mmrotate

MMRotate Author

2022 年 10 月 27 日

学习基础知识

CHAPTER 1

学习基础知识

本章将向您介绍旋转目标检测的基本概念，以及旋转目标检测的框架 MMRotate，并提供了详细教程的链接。

1.1 什么是旋转目标检测

1.1.1 问题定义

受益于通用检测的蓬勃发展，目前绝大多数的旋转检测模型都是基于经典的通用检测器。随着检测任务的发展，水平框在一些细分领域上已经无法满足研究人员的需求。通过重新定义目标表示形式以及增加回归自由度数量的操作来实现旋转矩形框、四边形甚至任意形状检测，我们称之为旋转目标检测。如何更加高效地进行高精度的旋转目标检测已成为当下的研究热点。下面列举一些旋转目标检测已经被应用或者有巨大潜力的领域：人脸识别、场景文字、遥感影像、自动驾驶、医学图像、机器人抓取等。

1.1.2 什么是旋转框

旋转目标检测与通用目标检测最大的不同就是用旋转框标注来代替水平框标注，它们的定义如下：

- 水平框: 宽沿 `x` 轴方向，高沿 `y` 轴方向的矩形。通常可以用 2 个对角顶点的坐标表示 `(x_i, y_i)` (i = 1, 2)，也可以用中心点坐标以及宽和高表示 `(x_center, y_center, width, height)`。
- 旋转框: 由水平框绕中心点旋转一个角度 `angle` 得到，通过添加一个弧度参数得到其旋转框定义法 `(x_center, y_center, width, height, theta)`。其中，`theta = angle * pi / 180`，单位为 `rad`。当旋转的角度为 90° 的倍数时，旋转框退化为水平框。标注软件导出的旋转框标注通常为多边形 `(xr_i, yr_i)` (i = 1, 2, 3, 4)，在训练时需要转换为旋转框定义法。

注解： 在 MMRotate 中，角度参数的单位均为弧度。

1.1.3 旋转方向

旋转框可以由水平框绕其中心点顺时针旋转或逆时针旋转得到。旋转方向和坐标系的选择密切相关。图像空间采用右手坐标系（y，x），其中 y 是上-> 下，x 是左-> 右。此时存在 2 种相反的旋转方向：

- 顺时针 (CW)

CW 的示意图

```
0-------------------> x (0 rad)
|  A-------------B
|  |             |
|  |     box     h
|  |   angle=0   |
|  D------w------C
v
y (pi/2 rad)
```

CW 的旋转矩阵

$$\begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}$$

CW 的旋转变换

$$\begin{aligned} P_A = \begin{pmatrix} x_A \\ y_A \end{pmatrix} &= \begin{pmatrix} x_{center} \\ y_{center} \end{pmatrix} + \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} -0.5w \\ -0.5h \end{pmatrix} \\ &= \begin{pmatrix} x_{center} - 0.5w\cos\alpha + 0.5h\sin\alpha \\ y_{center} - 0.5w\sin\alpha - 0.5h\cos\alpha \end{pmatrix} \end{aligned}$$

- 逆时针 (CCW)

CCW 的示意图

```
0-------------------> x (0 rad)
|  A-------------B
|  |             |
|  |     box     h
|  |   angle=0   |
|  D------w------C
v
y (-pi/2 rad)
```

CCW 的旋转矩阵

$$\begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix}$$

CCW 的旋转变换

$$\begin{aligned} P_A = \begin{pmatrix} x_A \\ y_A \end{pmatrix} &= \begin{pmatrix} x_{center} \\ y_{center} \end{pmatrix} + \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} -0.5w \\ -0.5h \end{pmatrix} \\ &= \begin{pmatrix} x_{center} - 0.5w\cos\alpha - 0.5h\sin\alpha \\ y_{center} + 0.5w\sin\alpha - 0.5h\cos\alpha \end{pmatrix} \end{aligned}$$

在 MMCV 中可以设置旋转方向的算子有：

- box_iou_rotated (默认为 CW)
- nms_rotated (默认为 CW)
- RoIAlignRotated (默认为 CCW)
- RiRoIAlignRotated (默认为 CCW)。

注解： 在 MMRotate 中，旋转框的旋转方向均为 CW。

1.1.4 旋转框定义法

由于 theta 定义范围的不同，在旋转目标检测中逐渐派生出如下 3 种旋转框定义法：

- $D_{oc'}$：OpenCV 定义法，angle∈(0, 90°]，theta∈(0, pi / 2]，width 与 x 正半轴之间的夹角为正的锐角。该定义法源于 OpenCV 中的 cv2.minAreaRect 函数，其返回值为 (0, 90°]。
- D_{le135}：长边 135° 定义法，angle∈[-45°, 135°)，theta∈[-pi / 4, 3 * pi / 4) 并且 width > height。
- D_{le90}：长边 90° 定义法，angle∈[-90°, 90°)，theta∈[-pi / 2, pi / 2) 并且 width > height。

三种定义法之间的转换关系在 MMRotate 内部并不涉及，因此不多做介绍。如果想了解更多的细节，可以参考这篇博客。

注解： MMRotate 同时支持上述三种旋转框定义法，可以通过配置文件灵活切换。

需要注意的是，在 4.5.1 之前的版本中，cv2.minAreaRect 的返回值为 [-90°, 0°) （参考资料）。为了便于区分，将老版本的 OpenCV 定义法记作 D_{oc}。

- $D_{oc'}$: OpenCV 定义法，opencv>=4.5.1，angle∈(0, 90°]，theta∈(0, pi / 2]。

- D_{oc}: 老版的 OpenCV 定义法，`opencv<4.5.1, angle∈[-90°, 0°), theta∈[-pi / 2, 0)`。

两种 OpenCV 定义法的转换关系如下：

$$D_{oc'}\left(w_{oc'}, h_{oc'}, \theta_{oc'}\right) = \begin{cases} D_{oc}\left(h_{oc}, w_{oc}, \theta_{oc} + \pi/2\right), otherwise \\ D_{oc}\left(w_{oc}, h_{oc}, \theta_{oc} + \pi\right), \theta_{oc} = -\pi/2 \end{cases}$$

$$D_{oc}\left(w_{oc}, h_{oc}, \theta_{oc}\right) = \begin{cases} D_{oc'}\left(h_{oc'}, w_{oc'}, \theta_{oc'} - \pi/2\right), otherwise \\ D_{oc'}\left(w_{oc'}, h_{oc'}, \theta_{oc'} - \pi\right), \theta_{oc'} = \pi/2 \end{cases}$$

注解： 不管您使用的 OpenCV 版本是多少, MMRotate 都会将 OpenCV 定义法的 theta 转换为 (0, pi / 2]。

1.1.5 评估

评估 mAP 的代码中涉及 IoU 的计算，可以直接计算旋转框 IoU，也可以将旋转框转换为多边形，然后计算多边形 IoU (DOTA 在线评估使用的是计算多边形 IoU)。

1.2 什么是 MMRotate

MMRotate 是一个为旋转目标检测方法提供统一训练和评估框架的工具箱，以下是其整体框架：：

MMRotate 包括四个部分, `datasets`, `models`, `core` and `apis`.

- `datasets` 用于数据加载和数据增强。在这部分, 我们支持了各种旋转目标检测数据集和数据增强预处理。
- `models` 包括模型和损失函数。
- `core` 为模型训练和评估提供工具。
- `apis` 为模型训练、测试和推理提供高级 API。

MMRotate 的模块设计如下图所示：

其中由于旋转框定义法不同而需要注意的地方有如下几个：

- 读取标注
- 数据增强
- 指派样本
- 评估指标

1.3 如何使用教程

下面是 MMRotate 详细的分步指南:

1. 关于安装说明, 请参阅安装.
2. 开始 介绍了 MMRotate 的基本用法.
3. 如果想要更加深入了解 MMRotate，请参阅以下教程:

- 配置
- 自定义数据集
- 自定义模型
- 自定义运行时

CHAPTER 2

依赖

在本节中，我们将演示如何使用 PyTorch 准备环境。

MMRotate 能够在 Linux 和 Windows 上运行。它依赖于 Python 3.7+, CUDA 9.2+ 和 PyTorch 1.6+。

注解： 如果您对 PyTorch 有经验并且已经安装了它，只需跳过此部分并跳到下一节 。否则，您可以按照以下步骤进行准备。

第 0 步： 从 官网 下载并安装 Miniconda。

第 1 步： 创建一个 conda 环境并激活它.

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

第 2 步： 根据 官方说明 安装 PyTorch, 例如：

```
conda install pytorch==1.8.0 torchvision==0.9.0 cudatoolkit=10.2 -c pytorch
```

CHAPTER 3

安装

我们建议用户按照我们的最佳实践安装 MMRotate。然而，整个过程是高度可定制的。有关详细信息，请参阅自定义安装 部分。

3.1 最佳实践

第 0 步： 使用 MIM 安装 MMCV 和 MMDetection

```
pip install -U openmim
mim install mmcv-full
mim install mmdet
```

第 1 步： 安装 MMRotate.

案例 a：如果您直接开发并运行 mmrotate，请从源代码安装：

```
git clone https://github.com/open-mmlab/mmrotate.git
cd mmrotate
pip install -v -e .
# "-v" 表示详细或更多输出
# "-e" 表示以可编辑模式安装项目，
# 因此，对代码进行的任何本地修改都将在不重新安装的情况下生效。
```

案例 b：如果将 mmrotate 作为依赖项或第三方软件包，请使用 pip 安装它：

```
pip install mmrotate
```

3.2 验证

为了验证是否正确安装了 MMRotate，我们提供了一些示例代码来运行推理演示。

第 1 步：我们需要下载配置文件和检查点文件。

```
mim download mmrotate --config oriented_rcnn_r50_fpn_1x_dota_le90 --dest .
```

下载需要几秒钟或更长时间，具体取决于您的网络环境。当下载完成之后，您将会在当前文件夹下找到 `oriented_rcnn_r50_fpn_1x_dota_le90.py` 和 `oriented_rcnn_r50_fpn_1x_dota_le90-6d2b2ce0.pth` 这两个文件。

第 2 步：验证推理演示

选项（a）：如果从源代码安装 mmrotate，只需运行以下命令。

```
python demo/image_demo.py demo/demo.jpg oriented_rcnn_r50_fpn_1x_dota_le90.py␣
↪oriented_rcnn_r50_fpn_1x_dota_le90-6d2b2ce0.pth --out-file result.jpg
```

您将在当前目录下看到一张名为 `result.jpg` 的新图片，其中旋转边界框绘制在汽车、公共汽车等目标上。

选项（b）：如果使用 pip 安装 mmrotate，请打开 python 解释器并复制和粘贴以下代码。

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

config_file = 'oriented_rcnn_r50_fpn_1x_dota_le90.py'
checkpoint_file = 'oriented_rcnn_r50_fpn_1x_dota_le90-6d2b2ce0.pth'
model = init_detector(config_file, checkpoint_file, device='cuda:0')
inference_detector(model, 'demo/demo.jpg')
```

您将看到打印的数组列表，表示检测到的旋转边界框。

3.3 自定义安装

3.3.1 CUDA 版本

安装 PyTorch 时，需要指定 CUDA 的版本。如果您不清楚选择哪一个，请遵循我们的建议：

- 对于基于安培架构的 NVIDIA GPU，如 GeForce 30 系列和 NVIDIA A100，必须使用 CUDA 11。
- 对于较旧的 NVIDIA GPU，CUDA 11 向后兼容，但 CUDA 10.2 更轻量并且具有更好的兼容性。

请确保 GPU 驱动程序满足最低版本要求。请查询 表格 以获得更多信息。

注解： 如果遵循我们的最佳实践，安装 CUDA 运行时库就足够了，因为不会在本地编译 CUDA 代码。但是，如果您希望从源代码处编译 MMCV 或开发其他 CUDA 算子，则需要从 NVIDIA 的 网站 安装完整的 CUDA 工具包，其版本应与 PyTorch 的 CUDA 版本匹配。例如使用 `conda install` 命令指定 cudatoolkit 的版本。

3.3.2 不使用 MIM 安装 MMCV

MMCV 包含 C++ 和 CUDA 扩展，因此以复杂的方式依赖于 PyTorch。MIM 会自动解决此类依赖关系，并使安装更容易。然而，这不是必须的。

要使用 pip 而不是 MIM 安装 MMCV，请遵循 MMCV 安装指南 。这需要根据 PyTorch 版本及其 CUDA 版本手动指定 find-url。

例如, 以下命令安装了为 PyTorch 1.9.x 和 CUDA 10.2 构建的 mmcv-full。

```
pip install mmcv-full -f https://download.openmmlab.com/mmcv/dist/cu102/torch1.8/
↪index.html
```

3.3.3 在 Google Colab 安装

Google Colab 通常已经安装了 PyTorch，因此，我们只需要使用以下命令安装 MMCV 和 MMDetection。

第 1 步： 使用 MIM 安装 MMCV 和 MMDetection 。

```
!pip3 install -U openmim
!mim install mmcv-full
!mim install mmdet
```

第 2 步： 从源码安装 MMRotate。

```
!git clone https://github.com/open-mmlab/mmrotate.git
%cd mmrotate
!pip install -e .
```

第 3 步： 验证。

```
import mmrotate
print(mmrotate.__version__)
# Example output: 0.3.1
```

注解： 在 Jupyter 中，感叹号 ！用于调用外部可执行文件，`%cd` 是一个 魔术命令 用于更改 Python 的当前工作目录。

3.3.4 在 Docker 镜像中使用 MMRotate

我们提供了 Dockerfile 用于创建镜像。请确保您的 docker 版本 >=19.03。

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

使用下列命令运行

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

3.4 故障排除

如果您在安装过程中遇到一些问题，请先查看*FAQ* 页面。如果没有找到解决方案，您可以在 GIthub 上 提问 。

CHAPTER 4

准备数据集

具体的细节可以参考 准备数据 下载并组织数据集。

CHAPTER 5

测试一个模型

- 单个 GPU
- 单个节点多个 GPU
- 多个节点多个 GPU

您可以使用以下命令来推理数据集。

```
# 单个 GPU
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [optional arguments]

# 多个 GPU
./tools/dist_test.sh ${CONFIG_FILE} ${CHECKPOINT_FILE} ${GPU_NUM} [optional arguments]

# slurm 环境中多个节点
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [optional arguments] --
↪launcher slurm
```

例子:

在 DOTA-1.0 数据集推理 RotatedRetinaNet 并生成压缩文件用于在线提交 (首先请修改 data_root)。

```
python ./tools/test.py  \
  configs/rotated_retinanet/rotated_retinanet_obb_r50_fpn_1x_dota_le90.py \
  checkpoints/SOME_CHECKPOINT.pth --format-only \
  --eval-options submission_dir=work_dirs/Task1_results
```

或者

```
./tools/dist_test.sh  \
  configs/rotated_retinanet/rotated_retinanet_obb_r50_fpn_1x_dota_le90.py \
  checkpoints/SOME_CHECKPOINT.pth 1 --format-only \
  --eval-options submission_dir=work_dirs/Task1_results
```

您可以修改 data_root 中测试集的路径为验证集或训练集路径用于离线的验证。

```
python ./tools/test.py \
  configs/rotated_retinanet/rotated_retinanet_obb_r50_fpn_1x_dota_le90.py \
  checkpoints/SOME_CHECKPOINT.pth --eval mAP
```

或者

```
./tools/dist_test.sh  \
  configs/rotated_retinanet/rotated_retinanet_obb_r50_fpn_1x_dota_le90.py \
  checkpoints/SOME_CHECKPOINT.pth 1 --eval mAP
```

您也可以可视化结果。

```
python ./tools/test.py \
  configs/rotated_retinanet/rotated_retinanet_obb_r50_fpn_1x_dota_le90.py \
  checkpoints/SOME_CHECKPOINT.pth \
  --show-dir work_dirs/vis
```

CHAPTER 6

训练一个模型

6.1 单 GPU 训练

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

如果您想在命令行中指定工作路径，您可以增加参数 `--work_dir ${YOUR_WORK_DIR}`。

6.2 多 GPU 训练

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

可选参数包括:

- `--no-validate` (**不建议**): 默认情况下代码将在训练期间进行评估。通过设置 `--no-validate` 关闭训练期间进行评估。
- `--work-dir ${WORK_DIR}`: 覆盖配置文件中指定的工作目录。
- `--resume-from ${CHECKPOINT_FILE}`: 从以前的检查点恢复训练。

`resume-from` 和 `load-from` 的不同点:

`resume-from` 读取模型的权重和优化器的状态，并且 epoch 也会继承于指定的检查点。通常用于恢复意外中断的训练过程。`load-from` 只读取模型的权重并且训练的 epoch 会从 0 开始。通常用于微调。

6.3 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。

6.4 使用 Slurm 来管理任务

如果您在 slurm 管理的集群上运行 MMRotate，您可以使用脚本 `slurm_train.sh` (此脚本还支持单机训练)。

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

如果您有多台机器联网，您可以参考 PyTorch launch utility。如果您没有像 InfiniBand 这样的高速网络，训练速度通常会很慢。

6.5 在一台机器上启动多个作业

如果您在一台机器上启动多个作业，如在一台机器上使用 8 张 GPU 训练 2 个作业，每个作业使用 4 张 GPU，您需要为每个作业指定不同的端口号 (默认为 29500) 进而避免通讯冲突。

如果您使用 `dist_train.sh` 启动训练，您可以在命令行中指定端口号。

```
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
```

如果您通过 Slurm 启动训练，您需要修改配置文件 (通常是配置文件底部的第 6 行) 进而设置不同的通讯端口。

在 `config1.py` 中,

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

在 `config2.py` 中,

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

之后您可以使用 config1.py 和 config2.py 开启两个作业。

```
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

基准和模型库

- Rotated RetinaNet-OBB/HBB (ICCV’ 2017)
- Rotated FasterRCNN-OBB (TPAMI’ 2017)
- Rotated RepPoints-OBB (ICCV’ 2019)
- Rotated FCOS (ICCV’ 2019)
- RoI Transformer (CVPR’ 2019)
- Gliding Vertex (TPAMI’ 2020)
- Rotated ATSS-OBB (CVPR’ 2020)
- CSL (ECCV’ 2020)
- R3Det (AAAI’ 2021)
- S2A-Net (TGRS’ 2021)
- ReDet (CVPR’ 2021)
- Beyond Bounding-Box (CVPR’ 2021)
- Oriented R-CNN (ICCV’ 2021)
- GWD (ICML’ 2021)
- KLD (NeurIPS’ 2021)
- SASM (AAAI’ 2022)
- KFIoU (arXiv)

- G-Rep (stay tuned)

7.1 DOTA v1.0 数据集上的结果

- `MS` 表示多尺度图像增强。
- `RR` 表示随机旋转增强。

上述模型都是使用 1 * 1080ti/2080ti 训练得到的，并且在 1 * 2080ti 上进行推理测试。

CHAPTER 8

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。mmrotate 是建立在 mmdet 之上的，因此强烈建议学习 mmdet 的基本知识。

8.1 通过脚本参数修改配置

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

- 更新字典链的配置

  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。

8.2 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]_
↪{dataset}_{data setting}_{angle version}
```

`{xxx}` 是被要求的文件 `[yyy]` 是可选的。

- `{model}`: 模型种类，例如 `rotated_faster_rcnn`, `rotated_retinanet` 等。
- `[model setting]`: 特定的模型，例如 `hbb` for `rotated_retinanet` 等。
- `{backbone}`: 主干网络种类例如 `r50` (ResNet-50), `swin_tiny` (SWIN-tiny) 。
- `{neck}`: Neck 模型的种类包括 `fpn`, `refpn`。
- `[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 `1xb2`。
- `{dataset}`: 数据集，例如 `dota`。
- `{angle version}`: 旋转定义方式，例如 `oc`, `le135` 或者 `le90`。

8.3 RotatedRetinaNet 配置文件示例

为了帮助用户对 MMRotate 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 RotatedRetinaNet 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。

```
angle_version = 'oc'  # 旋转定义方式
model = dict(
    type='RotatedRetinaNet',  # 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
        depth=50,  # 主干网络的深度
        num_stages=4,  # 主干网络阶段 (stages) 的数目
        out_indices=(0, 1, 2, 3),  # 每个阶段产生的特征图输出的索引
        frozen_stages=1,  # 第一个阶段的权重被冻结
        zero_init_residual=False,  # 是否对残差块 (resblocks) 中的最后一个归一化层使用零初始化
(zero init) 让它们表现为自身
        norm_cfg=dict(  # 归一化层 (norm layer) 的配置项
```

(下页继续)

(续上页)

```
            type='BN',  # 归一化层的类别，通常是 BN 或 GN
            requires_grad=True),  # 是否训练归一化里的 gamma 和 beta
        norm_eval=True,  # 是否冻结 BN 里的统计项
        style='pytorch',  # 主干网络的风格，'pytorch' 意思是步长为 2 的层为 3x3 卷积， 'caffe
↪' 意思是步长为 2 的层为 1x1 卷积。
        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 'ReFPN'
        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
        start_level=1,  # 用于构建特征金字塔的主干网络起始输入层索引值
        add_extra_convs='on_input',  # 决定是否在原始特征图之上添加卷积层
        num_outs=5),  # 决定输出多少个尺度的特征图 (scales)
    bbox_head=dict(
        type='RotatedRetinaHead',# bbox_head 的类型是 'RRetinaHead'
        num_classes=15,  # 分类的类别数量
        in_channels=256,  # bbox head 输入通道数
        stacked_convs=4,  # head 卷积层的层数
        feat_channels=256,  # head 卷积层的特征通道
        assign_by_circumhbbox='oc',  # obb2hbb 的旋转定义方式
        anchor_generator=dict(  # 锚点 (Anchor) 生成器的配置
            type='RotatedAnchorGenerator',  # 锚点生成器类别
            octave_base_scale=4,  # RetinaNet 用于生成锚点的超参数，特征图 anchor 的基本尺度。
值越大，所有 anchor 的尺度都会变大。
            scales_per_octave=3,  #  RetinaNet 用于生成锚点的超参数，每个特征图有 3 个尺度
            ratios=[1.0, 0.5, 2.0],  # 高度和宽度之间的比率
            strides=[8, 16, 32, 64, 128]),  # 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='DeltaXYWHAOBBoxCoder',  # 框编码器的类别
            angle_range='oc',  # 框编码器的旋转定义方式
            norm_factor=None,  # 框编码器的范数
            edge_swap=False,  # 设置是否启用框编码器的边缘交换
            proj_xy=False,  # 设置是否启用框编码器的投影
            target_means=(0.0, 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, 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='FocalLoss',  # 分类分支的损失函数类型
            use_sigmoid=True,  #  是否使用 sigmoid
            gamma=2.0,  # Focal Loss 用于解决难易不均衡的参数 gamma
            alpha=0.25,  # Focal Loss 用于解决样本数量不均衡的参数 alpha
            loss_weight=1.0),  # 分类分支的损失权重
```

(下页继续)

(续上页)

```
        loss_bbox=dict(  # 回归分支的损失函数配置
            type='L1Loss',  # 回归分支的损失类型
            loss_weight=1.0)),  # 回归分支的损失权重
    train_cfg=dict(  # 训练超参数的配置
        assigner=dict(  # 分配器 (assigner) 的配置
            type='MaxIoUAssigner',  # 分配器的类型
            pos_iou_thr=0.5,  # IoU >= 0.5(阈值) 被视为正样本
            neg_iou_thr=0.4,  # IoU < 0.4(阈值) 被视为负样本
            min_pos_iou=0,  # 将框作为正样本的最小 IoU 阈值
            ignore_iof_thr=-1,  # 忽略 bbox 的 IoF 阈值
            iou_calculator=dict(type='RBboxOverlaps2D')),  # IoU 的计算器类型
        allowed_border=-1,  # 填充有效锚点 (anchor) 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
    test_cfg=dict(  # 测试超参数的配置
        nms_pre=2000,  # NMS 前的 box 数
        min_bbox_size=0,  # box 允许的最小尺寸
        score_thr=0.05,  # bbox 的分数阈值
        nms=dict(iou_thr=0.1), # NMS 的阈值
        max_per_img=2000))  # 每张图像的最大检测次数
dataset_type = 'DOTADataset'  # 数据集类型，这将被用来定义数据集
data_root = '../datasets/split_1024_dota1_0/'  # 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),  # 是否加载标注框 (bounding box)，目标检测需要设置为 True
    dict(type='RResize',  # 变化图像和其注释大小的数据增广的流程
         img_scale=(1024, 1024)),  # 图像的最大规模
    dict(type='RRandomFlip',  # 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
         version='oc'),  # 定义旋转的方式
    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，
        std=[58.395, 57.12, 57.375],  # 因为 img_norm_cfg 被用作参数
        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包
```

(下页继续)

(续上页)

```
    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'])
]
test_pipeline = [ # 测试流程
    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=(1024, 1024),  # 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='RResize'),  # 使用改变图像大小的数据增广
            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='DefaultFormatBundle'),  # 用于在管道中收集数据的默认格式包
            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='DOTADataset',  # 数据集的类别
        ann_file=
        '../datasets/split_1024_dota1_0/trainval/annfiles/',  # 注释文件路径
        img_prefix=
        '../datasets/split_1024_dota1_0/trainval/images/',  # 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),
            dict(type='RResize', img_scale=(1024, 1024)),
            dict(type='RRandomFlip', flip_ratio=0.5, version='oc'),
            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),
```

(下页继续)

(续上页)

```
            dict(type='DefaultFormatBundle'),
            dict(type='Collect', keys=['img', 'gt_bboxes', 'gt_labels'])
        ],
        version='oc'),
    val=dict(  # 验证数据集的配置
        type='DOTADataset',
        ann_file=
        '../datasets/split_1024_dota1_0/trainval/annfiles/',
        img_prefix=
        '../datasets/split_1024_dota1_0/trainval/images/',
        pipeline=[
            dict(type='LoadImageFromFile'),
            dict(
                type='MultiScaleFlipAug',
                img_scale=(1024, 1024),
                flip=False,
                transforms=[
                    dict(type='RResize'),
                    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'])
                ])
        ],
        version='oc'),
    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='DOTADataset',
        ann_file=
        '../datasets/split_1024_dota1_0/test/images/',
        img_prefix=
        '../datasets/split_1024_dota1_0/test/images/',
        pipeline=[  # 由之前创建的 test_pipeline 传递的流程
            dict(type='LoadImageFromFile'),
            dict(
                type='MultiScaleFlipAug',
                img_scale=(1024, 1024),
                flip=False,
                transforms=[
                    dict(type='RResize'),
```

(下页继续)

(续上页)

```
                    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'])
                ])
        ],
        version='oc'))
evaluation = dict(  # evaluation hook 的配置
    interval=12,  # 验证的间隔
    metric='mAP')  # 验证期间使用的指标
optimizer = dict(  # 用于构建优化器的配置文件
    type='SGD',  # 优化器类型
    lr=0.0025,  # 优化器的学习率
    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
    grad_clip=dict(
        max_norm=35,
        norm_type=2))
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
    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=0.3333333333333333,  # 用于预热的起始学习率的比率
    step=[8, 11])  # 衰减学习率的起止回合数
runner = dict(
    type='EpochBasedRunner',  # 将使用的 runner 的类别 (例如 IterBasedRunner 或␣
↪EpochBasedRunner)
    max_epochs=12) # runner 总回合 (epoch) 数， 对于 IterBasedRunner 使用 `max_iters`
checkpoint_config = dict(  # checkpoint hook 的配置文件
    interval=12)  # 保存的间隔是 12
log_config = dict(  # register logger hook 的配置文件
    interval=50,  # 打印日志的间隔
    hooks=[
        # dict(type='TensorboardLoggerHook')  # 同样支持 Tensorboard 日志
        dict(type='TextLoggerHook')
    ])  # 用于记录训练过程的记录器 (logger)
dist_params = dict(backend='nccl')  # 用于设置分布式训练的参数，端口也同样可被设置
log_level = 'INFO'  # 日志的级别
```

(下页继续)

(续上页)

```
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 (epoch)。
work_dir = './work_dirs/rotated_retinanet_hbb_r50_fpn_1x_dota_oc'  # 用于保存当前实验的模型检查点 (checkpoints) 和日志的目录
```

8.4 常见问题 (FAQ)

8.4.1 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。值得注意的是，在修改子配置中的中间变量时，需要再次将中间变量传递到相应的字段中。例如，我们想使用离线多尺度策略 (multi scale strategy) 来训练 RoI-Trans。train_pipeline 是我们想要修改的中间变量。

```
_base_ = ['./roi_trans_r50_fpn_1x_dota_le90.py']

data_root = '../datasets/split_ms_dota1_0/'
angle_version = 'le90'
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='RResize', img_scale=(1024, 1024)),
    dict(
        type='RRandomFlip',
        flip_ratio=[0.25, 0.25, 0.25],
        direction=['horizontal', 'vertical', 'diagonal'],
        version=angle_version),
    dict(
        type='PolyRandomRotate',
        rotate_ratio=0.5,
        angles_range=180,
        auto_bound=False,
        version=angle_version),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='Pad', size_divisor=32),
    dict(type='DefaultFormatBundle'),
    dict(type='Collect', keys=['img', 'gt_bboxes', 'gt_labels'])
```

(下页继续)

(续上页)

```
]
data = dict(
    train=dict(
        pipeline=train_pipeline,
        ann_file=data_root + 'trainval/annfiles/',
        img_prefix=data_root + 'trainval/images/'),
    val=dict(
        ann_file=data_root + 'trainval/annfiles/',
        img_prefix=data_root + 'trainval/images/'),
    test=dict(
        ann_file=data_root + 'test/images/',
        img_prefix=data_root + 'test/images/'))
```

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_ = './roi_trans_r50_fpn_1x_dota_le90.py'
norm_cfg = dict(type='BN', requires_grad=True)
model = dict(
    backbone=dict(norm_cfg=norm_cfg),
    neck=dict(norm_cfg=norm_cfg),
    ...)
```

CHAPTER 9

教程 2：自定义数据集

9.1 支持新的数据格式

要支持新的数据格式，您可以将它们转换为现有的格式（DOTA 格式）。您可以选择离线（在通过脚本训练之前）或在线（实施新数据集并在训练时进行转换）进行转换。在 MMRotate 中，我们建议将数据转换为 DOTA 格式并离线进行转换，如此您只需在数据转换后修改 config 的数据标注路径和类别即可。

9.1.1 将新数据格式重构为现有格式

最简单的方法是将数据集转换为现有数据集格式 (DOTA) 。

DOTA 格式的注解 txt 文件：

```
184 2875 193 2923 146 2932 137 2885 plane 0
66 2095 75 2142 21 2154 11 2107 plane 0
...
```

