mmcv Documentation

发布 *1.0.0rc4*

MMCV Contributors

2022 年 08 月 17 日

开始你的第一步

CHAPTER 1

依赖

MMDetection3D 可以安装在 Linux, MacOS, (实验性支持 Windows) 的平台上，它具体需要下列安装包:

- Python 3.6+
- PyTorch 1.3+
- CUDA 9.2+ (如果你从源码编译 PyTorch, CUDA 9.0 也是兼容的。)
- GCC 5+
- MMCV

注解: 如果你已经装了 pytorch, 可以跳过这一部分，然后转到下一章节. 如果没有，可以参照以下步骤安装环境。

步骤 0. 安装 MiniConda 官网.

步骤 1. 使用 conda 新建虚拟环境，并进入该虚拟环境.

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

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

GPU 环境下

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

CPU 环境下

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

CHAPTER 2

安装

我们建议用户参照我们的最佳实践 MMDetection3D。不过，整个过程也是可定制化的，具体可参照自定义安装章节

2.1 最佳实践

如果你已经成功安装 CUDA 11.0，那么你可以使用这个快速安装命令进行 MMDetection3D 的安装。否则，则参考下一小节的详细安装流程。

```
pip install openmim
mim install mmcv-full
mim install mmdet
mim install mmsegmentation
git clone https://github.com/open-mmlab/mmdetection3d.git
cd mmdetection3d
pip install -e .
```

步骤 0. 通过MIM 安装 MMCV.

步骤 1. 安装 MMDetection.

```
pip install mmdet
```

同时，如果你想修改这部分的代码，也可以通过以下命令从源码编译 MMDetection：

```
git clone https://github.com/open-mmlab/mmdetection.git
cd mmdetection
git checkout v2.24.0  # switch to v2.24.0 branch
pip install -r requirements/build.txt
pip install -v -e .  # or "python setup.py develop"
```

步骤 2. 安装 MMSegmentation.

```
pip install mmsegmentation
```

同时，如果你想修改这部分的代码，也可以通过以下命令从源码编译 MMSegmentation：

```
git clone https://github.com/open-mmlab/mmsegmentation.git
cd mmsegmentation
git checkout v0.20.0  # switch to v0.20.0 branch
pip install -e .  # or "python setup.py develop"
```

步骤 3. 克隆 MMDetection3D 代码仓库.

```
git clone https://github.com/open-mmlab/mmdetection3d.git
cd mmdetection3d
```

步骤 4. 安装依赖包和 MMDetection3D.

```
pip install -v -e .  # or "python setup.py develop"
```

注意：

1. Git 的 commit id 在步骤 d 将会被写入到版本号当中，例 0.6.0+2e7045c 。版本号将保存在训练的模型里。推荐在每一次执行步骤 d 时，从 github 上获取最新的更新。如果基于 C++/CUDA 的代码被修改了，请执行以下步骤；

   重要: 如果你重装了不同版本的 CUDA 或者 PyTorch 的 mmdet，请务必移除 `./build` 文件。

   ```
   pip uninstall mmdet3d
   rm -rf ./build
   find . -name "*.so" | xargs rm
   ```

2. 按照上述说明，MMDetection3D 安装在 `dev` 模式下，因此在本地对代码做的任何修改都会生效，无需重新安装；

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

4. 一些安装依赖是可以选择的。例如只需要安装最低运行要求的版本，则可以使用 `pip install -v -e .` 命令。如果希望使用可选择的像 `albumentations` 和 `imagecorruptions` 这种依赖项，可以使用 `pip install -r requirements/optional.txt` 进行手动安装，或者在使用 `pip` 时指定

所需的附加功能（例如 `pip install -v -e .[optional]`），支持附加功能的有效键值包括 `all`、`tests`、`build` 以及 `optional` 。

我们已经支持 spconv2.0. 如果用户已经安装 spconv 2.0，代码会默认使用 spconv 2.0。它可以比原生 mmcv spconv 使用更少的内存。用户可以使用下列的命令来安装 spconv 2.0.

```
pip install cumm-cuxxx
pip install spconv-cuxxx
```

xxx 表示 CUDA 的版本。

例如，使用 CUDA 10.2，对应命令是 `pip install cumm-cu102 && pip install spconv-cu102`.

支持的 CUDA 版本包括 10.2, 11.1, 11.3, and 11.4. 用户可以通过源码编译来在这些版本上安装. 具体细节请参考 spconv v2.x.

我们同时也支持 Minkowski Engine 来作为稀疏卷机的后端. 如果需要，可以参照 安装指南 或使用 `pip`:

```
conda install openblas-devel -c anaconda
pip install -U git+https://github.com/NVIDIA/MinkowskiEngine -v --no-deps --
↪install-option="--blas_include_dirs=/opt/conda/include" --install-option="--
↪blas=openblas"
```

5. 我们的代码目前不能在只有 CPU 的环境（CUDA 不可用）下编译运行。

CHAPTER 3

验证

3.1 通过点云样例程序来验证

我们提供了一些样例脚本去测试单个样本，预训练的模型可以从模型库中下载. 运行如下命令可以去测试点云场景下一个单模态的 3D 检测算法。

```
python demo/pcd_demo.py ${PCD_FILE} ${CONFIG_FILE} ${CHECKPOINT_FILE} [--device ${GPU_
↪ID}] [--score-thr ${SCORE_THR}] [--out-dir ${OUT_DIR}]
```

例:

```
python demo/pcd_demo.py demo/data/kitti/kitti_000008.bin configs/second/hv_second_
↪secfpn_6x8_80e_kitti-3d-car.py checkpoints/hv_second_secfpn_6x8_80e_kitti-3d-car_
↪20200620_230238-393f000c.pth
```

如果你想输入一个 ply 格式的文件，你可以使用如下函数将它转换为 bin 的文件格式。然后就可以使用转化成 bin 格式的文件去运行样例程序。

请注意在使用此脚本前，你需要先安装 pandas 和 plyfile。这个函数也可使用在数据预处理当中，为了能够直接训练 ply data。

```
import numpy as np
import pandas as pd
from plyfile import PlyData
```

(下页继续)

(续上页)

```
def convert_ply(input_path, output_path):
    plydata = PlyData.read(input_path)  # read file
    data = plydata.elements[0].data  # read data
    data_pd = pd.DataFrame(data)  # convert to DataFrame
    data_np = np.zeros(data_pd.shape, dtype=np.float)  # initialize array to store␣
↪data
    property_names = data[0].dtype.names  # read names of properties
    for i, name in enumerate(
            property_names):  # read data by property
        data_np[:, i] = data_pd[name]
    data_np.astype(np.float32).tofile(output_path)
```

例:

```
convert_ply('./test.ply', './test.bin')
```

如果你有其他格式的点云文件 (例：`off`, `obj`), 你可以使用 `trimesh` 将它们转化成 `ply`.

```
import trimesh

def to_ply(input_path, output_path, original_type):
    mesh = trimesh.load(input_path, file_type=original_type)  # read file
    mesh.export(output_path, file_type='ply')  # convert to ply
```

例:

```
to_ply('./test.obj', './test.ply', 'obj')
```

更多的关于单/多模态和室内/室外的 3D 检测的样例可以在此找到.

3.2 测试点云的高级接口

3.2.1 同步接口

这里有一个例子去说明如何构建模型以及测试给出的点云：

```
from mmdet3d.apis import init_model, inference_detector

config_file = 'configs/votenet/votenet_8x8_scannet-3d-18class.py'
checkpoint_file = 'checkpoints/votenet_8x8_scannet-3d-18class_20200620_230238-
↪2cea9c3a.pth'
```

(下页继续)

(续上页)

```
# 从配置文件和预训练的模型文件中构建模型
model = init_model(config_file, checkpoint_file, device='cuda:0')

# 测试单个文件并可视化结果
point_cloud = 'test.bin'
result, data = inference_detector(model, point_cloud)
# 可视化结果并且将结果保存到 'results' 文件夹
model.show_results(data, result, out_dir='results')
```

3.3 自定义安装

3.3.1 CUDA 版本

当安装 PyTorch 的时候，你需要去指定 CUDA 的版本。如果你不清楚如何选择 CUDA 的版本，可以参考我们如下的建议：

- 对于 Ampere 的 NVIDIA GPU, 比如 GeForce 30 series 和 NVIDIA A100, CUDA 11 是必须的。
- 对于老款的 NVIDIA GPUs, CUDA 11 是可编译的，但是 CUDA 10.2 提供更好的可编译性，并且更轻量。

请确保 GPU 驱动版本大于最低需求。这个表格 提供更多的信息。

注解： 如果你参照最佳实践，你只需要安装 CUDA runtime libraries。这是因为没有代码需要在本地通过 CUDA 编译。然而如果你需要编译 MMCV 源码，或者编译其他 CUDA 代码，你需要基于 NVIDIA website 安装完整的 CUDA toolkit，并且要保证它的版本跟 PyTorch 匹配。比如在 ‘conda install‘ 里对应的 cudatoolkit 版本。

3.3.2 不通过 MIM 安装 MMCV

MMCV 包含一些 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。但是，这不是必须的。

如果想要使用 pip 而不是 MIM 安装 MMCV, 请参考 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, 下面的脚本安装的 mmcv-full 是对应的 PyTorch 1.10.x 和 CUDA 11.3.

```
pip install mmcv-full -f https://download.openmmlab.com/mmcv/dist/cu113/torch1.10/
↪index.html
```

3.3.3 通过 Docker 安装

我们提供了 Dockerfile 来建立一个镜像。

```
# 基于 PyTorch 1.6, CUDA 10.1 生成 docker 的镜像
docker build -t mmdetection3d docker/
```

运行命令:

```
docker run --gpus all --shm-size=8g -it -v {DATA_DIR}:/mmdetection3d/data␣
↪mmdetection3d
```

3.4 从零开始的安装脚本

以下是一个基于 conda 安装 MMdetection3D 的脚本

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

# install latest PyTorch prebuilt with the default prebuilt CUDA version (usually the␣
↪latest)
conda install -c pytorch pytorch torchvision -y

# install mmcv
pip install mmcv-full

# install mmdetection
pip install git+https://github.com/open-mmlab/mmdetection.git

# install mmsegmentation
pip install git+https://github.com/open-mmlab/mmsegmentation.git

# install mmdetection3d
git clone https://github.com/open-mmlab/mmdetection3d.git
cd mmdetection3d
pip install -v -e .
```

3.5 故障排除

如果在安装过程中遇到什么问题，可以先参考*FAQ* 页面. 如果没有找到对应的解决方案，你也可以在 Github 提一个 issue。

CHAPTER 4

样例

CHAPTER 5

介绍

我们提供了多模态/单模态（基于激光雷达/图像）、室内/室外场景的 3D 检测和 3D 语义分割样例的脚本，预训练模型可以从 Model Zoo 下载。我们也提供了 KITTI、SUN RGB-D、nuScenes 和 ScanNet 数据集的预处理样本数据，你可以根据我们的预处理步骤使用任何其它数据。

5.1 测试

5.1.1 3D 检测

单模态样例

在点云数据上测试 3D 检测器，运行：

```
python demo/pcd_demo.py ${PCD_FILE} ${CONFIG_FILE} ${CHECKPOINT_FILE} [--device ${GPU_
↪ID}] [--score-thr ${SCORE_THR}] [--out-dir ${OUT_DIR}] [--show]
```

点云和预测 3D 框的可视化结果会被保存在 `${OUT_DIR}/PCD_NAME`，它可以使用 MeshLab 打开。注意如果你设置了 `--show`，通过 Open3D 可以在线显示预测结果。

在 KITTI 数据上测试 SECOND 模型：

```
python demo/pcd_demo.py demo/data/kitti/kitti_000008.bin configs/second/hv_second_
↪secfpn_6x8_80e_kitti-3d-car.py checkpoints/hv_second_secfpn_6x8_80e_kitti-3d-car_
↪20200620_230238-393f000c.pth
```

在 SUN RGB-D 数据上测试 VoteNet 模型：

```
python demo/pcd_demo.py demo/data/sunrgbd/sunrgbd_000017.bin configs/votenet/votenet_
↪16x8_sunrgbd-3d-10class.py checkpoints/votenet_16x8_sunrgbd-3d-10class_20200620_
↪230238-4483c0c0.pth
```

如果你正在使用的 mmdetection3d 版本 >= 0.6.0，记住转换 VoteNet 的模型权重文件，查看 README 来获取转换模型权重文件的详细说明。

多模态样例

在多模态数据（通常是点云和图像）上测试 3D 检测器，运行：

```
python demo/multi_modality_demo.py ${PCD_FILE} ${IMAGE_FILE} ${ANNOTATION_FILE} $
↪{CONFIG_FILE} ${CHECKPOINT_FILE} [--device ${GPU_ID}] [--score-thr ${SCORE_THR}] [--
↪out-dir ${OUT_DIR}] [--show]
```

ANNOTATION_FILE 需要提供 3D 到 2D 的仿射矩阵，可视化结果会被保存在 ${OUT_DIR}/PCD_NAME，其中包括点云、图像、预测的 3D 框以及它们在图像上的投影。

在 KITTI 数据上测试 MVX-Net 模型：

```
python demo/multi_modality_demo.py demo/data/kitti/kitti_000008.bin demo/data/kitti/
↪kitti_000008.png demo/data/kitti/kitti_000008_infos.pkl configs/mvxnet/dv_mvx-fpn_
↪second_secfpn_adamw_2x8_80e_kitti-3d-3class.py checkpoints/dv_mvx-fpn_second_secfpn_
↪adamw_2x8_80e_kitti-3d-3class_20200621_003904-10140f2d.pth
```

在 SUN RGB-D 数据上测试 ImVoteNet 模型：

```
python demo/multi_modality_demo.py demo/data/sunrgbd/sunrgbd_000017.bin demo/data/
↪sunrgbd/sunrgbd_000017.jpg demo/data/sunrgbd/sunrgbd_000017_infos.pkl configs/
↪imvotenet/imvotenet_stage2_16x8_sunrgbd-3d-10class.py checkpoints/imvotenet_stage2_
↪16x8_sunrgbd-3d-10class_20210323_184021-d44dcb66.pth
```

5.1.2 单目 3D 检测

在图像数据上测试单目 3D 检测器，运行：

```
python demo/mono_det_demo.py ${IMAGE_FILE} ${ANNOTATION_FILE} ${CONFIG_FILE} $
↪{CHECKPOINT_FILE} [--device ${GPU_ID}] [--out-dir ${OUT_DIR}] [--show]
```

ANNOTATION_FILE 需要提供 3D 到 2D 的仿射矩阵（相机内参矩阵），可视化结果会被保存在 ${OUT_DIR}/PCD_NAME，其中包括图像以及预测 3D 框在图像上的投影。

在 nuScenes 数据上测试 FCOS3D 模型：

```
python demo/mono_det_demo.py demo/data/nuscenes/n015-2018-07-24-11-22-45+0800__CAM_
↪BACK__1532402927637525.jpg demo/data/nuscenes/n015-2018-07-24-11-22-45+0800__CAM_
↪BACK__1532402927637525_mono3d.coco.json configs/fcos3d/fcos3d_r101_caffe_fpn_gn-
↪head_dcn_2x8_1x_nus-mono3d_finetune.py checkpoints/fcos3d_r101_caffe_fpn_gn-head_
↪dcn_2x8_1x_nus-mono3d_finetune_20210717_095645-8d806dc2.pth
```

5.1.3 3D 分割

在点云数据上测试 3D 分割器，运行：

```
python demo/pc_seg_demo.py ${PCD_FILE} ${CONFIG_FILE} ${CHECKPOINT_FILE} [--device $
↪{GPU_ID}] [--out-dir ${OUT_DIR}] [--show]
```

可视化结果会被保存在 `${OUT_DIR}/PCD_NAME`，其中包括点云以及预测的 3D 分割掩码。

在 ScanNet 数据上测试 PointNet++ (SSG) 模型：

```
python demo/pc_seg_demo.py demo/data/scannet/scene0000_00.bin configs/pointnet2/
↪pointnet2_ssg_16x2_cosine_200e_scannet_seg-3d-20class.py checkpoints/pointnet2_ssg_
↪16x2_cosine_200e_scannet_seg-3d-20class_20210514_143644-ee73704a.pth
```

CHAPTER 6

模型库

6.1 通用设置

- 使用分布式训练；
- 为了和其他代码库做公平对比，本文展示的是使用 `torch.cuda.max_memory_allocated()` 在 8 个 GPUs 上得到的最大 GPU 显存占用值，需要注意的是，这些显存占用值通常小于 `nvidia-smi` 显示出来的显存占用值；
- 在模型库中所展示的推理时间是包括网络前向传播和后处理所需的总时间，不包括数据加载所需的时间，模型库中所展示的结果均由 benchmark.py 脚本文件在 2000 张图像上所计算的平均时间。

6.2 基准结果

6.2.1 SECOND

请参考 SECOND 获取更多的细节，我们在 KITTI 和 Waymo 数据集上都给出了相应的基准结果。

6.2.2 PointPillars

请参考 PointPillars 获取更多细节，我们在 KITTI 、nuScenes 、Lyft 、Waymo 数据集上给出了相应的基准结果。

6.2.3 Part-A2

请参考 Part-A2 获取更多细节。

6.2.4 VoteNet

请参考 VoteNet 获取更多细节，我们在 ScanNet 和 SUNRGBD 数据集上给出了相应的基准结果。

6.2.5 Dynamic Voxelization

请参考 Dynamic Voxelization 获取更多细节。

6.2.6 MVXNet

请参考 MVXNet 获取更多细节。

6.2.7 RegNetX

请参考 RegNet 获取更多细节，我们将 pointpillars 的主干网络替换成 RegNetX，并在 nuScenes 和 Lyft 数据集上给出了相应的基准结果。

6.2.8 nuImages

我们在 nuImages 数据集 上也提供基准模型，请参考 nuImages 获取更多细节，我们在该数据集上提供 Mask R-CNN ，Cascade Mask R-CNN 和 HTC 的结果。

6.2.9 H3DNet

请参考 H3DNet 获取更多细节。

6.2.10 3DSSD

请参考 3DSSD 获取更多细节。

6.2.11 CenterPoint

请参考 CenterPoint 获取更多细节。

6.2.12 SSN

请参考 SSN 获取更多细节，我们将 pointpillars 中的检测头替换成 SSN 模型中所使用的‘shape-aware grouping heads’，并在 nuScenes 和 Lyft 数据集上给出了相应的基准结果。

6.2.13 ImVoteNet

请参考 ImVoteNet 获取更多细节，我们在 SUNRGBD 数据集上给出了相应的结果。

6.2.14 FCOS3D

请参考 FCOS3D 获取更多细节，我们在 nuScenes 数据集上给出了相应的结果。

6.2.15 PointNet++

请参考 PointNet++ 获取更多细节，我们在 ScanNet 和 S3DIS 数据集上给出了相应的结果。

6.2.16 Group-Free-3D

请参考 Group-Free-3D 获取更多细节，我们在 ScanNet 数据集上给出了相应的结果。

6.2.17 ImVoxelNet

请参考 ImVoxelNet 获取更多细节，我们在 KITTI 数据集上给出了相应的结果。

6.2.18 PAConv

请参考 PAConv 获取更多细节，我们在 S3DIS 数据集上给出了相应的结果。

6.2.19 DGCNN

请参考 DGCNN 获取更多细节，我们在 S3DIS 数据集上给出了相应的结果。

6.2.20 SMOKE

请参考 SMOKE 获取更多细节，我们在 KITTI 数据集上给出了相应的结果。

6.2.21 PGD

请参考 PGD 获取更多细节，我们在 KITTI 和 nuScenes 数据集上给出了相应的结果。

6.2.22 PointRCNN

请参考 PointRCNN 获取更多细节，我们在 KITTI 数据集上给出了相应的结果。

6.2.23 MonoFlex

请参考 MonoFlex 获取更多细节，我们在 KITTI 数据集上给出了相应的结果。

6.2.24 SA-SSD

请参考 SA-SSD 获取更多的细节，我们在 KITTI 数据集上给出了相应的基准结果。

6.2.25 FCAF3D

请参考 FCAF3D 获取更多的细节，我们在 ScanNet, S3DIS 和 SUN RGB-D 数据集上给出了相应的基准结果。

6.2.26 Mixed Precision (FP16) Training

细节请参考 Mixed Precision (FP16) Training 在 PointPillars 训练的样例。

CHAPTER 7

数据预处理

7.1 在数据预处理前

我们推荐用户将数据集的路径软链接到 `$MMDETECTION3D/data`。如果你的文件夹结构和以下所展示的结构相异，你可能需要改变配置文件中相应的数据路径。

