- **dilation** (*int*) –The dilation rate of embedding conv. Default: 1.
- **bias** (*bool*) –Bias of embed conv. Default: True.
- **norm_cfg** (*dict, optional*) –Config dict for normalization layer. Default: None.
- **input_size** (*int | tuple | None*) –The size of input, which will be used to calculate the out size. Only work when *dynamic_size* is False. Default: None.
- **init_cfg** (*mmcv.ConfigDict*, optional) –The Config for initialization. Default: None.

forward(*x*)

参数

x (*Tensor*) –Has shape (B, C, H, W). In most case, C is 3.

返回

Contains merged results and its spatial shape.

- x (Tensor): Has shape (B, out_h * out_w, embed_dims)
- **out_size (tuple[int]): Spatial shape of x, arrange as** (out_h, out_w).

返回类型

tuple

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

ResLayer to build ResNet style backbone.

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.

备注: 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.**adaptive_avg_pool2d**(*input*, *output_size*)

Handle empty batch dimension to adaptive_avg_pool2d.

参数

- **input** (*tensor*) –4D tensor.
- **output_size** (*int, tuple[int,int]*) –the target output size.

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

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

- type (str): Layer type.

- layer args: Args needed to instantiate an linear layer.

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.

```
|<   width   >|

lt-+----------+         -
|  |          |         ^
+--x----------+--+
|  |          |  |
|  |          |  |    height
|  | overlap  |  |
|  |          |  |
|  |          |  |      v
+--+---------br--+      -
   |          |  |
   +----------+--x
```

To ensure IoU of generated box and gt box is larger than `min_overlap`:

$$\frac{(w-r)*(h-r)}{w*h+(w+h)r-r^2} \geq iou \quad \Rightarrow \quad r^2-(w+h)r+\frac{1-iou}{1+iou}*w*h \geq 0$$

$$a=1, \quad b=-(w+h), \quad c=\frac{1-iou}{1+iou}*w*h$$

$$r \leq \frac{-b-\sqrt{b^2-4*a*c}}{2*a}$$

- Case2: both two corners are inside the gt box.

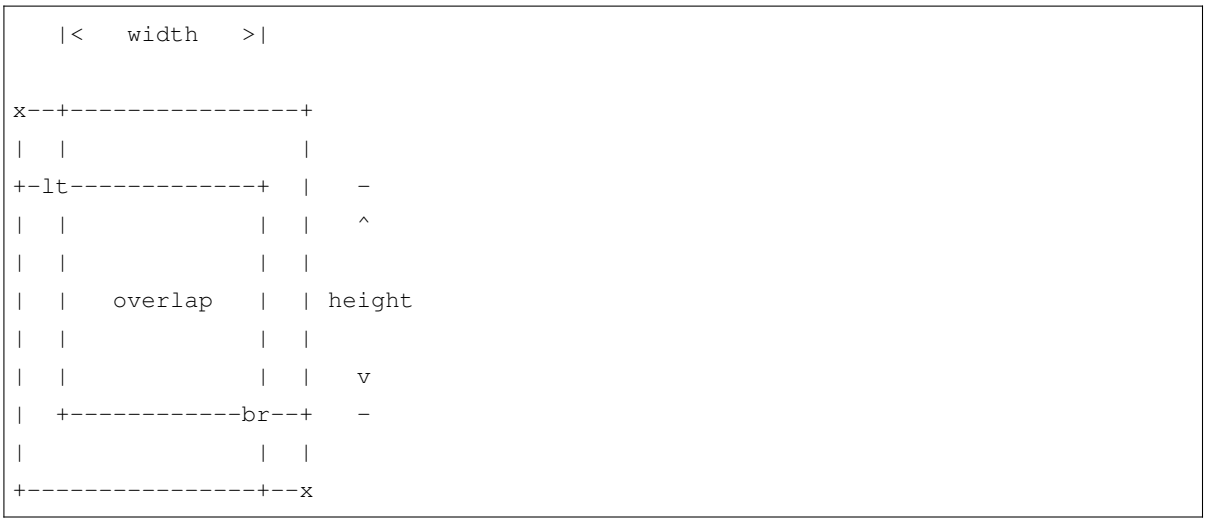
To ensure IoU of generated box and gt box is larger than `min_overlap`:

$$\frac{(w-2*r)*(h-2*r)}{w*h} \geq iou \quad \Rightarrow \quad 4r^2-2(w+h)r+(1-iou)*w*h \geq 0$$

$$a=4, \quad b=-2(w+h), \quad c=(1-iou)*w*h$$

$$r \leq \frac{-b-\sqrt{b^2-4*a*c}}{2*a}$$

- 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.**get_uncertain_point_coords_with_randomness**(*mask_pred*, *labels*, *num_points*, *oversample_ratio*, *importance_sample_ratio*)

Get `num_points` most uncertain points with random points during train.

Sample points in [0, 1] x [0, 1] coordinate space based on their uncertainty. The uncertainties are calculated for each point using ‘get_uncertainty()’ function that takes point’s logit prediction as input.