每行代表一个对象，并将其记录为一个 10 维数组 `A` 。

- `A[0:8]`: 多边形的格式 `(x1, y1, x2, y2, x3, y3, x4, y4)` 。
- `A[8]`: 类别
- `A[9]`: 困难

在数据预处理之后，用户可以通过两个步骤来训练具有现有格式（例如 DOTA 格式）的自定义新数据集：

1. 修改配置文件以使用自定义数据集。

2. 检查自定义数据集的标注。

下面给出两个例子展示上述两个步骤，它使用一个自定义的 5 类 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. 修改配置文件以使用自定义数据集

配置文件的修改主要涉及两个方面:

1. `data` 部分。具体来说，您需要在 `data.train`, `data.val` 和 `data.test` 中显式添加 classes 字段。
2. `data` 属性变量。具体来说，特别是您需要在 `data.train`, `data.val` 和 `data.test` 中添加 classes 字段。
3. `model` 部分中的 `num_classes` 属性变量。特别是将所有 num_classes 的默认值（例如 COCO 中的 80）覆盖到您的类别编号中。

在 `configs/my_custom_config.py`：

```
# 新配置继承了基础配置用于突出显示必要的修改
_base_ = './rotated_retinanet_hbb_r50_fpn_1x_dota_oc'

# 1. 数据集的设置
dataset_type = 'DOTADataset'
classes = ('a', 'b', 'c', 'd', 'e')
data = dict(
    samples_per_gpu=2,
    workers_per_gpu=2,
    train=dict(
        type=dataset_type,

        # 注意将你的类名添加到字段 `classes`
        classes=classes,
        ann_file='path/to/your/train/annotation_data',
        img_prefix='path/to/your/train/image_data'),
    val=dict(
        type=dataset_type,

        # 注意将你的类名添加到字段 `classes`
        classes=classes,
        ann_file='path/to/your/val/annotation_data',
        img_prefix='path/to/your/val/image_data'),
    test=dict(
        type=dataset_type,
```

(下页继续)

(续上页)

```
        # 注意将你的类名添加到字段 `classes`
        classes=classes,
        ann_file='path/to/your/test/annotation_data',
        img_prefix='path/to/your/test/image_data'))

# 2. 模型设置
model = dict(
    bbox_head=dict(
        type='RotatedRetinaHead',
        # 显式将所有 `num_classes` 字段从 15 重写为 5。。
        num_classes=15))
```

2. 查看自定义数据集的标注

假设您的自定义数据集是 DOTA 格式，请确保您在自定义数据集中具有正确的标注：

- 配置文件中的 `classes` 字段应该与 txt 标注的 `A[8]` 保持完全相同的元素和相同的顺序。MMRotate 会自动的将 `categories` 中不连续的 `id` 映射到连续的标签索引中，所以在 `categories` 中 `name` 的字符串顺序会影响标签索引的顺序。同时，配置文件中 `classes` 的字符串顺序也会影响预测边界框可视化过程中的标签文本信息。

9.2 通过封装器自定义数据集

MMRotate 还支持许多数据集封装器对数据集进行混合或修改数据集的分布以进行训练。目前它支持三个数据集封装器，如下所示：

- `RepeatDataset`: 简单地重复整个数据集。
- `ClassBalancedDataset`: 以类平衡的方式重复数据集。
- `ConcatDataset`: 拼接数据集。

9.2.1 重复数据集

我们使用 `RepeatDataset` 作为封装器来重复这个数据集。例如，假设原始数据集是 `Dataset_A`，我们就重复一遍这个数据集。配置信息如下所示：

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

(下页继续)

(续上页)

```
            ...
            pipeline=train_pipeline
        )
    )
```

9.2.2 类别平衡数据集

我们使用 `ClassBalancedDataset` 作为封装器，根据类别频率重复数据集。这个数据集的重复操作 `ClassBalancedDataset` 需要实例化函数 `self.get_cat_ids(idx)` 的支持。例如，`Dataset_A` 需要使用 `oversample_thr=1e-3`，配置信息如下所示：

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

9.2.3 拼接数据集

这里用三种方式对数据集进行拼接。

1. 如果要拼接的数据集属于同一类型且具有不同的标注文件，则可以通过如下所示的配置信息来拼接数据集：

```
dataset_A_train = dict(
    type='Dataset_A',
    ann_file = ['anno_file_1', 'anno_file_2'],
    pipeline=train_pipeline
)
```如果拼接后的数据集用于测试或评估，我们这种方式是可以支持对每个数据集分别进行评估。如果要测试的整个拼接数据集，如下所示您可以直接通过设置 separate_eval=False 来实现。

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

(下页继续)

(续上页)

```
    pipeline=train_pipeline
)
```

2. 如果您要拼接不同的数据集，您可以通过如下所示的方法对拼接数据集配置信息进行设置。

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

```如果拼接后的数据集用于测试或评估，这种方式还支持对每个数据集分别进行评估。

3. 我们也支持如下所示的方法对 ConcatDataset 进行明确的定义。

```
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))
```

```这种方式允许用户通过设置 `separate_eval=False` 将所有数据集转为单个数据集进行评估。

**笔记:**

1. 假设数据集在评估期间使用 self.data_infos，就要把选项设置为 separate_eval=False。因为 COCO 数据集不完全依赖 self.data_infos 进行评估，所以 COCO 数据集并不支持这种设置操作。没有在组合不同类型的数据集并对其进行整体评估的场景进行测试，因此我们不建议使用这样的操作。
2. 不支持评估 ClassBalancedDataset 和 RepeatDataset，所以也不支持评估这些类型的串联组合后的数据集。

一个更复杂的例子，分别将 Dataset_A 和 Dataset_B 重复 N 次和 M 次，然后将重复的数据集连接起来，如下所示。

```
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
)
```

CHAPTER 10

教程 3: 自定义模型

我们大致将模型组件分为了 5 种类型。

- 主干网络 (Backbone): 通常是一个全卷积网络 (FCN)，用来提取特征图，比如残差网络 (ResNet)。也可以是基于视觉 Transformer 的网络，比如 Swin Transformer 等。
- Neck: 主干网络和任务头 (Head) 之间的连接组件，比如 FPN, ReFPN。
- 任务头 (Head): 用于某种具体任务（比如边界框预测）的组件。
- 区域特征提取器 (Roi Extractor): 用于从特征图上提取区域特征的组件，比如 RoI Align Rotated。
- 损失 (loss): 任务头上用于计算损失函数的组件，比如 FocalLoss, GWDLoss, and KFIoULoss。

10.1 开发新的组件

10.1.1 添加新的主干网络

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）

新建文件 mmrotate/models/backbones/mobilenet.py。

```
import torch.nn as nn

from mmrotate.models.builder import ROTATED_BACKBONES


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

    def __init__(self, arg1, arg2):
        pass

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

2. 导入模块

你可以将下面的代码添加到 mmrotate/models/backbones/__init__.py 中：

```
from .mobilenet import MobileNet
```

或者添加如下代码

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

到配置文件中以避免修改原始代码。

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),
    ...
```

10.1.2 添加新的 Neck

1. 定义一个 Neck（以 PAFPN 为例）

新建文件 mmrotate/models/necks/pafpn.py。

```
from mmrotate.models.builder import ROTATED_NECKS

@ROTATED_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 mmrotate/models/necks/__init__.py 中

```
from .pafpn import PAFPN
```

或者添加

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

到配置文件中以避免修改原始代码。

3. 修改配置文件

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

10.1.3 添加新的 Head

这里，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 `mmrotate/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 mmrotate.models.builder import ROTATED_HEADS
from mmrotate.models.roi_heads.bbox_heads.bbox_head import BBoxHead

@ROTATED_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)
```

(下页继续)

(续上页)

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

然后，如有必要，我们需要实现一个新的 RoI Head。我们打算从 StandardRoIHead 继承出新的 DoubleHeadRoIHead。我们发现 StandardRoIHead 已经实现了下述函数。

```
import torch

from mmdet.core import bbox2result, bbox2roi, build_assigner, build_sampler
from mmrotate.models.builder import ROTATED_HEADS, build_head, build_roi_extractor
from mmrotate.models.roi_heads.base_roi_head import BaseRoIHead
from mmrotate.models.roi_heads.test_mixins import BBoxTestMixin, MaskTestMixin


@ROTATED_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 forward_dummy(self, x, proposals):


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

    def _bbox_forward(self, x, rois):

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

    def simple_test(self,
                    x,
                    proposal_list,
```

(下页继续)

(续上页)

```
                    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 mmrotate/models/roi_heads/double_roi_head.py 中, 我们实现如下的新的 RoI Head:

```
from mmrotate.models.builder import ROTATED_HEADS
from mmrotate.models.roi_heads.standard_roi_head import StandardRoIHead


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

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

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

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

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

最后，用户需要把这个新模块添加到 mmrotate/models/bbox_heads/__init__.py 以及 mmrotate/models/roi_heads/__init__.py 中。这样，注册机制就能找到并加载它们。

另外，用户也可以添加

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

到配置文件中来实现同样的目的。

10.1.4 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 mmrotate/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 mmrotate.models.builder import ROTATED_LOSSES
from mmdet.models.losses.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

@ROTATED_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)
```

(下页继续)

(续上页)

```
        return loss_bbox
```

然后，用户需要把下面的代码加到 mmrotate/models/losses/__init__.py 中。

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

或者，你可以添加：

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

到配置文件来实现相同的目的。

因为 MyLoss 是用于回归的，你需要在 Head 中修改 loss_bbox 字段：

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

CHAPTER 11

教程 4: 自定义训练设置

11.1 自定义优化设置

11.1.1 自定义 Pytorch 支持的优化器

我们已经支持了全部 Pytorch 自带的优化器，唯一需要修改的就是配置文件中 `optimizer` 部分。例如，如果您想使用 `ADAM` (注意如下操作可能会让模型表现下降)，可以使用如下修改：

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

为了修改模型训练的学习率，使用者仅需修改配置文件里 `optimizer` 的 `lr` 即可。使用者可以参考 PyTorch 的 API doc 直接设置参数。

11.1.2 自定义用户自己实现的优化器

1. 定义一个新的优化器

一个自定义的优化器可以这样定义：

假如您想增加一个叫做 `MyOptimizer` 的优化器，它的参数分别有 `a`, `b`, 和 `c`。您需要创建一个名为 `mmrotate/core/optimizer` 的新文件夹；然后参考如下代码段在 `mmrotate/core/optimizer/my_optimizer.py` 文件中实现新的优化器:

```
from mmdet.core.optimizer.registry import OPTIMIZERS
from torch.optim import Optimizer


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

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

2. 增加优化器到注册表 (registry)

为了能够使得上述添加的模块被 mmrotate 发现，需要先将该模块添加到主命名空间（main namespace）。

- 修改 mmrotate/core/optimizer/__init__.py 文件来导入该模块。

  新的被定义的模块应该被导入到 mmrotate/core/optimizer/__init__.py 中，这样注册表才会发现新的模块并添加它：

```
from .my_optimizer import MyOptimizer
```

- 在配置文件中使用 custom_imports 来手动添加该模块

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

mmrotate.core.optimizer.my_optimizer 模块将会在程序开始被导入，并且 MyOptimizer 类将会自动注册。需要注意只有包含 MyOptimizer 类的包 (package) 应当被导入。而 mmrotate.core.optimizer.my_optimizer.MyOptimizer **不能**被直接导入。

事实上，在这种导入方式下用户可以用完全不同的文件夹结构，只要这一模块的根目录已经被添加到 PYTHONPATH 里面。

3. 在配置文件中指定优化器

之后您可以在配置文件的 optimizer 部分里面使用 MyOptimizer。在配置文件里，优化器按照如下形式被定义在 optimizer 部分里：

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

要使用用户自定义的优化器，这部分应该改成：

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

11.1.3 自定义优化器的构造函数 (constructor)

有些模型的优化器可能有一些特别参数配置，例如批归一化层 (BatchNorm layers) 的权重衰减系数 (weight decay)。用户可以通过自定义优化器的构造函数去微调这些细粒度参数。

```
from mmcv.utils import build_from_cfg

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


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

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

    def __call__(self, model):

        return my_optimizer
```

mmcv 默认的优化器构造函数实现可以参考 这里 ，这也可以作为新的优化器构造函数的模板。

11.1.4 其他配置

优化器未实现的技巧应该通过修改优化器构造函数（如设置基于参数的学习率）或者钩子（hooks）去实现。我们列出一些常见的设置，它们可以稳定或加速模型的训练。如果您有更多的设置，欢迎在 PR 和 issue 里面提出。

- **使用梯度裁剪 (gradient clip) 来稳定训练**: 一些模型需要梯度裁剪来稳定训练过程。使用方式如下：

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

  如果您的配置继承了已经设置了 optimizer_config 的基础配置（base config），你可能需要设置 _delete_=True 来覆盖不必要的配置参数。请参考 配置文档 了解更多细节。

- **使用动量调度加速模型收敛**: 我们支持动量规划器（Momentum scheduler），以实现根据学习率调节模型优化过程中的动量设置，这可以使模型以更快速度收敛。动量规划器经常与学习率规划器（LR scheduler）一起使用，例如下面的配置经常被用于 3D 检测模型训练中以加速收敛。更多细节请参考 CyclicLrUpdater 和 CyclicMomentumUpdater。

  ```
  lr_config = dict(
      policy='cyclic',
  ```

(下页继续)

(续上页)

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

11.2 自定义训练计划

默认地，我们使用 1x 计划（1x schedule）的步进学习率（step learning rate），这在 MMCV 中被称为 `StepLRHook`。我们支持很多其他的学习率规划器，参考 这里 ，例如 `CosineAnnealing` 和 `Poly`。下面是一些例子：

- `Poly`：

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

- `ConsineAnnealing`：

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

11.3 自定义工作流 (workflow)

工作流是一个专门定义运行顺序和轮数（epochs）的列表。默认情况下它设置成：

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

这是指训练 1 个 epoch。有时候用户可能想检查一些模型在验证集上的指标，如损失函数值（Loss）和准确性（Accuracy）。在这种情况下，我们可以将工作流设置为：

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

这样以来，1 个 epoch 训练，1 个 epoch 验证将交替运行。

注意:

1. 模型参数在验证的阶段不会被自动更新。
2. 配置文件里的键值 `total_epochs` 仅控制训练的 epochs 数目，而不会影响验证工作流。
3. 工作流 `[('train', 1), ('val', 1)]` 和 `[('train', 1)]` 将不会改变 `EvalHook` 的行为，因为 `EvalHook` 被 `after_train_epoch` 调用而且验证的工作流仅仅影响通过调用 `after_val_epoch` 的钩子 (hooks)。因此，`[('train', 1), ('val', 1)]` 和 `[('train', 1)]` 的区别仅在于 runner 将在每次训练阶段（training epoch）结束后计算在验证集上的损失。

11.4 自定义钩 (hooks)

11.4.1 自定义用户自己实现的钩子（hooks）

1. 实现一个新的钩子（hook）

在某些情况下，用户可能需要实现一个新的钩子。MMRotate 支持训练中的自定义钩子。因此，用户可以直接在 mmrotate 或其基于 mmdet 的代码库中实现钩子，并通过仅在训练中修改配置来使用钩子。这里我们举一个例子：在 mmrotate 中创建一个新的钩子并在训练中使用它。

```
from mmcv.runner import HOOKS, Hook


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

    def __init__(self, a, b):
        pass

    def before_run(self, runner):
        pass

    def after_run(self, runner):
        pass

    def before_epoch(self, runner):
        pass

    def after_epoch(self, runner):
        pass

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

(下页继续)

(续上页)

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

用户需要根据钩子的功能指定钩子在训练各阶段中（before_run，after_run，before_epoch，after_epoch，before_iter，after_iter）做什么。

2. 注册新的钩子（hook）

接下来我们需要导入 MyHook。如果文件的路径是 mmrotate/core/utils/my_hook.py，有两种方式导入：

- 修改 mmrotate/core/utils/__init__.py 文件来导入

  新定义的模块需要在 mmrotate/core/utils/__init__.py 导入，注册表才会发现并添加该模块：

```
from .my_hook import MyHook
```

- 在配置文件中使用 custom_imports 来手动导入

```
custom_imports = dict(imports=['mmrotate.core.utils.my_hook'], allow_failed_
↪imports=False)
```

3. 修改配置

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

您也可以通过配置键值 priority 为 'NORMAL' 或 'HIGHEST' 来设置钩子的优先级：

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

默认地，钩子的优先级在注册时被设置为 NORMAL。

11.4.2 使用 MMCV 中实现的钩子 (hooks)

如果钩子已经在 MMCV 里实现了，您可以直接修改配置文件来使用钩子。

4. 示例: `NumClassCheckHook`

我们实现了一个自定义的钩子 NumClassCheckHook，用来检验 head 中的 `num_classes` 是否与 `dataset` 中的 `CLASSSES` 长度匹配。

我们在 default_runtime.py 中对其进行设置。

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

11.4.3 修改默认运行挂钩

有一些常见的钩子并不通过 `custom_hooks` 注册，这些钩子包括：

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

这些钩子中，只有记录器钩子（logger hook）是 `VERY_LOW` 优先级，其他钩子的优先级为 `NORMAL`。前面提到的教程已经介绍了如何修改 `optimizer_config`，`momentum_config` 以及 `lr_config`。这里我们介绍一下如何处理 `log_config`，`checkpoint_config` 以及 `evaluation`。

Checkpoint config

MMCV runner 将使用 `checkpoint_config` 来初始化 `CheckpointHook`。

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

用户可以设置 `max_keep_ckpts` 来仅保存一小部分检查点（checkpoint）或者通过设置 `save_optimizer` 来决定是否保存优化器的状态字典 (state dict of optimizer)。更多使用参数的细节请参考 这里。

Log config

log_config 包裹了许多日志钩 (logger hooks) 而且能去设置间隔 (intervals)。现在 MMCV 支持 WandbLoggerHook，MlflowLoggerHook 和 TensorboardLoggerHook。详细的使用请参照 文档。

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

Evaluation config

evaluation 的配置文件将被用来初始化 EvalHook。除了 interval 键，其他的像 metric 这样的参数将被传递给 dataset.evaluate()。

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

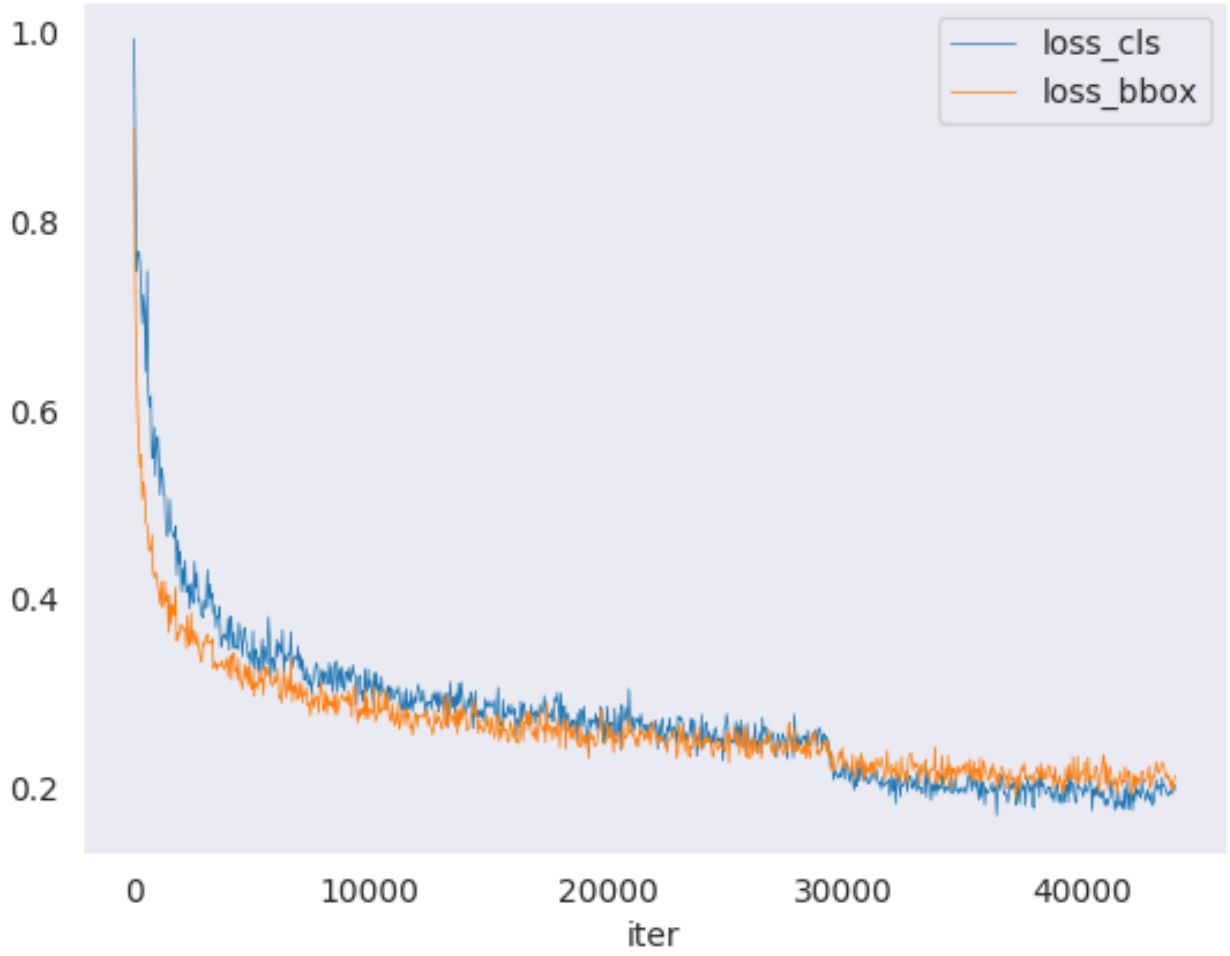
除了训练和测试脚本，我们在 tools/ 文件夹内还提供了一些有用的工具。

CHAPTER 12

日志分析

tools/analysis_tools/analyze_logs.py 通过给定的日志文件绘制 loss/mAP 曲线。需首先执行 pip install seaborn 安装依赖。

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



示例:

- 绘制某次执行的分类损失

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

- 绘制某次执行的分类和回归损失，同时将图像保存到 pdf 文件

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

- 在同一张图像中比较两次执行的 mAP

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

- 计算平均训练速度

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

预计输出如下

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

CHAPTER 13

可视化

13.1 可视化数据集

tools/misc/browse_dataset.py 帮助用户浏览检测的数据集（包括图像和检测框的标注），或将图像保存到指定目录。

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

CHAPTER 14

模型部署

为了使用 TorchServe 部署一个 MMRotate 模型，需要进行以下几步：

14.1 1. 转换 MMRotate 模型至 TorchServe

```
python tools/deployment/mmrotate2torchserve.py ${CONFIG_FILE} ${CHECKPOINT_FILE} \
--output-folder ${MODEL_STORE} \
--model-name ${MODEL_NAME}
```

示例:

```
wget -P checkpoint  \
https://download.openmmlab.com/mmrotate/v0.1.0/rotated_faster_rcnn/rotated_faster_
↪rcnn_r50_fpn_1x_dota_le90/rotated_faster_rcnn_r50_fpn_1x_dota_le90-0393aa5c.pth

python tools/deployment/mmrotate2torchserve.py configs/rotated_faster_rcnn/rotated_
↪faster_rcnn_r50_fpn_1x_dota_le90.py checkpoint/rotated_faster_rcnn_r50_fpn_1x_dota_
↪le90-0393aa5c.pth \
--output-folder ${MODEL_STORE} \
--model-name rotated_faster_rcnn
```

Note: ${MODEL_STORE} 需要是一个文件夹的绝对路径。

14.2 2. 构建 mmrotate-serve docker 镜像

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

14.3 3. 运行 mmrotate-serve 镜像

请参考官方文档 基于 docker 运行 TorchServe.

为了使镜像能够使用 GPU 资源，需要安装 nvidia-docker。之后可以传递 --gpus 参数以在 GPU 上运行。

示例：

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

参考 该文档 了解关于推理 (8080)，管理 (8081) 和指标 (8082) 等 API 的信息。

14.4 4. 测试部署

```
curl -O https://raw.githubusercontent.com/open-mmlab/mmrotate/main/demo/demo.jpg
curl http://127.0.0.1:8080/predictions/${MODEL_NAME} -T demo.jpg
```

您应该获得类似于以下内容的响应：

```
[
  {
    "class_name": "small-vehicle",
    "bbox": [
      584.9473266601562,
      327.2749938964844,
      38.45665740966797,
      16.898427963256836,
      -0.7229751944541931
    ],
    "score": 0.9766026139259338
  },
  {
```

(下页继续)

(续上页)


```
    "class_name": "small-vehicle",
    "bbox": [
      152.0239715576172,
      305.92572021484375,
      43.144744873046875,
      18.85024642944336,
      0.014928221702575684
    ],
    "score": 0.972826361656189
  },
  {
    "class_name": "large-vehicle",
    "bbox": [
      160.58056640625,
      437.3690185546875,
      55.6795654296875,
      19.31710433959961,
      0.007036328315734863
    ],
    "score": 0.888836681842804
  },
  {
    "class_name": "large-vehicle",
    "bbox": [
      666.2868041992188,
      1011.3961181640625,
      60.396209716796875,
      21.821645736694336,
      0.8549195528030396
    ],
    "score": 0.8240180015563965
  }
]
```


另外，你也可以使用 `test_torchserver.py` 来比较 TorchServe 和 PyTorch 的结果，并进行可视化。

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

示例：

```
python tools/deployment/test_torchserver.py \
demo/demo.jpg \
```

(下页继续)

(续上页)

```
configs/rotated_faster_rcnn/rotated_faster_rcnn_r50_fpn_1x_dota_le90.py \
rotated_faster_rcnn_r50_fpn_1x_dota_le90-0393aa5c.pth \
rotated_fater_rcnn
```

CHAPTER 15

模型复杂度

tools/analysis_tools/get_flops.py 是改编自 flops-counter.pytorch 的脚本，用于计算给定模型的 FLOPs 和参数量。

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

预计输出如下

```
==============================
Input shape: (3, 1024, 1024)
Flops: 215.92 GFLOPs
Params: 36.42 M
==============================
```

注意: 此工具仍处于实验阶段，我们并不能保证计算结果是绝对正确的。你可以将结果用于简单的比较，但在技术报告或论文中采用之前请仔细检查

1. FLOPs 与输入大小相关，但参数量与其无关。默认输入大小是 (1, 3, 1024, 1024)。
2. 一些算子例如 DCN 或自定义算子并未包含在 FLOPs 计算中，所以 S2A-Net 和基于 RepPoints 的模型的 FLOPs 计算是错误的。详细信息请查看 `mmcv.cnn.get_model_complexity_info()`。
3. 两阶段检测器的 FLOPs 取决于候选的数量。

15.1 准备发布模型

tools/model_converters/publish_model.py 帮助用户准备他们将发布的模型。

在将模型上传到 AWS 之前，你可能需要

1. 将模型权重转换至 CPU
2. 删除优化器的状态
3. 计算权重文件的哈希值并附加到文件名后

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

例如,

```
python tools/model_converters/publish_model.py work_dirs/rotated_faster_rcnn/latest.
↪pth rotated_faster_rcnn_r50_fpn_1x_dota_le90_20190801.pth
```

最终输出的文件名是 rotated_faster_rcnn_r50_fpn_1x_dota_le90_20190801-{hash id}.pth。

CHAPTER 16

基准测试

16.1 FPS 基准

`tools/analysis_tools/benchmark.py` 帮助用户计算 FPS。FPS 值包括模型前向传播和后处理。为了得到更准确的数值，目前只支持单 GPU 分布式启动。

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

示例: 假设你已经下载了 `Rotated Faster R-CNN` 模型权重到 `checkpoints/` 文件夹