```
mmdetection3d
├── mmdet3d
├── tools
├── configs
├── data
|   ├── nuscenes
|   |   ├── maps
|   |   ├── samples
|   |   ├── sweeps
|   |   ├── v1.0-test
|   |   ├── v1.0-trainval
|   ├── kitti
|   |   ├── ImageSets
|   |   ├── testing
|   |   |   ├── calib
|   |   |   ├── image_2
|   |   |   ├── velodyne
|   |   ├── training
|   |   |   ├── calib
```

(下页继续)

(续上页)

```
|   |   |   ├── image_2
|   |   |   ├── label_2
|   |   |   ├── velodyne
|   ├── waymo
|   |   ├── waymo_format
|   |   |   ├── training
|   |   |   ├── validation
|   |   |   ├── testing
|   |   |   ├── gt.bin
|   |   ├── kitti_format
|   |   |   ├── ImageSets
|   ├── lyft
|   |   ├── v1.01-train
|   |   |   ├── v1.01-train (训练数据)
|   |   |   ├── lidar (训练激光雷达)
|   |   |   ├── images (训练图片)
|   |   |   ├── maps (训练地图)
|   |   ├── v1.01-test
|   |   |   ├── v1.01-test (测试数据)
|   |   |   ├── lidar (测试激光雷达)
|   |   |   ├── images (测试图片)
|   |   |   ├── maps (测试地图)
|   |   ├── train.txt
|   |   ├── val.txt
|   |   ├── test.txt
|   |   ├── sample_submission.csv
|   ├── s3dis
|   |   ├── meta_data
|   |   ├── Stanford3dDataset_v1.2_Aligned_Version
|   |   ├── collect_indoor3d_data.py
|   |   ├── indoor3d_util.py
|   |   ├── README.md
|   ├── scannet
|   |   ├── meta_data
|   |   ├── scans
|   |   ├── scans_test
|   |   ├── batch_load_scannet_data.py
|   |   ├── load_scannet_data.py
|   |   ├── scannet_utils.py
|   |   ├── README.md
|   ├── sunrgbd
|   |   ├── OFFICIAL_SUNRGBD
|   |   ├── matlab
```

(下页继续)

(续上页)

```
|   |   ├── sunrgbd_data.py
|   |   ├── sunrgbd_utils.py
|   |   ├── README.md
```

7.2 数据下载和预处理

7.2.1 KITTI

在这里下载 KITTI 的 3D 检测数据。通过运行以下指令对 KITTI 数据进行预处理：

```
mkdir ./data/kitti/ && mkdir ./data/kitti/ImageSets

# 下载数据划分文件
wget -c  https://raw.githubusercontent.com/traveller59/second.pytorch/master/second/
↪data/ImageSets/test.txt --no-check-certificate --content-disposition -O ./data/
↪kitti/ImageSets/test.txt
wget -c  https://raw.githubusercontent.com/traveller59/second.pytorch/master/second/
↪data/ImageSets/train.txt --no-check-certificate --content-disposition -O ./data/
↪kitti/ImageSets/train.txt
wget -c  https://raw.githubusercontent.com/traveller59/second.pytorch/master/second/
↪data/ImageSets/val.txt --no-check-certificate --content-disposition -O ./data/kitti/
↪ImageSets/val.txt
wget -c  https://raw.githubusercontent.com/traveller59/second.pytorch/master/second/
↪data/ImageSets/trainval.txt --no-check-certificate --content-disposition -O ./data/
↪kitti/ImageSets/trainval.txt

python tools/create_data.py kitti --root-path ./data/kitti --out-dir ./data/kitti --
↪extra-tag kitti
```

7.2.2 Waymo

在这里下载 Waymo 公开数据集 1.2 版本，在这里下载其数据划分文件。然后，将 tfrecord 文件置于 `data/waymo/waymo_format/` 目录下的相应位置，并将数据划分的 txt 文件置于 `data/waymo/kitti_format/ImageSets` 目录下。在这里下载验证集的真实标签 (bin 文件) 并将其置于 `data/waymo/waymo_format/`。提示，你可以使用 `gsutil` 来用命令下载大规模的数据集。你可以参考这个工具来获取更多实现细节。完成以上各步后，可以通过运行以下指令对 Waymo 数据进行预处理：

```
python tools/create_data.py waymo --root-path ./data/waymo/ --out-dir ./data/waymo/ --
↪workers 128 --extra-tag waymo
```

注意，如果你的硬盘空间大小不足以存储转换后的数据，你可以将 `out-dir` 参数设定为别的路径。你只需要记得在那个路径下创建文件夹并下载数据，然后在数据预处理完成后将其链接回 `data/waymo/kitti_format` 即可。

7.2.3 NuScenes

在这里下载 nuScenes 数据集 1.0 版本的完整数据文件。通过运行以下指令对 nuScenes 数据进行预处理：

```
python tools/create_data.py nuscenes --root-path ./data/nuscenes --out-dir ./data/
↪nuscenes --extra-tag nuscenes
```

7.2.4 Lyft

在这里下载 Lyft 3D 检测数据。通过运行以下指令对 Lyft 数据进行预处理：

```
python tools/create_data.py lyft --root-path ./data/lyft --out-dir ./data/lyft --
↪extra-tag lyft --version v1.01
python tools/data_converter/lyft_data_fixer.py --version v1.01 --root-folder ./data/
↪lyft
```

注意，为了文件结构的清晰性，我们遵从了 Lyft 数据原先的文件夹名称。请按照上面展示出的文件结构对原始文件夹进行重命名。同样值得注意的是，第二行命令的目的是为了修复一个损坏的激光雷达数据文件。请参考这一讨论来获取更多细节。

7.2.5 S3DIS、ScanNet 和 SUN RGB-D

请参考 S3DIS README 文件以对其进行数据预处理。

请参考 ScanNet README 文件以对其进行数据预处理。

请参考 SUN RGB-D README 文件以对其进行数据预处理。

7.2.6 自定义数据集

关于如何使用自定义数据集，请参考教程 2: 自定义数据集。

CHAPTER 8

1: 使用已有模型在标准数据集上进行推理和训练

8.1 使用已有模型进行推理

这里我们提供了评测 SUNRGBD、ScanNet、KITTI 等多个数据集的测试脚本。

请参考开始下的验证/样例来获取更容易集成到其它项目和基本样例的高级接口。

8.1.1 在标准数据集上测试已有模型

- 单显卡
- CPU
- 单节点多显卡
- 多节点

你可以通过以下命令来测试数据集：

```
# 单块显卡测试
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--out ${RESULT_FILE}] [--eval
↪${EVAL_METRICS}] [--show] [--show-dir ${SHOW_DIR}]

# CPU: 禁用显卡并运行单块 CPU 测试脚本（实验性）
export CUDA_VISIBLE_DEVICES=-1
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--out ${RESULT_FILE}] [--eval
↪${EVAL_METRICS}] [--show] [--show-dir ${SHOW_DIR}]
```

(下页继续)

(续上页)

```
# 多块显卡测试
./tools/dist_test.sh ${CONFIG_FILE} ${CHECKPOINT_FILE} ${GPU_NUM} [--out ${RESULT_
↪FILE}] [--eval ${EVAL_METRICS}]
```

注意:

目前我们只支持 SMOKE 的 CPU 推理测试。

可选参数:

- `RESULT_FILE`: 输出结果（pickle 格式）的文件名，如果未指定，结果不会被保存。
- `EVAL_METRICS`: 在结果上评测的项，不同的数据集有不同的合法值。具体来说，我们默认对不同的数据集都使用各自的官方度量方法进行评测，所以对 nuScenes、Lyft、ScanNet 和 SUNRGBD 这些数据集来说在检测任务上可以简单设置为 `mAP`；对 KITTI 数据集来说，如果我们只想评测 2D 检测效果，可以将度量方法设置为 `img_bbox`；对于 Waymo 数据集，我们提供了 KITTI 风格（不稳定）和 Waymo 官方风格这两种评测方法，分别对应 `kitti` 和 `waymo`，我们推荐使用默认的官方度量方法，它的性能稳定而且可以与其它算法公平比较；同样地，对 S3DIS、ScanNet 这些数据集来说，在分割任务上的度量方法可以设置为 `mIoU`。
- `--show`: 如果被指定，检测结果会在静默模式下被保存，用于调试和可视化，但只在单块 GPU 测试的情况下生效，和 `--show-dir` 搭配使用。
- `--show-dir`: 如果被指定，检测结果会被保存在指定文件夹下的 `***_points.obj` 和 `***_pred.obj` 文件中，用于调试和可视化，但只在单块 GPU 测试的情况下生效，对于这个选项，图形化界面在你的环境中不是必需的。

示例:

假定你已经把模型权重文件下载到 `checkpoints/` 文件夹下，

1. 在 ScanNet 数据集上测试 VoteNet，保存模型，可视化预测结果

```
python tools/test.py configs/votenet/votenet_8x8_scannet-3d-18class.py \
    checkpoints/votenet_8x8_scannet-3d-18class_20200620_230238-2cea9c3a.pth \
    --show --show-dir ./data/scannet/show_results
```

2. 在 ScanNet 数据集上测试 VoteNet，保存模型，可视化预测结果，可视化真实标签，计算 mAP

```
python tools/test.py configs/votenet/votenet_8x8_scannet-3d-18class.py \
    checkpoints/votenet_8x8_scannet-3d-18class_20200620_230238-2cea9c3a.pth \
    --eval mAP
    --eval-options 'show=True' 'out_dir=./data/scannet/show_results'
```

3. 在 ScanNet 数据集上测试 VoteNet（不保存测试结果），计算 mAP

```
python tools/test.py configs/votenet/votenet_8x8_scannet-3d-18class.py \
    checkpoints/votenet_8x8_scannet-3d-18class_20200620_230238-2cea9c3a.pth \
    --eval mAP
```

4. 使用 8 块显卡在 KITTI 数据集上测试 SECOND，计算 mAP

```
./tools/slurm_test.sh ${PARTITION} ${JOB_NAME} configs/second/hv_second_secfpn_
↪6x8_80e_kitti-3d-3class.py \
    checkpoints/hv_second_secfpn_6x8_80e_kitti-3d-3class_20200620_230238-9208083a.
↪pth \
    --out results.pkl --eval mAP
```

5. 使用 8 块显卡在 nuScenes 数据集上测试 PointPillars，生成提交给官方评测服务器的 json 文件

```
./tools/slurm_test.sh ${PARTITION} ${JOB_NAME} configs/pointpillars/hv_
↪pointpillars_fpn_sbn-all_4x8_2x_nus-3d.py \
    checkpoints/hv_pointpillars_fpn_sbn-all_4x8_2x_nus-3d_20200620_230405-
↪2fa62f3d.pth \
    --format-only --eval-options 'jsonfile_prefix=./pointpillars_nuscenes_results'
```

生成的结果会保存在 `./pointpillars_nuscenes_results` 目录。

6. 使用 8 块显卡在 KITTI 数据集上测试 SECOND，生成提交给官方评测服务器的 txt 文件

```
./tools/slurm_test.sh ${PARTITION} ${JOB_NAME} configs/second/hv_second_secfpn_
↪6x8_80e_kitti-3d-3class.py \
    checkpoints/hv_second_secfpn_6x8_80e_kitti-3d-3class_20200620_230238-9208083a.
↪pth \
    --format-only --eval-options 'pklfile_prefix=./second_kitti_results'
↪'submission_prefix=./second_kitti_results'
```

生成的结果会保存在 `./second_kitti_results` 目录。

7. 使用 8 块显卡在 Lyft 数据集上测试 PointPillars，生成提交给排行榜的 pkl 文件

```
./tools/slurm_test.sh ${PARTITION} ${JOB_NAME} configs/pointpillars/hv_
↪pointpillars_fpn_sbn-2x8_2x_lyft-3d.py \
    checkpoints/hv_pointpillars_fpn_sbn-2x8_2x_lyft-3d_latest.pth --out results/
↪pp_lyft/results_challenge.pkl \
    --format-only --eval-options 'jsonfile_prefix=results/pp_lyft/results_
↪challenge' \
    'csv_savepath=results/pp_lyft/results_challenge.csv'
```

注意：为了生成 Lyft 数据集的提交结果，`--eval-options` 必须指定 `csv_savepath`。生成 csv 文件后，你可以使用网站上给出的 kaggle 命令提交结果。

注意在 Lyft 数据集的配置文件，test 中的 ann_file 值为 data_root + 'lyft_infos_test.pkl'，是没有标注的 Lyft 官方测试集。要在验证数据集上测试，请把它改为 data_root + 'lyft_infos_val.pkl'。

8. 使用 8 块显卡在 waymo 数据集上测试 PointPillars，使用 waymo 度量方法计算 mAP

```
./tools/slurm_test.sh ${PARTITION} ${JOB_NAME} configs/pointpillars/hv_
↪pointpillars_secfpn_sbn-2x16_2x_waymo-3d-car.py \
    checkpoints/hv_pointpillars_secfpn_sbn-2x16_2x_waymo-3d-car_latest.pth --out␣
↪results/waymo-car/results_eval.pkl \
    --eval waymo --eval-options 'pklfile_prefix=results/waymo-car/kitti_results' \
    'submission_prefix=results/waymo-car/kitti_results'
```

注意：对于 waymo 数据集上的评估，请根据说明构建二进制文件 compute_detection_metrics_main 来做度量计算，并把它放在 mmdet3d/core/evaluation/waymo_utils/。（在使用 bazel 构建 compute_detection_metrics_main 时，有时会出现 'round' is not a member of 'std' 的错误，我们只需要把那个文件中 round 前的 std:: 去掉。）二进制文件生成时需要在 --eval-options 中给定 pklfile_prefix。对于度量方法，waymo 是推荐的官方评估策略，目前 kitti 评估是依照 KITTI 而来的，每个难度的结果和 KITTI 的定义并不完全一致。目前大多数物体都被标记为 0 难度，会在未来修复。它的不稳定原因包括评估的计算大、转换后的数据缺乏遮挡和截断、难度的定义不同以及平均精度的计算方法不同。

9. 使用 8 块显卡在 waymo 数据集上测试 PointPillars，生成 bin 文件并提交到排行榜

```
./tools/slurm_test.sh ${PARTITION} ${JOB_NAME} configs/pointpillars/hv_
↪pointpillars_secfpn_sbn-2x16_2x_waymo-3d-car.py \
    checkpoints/hv_pointpillars_secfpn_sbn-2x16_2x_waymo-3d-car_latest.pth --out␣
↪results/waymo-car/results_eval.pkl \
    --format-only --eval-options 'pklfile_prefix=results/waymo-car/kitti_results'␣
↪\
    'submission_prefix=results/waymo-car/kitti_results'
```

注意：生成 bin 文件后，你可以简单地构建二进制文件 create_submission，并根据说明创建提交的文件。要在验证服务器上评测验证数据集，你也可以用同样的方式生成提交的文件。

8.2 在标准数据集上训练预定义模型

MMDetection3D 分别用 MMDistributedDataParallel and MMDataParallel 实现了分布式训练和非分布式训练。

所有的输出（日志文件和模型权重文件）都会被保存到工作目录下，通过配置文件里的 work_dir 指定。

默认我们每过一个周期都在验证数据集上评测模型，你可以通过在训练配置里添加间隔参数来改变评测的时间间隔：

```
evaluation = dict(interval=12)  # 每 12 个周期评估一次模型
```

重要：配置文件中的默认学习率对应 8 块显卡，配置文件名里有具体的批量大小，比如’2x8’表示一共 8 块显卡，每块显卡 2 个样本。根据 Linear Scaling Rule，当你使用不同数量的显卡或每块显卡有不同数量的图像时，需要依批量大小按比例调整学习率。如果用 4 块显卡、每块显卡 2 幅图像时学习率为 0.01，那么用 16 块显卡、每块显卡 4 幅图像时学习率应设为 0.08。然而，由于大多数模型使用 ADAM 而不是 SGD 进行优化，上述规则可能并不适用，用户需要自己调整学习率。

8.2.1 使用单块显卡进行训练

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

如果你想在命令中指定工作目录，添加参数 `--work-dir ${YOUR_WORK_DIR}`。

8.2.2 使用 CPU 进行训练 (实验性)

在 CPU 上训练的过程与单 GPU 训练一致。我们只需要在训练过程之前禁用显卡。

```
export CUDA_VISIBLE_DEVICES=-1
```

之后运行单显卡训练脚本即可。

注意：

目前，大多数点云相关算法都依赖于 3D CUDA 算子，无法在 CPU 上进行训练。一些单目 3D 物体检测算法，例如 FCOS3D、SMOKE 可以在 CPU 上进行训练。我们不推荐用户使用 CPU 进行训练，这太过缓慢。我们支持这个功能是为了方便用户在没有显卡的机器上调试某些特定的方法。

8.2.3 使用多块显卡进行训练

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

可选参数：

- `--no-validate`（**不推荐**）：默认情况下，代码在训练阶段每 k（默认值是 1，可以像这里一样修改）个周期做一次评测，如果要取消评测，使用 `--no-validate`。
- `--work-dir ${WORK_DIR}`：覆盖配置文件中的指定工作目录。
- `--resume-from ${CHECKPOINT_FILE}`：从之前的模型权重文件中恢复。
- `--options 'Key=value'`：覆盖使用的配置中的一些设定。

`resume-from` 和 `load-from` 的不同点：

- `resume-from` 加载模型权重和优化器状态，同时周期数也从特定的模型权重文件中继承，通常用于恢复偶然中断的训练过程。
- `load-from` 仅加载模型权重，训练周期从 0 开始，通常用于微调。

8.2.4 使用多个机器进行训练

如果要在 slurm 管理的集群上运行 MMDectection3D，你可以使用 `slurm_train.sh` 脚本（该脚本也支持单机训练）

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

下面是一个使用 16 块显卡在 dev 分区上训练 Mask R-CNN 的示例：

```
GPUS=16 ./tools/slurm_train.sh dev pp_kitti_3class hv_pointpillars_secfpn_6x8_160e_
↪kitti-3d-3class.py /nfs/xxxx/pp_kitti_3class
```

你可以查看 slurm_train.sh 来获取所有的参数和环境变量。

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 ./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 ./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。

8.2.5 在单个机器上启动多个任务

如果你在单个机器上启动多个任务，比如，在具有 8 块显卡的机器上进行 2 个 4 块显卡训练的任务，你需要为每个任务指定不同的端口（默认为 29500）以避免通信冲突。

如果你使用 `dist_train.sh` 启动训练任务，可以在命令中设置端口：

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

如果你使用 Slurm 启动训练任务，有两种方式指定端口：

1. 通过 `--options` 设置端口，这是更推荐的，因为它不改变原来的配置

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

2. 修改配置文件（通常在配置文件的倒数第 6 行）来设置不同的通信端口

   在 config1.py 中，

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

   在 config2.py 中，

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

   然后，你可以使用 config1.py and config2.py 启动两个任务