参数

- **mask_pred** (*Tensor*) –A tensor of shape (num_rois, num_classes, mask_height, mask_width) for class-specific or class-agnostic prediction.
- **labels** (*list*) –The ground truth class for each instance.
- **num_points** (*int*) –The number of points to sample.
- **oversample_ratio** (*int*) –Oversampling parameter.
- **importance_sample_ratio** (*float*) –Ratio of points that are sampled via importnace sampling.

返回

A tensor of shape (num_rois, num_points, 2)
that contains the coordinates sampled points.

返回类型
point_coords (Tensor)

mmdet.models.utils.**get_uncertainty**(*mask_pred*, *labels*)

Estimate uncertainty based on pred logits.

We estimate uncertainty as L1 distance between 0.0 and the logits prediction in 'mask_pred' for the foreground class in *classes*.

参数

- **mask_pred** (*Tensor*) –mask predication logits, shape (num_rois, num_classes, mask_height, mask_width).
- **labels** (*list[Tensor]*) –Either predicted or ground truth label for each predicted mask, of length num_rois.

返回

Uncertainty scores with the most uncertain
locations having the highest uncertainty score, shape (num_rois, 1, mask_height, mask_width)

返回类型
scores (Tensor)

mmdet.models.utils.**interpolate_as**(*source*, *target*, *mode='bilinear'*, *align_corners=False*)

Interpolate the *source* to the shape of the *target*.

The *source* must be a Tensor, but the *target* can be a Tensor or a np.ndarray with the shape (…, target_h, target_w).

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

mmdet.models.utils.**nchw_to_nlc**(*x*)

Flatten [N, C, H, W] shape tensor to [N, L, C] shape tensor.

参数

x (*Tensor*) –The input tensor of shape [N, C, H, W] before conversion.

返回

The output tensor of shape [N, L, C] after conversion.

返回类型

Tensor

mmdet.models.utils.**nlc_to_nchw**(*x*, *hw_shape*)

Convert [N, L, C] shape tensor to [N, C, H, W] shape tensor.

参数

- **x** (*Tensor*) –The input tensor of shape [N, L, C] before conversion.
- **hw_shape** (*Sequence[int]*) –The height and width of output feature map.

返回

The output tensor of shape [N, C, H, W] after conversion.

返回类型

Tensor

mmdet.models.utils.**preprocess_panoptic_gt**(*gt_labels*, *gt_masks*, *gt_semantic_seg*, *num_things*, *num_stuff*, *img_metas*)

Preprocess the ground truth for a image.

参数

- **gt_labels** (*Tensor*) –Ground truth labels of each bbox, with shape (num_gts,).
- **gt_masks** (*BitmapMasks*) –Ground truth masks of each instances of a image, shape (num_gts, h, w).
- **gt_semantic_seg** (*Tensor | None*) –Ground truth of semantic segmentation with the shape (1, h, w). [0, num_thing_class - 1] means things, [num_thing_class, num_class-1] means stuff, 255 means VOID. It's None when training instance segmentation.
- **img_metas** (*dict*) –List of image meta information.

返回

a tuple containing the following targets.

- **labels (Tensor): Ground truth class indices for a**
  image, with shape (n,), n is the sum of number of stuff type and number of instance in a image.
- **masks (Tensor): Ground truth mask for a image, with**
  shape (n, h, w). Contains stuff and things when training panoptic segmentation, and things only when training instance segmentation.

返回类型

tuple

CHAPTER 30

mmdet.utils

CHAPTER 31

NPU (华为🅵腾)

31.1 使用方法

请参考 MMCV 的安装文档 来安装 NPU 版本的 MMCV。

以下展示单机八卡场景的运行指令:

```
bash tools/dist_train.sh configs/ssd/ssd300_coco.py 8
```

以下展示单机单卡下的运行指令:

```
python tools/train.py configs/ssd/ssd300_coco.py
```

31.2 模型验证结果

注意:

- 如果没有特别标记，NPU 上的结果与使用 FP32 的 GPU 上的结果结果相同。
- (*) 这些模型在 NPU 上的结果与 GPU 上的混合精度训练结果一致，但低于 FP32 的结果。这种情况主要与模型本身在混合精度训练中的特点有关，用户可以自行调整超参数来获得更高精度。
- (**) GPU 上 yolox-s 在混合精度下的精度为 40.1 低于 readme 中 40.5 的水平; 默认情况下，yolox-s 启用 `persister_woker=True`，但这个参数目前在 NPU 上存在一些 bug，会导致在最后几个 epoch 由于资源耗尽报错退出，对整体精度影响有限可以忽略。

31.3 Ascend 加速模块验证结果

优化方案简介：

1. 修改循环计算为一次整体计算，目的是减少下发指令数量。
2. 修改索引计算为掩码计算，原因是 SIMD 架构芯片擅长处理连续数据计算。

以上模型结果由华为🄴腾团队提供

CHAPTER 32

Indices and tables

- genindex
- search

Python 模块索引

m

索引

G

H

I

L

M

N

P

Q

R

S

T

V

Y

模块