```
python -m torch.distributed.launch --nproc_per_node=1 --master_port=29500 tools/
↪analysis_tools/benchmark.py \
       configs/rotated_faster_rcnn/rotated_faster_rcnn_r50_fpn_1x_dota_le90.py \
       checkpoints/rotated_faster_rcnn_r50_fpn_1x_dota_le90-0393aa5c.pth \
       --launcher pytorch
```

CHAPTER 17

杂项

17.1 打印完整配置文件

tools/misc/print_config.py 输出整个配置文件并整合其所有导入。

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

CHAPTER 18

混淆矩阵

混淆矩阵是预测结果的概要

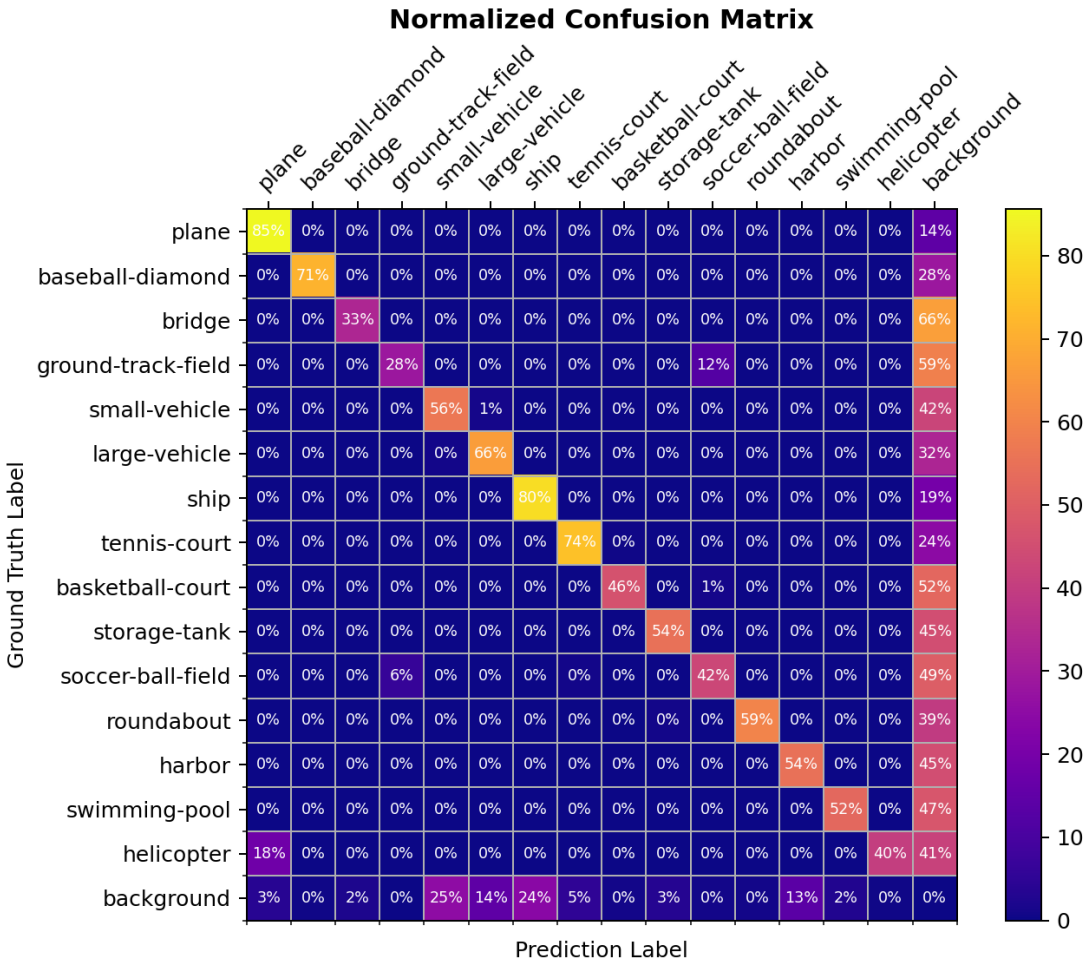
`tools/analysis_tools/confusion_matrix.py` 可以分析预测结果并绘制混淆矩阵。

首先执行 `tools/test.py` 将检测结果保存为 `.pkl` 文件。

之后执行

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

你会得到一个类似于下图的混淆矩阵:


Normalized Confusion Matrix
plane
baseball-diamond
bridge
ground-track-field
small-vehicle
large-vehicle
ship
tennis-court
basketball-court
storage-tank
soccer-ball-field
roundabout
harbor
swimming-pool
helicopter
background
Ground Truth Label
Prediction Label
80
70
60
50
40
30
20
10
0

CHAPTER 19

Changelog

CHAPTER 20

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。

20.1 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.”

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

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

  原因是安装了 `mmcv` 而不是 `mmcv-full`。

  1. 使用 `pip uninstall mmcv` 卸载。
  2. 根据 安装说明安装 `mmcv-full`。

20.2 PyTorch/CUDA 环境相关

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

  1. 常见报错信息为 `nvcc fatal: Unsupported gpu architecture 'compute_86'` 意思是你的编译器应该为 sm_86 进行优化，例如，英伟达 30 系列的显卡，但这样的优化 CUDA toolkit 11.0 并不支持。此解决方案通过添加 `MMCV_WITH_OPS=1 MMCV_CUDA_ARGS='-gencode=arch=compute_80,code=sm_80' pip install -e .` 来修改编译标志，这告诉编译器 `nvcc` 为 **sm_80** 进行优化，例如 Nvidia A100，尽管 A100 不同于 30 系列的显卡，但他们使用相似的图灵架构。这种解决方案可能会丧失一些性能但的确有效。

  2. PyTorch 开发者已经在 pytorch/pytorch#47585 更新了 PyTorch 默认的编译标志，所以使用 PyTorch-nightly 可能也能解决这个问题，但是我们对此并没有验证这种方式是否有效。

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

  1. 检查您的 cuda 运行时版本 (一般在 `/usr/local/`)、指令 `nvcc --version` 显示的版本以及 `conda list cudatoolkit` 指令显式的版本是否匹配。

  2. 通过运行 `python mmdet/utils/collect_env.py` 来检查是否为当前的 GPU 架构编译了正确的 PyTorch、torchvision 和 MMCV，你可能需要设置 `TORCH_CUDA_ARCH_LIST` 来重新安装 MMCV。可以参考 GPU 架构表，即通过运行 `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，例如，Tesla K80。

  3. 检查运行环境是否与 mmcv/mmdet 编译时相同，例如，您可能使用 CUDA 10.0 编译 MMCV，但在 CUDA 9.0 环境中运行它。

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

  1. 如果这些 symbols 属于 CUDA/C++ (例如，libcudart.so 或者 GLIBCXX)，检查 CUDA/GCC 运行时环境是否与编译 MMCV 的一致。例如使用 `python mmdet/utils/collect_env.py` 检查 `"MMCV Compiler"/"MMCV CUDA Compiler"` 是否和 `"GCC"/"CUDA_HOME"` 一致。

  2. 如果这些 symbols 属于 PyTorch，(例如，symbols containing caffe、aten 和 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。您需要先手动安装 Cpython 然后运命令 `pip install -r requirements.txt`。

  2. 检查环境中的 `setuptools`、`Cython` 和 `PyTorch` 相互之间版本是否匹配。

- “Segmentation fault”.

1. 检查 GCC 的版本并使用 GCC 5.4，通常是因为 PyTorch 版本与 GCC 版本不匹配（例如，对于 Pytorch 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. 检查是是否 PyTorch 被正确的安装并可以使用 CUDA 算子，例如在终端中键入如下的指令。

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

   并判断是否是否返回 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'
```

   如果 MMCV 被正确的安装了，那么上面的两条指令不会有问题。

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`。

20.3 E2CNN

- “ImportError: cannot import name ‘container_bacs’ from ‘torch._six’”

  1. 这是因为 `container_abcs` 在 PyTorch 1.9 之后被移除.

  2. 将文件 `python3.7/site-packages/e2cnn/nn/modules/module_list.py` 中的

```
from torch.six import container_abcs
```

     替换成

```
TORCH_MAJOR = int(torch.__version__.split('.')[0])
TORCH_MINOR = int(torch.__version__.split('.')[1])
if TORCH_MAJOR ==1 and TORCH_MINOR < 8:
    from torch.six import container_abcs
else:
    import collections.abs as container_abcs
```

  3. 或者降低 Pytorch 的版本。

20.4 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. 通过在配置文件中设置 `fp16 = dict(loss_scale='dynamic')` 来尝试混合精度训练。

- “RuntimeError: Expected to have finished reduction in the prior iteration before starting a new one”

  1. 错误表明，您的模块有没用于产生损失的参数，这种现象可能是由于在 DDP 模式下运行代码中的不同分支造成的。
  2. 您可以在配置中设置 `find_unused_parameters = True` 来解决上述问题，或者手动查找那些未使用的参数。

20.5 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物体即 GT，AP 与 AR 将被设置为 -1。

CHAPTER 21

English

CHAPTER 22

简体中文

CHAPTER 23

mmrotate.apis

mmrotate.apis.**inference_detector_by_patches**(*model*, *img*, *sizes*, *steps*, *ratios*, *merge_iou_thr*, *bs=1*)

inference patches with the detector.

Split huge image(s) into patches and inference them with the detector. Finally, merge patch results on one huge image by nms.

参数

- **model** (*nn.Module*) –The loaded detector.
- **img** (*str | ndarray or*) –Either an image file or loaded image.
- **sizes** (*list*) –The sizes of patches.
- **steps** (*list*) –The steps between two patches.
- **ratios** (*list*) –Image resizing ratios for multi-scale detecting.
- **merge_iou_thr** (*float*) –IoU threshold for merging results.
- **bs** (*int*) –Batch size, must greater than or equal to 1.

返回 Detection results.

返回类型 list[np.ndarray]

CHAPTER 24

mmrotate.core

24.1 anchor

class mmrotate.core.anchor.**PseudoAnchorGenerator**(*strides*)

Non-Standard pseudo anchor generator that is used to generate valid flags only!

property num_base_anchors

total number of base anchors in a feature grid

Type list[int]

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

Calling its grid_anchors() method will raise NotImplementedError!

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

Fake rotate anchor generator for 2D anchor-based detectors.

Horizontal bounding box represented by (x,y,w,h,theta).

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

Generate grid anchors of a single level.

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

参数

- **featmap_size** (*tuple[int]*) –Size of the feature maps.
- **level_idx** (*int*) –The index of corresponding feature map level.
- **(obj** (*dtype*) –*torch.dtype*): Date type of points.Defaults to
- **torch.float32.** –
- **device** (*str, optional*) –The device the tensor will be put on.
- **to 'cuda'.** (*Defaults*) –

返回 Anchors in the overall feature maps.

返回类型 torch.Tensor

mmrotate.core.anchor.**rotated_anchor_inside_flags**(*flat_anchors*, *valid_flags*, *img_shape*, *allowed_border=0*)

Check whether the rotated anchors are inside the border.

参数

- **flat_anchors** (*torch.Tensor*) –Flatten anchors, shape (n, 5).
- **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

24.2 bbox

class mmrotate.core.bbox.**ATSSKldAssigner**(*topk*, *use_reassign=False*)

Assign a corresponding gt bbox or background to each bbox.

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

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

参数

- **topk** (*float*) –Number of bbox selected in each level.
- **use_reassign** (*bool, optional*) –If true, it is used to reassign samples.

AspectRatio(*gt_rbboxes*)

compute the aspect ratio of all gts.

参数 **gt_rbboxes** (*torch.Tensor*) –Groundtruth polygons, shape (k, 8).

返回 The aspect ratio of gt_rbboxes, shape (k, 1).

返回类型 ratios (torch.Tensor)

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

Assign gt to bboxes.

The assignment is done in following steps

1. compute iou between all bbox (bbox of all pyramid levels) and gt
2. compute center distance between all bbox and gt
3. on each pyramid level, for each gt, select k bbox whose center are closest to the gt center, so we total select k*l bbox as candidates for each gt
4. get corresponding iou for the these candidates, and compute the mean and std, set mean + std as the iou threshold
5. compute the mean aspect ratio of all gts, and set exp((-mean aspect ratio / 4) * (mean + std) as the iou threshold
6. select these candidates whose iou are greater than or equal to the threshold as positive
7. limit the positive sample’s center in gt

参数

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

返回 The assign result.

返回类型 AssignResult

get_horizontal_bboxes(*gt_rbboxes*)

get_horizontal_bboxes from polygons.

参数 **gt_rbboxes** (*torch.Tensor*) –Groundtruth polygons, shape (k, 8).

返回 The horizontal bboxes, shape (k, 4).

返回类型 gt_rect_bboxes (torch.Tensor)

kld_mixture2single(*g1*, *g2*)

Compute Kullback-Leibler Divergence between two Gaussian distribution.

参数

- **g1** (*dict[str, torch.Tensor]*) –Gaussian distribution 1.
- **g2** (*torch.Tensor*) –Gaussian distribution 2.

返回 Kullback-Leibler Divergence.

返回类型 torch.Tensor

kld_overlaps(*gt_rbboxes*, *points*, *eps=1e-06*)

Compute overlaps between polygons and points by Kullback-Leibler Divergence loss.

参数

- **gt_rbboxes** (*torch.Tensor*) –Ground truth polygons, shape (k, 8).
- **points** (*torch.Tensor*) –Points to be assigned, shape(n, 18).
- **eps** (*float, optional*) –Defaults to 1e-6.

返回 Kullback-Leibler Divergence loss.

返回类型 Tensor

class mmrotate.core.bbox.**ATSSObbAssigner**(*topk*, *angle_version='oc'*, *iou_calculator={'type': 'RBboxOverlaps2D'}*)

Assign a corresponding gt bbox or background to each bbox.

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

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

参数 **topk** (*float*) –Number of bbox selected in each level.

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

Assign gt to bboxes.

The assignment is done in following steps

1. compute iou between all bbox (bbox of all pyramid levels) and gt

2. compute center distance between all bbox and gt
3. on each pyramid level, for each gt, select k bbox whose center are closest to the gt center, so we total select k*l bbox as candidates for each gt
4. get corresponding iou for the these candidates, and compute the mean and std, set mean + std as the iou threshold
5. select these candidates whose iou are greater than or equal to the threshold as positive
6. limit the positive sample’s center in gt

参数

- **bboxes** (*Tensor*) –Bounding boxes to be assigned, shape(n, 5).
- **num_level_bboxes** (*List*) –num of bboxes in each level
- **gt_bboxes** (*Tensor*) –Groundtruth boxes, shape (k, 5).
- **gt_bboxes_ignore** (*Tensor, optional*) –Ground truth bboxes that are labelled as *ignored*, e.g., crowd boxes in COCO.
- **gt_labels** (*Tensor, optional*) –Label of gt_bboxes, shape (k,).

返回 The assign result.

返回类型 AssignResult

class mmrotate.core.bbox.**CSLCoder**(*angle_version*, *omega=1*, *window='gaussian'*, *radius=6*)

Circular Smooth Label Coder.

Circular Smooth Label (CSL) .

参数

- **angle_version** (*str*) –Angle definition.
- **omega** (*float, optional*) –Angle discretization granularity. Default: 1.
- **window** (*str, optional*) –Window function. Default: gaussian.
- **radius** (*int/float*) –window radius, int type for [‘triangle’, ‘rect’, ‘pulse’], float type for [‘gaussian’]. Default: 6.

decode(*angle_preds*)

Circular Smooth Label Decoder.

参数 **angle_preds** (*Tensor*) –The csl encoding of angle offset for each scale level. Has shape (num_anchors * H * W, coding_len)

返回

Angle offset for each scale level. Has shape (num_anchors * H * W, 1)

返回类型 list[Tensor]

encode(*angle_targets*)

Circular Smooth Label Encoder.

参数 **angle_targets** (*Tensor*) –Angle offset for each scale level Has shape (num_anchors * H * W, 1)

返回

The csl encoding of angle offset for each scale level. Has shape (num_anchors * H * W, coding_len)

返回类型 list[Tensor]

class mmrotate.core.bbox.**ConvexAssigner**(*scale=4*, *pos_num=3*)

Assign a corresponding gt bbox or background to each bbox. Each proposals will be assigned with *0* or a positive integer indicating the ground truth index.

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

参数

- **scale** (*float*) –IoU threshold for positive bboxes.
- **pos_num** (*float*) –find the nearest pos_num points to gt center in this
- **level.** –

assign(*points*, *gt_rbboxes*, *gt_rbboxes_ignore=None*, *gt_labels=None*, *overlaps=None*)

Assign gt to bboxes.

The assignment is done in following steps

1. compute iou between all bbox (bbox of all pyramid levels) and gt
2. compute center distance between all bbox and gt
3. on each pyramid level, for each gt, select k bbox whose center are closest to the gt center, so we total select k*l bbox as candidates for each gt
4. get corresponding iou for the these candidates, and compute the mean and std, set mean + std as the iou threshold
5. select these candidates whose iou are greater than or equal to the threshold as positive
6. limit the positive sample’s center in gt

参数

- **points** (*torch.Tensor*) –Points to be assigned, shape(n, 18).
- **gt_rbboxes** (*torch.Tensor*) –Groundtruth polygons, shape (k, 8).

- **gt_rbboxes_ignore** (*Tensor, optional*) –Ground truth polygons that are labelled as *ignored*, e.g., crowd boxes in COCO.
- **gt_labels** (*Tensor, optional*) –Label of gt_bboxes, shape (k,).

返回 The assign result.

返回类型 AssignResult

get_horizontal_bboxes(*gt_rbboxes*)

get_horizontal_bboxes from polygons.

参数 **gt_rbboxes** (*torch.Tensor*) –Groundtruth polygons, shape (k, 8).

返回 The horizontal bboxes, shape (k, 4).

返回类型 gt_rect_bboxes (torch.Tensor)

class mmrotate.core.bbox.**DeltaXYWHAHBBoxCoder**(*target_means=(0.0, 0.0, 0.0, 0.0, 0.0)*, *target_stds=(1.0, 1.0, 1.0, 1.0, 1.0)*, *angle_range='oc'*, *norm_factor=None*, *edge_swap=False*, *clip_border=True*, *add_ctr_clamp=False*, *ctr_clamp=32*)

Delta XYWHA HBBox coder.

this coder encodes bbox (x1, y1, x2, y2) into delta (dx, dy, dw, dh, da) and decodes delta (dx, dy, dw, dh, da) back to original bbox (cx, cy, w, h, a).

参数

- **target_means** (*Sequence[float]*) –Denormalizing means of target for delta coordinates
- **target_stds** (*Sequence[float]*) –Denormalizing standard deviation of target for delta coordinates
- **angle_range** (*str, optional*) –Angle representations. Defaults to 'oc'.
- **norm_factor** (*None|float, optional*) –Regularization factor of angle.
- **edge_swap** (*bool, optional*) –Whether swap the edge if w < h. Defaults to False.
- **clip_border** (*bool, optional*) –Whether clip the objects outside the border of the image. Defaults to True.
- **add_ctr_clamp** (*bool*) –Whether to add center clamp, when added, the predicted box is clamped is its center is too far away from the original anchor's center. Only used by YOLOF. Default False.
- **ctr_clamp** (*int*) –the maximum pixel shift to clamp. Only used by YOLOF. Default 32.

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

Apply transformation *pred_bboxes* to *boxes*.

参数

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

  Encoded offsets with respect to each roi. Has shape (B, N, num_classes * 5) or (B, N, 5) or

  (N, num_classes * 5) or (N, 5). Note N = num_anchors * W * H when rois is a grid of anchors.
- **(Sequence[int] or torch.Tensor or Sequence[** (*max_shape*) –Sequence[int]],optional): Maximum bounds for boxes, specifies (H, W, C) or (H, W). If bboxes shape is (B, N, 5), then the max_shape should be a Sequence[Sequence[int]] and the length of max_shape should also be B.
- **wh_ratio_clip** (*float, optional*) –The allowed ratio between width and height.

返回 Decoded boxes.

返回类型 torch.Tensor

encode(*bboxes*, *gt_bboxes*)

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

参数

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

返回 Box transformation deltas

返回类型 torch.Tensor

class mmrotate.core.bbox.**DeltaXYWHAOBBoxCoder**(*target_means=(0.0, 0.0, 0.0, 0.0, 0.0)*, *target_stds=(1.0, 1.0, 1.0, 1.0, 1.0)*, *angle_range='oc'*, *norm_factor=None*, *edge_swap=False*, *proj_xy=False*, *add_ctr_clamp=False*, *ctr_clamp=32*)

Delta XYWHA OBBox coder. This coder is used for rotated objects detection (for example on task1 of DOTA dataset). this coder encodes bbox (xc, yc, w, h, a) into delta (dx, dy, dw, dh, da) and decodes delta (dx, dy, dw, dh, da) back to original bbox (xc, yc, w, h, a).

参数

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

- **angle_range** (*str, optional*) –Angle representations. Defaults to 'oc'.
- **norm_factor** (*None|float, optional*) –Regularization factor of angle.
- **edge_swap** (*bool, optional*) –Whether swap the edge if w < h. Defaults to False.
- **proj_xy** (*bool, optional*) –Whether project x and y according to angle. Defaults to False.
- **add_ctr_clamp** (*bool*) –Whether to add center clamp, when added, the predicted box is clamped is its center is too far away from the original anchor's center. Only used by YOLOF. Default False.
- **ctr_clamp** (*int*) –the maximum pixel shift to clamp. Only used by YOLOF. Default 32.

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

Apply transformation *pred_bboxes* to *boxes*.

参数

- **bboxes** (*torch.Tensor*) –Basic boxes. Shape (B, N, 5) or (N, 5)
- **pred_bboxes** (*torch.Tensor*) –Encoded offsets with respect to each roi. Has shape (B, N, num_classes * 5) or (B, N, 5) or (N, num_classes * 5) or (N, 5). Note N = num_anchors * W * H when rois is a grid of anchors.
- **max_shape** (*Sequence[int] or torch.Tensor or Sequence[Sequence[int]],optional*) –Maximum bounds for boxes, specifies (H, W, C) or (H, W). If bboxes shape is (B, N, 5), then the max_shape should be a Sequence[Sequence[int]] and the length of max_shape should also be B.
- **wh_ratio_clip** (*float, optional*) –The allowed ratio between width and height.

返回 Decoded boxes.

返回类型 torch.Tensor

encode(*bboxes*, *gt_bboxes*)

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

参数

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

返回 Box transformation deltas

返回类型 torch.Tensor

class mmrotate.core.bbox.**GVFixCoder**(*angle_range='oc'*, ***kwargs*)

Gliding vertex fix coder.

this coder encodes bbox (cx, cy, w, h, a) into delta (dt, dr, dd, dl) and decodes delta (dt, dr, dd, dl) back to original bbox (cx, cy, w, h, a).

参数 **angle_range** (*str, optional*) –Angle representations. Defaults to 'oc'.

decode (*hbboxes*, *fix_deltas*)

Apply transformation *fix_deltas* to *boxes*.

参数

- **hbboxes** (*torch.Tensor*) –Basic boxes. Shape (B, N, 4) or (N, 4)
- **fix_deltas** (*torch.Tensor*) –Encoded offsets with respect to each roi. Has shape (B, N, num_classes * 4) or (B, N, 4) or (N, num_classes * 4) or (N, 4). Note N = num_anchors * W * H when rois is a grid of anchors.

返回 Decoded boxes.

返回类型 torch.Tensor

encode (*rbboxes*)

Get box regression transformation deltas.

参数 **rbboxes** (*torch.Tensor*) –Source boxes, e.g., object proposals.

返回 Box transformation deltas

返回类型 torch.Tensor

class mmrotate.core.bbox.**GVRatioCoder** (*angle_range='oc'*, ***kwargs*)

Gliding vertex ratio coder.

this coder encodes bbox (cx, cy, w, h, a) into delta (ratios).

参数 **angle_range** (*str, optional*) –Angle representations. Defaults to 'oc'.

decode (*bboxes*, *bboxes_pred*)

Apply transformation *fix_deltas* to *boxes*.

参数

- **bboxes** (*torch.Tensor*) –
- **bboxes_pred** (*torch.Tensor*) –

返回 NotImplementedError

encode (*rbboxes*)

Get box regression transformation deltas.

参数 **rbboxes** (*torch.Tensor*) –Source boxes, e.g., object proposals.

返回 Box transformation deltas

返回类型 torch.Tensor

class mmrotate.core.bbox.**GaussianMixture**(*n_components*, *n_features=2*, *mu_init=None*, *var_init=None*, *eps=1e-06*, *requires_grad=False*)

Initializes the Gaussian mixture model and brings all tensors into their required shape.

参数

- **n_components** (*int*) –number of components.
- **n_features** (*int, optional*) –number of features.
- **mu_init** (*torch.Tensor, optional*) –(T, k, d)
- **var_init** (*torch.Tensor, optional*) –(T, k, d) or (T, k, d, d)
- **eps** (*float, optional*) –Defaults to 1e-6.
- **requires_grad** (*bool, optional*) –Defaults to False.

EM_step(*x*, *log_resp*)

From the log-probabilities, computes new parameters pi, mu, var (that maximize the log-likelihood). This is the maximization step of the EM-algorithm.

参数

- **x** (*torch.Tensor*) –(T, n, d) or (T, n, 1, d)
- **log_resp** (*torch.Tensor*) –(T, n, k, 1)

返回 pi (torch.Tensor): (T, k, 1) mu (torch.Tensor): (T, k, d) var (torch.Tensor): (T, k, d) or (T, k, d, d)

返回类型 tuple

check_size(*x*)

Make sure that the shape of x is (T, n, 1, d).

参数 **x** (*torch.Tensor*) –input tensor.

返回 output tensor.

返回类型 torch.Tensor

em_runner(*x*)

Performs one iteration of the expectation-maximization algorithm by calling the respective subroutines.

参数 **x** (*torch.Tensor*) –(n, 1, d)

estimate_log_prob(*x*)

Estimate the log-likelihood probability that samples belong to the k-th Gaussian.

参数 **x** (*torch.Tensor*) –(T, n, d) or (T, n, 1, d)

返回 log-likelihood probability that samples belong to the k-th Gaussian with dimensions (T, n, k, 1).

返回类型 torch.Tensor

fit (*x*, *delta=0.001*, *n_iter=10*)

Fits Gaussian mixture model to the data.

参数

- **x** (*torch.Tensor*) –input tensor.
- **delta** (*float, optional*) –threshold.
- **n_iter** (*int, optional*) –number of iterations.

get_score (*x*, *sum_data=True*)

Computes the log-likelihood of the data under the model.

参数

- **x** (*torch.Tensor*) –(T, n, 1, d)
- **sum_data** (*bool,optional*) –Flag of whether to sum scores.

返回 score or per_sample_score.

返回类型 torch.Tensor

log_resp_step (*x*)

Computes log-responses that indicate the (logarithmic) posterior belief (sometimes called responsibilities) that a data point was generated by one of the k mixture components. Also returns the mean of the mean of the logarithms of the probabilities (as is done in sklearn). This is the so-called expectation step of the EM-algorithm.

参数 **x** (*torch.Tensor*) –(T, n, d) or (T, n, 1, d)

返回 log_prob_norm (torch.Tensor): the mean of the mean of the logarithms of the probabilities.
log_resp (torch.Tensor): log-responses that indicate the posterior belief.

返回类型 tuple

update_mu (*mu*)

Updates mean to the provided value.

参数 **mu** (*torch.Tensor*) –

update_pi (*pi*)

Updates pi to the provided value.

参数 **pi** (*torch.Tensor*) –(T, k, 1)

update_var (*var*)

Updates variance to the provided value.

参数 **var** (*torch.Tensor*) –(T, k, d) or (T, k, d, d)

class mmrotate.core.bbox.**MaxConvexIoUAssigner**(*pos_iou_thr*, *neg_iou_thr*, *min_pos_iou=0.0*, *gt_max_assign_all=True*, *ignore_iof_thr=-1*, *ignore_wrt_candidates=True*, *gpu_assign_thr=-1*)

Assign a corresponding gt bbox or background to each bbox. Each proposals will be assigned with *-1*, or a semi-positive integer indicating the ground truth index.

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

参数

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

assign(*points*, *gt_rbboxes*, *overlaps*, *gt_rbboxes_ignore=None*, *gt_labels=None*)

Assign gt to bboxes.

The assignment is done in following steps

1. compute iou between all bbox (bbox of all pyramid levels) and gt
2. compute center distance between all bbox and gt
3. on each pyramid level, for each gt, select k bbox whose center are closest to the gt center, so we total select k*l bbox as candidates for each gt
4. get corresponding iou for the these candidates, and compute the mean and std, set mean + std as the iou threshold
5. select these candidates whose iou are greater than or equal to the threshold as positive
6. limit the positive sample’s center in gt

参数

- **points** (*torch.Tensor*) –Points to be assigned, shape(n, 18).
- **gt_rbboxes** (*torch.Tensor*) –Groundtruth polygons, shape (k, 8).
- **overlaps** (*torch.Tensor*) –Overlaps between k gt_bboxes and n bboxes, shape(k, n).
- **gt_rbboxes_ignore** (*Tensor, optional*) –Ground truth polygons that are labelled as *ignored*, e.g., crowd boxes in COCO.
- **gt_labels** (*Tensor, optional*) –Label of gt_bboxes, shape (k,).

返回 The assign result.

返回类型 AssignResult

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

Assign w.r.t.

the overlaps of bboxes with gts.

参数

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

返回 The assign result.

返回类型 AssignResult

convex_overlaps(*gt_rbboxes*, *points*)

Compute overlaps between polygons and points.

参数

- **gt_rbboxes** (*torch.Tensor*) –Groundtruth polygons, shape (k, 8).
- **points** (*torch.Tensor*) –Points to be assigned, shape(n, 18).

返回 Overlaps between k gt_bboxes and n bboxes, shape(k, n).

返回类型 overlaps (torch.Tensor)

class mmrotate.core.bbox.**MidpointOffsetCoder**(*target_means=(0.0, 0.0, 0.0, 0.0, 0.0, 0.0)*, *target_stds=(1.0, 1.0, 1.0, 1.0, 1.0, 1.0)*, *angle_range='oc'*)

Mid point offset coder. This coder encodes bbox (x1, y1, x2, y2) into delta (dx, dy, dw, dh, da, db) and decodes delta (dx, dy, dw, dh, da, db) back to original bbox (x1, y1, x2, y2).

参数

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

- **target_stds** (*Sequence[float]*) –Denormalizing standard deviation of target for delta coordinates
- **angle_range** (*str, optional*) –Angle representations. Defaults to 'oc'.

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

Apply transformation *pred_bboxes* to *bboxes*.

参数

- **bboxes** (*torch.Tensor*) –Basic boxes. Shape (B, N, 4) or (N, 4)
- **pred_bboxes** (*torch.Tensor*) –Encoded offsets with respect to each roi. Has shape (B, N, 5) or (N, 5). Note N = num_anchors * W * H when rois is a grid of anchors.
- **(Sequence[int] or torch.Tensor or Sequence[** (*max_shape*) –Sequence[int]],optional): Maximum bounds for boxes, specifies (H, W, C) or (H, W). If bboxes shape is (B, N, 6), then the max_shape should be a Sequence[Sequence[int]] and the length of max_shape should also be B.
- **wh_ratio_clip** (*float, optional*) –The allowed ratio between width and height.

返回 Decoded boxes.

返回类型 torch.Tensor

encode(*bboxes*, *gt_bboxes*)

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

参数

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

返回 Box transformation deltas

返回类型 torch.Tensor

class mmrotate.core.bbox.**RBboxOverlaps2D**

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

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

Random sampler.

参数

- **num** (*int*) –Number of samples
- **pos_fraction** (*float*) –Fraction of positive samples
- **neg_pos_up** (*int, optional*) –Upper bound number of negative and positive samples. Defaults to -1.

- **add_gt_as_proposals** (*bool, optional*) –Whether to add ground truth boxes as proposals. Defaults to True.

random_choice(*gallery*, *num*)

Random select some elements from the gallery.

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

参数

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

返回 sampled indices.

返回类型 Tensor or ndarray

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

Sample positive and negative bboxes.

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

参数

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

返回 Sampling result.

返回类型 SamplingResult

示例

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

class mmrotate.core.bbox.**SASAssigner**(*topk*)

Assign a corresponding gt bbox or background to each bbox. Each proposals will be assigned with *0* or a positive integer indicating the ground truth index.

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

参数

- **scale** (*float*) –IoU threshold for positive bboxes.
- **pos_num** (*float*) –find the nearest pos_num points to gt center in this
- **level.** –

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

Assign gt to bboxes.

The assignment is done in following steps

1. compute iou between all bbox (bbox of all pyramid levels) and gt
2. compute center distance between all bbox and gt
3. on each pyramid level, for each gt, select k bbox whose center are closest to the gt center, so we total select k*l bbox as candidates for each gt
4. get corresponding iou for the these candidates, and compute the mean and std, set mean + std as the iou threshold
5. select these candidates whose iou are greater than or equal to the threshold as positive
6. limit the positive sample’s center in gt

参数

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

返回 The assign result.

返回类型 AssignResult

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

Map bboxes from testing scale to original image scale.