```
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 9

2: 在自定义数据集上进行训练

本文将主要介绍如何使用自定义数据集来进行模型的训练和测试，以 Waymo 数据集作为示例来说明整个流程。

基本步骤如下所示：

1. 准备自定义数据集；
2. 准备配置文件；
3. 在自定义数据集上进行模型的训练、测试和推理。

9.1 准备自定义数据集

在 MMDetection3D 中有三种方式来自定义一个新的数据集：

1. 将新数据集的数据格式重新组织成已支持的数据集格式；
2. 将新数据集的数据格式重新组织成已支持的一种中间格式；
3. 从头开始创建一个新的数据集。

由于前两种方式比第三种方式更加容易，我们更加建议采用前两种方式来自定义数据集。

在本文中，我们采用第一种方式来将 Waymo 数据集重新组织成 KITTI 数据集的数据格式。

注意：考虑到 Waymo 数据集的格式与现有的其他数据集的格式的差别较大，因此本文以该数据集为例来讲解如何自定义数据集，从而方便理解数据集自定义的过程。若需要创建的新数据集与现有的数据集的组织格

式较为相似，如 Lyft 数据集和 nuScenes 数据集，采用对数据集的中间格式进行转换的方式（第二种方式）相比于采用对数据格式进行转换的方式（第一种方式）会更加简单易行。

9.1.1 KITTI 数据集格式

应用于 3D 目标检测的 KITTI 原始数据集的组织方式通常如下所示，其中 `ImageSets` 包含数据集划分文件，用以划分训练集/验证集/测试集，`calib` 包含对于每个数据样本的标定信息，`image_2` 和 `velodyne` 分别包含图像数据和点云数据，`label_2` 包含与 3D 目标检测相关的标注文件。

```
mmdetection3d
├── mmdet3d
├── tools
├── configs
├── data
│   ├── kitti
│   │   ├── ImageSets
│   │   ├── testing
│   │   │   ├── calib
│   │   │   ├── image_2
│   │   │   ├── velodyne
│   │   ├── training
│   │   │   ├── calib
│   │   │   ├── image_2
│   │   │   ├── label_2
│   │   │   ├── velodyne
```

KITTI 官方提供的目标检测开发工具包详细描述了 KITTI 数据集的标注格式，例如，KITTI 标注格式包含了以下的标注信息：

```
#  值     名称       描述
-------------------------------------------------------------------------------
   1     类型       描述检测目标的类型：'Car', 'Van', 'Truck',
                     'Pedestrian', 'Person_sitting', 'Cyclist', 'Tram',
                     'Misc' 或 'DontCare'
   1     截断程度   从 0（非截断）到 1（截断）的浮点数，其中截断指的是离开检测图像边界的检测目标
   1     遮挡程度   用来表示遮挡状态的四种整数（0，1，2，3）：
                     0 = 可见，1 = 部分遮挡
                     2 = 大面积遮挡，3 = 未知
   1     观测角     观测目标的角度，取值范围为 [-pi..pi]
   4     标注框     检测目标在图像中的二维标注框（以 0 为初始下标）：包括每个检测目标的左上角和右下角的
坐标
   3     维度       检测目标的三维维度：高度、宽度、长度（以米为单位）
   3     位置       相机坐标系下的三维位置 x，y，z（以米为单位）
   1     y 旋转     相机坐标系下检测目标绕着 Y 轴的旋转角，取值范围为 [-pi..pi]
```

(下页继续)

(续上页)

```
    1      得分        仅在计算结果时使用，检测中表示置信度的浮点数，用于生成 p/r 曲线，在 p/r 图中，越高
的曲线表示结果越好。
```

接下来本文将对 Waymo 数据集原始格式进行转换。首先需要将下载的 Waymo 数据集的数据文件和标注文件转换到 KITTI 数据集的格式，接着定义一个从 KittiDataset 类继承而来的 WaymoDataset 类，来帮助数据的加载、模型的训练和评估。

具体来说，首先使用数据转换器将 Waymo 数据集转换成 KITTI 数据集的格式，并定义 Waymo 类对转换的数据进行处理。因为我们将 Waymo 原始数据集进行预处理并重新组织成 KITTI 数据集的格式，因此可以比较容易通过继承 KittiDataset 类来实现 WaymoDataset 类。需要注意的是，由于 Waymo 数据集有相应的官方评估方法，我们需要在定义新数据类的过程中引入官方评估方法，此时用户可以顺利的转换 Waymo 数据的格式，并使用 `WaymoDataset` 数据类进行模型的训练和评估。

更多关于 Waymo 数据集预处理的中间结果的细节，请参照对应的说明文档。

9.2 准备配置文件

第二步是准备配置文件来帮助数据集的读取和使用，另外，为了在 3D 检测中获得不错的性能，调整超参数通常是必要的。

假设我们想要使用 PointPillars 模型在 Waymo 数据集上实现三类的 3D 目标检测：vehicle、cyclist、pedestrian，参照 KITTI 数据集配置文件、模型配置文件和整体配置文件，我们需要准备数据集配置文件、模型配置文件，并将这两种文件进行结合得到整体配置文件。

9.3 训练一个新的模型

为了使用一个新的配置文件来训练模型，可以通过下面的命令来实现：

```
python tools/train.py configs/pointpillars/hv_pointpillars_secfpn_sbn_2x16_2x_waymoD5-
↪3d-3class.py
```

更多的使用细节，请参考案例 1。

9.4 测试和推理

为了测试已经训练好的模型的性能，可以通过下面的命令来实现：

```
python tools/test.py configs/pointpillars/hv_pointpillars_secfpn_sbn_2x16_2x_waymoD5-
↪3d-3class.py work_dirs/hv_pointpillars_secfpn_sbn_2x16_2x_waymoD5-3d-3class/latest.
↪pth --eval waymo
```

注意：为了使用 Waymo 数据集的评估方法，需要参考说明文档并按照官方指导来准备与评估相关联的文件。更多有关测试和推理的使用细节，请参考案例 1 。

CHAPTER 10

基于 LiDAR 的 3D 检测

基于 LiDAR 的 3D 检测算法是 MMDetection3D 支持的最基础的任务之一。对于给定的算法模型，输入为任意数量的、附有 LiDAR 采集的特征的点，输出为每个感兴趣目标的 3D 矩形框 (Bounding Box) 和类别标签。接下来，我们将以在 KITTI 数据集上训练 PointPillars 为例，介绍如何准备数据，如何在标准 3D 检测基准数据集上训练和测试模型，以及如何可视化并验证结果。

10.1 数据预处理

最开始，我们需要下载原始数据，并按文档中介绍的那样，把数据重新整理成标准格式。值得注意的是，对于 KIITI 数据集，我们需要额外的 txt 文件用于数据整理。

由于不同数据集上的原始数据有不同的组织方式，我们通常需要用.pkl 或者.json 文件收集有用的数据信息。在准备好原始数据后，我们需要运行脚本 `create_data.py`，为不同的数据集生成数据。如，对于 KITTI 数据集，我们需要执行:

```
python tools/create_data.py kitti --root-path ./data/kitti --out-dir ./data/kitti --
↪extra-tag kitti
```

随后，相对目录结构将变成如下形式：

```
mmdetection3d
├── mmdet3d
├── tools
├── configs
```

(下页继续)

(续上页)

```
├── data
│   ├── kitti
│   │   ├── ImageSets
│   │   ├── testing
│   │   │   ├── calib
│   │   │   ├── image_2
│   │   │   ├── velodyne
│   │   ├── training
│   │   │   ├── calib
│   │   │   ├── image_2
│   │   │   ├── label_2
│   │   │   ├── velodyne
│   │   ├── kitti_gt_database
│   │   ├── kitti_infos_train.pkl
│   │   ├── kitti_infos_trainval.pkl
│   │   ├── kitti_infos_val.pkl
│   │   ├── kitti_infos_test.pkl
│   │   ├── kitti_dbinfos_train.pkl
```

10.2 训练

接着，我们将使用提供的配置文件训练 PointPillars。当你使用不同的 GPU 设置进行训练时，你基本上可以按照这个教程的示例脚本进行训练。假设我们在一台具有 8 块 GPU 的机器上进行分布式训练：

```
./tools/dist_train.sh configs/pointpillars/hv_pointpillars_secfpn_6x8_160e_kitti-3d-
↪3class.py 8
```

注意到，配置文件名字中的 6x8 是指训练时是用了 8 块 GPU，每块 GPU 上有 6 个样本。如果你有不同的自定义的设置，那么有时你可能需要调整学习率。可以参考这篇文献。

10.3 定量评估

在训练期间，模型将会根据配置文件中的 `evaluation = dict(interval=xxx)` 设置，被周期性地评估。我们支持不同数据集的官方评估方案。对于 KITTI, 模型的评价指标为平均精度 (mAP, mean average precision)。3 种类型的 mAP 的交并比 (IoU, Intersection over Union) 阈值可以取 0.5/0.7。评估结果将会被打印到终端中，如下所示：

```
Car AP@0.70, 0.70, 0.70:
bbox AP:98.1839, 89.7606, 88.7837
bev AP:89.6905, 87.4570, 85.4865
```

(下页继续)

(续上页)

```
3d AP:87.4561, 76.7569, 74.1302
aos AP:97.70, 88.73, 87.34
Car AP@0.70, 0.50, 0.50:
bbox AP:98.1839, 89.7606, 88.7837
bev AP:98.4400, 90.1218, 89.6270
3d AP:98.3329, 90.0209, 89.4035
aos AP:97.70, 88.73, 87.34
```

评估某个特定的模型权重文件。你可以简单地执行下列的脚本:

```
./tools/dist_test.sh configs/pointpillars/hv_pointpillars_secfpn_6x8_160e_kitti-3d-
↪3class.py \
    work_dirs/pointpillars/latest.pth --eval mAP
```

10.4 测试与提交

如果你只想在线上基准上进行推理或者测试模型的表现，你只需要把上面评估脚本中的 `--eval mAP` 替换为 `--format-only`。如果需要的话，还可以指定 `pklfile_prefix` 和 `submission_prefix`，如，添加命令行选项 `--eval-options submission_prefix=work_dirs/pointpillars/test_submission`。请确保配置文件中的测试信息与测试集对应，而不是验证集。在生成结果后，你可以压缩文件夹，并上传到 KITTI 的评估服务器上。

10.5 定性验证

MMDetection3D 还提供了通用的可视化工具，以便于我们可以对训练好的模型的预测结果有一个直观的感受。你可以在命令行中添加 `--eval-options 'show=True' 'out_dir=${SHOW_DIR}'` 选项，在评估过程中在线地可视化检测结果；你也可以使用 `tools/misc/visualize_results.py`，离线地进行可视化。另外，我们还提供了脚本 `tools/misc/browse_dataset.py`，可视化数据集而不做推理。更多的细节请参考可视化文档

CHAPTER 11

基于视觉的 3D 检测

基于视觉的 3D 检测是指基于纯视觉输入的 3D 检测方法，例如基于单目、双目和多视图图像的 3D 检测。目前，我们只支持单目和多视图的 3D 检测方法。其他方法也应该与我们的框架兼容，并在将来得到支持。

它期望给定的模型以任意数量的图像作为输入，并为每一个感兴趣的目标预测 3D 框及类别标签。以 nuScenes 数据集 FCOS3D 为例，我们将展示如何准备数据，在标准的 3D 检测基准上训练并测试模型，以及可视化并验证结果。

11.1 数据准备

首先，我们需要下载原始数据并按照数据准备文档中提供的标准方式重新组织数据。

由于不同数据集的原始数据有不同的组织方式，我们通常需要用 pkl 或 json 文件收集有用的数据信息。因此，在准备好所有的原始数据之后，我们需要运行 `create_data.py` 中提供的脚本来为不同的数据集生成数据信息。例如，对于 nuScenes，我们需要运行如下命令：

```
python tools/create_data.py nuscenes --root-path ./data/nuscenes --out-dir ./data/
↪nuscenes --extra-tag nuscenes
```

随后，相关的目录结构将如下所示：

```
mmdetection3d
├── mmdet3d
├── tools
├── configs
```

(下页继续)

(续上页)

```
├── data
|   ├── nuscenes
|   |   ├── maps
|   |   ├── samples
|   |   ├── sweeps
|   |   ├── v1.0-test
|   |   ├── v1.0-trainval
|   |   ├── nuscenes_database
|   |   ├── nuscenes_infos_train.pkl
|   |   ├── nuscenes_infos_trainval.pkl
|   |   ├── nuscenes_infos_val.pkl
|   |   ├── nuscenes_infos_test.pkl
|   |   ├── nuscenes_dbinfos_train.pkl
|   |   ├── nuscenes_infos_train_mono3d.coco.json
|   |   ├── nuscenes_infos_trainval_mono3d.coco.json
|   |   ├── nuscenes_infos_val_mono3d.coco.json
|   |   ├── nuscenes_infos_test_mono3d.coco.json
```

注意，此处的 pkl 文件主要用于使用 LiDAR 数据的方法，json 文件用于 2D 检测/纯视觉的 3D 检测。在 v0.13.0 支持单目 3D 检测之前，json 文件只包含 2D 检测的信息，因此如果你需要最新的信息，请切换到 v0.13.0 之后的分支。

11.2 训练

接着，我们将使用提供的配置文件训练 FCOS3D。基本的脚本与其他模型一样。当你使用不同的 GPU 设置进行训练时，你基本上可以按照这个教程的示例。假设我们在一台具有 8 块 GPU 的机器上使用分布式训练：

```
./tools/dist_train.sh configs/fcos3d/fcos3d_r101_caffe_fpn_gn-head_dcn_2x8_1x_nus-
↪mono3d.py 8
```

注意，配置文件名中的 2x8 是指训练时用了 8 块 GPU，每块 GPU 上有 2 个数据样本。如果你的自定义设置不同于此，那么有时候你需要相应的调整学习率。基本规则可以参考此处。

我们也可以通过运行以下命令微调 FCOS3D，从而达到更好的性能：

```
./tools/dist_train.sh fcos3d_r101_caffe_fpn_gn-head_dcn_2x8_1x_nus-mono3d_finetune.py␣
↪8
```

通过先前的脚本训练好一个基准模型后，请记得相应的修改此处的路径。

11.3 定量评估

在训练期间，模型权重文件将会根据配置文件中的 `evaluation = dict(interval=xxx)` 设置被周期性地评估。

我们支持不同数据集的官方评估方案。由于输出格式与基于其他模态的 3D 检测相同，因此评估方法也是一样的。

对于 nuScenes，将使用基于距离的平均精度（mAP）以及 nuScenes 检测分数（NDS）分别对 10 个类别进行评估。评估结果将会被打印到终端中，如下所示：

```
mAP: 0.3197
mATE: 0.7595
mASE: 0.2700
mAOE: 0.4918
mAVE: 1.3307
mAAE: 0.1724
NDS: 0.3905
Eval time: 170.8s

Per-class results:
Object Class    AP      ATE     ASE     AOE     AVE     AAE
car     0.503   0.577   0.152   0.111   2.096   0.136
truck   0.223   0.857   0.224   0.220   1.389   0.179
bus     0.294   0.855   0.204   0.190   2.689   0.283
trailer 0.081   1.094   0.243   0.553   0.742   0.167
construction_vehicle    0.058   1.017   0.450   1.019   0.137   0.341
pedestrian      0.392   0.687   0.284   0.694   0.876   0.158
motorcycle      0.317   0.737   0.265   0.580   2.033   0.104
bicycle 0.308   0.704   0.299   0.892   0.683   0.010
traffic_cone    0.555   0.486   0.309   nan     nan     nan
barrier 0.466   0.581   0.269   0.169   nan     nan
```

此外，在训练完成后你也可以评估特定的模型权重文件。你可以简单地执行以下脚本：

```
./tools/dist_test.sh configs/fcos3d/fcos3d_r101_caffe_fpn_gn-head_dcn_2x8_1x_nus-
↪mono3d.py \
    work_dirs/fcos3d/latest.pth --eval mAP
```

11.4 测试与提交

如果你只想在在线基准上进行推理或测试模型性能，你需要将之前评估脚本中的 `--eval mAP` 替换成 `--format-only`，并在需要的情况下指定 `jsonfile_prefix`，例如，添加选项 `--eval-options jsonfile_prefix=work_dirs/fcos3d/test_submission`。请确保配置文件中的测试信息由验证集相应地改为测试集。

在生成结果后，你可以压缩文件夹并上传至 nuScenes 3D 检测挑战的 evalAI 评估服务器上。

11.5 定性评估

MMDetection3D 还提供了通用的可视化工具，以便于我们可以对训练好的模型预测的检测结果有一个直观的感受。你也可以在评估阶段通过设置 `--eval-options 'show=True' 'out_dir=${SHOW_DIR}'` 来在线可视化检测结果，或者使用 `tools/misc/visualize_results.py` 来离线地进行可视化。

此外，我们还提供了脚本 `tools/misc/browse_dataset.py` 用于可视化数据集而不做推理。更多的细节请参考可视化文档。

注意，目前我们仅支持纯视觉方法在图像上的可视化。将来我们将集成在前景图以及鸟瞰图（BEV）中的可视化。

CHAPTER 12

基于激光雷达的 3D 语义分割

基于激光雷达的 3D 语义分割是 MMDetection3D 支持的最基础的任务之一。它期望给定的模型以激光雷达采集的任意数量的特征点为输入，并预测每个输入点的语义标签。接下来，我们以 ScanNet 数据集上的 PointNet++ (SSG) 为例，展示如何准备数据，在标准的 3D 语义分割基准上训练并测试模型，以及可视化并验证结果。

12.1 数据准备

首先，我们需要从 ScanNet 官方网站下载原始数据。

由于不同数据集的原始数据有不同的组织方式，我们通常需要用 pkl 或 json 文件收集有用的数据信息。

因此，在准备好所有的原始数据之后，我们可以遵循 ScanNet 文档中的说明生成数据信息。

随后，相关的目录结构将如下所示：

```
mmdetection3d
├── mmdet3d
├── tools
├── configs
├── data
|   ├── scannet
|   |   ├── scannet_utils.py
|   |   ├── batch_load_scannet_data.py
|   |   ├── load_scannet_data.py
|   |   ├── scannet_utils.py
```

(下页继续)

(续上页)

```
|   |   ├── README.md
|   |   ├── scans
|   |   ├── scans_test
|   |   ├── scannet_instance_data
|   |   ├── points
|   |   ├── instance_mask
|   |   ├── semantic_mask
|   |   ├── seg_info
|   |   |   ├── train_label_weight.npy
|   |   |   ├── train_resampled_scene_idxs.npy
|   |   |   ├── val_label_weight.npy
|   |   |   ├── val_resampled_scene_idxs.npy
|   |   ├── scannet_infos_train.pkl
|   |   ├── scannet_infos_val.pkl
|   |   ├── scannet_infos_test.pkl
```

12.2 训练

接着，我们将使用提供的配置文件训练 PointNet++ (SSG) 模型。当你使用不同的 GPU 设置进行训练时，你基本上可以按照这个教程的示例脚本。假设我们在一台具有 2 块 GPU 的机器上使用分布式训练：

```
./tools/dist_train.sh configs/pointnet2/pointnet2_ssg_16x2_cosine_200e_scannet_seg-3d-
↪20class.py 2
```

注意，配置文件名中的 `16x2` 是指训练时用了 2 块 GPU，每块 GPU 上有 16 个样本。如果你的自定义设置不同于此，那么有时候你需要相应的调整学习率。基本规则可以参考此处。

12.3 定量评估

在训练期间，模型权重将会根据配置文件中的 `evaluation = dict(interval=xxx)` 设置被周期性地评估。我们支持不同数据集的官方评估方案。对于 ScanNet，将使用 20 个类别的平均交并比 (mIoU) 对模型进行评估。评估结果将会被打印到终端中，如下所示：

```
+---------+--------+--------+---------+--------+--------+--------+--------+--------+--
↪------+-----------+---------+---------+--------+---------+--------------+-----------
↪-----+--------+--------+---------+----------------+--------+--------+---------+
| classes | wall   | floor  | cabinet | bed    | chair  | sofa   | table  | door   |␣
↪window | bookshelf | picture | counter | desk   | curtain | refrigerator |␣
↪showercurtrain | toilet | sink   | bathtub | otherfurniture | miou   | acc    | acc_
↪cls |
```