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

Builder of box assigner.

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

Builder of box coder.

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

Builder of box sampler.

mmrotate.core.bbox.**gaussian2bbox**(*gmm*)

Convert Gaussian distribution to polygons by SVD.

参数 **gmm** (*dict[str, torch.Tensor]*) –Dict of Gaussian distribution.

返回 Polygons.

返回类型 torch.Tensor

mmrotate.core.bbox.**gt2gaussian**(*target*)

Convert polygons to Gaussian distributions.

参数 **target** (*torch.Tensor*) –Polygons with shape (N, 8).

返回 Gaussian distributions.

返回类型 dict[str, torch.Tensor]

mmrotate.core.bbox.**hbb2obb**(*hbboxes*, *version='oc'*)

Convert horizontal bounding boxes to oriented bounding boxes.

参数

- **hbbs** (*torch.Tensor*) –[x_lt,y_lt,x_rb,y_rb]
- **version** (*Str*) –angle representations.

返回 [x_ctr,y_ctr,w,h,angle]

返回类型 obbs (torch.Tensor)

mmrotate.core.bbox.**norm_angle**(*angle*, *angle_range*)

Limit the range of angles.

参数

- **angle** (*ndarray*) –shape(n,).
- **angle_range** (*Str*) –angle representations.

返回 shape(n,).

返回类型 angle (ndarray)

mmrotate.core.bbox.**obb2hbb**(*rbboxes*, *version='oc'*)

Convert oriented bounding boxes to horizontal bounding boxes.

参数

- **obbs** (*torch.Tensor*) –[x_ctr,y_ctr,w,h,angle]
- **version** (*Str*) –angle representations.

返回 [x_ctr,y_ctr,w,h,-pi/2]

返回类型 hbbs (torch.Tensor)

mmrotate.core.bbox.**obb2poly**(*rbboxes*, *version='oc'*)

Convert oriented bounding boxes to polygons.

参数

- **obbs** (*torch.Tensor*) –[x_ctr,y_ctr,w,h,angle]
- **version** (*Str*) –angle representations.

返回 [x0,y0,x1,y1,x2,y2,x3,y3]

返回类型 polys (torch.Tensor)

mmrotate.core.bbox.**obb2poly_np**(*rbboxes*, *version='oc'*)

Convert oriented bounding boxes to polygons.

参数

- **obbs** (*ndarray*) –[x_ctr,y_ctr,w,h,angle]
- **version** (*Str*) –angle representations.

返回 [x0,y0,x1,y1,x2,y2,x3,y3]

返回类型 polys (ndarray)

mmrotate.core.bbox.**obb2xyxy**(*rbboxes*, *version='oc'*)

Convert oriented bounding boxes to horizontal bounding boxes.

参数

- **obbs** (*torch.Tensor*) –[x_ctr,y_ctr,w,h,angle]
- **version** (*Str*) –angle representations.

返回 [x_lt,y_lt,x_rb,y_rb]

返回类型 hbbs (torch.Tensor)

mmrotate.core.bbox.**poly2obb**(*polys*, *version='oc'*)

Convert polygons to oriented bounding boxes.

参数

- **polys** (*torch.Tensor*) –[x0,y0,x1,y1,x2,y2,x3,y3]
- **version** (*Str*) –angle representations.

返回 [x_ctr,y_ctr,w,h,angle]

返回类型 obbs (torch.Tensor)

mmrotate.core.bbox.**poly2obb_np**(*polys*, *version='oc'*)

Convert polygons to oriented bounding boxes.

参数

- **polys** (*ndarray*) –[x0,y0,x1,y1,x2,y2,x3,y3]
- **version** (*Str*) –angle representations.

返回 [x_ctr,y_ctr,w,h,angle]

返回类型 obbs (ndarray)

mmrotate.core.bbox.**rbbox2result**(*bboxes*, *labels*, *num_classes*)

Convert detection results to a list of numpy arrays.

参数

- **bboxes** (*torch.Tensor*) –shape (n, 6)
- **labels** (*torch.Tensor*) –shape (n,)
- **num_classes** (*int*) –class number, including background class

返回 bbox results of each class

返回类型 list(ndarray)

mmrotate.core.bbox.**rbbox2roi**(*bbox_list*)

Convert a list of bboxes to roi format.

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

返回 shape (n, 6), [batch_ind, cx, cy, w, h, a]

返回类型 Tensor

mmrotate.core.bbox.**rbbox_overlaps**(*bboxes1*, *bboxes2*, *mode='iou'*, *is_aligned=False*)

Calculate overlap between two set of bboxes.

参数

- **bboxes1** (*torch.Tensor*) –shape (B, m, 5) in <cx, cy, w, h, a> format or empty.
- **bboxes2** (*torch.Tensor*) –shape (B, n, 5) in <cx, cy, w, h, a> format or empty.
- **mode** (*str*) –"iou" (intersection over union), "iof" (intersection over foreground) or "giou" (generalized intersection over union). Default "iou".
- **is_aligned** (*bool, optional*) –If True, then m and n must be equal. Default False.

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

返回类型 Tensor

24.3 patch

`mmrotate.core.patch.`**`get_multiscale_patch`**(*sizes*, *steps*, *ratios*)

Get multiscale patch sizes and steps.

参数

- **sizes** (*list*) –A list of patch sizes.
- **steps** (*list*) –A list of steps to slide patches.
- **ratios** (*list*) –Multiscale ratios. devidie to each size and step and generate patches in new scales.

返回 A list of multiscale patch sizes. new_steps (list): A list of steps corresponding to new_sizes.

返回类型 new_sizes (list)

`mmrotate.core.patch.`**`merge_results`**(*results*, *offsets*, *img_shape*, *iou_thr=0.1*, *device='cpu'*)

Merge patch results via nms.

参数

- **results** (*list[np.ndarray] | list[tuple]*) –A list of patches results.
- **offsets** (*np.ndarray*) –Positions of the left top points of patches.
- **img_shape** (*tuple*) –A tuple of the huge image's width and height.
- **iou_thr** (*float*) –The IoU threshold of NMS.
- **device** (*str*) –The device to call nms.

Retunrns: list[np.ndarray]: Detection results after merging.

`mmrotate.core.patch.`**`slide_window`**(*width*, *height*, *sizes*, *steps*, *img_rate_thr=0.6*)

Slide windows in images and get window position.

参数

- **width** (*int*) –The width of the image.
- **height** (*int*) –The height of the image.
- **sizes** (*list*) –List of window's sizes.
- **steps** (*list*) –List of window's steps.
- **img_rate_thr** (*float*) –Threshold of window area divided by image area.

返回 Information of valid windows.

返回类型 np.ndarray

24.4 evaluation

mmrotate.core.evaluation.**eval_rbbox_map**(*det_results*, *annotations*, *scale_ranges=None*, *iou_thr=0.5*, *use_07_metric=True*, *dataset=None*, *logger=None*, *nproc=4*)

Evaluate mAP of a rotated dataset.

参数

- **det_results** (*list[list]*) –[[cls1_det, cls2_det, ···], ···]. The outer list indicates images, and the inner list indicates per-class detected bboxes.
- **annotations** (*list[dict]*) –Ground truth annotations where each item of the list indicates an image. Keys of annotations are:
  - *bboxes*: numpy array of shape (n, 5)
  - *labels*: numpy array of shape (n,)
  - *bboxes_ignore* (optional): numpy array of shape (k, 5)
  - *labels_ignore* (optional): numpy array of shape (k,)
- **scale_ranges** (*list[tuple] | None*) –Range of scales to be evaluated, in the format [(min1, max1), (min2, max2), ···]. A range of (32, 64) means the area range between (32**2, 64**2). Default: None.
- **iou_thr** (*float*) –IoU threshold to be considered as matched. Default: 0.5.
- **use_07_metric** (*bool*) –Whether to use the voc07 metric.
- **dataset** (*list[str] | str | None*) –Dataset name or dataset classes, there are minor differences in metrics for different datasets, e.g. “voc07” , “imagenet_det” , etc. Default: None.
- **logger** (*logging.Logger | str | None*) –The way to print the mAP summary. See *mmcv.utils.print_log()* for details. Default: None.
- **nproc** (*int*) –Processes used for computing TP and FP. Default: 4.

返回 (mAP, [dict, dict, ···])

返回类型 tuple

24.5 post_processing

mmrotate.core.post_processing.**aug_multiclass_nms_rotated**(*merged_bboxes*, *merged_labels*, *score_thr*, *nms*, *max_num*, *classes*)

NMS for aug multi-class bboxes.

参数

- **multi_bboxes** (*torch.Tensor*) –shape (n, #class*5) or (n, 5)
- **multi_scores** (*torch.Tensor*) –shape (n, #class), where the last column contains scores of the background class, but this will be ignored.
- **score_thr** (*float*) –bbox threshold, bboxes with scores lower than it will not be considered.
- **nms** (*float*) –Config of NMS.
- **max_num** (*int, optional*) –if there are more than max_num bboxes after NMS, only top max_num will be kept. Default to -1.
- **classes** (*int*) –number of classes.

返回

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

返回类型 tuple (dets, labels)

mmrotate.core.post_processing.**multiclass_nms_rotated**(*multi_bboxes*, *multi_scores*, *score_thr*, *nms*, *max_num=- 1*, *score_factors=None*, *return_inds=False*)

NMS for multi-class bboxes.

参数

- **multi_bboxes** (*torch.Tensor*) –shape (n, #class*5) or (n, 5)
- **multi_scores** (*torch.Tensor*) –shape (n, #class), where the last column contains scores of the background class, but this will be ignored.
- **score_thr** (*float*) –bbox threshold, bboxes with scores lower than it will not be considered.
- **nms** (*float*) –Config of NMS.
- **max_num** (*int, optional*) –if there are more than max_num bboxes after NMS, only top max_num will be kept. Default to -1.

- **score_factors** (*Tensor, optional*) –The factors multiplied to scores before applying NMS. Default to None.
- **return_inds** (*bool, optional*) –Whether return the indices of kept bboxes. Default to False.

返回 tensors of shape (k, 5), (k), and (k). Dets are boxes with scores. Labels are 0-based.

返回类型 tuple (dets, labels, indices (optional))

24.6 visualization

mmrotate.core.visualization.**get_palette**(*palette*, *num_classes*)

Get palette from various inputs.

参数

- **palette** (list[tuple] | str | tuple | Color) –palette inputs.
- **num_classes** (*int*) –the number of classes.

返回 A list of color tuples.

返回类型 list[tuple[int]]

mmrotate.core.visualization.**imshow_det_rbboxes**(*img*, *bboxes=None*, *labels=None*, *segms=None*, *class_names=None*, *score_thr=0*, *bbox_color='green'*, *text_color='green'*, *mask_color=None*, *thickness=2*, *font_size=13*, *win_name=''*, *show=True*, *wait_time=0*, *out_file=None*)

Draw bboxes and class labels (with scores) on an image.

参数

- **img** (*str | ndarray*) –The image to be displayed.
- **bboxes** (*ndarray*) –Bounding boxes (with scores), shaped (n, 5) or (n, 6).
- **labels** (*ndarray*) –Labels of bboxes.
- **segms** (*ndarray | None*) –Masks, shaped (n,h,w) or None.
- **class_names** (*list[str]*) –Names of each classes.
- **score_thr** (*float*) –Minimum score of bboxes to be shown. Default: 0.
- **bbox_color** (*list[tuple] | tuple | str | None*) –Colors of bbox lines. If a single color is given, it will be applied to all classes. The tuple of color should be in RGB order. Default: 'green'.

- **text_color** (*list[tuple] | tuple | str | None*) –Colors of texts. If a single color is given, it will be applied to all classes. The tuple of color should be in RGB order. Default: ‘green’.
- **mask_color** (*list[tuple] | tuple | str | None, optional*) –Colors of masks. If a single color is given, it will be applied to all classes. The tuple of color should be in RGB order. Default: None.
- **thickness** (*int*) –Thickness of lines. Default: 2.
- **font_size** (*int*) –Font size of texts. Default: 13.
- **show** (*bool*) –Whether to show the image. Default: True.
- **win_name** (*str*) –The window name. Default: ‘’.
- **wait_time** (*float*) –Value of waitKey param. Default: 0.
- **out_file** (*str, optional*) –The filename to write the image. Default: None.

返回 The image with bboxes drawn on it.

返回类型 ndarray

CHAPTER 25

mmrotate.datasets

25.1 datasets

class mmrotate.datasets.**DOTADataset**(*ann_file*, *pipeline*, *version='oc'*, *difficulty=100*, ***kwargs*)

DOTA dataset for detection.

参数

- **ann_file** (*str*) –Annotation file path.
- **pipeline** (*list[dict]*) –Processing pipeline.
- **version** (*str, optional*) –Angle representations. Defaults to 'oc'.
- **difficulty** (*bool, optional*) –The difficulty threshold of GT.

evaluate(*results*, *metric='mAP'*, *logger=None*, *proposal_nums=(100, 300, 1000)*, *iou_thr=0.5*, *scale_ranges=None*, *nproc=4*)

Evaluate the dataset.

参数

- **results** (*list*) –Testing results of the dataset.
- **metric** (*str | list[str]*) –Metrics to be evaluated.
- **logger** (*logging.Logger | None | str*) –Logger used for printing related information during evaluation. Default: None.

- **proposal_nums** (*Sequence[int]*) –Proposal number used for evaluating recalls, such as recall@100, recall@1000. Default: (100, 300, 1000).
- **iou_thr** (*float | list[float]*) –IoU threshold. It must be a float when evaluating mAP, and can be a list when evaluating recall. Default: 0.5.
- **scale_ranges** (*list[tuple] | None*) –Scale ranges for evaluating mAP. Default: None.
- **nproc** (*int*) –Processes used for computing TP and FP. Default: 4.

format_results(*results*, *submission_dir=None*, *nproc=4*, ***kwargs*)

Format the results to submission text (standard format for DOTA evaluation).

参数

- **results** (*list*) –Testing results of the dataset.
- **submission_dir** (*str, optional*) –The folder that contains submission files. If not specified, a temp folder will be created. Default: None.
- **nproc** (*int, optional*) –number of process.

返回

- result_files (dict): a dict containing the json filepaths
- tmp_dir (str): the temporal directory created for saving json files when submission_dir is not specified.

返回类型 tuple

load_annotations(*ann_folder*)

参数 **ann_folder** –folder that contains DOTA v1 annotations txt files

merge_det(*results*, *nproc=4*)

Merging patch bboxes into full image.

参数

- **results** (*list*) –Testing results of the dataset.
- **nproc** (*int*) –number of process. Default: 4.

class mmrotate.datasets.**HRSCDataset**(*ann_file*, *pipeline*, *img_subdir='JPEGImages'*, *ann_subdir='Annotations'*, *classwise=False*, *version='oc'*, ***kwargs*)

HRSC dataset for detection.

参数

- **ann_file** (*str*) –Annotation file path.

- **pipeline** (*list[dict]*) –Processing pipeline.
- **img_subdir** (*str*) –Subdir where images are stored. Default: JPEGImages.
- **ann_subdir** (*str*) –Subdir where annotations are. Default: Annotations.
- **classwise** (*bool*) –Whether to use all classes or only ship.
- **version** (*str, optional*) –Angle representations. Defaults to 'oc'.

evaluate(*results*, *metric='mAP'*, *logger=None*, *proposal_nums=(100, 300, 1000)*, *iou_thr=[0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95]*, *scale_ranges=None*, *use_07_metric=True*, *nproc=4*)

Evaluate the dataset.

参数

- **results** (*list*) –Testing results of the dataset.
- **metric** (*str | list[str]*) –Metrics to be evaluated.
- **logger** (*logging.Logger | None | str*) –Logger used for printing related information during evaluation. Default: None.
- **proposal_nums** (*Sequence[int]*) –Proposal number used for evaluating recalls, such as recall@100, recall@1000. Default: (100, 300, 1000).
- **iou_thr** (*float | list[float]*) –IoU threshold. It must be a float when evaluating mAP, and can be a list when evaluating recall. Default: 0.5.
- **scale_ranges** (*list[tuple] | None*) –Scale ranges for evaluating mAP. Default: None.
- **use_07_metric** (*bool*) –Whether to use the voc07 metric.
- **nproc** (*int*) –Processes used for computing TP and FP. Default: 4.

load_annotations(*ann_file*)

Load annotation from XML style ann_file.

参数 **ann_file** (*str*) –Path of Imageset file.

返回 Annotation info from XML file.

返回类型 list[dict]

class mmrotate.datasets.**SARDataset**(*ann_file*, *pipeline*, *version='oc'*, *difficulty=100*, ***kwargs*)

SAR ship dataset for detection (Support RSSDD and HRSID).

25.2 pipelines

class mmrotate.datasets.pipelines.**LoadPatchFromImage**(*to_float32=False*, *color_type='color'*, *channel_order='bgr'*, *file_client_args={'backend': 'disk'}*)

Load an patch from the huge image.

Similar with `LoadImageFromFile`, but only reserve a patch of `results['img']` according to `results['win']`.

class mmrotate.datasets.pipelines.**PolyRandomRotate**(*rotate_ratio=0.5*, *mode='range'*, *angles_range=180*, *auto_bound=False*, *rect_classes=None*, *version='le90'*)

Rotate img & bbox. Reference: https://github.com/hukaixuan19970627/OrientedRepPoints_DOTA

参数

- **rotate_ratio** (*float, optional*) –The rotating probability. Default: 0.5.
- **mode** (*str, optional*) –Indicates whether the angle is chosen in a random range (mode='range') or in a preset list of angles (mode='value'). Defaults to 'range'.
- **angles_range** (*int|list[int], optional*) –The range of angles. If mode='range', angle_ranges is an int and the angle is chosen in (-angles_range, +angles_ranges). If mode='value', angles_range is a non-empty list of int and the angle is chosen in angles_range. Defaults to 180 as default mode is 'range'.
- **auto_bound** (*bool, optional*) –whether to find the new width and height bounds.
- **rect_classes** (*None|list, optional*) –Specifies classes that needs to be rotated by a multiple of 90 degrees.
- **version** (*str, optional*) –Angle representations. Defaults to 'le90'.

apply_coords(*coords*)

coords should be a N * 2 array-like, containing N couples of (x, y) points

apply_image(*img*, *bound_h*, *bound_w*, *interp=1*)

img should be a numpy array, formatted as Height * Width * Nchannels

create_rotation_matrix(*center*, *angle*, *bound_h*, *bound_w*, *offset=0*)

Create rotation matrix.

filter_border(*bboxes*, *h*, *w*)

Filter the box whose center point is outside or whose side length is less than 5.

property is_rotate

Randomly decide whether to rotate.

class mmrotate.datasets.pipelines.**RMosaic**(*img_scale=(640, 640)*, *center_ratio_range=(0.5, 1.5)*, *min_bbox_size=10*, *bbox_clip_border=True*, *skip_filter=True*, *pad_val=114*, *prob=1.0*, *version='oc'*)

Rotate Mosaic augmentation. Inherit from *mmdet.datasets.pipelines.transforms.Mosaic*.

Given 4 images, mosaic transform combines them into one output image. The output image is composed of the parts from each sub- image.

参数

- **img_scale** (*Sequence[int]*) –Image size after mosaic pipeline of single image. The shape order should be (height, width). Defaults to (640, 640).
- **center_ratio_range** (*Sequence[float]*) –Center ratio range of mosaic output. Defaults to (0.5, 1.5).
- **min_bbox_size** (*int | float*) –The minimum pixel for filtering invalid bboxes after the mosaic pipeline. Defaults to 0.
- **bbox_clip_border** (*bool, optional*) –Whether to clip the objects outside the border of the image. In some dataset like MOT17, the gt bboxes are allowed to cross the border of images. Therefore, we don't need to clip the gt bboxes in these cases. Defaults to True.
- **skip_filter** (*bool*) –Whether to skip filtering rules. If it is True, the filter rule will not be applied, and the *min_bbox_size* is invalid. Defaults to True.
- **pad_val** (*int*) –Pad value. Defaults to 114.
- **prob** (*float*) –Probability of applying this transformation. Defaults to 1.0.
- **version** (*str, optional*) –Angle representations. Defaults to *oc*.

class mmrotate.datasets.pipelines.**RRandomFlip**(*flip_ratio=None*, *direction='horizontal'*, *version='oc'*)

参数

- **flip_ratio** (*float | list[float], optional*) –The flipping probability. Default: None.
- **direction** (*str | list[str], optional*) –The flipping direction. Options are 'horizontal', 'vertical', 'diagonal'.
- **version** (*str, optional*) –Angle representations. Defaults to 'oc'.

bbox_flip(*bboxes*, *img_shape*, *direction*)

Flip bboxes horizontally or vertically.

参数

- **bboxes** (*ndarray*) –shape (…, 5*k)

- **img_shape** (*tuple*) –(height, width)

返回 Flipped bounding boxes.

返回类型 numpy.ndarray

class mmrotate.datasets.pipelines.**RResize**(*img_scale=None*, *multiscale_mode='range'*, *ratio_range=None*)

Resize images & rotated bbox Inherit Resize pipeline class to handle rotated bboxes.

参数

- **img_scale** (*tuple or list[tuple]*) –Images scales for resizing.
- **multiscale_mode** (*str*) –Either “range” or “value”.
- **ratio_range** (*tuple[float]*) –(min_ratio, max_ratio).

CHAPTER 26

mmrotate.models

26.1 detectors

class mmrotate.models.detectors.**GlidingVertex**(*backbone*, *rpn_head*, *roi_head*, *train_cfg*, *test_cfg*, *neck=None*, *pretrained=None*, *init_cfg=None*)

Implementation of Gliding Vertex on the Horizontal Bounding Box for Multi-Oriented Object Detection

class mmrotate.models.detectors.**OrientedRCNN**(*backbone*, *rpn_head*, *roi_head*, *train_cfg*, *test_cfg*, *neck=None*, *pretrained=None*, *init_cfg=None*)

Implementation of Oriented R-CNN for Object Detection.

forward_dummy(*img*)

Used for computing network flops.

See *mmrotate/tools/analysis_tools/get_flops.py*

class mmrotate.models.detectors.**R3Det**(*num_refine_stages*, *backbone*, *neck=None*, *bbox_head=None*, *frm_cfgs=None*, *refine_heads=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

Rotated Refinement RetinaNet.

aug_test(*imgs*, *img_metas*, ***kwargs*)

Test function with test time augmentation.

extract_feat(*img*)

Directly extract features from the backbone+neck.

forward_dummy(*img*)

Used for computing network flops.

See *mmedetection/tools/get_flops.py*

forward_train(*img*, *img_metas*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*)

Forward function.

simple_test(*img*, *img_meta*, *rescale=False*)

Test function without test time augmentation.

参数

- **imgs** (*list[torch.Tensor]*) –List of multiple images
- **img_metas** (*list[dict]*) –List of image information.
- **rescale** (*bool, optional*) –Whether to rescale the results. Defaults to False.

返回 BBox results of each image and classes. The outer list corresponds to each image. The inner list corresponds to each class.

返回类型 list[list[np.ndarray]]

class mmrotate.models.detectors.**ReDet**(*backbone*, *rpn_head*, *roi_head*, *train_cfg*, *test_cfg*, *neck=None*, *pretrained=None*, *init_cfg=None*)

Implementation of ReDet: A Rotation-equivariant Detector for Aerial Object Detection.

class mmrotate.models.detectors.**RoITransformer**(*backbone*, *rpn_head*, *roi_head*, *train_cfg*, *test_cfg*, *neck=None*, *pretrained=None*, *init_cfg=None*)

Implementation of Learning RoI Transformer for Oriented Object Detection in Aerial Images.

class mmrotate.models.detectors.**RotatedBaseDetector**(*init_cfg=None*)

Base class for rotated detectors.

show_result(*img*, *result*, *score_thr=0.3*, *bbox_color=(72, 101, 241)*, *text_color=(72, 101, 241)*, *mask_color=None*, *thickness=2*, *font_size=13*, *win_name=''*, *show=False*, *wait_time=0*, *out_file=None*, ***kwargs*)

Draw *result* over *img*.

参数

- **img** (*str or Tensor*) –The image to be displayed.
- **result** (*Tensor or tuple*) –The results to draw over *img* bbox_result or (bbox_result, segm_result).
- **score_thr** (*float, optional*) –Minimum score of bboxes to be shown. Default: 0.3.
- **bbox_color** (str or tuple(int) or Color) –Color of bbox lines. The tuple of color should be in BGR order. Default: 'green'

- **text_color** (str or tuple(int) or Color) –Color of texts. The tuple of color should be in BGR order. Default: 'green'
- **mask_color** (None or str or tuple(int) or Color) –Color of masks. The tuple of color should be in BGR order. Default: None
- **thickness** (*int*) –Thickness of lines. Default: 2
- **font_size** (*int*) –Font size of texts. Default: 13
- **win_name** (*str*) –The window name. Default: ''
- **wait_time** (*float*) –Value of waitKey param. Default: 0.
- **show** (*bool*) –Whether to show the image. Default: False.
- **out_file** (*str or None*) –The filename to write the image. Default: None.

返回 Only if not *show* or *out_file*

返回类型 img (torch.Tensor)

class mmrotate.models.detectors.**RotatedFCOS**(*backbone*, *neck*, *bbox_head*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

Implementation of Rotated FCOS.

class mmrotate.models.detectors.**RotatedFasterRCNN**(*backbone*, *rpn_head*, *roi_head*, *train_cfg*, *test_cfg*, *neck=None*, *pretrained=None*, *init_cfg=None*)

Implementation of Rotated Faster R-CNN.

class mmrotate.models.detectors.**RotatedRepPoints**(*backbone*, *neck*, *bbox_head*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*)

Implementation of Rotated RepPoints.

class mmrotate.models.detectors.**RotatedRetinaNet**(*backbone*, *neck*, *bbox_head*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

Implementation of Rotated RetinaNet.

class mmrotate.models.detectors.**RotatedSingleStageDetector**(*backbone*, *neck=None*, *bbox_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

Base class for rotated single-stage detectors.

Single-stage detectors directly and densely predict bounding boxes on the output features of the backbone+neck.

aug_test(*imgs*, *img_metas*, *rescale=False*)

Test function with test time augmentation.

参数

- **imgs** (*list[Tensor]*) –the outer list indicates test-time augmentations and inner Tensor should have a shape NxCxHxW, which contains all images in the batch.
- **img_metas** (*list[list[dict]]*) –the outer list indicates test-time augs (multiscale, flip, etc.) and the inner list indicates images in a batch. each dict has image information.
- **rescale** (*bool, optional*) –Whether to rescale the results. Defaults to False.

返回

BBox results of each image and classes. The outer list corresponds to each image. The inner list corresponds to each class.

返回类型 list[list[np.ndarray]]

extract_feat(*img*)

Directly extract features from the backbone+neck.

forward_dummy(*img*)

Used for computing network flops.

See *mmdetection/tools/analysis_tools/get_flops.py*

forward_train(*img*, *img_metas*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*)

参数

- **img** (*Tensor*) –Input images of shape (N, C, H, W). Typically these should be mean centered and std scaled.
- **img_metas** (*list[dict]*) –A List of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see `mmdet.datasets.pipelines.Collect`.
- **gt_bboxes** (*list[Tensor]*) –Each item are the truth boxes for each image in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) –Class indices corresponding to each box
- **gt_bboxes_ignore** (*None | list[Tensor]*) –Specify which bounding boxes can be ignored when computing the loss.

返回 A dictionary of loss components.

返回类型 dict[str, Tensor]

simple_test(*img*, *img_metas*, *rescale=False*)

Test function without test time augmentation.

参数

- **imgs** (*list[torch.Tensor]*) –List of multiple images
- **img_metas** (*list[dict]*) –List of image information.
- **rescale** (*bool, optional*) –Whether to rescale the results. Defaults to False.

返回 BBox results of each image and classes. The outer list corresponds to each image. The inner list corresponds to each class.

返回类型 list[list[np.ndarray]]

class mmrotate.models.detectors.**RotatedTwoStageDetector**(*backbone*, *neck=None*, *rpn_head=None*, *roi_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

Base class for rotated two-stage detectors.

Two-stage detectors typically consisting of a region proposal network and a task-specific regression head.

async async_simple_test(*img*, *img_meta*, *proposals=None*, *rescale=False*)

Async test without augmentation.

aug_test(*imgs*, *img_metas*, *rescale=False*)

Test with augmentations.

If rescale is False, then returned bboxes and masks will fit the scale of imgs[0].

extract_feat(*img*)

Directly extract features from the backbone+neck.

forward_dummy(*img*)

Used for computing network flops.

See *mmdetection/tools/analysis_tools/get_flops.py*

forward_train(*img*, *img_metas*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*, *gt_masks=None*, *proposals=None*, ***kwargs*)

参数

- **img** (*Tensor*) –of shape (N, C, H, W) encoding input images. Typically these should be mean centered and std scaled.
- **img_metas** (*list[dict]*) –list of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmdet/datasets/pipelines/formatting.py:Collect*.

- **gt_bboxes** (*list[Tensor]*) –Ground truth bboxes for each image with shape (num_gts, 5) in [cx, cy, w, h, a] format.
- **gt_labels** (*list[Tensor]*) –class indices corresponding to each box
- **gt_bboxes_ignore** (*None | list[Tensor]*) –specify which bounding boxes can be ignored when computing the loss.
- **gt_masks** (*None | Tensor*) –true segmentation masks for each box used if the architecture supports a segmentation task.
- **proposals** –override rpn proposals with custom proposals. Use when *with_rpn* is False.

返回 a dictionary of loss components

返回类型 dict[str, Tensor]

simple_test(*img*, *img_metas*, *proposals=None*, *rescale=False*)

Test without augmentation.

property with_roi_head

whether the detector has a RoI head

Type bool

property with_rpn

whether the detector has RPN

Type bool

class mmrotate.models.detectors.**S2ANet**(*backbone*, *neck=None*, *fam_head=None*, *align_cfgs=None*, *odm_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*)

Implementation of Align Deep Features for Oriented Object Detection.

aug_test(*imgs*, *img_metas*, ***kwargs*)

Test function with test time augmentation.

extract_feat(*img*)

Directly extract features from the backbone+neck.

forward_dummy(*img*)

Used for computing network flops.

See *mmedetection/tools/get_flops.py*

forward_train(*img*, *img_metas*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*)

Forward function of S2ANet.

simple_test(*img*, *img_meta*, *rescale=False*)

Test function without test time augmentation.

参数

- **imgs** (*list[torch.Tensor]*) –List of multiple images
- **img_metas** (*list[dict]*) –List of image information.
- **rescale** (*bool, optional*) –Whether to rescale the results. Defaults to False.

返回 BBox results of each image and classes. The outer list corresponds to each image. The inner list corresponds to each class.

返回类型 list[list[np.ndarray]]

26.2 backbones

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

ReResNet backbone.

Please refer to the paper for details.

参数

- **depth** (*int*) –Network depth, from {18, 34, 50, 101, 152}.
- **in_channels** (*int*) –Number of input image channels. Default: 3.
- **stem_channels** (*int*) –Output channels of the stem layer. Default: 64.
- **base_channels** (*int*) –Middle channels of the first stage. Default: 64.
- **num_stages** (*int*) –Stages of the network. Default: 4.
- **strides** (*Sequence[int]*) –Strides of the first block of each stage. Default: `(1, 2, 2, 2)`.
- **dilations** (*Sequence[int]*) –Dilation of each stage. Default: `(1, 1, 1, 1)`.
- **out_indices** (*Sequence[int]*) –Output from which stages. If only one stage is specified, a single tensor (feature map) is returned, otherwise multiple stages are specified, a tuple of tensors will be returned. Default: `(3, )`.
- **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. Default: False.
- **avg_down** (*bool*) –Use AvgPool instead of stride conv when downsampling in the bottleneck. Default: False.
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Default: -1.
- **conv_cfg** (*dict | None*) –The config dict for conv layers. Default: None.
- **norm_cfg** (*dict*) –The config dict for norm layers.
- **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.
- **zero_init_residual** (*bool*) –Whether to use zero init for last norm layer in resblocks to let them behave as identity. Default: True.

forward(*x*)

Forward function of ReResNet.

make_res_layer(***kwargs*)

Build Reslayer.

property norm1

Get normalizion layer’ s name.

train(*mode=True*)

Train function of ReResNet.

26.3 necks

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

ReFPN.

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, optional*) –Index of the start input backbone level used to build the feature pyramid. Default: 0.
- **end_level** (*int, optional*) –Index of the end input backbone level (exclusive) to build the feature pyramid. Default: -1, which means the last level.
- **add_extra_convs** (*bool, optional*) –It decides whether to add conv layers on top of the original feature maps. Default to False.
- **extra_convs_on_inputs** (*bool, optional*) –It specifies the source feature map of the extra convs is the last feat map of neck inputs.
- **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, optional*) –Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict, optional*) –Config dict for normalization layer. Default: None.
- **activation** (*str, optional*) –Activation layer in ConvModule. Default: None.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

forward(*inputs*)

Forward function of ReFPN.

26.4 dense_heads

class mmrotate.models.dense_heads.**CSLRFCOSHead**(*separate_angle=True*, *scale_angle=False*, *angle_coder={'angle_version': 'le90', 'omega': 1, 'radius': 6, 'type': 'CSLCoder', 'window': 'gaussian'}*, ***kwargs*)

Use ‘Circular Smooth Label (CSL)

<https://link.springer.com/chapter/10.1007/978-3-030-58598-3_40>‘_ . in FCOS.

参数

- **separate_angle** (*bool*) –If true, angle prediction is separated from bbox regression loss. In CSL only support True. Default: True. scale_angle (bool): If true, add scale to angle pred branch.

  In CSL only support False. Default: False.
- **angle_coder** (*dict*) –Config of angle coder.

loss(*cls_scores*, *bbox_preds*, *angle_preds*, *centernesses*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)

Compute loss of the head. :param cls_scores: Box scores for each scale level,

each is a 4D-tensor, the channel number is num_points * num_classes.

参数

- **bbox_preds** (*list[Tensor]*) –Box energies / deltas for each scale level, each is a 4D-tensor, the channel number is num_points * 4.
- **angle_preds** (*list[Tensor]*) –Box angle for each scale level, each is a 4D-tensor, the channel number is num_points * 1.
- **centernesses** (*list[Tensor]*) –centerness for each scale level, each is a 4D-tensor, the channel number is num_points * 1.
- **gt_bboxes** (*list[Tensor]*) –Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) –class indices corresponding to each box
- **img_metas** (*list[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | list[Tensor]*) –specify which bounding boxes can be ignored when computing the loss.

返回 A dictionary of loss components.

返回类型 dict[str, Tensor]

refine_bboxes(*cls_scores*, *bbox_preds*, *angle_preds*, *centernesses*)

This function will be used in S2ANet, whose num_anchors=1.

class mmrotate.models.dense_heads.**CSLRRetinaHead**(*use_encoded_angle=True*, *shield_reg_angle=False*, *angle_coder={'angle_version': 'le90', 'omega': 1, 'radius': 6, 'type': 'CSLCoder', 'window': 'gaussian'}*, *loss_angle={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, *init_cfg={'layer': 'Conv2d', 'override': [{'type': 'Normal', 'name': 'retina_cls', 'std': 0.01, 'bias_prob': 0.01}, {'type': 'Normal', 'name': 'retina_angle_cls', 'std': 0.01, 'bias_prob': 0.01}], 'std': 0.01, 'type': 'Normal'}*, ***kwargs*)

Rotational Anchor-based refine head.

参数

- **use_encoded_angle** (*bool*) –Decide whether to use encoded angle or gt angle as target. Default: True.
- **shield_reg_angle** (*bool*) –Decide whether to shield the angle loss from reg branch. Default: False.
- **angle_coder** (*dict*) –Config of angle coder.
- **loss_angle** (*dict*) –Config of angle classification loss.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

forward_single(*x*)

Forward feature of a single scale level.

参数 **x** (*torch.Tensor*) –Features of a single scale level.

返回

- cls_score (torch.Tensor): Cls scores for a single scale level the channels number is num_anchors * num_classes.
- bbox_pred (torch.Tensor): Box energies / deltas for a single scale level, the channels number is num_anchors * 5.
- angle_cls (torch.Tensor): Angle for a single scale level the channels number is num_anchors * coding_len.

返回类型 tuple (torch.Tensor)

get_bboxes(*cls_scores*, *bbox_preds*, *angle_clses*, *img_metas*, *cfg=None*, *rescale=False*, *with_nms=True*)

Transform network output for a batch into bbox predictions.

参数

- **cls_scores** (*list[Tensor]*) –Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) –Box energies / deltas for each scale level with shape (N, num_anchors * 5, H, W)
- **angle_clses** (*list[Tensor]*) –Box angles for each scale level with shape (N, num_anchors * coding_len, H, W)
- **img_metas** (*list[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.
- **cfg** (*mmcv.Config | None*) –Test / postprocessing configuration, if None, test_cfg would be used
- **rescale** (*bool*) –If True, return boxes in original image space. Default: False.
- **with_nms** (*bool*) –If True, do nms before return boxes. Default: True.

返回

Each item in result_list is 2-tuple. The first item is an (n, 6) tensor, where the first 5 columns are bounding box positions (cx, cy, w, h, a) and the 6-th column is a score between 0 and 1. The second item is a (n,) tensor where each item is the predicted class label of the corresponding box.

返回类型 list[tuple[Tensor, Tensor]]

示例

```
>>> import mmcv
>>> self = AnchorHead(
>>>     num_classes=9,
>>>     in_channels=1,
>>>     anchor_generator=dict(
>>>         type='AnchorGenerator',
>>>         scales=[8],
>>>         ratios=[0.5, 1.0, 2.0],
>>>         strides=[4,]))
>>> img_metas = [{'img_shape': (32, 32, 3), 'scale_factor': 1}]
>>> cfg = mmcv.Config(dict(
>>>     score_thr=0.00,
>>>     nms=dict(type='nms', iou_thr=1.0),
>>>     max_per_img=10))
>>> feat = torch.rand(1, 1, 3, 3)
>>> cls_score, bbox_pred = self.forward_single(feat)
>>> # Note the input lists are over different levels, not images
>>> cls_scores, bbox_preds = [cls_score], [bbox_pred]
>>> result_list = self.get_bboxes(cls_scores, bbox_preds,
>>>                               img_metas, cfg)
>>> det_bboxes, det_labels = result_list[0]
>>> assert len(result_list) == 1
>>> assert det_bboxes.shape[1] == 5
>>> assert len(det_bboxes) == len(det_labels) == cfg.max_per_img
```

loss(*cls_scores*, *bbox_preds*, *angle_clses*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)

Compute losses of the head.

参数

- **cls_scores** (*list[Tensor]*) –Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) –Box energies / deltas for each scale level with shape (N, num_anchors * 5, H, W)

- **angle_clses** (*list[Tensor]*) –Box angles for each scale level with shape (N, num_anchors * coding_len, H, W)
- **gt_bboxes** (*list[Tensor]*) –Ground truth bboxes for each image with shape (num_gts, 5) in [cx, cy, w, h, a] format.
- **gt_labels** (*list[Tensor]*) –class indices corresponding to each box
- **img_metas** (*list[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | list[Tensor]*) –specify which bounding boxes can be ignored when computing the loss. Default: None

返回 A dictionary of loss components.

返回类型 dict[str, Tensor]

loss_single(*cls_score*, *bbox_pred*, *angle_cls*, *anchors*, *labels*, *label_weights*, *bbox_targets*, *bbox_weights*, *angle_targets*, *angle_weights*, *num_total_samples*)

Compute loss of a single scale level.

参数

- **cls_score** (*torch.Tensor*) –Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W).
- **bbox_pred** (*torch.Tensor*) –Box energies / deltas for each scale level with shape (N, num_anchors * 5, H, W).
- **anchors** (*torch.Tensor*) –Box reference for each scale level with shape (N, num_total_anchors, 5).
- **labels** (*torch.Tensor*) –Labels of each anchors with shape (N, num_total_anchors).
- **label_weights** (*torch.Tensor*) –Label weights of each anchor with shape (N, num_total_anchors)
- **bbox_targets** (*torch.Tensor*) –BBox regression targets of each anchor weight shape (N, num_total_anchors, 5).
- **bbox_weights** (*torch.Tensor*) –BBox regression loss weights of each anchor with shape (N, num_total_anchors, 5).
- **angle_targets** (*torch.Tensor*) –Angle classification targets of each anchor weight shape (N, num_total_anchors, coding_len).
- **angle_weights** (*torch.Tensor*) –Angle classification loss weights of each anchor with shape (N, num_total_anchors, 1).
- **num_total_samples** (*int*) –If sampling, num total samples equal to the number of total anchors; Otherwise, it is the number of positive anchors.

返回

- loss_cls (torch.Tensor): cls. loss for each scale level.
- loss_bbox (torch.Tensor): reg. loss for each scale level.
- loss_angle (torch.Tensor): angle cls. loss for each scale level.

返回类型 tuple (torch.Tensor)

class mmrotate.models.dense_heads.**KFIoUODMRefineHead**(*num_classes*, *in_channels*, *stacked_convs=2*, *conv_cfg=None*, *norm_cfg=None*, *anchor_generator={'strides': [8, 16, 32, 64, 128], 'type': 'PseudoAnchorGenerator'}*, *init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'odm_cls', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'}*, ***kwargs*)

Rotated Anchor-based refine head for KFIoU. It's a part of the Oriented Detection Module (ODM), which produces orientation-sensitive features for classification and orientation-invariant features for localization. The difference from *ODMRefineHead* is that its loss_bbox requires bbox_pred, bbox_targets, pred_decode and targets_decode as inputs.

参数

- **num_classes** (*int*) –Number of categories excluding the background category.
- **in_channels** (*int*) –Number of channels in the input feature map.
- **feat_channels** (*int*) –Number of hidden channels. Used in child classes.
- **anchor_generator** (*dict*) –Config dict for anchor generator
- **bbox_coder** (*dict*) –Config of bounding box coder.
- **reg_decoded_bbox** (*bool*) –If true, the regression loss would be applied on decoded bounding boxes. Default: False
- **background_label** (*int | None*) –Label ID of background, set as 0 for RPN and num_classes for other heads. It will automatically set as num_classes if None is given.
- **loss_cls** (*dict*) –Config of classification loss.
- **loss_bbox** (*dict*) –Config of localization loss.
- **train_cfg** (*dict*) –Training config of anchor head.
- **test_cfg** (*dict*) –Testing config of anchor head.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

forward_single(*x*)

Forward feature of a single scale level.

参数 **x** (*torch.Tensor*) –Features of a single scale level.

返回

- cls_score (torch.Tensor): Cls scores for a single scale level the channels number is num_anchors * num_classes.
- bbox_pred (torch.Tensor): Box energies / deltas for a single scale level, the channels number is num_anchors * 4.

返回类型 tuple (torch.Tensor)

get_anchors(*featmap_sizes*, *img_metas*, *device='cuda'*)

Get anchors according to feature map sizes.

参数

- **featmap_sizes** (*list[tuple]*) –Multi-level feature map sizes.
- **img_metas** (*list[dict]*) –Image meta info.
- **bboxes_as_anchors** (*list[list[Tensor]]*) –before further regression just like anchors.
- **device** (*torch.device | str*) –Device for returned tensors

返回

- anchor_list (list[Tensor]): Anchors of each image
- valid_flag_list (list[Tensor]): Valid flags of each image

返回类型 tuple

get_bboxes(*cls_scores*, *bbox_preds*, *img_metas*, *cfg=None*, *rescale=False*, *rois=None*)

Transform network output for a batch into labeled boxes.

参数

- **cls_scores** (*list[Tensor]*) –Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) –Box energies / deltas for each scale level with shape (N, num_anchors * 5, H, W)
- **img_metas** (*list[dict]*) –size / scale info for each image
- **cfg** (*mmcv.Config*) –test / postprocessing configuration
- **rescale** (*bool*) –if True, return boxes in original image space
- **rois** (*list[list[Tensor]]*) –input rbboxes of each level of each image. rois output by former stages and are to be refined.

返回

each item in result_list is 2-tuple. The first item is an (n, 6) tensor, where the first 5 columns are bounding box positions (xc, yc, w, h, a) and the 6-th column is a score between 0 and 1. The second item is a (n,) tensor where each item is the class index of the corresponding box.

返回类型 list[tuple[Tensor, Tensor]]

loss(*cls_scores*, *bbox_preds*, *gt_bboxes*, *gt_labels*, *img_metas*, *rois=None*, *gt_bboxes_ignore=None*)

Loss function of KFIoUODMRefineHead.

class mmrotate.models.dense_heads.**KFIoURRetinaHead**(*num_classes*, *in_channels*, *stacked_convs=4*, *conv_cfg=None*, *norm_cfg=None*, *anchor_generator={'octave_base_scale': 4, 'ratios': [0.5, 1.0, 2.0], 'scales_per_octave': 3, 'strides': [8, 16, 32, 64, 128], 'type': 'AnchorGenerator'}*, *init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'retina_cls', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'}*, ***kwargs*)

Rotated Anchor-based head for KFIoU. The difference from *RRetinaHead* is that its loss_bbox requires bbox_pred, bbox_targets, pred_decode and targets_decode as inputs.

参数

- **num_classes** (*int*) –Number of categories excluding the background category.
- **in_channels** (*int*) –Number of channels in the input feature map.
- **stacked_convs** (*int, optional*) –Number of stacked convolutions.
- **conv_cfg** (*dict, optional*) –Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict, optional*) –Config dict for normalization layer. Default: None.
- **anchor_generator** (*dict*) –Config dict for anchor generator
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

loss_single(*cls_score*, *bbox_pred*, *anchors*, *labels*, *label_weights*, *bbox_targets*, *bbox_weights*, *num_total_samples*)

Compute loss of a single scale level.

参数

- **cls_score** (*torch.Tensor*) –Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W).

- **bbox_pred** (*torch.Tensor*) –Box energies / deltas for each scale level with shape (N, num_anchors * 5, H, W).
- **anchors** (*torch.Tensor*) –Box reference for each scale level with shape (N, num_total_anchors, 5).
- **labels** (*torch.Tensor*) –Labels of each anchors with shape (N, num_total_anchors).
- **label_weights** (*torch.Tensor*) –Label weights of each anchor with shape (N, num_total_anchors)
- **bbox_targets** (*torch.Tensor*) –BBox regression targets of each anchor weight shape (N, num_total_anchors, 5).
- **bbox_weights** (*torch.Tensor*) –BBox regression loss weights of each anchor with shape (N, num_total_anchors, 5).
- **num_total_samples** (*int*) –If sampling, num total samples equal to the number of total anchors; Otherwise, it is the number of positive anchors.

返回

- loss_cls (torch.Tensor): cls. loss for each scale level.
- loss_bbox (torch.Tensor): reg. loss for each scale level.

返回类型 tuple (torch.Tensor)

class mmrotate.models.dense_heads.**KFIoURRetinaRefineHead**(*num_classes*, *in_channels*, *stacked_convs=4*, *conv_cfg=None*, *norm_cfg=None*, *anchor_generator={'strides': [8, 16, 32, 64, 128], 'type': 'PseudoAnchorGenerator'}*, *bbox_coder={'target_means': (0.0, 0.0, 0.0, 0.0, 0.0), 'target_stds': (1.0, 1.0, 1.0, 1.0, 1.0), 'type': 'DeltaXYWHABBoxCoder'}*, *init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'retina_cls', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'}*, ***kwargs*)

Rotational Anchor-based refine head. The difference from *RRetinaRefineHead* is that its loss_bbox requires bbox_pred, bbox_targets, pred_decode and targets_decode as inputs.

参数

- **num_classes** (*int*) –Number of categories excluding the background category.
- **in_channels** (*int*) –Number of channels in the input feature map.
- **stacked_convs** (*int, optional*) –Number of stacked convolutions.
- **conv_cfg** (*dict, optional*) –Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict, optional*) –Config dict for normalization layer. Default: None.
- **anchor_generator** (*dict*) –Config dict for anchor generator
- **bbox_coder** (*dict*) –Config of bounding box coder.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

get_anchors(*featmap_sizes*, *img_metas*, *device='cuda'*)

Get anchors according to feature map sizes.

参数

- **featmap_sizes** (*list[tuple]*) –Multi-level feature map sizes.
- **img_metas** (*list[dict]*) –Image meta info.
- **bboxes_as_anchors** (*list[list[Tensor]]*) –before further regression just like anchors.
- **device** (*torch.device | str*) –Device for returned tensors

返回

- anchor_list (list[Tensor]): Anchors of each image
- valid_flag_list (list[Tensor]): Valid flags of each image

返回类型 tuple (list[Tensor])

get_bboxes(*cls_scores*, *bbox_preds*, *img_metas*, *cfg=None*, *rescale=False*, *rois=None*)

Transform network output for a batch into labeled boxes.

参数

- **cls_scores** (*list[Tensor]*) –Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) –Box energies / deltas for each scale level with shape (N, num_anchors * 5, H, W)
- **img_metas** (*list[dict]*) –size / scale info for each image
- **cfg** (*mmcv.Config*) –test / postprocessing configuration
- **rois** (*list[list[Tensor]]*) –input rbboxes of each level of each image. rois output by former stages and are to be refined
- **rescale** (*bool*) –if True, return boxes in original image space

返回

each item in result_list is 2-tuple. The first item is an (n, 6) tensor, where the first 5 columns are bounding box positions (xc, yc, w, h, a) and the 6-th column is a score between 0 and 1. The second item is a (n,) tensor where each item is the class index of the corresponding box.

返回类型 list[tuple[Tensor, Tensor]]

loss(*cls_scores*, *bbox_preds*, *gt_bboxes*, *gt_labels*, *img_metas*, *rois=None*, *gt_bboxes_ignore=None*)

Loss function of KFIoURRetinaRefineHead.

refine_bboxes(*cls_scores*, *bbox_preds*, *rois*)

Refine predicted bounding boxes at each position of the feature maps. This method will be used in R3Det in refinement stages.

参数

- **cls_scores** (*list[Tensor]*) –Box scores for each scale level Has shape (N, num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) –Box energies / deltas for each scale level with shape (N, 5, H, W)
- **rois** (*list[list[Tensor]]*) –input rbboxes of each level of each image. rois output by former stages and are to be refined

返回 best or refined rbboxes of each level of each image.

返回类型 list[list[Tensor]]

class mmrotate.models.dense_heads.**ODMRefineHead**(*num_classes*, *in_channels*, *stacked_convs=2*, *conv_cfg=None*, *norm_cfg=None*, *anchor_generator={'strides': [8, 16, 32, 64, 128], 'type': 'PseudoAnchorGenerator'}*, *init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'odm_cls', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'}*, ***kwargs*)

Rotated Anchor-based refine head. It's a part of the Oriented Detection Module (ODM), which produces orientation-sensitive features for classification and orientation-invariant features for localization.

参数

- **num_classes** (*int*) –Number of categories excluding the background category.
- **in_channels** (*int*) –Number of channels in the input feature map.
- **stacked_convs** (*int, optional*) –Number of stacked convolutions.
- **conv_cfg** (*dict, optional*) –Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict, optional*) –Config dict for normalization layer. Default: None.

- **anchor_generator** (*dict*) –Config dict for anchor generator
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

forward_single(*x*)

Forward feature of a single scale level.

参数 **x** (*torch.Tensor*) –Features of a single scale level.

返回

- cls_score (torch.Tensor): Cls scores for a single scale level the channels number is num_anchors * num_classes.
- bbox_pred (torch.Tensor): Box energies / deltas for a single scale level, the channels number is num_anchors * 4.

返回类型 tuple (torch.Tensor)

get_anchors(*featmap_sizes*, *img_metas*, *device='cuda'*)

Get anchors according to feature map sizes.

参数

- **featmap_sizes** (*list[tuple]*) –Multi-level feature map sizes.
- **img_metas** (*list[dict]*) –Image meta info.
- **bboxes_as_anchors** (*list[list[Tensor]]*) –before further regression just like anchors.
- **device** (*torch.device | str*) –Device for returned tensors

返回

- anchor_list (list[Tensor]): Anchors of each image
- valid_flag_list (list[Tensor]): Valid flags of each image

返回类型 tuple (list[Tensor])

get_bboxes(*cls_scores*, *bbox_preds*, *img_metas*, *cfg=None*, *rescale=False*, *rois=None*)

Transform network output for a batch into labeled boxes.

参数

- **cls_scores** (*list[Tensor]*) –Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) –Box energies / deltas for each scale level with shape (N, num_anchors * 5, H, W)
- **img_metas** (*list[dict]*) –size / scale info for each image
- **cfg** (*mmcv.Config*) –test / postprocessing configuration

- **rois** (*list[list[Tensor]]*) –input rbboxes of each level of
- **image. rois output by former stages and are to be refined** (*each*) –
- **rescale** (*bool*) –if True, return boxes in original image space

返回

each item in result_list is 2-tuple. The first item is an (n, 6) tensor, where the first 5 columns are bounding box positions (xc, yc, w, h, a) and the 6-th column is a score between 0 and 1. The second item is a (n,) tensor where each item is the class index of the corresponding box.

返回类型 list[tuple[Tensor, Tensor]]

loss(*cls_scores*, *bbox_preds*, *gt_bboxes*, *gt_labels*, *img_metas*, *rois=None*, *gt_bboxes_ignore=None*)

Loss function of ODMRefineHead.

class mmrotate.models.dense_heads.**OrientedRPNHead**(*in_channels*, *init_cfg={'layer': 'Conv2d', 'std': 0.01, 'type': 'Normal'}*, *version='oc'*, ***kwargs*)

Oriented RPN head for Oriented R-CNN.

loss_single(*cls_score*, *bbox_pred*, *anchors*, *labels*, *label_weights*, *bbox_targets*, *bbox_weights*, *num_total_samples*)

Compute loss of a single scale level.

参数

- **cls_score** (*torch.Tensor*) –Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W).
- **bbox_pred** (*torch.Tensor*) –Box energies / deltas for each scale level with shape (N, num_anchors * 5, H, W).
- **anchors** (*torch.Tensor*) –Box reference for each scale level with shape (N, num_total_anchors, 4).
- **labels** (*torch.Tensor*) –Labels of each anchors with shape (N, num_total_anchors).
- **label_weights** (*torch.Tensor*) –Label weights of each anchor with shape (N, num_total_anchors)
- **bbox_targets** (*torch.Tensor*) –BBox regression targets of each anchor
- **shape** (*weight*) –
- **bbox_weights** (*torch.Tensor*) –BBox regression loss weights of each anchor with shape (N, num_total_anchors, 4).
- **num_total_samples** (*int*) –If sampling, num total samples equal to the number of total anchors; Otherwise, it is the number of positive anchors.

返回

- loss_cls (torch.Tensor): cls. loss for each scale level.
- loss_bbox (torch.Tensor): reg. loss for each scale level.

返回类型 tuple (torch.Tensor)

class mmrotate.models.dense_heads.**OrientedRepPointsHead**(*num_classes*, *in_channels*, *feat_channels*, *point_feat_channels=256*, *stacked_convs=3*, *num_points=9*, *gradient_mul=0.1*, *point_strides=[8, 16, 32, 64, 128]*, *point_base_scale=4*, *conv_bias='auto'*, *loss_cls={'alpha': 0.25, 'gamma': 2.0, 'loss_weight': 1.0, 'type': 'FocalLoss', 'use_sigmoid': True}*, *loss_bbox_init={'beta': 0.1111111111111111, 'loss_weight': 0.5, 'type': 'SmoothL1Loss'}*, *loss_bbox_refine={'beta': 0.1111111111111111, 'loss_weight': 1.0, 'type': 'SmoothL1Loss'}*, *loss_spatial_init={'loss_weight': 0.05, 'type': 'SpatialBorderLoss'}*, *loss_spatial_refine={'loss_weight': 0.1, 'type': 'SpatialBorderLoss'}*, *conv_cfg=None*, *norm_cfg=None*, *train_cfg=None*, *test_cfg=None*, *center_init=True*, *version='oc'*, *top_ratio=0.4*, *init_qua_weight=0.2*, *ori_qua_weight=0.3*, *poc_qua_weight=0.1*, *init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'reppoints_cls_out', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'}*, ***kwargs*)

Oriented RepPoints head -<https://arxiv.org/pdf/2105.11111v4.pdf>. The head contains initial and refined stages

based on RepPoints. The initial stage regresses coarse point sets, and the refine stage further regresses the fine point sets. The APAA scheme based on the quality of point set samples in the paper is employed in refined stage.

参数

- **num_classes** (*int*) –Number of classes.
- **in_channels** (*int*) –Number of input channels.
- **feat_channels** (*int*) –Number of feature channels.
- **point_feat_channels** (*int, optional*) –Number of channels of points features.
- **stacked_convs** (*int, optional*) –Number of stacked convolutions.
- **num_points** (*int, optional*) –Number of points in points set.
- **gradient_mul** (*float, optional*) –The multiplier to gradients from points refinement and recognition.
- **point_strides** (*Iterable, optional*) –points strides.
- **point_base_scale** (*int, optional*) –Bbox scale for assigning labels.
- **conv_bias** (*str, optional*) –The bias of convolution.
- **loss_cls** (*dict, optional*) –Config of classification loss.
- **loss_bbox_init** (*dict, optional*) –Config of initial points loss.
- **loss_bbox_refine** (*dict, optional*) –Config of points loss in refinement.
- **conv_cfg** (*dict, optional*) –The config of convolution.
- **norm_cfg** (*dict, optional*) –The config of normlization.
- **train_cfg** (*dict, optional*) –The config of train.
- **test_cfg** (*dict, optional*) –The config of test.
- **center_init** (*bool, optional*) –Whether to use center point assignment.
- **top_ratio** (*float, optional*) –Ratio of top high-quality point sets. Defaults to 0.4.
- **init_qua_weight** (*float, optional*) –Quality weight of initial stage.
- **ori_qua_weight** (*float, optional*) –Orientation quality weight.
- **poc_qua_weight** (*float, optional*) –Point-wise correlation quality weight.
- **version** (*str, optional*) –Angle representations. Defaults to ‘oc’.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

dynamic_pointset_samples_selection(*quality*, *label*, *label_weight*, *bbox_weight*, *pos_inds*, *pos_gt_inds*, *num_proposals_each_level=None*, *num_level=None*)

The dynamic top k selection of point set samples based on the quality assessment values.

参数

- **quality** (*torch.tensor*) –the quality values of positive point set samples
- **label** (*torch.tensor*) –gt label with shape (N)
- **bbox_gt** (*torch.tensor*) –gt bbox of polygon with shape (N, 8)
- **label_weight** (*torch.tensor*) –label weight with shape (N)
- **bbox_weight** (*torch.tensor*) –box weight with shape (N)
- **pos_inds** (*torch.tensor*) –the inds of positive point set samples
- **num_proposals_each_level** (*list[int]*) –proposals number of each level
- **num_level** (*int*) –the level number

返回

gt label with shape (N) label_weight: label weight with shape (N) bbox_weight: box weight with shape (N) num_pos (int): the number of selected positive point samples

with high-qualty

pos_normalize_term (torch.tensor): the corresponding positive normalize term

返回类型 label

feature_cosine_similarity(*points_features*)

Compute the points features similarity for points-wise correlation.

参数 points_features (*torch.tensor*) –sampling point feature with shape (N_pointsets, N_points, C)

返回

max feature similarity in each point set with shape (N_points_set, N_points, C)

返回类型 max_correlation

forward(*feats*)

Forward function.

forward_single(*x*)

Forward feature map of a single FPN level. :param x: single-level feature map sizes. :type x: torch.tensor

返回 classification score prediction pts_out_init (torch.tensor): initial point sets prediction pts_out_refine (torch.tensor): refined point sets prediction base_feat: single-level feature as the basic feature map

返回类型 cls_out (torch.tensor)

get_adaptive_points_feature(*features*, *pt_locations*, *stride*)

Get the points features from the locations of predicted points.

参数

- **features** (*torch.tensor*) –base feature with shape (B,C,W,H)
- **pt_locations** (*torch.tensor*) –locations of points in each point set with shape (B, N_points_set(number of point set), N_points(number of points in each point set) *2)

返回 sampling features with (B, C, N_points_set, N_points)

返回类型 tensor

get_bboxes(*cls_scores*, *pts_preds_init*, *pts_preds_refine*, *base_feats*, *img_metas*, *cfg=None*, *rescale=False*, *with_nms=True*, ***kwargs*)

Transform network outputs of a batch into bbox results.

参数

- **cls_scores** (*list[Tensor]*) –Classification scores for all scale levels, each is a 4D-tensor, has shape (batch_size, num_priors * num_classes, H, W).
- **pts_preds_init** (*list[Tensor]*) –Box energies / deltas for all scale levels, each is a 18D-tensor, has shape (batch_size, num_points * 2, H, W).
- **pts_preds_refine** (*list[Tensor]*) –Box energies / deltas for all scale levels, each is a 18D-tensor, has shape (batch_size, num_points * 2, H, W).
- **img_metas** (*list[dict], Optional*) –Image meta info. Default None.
- **cfg** (*mmcv.Config, Optional*) –Test / postprocessing configuration, if None, test_cfg would be used. Default None.
- **rescale** (*bool*) –If True, return boxes in original image space. Default False.
- **with_nms** (*bool*) –If True, do nms before return boxes. Default True.