(下页继续)

(续上页)

```
+---------+--------+--------+---------+--------+--------+--------+--------+--------+--
↪------+----------+---------+---------+--------+---------+-------------+----------
↪-----+--------+--------+---------+----------------+--------+--------+---------+
| results | 0.7257 | 0.9373 | 0.4625  | 0.6613 | 0.7707 | 0.5562 | 0.5864 | 0.4010 |␣
↪0.4558 | 0.7011    | 0.2500  | 0.4645  | 0.4540 | 0.5399  | 0.2802       | 0.3488  ␣
↪       | 0.7359 | 0.4971 | 0.6922  | 0.3681          | 0.5444 | 0.8118 | 0.6695  |
+---------+--------+--------+---------+--------+--------+--------+--------+--------+--
↪------+----------+---------+---------+--------+---------+-------------+----------
↪-----+--------+--------+---------+----------------+--------+--------+---------+
```

此外，在训练完成后你也可以评估特定的模型权重文件。你可以简单地执行以下脚本：

```
./tools/dist_test.sh configs/pointnet2/pointnet2_ssg_16x2_cosine_200e_scannet_seg-3d-
↪20class.py \
    work_dirs/pointnet2_ssg/latest.pth --eval mIoU
```

12.4 测试与提交

如果你只想在在线基准上进行推理或测试模型性能，你需要将之前评估脚本中的 `--eval mIoU` 替换成 `--format-only`，并将 ScanNet 数据集配置文件中的 `ann_file=data_root + 'scannet_infos_val.pkl'` 变成 `ann_file=data_root + 'scannet_infos_test.pkl'`。记住将 `txt_prefix` 指定为保存测试结果的目录，例如，添加选项 `--eval-options txt_prefix=work_dirs/pointnet2_ssg/test_submission`。在生成结果后，你可以压缩文件夹并上传至 ScanNet 评估服务器上。

12.5 定性评估

MMDetection3D 还提供了通用的可视化工具，以便于我们可以对训练好的模型预测的分割结果有一个直观的感受。你也可以在评估阶段通过设置 `--eval-options 'show=True' 'out_dir=${SHOW_DIR}'` 来在线可视化分割结果，或者使用 `tools/misc/visualize_results.py` 来离线地进行可视化。此外，我们还提供了脚本 `tools/misc/browse_dataset.py` 用于可视化数据集而不做推理。更多的细节请参考可视化文档。

CHAPTER 13

3D 目标检测 KITTI 数据集

本页提供了有关在 MMDetection3D 中使用 KITTI 数据集的具体教程。

注意：此教程目前仅适用于基于雷达和多模态的 3D 目标检测的相关方法，与基于单目图像的 3D 目标检测相关的内容会在之后进行补充。

13.1 数据准备

您可以在这里下载 KITTI 3D 检测数据并解压缩所有 zip 文件。此外，您可以在这里下载道路平面信息，其在训练过程中作为一个可选项，用来提高模型的性能。道路平面信息由 AVOD 生成，你可以在这里查看更多细节。

像准备数据集的一般方法一样，建议将数据集根目录链接到 `$MMDETECTION3D/data`。

在我们处理之前，文件夹结构应按如下方式组织：

```
mmdetection3d
├── mmdet3d
├── tools
├── configs
├── data
│   ├── kitti
│   │   ├── ImageSets
│   │   ├── testing
│   │   │   ├── calib
```

(下页继续)

(续上页)

```
|   |   |   ├── image_2
|   |   |   ├── velodyne
|   |   ├── training
|   |   |   ├── calib
|   |   |   ├── image_2
|   |   |   ├── label_2
|   |   |   ├── velodyne
|   |   |   ├── planes (optional)
```

13.1.1 创建 KITTI 数据集

为了创建 KITTI 点云数据，首先需要加载原始的点云数据并生成相关的包含目标标签和标注框的数据标注文件，同时还需要为 KITTI 数据集生成每个单独的训练目标的点云数据，并将其存储在 `data/kitti/kitti_gt_database` 的 `.bin` 格式的文件中，此外，需要为训练数据或者验证数据生成 `.pkl` 格式的包含数据信息的文件。随后，通过运行下面的命令来创建最终的 KITTI 数据：

```
mkdir ./data/kitti/ && mkdir ./data/kitti/ImageSets

# Download data split
wget -c  https://raw.githubusercontent.com/traveller59/second.pytorch/master/second/
↪data/ImageSets/test.txt --no-check-certificate --content-disposition -O ./data/
↪kitti/ImageSets/test.txt
wget -c  https://raw.githubusercontent.com/traveller59/second.pytorch/master/second/
↪data/ImageSets/train.txt --no-check-certificate --content-disposition -O ./data/
↪kitti/ImageSets/train.txt
wget -c  https://raw.githubusercontent.com/traveller59/second.pytorch/master/second/
↪data/ImageSets/val.txt --no-check-certificate --content-disposition -O ./data/kitti/
↪ImageSets/val.txt
wget -c  https://raw.githubusercontent.com/traveller59/second.pytorch/master/second/
↪data/ImageSets/trainval.txt --no-check-certificate --content-disposition -O ./data/
↪kitti/ImageSets/trainval.txt

python tools/create_data.py kitti --root-path ./data/kitti --out-dir ./data/kitti --
↪extra-tag kitti --with-plane
```

需要注意的是，如果您的本地磁盘没有充足的存储空间来存储转换后的数据，您可以通过改变 `out-dir` 来指定其他任意的存储路径。如果您没有准备 `planes` 数据，您需要移除 `--with-plane` 标志。

处理后的文件夹结构应该如下：

```
kitti
├── ImageSets
|   ├── test.txt
```

(下页继续)

(续上页)

```
|   ├── train.txt
|   ├── trainval.txt
|   ├── val.txt
├── testing
|   ├── calib
|   ├── image_2
|   ├── velodyne
|   ├── velodyne_reduced
├── training
|   ├── calib
|   ├── image_2
|   ├── label_2
|   ├── velodyne
|   ├── velodyne_reduced
|   ├── planes (optional)
├── kitti_gt_database
|   ├── xxxxx.bin
├── kitti_infos_train.pkl
├── kitti_infos_val.pkl
├── kitti_dbinfos_train.pkl
├── kitti_infos_test.pkl
├── kitti_infos_trainval.pkl
├── kitti_infos_train_mono3d.coco.json
├── kitti_infos_trainval_mono3d.coco.json
├── kitti_infos_test_mono3d.coco.json
├── kitti_infos_val_mono3d.coco.json
```

其中的各项文件的含义如下所示：

- `kitti_gt_database/xxxxx.bin`：训练数据集中包含在 3D 标注框中的点云数据
- `kitti_infos_train.pkl`：训练数据集的信息，其中每一帧的信息包含下面的内容：
  - info[‘point_cloud’]: {‘num_features’: 4, ‘velodyne_path’: velodyne_path}.
  - info[‘annos’]: {
    - 位置：其中 x,y,z 为相机参考坐标系下的目标的底部中心（单位为米），是一个尺寸为 Nx3 的数组
    - 维度: 目标的高、宽、长（单位为米），是一个尺寸为 Nx3 的数组
    - 旋转角：相机坐标系下目标绕着 Y 轴的旋转角 ry，其取值范围为 [-pi..pi] ，是一个尺寸为 N 的数组
    - 名称：标准框所包含的目标的名称，是一个尺寸为 N 的数组
    - 困难度：kitti 官方所定义的困难度，包括简单，适中，困难

        * 组别标识符：用于多部件的目标 }

    - (optional) info[‘calib’]: {
        * P0：校对后的 camera0 投影矩阵，是一个 3x4 数组
        * P1：校对后的 camera1 投影矩阵，是一个 3x4 数组
        * P2：校对后的 camera2 投影矩阵，是一个 3x4 数组
        * P3：校对后的 camera3 投影矩阵，是一个 3x4 数组
        * R0_rect：校准旋转矩阵，是一个 4x4 数组
        * Tr_velo_to_cam：从 Velodyne 坐标到相机坐标的变换矩阵，是一个 4x4 数组
        * Tr_imu_to_velo：从 IMU 坐标到 Velodyne 坐标的变换矩阵，是一个 4x4 数组 }
    - (optional) info[‘image’]:{‘image_idx’: idx, ‘image_path’: image_path, ‘image_shape’, image_shape}.

注意：其中的 info[‘annos’] 中的数据均位于相机参考坐标系中，更多的细节请参考此处。

获取 kitti_infos_xxx.pkl 和 kitti_infos_xxx_mono3d.coco.json 的核心函数分别为 get_kitti_image_info 和 get_2d_boxes.

13.2 训练流程

下面展示了一个使用 KITTI 数据集进行 3D 目标检测的典型流程：

```
train_pipeline = [
    dict(
        type='LoadPointsFromFile',
        coord_type='LIDAR',
        load_dim=4, # x, y, z, intensity
        use_dim=4, # x, y, z, intensity
        file_client_args=file_client_args),
    dict(
        type='LoadAnnotations3D',
        with_bbox_3d=True,
        with_label_3d=True,
        file_client_args=file_client_args),
    dict(type='ObjectSample', db_sampler=db_sampler),
    dict(
        type='ObjectNoise',
        num_try=100,
        translation_std=[1.0, 1.0, 0.5],
        global_rot_range=[0.0, 0.0],
        rot_range=[-0.78539816, 0.78539816]),
    dict(type='RandomFlip3D', flip_ratio_bev_horizontal=0.5),
```

(下页继续)

(续上页)

```
    dict(
        type='GlobalRotScaleTrans',
        rot_range=[-0.78539816, 0.78539816],
        scale_ratio_range=[0.95, 1.05]),
    dict(type='PointsRangeFilter', point_cloud_range=point_cloud_range),
    dict(type='ObjectRangeFilter', point_cloud_range=point_cloud_range),
    dict(type='PointShuffle'),
    dict(type='DefaultFormatBundle3D', class_names=class_names),
    dict(type='Collect3D', keys=['points', 'gt_bboxes_3d', 'gt_labels_3d'])
]
```

- 数据增强：
  - `ObjectNoise`：对场景中的每个真实标注框目标添加噪音。
  - `RandomFlip3D`：对输入点云数据进行随机地水平翻转或者垂直翻转。
  - `GlobalRotScaleTrans`：对输入点云数据进行旋转。

13.3 评估

使用 8 个 GPU 以及 KITTI 指标评估的 PointPillars 的示例如下：

```
bash tools/dist_test.sh configs/pointpillars/hv_pointpillars_secfpn_6x8_160e_kitti-3d-
↪3class.py work_dirs/hv_pointpillars_secfpn_6x8_160e_kitti-3d-3class/latest.pth 8 --
↪eval bbox
```

13.4 度量指标

KITTI 官方使用全类平均精度（mAP）和平均方向相似度（AOS）来评估 3D 目标检测的性能，请参考官方网站和论文获取更多细节。

MMDetection3D 采用相同的方法在 KITTI 数据集上进行评估，下面展示了一个评估结果的例子：

```
Car AP@0.70, 0.70, 0.70:
bbox AP:97.9252, 89.6183, 88.1564
bev  AP:90.4196, 87.9491, 85.1700
3d   AP:88.3891, 77.1624, 74.4654
aos  AP:97.70, 89.11, 87.38
Car AP@0.70, 0.50, 0.50:
bbox AP:97.9252, 89.6183, 88.1564
bev  AP:98.3509, 90.2042, 89.6102
```

(下页继续)

(续上页)

```
3d   AP:98.2800, 90.1480, 89.4736
aos  AP:97.70, 89.11, 87.38
```

13.5 测试和提交

使用 8 个 GPU 在 KITTI 上测试 PointPillars 并生成对排行榜的提交的示例如下：

```
mkdir -p results/kitti-3class

./tools/dist_test.sh configs/pointpillars/hv_pointpillars_secfpn_6x8_160e_kitti-3d-
↪3class.py work_dirs/hv_pointpillars_secfpn_6x8_160e_kitti-3d-3class/latest.pth 8 --
↪out results/kitti-3class/results_eval.pkl --format-only --eval-options 'pklfile_
↪prefix=results/kitti-3class/kitti_results' 'submission_prefix=results/kitti-3class/
↪kitti_results'
```

在生成 `results/kitti-3class/kitti_results/xxxxx.txt` 后，您可以提交这些文件到 KITTI 官方网站进行基准测试，请参考 KITTI 官方网站获取更多细节。

CHAPTER 14

3D 目标检测 NuScenes 数据集

本页提供了有关在 MMDetection3D 中使用 nuScenes 数据集的具体教程。

14.1 准备之前

您可以在这里下载 nuScenes 3D 检测数据并解压缩所有 zip 文件。

像准备数据集的一般方法一样，建议将数据集根目录链接到 `$MMDETECTION3D/data`。

在我们处理之前，文件夹结构应按如下方式组织。

```
mmdetection3d
├── mmdet3d
├── tools
├── configs
├── data
|   ├── nuscenes
|   |   ├── maps
|   |   ├── samples
|   |   ├── sweeps
|   |   ├── v1.0-test
|   |   ├── v1.0-trainval
```

14.2 数据准备

我们通常需要通过特定样式来使用.pkl 或.json 文件组织有用的数据信息，例如用于组织图像及其标注的 coco 样式。要为 nuScenes 准备这些文件，请运行以下命令：

```
python tools/create_data.py nuscenes --root-path ./data/nuscenes --out-dir ./data/
↪nuscenes --extra-tag nuscenes
```

处理后的文件夹结构应该如下

```
mmdetection3d
├── mmdet3d
├── tools
├── configs
├── data
|   ├── nuscenes
|   |   ├── maps
|   |   ├── samples
|   |   ├── sweeps
|   |   ├── v1.0-test
|   |   ├── v1.0-trainval
|   |   ├── nuscenes_database
|   |   ├── nuscenes_infos_train.pkl
|   |   ├── nuscenes_infos_trainval.pkl
|   |   ├── nuscenes_infos_val.pkl
|   |   ├── nuscenes_infos_test.pkl
|   |   ├── nuscenes_dbinfos_train.pkl
|   |   ├── nuscenes_infos_train_mono3d.coco.json
|   |   ├── nuscenes_infos_trainval_mono3d.coco.json
|   |   ├── nuscenes_infos_val_mono3d.coco.json
|   |   ├── nuscenes_infos_test_mono3d.coco.json
```

这里，.pkl 文件一般用于涉及点云的方法，coco 风格的.json 文件更适合基于图像的方法，例如基于图像的 2D 和 3D 检测。接下来，我们将详细说明这些信息文件中记录的细节。

- `nuscenes_database/xxxxx.bin`：训练数据集的每个 3D 包围框中包含的点云数据。
- `nuscenes_infos_train.pkl`：训练数据集信息，每帧信息有两个键值：`metadata` 和 `infos`。`metadata` 包含数据集本身的基本信息，例如 `{'version': 'v1.0-trainval'}`，而 `infos` 包含详细信息如下：
  - info[‘lidar_path’]：激光雷达点云数据的文件路径。
  - info[‘token’]：样本数据标记。
  - info[‘sweeps’]：扫描信息（nuScenes 中的 `sweeps` 是指没有标注的中间帧，而 `samples` 是指那些带有标注的关键帧）。

    * info[‘sweeps’][i][‘data_path’]：第 i 次扫描的数据路径。
    * info[‘sweeps’][i][‘type’]：扫描数据类型，例如“激光雷达”。
    * info[‘sweeps’][i][‘sample_data_token’]：扫描样本数据标记。
    * info[‘sweeps’][i][‘sensor2ego_translation’]：从当前传感器（用于收集扫描数据）到自车（包含感知周围环境传感器的车辆，车辆坐标系固连在自车上）的转换（1x3 列表）。
    * info[‘sweeps’][i][‘sensor2ego_rotation’]：从当前传感器（用于收集扫描数据）到自车的旋转（四元数格式的 1x4 列表）。
    * info[‘sweeps’][i][‘ego2global_translation’]：从自车到全局坐标的转换（1x3 列表）。
    * info[‘sweeps’][i][‘ego2global_rotation’]：从自车到全局坐标的旋转（四元数格式的 1x4 列表）。
    * info[‘sweeps’][i][‘timestamp’]：扫描数据的时间戳。
    * info[‘sweeps’][i][‘sensor2lidar_translation’]：从当前传感器（用于收集扫描数据）到激光雷达的转换（1x3 列表）。
    * info[‘sweeps’][i][‘sensor2lidar_rotation’]：从当前传感器（用于收集扫描数据）到激光雷达的旋转（四元数格式的 1x4 列表）。

- info[‘cams’]：相机校准信息。它包含与每个摄像头对应的六个键值：`'CAM_FRONT'`, `'CAM_FRONT_RIGHT'`, `'CAM_FRONT_LEFT'`, `'CAM_BACK'`, `'CAM_BACK_LEFT'`, `'CAM_BACK_RIGHT'`。每个字典包含每个扫描数据按照上述方式的详细信息（每个信息的关键字与上述相同）。除此之外，每个相机还包含了一个键值 `'cam_intrinsic'` 用来保存 3D 点投影到图像平面上需要的内参信息。
- info[‘lidar2ego_translation’]：从激光雷达到自车的转换（1x3 列表）。
- info[‘lidar2ego_rotation’]：从激光雷达到自车的旋转（四元数格式的 1x4 列表）。
- info[‘ego2global_translation’]：从自车到全局坐标的转换（1x3 列表）。
- info[‘ego2global_rotation’]：从自我车辆到全局坐标的旋转（四元数格式的 1x4 列表）。
- info[‘timestamp’]：样本数据的时间戳。
- info[‘gt_boxes’]：7 个自由度的 3D 包围框，一个 Nx7 数组。
- info[‘gt_names’]：3D 包围框的类别，一个 1xN 数组。
- info[‘gt_velocity’]：3D 包围框的速度（由于不准确，没有垂直测量），一个 Nx2 数组。
- info[‘num_lidar_pts’]：每个 3D 包围框中包含的激光雷达点数。
- info[‘num_radar_pts’]：每个 3D 包围框中包含的雷达点数。
- info[‘valid_flag’]：每个包围框是否有效。一般情况下，我们只将包含至少一个激光雷达或雷达点的 3D 框作为有效框。

- `nuscenes_infos_train_mono3d.coco.json`：训练数据集 coco 风格的信息。该文件将基于图像的数据组织为三类（键值）：`'categories'`, `'images'`, `'annotations'`。
  - info[‘categories’]：包含所有类别名称的列表。每个元素都遵循字典格式并由两个键值组成：`'id'` 和 `'name'`。
  - info[‘images’]：包含所有图像信息的列表。
    * info[‘images’][i][‘file_name’]：第 i 张图像的文件名。
    * info[‘images’][i][‘id’]：第 i 张图像的样本数据标记。
    * info[‘images’][i][‘token’]：与该帧对应的样本标记。
    * info[‘images’][i][‘cam2ego_rotation’]：从相机到自车的旋转（四元数格式的 1x4 列表）。
    * info[‘images’][i][‘cam2ego_translation’]：从相机到自车的转换（1x3 列表）。
    * info[‘images’][i][‘ego2global_rotation’’]：从自车到全局坐标的旋转（四元数格式的 1x4 列表）。
    * info[‘images’][i][‘ego2global_translation’]：从自车到全局坐标的转换（1x3 列表）。
    * info[‘images’][i][‘cam_intrinsic’]: 相机内参矩阵（3x3 列表）。
    * info[‘images’][i][‘width’]：图片宽度，nuScenes 中默认为 1600。
    * info[‘images’][i][‘height’]：图像高度，nuScenes 中默认为 900。
  - info[‘annotations’]: 包含所有标注信息的列表。
    * info[‘annotations’][i][‘file_name’]：对应图像的文件名。
    * info[‘annotations’][i][‘image_id’]：对应图像的图像 ID （标记）。
    * info[‘annotations’][i][‘area’]：2D 包围框的面积。
    * info[‘annotations’][i][‘category_name’]：类别名称。
    * info[‘annotations’][i][‘category_id’]：类别 id。
    * info[‘annotations’][i][‘bbox’]：2D 包围框标注（3D 投影框的外部矩形），1x4 列表跟随 [x1, y1, x2-x1, y2-y1]。x1/y1 是沿图像水平/垂直方向的最小坐标。
    * info[‘annotations’][i][‘iscrowd’]：该区域是否拥挤。默认为 0。
    * info[‘annotations’][i][‘bbox_cam3d’]：3D 包围框（重力）中心位置（3）、大小（3）、（全局）偏航角（1）、1x7 列表。
    * info[‘annotations’][i][‘velo_cam3d’]：3D 包围框的速度（由于不准确，没有垂直测量），一个 Nx2 数组。
    * info[‘annotations’][i][‘center2d’]：包含 2.5D 信息的投影 3D 中心：图像上的投影中心位置（2）和深度（1），1x3 列表。
    * info[‘annotations’][i][‘attribute_name’]：属性名称。