返回

Each item in result_list is 2-tuple. The first item is an (n, 6) tensor, where the first 4 columns are bounding box positions (cx, cy, w, h, a) and the 6-th column is a score between 0 and 1. The second item is a (n,) tensor where each item is the predicted class label of the corresponding box.

返回类型 list[list[Tensor, Tensor]]

get_points(*featmap_sizes*, *img_metas*, *device*)

Get points according to feature map sizes.

参数

- **featmap_sizes** (*list[tuple]*) –Multi-level feature map sizes.
- **img_metas** (*list[dict]*) –Image meta info.

返回 points of each image, valid flags of each image

返回类型 tuple

get_targets(*proposals_list*, *valid_flag_list*, *gt_bboxes_list*, *img_metas*, *gt_bboxes_ignore_list=None*, *gt_labels_list=None*, *stage='init'*, *label_channels=1*, *unmap_outputs=True*)

Compute corresponding GT box and classification targets for proposals in initial stage.

参数

- **proposals_list** (*list[list]*) –Multi level points/bboxes of each image.
- **valid_flag_list** (*list[list]*) –Multi level valid flags of each image.
- **gt_bboxes_list** (*list[Tensor]*) –Ground truth bboxes of each image.
- **img_metas** (*list[dict]*) –Meta info of each image.
- **gt_bboxes_ignore_list** (*list[Tensor]*) –Ground truth bboxes to be ignored.
- **gt_bboxes_list** –Ground truth labels of each box.
- **stage** (*str*) –*init* or *refine*. Generate target for init stage or refine stage
- **label_channels** (*int*) –Channel of label.
- **unmap_outputs** (*bool*) –Whether to map outputs back to the original set of anchors.

返回

- labels_list (list[Tensor]): Labels of each level.
- label_weights_list (list[Tensor]): Label weights of each level.
- bbox_gt_list (list[Tensor]): Ground truth bbox of each level.
- proposal_list (list[Tensor]): Proposals(points/bboxes) of each level.
- proposal_weights_list (list[Tensor]): Proposal weights of each level.
- num_total_pos (int): Number of positive samples in all images.
- num_total_neg (int): Number of negative samples in all images.

返回类型 tuple (list[Tensor])

init_loss_single(*pts_pred_init*, *bbox_gt_init*, *bbox_weights_init*, *stride*)

Single initial stage loss function.

loss(*cls_scores*, *pts_preds_init*, *pts_preds_refine*, *base_features*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)

Loss function of OrientedRepPoints head.

offset_to_pts(*center_list*, *pred_list*)

Change from point offset to point coordinate.

pointsets_quality_assessment (*pts_features*, *cls_score*, *pts_pred_init*, *pts_pred_refine*, *label*, *bbox_gt*, *label_weight*, *bbox_weight*, *pos_inds*)

Assess the quality of each point set from the classification, localization, orientation, and point-wise correlation based on the assigned point sets samples. :param pts_features: points features with shape (N, 9, C) :type pts_features: torch.tensor :param cls_score: classification scores with

shape (N, class_num)

参数

- **pts_pred_init** (*torch.tensor*) –initial point sets prediction with shape (N, 9*2)
- **pts_pred_refine** (*torch.tensor*) –refined point sets prediction with shape (N, 9*2)
- **label** (*torch.tensor*) –gt label with shape (N)
- **bbox_gt** (*torch.tensor*) –gt bbox of polygon with shape (N, 8)
- **label_weight** (*torch.tensor*) –label weight with shape (N)
- **bbox_weight** (*torch.tensor*) –box weight with shape (N)
- **pos_inds** (*torch.tensor*) –the inds of positive point set samples

返回

weighted quality values for positive point set samples.

返回类型 qua (torch.tensor)

sampling_points (*polygons*, *points_num*, *device*)

Sample edge points for polygon.

参数

- **polygons** (*torch.tensor*) –polygons with shape (N, 8)
- **points_num** (*int*) –number of sampling points for each polygon edge. 10 by default.

返回

sampling points with shape (N, points_num*4, 2)

返回类型 sampling_points (torch.tensor)

class mmrotate.models.dense_heads.**RotatedATSSHead**(*num_classes*, *in_channels*, *stacked_convs=4*, *conv_cfg=None*, *norm_cfg=None*, *anchor_generator={'octave_base_scale': 4, 'ratios': [0.5, 1.0, 2.0], 'scales_per_octave': 3, 'strides': [8, 16, 32, 64, 128], 'type': 'AnchorGenerator'}*, *init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'retina_cls', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'}*, ***kwargs*)

An anchor-based head used in ATSS.

The head contains two subnetworks. The first classifies anchor boxes and the second regresses deltas for the anchors.

get_targets(*anchor_list*, *valid_flag_list*, *gt_bboxes_list*, *img_metas*, *gt_bboxes_ignore_list=None*, *gt_labels_list=None*, *label_channels=1*, *unmap_outputs=True*, *return_sampling_results=False*)

Compute regression and classification targets for anchors in multiple images.

参数

- **anchor_list** (*list[list[Tensor]]*) –Multi level anchors of each image. The outer list indicates images, and the inner list corresponds to feature levels of the image. Each element of the inner list is a tensor of shape (num_anchors, 5).
- **valid_flag_list** (*list[list[Tensor]]*) –Multi level valid flags of each image. The outer list indicates images, and the inner list corresponds to feature levels of the image. Each element of the inner list is a tensor of shape (num_anchors,)
- **gt_bboxes_list** (*list[Tensor]*) –Ground truth bboxes of each image.
- **img_metas** (*list[dict]*) –Meta info of each image.
- **gt_bboxes_ignore_list** (*list[Tensor]*) –Ground truth bboxes to be ignored.
- **gt_labels_list** (*list[Tensor]*) –Ground truth labels of each box.
- **label_channels** (*int*) –Channel of label.
- **unmap_outputs** (*bool*) –Whether to map outputs back to the original set of anchors.

返回

Usually returns a tuple containing learning targets.

- labels_list (list[Tensor]): Labels of each level.
- label_weights_list (list[Tensor]): Label weights of each level
- bbox_targets_list (list[Tensor]): BBox targets of each level

- bbox_weights_list (list[Tensor]): BBox weights of each level
- num_total_pos (int): Number of positive samples in all images
- num_total_neg (int): Number of negative samples in all images

additional_returns: This function enables user-defined returns from self._get_targets_single‘. These returns
The results will be concatenated after the end

返回类型 tuple

class mmrotate.models.dense_heads.**RotatedAnchorFreeHead**(*num_classes*, *in_channels*, *feat_channels=256*, *stacked_convs=4*, *strides=(4, 8, 16, 32, 64)*, *dcn_on_last_conv=False*, *conv_bias='auto'*, *loss_cls={'alpha': 0.25, 'gamma': 2.0, 'loss_weight': 1.0, 'type': 'FocalLoss', 'use_sigmoid': True}*, *loss_bbox={'loss_weight': 1.0, 'type': 'IoULoss'}*, *bbox_coder={'type': 'DistancePointBBoxCoder'}*, *conv_cfg=None*, *norm_cfg=None*, *train_cfg=None*, *test_cfg=None*, *init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'conv_cls', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'}*)

Rotated Anchor-free head (Rotated FCOS, etc.).

参数

- **num_classes** (*int*) –Number of categories excluding the background category.
- **in_channels** (*int*) –Number of channels in the input feature map.
- **feat_channels** (*int*) –Number of hidden channels. Used in child classes.
- **stacked_convs** (*int*) –Number of stacking convs of the head.
- **strides** (*tuple*) –Downsample factor of each feature map.
- **dcn_on_last_conv** (*bool*) –If true, use dcn in the last layer of towers. Default: False.
- **conv_bias** (*bool | str*) –If specified as *auto*, it will be decided by the norm_cfg. Bias of conv will be set as True if *norm_cfg* is None, otherwise False. Default: “auto”.

- **loss_cls** (*dict*) –Config of classification loss.
- **loss_bbox** (*dict*) –Config of localization loss.
- **bbox_coder** (*dict*) –Config of bbox coder. Defaults ‘DistancePointBBoxCoder’.
- **conv_cfg** (*dict*) –Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: None.
- **train_cfg** (*dict*) –Training config of anchor head.
- **test_cfg** (*dict*) –Testing config of anchor head.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

class mmrotate.models.dense_heads.**RotatedAnchorHead**(*num_classes*, *in_channels*, *feat_channels=256*, *anchor_generator={'octave_base_scale': 4, 'ratios': [1.0, 0.5, 2.0], 'scales_per_octave': 3, 'strides': [8, 16, 32, 64, 128], 'type': 'RotatedAnchorGenerator'}*, *bbox_coder={'target_means': (0.0, 0.0, 0.0, 0.0, 0.0), 'target_stds': (1.0, 1.0, 1.0, 1.0, 1.0), 'type': 'DeltaXYWHAOBBoxCoder'}*, *reg_decoded_bbox=False*, *assign_by_circumhbbox='oc'*, *loss_cls={'alpha': 0.25, 'gamma': 2.0, 'loss_weight': 1.0, 'type': 'FocalLoss', 'use_sigmoid': True}*, *loss_bbox={'loss_weight': 1.0, 'type': 'L1Loss'}*, *train_cfg=None*, *test_cfg=None*, *init_cfg={'layer': 'Conv2d', 'std': 0.01, 'type': 'Normal'}*)

Rotated Anchor-based head (RotatedRPN, RotatedRetinaNet, etc.).

参数

- **num_classes** (*int*) –Number of categories excluding the background category.
- **in_channels** (*int*) –Number of channels in the input feature map.
- **feat_channels** (*int*) –Number of hidden channels. Used in child classes.
- **anchor_generator** (*dict*) –Config dict for anchor generator
- **bbox_coder** (*dict*) –Config of bounding box coder.

- **reg_decoded_bbox** (*bool*) –If true, the regression loss would be applied on decoded bounding boxes. Default: False
- **assign_by_circumhbbox** (*str*) –If None, assigner will assign according to the IoU between anchor and GT (OBB), called RetinaNet-OBB. If angle definition method, assigner will assign according to the IoU between anchor and GT' s circumbox (HBB), called RetinaNet-HBB.
- **loss_cls** (*dict*) –Config of classification loss.
- **loss_bbox** (*dict*) –Config of localization loss.
- **train_cfg** (*dict*) –Training config of anchor head.
- **test_cfg** (*dict*) –Testing config of anchor head.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

aug_test(*feats*, *img_metas*, *rescale=False*)

Test det bboxes with test time augmentation, can be applied in DenseHead except for `RPNHead` and its variants, e.g., `GARPNHead`, etc.

参数

- **feats** (*list[Tensor]*) –the outer list indicates test-time augmentations and inner Tensor should have a shape NxCxHxW, which contains features for all images in the batch.
- **img_metas** (*list[list[dict]]*) –the outer list indicates test-time augs (multiscale, flip, etc.) and the inner list indicates images in a batch. each dict has image information.
- **rescale** (*bool, optional*) –Whether to rescale the results. Defaults to False.

返回

Each item in result_list is 2-tuple. The first item is `bboxes` with shape (n, 6), where 6 represent (x, y, w, h, a, score). The shape of the second tensor in the tuple is `labels` with shape (n,). The length of list should always be 1.

返回类型 list[tuple[Tensor, Tensor]]

forward(*feats*)

Forward features from the upstream network.

参数 **feats** (*tuple[Tensor]*) –Features from the upstream network, each is a 4D-tensor.

返回

A tuple of classification scores and bbox prediction.

- cls_scores (list[Tensor]): Classification scores for all scale levels, each is a 4D-tensor, the channels number is num_anchors * num_classes.
- bbox_preds (list[Tensor]): Box energies / deltas for all scale levels, each is a 4D-tensor, the channels number is num_anchors * 5.

返回类型 tuple

forward_single(*x*)

Forward feature of a single scale level.

参数 **x** (*torch.Tensor*) –Features of a single scale level.

返回

- cls_score (torch.Tensor): Cls scores for a single scale level the channels number is num_anchors * num_classes.
- bbox_pred (torch.Tensor): Box energies / deltas for a single scale level, the channels number is num_anchors * 5.

返回类型 tuple (torch.Tensor)

get_anchors(*featmap_sizes*, *img_metas*, *device='cuda'*)

Get anchors according to feature map sizes.

参数

- **featmap_sizes** (*list[tuple]*) –Multi-level feature map sizes.
- **img_metas** (*list[dict]*) –Image meta info.
- **device** (*torch.device | str*) –Device for returned tensors

返回

- anchor_list (list[Tensor]): Anchors of each image.
- valid_flag_list (list[Tensor]): Valid flags of each image.

返回类型 tuple (list[Tensor])

get_bboxes(*cls_scores*, *bbox_preds*, *img_metas*, *cfg=None*, *rescale=False*, *with_nms=True*)

Transform network output for a batch into bbox predictions.

参数

- **cls_scores** (*list[Tensor]*) –Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) –Box energies / deltas for each scale level with shape (N, num_anchors * 5, H, W)
- **img_metas** (*list[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.
- **cfg** (*mmcv.Config | None*) –Test / postprocessing configuration, if None, test_cfg would be used
- **rescale** (*bool*) –If True, return boxes in original image space. Default: False.
- **with_nms** (*bool*) –If True, do nms before return boxes. Default: True.

返回

Each item in result_list is 2-tuple. The first item is an (n, 6) tensor, where the first 5 columns are bounding box positions (cx, cy, w, h, a) and the 6-th column is a score between 0 and 1. The second item is a (n,) tensor where each item is the predicted class label of the corresponding box.

返回类型 list[tuple[Tensor, Tensor]]

示例

```
>>> import mmcv
>>> self = AnchorHead(
>>>     num_classes=9,
>>>     in_channels=1,
>>>     anchor_generator=dict(
>>>         type='AnchorGenerator',
>>>         scales=[8],
>>>         ratios=[0.5, 1.0, 2.0],
>>>         strides=[4,]))
>>> img_metas = [{'img_shape': (32, 32, 3), 'scale_factor': 1}]
>>> cfg = mmcv.Config(dict(
>>>     score_thr=0.00,
>>>     nms=dict(type='nms', iou_thr=1.0),
>>>     max_per_img=10))
>>> feat = torch.rand(1, 1, 3, 3)
>>> cls_score, bbox_pred = self.forward_single(feat)
>>> # note the input lists are over different levels, not images
>>> cls_scores, bbox_preds = [cls_score], [bbox_pred]
>>> result_list = self.get_bboxes(cls_scores, bbox_preds,
>>>                               img_metas, cfg)
>>> det_bboxes, det_labels = result_list[0]
>>> assert len(result_list) == 1
>>> assert det_bboxes.shape[1] == 5
>>> assert len(det_bboxes) == len(det_labels) == cfg.max_per_img
```

get_targets(*anchor_list*, *valid_flag_list*, *gt_bboxes_list*, *img_metas*, *gt_bboxes_ignore_list=None*, *gt_labels_list=None*, *label_channels=1*, *unmap_outputs=True*, *return_sampling_results=False*)

Compute regression and classification targets for anchors in multiple images.

参数

- **anchor_list** (*list[list[Tensor]]*) –Multi level anchors of each image. The outer list indicates images, and the inner list corresponds to feature levels of the image. Each element of the inner list is a tensor of shape (num_anchors, 5).

- **valid_flag_list** (*list[list[Tensor]]*) –Multi level valid flags of each image. The outer list indicates images, and the inner list corresponds to feature levels of the image. Each element of the inner list is a tensor of shape (num_anchors,)
- **gt_bboxes_list** (*list[Tensor]*) –Ground truth bboxes of each image.
- **img_metas** (*list[dict]*) –Meta info of each image.
- **gt_bboxes_ignore_list** (*list[Tensor]*) –Ground truth bboxes to be ignored.
- **gt_labels_list** (*list[Tensor]*) –Ground truth labels of each box.
- **label_channels** (*int*) –Channel of label.
- **unmap_outputs** (*bool*) –Whether to map outputs back to the original set of anchors.

返回

Usually returns a tuple containing learning targets.

- labels_list (list[Tensor]): Labels of each level.
- label_weights_list (list[Tensor]): Label weights of each level.
- bbox_targets_list (list[Tensor]): BBox targets of each level.
- bbox_weights_list (list[Tensor]): BBox weights of each level.
- num_total_pos (int): Number of positive samples in all images.
- num_total_neg (int): Number of negative samples in all images.

additional_returns: This function enables user-defined returns from
self._get_targets_single. These returns are currently refined to properties at each feature map (i.e. having HxW dimension). The results will be concatenated after the end

返回类型 tuple

loss(*cls_scores*, *bbox_preds*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)

Compute losses of the head.

参数

- **cls_scores** (*list[Tensor]*) –Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) –Box energies / deltas for each scale level with shape (N, num_anchors * 5, H, W)
- **gt_bboxes** (*list[Tensor]*) –Ground truth bboxes for each image with shape (num_gts, 5) in [cx, cy, w, h, a] format.
- **gt_labels** (*list[Tensor]*) –class indices corresponding to each box

- **img_metas** (*list[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | list[Tensor]*) –specify which bounding boxes can be ignored when computing the loss. Default: None

返回 A dictionary of loss components.

返回类型 dict[str, Tensor]

loss_single(*cls_score*, *bbox_pred*, *anchors*, *labels*, *label_weights*, *bbox_targets*, *bbox_weights*, *num_total_samples*)

Compute loss of a single scale level.

参数

- **cls_score** (*torch.Tensor*) –Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W).
- **bbox_pred** (*torch.Tensor*) –Box energies / deltas for each scale level with shape (N, num_anchors * 5, H, W).
- **anchors** (*torch.Tensor*) –Box reference for each scale level with shape (N, num_total_anchors, 5).
- **labels** (*torch.Tensor*) –Labels of each anchors with shape (N, num_total_anchors).
- **label_weights** (*torch.Tensor*) –Label weights of each anchor with shape (N, num_total_anchors)
- **bbox_targets** (*torch.Tensor*) –BBox regression targets of each anchor
- **shape** (*weight*) –
- **bbox_weights** (*torch.Tensor*) –BBox regression loss weights of each anchor with shape (N, num_total_anchors, 5).
- **num_total_samples** (*int*) –If sampling, num total samples equal to the number of total anchors; Otherwise, it is the number of positive anchors.

返回

- loss_cls (torch.Tensor): cls. loss for each scale level.
- loss_bbox (torch.Tensor): reg. loss for each scale level.

返回类型 tuple (torch.Tensor)

merge_aug_bboxes(*aug_bboxes*, *aug_scores*, *img_metas*)

Merge augmented detection bboxes and scores.

参数

- **aug_bboxes** (*list[Tensor]*) –shape (n, 4*#class)

- **aug_scores** (*list[Tensor] or None*) –shape (n, #class)
- **img_shapes** (*list[Tensor]*) –shape (3,).

返回 bboxes with shape (n,4), where 4 represent (tl_x, tl_y, br_x, br_y) and scores with shape (n,).

返回类型 tuple[Tensor]

class mmrotate.models.dense_heads.**RotatedFCOSHead**(*num_classes*, *in_channels*, *regress_ranges=((- 1, 64), (64, 128), (128, 256), (256, 512), (512, 100000000.0))*, *center_sampling=False*, *center_sample_radius=1.5*, *norm_on_bbox=False*, *centerness_on_reg=False*, *separate_angle=False*, *scale_angle=True*, *h_bbox_coder={'type': 'DistancePointBBoxCoder'}*, *loss_cls={'alpha': 0.25, 'gamma': 2.0, 'loss_weight': 1.0, 'type': 'FocalLoss', 'use_sigmoid': True}*, *loss_bbox={'loss_weight': 1.0, 'type': 'IoULoss'}*, *loss_angle={'loss_weight': 1.0, 'type': 'L1Loss'}*, *loss_centerness={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, *norm_cfg={'num_groups': 32, 'requires_grad': True, 'type': 'GN'}*, *init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'conv_cls', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'}*, ***kwargs*)

Anchor-free head used in FCOS. The FCOS head does not use anchor boxes. Instead bounding boxes are predicted at each pixel and a centerness measure is used to suppress low-quality predictions. Here norm_on_bbox, centerness_on_reg, dcn_on_last_conv are training tricks used in official repo, which will bring remarkable mAP gains of up to 4.9. Please see https://github.com/tianzhi0549/FCOS for more detail. :param num_classes: Number of categories excluding the background

category.

参数

- **in_channels** (*int*) –Number of channels in the input feature map.
- **strides** (*list[int] | list[tuple[int, int]]*) –Strides of points in multiple

feature levels. Default: (4, 8, 16, 32, 64).

- **regress_ranges** (*tuple[tuple[int, int]]*) –Regress range of multiple level points.
- **center_sampling** (*bool*) –If true, use center sampling. Default: False.
- **center_sample_radius** (*float*) –Radius of center sampling. Default: 1.5.
- **norm_on_bbox** (*bool*) –If true, normalize the regression targets with FPN strides. Default: False.
- **centerness_on_reg** (*bool*) –If true, position centerness on the regress branch. Please refer to https://github.com/tianzhi0549/FCOS/issues/89#issuecomment-516877042. Default: False.
- **separate_angle** (*bool*) –If true, angle prediction is separated from bbox regression loss. Default: False.
- **scale_angle** (*bool*) –If true, add scale to angle pred branch. Default: True.
- **h_bbox_coder** (*dict*) –Config of horzional bbox coder, only used when separate_angle is True.
- **conv_bias** (*bool | str*) –If specified as *auto*, it will be decided by the norm_cfg. Bias of conv will be set as True if *norm_cfg* is None, otherwise False. Default: “auto”.
- **loss_cls** (*dict*) –Config of classification loss.
- **loss_bbox** (*dict*) –Config of localization loss.
- **loss_angle** (*dict*) –Config of angle loss, only used when separate_angle is True.
- **loss_centerness** (*dict*) –Config of centerness loss.
- **norm_cfg** (*dict*) –dictionary to construct and config norm layer. Default: norm_cfg=dict(type=’GN’, num_groups=32, requires_grad=True).
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

示例

```
>>> self = RotatedFCOSHead(11, 7)
>>> feats = [torch.rand(1, 7, s, s) for s in [4, 8, 16, 32, 64]]
>>> cls_score, bbox_pred, angle_pred, centerness = self.forward(feats)
>>> assert len(cls_score) == len(self.scales)
```

centerness_target(*pos_bbox_targets*)

Compute centerness targets.

参数 **pos_bbox_targets** (*Tensor*) –BBox targets of positive bboxes in shape (num_pos, 4)

返回 Centerness target.

返回类型 Tensor

forward(*feats*)

Forward features from the upstream network. :param feats: Features from the upstream network, each is a 4D-tensor.

返回 cls_scores (list[Tensor]): Box scores for each scale level, each is a 4D-tensor, the channel number is num_points * num_classes. bbox_preds (list[Tensor]): Box energies / deltas for each scale level, each is a 4D-tensor, the channel number is num_points * 4. angle_preds (list[Tensor]): Box angle for each scale level, each is a 4D-tensor, the channel number is num_points * 1. centernesses (list[Tensor]): centerness for each scale level, each is a 4D-tensor, the channel number is num_points * 1.

返回类型 tuple

forward_single(*x*, *scale*, *stride*)

Forward features of a single scale level.

参数

- **x** (*Tensor*) –FPN feature maps of the specified stride.
- **(** (*scale*) –obj: *mmcv.cnn.Scale*): Learnable scale module to resize the bbox prediction.
- **stride** (*int*) –The corresponding stride for feature maps, only used to normalize the bbox prediction when self.norm_on_bbox is True.

返回 scores for each class, bbox predictions, angle predictions and centerness predictions of input feature maps.

返回类型 tuple

get_bboxes(*cls_scores*, *bbox_preds*, *angle_preds*, *centernesses*, *img_metas*, *cfg=None*, *rescale=None*)

Transform network output for a batch into bbox predictions.

参数

- **cls_scores** (*list[Tensor]*) –Box scores for each scale level Has shape (N, num_points * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) –Box energies / deltas for each scale level with shape (N, num_points * 4, H, W)
- **angle_preds** (*list[Tensor]*) –Box angle for each scale level with shape (N, num_points * 1, H, W)
- **centernesses** (*list[Tensor]*) –Centerness for each scale level with shape (N, num_points * 1, H, W)

- **img_metas** (*list[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.
- **cfg** (*mmcv.Config*) –Test / postprocessing configuration, if None, test_cfg would be used
- **rescale** (*bool*) –If True, return boxes in original image space

返回

Each item in result_list is 2-tuple. The first item is an (n, 6) tensor, where the first 5 columns are bounding box positions (x, y, w, h, angle) and the 6-th column is a score between 0 and 1. The second item is a (n,) tensor where each item is the predicted class label of the corresponding box.

返回类型 list[tuple[Tensor, Tensor]]

get_targets(*points*, *gt_bboxes_list*, *gt_labels_list*)

Compute regression, classification and centerness targets for points in multiple images.

参数

- **points** (*list[Tensor]*) –Points of each fpn level, each has shape (num_points, 2).
- **gt_bboxes_list** (*list[Tensor]*) –Ground truth bboxes of each image, each has shape (num_gt, 4).
- **gt_labels_list** (*list[Tensor]*) –Ground truth labels of each box, each has shape (num_gt,).

返回 concat_lvl_labels (list[Tensor]): Labels of each level. concat_lvl_bbox_targets (list[Tensor]): BBox targets of each level. concat_lvl_angle_targets (list[Tensor]): Angle targets of each level.

返回类型 tuple

loss(*cls_scores*, *bbox_preds*, *angle_preds*, *centernesses*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)

Compute loss of the head. :param cls_scores: Box scores for each scale level,

each is a 4D-tensor, the channel number is num_points * num_classes.

参数

- **bbox_preds** (*list[Tensor]*) –Box energies / deltas for each scale level, each is a 4D-tensor, the channel number is num_points * 4.
- **angle_preds** (*list[Tensor]*) –Box angle for each scale level, each is a 4D-tensor, the channel number is num_points * 1.
- **centernesses** (*list[Tensor]*) –centerness for each scale level, each is a 4D-tensor, the channel number is num_points * 1.
- **gt_bboxes** (*list[Tensor]*) –Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.

- **gt_labels** (*list[Tensor]*) –class indices corresponding to each box
- **img_metas** (*list[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | list[Tensor]*) –specify which bounding boxes can be ignored when computing the loss.

返回 A dictionary of loss components.

返回类型 dict[str, Tensor]

refine_bboxes(*cls_scores*, *bbox_preds*, *angle_preds*, *centernesses*)

This function will be used in S2ANet, whose num_anchors=1.

class mmrotate.models.dense_heads.**RotatedRPNHead**(*in_channels*, *init_cfg={'layer': 'Conv2d', 'std': 0.01, 'type': 'Normal'}*, *version='oc'*, ***kwargs*)

Rotated RPN head for rotated bboxes.

参数

- **in_channels** (*int*) –Number of channels in the input feature map.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

forward_single(*x*)

Forward feature map of a single scale level.

get_bboxes(*cls_scores*, *bbox_preds*, *img_metas*, *cfg=None*, *rescale=False*, *with_nms=True*)

Transform network output for a batch into bbox predictions.

参数

- **cls_scores** (*list[Tensor]*) –Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) –Box energies / deltas for each scale level with shape (N, num_anchors * 5, H, W)
- **img_metas** (*list[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.
- **cfg** (*mmcv.Config | None*) –Test / postprocessing configuration, if None, test_cfg would be used
- **rescale** (*bool*) –If True, return boxes in original image space. Default: False.
- **with_nms** (*bool*) –If True, do nms before return boxes. Default: True.

返回

Each item in result_list is 2-tuple. The first item is an (n, 6) tensor, where the first 5 columns are bounding box positions (cx, cy, w, h, a) and the 6-th column is a score between 0 and 1. The second item is a (n,) tensor where each item is the predicted class label of the corresponding box.

返回类型 list[tuple[Tensor, Tensor]]

get_targets(*anchor_list*, *valid_flag_list*, *gt_bboxes_list*, *img_metas*, *gt_bboxes_ignore_list=None*, *gt_labels_list=None*, *label_channels=1*, *unmap_outputs=True*, *return_sampling_results=False*)

Compute regression and classification targets for anchors in multiple images.

参数

- **anchor_list** (*list[list[Tensor]]*) –Multi level anchors of each image. The outer list indicates images, and the inner list corresponds to feature levels of the image. Each element of the inner list is a tensor of shape (num_anchors, 4).
- **valid_flag_list** (*list[list[Tensor]]*) –Multi level valid flags of each image. The outer list indicates images, and the inner list corresponds to feature levels of the image. Each element of the inner list is a tensor of shape (num_anchors,)
- **gt_bboxes_list** (*list[Tensor]*) –Ground truth bboxes of each image.
- **img_metas** (*list[dict]*) –Meta info of each image.
- **gt_bboxes_ignore_list** (*list[Tensor]*) –Ground truth bboxes to be ignored.
- **gt_labels_list** (*list[Tensor]*) –Ground truth labels of each box.
- **label_channels** (*int*) –Channel of label.
- **unmap_outputs** (*bool*) –Whether to map outputs back to the original set of anchors.

返回

Usually returns a tuple containing learning targets.

- labels_list (list[Tensor]): Labels of each level.
- label_weights_list (list[Tensor]): Label weights of each level.
- bbox_targets_list (list[Tensor]): BBox targets of each level.
- bbox_weights_list (list[Tensor]): BBox weights of each level.
- num_total_pos (int): Number of positive samples in all images.
- num_total_neg (int): Number of negative samples in all images.

additional_returns: This function enables user-defined returns from *self._get_targets_single*. These returns are currently refined to properties at each feature map (i.e. having HxW dimension). The results will be concatenated after the end

返回类型 tuple

loss(*cls_scores*, *bbox_preds*, *gt_bboxes*, *img_metas*, *gt_bboxes_ignore=None*)

Compute losses of the head.

参数

- **cls_scores** (*list[Tensor]*) –Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) –Box energies / deltas for each scale level with shape (N, num_anchors * 5, H, W)
- **gt_bboxes** (*list[Tensor]*) –Ground truth bboxes for each image with shape (num_gts, 5) in [cx, cy, w, h, a] format.
- **gt_labels** (*list[Tensor]*) –class indices corresponding to each box
- **img_metas** (*list[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*None | list[Tensor]*) –specify which bounding boxes can be ignored when computing the loss. Default: None

返回 A dictionary of loss components.

返回类型 dict[str, Tensor]

loss_single(*cls_score*, *bbox_pred*, *anchors*, *labels*, *label_weights*, *bbox_targets*, *bbox_weights*, *num_total_samples*)

Compute loss of a single scale level.

参数

- **cls_score** (*torch.Tensor*) –Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W).
- **bbox_pred** (*torch.Tensor*) –Box energies / deltas for each scale level with shape (N, num_anchors * 5, H, W).
- **anchors** (*torch.Tensor*) –Box reference for each scale level with shape (N, num_total_anchors, 4).
- **labels** (*torch.Tensor*) –Labels of each anchors with shape (N, num_total_anchors).
- **label_weights** (*torch.Tensor*) –Label weights of each anchor with shape (N, num_total_anchors)
- **bbox_targets** (*torch.Tensor*) –BBox regression targets of each anchor
- **shape** (*weight*) –
- **bbox_weights** (*torch.Tensor*) –BBox regression loss weights of each anchor with shape (N, num_total_anchors, 4).

- **num_total_samples** (*int*) –If sampling, num total samples equal to the number of total anchors; Otherwise, it is the number of positive anchors.

返回 A dictionary of loss components.

返回类型 dict[str, Tensor]

class mmrotate.models.dense_heads.**RotatedRepPointsHead**(*num_classes*, *in_channels*, *feat_channels*, *point_feat_channels=256*, *stacked_convs=3*, *num_points=9*, *gradient_mul=0.1*, *point_strides=[8, 16, 32, 64, 128]*, *point_base_scale=4*, *conv_bias='auto'*, *loss_cls={'alpha': 0.25, 'gamma': 2.0, 'loss_weight': 1.0, 'type': 'FocalLoss', 'use_sigmoid': True}*, *loss_bbox_init={'beta': 0.1111111111111111, 'loss_weight': 0.5, 'type': 'SmoothL1Loss'}*, *loss_bbox_refine={'beta': 0.1111111111111111, 'loss_weight': 1.0, 'type': 'SmoothL1Loss'}*, *conv_cfg=None*, *norm_cfg=None*, *train_cfg=None*, *test_cfg=None*, *center_init=True*, *transform_method='rotrect'*, *use_reassign=False*, *topk=6*, *anti_factor=0.75*, *version='oc'*, *init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'reppoints_cls_out', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'}*, ***kwargs*)

Rotated RepPoints head.

参数

- **num_classes** (*int*) –Number of classes.
- **in_channels** (*int*) –Number of input channels.
- **feat_channels** (*int*) –Number of feature channels.

- **point_feat_channels** (*int, optional*) –Number of channels of points features.
- **stacked_convs** (*int, optional*) –Number of stacked convolutions.
- **num_points** (*int, optional*) –Number of points in points set.
- **gradient_mul** (*float, optional*) –The multiplier to gradients from points refinement and recognition.
- **point_strides** (*Iterable, optional*) –points strides.
- **point_base_scale** (*int, optional*) –Bbox scale for assigning labels.
- **conv_bias** (*str, optional*) –The bias of convolution.
- **loss_cls** (*dict, optional*) –Config of classification loss.
- **loss_bbox_init** (*dict, optional*) –Config of initial points loss.
- **loss_bbox_refine** (*dict, optional*) –Config of points loss in refinement.
- **conv_cfg** (*dict, optional*) –The config of convolution.
- **norm_cfg** (*dict, optional*) –The config of normlization.
- **train_cfg** (*dict, optional*) –The config of train.
- **test_cfg** (*dict, optional*) –The config of test.
- **center_init** (*bool, optional*) –Whether to use center point assignment.
- **transform_method** (*str, optional*) –The methods to transform RepPoints to bbox.
- **use_reassign** (*bool, optional*) –Whether to reassign samples.
- **topk** (*int, optional*) –Number of the highest topk points. Defaults to 9.
- **anti_factor** (*float, optional*) –Feature anti-aliasing coefficient.
- **version** (*str, optional*) –Angle representations. Defaults to ‘oc’.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

forward (*feats*)

Forward function.

forward_single (*x*)

Forward feature map of a single FPN level.

get_bboxes (*cls_scores*, *pts_preds_init*, *pts_preds_refine*, *img_metas*, *cfg=None*, *rescale=False*, *with_nms=True*, ***kwargs*)

Transform network outputs of a batch into bbox results.

参数

- **cls_scores** (*list[Tensor]*) –Classification scores for all scale levels, each is a 4D-tensor, has shape (batch_size, num_priors * num_classes, H, W).

- **pts_preds_init** (*list[Tensor]*) –Box energies / deltas for all scale levels, each is a 18D-tensor, has shape (batch_size, num_points * 2, H, W).
- **pts_preds_refine** (*list[Tensor]*) –Box energies / deltas for all scale levels, each is a 18D-tensor, has shape (batch_size, num_points * 2, H, W).
- **img_metas** (*list[dict], Optional*) –Image meta info. Default None.
- **cfg** (*mmcv.Config, Optional*) –Test / postprocessing configuration, if None, test_cfg would be used. Default None.
- **rescale** (*bool*) –If True, return boxes in original image space. Default False.
- **with_nms** (*bool*) –If True, do nms before return boxes. Default True.

返回

Each item in result_list is 2-tuple. The first item is an (n, 6) tensor, where the first 4 columns are bounding box positions (cx, cy, w, h, a) and the 6-th column is a score between 0 and 1. The second item is a (n,) tensor where each item is the predicted class label of the corresponding box.

返回类型 list[list[Tensor, Tensor]]

get_cfa_targets(*proposals_list*, *valid_flag_list*, *gt_bboxes_list*, *img_metas*, *gt_bboxes_ignore_list=None*, *gt_labels_list=None*, *stage='init'*, *label_channels=1*, *unmap_outputs=True*)

Compute corresponding GT box and classification targets for proposals.

参数

- **proposals_list** (*list[list]*) –Multi level points/bboxes of each image.
- **valid_flag_list** (*list[list]*) –Multi level valid flags of each image.
- **gt_bboxes_list** (*list[Tensor]*) –Ground truth bboxes of each image.
- **img_metas** (*list[dict]*) –Meta info of each image.
- **gt_bboxes_ignore_list** (*list[Tensor]*) –Ground truth bboxes to be ignored.
- **gt_bboxes_list** –Ground truth labels of each box.
- **stage** (*str*) –*init* or *refine*. Generate target for init stage or refine stage
- **label_channels** (*int*) –Channel of label.
- **unmap_outputs** (*bool*) –Whether to map outputs back to the original set of anchors.

返回

- all_labels (list[Tensor]): Labels of each level.
- all_label_weights (list[Tensor]): Label weights of each level.
- all_bbox_gt (list[Tensor]): Ground truth bbox of each level.

- all_proposals (list[Tensor]): Proposals(points/bboxes) of each level.
- all_proposal_weights (list[Tensor]): Proposal weights of each level.
- pos_inds (list[Tensor]): Index of positive samples in all images.
- gt_inds (list[Tensor]): Index of ground truth bbox in all images.

返回类型 tuple

get_points(*featmap_sizes*, *img_metas*, *device*)

Get points according to feature map sizes.

参数

- **featmap_sizes** (*list[tuple]*) –Multi-level feature map sizes.
- **img_metas** (*list[dict]*) –Image meta info.

返回 points of each image, valid flags of each image

返回类型 tuple

get_pos_loss(*cls_score*, *pts_pred*, *label*, *bbox_gt*, *label_weight*, *convex_weight*, *pos_inds*)

Calculate loss of all potential positive samples obtained from first match process.

参数

- **cls_score** (*Tensor*) –Box scores of single image with shape (num_anchors, num_classes)
- **pts_pred** (*Tensor*) –Box energies / deltas of single image with shape (num_anchors, 4)
- **label** (*Tensor*) –classification target of each anchor with shape (num_anchors,)
- **bbox_gt** (*Tensor*) –Ground truth box.
- **label_weight** (*Tensor*) –Classification loss weight of each anchor with shape (num_anchors).
- **convex_weight** (*Tensor*) –Bbox weight of each anchor with shape (num_anchors, 4).
- **pos_inds** (*Tensor*) –Index of all positive samples got from first assign process.

返回 Losses of all positive samples in single image.

返回类型 Tensor

get_targets(*proposals_list*, *valid_flag_list*, *gt_bboxes_list*, *img_metas*, *gt_bboxes_ignore_list=None*, *gt_labels_list=None*, *stage='init'*, *label_channels=1*, *unmap_outputs=True*)

Compute corresponding GT box and classification targets for proposals.

参数

- **proposals_list** (*list[list]*) –Multi level points/bboxes of each image.
- **valid_flag_list** (*list[list]*) –Multi level valid flags of each image.

- **gt_bboxes_list** (*list[Tensor]*) –Ground truth bboxes of each image.
- **img_metas** (*list[dict]*) –Meta info of each image.
- **gt_bboxes_ignore_list** (*list[Tensor]*) –Ground truth bboxes to be ignored.
- **gt_bboxes_list** –Ground truth labels of each box.
- **stage** (*str*) –*init* or *refine*. Generate target for init stage or refine stage
- **label_channels** (*int*) –Channel of label.
- **unmap_outputs** (*bool*) –Whether to map outputs back to the original set of anchors.

返回

- labels_list (list[Tensor]): Labels of each level.
- label_weights_list (list[Tensor]): Label weights of each level.
- bbox_gt_list (list[Tensor]): Ground truth bbox of each level.
- proposal_list (list[Tensor]): Proposals(points/bboxes) of each level.
- proposal_weights_list (list[Tensor]): Proposal weights of each level.
- num_total_pos (int): Number of positive samples in all images.
- num_total_neg (int): Number of negative samples in all images.

返回类型 tuple (list[Tensor])

loss(*cls_scores*, *pts_preds_init*, *pts_preds_refine*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)

Loss function of CFA head.

loss_single(*cls_score*, *pts_pred_init*, *pts_pred_refine*, *labels*, *label_weights*, *rbbox_gt_init*, *convex_weights_init*, *rbbox_gt_refine*, *convex_weights_refine*, *stride*, *num_total_samples_refine*)

Single loss function.

offset_to_pts(*center_list*, *pred_list*)

Change from point offset to point coordinate.

points2rotrect(*pts*, *y_first=True*)

Convert points to oriented bboxes.

reassign(*pos_losses*, *label*, *label_weight*, *pts_pred_init*, *convex_weight*, *gt_bbox*, *pos_inds*, *pos_gt_inds*, *num_proposals_each_level=None*, *num_level=None*)

CFA reassign process.

参数

- **pos_losses** (*Tensor*) –Losses of all positive samples in single image.
- **label** (*Tensor*) –classification target of each anchor with shape (num_anchors,)

- **label_weight** (*Tensor*) –Classification loss weight of each anchor with shape (num_anchors).
- **pts_pred_init** (*Tensor*) –
- **convex_weight** (*Tensor*) –Bbox weight of each anchor with shape (num_anchors, 4).
- **gt_bbox** (*Tensor*) –Ground truth box.
- **pos_inds** (*Tensor*) –Index of all positive samples got from first assign process.
- **pos_gt_inds** (*Tensor*) –Gt_index of all positive samples got from first assign process.
- **num_proposals_each_level** (*list, optional*) –Number of proposals of each level.
- **num_level** (*int, optional*) –Number of level.

返回

Usually returns a tuple containing learning targets.

- label (Tensor): classification target of each anchor after paa assign, with shape (num_anchors,)
- label_weight (Tensor): Classification loss weight of each anchor after paa assign, with shape (num_anchors).
- convex_weight (Tensor): Bbox weight of each anchor with shape (num_anchors, 4).
- pos_normalize_term (list): pos normalize term for refine points losses.

返回类型 tuple

class mmrotate.models.dense_heads.**RotatedRetinaHead**(*num_classes*, *in_channels*, *stacked_convs=4*, *conv_cfg=None*, *norm_cfg=None*, *anchor_generator={'octave_base_scale': 4, 'ratios': [0.5, 1.0, 2.0], 'scales_per_octave': 3, 'strides': [8, 16, 32, 64, 128], 'type': 'AnchorGenerator'}*, *init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'retina_cls', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'}*, ***kwargs*)

An anchor-based head used in RotatedRetinaNet.

The head contains two subnetworks. The first classifies anchor boxes and the second regresses deltas for the anchors.

参数

- **num_classes** (*int*) –Number of categories excluding the background category.

- **in_channels** (*int*) –Number of channels in the input feature map.
- **stacked_convs** (*int, optional*) –Number of stacked convolutions.
- **conv_cfg** (*dict, optional*) –Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict, optional*) –Config dict for normalization layer. Default: None.
- **anchor_generator** (*dict*) –Config dict for anchor generator
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

filter_bboxes(*cls_scores*, *bbox_preds*)

Filter predicted bounding boxes at each position of the feature maps. Only one bounding boxes with highest score will be left at each position. This filter will be used in R3Det prior to the first feature refinement stage.

参数

- **cls_scores** (*list[Tensor]*) –Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) –Box energies / deltas for each scale level with shape (N, num_anchors * 5, H, W)

返回 best or refined rbboxes of each level of each image.

返回类型 list[list[Tensor]]

forward_single(*x*)

Forward feature of a single scale level.

参数 **x** (*torch.Tensor*) –Features of a single scale level.

返回

- cls_score (torch.Tensor): Cls scores for a single scale level the channels number is num_anchors * num_classes.
- bbox_pred (torch.Tensor): Box energies / deltas for a single scale level, the channels number is num_anchors * 5.

返回类型 tuple (torch.Tensor)

refine_bboxes(*cls_scores*, *bbox_preds*)

This function will be used in S2ANet, whose num_anchors=1.

参数

- **cls_scores** (*list[Tensor]*) –Box scores for each scale level Has shape (N, num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) –Box energies / deltas for each scale level with shape (N, 5, H, W)

返回 refined rbboxes of each level of each image.

返回类型 list[list[Tensor]]

class mmrotate.models.dense_heads.**RotatedRetinaRefineHead**(*num_classes*, *in_channels*, *stacked_convs=4*, *conv_cfg=None*, *norm_cfg=None*, *anchor_generator={'strides': [8, 16, 32, 64, 128], 'type': 'PseudoAnchorGenerator'}*, *bbox_coder={'target_means': (0.0, 0.0, 0.0, 0.0, 0.0), 'target_stds': (1.0, 1.0, 1.0, 1.0, 1.0), 'type': 'DeltaXYWHABBoxCoder'}*, *init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'retina_cls', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'}*, ***kwargs*)

Rotated Anchor-based refine head.

参数

- **num_classes** (*int*) –Number of categories excluding the background category.
- **in_channels** (*int*) –Number of channels in the input feature map.
- **stacked_convs** (*int, optional*) –Number of stacked convolutions.
- **conv_cfg** (*dict, optional*) –Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict, optional*) –Config dict for normalization layer. Default: None.
- **anchor_generator** (*dict*) –Config dict for anchor generator
- **bbox_coder** (*dict*) –Config of bounding box coder.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

get_anchors(*featmap_sizes*, *img_metas*, *device='cuda'*)

Get anchors according to feature map sizes.

参数

- **featmap_sizes** (*list[tuple]*) –Multi-level feature map sizes.
- **img_metas** (*list[dict]*) –Image meta info.
- **bboxes_as_anchors** (*list[list[Tensor]]*) –before further regression just like anchors.

- **device** (*torch.device | str*) –Device for returned tensors

返回

- anchor_list (list[Tensor]): Anchors of each image
- valid_flag_list (list[Tensor]): Valid flags of each image

返回类型 tuple (list[Tensor])

get_bboxes(*cls_scores*, *bbox_preds*, *img_metas*, *cfg=None*, *rescale=False*, *rois=None*)

Transform network output for a batch into labeled boxes.

参数

- **cls_scores** (*list[Tensor]*) –Box scores for each scale level Has shape (N, num_anchors * num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) –Box energies / deltas for each scale level with shape (N, num_anchors * 5, H, W)
- **img_metas** (*list[dict]*) –size / scale info for each image
- **cfg** (*mmcv.Config*) –test / postprocessing configuration
- **rois** (*list[list[Tensor]]*) –input rbboxes of each level of each image. rois output by former stages and are to be refined
- **rescale** (*bool*) –if True, return boxes in original image space

返回

each item in result_list is 2-tuple. The first item is an (n, 6) tensor, where the first 5 columns are bounding box positions (xc, yc, w, h, a) and the 6-th column is a score between 0 and 1. The second item is a (n,) tensor where each item is the class index of the corresponding box.

返回类型 list[tuple[Tensor, Tensor]]

loss(*cls_scores*, *bbox_preds*, *gt_bboxes*, *gt_labels*, *img_metas*, *rois=None*, *gt_bboxes_ignore=None*)

Loss function of RotatedRetinaRefineHead.

refine_bboxes(*cls_scores*, *bbox_preds*, *rois*)

Refine predicted bounding boxes at each position of the feature maps. This method will be used in R3Det in refinement stages.

参数

- **cls_scores** (*list[Tensor]*) –Box scores for each scale level Has shape (N, num_classes, H, W)
- **bbox_preds** (*list[Tensor]*) –Box energies / deltas for each scale level with shape (N, 5, H, W)

- **rois** (*list[list[Tensor]]*) –input rbboxes of each level of each image. rois output by former stages and are to be refined

返回 best or refined rbboxes of each level of each image.

返回类型 list[list[Tensor]]

class mmrotate.models.dense_heads.**SAMRepPointsHead**(*num_classes*, *in_channels*, *feat_channels*, *point_feat_channels=256*, *stacked_convs=3*, *num_points=9*, *gradient_mul=0.1*, *point_strides=[8, 16, 32, 64, 128]*, *point_base_scale=4*, *conv_bias='auto'*, *loss_cls={'alpha': 0.25, 'gamma': 2.0, 'loss_weight': 1.0, 'type': 'FocalLoss', 'use_sigmoid': True}*, *loss_bbox_init={'beta': 0.1111111111111111, 'loss_weight': 0.5, 'type': 'SmoothL1Loss'}*, *loss_bbox_refine={'beta': 0.1111111111111111, 'loss_weight': 1.0, 'type': 'SmoothL1Loss'}*, *conv_cfg=None*, *norm_cfg=None*, *train_cfg=None*, *test_cfg=None*, *center_init=True*, *transform_method='rotrect'*, *topk=6*, *anti_factor=0.75*, *version='oc'*, *init_cfg={'layer': 'Conv2d', 'override': {'bias_prob': 0.01, 'name': 'reppoints_cls_out', 'std': 0.01, 'type': 'Normal'}, 'std': 0.01, 'type': 'Normal'}*, ***kwargs*)

Rotated RepPoints head for SASM.

参数

- **num_classes** (*int*) –Number of classes.
- **in_channels** (*int*) –Number of input channels.
- **feat_channels** (*int*) –Number of feature channels.
- **point_feat_channels** (*int, optional*) –Number of channels of points features.
- **stacked_convs** (*int, optional*) –Number of stacked convolutions.
- **num_points** (*int, optional*) –Number of points in points set.
- **gradient_mul** (*float, optional*) –The multiplier to gradients from points refinement and recognition.

- **point_strides** (*Iterable, optional*) –points strides.
- **point_base_scale** (*int, optional*) –Bbox scale for assigning labels.
- **conv_bias** (*str, optional*) –The bias of convolution.
- **loss_cls** (*dict, optional*) –Config of classification loss.
- **loss_bbox_init** (*dict, optional*) –Config of initial points loss.
- **loss_bbox_refine** (*dict, optional*) –Config of points loss in refinement.
- **conv_cfg** (*dict, optional*) –The config of convolution.
- **norm_cfg** (*dict, optional*) –The config of normlization.
- **train_cfg** (*dict, optional*) –The config of train.
- **test_cfg** (*dict, optional*) –The config of test.
- **center_init** (*bool, optional*) –Whether to use center point assignment.
- **transform_method** (*str, optional*) –The methods to transform RepPoints to bbox.
- **topk** (*int, optional*) –Number of the highest topk points. Defaults to 9.
- **anti_factor** (*float, optional*) –Feature anti-aliasing coefficient.
- **version** (*str, optional*) –Angle representations. Defaults to 'oc'.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

forward(*feats*)

Forward function.

forward_single(*x*)

Forward feature map of a single FPN level.

get_bboxes(*cls_scores*, *pts_preds_init*, *pts_preds_refine*, *img_metas*, *cfg=None*, *rescale=False*, *with_nms=True*, ***kwargs*)

Transform network outputs of a batch into bbox results.

参数

- **cls_scores** (*list[Tensor]*) –Classification scores for all scale levels, each is a 4D-tensor, has shape (batch_size, num_priors * num_classes, H, W).
- **pts_preds_init** (*list[Tensor]*) –Box energies / deltas for all scale levels, each is a 18D-tensor, has shape (batch_size, num_points * 2, H, W).
- **pts_preds_refine** (*list[Tensor]*) –Box energies / deltas for all scale levels, each is a 18D-tensor, has shape (batch_size, num_points * 2, H, W).
- **img_metas** (*list[dict], Optional*) –Image meta info. Default None.

- **cfg** (*mmcv.Config, Optional*) –Test / postprocessing configuration, if None, test_cfg would be used. Default None.
- **rescale** (*bool*) –If True, return boxes in original image space. Default False.
- **with_nms** (*bool*) –If True, do nms before return boxes. Default True.

返回

Each item in result_list is 2-tuple. The first item is an (n, 6) tensor, where the first 4 columns are bounding box positions (cx, cy, w, h, a) and the 6-th column is a score between 0 and 1. The second item is a (n,) tensor where each item is the predicted class label of the corresponding box.

返回类型 list[list[Tensor, Tensor]]

get_points(*featmap_sizes*, *img_metas*, *device*)

Get points according to feature map sizes.

参数

- **featmap_sizes** (*list[tuple]*) –Multi-level feature map sizes.
- **img_metas** (*list[dict]*) –Image meta info.

返回 points of each image, valid flags of each image

返回类型 tuple

get_targets(*proposals_list*, *valid_flag_list*, *gt_bboxes_list*, *img_metas*, *gt_bboxes_ignore_list=None*, *gt_labels_list=None*, *stage='init'*, *label_channels=1*, *unmap_outputs=True*)

Compute corresponding GT box and classification targets for proposals.

参数

- **proposals_list** (*list[list]*) –Multi level points/bboxes of each image.
- **valid_flag_list** (*list[list]*) –Multi level valid flags of each image.
- **gt_bboxes_list** (*list[Tensor]*) –Ground truth bboxes of each image.
- **img_metas** (*list[dict]*) –Meta info of each image.
- **gt_bboxes_ignore_list** (*list[Tensor]*) –Ground truth bboxes to be ignored.
- **gt_bboxes_list** –Ground truth labels of each box.
- **stage** (*str*) –*init* or *refine*. Generate target for init stage or refine stage
- **label_channels** (*int*) –Channel of label.
- **unmap_outputs** (*bool*) –Whether to map outputs back to the original set of anchors.

返回

- labels_list (list[Tensor]): Labels of each level.

- label_weights_list (list[Tensor]): Label weights of each level.
- bbox_gt_list (list[Tensor]): Ground truth bbox of each level.
- proposal_list (list[Tensor]): Proposals(points/bboxes) of each level.
- proposal_weights_list (list[Tensor]): Proposal weights of each level.
- num_total_pos (int): Number of positive samples in all images.
- num_total_neg (int): Number of negative samples in all images.

返回类型 tuple (list[Tensor])

loss(*cls_scores*, *pts_preds_init*, *pts_preds_refine*, *gt_bboxes*, *gt_labels*, *img_metas*, *gt_bboxes_ignore=None*)

Loss function of SAM RepPoints head.

loss_single(*cls_score*, *pts_pred_init*, *pts_pred_refine*, *labels*, *label_weights*, *rbbox_gt_init*, *convex_weights_init*, *sam_weights_init*, *rbbox_gt_refine*, *convex_weights_refine*, *sam_weights_refine*, *stride*, *num_total_samples_refine*)

Single loss function.

offset_to_pts(*center_list*, *pred_list*)

Change from point offset to point coordinate.

points2rotrect(*pts*, *y_first=True*)

Convert points to oriented bboxes.

26.5 roi_heads

class mmrotate.models.roi_heads.**GVRatioRoIHead**(*bbox_roi_extractor=None*, *bbox_head=None*, *shared_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*, *version='oc'*)

Gliding vertex roi head including one bbox head.

forward_dummy(*x*, *proposals*)

Dummy forward function.

参数

- **x** (*list[Tensors]*) –list of multi-level img features.
- **proposals** (*list[Tensors]*) –list of region proposals.

返回 list of region of interest.

返回类型 list[Tensors]

simple_test_bboxes(*x*, *img_metas*, *proposals*, *rcnn_test_cfg*, *rescale=False*)

Test only det bboxes without augmentation.

参数

- **x** (*tuple[Tensor]*) –Feature maps of all scale level.
- **img_metas** (*list[dict]*) –Image meta info.
- **proposals** (*List[Tensor]*) –Region proposals.
- **(obj** (*rcnn_test_cfg*) –*ConfigDict*): *test_cfg* of R-CNN.
- **rescale** (*bool*) –If True, return boxes in original image space. Default: False.

返回 The first list contains the boxes of the corresponding image in a batch, each tensor has the shape (num_boxes, 5) and last dimension 5 represent (cx, cy, w, h, a, score). Each Tensor in the second list is the labels with shape (num_boxes,). The length of both lists should be equal to batch_size.

返回类型 tuple[list[Tensor], list[Tensor]]

class mmrotate.models.roi_heads.**OrientedStandardRoIHead**(*bbox_roi_extractor=None*, *bbox_head=None*, *shared_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*, *version='oc'*)

Oriented RCNN roi head including one bbox head.

forward_dummy(*x*, *proposals*)

Dummy forward function.

参数

- **x** (*list[Tensors]*) –list of multi-level img features.
- **proposals** (*list[Tensors]*) –list of region proposals.

返回 list of region of interest.

返回类型 list[Tensors]

forward_train(*x*, *img_metas*, *proposal_list*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*, *gt_masks=None*)

参数

- **x** (*list[Tensor]*) –list of multi-level img features.
- **img_metas** (*list[dict]*) –list of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmdet/datasets/pipelines/formatting.py:Collect*.
- **proposals** (*list[Tensors]*) –list of region proposals.

- **gt_bboxes** (*list[Tensor]*) –Ground truth bboxes for each image with shape (num_gts, 5) in [cx, cy, w, h, a] format.
- **gt_labels** (*list[Tensor]*) –class indices corresponding to each box
- **gt_bboxes_ignore** (*None | list[Tensor]*) –specify which bounding boxes can be ignored when computing the loss.
- **gt_masks** (*None | Tensor*) –true segmentation masks for each box used if the architecture supports a segmentation task. Always set to None.

返回 a dictionary of loss components

返回类型 dict[str, Tensor]

simple_test_bboxes(*x*, *img_metas*, *proposals*, *rcnn_test_cfg*, *rescale=False*)

Test only det bboxes without augmentation.

参数

- **x** (*tuple[Tensor]*) –Feature maps of all scale level.
- **img_metas** (*list[dict]*) –Image meta info.
- **proposals** (*List[Tensor]*) –Region proposals.
- **(obj** (*rcnn_test_cfg*) –*ConfigDict*): *test_cfg* of R-CNN.
- **rescale** (*bool*) –If True, return boxes in original image space. Default: False.

返回 The first list contains the boxes of the corresponding image in a batch, each tensor has the shape (num_boxes, 5) and last dimension 5 represent (cx, cy, w, h, a, score). Each Tensor in the second list is the labels with shape (num_boxes,). The length of both lists should be equal to batch_size.

返回类型 tuple[list[Tensor], list[Tensor]]

class mmrotate.models.roi_heads.**RoITransRoIHead**(*num_stages*, *stage_loss_weights*, *bbox_roi_extractor=None*, *bbox_head=None*, *shared_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *version='oc'*, *init_cfg=None*)

RoI Trans cascade roi head including one bbox head.

参数

- **num_stages** (*int*) –number of cascade stages.
- **stage_loss_weights** (*list[float]*) –loss weights of cascade stages.
- **bbox_roi_extractor** (*dict, optional*) –Config of bbox_roi_extractor.
- **bbox_head** (*dict, optional*) –Config of bbox_head.

- **shared_head** (*dict, optional*) –Config of shared_head.
- **train_cfg** (*dict, optional*) –Config of train.
- **test_cfg** (*dict, optional*) –Config of test.
- **pretrained** (*str, optional*) –Path of pretrained weight.
- **version** (*str, optional*) –Angle representations. Defaults to 'oc'.
- **init_cfg** (*dict, optional*) –Config of initialization.

aug_test(*features*, *proposal_list*, *img_metas*, *rescale=False*)

Test with augmentations.

forward_dummy(*x*, *proposals*)

Dummy forward function.

参数

- **x** (*list[Tensors]*) –list of multi-level img features.
- **proposals** (*list[Tensors]*) –list of region proposals.

返回 list of region of interest.

返回类型 list[Tensors]

forward_train(*x*, *img_metas*, *proposal_list*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*, *gt_masks=None*)

参数

- **x** (*list[Tensor]*) –list of multi-level img features.
- **img_metas** (*list[dict]*) –list of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmdet/datasets/pipelines/formatting.py:Collect*.
- **proposals** (*list[Tensors]*) –list of region proposals.
- **gt_bboxes** (*list[Tensor]*) –Ground truth bboxes for each image with shape (num_gts, 5) in [cx, cy, w, h, a] format.
- **gt_labels** (*list[Tensor]*) –class indices corresponding to each box
- **gt_bboxes_ignore** (*None | list[Tensor]*) –specify which bounding boxes can be ignored when computing the loss.
- **gt_masks** (*None | Tensor*) –true segmentation masks for each box used if the architecture supports a segmentation task. Always set to None.

返回 a dictionary of loss components

返回类型 dict[str, Tensor]

init_assigner_sampler()

Initialize assigner and sampler for each stage.

init_bbox_head(*bbox_roi_extractor*, *bbox_head*)

Initialize box head and box roi extractor.

参数

- **bbox_roi_extractor** (*dict*) –Config of box roi extractor.
- **bbox_head** (*dict*) –Config of box in box head.

simple_test(*x*, *proposal_list*, *img_metas*, *rescale=False*)

Test without augmentation.

参数

- **x** (*list[Tensor]*) –list of multi-level img features.
- **proposal_list** (*list[Tensors]*) –list of region proposals.
- **img_metas** (*list[dict]*) –list of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'.
- **rescale** (*bool*) –If True, return boxes in original image space. Default: False.