  * info[‘annotations’][i][‘attribute_id’]：属性 ID。我们为属性分类维护了一个属性集合和映射。更多的细节请参考这里。
  * info[‘annotations’][i][‘id’]：标注 ID。默认为 `i`。

这里我们只解释训练信息文件中记录的数据。这同样适用于验证和测试集。获取 `nuscenes_infos_xxx.pkl` 和 `nuscenes_infos_xxx_mono3d.coco.json` 的核心函数分别为 _fill_trainval_infos 和 get_2d_boxes。更多细节请参考 nuscenes_converter.py。

14.3 训练流程

14.3.1 基于 LiDAR 的方法

nuScenes 上基于 LiDAR 的 3D 检测（包括多模态方法）的典型训练流程如下。

```
train_pipeline = [
    dict(
        type='LoadPointsFromFile',
        coord_type='LIDAR',
        load_dim=5,
        use_dim=5,
        file_client_args=file_client_args),
    dict(
        type='LoadPointsFromMultiSweeps',
        sweeps_num=10,
        file_client_args=file_client_args),
    dict(type='LoadAnnotations3D', with_bbox_3d=True, with_label_3d=True),
    dict(
        type='GlobalRotScaleTrans',
        rot_range=[-0.3925, 0.3925],
        scale_ratio_range=[0.95, 1.05],
        translation_std=[0, 0, 0]),
    dict(type='RandomFlip3D', flip_ratio_bev_horizontal=0.5),
    dict(type='PointsRangeFilter', point_cloud_range=point_cloud_range),
    dict(type='ObjectRangeFilter', point_cloud_range=point_cloud_range),
    dict(type='ObjectNameFilter', classes=class_names),
    dict(type='PointShuffle'),
    dict(type='DefaultFormatBundle3D', class_names=class_names),
    dict(type='Collect3D', keys=['points', 'gt_bboxes_3d', 'gt_labels_3d'])
]
```

与一般情况相比，nuScenes 有一个特定的 `'LoadPointsFromMultiSweeps'` 流水线来从连续帧加载点云。这是此设置中使用的常见做法。更多细节请参考 nuScenes 原始论文。`'LoadPointsFromMultiSweeps'` 中的默认 `use_dim` 是 `[0, 1, 2, 4]`，其中前 3 个维度是指点坐标，最后一个是指时间戳差异。由于在拼接

来自不同帧的点时使用点云的强度信息会产生噪声，因此默认情况下不使用点云的强度信息。

14.3.2 基于视觉的方法

nuScenes 上基于图像的 3D 检测的典型训练流水线如下。

```
train_pipeline = [
    dict(type='LoadImageFromFileMono3D'),
    dict(
        type='LoadAnnotations3D',
        with_bbox=True,
        with_label=True,
        with_attr_label=True,
        with_bbox_3d=True,
        with_label_3d=True,
        with_bbox_depth=True),
    dict(type='Resize', img_scale=(1600, 900), keep_ratio=True),
    dict(type='RandomFlip3D', flip_ratio_bev_horizontal=0.5),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='Pad', size_divisor=32),
    dict(type='DefaultFormatBundle3D', class_names=class_names),
    dict(
        type='Collect3D',
        keys=[
            'img', 'gt_bboxes', 'gt_labels', 'attr_labels', 'gt_bboxes_3d',
            'gt_labels_3d', 'centers2d', 'depths'
        ]),
]
```

它遵循 2D 检测的一般流水线，但在一些细节上有所不同：

- 它使用单目流水线加载图像，其中包括额外的必需信息，如相机内参矩阵。
- 它需要加载 3D 标注。
- 一些数据增强技术需要调整，例如 `RandomFlip3D`。目前我们不支持更多的增强方法，因为如何迁移和应用其他技术仍在探索中。

14.4 评估

使用 8 个 GPU 以及 nuScenes 指标评估的 PointPillars 的示例如下

```
bash ./tools/dist_test.sh configs/pointpillars/hv_pointpillars_fpn_sbn-all_4x8_2x_nus-
↪3d.py checkpoints/hv_pointpillars_fpn_sbn-all_4x8_2x_nus-3d_20200620_230405-
↪2fa62f3d.pth 8 --eval bbox
```

14.5 指标

NuScenes 提出了一个综合指标，即 nuScenes 检测分数（NDS），以评估不同的方法并设置基准测试。它由平均精度（mAP）、平均平移误差（ATE）、平均尺度误差（ASE）、平均方向误差（AOE）、平均速度误差（AVE）和平均属性误差（AAE）组成。更多细节请参考其官方网站。

我们也采用这种方法对 nuScenes 进行评估。打印的评估结果示例如下：

```
mAP: 0.3197
mATE: 0.7595
mASE: 0.2700
mAOE: 0.4918
mAVE: 1.3307
mAAE: 0.1724
NDS: 0.3905
Eval time: 170.8s

Per-class results:
Object Class    AP      ATE     ASE     AOE     AVE     AAE
car     0.503   0.577   0.152   0.111   2.096   0.136
truck   0.223   0.857   0.224   0.220   1.389   0.179
bus     0.294   0.855   0.204   0.190   2.689   0.283
trailer 0.081   1.094   0.243   0.553   0.742   0.167
construction_vehicle    0.058   1.017   0.450   1.019   0.137   0.341
pedestrian      0.392   0.687   0.284   0.694   0.876   0.158
motorcycle      0.317   0.737   0.265   0.580   2.033   0.104
bicycle 0.308   0.704   0.299   0.892   0.683   0.010
traffic_cone    0.555   0.486   0.309   nan     nan     nan
barrier 0.466   0.581   0.269   0.169   nan     nan
```

14.6 测试和提交

使用 8 个 GPU 在 nuScenes 上测试 PointPillars 并生成对排行榜的提交的示例如下

```
./tools/dist_test.sh configs/pointpillars/hv_pointpillars_fpn_sbn-all_4x8_2x_nus-3d.
↪py work_dirs/hv_pointpillars_secfpn_6x8_160e_kitti-3d-3class/latest.pth 8 --out␣
↪work_dirs/pp-nus/results_eval.pkl --format-only --eval-options 'jsonfile_
↪prefix=work_dirs/pp-nus/results_eval'
```

请注意，在这里测试信息应更改为测试集而不是验证集。

生成 `work_dirs/pp-nus/results_eval.json` 后，您可以压缩并提交给 nuScenes 基准测试。更多信息请参考 nuScenes 官方网站。

我们还可以使用我们开发的可视化工具将预测结果可视化。更多细节请参考可视化文档。

14.7 注意

14.7.1 `NuScenesBox` 和我们的 `CameraInstanceBoxes` 之间的转换。

总的来说，`NuScenesBox` 和我们的 `CameraInstanceBoxes` 的主要区别主要体现在转向角（yaw）定义上。`NuScenesBox` 定义了一个四元数或三个欧拉角的旋转，而我们的由于实际情况只定义了一个转向角（yaw），它需要我们在预处理和后处理中手动添加一些额外的旋转，例如这里。

另外，请注意，角点和位置的定义在 `NuScenesBox` 中是分离的。例如，在单目 3D 检测中，框位置的定义在其相机坐标中（有关汽车设置，请参阅其官方插图），即与我们的一致。相比之下，它的角点是通过惯例定义的，“x 向前，y 向左，z 向上”。它导致了与我们的 `CameraInstanceBoxes` 不同的维度和旋转定义理念。一个移除相似冲突的例子是 PR #744。同样的问题也存在于 LiDAR 系统中。为了解决它们，我们通常会在预处理和后处理中添加一些转换，以保证在整个训练和推理过程中框都在我们的坐标系系统里。

CHAPTER 15

3D 目标检测 Lyft 数据集

本页提供了有关在 MMDetection3D 中使用 Lyft 数据集的具体教程。

15.1 准备之前

您可以在这里下载 Lyft 3D 检测数据并解压缩所有 zip 文件。

像准备数据集的一般方法一样，建议将数据集根目录链接到 `$MMDETECTION3D/data`。

在进行处理之前，文件夹结构应按如下方式组织：

```
mmdetection3d
├── mmdet3d
├── tools
├── configs
├── data
│   ├── lyft
│   │   ├── v1.01-train
│   │   │   ├── v1.01-train (train_data)
│   │   │   ├── lidar (train_lidar)
│   │   │   ├── images (train_images)
│   │   │   ├── maps (train_maps)
│   │   ├── v1.01-test
│   │   │   ├── v1.01-test (test_data)
│   │   │   ├── lidar (test_lidar)
```

(下页继续)

(续上页)

```
|   |   |   ├── images (test_images)
|   |   |   ├── maps (test_maps)
|   |   ├── train.txt
|   |   ├── val.txt
|   |   ├── test.txt
|   |   ├── sample_submission.csv
```

其中 `v1.01-train` 和 `v1.01-test` 包含与 nuScenes 数据集相同的元文件，`.txt` 文件包含数据划分的信息。Lyft 不提供训练集和验证集的官方划分方案，因此 MMDetection3D 对不同场景下的不同类别的目标数量进行分析，并提供了一个数据集划分方案。`sample_submission.csv` 是用于提交到 Kaggle 评估服务器的基本文件。需要注意的是，我们遵循了 Lyft 最初的文件夹命名以实现更清楚的文件组织。请将下载下来的原始文件夹重命名按照上述组织结构重新命名。

15.2 数据准备

组织 Lyft 数据集的方式和组织 nuScenes 的方式相同，首先会生成几乎具有相同结构的.pkl 和.json 文件，接着需要重点关注这两个数据集之间的不同点，请参考 nuScenes 教程获取更加详细的数据集信息文件结构的说明。

请通过运行下面的命令来生成 Lyft 的数据集信息文件：

```
python tools/create_data.py lyft --root-path ./data/lyft --out-dir ./data/lyft --
↪extra-tag lyft --version v1.01
python tools/data_converter/lyft_data_fixer.py --version v1.01 --root-folder ./data/
↪lyft
```

请注意，上面的第二行命令用于修复损坏的 lidar 数据文件，请参考此处获取更多细节。

处理后的文件夹结构应该如下：

```
mmdetection3d
├── mmdet3d
├── tools
├── configs
├── data
|   ├── lyft
|   |   ├── v1.01-train
|   |   |   ├── v1.01-train (train_data)
|   |   |   ├── lidar (train_lidar)
|   |   |   ├── images (train_images)
|   |   |   ├── maps (train_maps)
|   |   ├── v1.01-test
|   |   |   ├── v1.01-test (test_data)
```

(下页继续)

(续上页)

```
|   |   |   ├── lidar (test_lidar)
|   |   |   ├── images (test_images)
|   |   |   ├── maps (test_maps)
|   |   ├── train.txt
|   |   ├── val.txt
|   |   ├── test.txt
|   |   ├── sample_submission.csv
|   |   ├── lyft_infos_train.pkl
|   |   ├── lyft_infos_val.pkl
|   |   ├── lyft_infos_test.pkl
|   |   ├── lyft_infos_train_mono3d.coco.json
|   |   ├── lyft_infos_val_mono3d.coco.json
|   |   ├── lyft_infos_test_mono3d.coco.json
```

其中，.pkl 文件通常适用于涉及到点云的相关方法，coco 类型的.json 文件更加适用于涉及到基于图像的相关方法，如基于图像的 2D 和 3D 目标检测。不同于 nuScenes 数据集，这里仅能使用 json 文件进行 2D 检测相关的实验，未来将会进一步支持基于图像的 3D 检测。

接下来将详细介绍 Lyft 数据集和 nuScenes 数据集之间的数据集信息文件中的不同点：

- `lyft_database/xxxxx.bin` 文件不存在：由于真实标注框的采样对实验的影响可以忽略不计，在 Lyft 数据集中不会提取该目录和相关的 `.bin` 文件。
- `lyft_infos_train.pkl`：包含训练数据集信息，每一帧包含两个关键字：`metadata` 和 `infos`。`metadata` 包含数据集自身的基础信息，如 `{'version': 'v1.01-train'}`，然而 `infos` 包含和 nuScenes 数据集相似的数据集详细信息，但是并不包含一下几点：
  - info[‘sweeps’]：扫描信息.
    * info[‘sweeps’][i][‘type’]：扫描信息的数据类型，如 `'lidar'`。Lyft 数据集中的一些样例具有不同的 LiDAR 设置，然而为了数据分布的一致性，这里将一直采用顶部的 LiDAR 设备所采集的数据点信息。
  - info[‘gt_names’]：在 Lyft 数据集中有 9 个类别，相比于 nuScenes 数据集，不同类别的标注不平衡问题更加突出。
  - info[‘gt_velocity’] 不存在：Lyft 数据集中不存在速度评估信息。
  - info[‘num_lidar_pts’]：默认值设置为 -1。
  - info[‘num_radar_pts’]：默认值设置为 0。
  - info[‘valid_flag’] 不存在：这个标志信息因无效的 `num_lidar_pts` 和 `num_radar_pts` 的存在而存在。
- `nuscenes_infos_train_mono3d.coco.json`：包含 coco 类型的训练数据集相关的信息。这个文件仅包含 2D 相关的信息，不包含 3D 目标检测所需要的信息，如相机内参。
  - info[‘images’]：包含所有图像信息的列表。

        * 仅包含 'file_name', 'id', 'width', 'height'。

    - info[‘annotations’]：包含所有标注信息的列表。

        * 仅包含 'file_name', 'image_id', 'area', 'category_name', 'category_id', 'bbox','is_crowd','segmentation','id',其中 'is_crowd' 和 'segmentation' 默认设置为 0 和 []。Lyft 数据集中不包含属性标注信息。

这里仅介绍存储在训练数据文件的数据记录信息，在测试数据集也采用上述的数据记录方式。

获取 lyft_infos_xxx.pkl 的核心函数是 _fill_trainval_infos。请参考 lyft_converter.py 获取更多细节。

15.3 训练流程

15.3.1 基于 LiDAR 的方法

Lyft 上基于 LiDAR 的 3D 检测（包括多模态方法）的训练流程与 nuScenes 几乎相同，如下所示：

```
train_pipeline = [
    dict(
        type='LoadPointsFromFile',
        coord_type='LIDAR',
        load_dim=5,
        use_dim=5,
        file_client_args=file_client_args),
    dict(
        type='LoadPointsFromMultiSweeps',
        sweeps_num=10,
        file_client_args=file_client_args),
    dict(type='LoadAnnotations3D', with_bbox_3d=True, with_label_3d=True),
    dict(
        type='GlobalRotScaleTrans',
        rot_range=[-0.3925, 0.3925],
        scale_ratio_range=[0.95, 1.05],
        translation_std=[0, 0, 0]),
    dict(type='RandomFlip3D', flip_ratio_bev_horizontal=0.5),
    dict(type='PointsRangeFilter', point_cloud_range=point_cloud_range),
    dict(type='ObjectRangeFilter', point_cloud_range=point_cloud_range),
    dict(type='PointShuffle'),
    dict(type='DefaultFormatBundle3D', class_names=class_names),
    dict(type='Collect3D', keys=['points', 'gt_bboxes_3d', 'gt_labels_3d'])
]
```

与 nuScenes 相似，在 Lyft 上进行训练的模型也需要 LoadPointsFromMultiSweeps 步骤来从连续帧中加载点云数据。另外，考虑到 Lyft 中所收集的激光雷达点的强度是无效的，因此将

`LoadPointsFromMultiSweeps` 中的 `use_dim` 默认值设置为 `[0, 1, 2, 4]`，其中前三个维度表示点的坐标，最后一个维度表示时间戳的差异。

15.4 评估

使用 8 个 GPU 以及 Lyft 指标评估的 PointPillars 的示例如下：

```
bash ./tools/dist_test.sh configs/pointpillars/hv_pointpillars_fpn_sbn-all_2x8_2x_
↪lyft-3d.py checkpoints/hv_pointpillars_fpn_sbn-all_2x8_2x_lyft-3d_20210517_202818-
↪fc6904c3.pth 8 --eval bbox
```

15.5 度量指标

Lyft 提出了一个更加严格的用以评估所预测的 3D 检测框的度量指标。判断一个预测框是否是正类的基本评判标准和 KITTI 一样，如基于 3D 交并比进行评估，然而，Lyft 采用与 COCO 相似的方式来计算平均精度–计算 3D 交并比在 0.5-0.95 之间的不同阈值下的平均精度。实际上，重叠部分大于 0.7 的 3D 交并比是一项对于 3D 检测方法比较严格的标准，因此整体的性能似乎会偏低。相比于其他数据集，Lyft 上不同类别的标注不平衡是导致最终结果偏低的另一个重要原因。请参考官方网址获取更多关于度量指标的定义的细节。

这里将采用官方方法对 Lyft 进行评估，下面展示了一个评估结果的例子：

```
+mAPs@0.5:0.95------+--------------+
| class             | mAP@0.5:0.95 |
+-------------------+--------------+
| animal            | 0.0          |
| bicycle           | 0.099        |
| bus               | 0.177        |
| car               | 0.422        |
| emergency_vehicle | 0.0          |
| motorcycle        | 0.049        |
| other_vehicle     | 0.359        |
| pedestrian        | 0.066        |
| truck             | 0.176        |
| Overall           | 0.15         |
+-------------------+--------------+
```

15.6 测试和提交

使用 8 个 GPU 在 Lyft 上测试 PointPillars 并生成对排行榜的提交的示例如下：

```
./tools/dist_test.sh configs/pointpillars/hv_pointpillars_fpn_sbn-all_2x8_2x_lyft-3d.
↪py work_dirs/pp-lyft/latest.pth 8 --out work_dirs/pp-lyft/results_challenge.pkl --
↪format-only --eval-options 'jsonfile_prefix=work_dirs/pp-lyft/results_challenge'
↪'csv_savepath=results/pp-lyft/results_challenge.csv'
```

在生成 `work_dirs/pp-lyft/results_challenge.csv`，您可以将生成的文件提交到 Kaggle 评估服务器，请参考官方网址获取更多细节。

同时还可以使用可视化工具将预测结果进行可视化，请参考可视化文档获取更多细节。

CHAPTER 16

Waymo 数据集

本文档页包含了关于 MMDetection3D 中 Waymo 数据集用法的教程。

16.1 数据集准备

在准备 Waymo 数据集之前，如果您之前只安装了 `requirements/build.txt` 和 `requirements/runtime.txt` 中的依赖，请通过运行如下指令额外安装 Waymo 数据集所依赖的官方包：

```
# tf 2.1.0.
pip install waymo-open-dataset-tf-2-1-0==1.2.0
# tf 2.0.0
# pip install waymo-open-dataset-tf-2-0-0==1.2.0
# tf 1.15.0
# pip install waymo-open-dataset-tf-1-15-0==1.2.0
```