返回 a dictionary of bbox_results.

返回类型 dict[str, Tensor]

class mmrotate.models.roi_heads.**RotatedBBoxHead**(*with_avg_pool=False*, *with_cls=True*, *with_reg=True*, *roi_feat_size=7*, *in_channels=256*, *num_classes=80*, *bbox_coder={'clip_border': True, 'target_means': [0.0, 0.0, 0.0, 0.0], 'target_stds': [0.1, 0.1, 0.2, 0.2], 'type': 'DeltaXYWHBBoxCoder'}*, *reg_class_agnostic=False*, *reg_decoded_bbox=False*, *reg_predictor_cfg={'type': 'Linear'}*, *cls_predictor_cfg={'type': 'Linear'}*, *loss_cls={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_sigmoid': False}*, *loss_bbox={'beta': 1.0, 'loss_weight': 1.0, 'type': 'SmoothL1Loss'}*, *init_cfg=None*)

Simplest RoI head, with only two fc layers for classification and regression respectively.

参数

- **with_avg_pool** (*bool, optional*) –If True, use avg_pool.

- **with_cls** (*bool, optional*) –If True, use classification branch.
- **with_reg** (*bool, optional*) –If True, use regression branch.
- **roi_feat_size** (*int, optional*) –Size of RoI features.
- **in_channels** (*int, optional*) –Input channels.
- **num_classes** (*int, optional*) –Number of classes.
- **bbox_coder** (*dict, optional*) –Config of bbox coder.
- **reg_class_agnostic** (*bool, optional*) –If True, regression branch are class agnostic.
- **reg_decoded_bbox** (*bool, optional*) –If True, regression branch use decoded bbox to compute loss.
- **reg_predictor_cfg** (*dict, optional*) –Config of regression predictor.
- **cls_predictor_cfg** (*dict, optional*) –Config of classification predictor.
- **loss_cls** (*dict, optional*) –Config of classification loss.
- **loss_bbox** (*dict, optional*) –Config of regression loss.
- **init_cfg** (*dict, optional*) –Config of initialization.

property custom_accuracy

The custom accuracy.

property custom_activation

The custom activation.

property custom_cls_channels

The custom cls channels.

forward(*x*)

Forward function of Rotated BBoxHead.

get_bboxes(*rois*, *cls_score*, *bbox_pred*, *img_shape*, *scale_factor*, *rescale=False*, *cfg=None*)

Transform network output for a batch into bbox predictions.

参数

- **rois** (*torch.Tensor*) –Boxes to be transformed. Has shape (num_boxes, 5). last dimension 5 arrange as (batch_index, x1, y1, x2, y2).
- **cls_score** (*torch.Tensor*) –Box scores, has shape (num_boxes, num_classes + 1).
- **bbox_pred** (*Tensor, optional*) –Box energies / deltas. has shape (num_boxes, num_classes * 5).
- **img_shape** (*Sequence[int], optional*) –Maximum bounds for boxes, specifies (H, W, C) or (H, W).

- **scale_factor** (*ndarray*) –Scale factor of the image arrange as (w_scale, h_scale, w_scale, h_scale).
- **rescale** (*bool*) –If True, return boxes in original image space. Default: False.
- **(obj** (*cfg*) –*ConfigDict*): *test_cfg* of Bbox Head. Default: None

返回 First tensor is *det_bboxes*, has the shape (num_boxes, 6) and last dimension 6 represent (cx, cy, w, h, a, score). Second tensor is the labels with shape (num_boxes,).

返回类型 tuple[Tensor, Tensor]

get_targets(*sampling_results*, *gt_bboxes*, *gt_labels*, *rcnn_train_cfg*, *concat=True*)

Calculate the ground truth for all samples in a batch according to the sampling_results.

Almost the same as the implementation in bbox_head, we passed additional parameters pos_inds_list and neg_inds_list to *_get_target_single* function.

参数

- **(List[obj** (*sampling_results*) –SamplingResults]): Assign results of all images in a batch after sampling.
- **gt_bboxes** (*list[Tensor]*) –Gt_bboxes of all images in a batch, each tensor has shape (num_gt, 5), the last dimension 5 represents [cx, cy, w, h, a].
- **gt_labels** (*list[Tensor]*) –Gt_labels of all images in a batch, each tensor has shape (num_gt,).
- **(obj** (*rcnn_train_cfg*) –ConfigDict): *train_cfg* of RCNN.
- **concat** (*bool*) –Whether to concatenate the results of all the images in a single batch.

返回

Ground truth for proposals in a single image. Containing the following list of Tensors:

- labels (list[Tensor],Tensor): Gt_labels for all proposals in a batch, each tensor in list has shape (num_proposals,) when *concat=False*, otherwise just a single tensor has shape (num_all_proposals,).
- label_weights (list[Tensor]): Labels_weights for all proposals in a batch, each tensor in list has shape (num_proposals,) when *concat=False*, otherwise just a single tensor has shape (num_all_proposals,).
- bbox_targets (list[Tensor],Tensor): Regression target for all proposals in a batch, each tensor in list has shape (num_proposals, 5) when *concat=False*, otherwise just a single tensor has shape (num_all_proposals, 5), the last dimension 4 represents [cx, cy, w, h, a].
- bbox_weights (list[tensor],Tensor): Regression weights for all proposals in a batch, each tensor in list has shape (num_proposals, 5) when *concat=False*, otherwise just a single tensor has shape (num_all_proposals, 5).

返回类型 Tuple[Tensor]

loss(*cls_score*, *bbox_pred*, *rois*, *labels*, *label_weights*, *bbox_targets*, *bbox_weights*, *reduction_override=None*)

Loss function.

参数

- **cls_score** (*torch.Tensor*) –Box scores, has shape (num_boxes, num_classes + 1).
- **bbox_pred** (*Tensor, optional*) –Box energies / deltas. has shape (num_boxes, num_classes * 5).
- **rois** (*torch.Tensor*) –Boxes to be transformed. Has shape (num_boxes, 5). last dimension 5 arrange as (batch_index, x1, y1, x2, y2).
- **labels** (*torch.Tensor*) –Shape (n*bs,).
- **label_weights** (*torch.Tensor*) –Labels_weights for all proposals, has shape (num_proposals,).
- **bbox_targets** (*torch.Tensor*) –Regression target for all proposals, has shape (num_proposals, 5), the last dimension 5 represents [cx, cy, w, h, a].
- **bbox_weights** (*list[tensor],Tensor*) –Regression weights for all proposals in a batch, each tensor in list has shape (num_proposals, 5) when *concat=False*, otherwise just a single tensor has shape (num_all_proposals, 5).
- **reduction_override** (*str, optional*) –The reduction method used to override the original reduction method of the loss. Defaults to None.

refine_bboxes(*rois*, *labels*, *bbox_preds*, *pos_is_gts*, *img_metas*)

Refine bboxes during training.

参数

- **rois** (*torch.Tensor*) –Shape (n*bs, 5), where n is image number per GPU, and bs is the sampled RoIs per image. The first column is the image id and the next 4 columns are x1, y1, x2, y2.
- **labels** (*torch.Tensor*) –Shape (n*bs,).
- **bbox_preds** (*torch.Tensor*) –Shape (n*bs, 5) or (n*bs, 5*#class).
- **pos_is_gts** (*list[Tensor]*) –Flags indicating if each positive bbox is a gt bbox.
- **img_metas** (*list[dict]*) –Meta info of each image.

返回 Refined bboxes of each image in a mini-batch.

返回类型 list[Tensor]

regress_by_class(*rois*, *label*, *bbox_pred*, *img_meta*)

Regress the bbox for the predicted class. Used in Cascade R-CNN.

参数

- **rois** (*torch.Tensor*) –shape (n, 4) or (n, 5)
- **label** (*torch.Tensor*) –shape (n,)
- **bbox_pred** (*torch.Tensor*) –shape (n, 5*(#class)) or (n, 5)
- **img_meta** (*dict*) –Image meta info.

返回 Regressed bboxes, the same shape as input rois.

返回类型 Tensor

class mmrotate.models.roi_heads.**RotatedConvFCBBoxHead**(*num_shared_convs=0*, *num_shared_fcs=0*, *num_cls_convs=0*, *num_cls_fcs=0*, *num_reg_convs=0*, *num_reg_fcs=0*, *conv_out_channels=256*, *fc_out_channels=1024*, *conv_cfg=None*, *norm_cfg=None*, *init_cfg=None*, **args*, ***kwargs*)

More general bbox head, with shared conv and fc layers and two optional separated branches.

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

参数

- **num_shared_convs** (*int, optional*) –number of `shared_convs`.
- **num_shared_fcs** (*int, optional*) –number of `shared_fcs`.
- **num_cls_convs** (*int, optional*) –number of `cls_convs`.
- **num_cls_fcs** (*int, optional*) –number of `cls_fcs`.
- **num_reg_convs** (*int, optional*) –number of `reg_convs`.
- **num_reg_fcs** (*int, optional*) –number of `reg_fcs`.
- **conv_out_channels** (*int, optional*) –output channels of convolution.
- **fc_out_channels** (*int, optional*) –output channels of fc.
- **conv_cfg** (*dict, optional*) –Config of convolution.
- **norm_cfg** (*dict, optional*) –Config of normalization.
- **init_cfg** (*dict, optional*) –Config of initialization.

forward(*x*)

Forward function.

class mmrotate.models.roi_heads.**RotatedShared2FCBBoxHead**(*fc_out_channels=1024*, **args*, ***kwargs*)

Shared2FC RBBox head.

class mmrotate.models.roi_heads.**RotatedSingleRoIExtractor**(*roi_layer*, *out_channels*, *featmap_strides*, *finest_scale=56*, *init_cfg=None*)

Extract RoI features from a single level feature map.

If there are multiple input feature levels, each RoI is mapped to a level according to its scale. The mapping rule is proposed in FPN.

参数

- **roi_layer** (*dict*) –Specify RoI layer type and arguments.
- **out_channels** (*int*) –Output channels of RoI layers.
- **featmap_strides** (*List[int]*) –Strides of input feature maps.
- **finest_scale** (*int*) –Scale threshold of mapping to level 0. Default: 56.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None

build_roi_layers(*layer_cfg*, *featmap_strides*)

Build RoI operator to extract feature from each level feature map.

参数

- **layer_cfg** (*dict*) –Dictionary to construct and config RoI layer operation. Options are modules under `mmcv/ops` such as `RoIAlign`.
- **featmap_strides** (*List[int]*) –The stride of input feature map w.r.t to the original image size, which would be used to scale RoI coordinate (original image coordinate system) to feature coordinate system.

返回 The RoI extractor modules for each level feature map.

返回类型 nn.ModuleList

forward(*feats*, *rois*, *roi_scale_factor=None*)

Forward function.

参数

- **feats** (*torch.Tensor*) –Input features.
- **rois** (*torch.Tensor*) –Input RoIs, shape (k, 5).

- **scale_factor** (*float*) –Scale factor that RoI will be multiplied by.

返回 Scaled RoI features.

返回类型 torch.Tensor

map_roi_levels(*rois*, *num_levels*)

Map rois to corresponding feature levels by scales.

- scale < finest_scale * 2: level 0
- finest_scale * 2 <= scale < finest_scale * 4: level 1
- finest_scale * 4 <= scale < finest_scale * 8: level 2
- scale >= finest_scale * 8: level 3

参数

- **rois** (*torch.Tensor*) –Input RoIs, shape (k, 5).
- **num_levels** (*int*) –Total level number.

返回 Level index (0-based) of each RoI, shape (k,)

返回类型 Tensor

roi_rescale(*rois*, *scale_factor*)

Scale RoI coordinates by scale factor.

参数

- **rois** (*torch.Tensor*) –RoI (Region of Interest), shape (n, 6)
- **scale_factor** (*float*) –Scale factor that RoI will be multiplied by.

返回 Scaled RoI.

返回类型 torch.Tensor

class mmrotate.models.roi_heads.**RotatedStandardRoIHead**(*bbox_roi_extractor=None*, *bbox_head=None*, *shared_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*, *version='oc'*)

Simplest base rotated roi head including one bbox head.

参数

- **bbox_roi_extractor** (*dict, optional*) –Config of bbox_roi_extractor.
- **bbox_head** (*dict, optional*) –Config of bbox_head.
- **shared_head** (*dict, optional*) –Config of shared_head.

- **train_cfg** (*dict, optional*) –Config of train.
- **test_cfg** (*dict, optional*) –Config of test.
- **pretrained** (*str, optional*) –Path of pretrained weight.
- **init_cfg** (*dict, optional*) –Config of initialization.
- **version** (*str, optional*) –Angle representations. Defaults to 'oc'.

async async_simple_test(*x*, *proposal_list*, *img_metas*, *rescale=False*)

Async test without augmentation.

参数

- **x** (*list[Tensor]*) –list of multi-level img features.
- **proposal_list** (*list[Tensors]*) –list of region proposals.
- **img_metas** (*list[dict]*) –list of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'.
- **rescale** (*bool*) –If True, return boxes in original image space. Default: False.

返回 a dictionary of bbox_results.

返回类型 dict[str, Tensor]

aug_test(*x*, *proposal_list*, *img_metas*, *rescale=False*)

Test with augmentations.

forward_dummy(*x*, *proposals*)

Dummy forward function.

参数

- **x** (*list[Tensors]*) –list of multi-level img features.
- **proposals** (*list[Tensors]*) –list of region proposals.

返回 list of region of interest.

返回类型 list[Tensors]

forward_train(*x*, *img_metas*, *proposal_list*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*, *gt_masks=None*)

参数

- **x** (*list[Tensor]*) –list of multi-level img features.
- **img_metas** (*list[dict]*) –list of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'. For details on the values of these keys see *mmdet/datasets/pipelines/formatting.py:Collect*.

- **proposals** (*list[Tensors]*) –list of region proposals.
- **gt_bboxes** (*list[Tensor]*) –Ground truth bboxes for each image with shape (num_gts, 5) in [cx, cy, w, h, a] format.
- **gt_labels** (*list[Tensor]*) –class indices corresponding to each box
- **gt_bboxes_ignore** (*None | list[Tensor]*) –specify which bounding boxes can be ignored when computing the loss.
- **gt_masks** (*None | Tensor*) –true segmentation masks for each box used if the architecture supports a segmentation task. Always set to None.

返回 a dictionary of loss components.

返回类型 dict[str, Tensor]

init_assigner_sampler()

Initialize assigner and sampler.

init_bbox_head(*bbox_roi_extractor*, *bbox_head*)

Initialize `bbox_head`.

参数

- **bbox_roi_extractor** (*dict*) –Config of `bbox_roi_extractor`.
- **bbox_head** (*dict*) –Config of `bbox_head`.

simple_test(*x*, *proposal_list*, *img_metas*, *rescale=False*)

Test without augmentation.

参数

- **x** (*list[Tensor]*) –list of multi-level img features.
- **proposal_list** (*list[Tensors]*) –list of region proposals.
- **img_metas** (*list[dict]*) –list of image info dict where each dict has: 'img_shape', 'scale_factor', 'flip', and may also contain 'filename', 'ori_shape', 'pad_shape', and 'img_norm_cfg'.
- **rescale** (*bool*) –If True, return boxes in original image space. Default: False.

返回 a dictionary of bbox_results.

返回类型 dict[str, Tensor]

simple_test_bboxes(*x*, *img_metas*, *proposals*, *rcnn_test_cfg*, *rescale=False*)

Test only det bboxes without augmentation.

参数

- **x** (*tuple[Tensor]*) –Feature maps of all scale level.
- **img_metas** (*list[dict]*) –Image meta info.

- **proposals** (*List[Tensor]*) –Region proposals.
- **(obj** (*rcnn_test_cfg*) –*ConfigDict*): *test_cfg* of R-CNN.
- **rescale** (*bool*) –If True, return boxes in original image space. Default: False.

返回 The first list contains the boxes of the corresponding image in a batch, each tensor has the shape (num_boxes, 5) and last dimension 5 represent (tl_x, tl_y, br_x, br_y, score). Each Tensor in the second list is the labels with shape (num_boxes,). The length of both lists should be equal to batch_size.

返回类型 tuple[list[Tensor], list[Tensor]]

26.6 losses

class mmrotate.models.losses.**BCConvexGIoULoss**(*reduction='mean'*, *loss_weight=1.0*)

BCConvex GIoU loss.

Computing the BCConvex GIoU loss between a set of predicted convexes and target convexes.

参数

- **reduction** (*str, optional*) –The reduction method of the loss. Defaults to ‘mean’.
- **loss_weight** (*float, optional*) –The weight of loss. Defaults to 1.0.

返回 Loss tensor.

返回类型 torch.Tensor

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*, ***kwargs*)

Forward function.

参数

- **pred** (*torch.Tensor*) –Predicted convexes.
- **target** (*torch.Tensor*) –Corresponding gt convexes.
- **weight** (*torch.Tensor, optional*) –The weight of loss for each prediction. Defaults to None.
- **avg_factor** (*int, optional*) –Average factor that is used to average the loss. Defaults to None.
- **reduction_override** (*str, optional*) –The reduction method used to override the original reduction method of the loss. Defaults to None.

class mmrotate.models.losses.**ConvexGIoULoss**(*reduction='mean'*, *loss_weight=1.0*)

Convex GIoU loss.

Computing the Convex GIoU loss between a set of predicted convexes and target convexes.

参数

- **reduction** (*str, optional*) –The reduction method of the loss. Defaults to ‘mean’.
- **loss_weight** (*float, optional*) –The weight of loss. Defaults to 1.0.

返回 Loss tensor.

返回类型 torch.Tensor

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*, ***kwargs*)

Forward function.

参数

- **pred** (*torch.Tensor*) –Predicted convexes.
- **target** (*torch.Tensor*) –Corresponding gt convexes.
- **weight** (*torch.Tensor, optional*) –The weight of loss for each prediction. Defaults to None.
- **avg_factor** (*int, optional*) –Average factor that is used to average the loss. Defaults to None.
- **reduction_override** (*str, optional*) –The reduction method used to override the original reduction method of the loss. Defaults to None.

class mmrotate.models.losses.**GDLoss**(*loss_type*, *representation='xy_wh_r'*, *fun='log1p'*, *tau=0.0*, *alpha=1.0*, *reduction='mean'*, *loss_weight=1.0*, ***kwargs*)

Gaussian based loss.

参数

- **loss_type** (*str*) –Type of loss.
- **representation** (*str, optional*) –Coordinate System.
- **fun** (*str, optional*) –The function applied to distance. Defaults to ‘log1p’.
- **tau** (*float, optional*) –Defaults to 1.0.
- **alpha** (*float, optional*) –Defaults to 1.0.
- **reduction** (*str, optional*) –The reduction method of the loss. Defaults to ‘mean’.
- **loss_weight** (*float, optional*) –The weight of loss. Defaults to 1.0.

返回 loss (torch.Tensor)

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*, ***kwargs*)

Forward function.

参数

- **pred** (*torch.Tensor*) –Predicted convexes.
- **target** (*torch.Tensor*) –Corresponding gt convexes.
- **weight** (*torch.Tensor, optional*) –The weight of loss for each prediction. Defaults to None.
- **avg_factor** (*int, optional*) –Average factor that is used to average the loss. Defaults to None.
- **reduction_override** (*str, optional*) –The reduction method used to override the original reduction method of the loss. Defaults to None.

class mmrotate.models.losses.**GDLoss_v1**(*loss_type*, *fun='sqrt'*, *tau=1.0*, *reduction='mean'*, *loss_weight=1.0*, ***kwargs*)

Gaussian based loss.

参数

- **loss_type** (*str*) –Type of loss.
- **fun** (*str, optional*) –The function applied to distance. Defaults to ‘log1p’.
- **tau** (*float, optional*) –Defaults to 1.0.
- **reduction** (*str, optional*) –The reduction method of the loss. Defaults to ‘mean’.
- **loss_weight** (*float, optional*) –The weight of loss. Defaults to 1.0.

返回 loss (torch.Tensor)

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*, ***kwargs*)

Forward function.

参数

- **pred** (*torch.Tensor*) –Predicted convexes.
- **target** (*torch.Tensor*) –Corresponding gt convexes.
- **weight** (*torch.Tensor, optional*) –The weight of loss for each prediction. Defaults to None.
- **avg_factor** (*int, optional*) –Average factor that is used to average the loss. Defaults to None.
- **reduction_override** (*str, optional*) –The reduction method used to override the original reduction method of the loss. Defaults to None.

class mmrotate.models.losses.**KFLoss**(*fun='none'*, *reduction='mean'*, *loss_weight=1.0*, ***kwargs*)

Kalman filter based loss.

参数

- **fun** (*str, optional*) –The function applied to distance. Defaults to ‘log1p’.
- **reduction** (*str, optional*) –The reduction method of the loss. Defaults to ‘mean’.
- **loss_weight** (*float, optional*) –The weight of loss. Defaults to 1.0.

返回 loss (torch.Tensor)

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *pred_decode=None*, *targets_decode=None*, *reduction_override=None*, ***kwargs*)

Forward function.

参数

- **pred** (*torch.Tensor*) –Predicted convexes.
- **target** (*torch.Tensor*) –Corresponding gt convexes.
- **weight** (*torch.Tensor, optional*) –The weight of loss for each prediction. Defaults to None.
- **avg_factor** (*int, optional*) –Average factor that is used to average the loss. Defaults to None.
- **pred_decode** (*torch.Tensor*) –Predicted decode bboxes.
- **targets_decode** (*torch.Tensor*) –Corresponding gt decode bboxes.
- **reduction_override** (*str, optional*) –The reduction method used to override the original reduction method of the loss. Defaults to None.

返回 loss (torch.Tensor)

class mmrotate.models.losses.**KLDRepPointsLoss**(*eps=1e-06*, *reduction='mean'*, *loss_weight=1.0*)

Kullback-Leibler Divergence loss for RepPoints.

参数

- **eps** (*float*) –Defaults to 1e-6.
- **reduction** (*str, optional*) –The reduction method of the loss. Defaults to ‘mean’.
- **loss_weight** (*float, optional*) –The weight of loss. Defaults to 1.0.

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*, ***kwargs*)

Forward function.

参数

- **pred** (*torch.Tensor*) –Predicted convexes.
- **target** (*torch.Tensor*) –Corresponding gt convexes.

- **weight** (*torch.Tensor, optional*) –The weight of loss for each prediction. Defaults to None.
- **avg_factor** (*int, optional*) –Average factor that is used to average the loss. Defaults to None.
- **reduction_override** (*str, optional*) –The reduction method used to override the original reduction method of the loss. Defaults to None.

返回 loss (torch.Tensor)

class mmrotate.models.losses.**RotatedIoULoss**(*linear=False*, *eps=1e-06*, *reduction='mean'*, *loss_weight=1.0*, *mode='log'*)

RotatedIoULoss.

Computing the IoU loss between a set of predicted rbboxes and target rbboxes. :param linear: If True, use linear scale of loss else determined

by mode. Default: False.

参数

- **eps** (*float*) –Eps to avoid log(0).
- **reduction** (*str*) –Options are "none", "mean" and "sum".
- **loss_weight** (*float*) –Weight of loss.
- **mode** (*str*) –Loss scaling mode, including "linear", "square", and "log". Default: 'log'

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*, ***kwargs*)

Forward function.

参数

- **pred** (*torch.Tensor*) –The prediction.
- **target** (*torch.Tensor*) –The learning target of the prediction.
- **weight** (*torch.Tensor, optional*) –The weight of loss for each prediction. Defaults to None.
- **avg_factor** (*int, optional*) –Average factor that is used to average the loss. Defaults to None.
- **reduction_override** (*str, optional*) –The reduction method used to override the original reduction method of the loss. Defaults to None. Options are "none", "mean" and "sum".

class mmrotate.models.losses.**SmoothFocalLoss**(*gamma=2.0*, *alpha=0.25*, *reduction='mean'*, *loss_weight=1.0*)

Smooth Focal Loss. Implementation of Circular Smooth Label (CSL).

参数

- **gamma** (*float, optional*) –The gamma for calculating the modulating factor. Defaults to 2.0.
- **alpha** (*float, optional*) –A balanced form for Focal Loss. Defaults to 0.25.
- **reduction** (*str, optional*) –The method used to reduce the loss into a scalar. Defaults to 'mean'. Options are "none", "mean" and "sum".
- **loss_weight** (*float, optional*) –Weight of loss. Defaults to 1.0.

返回 loss (torch.Tensor)

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*)

Forward function.

参数

- **pred** (*torch.Tensor*) –The prediction.
- **target** (*torch.Tensor*) –The learning label of the prediction.
- **weight** (*torch.Tensor, optional*) –The weight of loss for each prediction. Defaults to None.
- **avg_factor** (*int, optional*) –Average factor that is used to average the loss. Defaults to None.
- **reduction_override** (*str, optional*) –The reduction method used to override the original reduction method of the loss. Options are "none", "mean" and "sum".

返回 The calculated loss

返回类型 torch.Tensor

class mmrotate.models.losses.**SpatialBorderLoss**(*loss_weight=1.0*)

Spatial Border loss for learning points in Oriented RepPoints.

参数

- **pts** (*torch.Tensor*) –point sets with shape (N, 9*2).
- **points number in each point set is 9.** (*Default*) –
- **gt_bboxes** (*torch.Tensor*) –gt_bboxes with polygon form with shape(N, 8)

返回 loss (torch.Tensor)

forward(*pts*, *gt_bboxes*, *weight*, **args*, ***kwargs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

26.7 utils

class mmrotate.models.utils.**ORConv2d**(*in_channels*, *out_channels*, *kernel_size=3*, *arf_config=None*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*)

Oriented 2-D convolution.

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*) –The size of kernel.
- **arf_config** (*tuple, optional*) –a tuple consist of nOrientation and nRotation.
- **stride** (*int, optional*) –Stride of the convolution. Default: 1.
- **padding** (*int or tuple*) –Zero-padding added to both sides of the input. Default: 0.
- **dilation** (*int or tuple*) –Spacing between kernel elements. Default: 1.
- **groups** (*int*) –Number of blocked connections from input. channels to output channels. Default: 1.
- **bias** (*bool*) –If True, adds a learnable bias to the output. Default: False.

forward(*input*)

Forward function.

get_indices()

Get the indices of ORConv2d.

reset_parameters()

Reset the parameters of ORConv2d.

rotate_arf()

Build active rotating filter module.

class mmrotate.models.utils.**RotationInvariantPooling**(*nInputPlane*, *nOrientation=8*)

Rotating invariant pooling module.

参数

- **nInputPlane** (*int*) –The number of Input plane.
- **nOrientation** (*int, optional*) –The number of oriented channels.

forward(*x*)

Forward function.

mmrotate.models.utils.**build_enn_divide_feature**(*planes*)

build a enn regular feature map with the specified number of channels divided by N.

mmrotate.models.utils.**build_enn_feature**(*planes*)

build a enn regular feature map with the specified number of channels.

mmrotate.models.utils.**build_enn_norm_layer**(*num_features*, *postfix=''*)

build an enn normalizion layer.

mmrotate.models.utils.**build_enn_trivial_feature**(*planes*)

build a enn trivial feature map with the specified number of channels.

mmrotate.models.utils.**ennAvgPool**(*inplanes*, *kernel_size=1*, *stride=None*, *padding=0*, *ceil_mode=False*)

enn Average Pooling.

参数

- **inplanes** (*int*) –The number of input channel.
- **kernel_size** (*int, optional*) –The size of kernel.
- **stride** (*int, optional*) –Stride of the convolution. Default: 1.
- **padding** (*int or tuple*) –Zero-padding added to both sides of the input. Default: 0.
- **ceil_mode** (*bool, optional*) –if True, keep information in the corner of feature map.

mmrotate.models.utils.**ennConv**(*inplanes*, *outplanes*, *kernel_size=3*, *stride=1*, *padding=0*, *groups=1*, *bias=False*, *dilation=1*)

enn convolution.

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*) –The size of kernel.
- **stride** (*int, optional*) –Stride of the convolution. Default: 1.
- **padding** (*int or tuple*) –Zero-padding added to both sides of the input. Default: 0.
- **groups** (*int*) –Number of blocked connections from input. channels to output channels. Default: 1.
- **bias** (*bool*) –If True, adds a learnable bias to the output. Default: False.
- **dilation** (*int or tuple*) –Spacing between kernel elements. Default: 1.

mmrotate.models.utils.**ennInterpolate**(*inplanes*, *scale_factor*, *mode='nearest'*, *align_corners=False*)

enn Interpolate.

mmrotate.models.utils.**ennMaxPool**(*inplanes*, *kernel_size*, *stride=1*, *padding=0*)

enn Max Pooling.

mmrotate.models.utils.**ennReLU**(*inplanes*)

enn ReLU.

mmrotate.models.utils.**ennTrivialConv**(*inplanes*, *outplanes*, *kernel_size=3*, *stride=1*, *padding=0*, *groups=1*, *bias=False*, *dilation=1*)

enn convolution with trivial input featurn.

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*) –The size of kernel.
- **stride** (*int, optional*) –Stride of the convolution. Default: 1.
- **padding** (*int or tuple*) –Zero-padding added to both sides of the input. Default: 0.
- **groups** (*int*) –Number of blocked connections from input. channels to output channels. Default: 1.
- **bias** (*bool*) –If True, adds a learnable bias to the output. Default: False.
- **dilation** (*int or tuple*) –Spacing between kernel elements. Default: 1.

CHAPTER 27

mmrotate.utils

mmrotate.utils.**collect_env**()

Collect environment information.

mmrotate.utils.**compat_cfg**(*cfg*)

This function would modify some filed to keep the compatibility of config.

For example, it will move some args which will be deprecated to the correct fields.

mmrotate.utils.**find_latest_checkpoint**(*path*, *suffix='pth'*)

Find the latest checkpoint from the working directory.

参数

- **path** (*str*) –The path to find checkpoints.
- **suffix** (*str*) –File extension. Defaults to pth.

返回 File path of the latest checkpoint.

返回类型 latest_path(str | None)

引用

mmrotate.utils.**get_root_logger**(*log_file=None*, *log_level=20*)

Get root logger.

参数

- **log_file** (*str, optional*) –File path of log. Defaults to None.
- **log_level** (*int, optional*) –The level of logger. Defaults to logging.INFO.

返回 The obtained logger

返回类型 logging.Logger

mmrotate.utils.**setup_multi_processes**(*cfg*)

Setup multi-processing environment variables.

CHAPTER 28

Indices and tables

- genindex
- search

Python 模块索引

m

索引

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

W

模块