或者

```
pip install -r requirements/optional.txt
```

和准备数据集的通用方法一致，我们推荐将数据集根目录软链接至 `$MMDETECTION3D/data`。由于原始 Waymo 数据的格式基于 `tfrecord`，我们需要将原始数据进行预处理，以便于训练和测试时使用。我们的方法是将它们转换为 KITTI 格式。

处理之前，文件目录结构组织如下：

```
mmdetection3d
├── mmdet3d
├── tools
├── configs
├── data
│   ├── waymo
│   │   ├── waymo_format
│   │   │   ├── training
│   │   │   ├── validation
│   │   │   ├── testing
│   │   │   ├── gt.bin
│   │   ├── kitti_format
│   │   │   ├── ImageSets
```

您可以在这里下载 1.2 版本的 Waymo 公开数据集，并在这里下载其训练/验证/测试集拆分文件。接下来，请将 `tfrecord` 文件放入 `data/waymo/waymo_format/` 下的对应文件夹，并将 txt 格式的数据集拆分文件放入 `data/waymo/kitti_format/ImageSets`。在这里下载验证集使用的 bin 格式真实标注 (Ground Truth) 文件并放入 `data/waymo/waymo_format/`。小窍门：您可以使用 `gsutil` 来在命令行下载大规模数据集。您可以将该工具 作为一个例子来查看更多细节。之后，通过运行如下指令准备 Waymo 数据：

```
python tools/create_data.py waymo --root-path ./data/waymo/ --out-dir ./data/waymo/ --
↪workers 128 --extra-tag waymo
```

请注意，如果您的本地磁盘没有足够空间保存转换后的数据，您可以将 `--out-dir` 改为其他目录；只要在创建文件夹、准备数据并转换格式后，将数据文件链接到 `data/waymo/kitti_format` 即可。

在数据转换后，文件目录结构应组织如下：

```
mmdetection3d
├── mmdet3d
├── tools
├── configs
├── data
│   ├── waymo
│   │   ├── waymo_format
│   │   │   ├── training
│   │   │   ├── validation
│   │   │   ├── testing
│   │   │   ├── gt.bin
│   │   ├── kitti_format
│   │   │   ├── ImageSets
│   │   │   ├── training
│   │   │   │   ├── calib
```

(下页继续)

(续上页)

```
|   |   |   |   ├── image_0
|   |   |   |   ├── image_1
|   |   |   |   ├── image_2
|   |   |   |   ├── image_3
|   |   |   |   ├── image_4
|   |   |   |   ├── label_0
|   |   |   |   ├── label_1
|   |   |   |   ├── label_2
|   |   |   |   ├── label_3
|   |   |   |   ├── label_4
|   |   |   |   ├── label_all
|   |   |   |   ├── pose
|   |   |   |   ├── velodyne
|   |   |   ├── testing
|   |   |   |   ├── (the same as training)
|   |   |   ├── waymo_gt_database
|   |   |   ├── waymo_infos_trainval.pkl
|   |   |   ├── waymo_infos_train.pkl
|   |   |   ├── waymo_infos_val.pkl
|   |   |   ├── waymo_infos_test.pkl
|   |   |   ├── waymo_dbinfos_train.pkl
```

因为 Waymo 数据的来源包含数个相机，这里我们将每个相机对应的图像和标签文件分别存储，并将相机位姿 (pose) 文件存储下来以供后续处理连续多帧的点云。我们使用 `{a}{bbb}{ccc}` 的名称编码方式为每帧数据命名，其中 `a` 是不同数据拆分的前缀（`0` 指代训练集，`1` 指代验证集，`2` 指代测试集），`bbb` 是分割部分 (segment) 的索引，而 `ccc` 是帧索引。您可以轻而易举地按照如上命名规则定位到所需的帧。我们将训练和验证所需数据按 KITTI 的方式集合在一起，然后将训练集/验证集/测试集的索引存储在 `ImageSet` 下的文件中。

16.2 训练

考虑到原始数据集中的数据有很多相似的帧，我们基本上可以主要使用一个子集来训练我们的模型。在我们初步的基线中，我们在每五帧图片中加载一帧。得益于我们的超参数设置和数据增强方案，我们得到了比 Waymo 原论文中更好的性能。请移步 `configs/pointpillars/` 下的 README.md 以查看更多配置和性能相关的细节。我们会尽快发布一个更完整的 Waymo 基准榜单 (benchmark)。

16.3 评估

为了在 Waymo 数据集上进行检测性能评估，请按照此处指示构建用于计算评估指标的二进制文件 compute_detection_metrics_main，并将它置于 mmdet3d/core/evaluation/waymo_utils/ 下。您基本上可以按照下方命令安装 bazel，然后构建二进制文件：

```
# download the code and enter the base directory
git clone https://github.com/waymo-research/waymo-open-dataset.git waymo-od
cd waymo-od
git checkout remotes/origin/master

# use the Bazel build system
sudo apt-get install --assume-yes pkg-config zip g++ zlib1g-dev unzip python3 python3-
↪pip
BAZEL_VERSION=3.1.0
wget https://github.com/bazelbuild/bazel/releases/download/${BAZEL_VERSION}/bazel-$
↪{BAZEL_VERSION}-installer-linux-x86_64.sh
sudo bash bazel-${BAZEL_VERSION}-installer-linux-x86_64.sh
sudo apt install build-essential

# configure .bazelrc
./configure.sh
# delete previous bazel outputs and reset internal caches
bazel clean

bazel build waymo_open_dataset/metrics/tools/compute_detection_metrics_main
cp bazel-bin/waymo_open_dataset/metrics/tools/compute_detection_metrics_main ../
↪mmdetection3d/mmdet3d/core/evaluation/waymo_utils/
```

接下来，您就可以在 Waymo 上评估您的模型了。如下示例是使用 8 个图形处理器 (GPU) 在 Waymo 上用 Waymo 评价指标评估 PointPillars 模型的情景：

```
./tools/slurm_test.sh ${PARTITION} ${JOB_NAME} configs/pointpillars/hv_pointpillars_
↪secfpn_sbn-2x16_2x_waymo-3d-car.py \
    checkpoints/hv_pointpillars_secfpn_sbn-2x16_2x_waymo-3d-car_latest.pth --out␣
↪results/waymo-car/results_eval.pkl \
    --eval waymo --eval-options 'pklfile_prefix=results/waymo-car/kitti_results' \
    'submission_prefix=results/waymo-car/kitti_results'
```

如果需要生成 bin 文件，应在 --eval-options 中给出 pklfile_prefix。对于评价指标，waymo 是我们推荐的官方评估原型。目前，kitti 这一评估选项是从 KITTI 迁移而来的，且每个难度下的评估结果和 KITTI 数据集中定义得到的不尽相同——目前大多数物体被标记为难度 0（日后会修复）。kitti 评估选项的不稳定来源于很大的计算量，转换的数据中遮挡 (occlusion) 和截断 (truncation) 的缺失，难度的不同定义方式，以及不同的平均精度 (Average Precision) 计算方式。

注意:

1. 有时用 `bazel` 构建 `compute_detection_metrics_main` 的过程中会出现如下错误：`'round' 不是 'std' 的成员 ('round' is not a member of 'std')`。我们只需要移除该文件中，`round` 前的 `std::`。
2. 考虑到 Waymo 上评估一次耗时不短，我们建议只在模型训练结束时进行评估。
3. 为了在 CUDA 9 环境使用 TensorFlow，我们建议通过编译 TensorFlow 源码的方式使用。除了官方教程之外，您还可以参考该链接以寻找可能合适的预编译包以及编译源码的实用攻略。

16.4 测试并提交到官方服务器

如下是一个使用 8 个图形处理器在 Waymo 上测试 PointPillars，生成 bin 文件并提交结果到官方榜单的例子：

```
./tools/slurm_test.sh ${PARTITION} ${JOB_NAME} configs/pointpillars/hv_pointpillars_
↪secfpn_sbn-2x16_2x_waymo-3d-car.py \
    checkpoints/hv_pointpillars_secfpn_sbn-2x16_2x_waymo-3d-car_latest.pth --out␣
↪results/waymo-car/results_eval.pkl \
    --format-only --eval-options 'pklfile_prefix=results/waymo-car/kitti_results' \
    'submission_prefix=results/waymo-car/kitti_results'
```

在生成 bin 文件后，您可以简单地构建二进制文件 `create_submission`，并按照指示 创建一个提交文件。下面是一些示例：

```
cd ../waymo-od/
bazel build waymo_open_dataset/metrics/tools/create_submission
cp bazel-bin/waymo_open_dataset/metrics/tools/create_submission ../mmdetection3d/
↪mmdet3d/core/evaluation/waymo_utils/
vim waymo_open_dataset/metrics/tools/submission.txtpb  # set the metadata information
cp waymo_open_dataset/metrics/tools/submission.txtpb ../mmdetection3d/mmdet3d/core/
↪evaluation/waymo_utils/

cd ../mmdetection3d
# suppose the result bin is in `results/waymo-car/submission`
mmdet3d/core/evaluation/waymo_utils/create_submission  --input_filenames='results/
↪waymo-car/kitti_results_test.bin' --output_filename='results/waymo-car/submission/
↪model' --submission_filename='mmdet3d/core/evaluation/waymo_utils/submission.txtpb'

tar cvf results/waymo-car/submission/my_model.tar results/waymo-car/submission/my_
↪model/
gzip results/waymo-car/submission/my_model.tar
```

如果想用官方评估服务器评估您在验证集上的结果，您可以使用同样的方法生成提交文件，只需确保您在运行如上指令前更改 `submission.txtpb` 中的字段值即可。

CHAPTER 17

3D 目标检测 SUN RGB-D 数据集

17.1 数据集的准备

对于数据集准备的整体流程，请参考 SUN RGB-D 的指南。

17.1.1 下载 SUN RGB-D 数据与工具包

在这里下载 SUN RGB-D 的数据。接下来，将 SUNRGBD.zip、SUNRGBDMeta2DBB_v2.mat、SUNRGBDMeta3DBB_v2.mat 和 SUNRGBDtoolbox.zip 移动到 OFFICIAL_SUNRGBD 文件夹，并解压文件。

下载完成后，数据处理之前的文件目录结构如下：

```
sunrgbd
├── README.md
├── matlab
│   ├── extract_rgbd_data_v1.m
│   ├── extract_rgbd_data_v2.m
│   ├── extract_split.m
├── OFFICIAL_SUNRGBD
│   ├── SUNRGBD
│   ├── SUNRGBDMeta2DBB_v2.mat
│   ├── SUNRGBDMeta3DBB_v2.mat
│   ├── SUNRGBDtoolbox
```

17.1.2 从原始数据中提取 3D 检测所需数据与标注

通过运行如下指令从原始文件中提取出 SUN RGB-D 的标注（这需要您的机器中安装了 MATLAB）：

```
matlab -nosplash -nodesktop -r 'extract_split;quit;'
matlab -nosplash -nodesktop -r 'extract_rgbd_data_v2;quit;'
matlab -nosplash -nodesktop -r 'extract_rgbd_data_v1;quit;'
```

主要的步骤包括：

- 提取出训练集和验证集的索引文件；
- 从原始数据中提取出 3D 检测所需要的数据；
- 从原始的标注数据中提取并组织检测任务使用的标注数据。

用于从深度图中提取点云数据的 `extract_rgbd_data_v2.m` 的主要部分如下：

```
data = SUNRGBDMeta(imageId);
data.depthpath(1:16) = '';
data.depthpath = strcat('../OFFICIAL_SUNRGBD', data.depthpath);
data.rgbpath(1:16) = '';
data.rgbpath = strcat('../OFFICIAL_SUNRGBD', data.rgbpath);

% 从深度图获取点云
[rgb,points3d,depthInpaint,imsize]=read3dPoints(data);
rgb(isnan(points3d(:,1)),:) = [];
points3d(isnan(points3d(:,1)),:) = [];
points3d_rgb = [points3d, rgb];

% MAT 文件比 TXT 文件小三倍。在 Python 中我们可以使用
% scipy.io.loadmat('xxx.mat')['points3d_rgb'] 来加载数据
mat_filename = strcat(num2str(imageId,'%06d'), '.mat');
txt_filename = strcat(num2str(imageId,'%06d'), '.txt');
% 保存点云数据
parsave(strcat(depth_folder, mat_filename), points3d_rgb);
```

用于提取并组织检测任务标注的 `extract_rgbd_data_v1.m` 的主要部分如下：

```
% 输出 2D 和 3D 包围框
data2d = data;
fid = fopen(strcat(det_label_folder, txt_filename), 'w');
for j = 1:length(data.groundtruth3DBB)
    centroid = data.groundtruth3DBB(j).centroid;  % 3D 包围框中心
    classname = data.groundtruth3DBB(j).classname;  % 类名
    orientation = data.groundtruth3DBB(j).orientation;  % 3D 包围框方向
    coeffs = abs(data.groundtruth3DBB(j).coeffs);  % 3D 包围框大小
```

(下页继续)

(续上页)

```
    box2d = data2d.groundtruth2DBB(j).gtBb2D;  % 2D 包围框
    fprintf(fid, '%s %d %d %d %d %f %f %f %f %f %f %f %f\n', classname, box2d(1),␣
↪box2d(2), box2d(3), box2d(4), centroid(1), centroid(2), centroid(3), coeffs(1),␣
↪coeffs(2), coeffs(3), orientation(1), orientation(2));
end
fclose(fid);
```

上面的两个脚本调用了 SUN RGB-D 提供的工具包中的一些函数，如 `read3dPoints`。

使用上述脚本提取数据后，文件目录结构应如下：

```
sunrgbd
├── README.md
├── matlab
│   ├── extract_rgbd_data_v1.m
│   ├── extract_rgbd_data_v2.m
│   ├── extract_split.m
├── OFFICIAL_SUNRGBD
│   ├── SUNRGBD
│   ├── SUNRGBDMeta2DBB_v2.mat
│   ├── SUNRGBDMeta3DBB_v2.mat
│   ├── SUNRGBDtoolbox
├── sunrgbd_trainval
│   ├── calib
│   ├── depth
│   ├── image
│   ├── label
│   ├── label_v1
│   ├── seg_label
│   ├── train_data_idx.txt
│   ├── val_data_idx.txt
```

在如下每个文件夹下，都有总计 5285 个训练集样本和 5050 个验证集样本：

- `calib`：`.txt` 后缀的相机标定文件。
- `depth`：`.mat` 后缀的点云文件，包含 xyz 坐标和 rgb 色彩值。
- `image`：`.jpg` 后缀的二维图像文件。
- `label`：`.txt` 后缀的用于检测任务的标注数据（版本二）。
- `label_v1`：`.txt` 后缀的用于检测任务的标注数据（版本一）。
- `seg_label`：`.txt` 后缀的用于分割任务的标注数据。

目前，我们使用版本一的数据用于训练与测试，因此版本二的标注并未使用。

17.1.3 创建数据集

请运行如下指令创建数据集：

```
python tools/create_data.py sunrgbd --root-path ./data/sunrgbd \
--out-dir ./data/sunrgbd --extra-tag sunrgbd
```

或者，如果使用 slurm，可以使用如下指令替代：

```
bash tools/create_data.sh <job_name> sunrgbd
```

之前提到的点云数据就会被处理并以 .bin 格式重新存储。与此同时，.pkl 文件也被生成，用于存储数据标注和元信息。这一步处理中，用于生成 .pkl 文件的核心函数 process_single_scene 如下：

```
def process_single_scene(sample_idx):
    print(f'{self.split} sample_idx: {sample_idx}')
    # 将深度图转换为点云并降采样点云
    pc_upright_depth = self.get_depth(sample_idx)
    pc_upright_depth_subsampled = random_sampling(
        pc_upright_depth, self.num_points)

    info = dict()
    pc_info = {'num_features': 6, 'lidar_idx': sample_idx}
    info['point_cloud'] = pc_info

    # 将点云保存为 `.bin` 格式
    mmcv.mkdir_or_exist(osp.join(self.root_dir, 'points'))
    pc_upright_depth_subsampled.tofile(
        osp.join(self.root_dir, 'points', f'{sample_idx:06d}.bin'))

    # 存储点云存储路径
    info['pts_path'] = osp.join('points', f'{sample_idx:06d}.bin')

    # 存储图像存储路径以及其元信息
    img_path = osp.join('image', f'{sample_idx:06d}.jpg')
    image_info = {
        'image_idx': sample_idx,
        'image_shape': self.get_image_shape(sample_idx),
        'image_path': img_path
    }
    info['image'] = image_info

    # 保存标定信息
    K, Rt = self.get_calibration(sample_idx)
    calib_info = {'K': K, 'Rt': Rt}
```

(下页继续)

(续上页)

```
    info['calib'] = calib_info

    # 保存所有数据标注
    if has_label:
        obj_list = self.get_label_objects(sample_idx)
        annotations = {}
        annotations['gt_num'] = len([
            obj.classname for obj in obj_list
            if obj.classname in self.cat2label.keys()
        ])
        if annotations['gt_num'] != 0:
            # 类别名称
            annotations['name'] = np.array([
                obj.classname for obj in obj_list
                if obj.classname in self.cat2label.keys()
            ])
            # 二维图像包围框
            annotations['bbox'] = np.concatenate([
                obj.box2d.reshape(1, 4) for obj in obj_list
                if obj.classname in self.cat2label.keys()
            ], axis=0)
            # depth 坐标系下的三维包围框中心坐标
            annotations['location'] = np.concatenate([
                obj.centroid.reshape(1, 3) for obj in obj_list
                if obj.classname in self.cat2label.keys()
            ], axis=0)
            # depth 坐标系下的三维包围框大小
            annotations['dimensions'] = 2 * np.array([
                [obj.l, obj.h, obj.w] for obj in obj_list
                if obj.classname in self.cat2label.keys()
            ])
            # depth 坐标系下的三维包围框旋转角
            annotations['rotation_y'] = np.array([
                obj.heading_angle for obj in obj_list
                if obj.classname in self.cat2label.keys()
            ])
            annotations['index'] = np.arange(
                len(obj_list), dtype=np.int32)
            # 类别标签（数字）
            annotations['class'] = np.array([
                self.cat2label[obj.classname] for obj in obj_list
                if obj.classname in self.cat2label.keys()
            ])
```

(下页继续)

(续上页)

```
            # depth 坐标系下的三维包围框
            annotations['gt_boxes_upright_depth'] = np.stack(
                [
                    obj.box3d for obj in obj_list
                    if obj.classname in self.cat2label.keys()
                ], axis=0)  # (K,8)
        info['annos'] = annotations
    return info
```

如上数据处理后，文件目录结构应如下：

```
sunrgbd
├── README.md
├── matlab
│   ├── ...
├── OFFICIAL_SUNRGBD
│   ├── ...
├── sunrgbd_trainval
│   ├── ...
├── points
├── sunrgbd_infos_train.pkl
├── sunrgbd_infos_val.pkl
```

- `points/0xxxxx.bin`：降采样后的点云数据。
- `sunrgbd_infos_train.pkl`：训练集数据信息（标注与元信息），每个场景所含数据信息具体如下：
  - info[‘point_cloud’]：`{'num_features': 6, 'lidar_idx': sample_idx}`，其中 `sample_idx` 为该场景的索引。
  - info[‘pts_path’]：`points/0xxxxx.bin` 的路径。
  - info[‘image’]：图像路径与元信息：
    - image[‘image_idx’]：图像索引。
    - image[‘image_shape’]：图像张量的形状（即其尺寸）。
    - image[‘image_path’]：图像路径。
  - info[‘annos’]：每个场景的标注：
    - annotations[‘gt_num’]：真实物体 (ground truth) 的数量。
    - annotations[‘name’]：所有真实物体的语义类别名称，比如 `chair`（椅子）。
    - annotations[‘location’]：depth 坐标系下三维包围框的重力中心 (gravity center)，形状为 [K, 3]，其中 K 是真实物体的数量。
    - annotations[‘dimensions’]：depth 坐标系下三维包围框的大小，形状为 [K, 3]。

    * annotations[‘rotation_y’]：depth 坐标系下三维包围框的旋转角，形状为 [K,]。
    * annotations[‘gt_boxes_upright_depth’]：depth 坐标系下三维包围框 (x, y, z, x_size, y_size, z_size, yaw)，形状为 [K, 7]。
    * annotations[‘bbox’]：二维包围框 (x, y, x_size, y_size)，形状为 [K, 4]。
    * annotations[‘index’]：所有真实物体的索引，范围为 [0, K)。
    * annotations[‘class’]：所有真实物体类别的标号，范围为 [0, 10)，形状为 [K,]。

- sunrgbd_infos_val.pkl：验证集上的数据信息，与 sunrgbd_infos_train.pkl 格式完全一致。

17.2 训练流程

SUN RGB-D 上纯点云 3D 物体检测的典型流程如下：

```
train_pipeline = [
    dict(
        type='LoadPointsFromFile',
        coord_type='DEPTH',
        shift_height=True,
        load_dim=6,
        use_dim=[0, 1, 2]),
    dict(type='LoadAnnotations3D'),
    dict(
        type='RandomFlip3D',
        sync_2d=False,
        flip_ratio_bev_horizontal=0.5,
    ),
    dict(
        type='GlobalRotScaleTrans',
        rot_range=[-0.523599, 0.523599],
        scale_ratio_range=[0.85, 1.15],
        shift_height=True),
    dict(type='PointSample', num_points=20000),
    dict(type='DefaultFormatBundle3D', class_names=class_names),
    dict(type='Collect3D', keys=['points', 'gt_bboxes_3d', 'gt_labels_3d'])
]
```

点云上的数据增强

- RandomFlip3D：随机左右或前后翻转输入点云。
- GlobalRotScaleTrans：旋转输入点云，对于 SUN RGB-D 角度通常落入 [-30, 30]（度）的范围；并放缩输入点云，对于 SUN RGB-D 比例通常落入 [0.85, 1.15] 的范围；最后平移输入点云，对于 SUN RGB-D

通常位移量为 0（即不做位移）。

- `PointSample`：降采样输入点云。

SUN RGB-D 上多模态（点云和图像）3D 物体检测的典型流程如下：

```
train_pipeline = [
    dict(
        type='LoadPointsFromFile',
        coord_type='DEPTH',
        shift_height=True,
        load_dim=6,
        use_dim=[0, 1, 2]),
    dict(type='LoadImageFromFile'),
    dict(type='LoadAnnotations3D'),
    dict(type='LoadAnnotations', with_bbox=True),
    dict(type='Resize', img_scale=(1333, 600), keep_ratio=True),
    dict(type='RandomFlip', flip_ratio=0.0),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='Pad', size_divisor=32),
    dict(
        type='RandomFlip3D',
        sync_2d=False,
        flip_ratio_bev_horizontal=0.5,
    ),
    dict(
        type='GlobalRotScaleTrans',
        rot_range=[-0.523599, 0.523599],
        scale_ratio_range=[0.85, 1.15],
        shift_height=True),
    dict(type='PointSample', num_points=20000),
    dict(type='DefaultFormatBundle3D', class_names=class_names),
    dict(
        type='Collect3D',
        keys=[
            'img', 'gt_bboxes', 'gt_labels', 'points', 'gt_bboxes_3d',
            'gt_labels_3d'
        ])
]
```

图像上的数据增强/归一化

- `Resize`: 改变输入图像的大小, `keep_ratio=True` 意味着图像的比例不改变。
- `Normalize`: 归一化图像的 RGB 通道。
- `RandomFlip`: 随机地翻折图像。

- `Pad`: 扩大图像，默认情况下用零填充图像的边缘。

图像增强和归一化函数的实现取自 MMDetection。

17.3 度量指标

与 ScanNet 一样，通常 mAP（全类平均精度）被用于 SUN RGB-D 的检测任务的评估，比如 `mAP@0.25` 和 `mAP@0.5`。具体来说，评估时一个通用的计算 3D 物体检测多个类别的精度和召回率的函数被调用，可以参考 `indoor_eval.py`。

因为 SUN RGB-D 包含有图像数据，所以图像上的物体检测也是可行的。举个例子，在 ImVoteNet 中，我们首先训练了一个图像检测器，并且也使用 mAP 指标，如 `mAP@0.5`，来评估其表现。我们使用 MMDetection 库中的 `eval_map` 函数来计算 mAP。

CHAPTER 18

3D 目标检测 Scannet 数据集

18.1 数据集准备

请参考 ScanNet 的指南以查看总体流程。

18.1.1 提取 ScanNet 点云数据

通过提取 ScanNet 数据，我们加载原始点云文件，并生成包括语义标签、实例标签和真实物体包围框在内的相关标注。

```
python batch_load_scannet_data.py
```

数据处理之前的文件目录结构如下：

```
mmdetection3d
├── mmdet3d
├── tools
├── configs
├── data
|   ├── scannet
|   |   ├── meta_data
|   |   ├── scans
|   |   |   ├── scenexxxx_xx
|   |   ├── batch_load_scannet_data.py
```

(下页继续)

(续上页)

```
|   |   ├── load_scannet_data.py
|   |   ├── scannet_utils.py
|   |   ├── README.md
```

在 scans 文件夹下总共有 1201 个训练样本文件夹和 312 个验证样本文件夹，其中存有未处理的点云数据和相关的标注。比如说，在文件夹 scene0001_01 下文件是这样组织的：

- scene0001_01_vh_clean_2.ply: 存有每个顶点坐标和颜色的网格文件。网格的顶点被直接用作未处理的点云数据。
- scene0001_01.aggregation.json: 包含物体 ID、分割部分 ID、标签的标注文件。
- scene0001_01_vh_clean_2.0.010000.segs.json: 包含分割部分 ID 和顶点的分割标注文件。
- scene0001_01.txt: 包括对齐矩阵等的元文件。
- scene0001_01_vh_clean_2.labels.ply：包含每个顶点类别的标注文件。

通过运行 python batch_load_scannet_data.py 来提取 ScanNet 数据。主要步骤包括：

- 从原始文件中提取出点云、实例标签、语义标签和包围框标签文件。
- 下采样原始点云并过滤掉不合法的类别。
- 保存处理后的点云数据和相关的标注文件。

load_scannet_data.py 中的核心函数 export 如下：

```python
def export(mesh_file,
           agg_file,
           seg_file,
           meta_file,
           label_map_file,
           output_file=None,
           test_mode=False):

    # 标签映射文件：./data/scannet/meta_data/scannetv2-labels.combined.tsv
    # 该标签映射文件中有多种标签标准，比如 'nyu40id'
    label_map = scannet_utils.read_label_mapping(
        label_map_file, label_from='raw_category', label_to='nyu40id')
    # 加载原始点云数据，特征包括 6 维：XYZRGB
    mesh_vertices = scannet_utils.read_mesh_vertices_rgb(mesh_file)

    # 加载场景坐标轴对齐矩阵：一个 4x4 的变换矩阵
    # 将传感器坐标系下的原始点转化到另一个坐标系下
    # 该坐标系与房屋的两边平行（也就是与坐标轴平行）
    lines = open(meta_file).readlines()
    # 测试集的数据没有对齐矩阵
```

(下页继续)

(续上页)

```
    axis_align_matrix = np.eye(4)
    for line in lines:
        if 'axisAlignment' in line:
            axis_align_matrix = [
                float(x)
                for x in line.rstrip().strip('axisAlignment = ').split(' ')
            ]
            break
    axis_align_matrix = np.array(axis_align_matrix).reshape((4, 4))

    # 对网格顶点进行全局的对齐
    pts = np.ones((mesh_vertices.shape[0], 4))
    # 同种类坐标下的原始点云，每一行的数据是 [x, y, z, 1]
    pts[:, 0:3] = mesh_vertices[:, 0:3]
    # 将原始网格顶点转换为对齐后的顶点
    pts = np.dot(pts, axis_align_matrix.transpose())  # Nx4
    aligned_mesh_vertices = np.concatenate([pts[:, 0:3], mesh_vertices[:, 3:]],
                                           axis=1)

    # 加载语义与实例标签
    if not test_mode:
        # 每个物体都有一个语义标签，并且包含几个分割部分
        object_id_to_segs, label_to_segs = read_aggregation(agg_file)
        # 很多点属于同一分割部分
        seg_to_verts, num_verts = read_segmentation(seg_file)
        label_ids = np.zeros(shape=(num_verts), dtype=np.uint32)
        object_id_to_label_id = {}
        for label, segs in label_to_segs.items():
            label_id = label_map[label]
            for seg in segs:
                verts = seg_to_verts[seg]
                # 每个点都有一个语义标签
                label_ids[verts] = label_id
        instance_ids = np.zeros(
            shape=(num_verts), dtype=np.uint32)  # 0: 未标注的
        for object_id, segs in object_id_to_segs.items():
            for seg in segs:
                verts = seg_to_verts[seg]
                # object_id 从 1 开始计数，比如 1,2,3,.,,,.NUM_INSTANCES
                # 每个点都属于一个物体
                instance_ids[verts] = object_id
                if object_id not in object_id_to_label_id:
                    object_id_to_label_id[object_id] = label_ids[verts][0]
```

(下页继续)

(续上页)

```
        # 包围框格式为 [x, y, z, dx, dy, dz, label_id]
        # [x, y, z] 是包围框的重力中心, [dx, dy, dz] 是与坐标轴平行的
        # [label_id] 是 'nyu40id' 标准下的语义标签
        # 注意: 因为三维包围框是与坐标轴平行的, 所以旋转角是 0
        unaligned_bboxes = extract_bbox(mesh_vertices, object_id_to_segs,
                                        object_id_to_label_id, instance_ids)
        aligned_bboxes = extract_bbox(aligned_mesh_vertices, object_id_to_segs,
                                      object_id_to_label_id, instance_ids)
    ...

    return mesh_vertices, label_ids, instance_ids, unaligned_bboxes, \
        aligned_bboxes, object_id_to_label_id, axis_align_matrix
```

在从每个场景的扫描文件提取数据后，如果原始点云点数过多，可以将其下采样（比如到 50000 个点），但在三维语义分割任务中，点云不会被下采样。此外，在 `nyu40id` 标准之外的不合法语义标签或者可选的 `DONOT CARE` 类别标签应被过滤。最终，点云文件、语义标签、实例标签和真实物体的集合应被存储于 `.npy` 文件中。

18.1.2 提取 ScanNet RGB 色彩数据（可选的）

通过提取 ScanNet RGB 色彩数据，对于每个场景我们加载 RGB 图像与配套 4x4 位姿矩阵、单个 4x4 相机内参矩阵的集合。请注意，这一步是可选的，除非要运行多视图物体检测，否则可以略去这步。

```
python extract_posed_images.py
```

1201 个训练样本，312 个验证样本和 100 个测试样本中的每一个都包含一个单独的 `.sens` 文件。比如说，对于场景 `0001_01` 我们有 `data/scannet/scans/scene0001_01/0001_01.sens`。对于这个场景所有图像和位姿数据都被提取至 `data/scannet/posed_images/scene0001_01`。具体来说，该文件夹下会有 300 个 xxxxx.jpg 格式的图像数据，300 个 xxxxx.txt 格式的相机位姿数据和一个单独的 `intrinsic.txt` 内参文件。通常来说，一个场景包含数千张图像。默认情况下，我们只会提取其中的 300 张，从而只占用少于 100 Gb 的空间。要想提取更多图像，请使用 `--max-images-per-scene` 参数。

18.1.3 创建数据集

```
python tools/create_data.py scannet --root-path ./data/scannet \
--out-dir ./data/scannet --extra-tag scannet
```

上述提取的点云文件，语义类别标注文件，和物体实例标注文件被进一步以 `.bin` 格式保存。与此同时 `.pkl` 格式的文件被生成并用于训练和验证。获取数据信息的核心函数 `process_single_scene` 如下：

```
def process_single_scene(sample_idx):

    # 分别以 .bin 格式保存点云文件，语义类别标注文件和物体实例标注文件
    # 获取 info['pts_path']，info['pts_instance_mask_path'] 和 info['pts_semantic_mask_
↪path']
    ...

    # 获取标注
    if has_label:
        annotations = {}
        # 包围框的形状为 [k, 6 + class]
        aligned_box_label = self.get_aligned_box_label(sample_idx)
        unaligned_box_label = self.get_unaligned_box_label(sample_idx)
        annotations['gt_num'] = aligned_box_label.shape[0]
        if annotations['gt_num'] != 0:
            aligned_box = aligned_box_label[:, :-1]  # k, 6
            unaligned_box = unaligned_box_label[:, :-1]
            classes = aligned_box_label[:, -1]  # k
            annotations['name'] = np.array([
                self.label2cat[self.cat_ids2class[classes[i]]]
                for i in range(annotations['gt_num'])
            ])
            # 为了向后兼容，默认的参数名赋予了与坐标轴平行的包围框
            # 我们同时保存了对应的与坐标轴不平行的包围框的信息
            annotations['location'] = aligned_box[:, :3]
            annotations['dimensions'] = aligned_box[:, 3:6]
            annotations['gt_boxes_upright_depth'] = aligned_box
            annotations['unaligned_location'] = unaligned_box[:, :3]
            annotations['unaligned_dimensions'] = unaligned_box[:, 3:6]
            annotations[
                'unaligned_gt_boxes_upright_depth'] = unaligned_box
            annotations['index'] = np.arange(
                annotations['gt_num'], dtype=np.int32)
            annotations['class'] = np.array([
                self.cat_ids2class[classes[i]]
                for i in range(annotations['gt_num'])
            ])
        axis_align_matrix = self.get_axis_align_matrix(sample_idx)
        annotations['axis_align_matrix'] = axis_align_matrix  # 4x4
        info['annos'] = annotations
    return info
```

如上数据处理后，文件目录结构应如下：

```
scannet
├── meta_data
├── batch_load_scannet_data.py
├── load_scannet_data.py
├── scannet_utils.py
├── README.md
├── scans
├── scans_test
├── scannet_instance_data
├── points
│   ├── xxxxx.bin
├── instance_mask
│   ├── xxxxx.bin
├── semantic_mask
│   ├── xxxxx.bin
├── seg_info
│   ├── train_label_weight.npy
│   ├── train_resampled_scene_idxs.npy
│   ├── val_label_weight.npy
│   ├── val_resampled_scene_idxs.npy
├── posed_images
│   ├── scenexxxx_xx
│   │   ├── xxxxxx.txt
│   │   ├── xxxxxx.jpg
│   │   ├── intrinsic.txt
├── scannet_infos_train.pkl
├── scannet_infos_val.pkl
├── scannet_infos_test.pkl
```

- `points/xxxxx.bin`：下采样后，未与坐标轴平行（即没有对齐）的点云。因为 ScanNet 3D 检测任务将与坐标轴平行的点云作为输入，而 ScanNet 3D 语义分割任务将对齐前的点云作为输入，我们选择存储对齐前的点云和它们的对齐矩阵。请注意：在 3D 检测的预处理流程 `GlobalAlignment` 后，点云就都是与坐标轴平行的了。
- `instance_mask/xxxxx.bin`：每个点的实例标签，值的范围为：[0, NUM_INSTANCES]，其中 0 表示没有标注。
- `semantic_mask/xxxxx.bin`：每个点的语义标签，值的范围为：[1, 40], 也就是 `nyu40id` 的标准。请注意：在训练流程 `PointSegClassMapping` 中，`nyu40id` 的 ID 会被映射到训练 ID。
- `posed_images/scenexxxx_xx`：`.jpg` 图像的集合，还包含 `.txt` 格式的 4x4 相机姿态和单个 `.txt` 格式的相机内参矩阵文件。
- `scannet_infos_train.pkl`：训练集的数据信息，每个场景的具体信息如下：
  - info[‘point_cloud’]：`{'num_features': 6, 'lidar_idx': sample_idx}`，其中 `sample_idx` 为该场景的索引。

  - info[‘pts_path’]：`points/xxxxx.bin` 的路径。
  - info[‘pts_instance_mask_path’]：`instance_mask/xxxxx.bin` 的路径。
  - info[‘pts_semantic_mask_path’]：`semantic_mask/xxxxx.bin` 的路径。
  - info[‘annos’]：每个场景的标注。
    * annotations[‘gt_num’]：真实物体 (ground truth) 的数量。
    * annotations[‘name’]：所有真实物体的语义类别名称，比如 `chair`（椅子）。
    * annotations[‘location’]：depth 坐标系下与坐标轴平行的三维包围框的重力中心 (gravity center)，形状为 [K, 3]，其中 K 是真实物体的数量。
    * annotations[‘dimensions’]：depth 坐标系下与坐标轴平行的三维包围框的大小，形状为 [K, 3]。
    * annotations[‘gt_boxes_upright_depth’]：depth 坐标系下与坐标轴平行的三维包围框 `(x, y, z, x_size, y_size, z_size, yaw)`，形状为 [K, 6]。
    * annotations[‘unaligned_location’]：depth 坐标系下与坐标轴不平行（对齐前）的三维包围框的重力中心。
    * annotations[‘unaligned_dimensions’]：depth 坐标系下与坐标轴不平行的三维包围框的大小。
    * annotations[‘unaligned_gt_boxes_upright_depth’]：depth 坐标系下与坐标轴不平行的三维包围框。
    * annotations[‘index’]：所有真实物体的索引，范围为 [0, K)。
    * annotations[‘class’]：所有真实物体类别的标号，范围为 [0, 18)，形状为 [K,]。

- `scannet_infos_val.pkl`：验证集上的数据信息，与 `scannet_infos_train.pkl` 格式完全一致。
- `scannet_infos_test.pkl`：测试集上的数据信息，与 `scannet_infos_train.pkl` 格式几乎完全一致，除了缺少标注。

18.2 训练流程

ScanNet 上 3D 物体检测的典型流程如下：

```
train_pipeline = [
    dict(
        type='LoadPointsFromFile',
        coord_type='DEPTH',
        shift_height=True,
        load_dim=6,
        use_dim=[0, 1, 2]),
    dict(
```

(下页继续)

(续上页)

```
        type='LoadAnnotations3D',
        with_bbox_3d=True,
        with_label_3d=True,
        with_mask_3d=True,
        with_seg_3d=True),
    dict(type='GlobalAlignment', rotation_axis=2),
    dict(
        type='PointSegClassMapping',
        valid_cat_ids=(3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 24, 28, 33, 34,
                       36, 39),
        max_cat_id=40),
    dict(type='PointSample', num_points=40000),
    dict(
        type='RandomFlip3D',
        sync_2d=False,
        flip_ratio_bev_horizontal=0.5,
        flip_ratio_bev_vertical=0.5),
    dict(
        type='GlobalRotScaleTrans',
        rot_range=[-0.087266, 0.087266],
        scale_ratio_range=[1.0, 1.0],
        shift_height=True),
    dict(type='DefaultFormatBundle3D', class_names=class_names),
    dict(
        type='Collect3D',
        keys=[
            'points', 'gt_bboxes_3d', 'gt_labels_3d', 'pts_semantic_mask',
            'pts_instance_mask'
        ])
]
```

- `GlobalAlignment`：输入的点云在施加了坐标轴平行的矩阵后应被转换为与坐标轴平行的形式。
- `PointSegClassMapping`：训练中，只有合法的类别 ID 才会被映射到类别标签，比如 [0, 18)。
- 数据增强:
  - `PointSample`：下采样输入点云。
  - `RandomFlip3D`：随机左右或前后翻转点云。
  - `GlobalRotScaleTrans`: 旋转输入点云，对于 ScanNet 角度通常落入 [-5, 5]（度）的范围；并放缩输入点云，对于 ScanNet 比例通常为 1.0（即不做缩放）；最后平移输入点云，对于 ScanNet 通常位移量为 0（即不做位移）。

18.3 评估指标

通常 mAP（全类平均精度）被用于 ScanNet 的检测任务的评估，比如 `mAP@0.25` 和 `mAP@0.5`。具体来说，评估时一个通用的计算 3D 物体检测多个类别的精度和召回率的函数被调用，可以参考 indoor_eval。

与在章节提取 `ScanNet` 数据中介绍的那样，所有真实物体的三维包围框是与坐标轴平行的，也就是说旋转角为 0。因此，预测包围框的网络接受的包围框旋转角监督也是 0，且在后处理阶段我们使用适用于与坐标轴平行的包围框的非极大值抑制 (NMS)，该过程不会考虑包围框的旋转。

CHAPTER 19

3D 语义分割 ScanNet 数据集

19.1 数据集的准备

ScanNet 3D 语义分割数据集的准备和 3D 检测任务的准备很相似，请查看此文档以获取更多细节。以下我们只罗列部分 3D 语义分割特有的处理步骤和数据信息。

19.1.1 提取 ScanNet 数据

因为 ScanNet 测试集对 3D 语义分割任务提供在线评测的基准，我们也需要下载其测试集并置于 `scannet` 目录下。数据预处理前的文件目录结构应如下所示：

```
mmdetection3d
├── mmdet3d
├── tools
├── configs
├── data
│   ├── scannet
│   │   ├── meta_data
│   │   ├── scans
│   │   │   ├── scenexxxx_xx
│   │   ├── scans_test
│   │   │   ├── scenexxxx_xx
│   │   ├── batch_load_scannet_data.py
│   │   ├── load_scannet_data.py
```

(下页继续)

(续上页)

```
|   |   ├── scannet_utils.py
|   |   ├── README.md
```

在 scans_test 目录下有 100 个测试集 scan 的文件夹，每个文件夹仅包含了原始点云数据和基础的数据元文件。例如，在 scene0707_00 这一目录下的文件如下所示：

- scene0707_00_vh_clean_2.ply：原始网格文件，存储有每个顶点的坐标和颜色。网格的顶点会被选取作为处理后点云中的点。
- scene0707_00.txt：数据的元文件，包含数据采集传感器的参数等信息。注意，与 scans 目录下的数据 (训练集和验证集) 不同，测试集 scan 并没有提供用于和坐标轴对齐的变换矩阵 (axis-aligned matrix)。

用户可以通过运行 python batch_load_scannet_data.py 指令来从原始文件中提取 ScanNet 数据。注意，测试集只会保存下点云数据，因为没有提供标注信息。

19.1.2 创建数据集

与 3D 检测任务类似，我们通过运行 python tools/create_data.py scannet --root-path ./data/scannet --out-dir ./data/scannet --extra-tag scannet 指令即可创建 ScanNet 数据集。预处理后的数据目录结构如下所示：

```
scannet
├── scannet_utils.py
├── batch_load_scannet_data.py
├── load_scannet_data.py
├── scannet_utils.py
├── README.md
├── scans
├── scans_test
├── scannet_instance_data
├── points
|   ├── xxxxx.bin
├── instance_mask
|   ├── xxxxx.bin
├── semantic_mask
|   ├── xxxxx.bin
├── seg_info
|   ├── train_label_weight.npy
|   ├── train_resampled_scene_idxs.npy
|   ├── val_label_weight.npy
|   ├── val_resampled_scene_idxs.npy
├── scannet_infos_train.pkl
```

(下页继续)

(续上页)

```
├── scannet_infos_val.pkl
├── scannet_infos_test.pkl
```

- `seg_info`：为支持语义分割任务所生成的信息文件。
  - `train_label_weight.npy`：每一语义类别的权重系数。因为 ScanNet 中属于不同类的点的数量相差很大，一个常见的操作是在计算损失时对不同类别进行加权 (label re-weighting) 以得到更好的分割性能。
  - `train_resampled_scene_idxs.npy`：每一个场景 (房间) 的重采样标签。在训练过程中，我们依据每个场景的点的数量，会对其进行不同次数的重采样，以保证训练数据均衡。

19.2 训练流程

ScanNet 上 3D 语义分割的一种典型数据载入流程如下所示：

```
train_pipeline = [
    dict(
        type='LoadPointsFromFile',
        coord_type='DEPTH',
        shift_height=False,
        use_color=True,
        load_dim=6,
        use_dim=[0, 1, 2, 3, 4, 5]),
    dict(
        type='LoadAnnotations3D',
        with_bbox_3d=False,
        with_label_3d=False,
        with_mask_3d=False,
        with_seg_3d=True),
    dict(
        type='PointSegClassMapping',
        valid_cat_ids=(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 24, 28,
                       33, 34, 36, 39),
        max_cat_id=40),
    dict(
        type='IndoorPatchPointSample',
        num_points=num_points,
        block_size=1.5,
        ignore_index=len(class_names),
        use_normalized_coord=False,
        enlarge_size=0.2,
        min_unique_num=None),
```

(下页继续)

(续上页)

```
    dict(type='NormalizePointsColor', color_mean=None),
    dict(type='DefaultFormatBundle3D', class_names=class_names),
    dict(type='Collect3D', keys=['points', 'pts_semantic_mask'])
]
```

- `PointSegClassMapping`：在训练过程中，只有被使用的类别的序号会被映射到类似 [0, 20) 范围内的类别标签。其余的类别序号会被转换为 `ignore_index` 所制定的忽略标签，在本例中是 `20`。
- `IndoorPatchPointSample`：从输入点云中裁剪一个含有固定数量点的小块 (patch)。`block_size` 指定了裁剪块的边长，在 ScanNet 上这个数值一般设置为 `1.5`。
- `NormalizePointsColor`：将输入点的颜色信息归一化，通过将 RGB 值除以 `255` 来实现。

19.3 度量指标

通常我们使用平均交并比 (mean Intersection over Union, mIoU) 作为 ScanNet 语义分割任务的度量指标。具体而言，我们先计算所有类别的 IoU，然后取平均值作为 mIoU。更多实现细节请参考 seg_eval.py。

19.4 在测试集上测试并提交结果

默认情况下，MMDet3D 的代码是在训练集上进行模型训练，然后在验证集上进行模型测试。如果你也想在在线基准上测试模型的性能，请在测试命令中加上 `--format-only` 的标记，同时也要将 ScanNet 数据集配置文件中的 `ann_file=data_root + 'scannet_infos_val.pkl'` 改成 `ann_file=data_root + 'scannet_infos_test.pkl'`。请记得通过 `txt_prefix` 来指定想要保存测试结果的文件夹名称。

以 PointNet++ (SSG) 在 ScanNet 上的测试为例，你可以运行以下命令来完成测试结果的保存：

```
./tools/dist_test.sh configs/pointnet2/pointnet2_ssg_16x2_cosine_200e_scannet_seg-3d-
↪20class.py \
    work_dirs/pointnet2_ssg/latest.pth --format-only \
    --eval-options txt_prefix=work_dirs/pointnet2_ssg/test_submission
```

在保存测试结果后，你可以将该文件夹压缩，然后提交到 ScanNet 在线测试服务器上进行验证。

CHAPTER 20

3D 语义分割 S3DIS 数据集

20.1 数据集的准备

对于数据集准备的整体流程，请参考 S3DIS 的指南。

20.1.1 提取 S3DIS 数据

通过从原始数据中提取 S3DIS 数据，我们将点云数据读取并保存下相关的标注信息，例如语义分割标签和实例分割标签。

数据提取前的目录结构应该如下所示：

```
mmdetection3d
├── mmdet3d
├── tools
├── configs
├── data
│   ├── s3dis
│   │   ├── meta_data
│   │   ├── Stanford3dDataset_v1.2_Aligned_Version
│   │   │   ├── Area_1
│   │   │   │   ├── conferenceRoom_1
│   │   │   │   ├── office_1
│   │   │   │   ├── ...
```

(下页继续)

(续上页)

```
|   |   |   ├── Area_2
|   |   |   ├── Area_3
|   |   |   ├── Area_4
|   |   |   ├── Area_5
|   |   |   ├── Area_6
|   |   ├── indoor3d_util.py
|   |   ├── collect_indoor3d_data.py
|   |   ├── README.md
```

在 Stanford3dDataset_v1.2_Aligned_Version 目录下，所有房间依据所属区域被分为 6 组。我们通常使用 5 个区域进行训练，然后在余下 1 个区域上进行测试 (被余下的 1 个区域通常为区域 5)。在每个区域的目录下包含有多个房间的文件夹，每个文件夹是一个房间的原始点云数据和相关的标注信息。例如，在 Area_1/office_1 目录下的文件如下所示：

- office_1.txt：一个 txt 文件存储着原始点云数据每个点的坐标和颜色信息。
- Annotations/：这个文件夹里包含有此房间中实例物体的信息 (以 txt 文件的形式存储)。每个 txt 文件表示一个实例，例如：
  - chair_1.txt：存储有该房间中一把椅子的点云数据。

  如果我们将 Annotations/ 下的所有 txt 文件合并起来，得到的点云就和 office_1.txt 中的点云是一致的。

你可以通过 python collect_indoor3d_data.py 指令进行 S3DIS 数据的提取。主要步骤包括：

- 从原始 txt 文件中读取点云数据、语义分割标签和实例分割标签。
- 将点云数据和相关标注文件存储下来。

这其中的核心函数 indoor3d_util.py 中的 export 函数实现如下：

```python
def export(anno_path, out_filename):
    """ 将原始数据集的文件转化为点云、语义分割标签和实例分割掩码文件。
    我们将同一房间中所有实例的点进行聚合。

    参数列表:
        anno_path (str): 标注信息的路径，例如 Area_1/office_2/Annotations/
        out_filename (str): 保存点云和标签的路径
        file_format (str): txt 或 numpy，指定保存的文件格式

    注意:
        点云在处理过程中被整体移动了，保存下的点最小位于原点 (即没有负数坐标值)
    """
    points_list = []
    ins_idx = 1  # 实例标签从 1 开始，因此最终实例标签为 0 的点就是无标注的点
```

(下页继续)

(续上页)

```
    # `anno_path` 的一个例子: Area_1/office_1/Annotations
    # 其中以 txt 文件存储有该房间中所有实例物体的点云
    for f in glob.glob(osp.join(anno_path, '*.txt')):
        # get class name of this instance
        one_class = osp.basename(f).split('_')[0]
        if one_class not in class_names:  # 某些房间有 'staris' 类物体
            one_class = 'clutter'
        points = np.loadtxt(f)
        labels = np.ones((points.shape[0], 1)) * class2label[one_class]
        ins_labels = np.ones((points.shape[0], 1)) * ins_idx
        ins_idx += 1
        points_list.append(np.concatenate([points, labels, ins_labels], 1))

    data_label = np.concatenate(points_list, 0)  # [N, 8], (pts, rgb, sem, ins)
    # 将点云对齐到原点
    xyz_min = np.amin(data_label, axis=0)[0:3]
    data_label[:, 0:3] -= xyz_min

    np.save(f'{out_filename}_point.npy', data_label[:, :6].astype(np.float32))
    np.save(f'{out_filename}_sem_label.npy', data_label[:, 6].astype(np.int))
    np.save(f'{out_filename}_ins_label.npy', data_label[:, 7].astype(np.int))
```

上述代码中，我们读取 Annotations/ 下的所有点云实例，将其合并得到整体房屋的点云，同时生成语义/实例分割的标签。在提取完每个房间的数据后，点云、语义分割和实例分割的标签文件应以 .npy 的格式被保存下来。

20.1.2 创建数据集

```
python tools/create_data.py s3dis --root-path ./data/s3dis \
--out-dir ./data/s3dis --extra-tag s3dis
```

上述指令首先读取以 .npy 格式存储的点云、语义分割和实例分割标签文件，然后进一步将它们以 .bin 格式保存。同时，每个区域 .pkl 格式的信息文件也会被保存下来。

数据预处理后的目录结构如下所示：

```
s3dis
├── meta_data
├── indoor3d_util.py
├── collect_indoor3d_data.py
├── README.md
├── Stanford3dDataset_v1.2_Aligned_Version
```

(下页继续)

(续上页)

```
├── s3dis_data
├── points
│   ├── xxxxx.bin
├── instance_mask
│   ├── xxxxx.bin
├── semantic_mask
│   ├── xxxxx.bin
├── seg_info
│   ├── Area_1_label_weight.npy
│   ├── Area_1_resampled_scene_idxs.npy
│   ├── Area_2_label_weight.npy
│   ├── Area_2_resampled_scene_idxs.npy
│   ├── Area_3_label_weight.npy
│   ├── Area_3_resampled_scene_idxs.npy
│   ├── Area_4_label_weight.npy
│   ├── Area_4_resampled_scene_idxs.npy
│   ├── Area_5_label_weight.npy
│   ├── Area_5_resampled_scene_idxs.npy
│   ├── Area_6_label_weight.npy
│   ├── Area_6_resampled_scene_idxs.npy
├── s3dis_infos_Area_1.pkl
├── s3dis_infos_Area_2.pkl
├── s3dis_infos_Area_3.pkl
├── s3dis_infos_Area_4.pkl
├── s3dis_infos_Area_5.pkl
├── s3dis_infos_Area_6.pkl
```

- `points/xxxxx.bin`：提取的点云数据。
- `instance_mask/xxxxx.bin`：每个点云的实例标签，取值范围为 [0, ${实例个数}]，其中 0 代表未标注的点。
- `semantic_mask/xxxxx.bin`：每个点云的语义标签，取值范围为 [0, 12]。
- `s3dis_infos_Area_1.pkl`：区域 1 的数据信息，每个房间的详细信息如下：
  - info[‘point_cloud’]: {‘num_features’: 6, ‘lidar_idx’: sample_idx}.
  - info[‘pts_path’]: `points/xxxxx.bin` 点云的路径。
  - info[‘pts_instance_mask_path’]: `instance_mask/xxxxx.bin` 实例标签的路径。
  - info[‘pts_semantic_mask_path’]: `semantic_mask/xxxxx.bin` 语义标签的路径。
- `seg_info`：为支持语义分割任务所生成的信息文件。
  - `Area_1_label_weight.npy`：每一语义类别的权重系数。因为 S3DIS 中属于不同类的点的数量相差很大，一个常见的操作是在计算损失时对不同类别进行加权 (label re-weighting) 以得到更好

的分割性能。

- `Area_1_resampled_scene_idxs.npy`：每一个场景 (房间) 的重采样标签。在训练过程中，我们依据每个场景的点的数量，会对其进行不同次数的重采样，以保证训练数据均衡。

20.2 训练流程

S3DIS 上 3D 语义分割的一种典型数据载入流程如下所示：

```
class_names = ('ceiling', 'floor', 'wall', 'beam', 'column', 'window', 'door',
               'table', 'chair', 'sofa', 'bookcase', 'board', 'clutter')
num_points = 4096
train_pipeline = [
    dict(
        type='LoadPointsFromFile',
        coord_type='DEPTH',
        shift_height=False,
        use_color=True,
        load_dim=6,
        use_dim=[0, 1, 2, 3, 4, 5]),
    dict(
        type='LoadAnnotations3D',
        with_bbox_3d=False,
        with_label_3d=False,
        with_mask_3d=False,
        with_seg_3d=True),
    dict(
        type='PointSegClassMapping',
        valid_cat_ids=tuple(range(len(class_names))),
        max_cat_id=13),
    dict(
        type='IndoorPatchPointSample',
        num_points=num_points,
        block_size=1.0,
        ignore_index=None,
        use_normalized_coord=True,
        enlarge_size=None,
        min_unique_num=num_points // 4,
        eps=0.0),
    dict(type='NormalizePointsColor', color_mean=None),
    dict(
        type='GlobalRotScaleTrans',
        rot_range=[-3.141592653589793, 3.141592653589793],  # [-pi, pi]
        scale_ratio_range=[0.8, 1.2],
```

(下页继续)

(续上页)

```
        translation_std=[0, 0, 0]),
    dict(
        type='RandomJitterPoints',
        jitter_std=[0.01, 0.01, 0.01],
        clip_range=[-0.05, 0.05]),
    dict(type='RandomDropPointsColor', drop_ratio=0.2),
    dict(type='DefaultFormatBundle3D', class_names=class_names),
    dict(type='Collect3D', keys=['points', 'pts_semantic_mask'])
]
```

- `PointSegClassMapping`：在训练过程中，只有被使用的类别的序号会被映射到类似 [0, 13) 范围内的类别标签。其余的类别序号会被转换为 `ignore_index` 所制定的忽略标签，在本例中是 `13`。
- `IndoorPatchPointSample`：从输入点云中裁剪一个含有固定数量点的小块 (patch)。`block_size` 指定了裁剪块的边长，在 S3DIS 上这个数值一般设置为 `1.0`。
- `NormalizePointsColor`：将输入点的颜色信息归一化，通过将 RGB 值除以 `255` 来实现。
- 数据增广：
  - `GlobalRotScaleTrans`：对输入点云进行随机旋转和放缩变换。
  - `RandomJitterPoints`：通过对每一个点施加不同的噪声向量以实现对点云的随机扰动。
  - `RandomDropPointsColor`：以 `drop_ratio` 的概率随机将点云的颜色值全部置零。

20.3 度量指标

通常我们使用平均交并比 (mean Intersection over Union, mIoU) 作为 ScanNet 语义分割任务的度量指标。具体而言，我们先计算所有类别的 IoU，然后取平均值作为 mIoU。更多实现细节请参考 seg_eval.py。

正如在 提取 `S3DIS` 数据一节中所提及的，S3DIS 通常在 5 个区域上进行训练，然后在余下的 1 个区域上进行测试。但是在其他论文中，也有不同的划分方式。为了便于灵活划分训练和测试的子集，我们首先定义子数据集 (sub-dataset) 来表示每一个区域，然后根据区域划分对其进行合并，以得到完整的训练集。以下是在区域 1、2、3、4、6 上训练并在区域 5 上测试的一个配置文件例子：

```
dataset_type = 'S3DISSegDataset'
data_root = './data/s3dis/'
class_names = ('ceiling', 'floor', 'wall', 'beam', 'column', 'window', 'door',
               'table', 'chair', 'sofa', 'bookcase', 'board', 'clutter')
train_area = [1, 2, 3, 4, 6]
test_area = 5
data = dict(
    train=dict(
        type=dataset_type,
```

(下页继续)

(续上页)

```
        data_root=data_root,
        ann_files=[
            data_root + f's3dis_infos_Area_{i}.pkl' for i in train_area
        ],
        pipeline=train_pipeline,
        classes=class_names,
        test_mode=False,
        ignore_index=len(class_names),
        scene_idxs=[
            data_root + f'seg_info/Area_{i}_resampled_scene_idxs.npy'
            for i in train_area
        ]),
    val=dict(
        type=dataset_type,
        data_root=data_root,
        ann_files=data_root + f's3dis_infos_Area_{test_area}.pkl',
        pipeline=test_pipeline,
        classes=class_names,
        test_mode=True,
        ignore_index=len(class_names),
        scene_idxs=data_root +
        f'seg_info/Area_{test_area}_resampled_scene_idxs.npy'))
```

可以看到，我们通过将多个相应路径构成的列表 (list) 输入 `ann_files` 和 `scene_idxs` 以实现训练测试集的划分。如果修改训练测试区域的划分，只需要简单修改 `train_area` 和 `test_area` 即可。

CHAPTER 21

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来方便进行各种实验。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。你也可以传入 `--options xxx.yyy=zzz` 参数来查看更新后的配置。

21.1 配置文件结构

在 `config/_base_` 文件夹下有 4 个基本组件类型，分别是：数据集 (dataset)，模型 (model)，训练策略 (schedule) 和运行时的默认设置 (default runtime)。通过从上述每个文件夹中选取一个组件进行组合，许多方法如 SECOND、PointPillars、PartA2 和 VoteNet 都能够很容易地构建出来。由 `_base_` 下的组件组成的配置，被我们称为 *原始配置 (primitive)*。

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

为了便于理解，我们建议贡献者继承现有方法。例如，如果在 PointPillars 的基础上做了一些修改，用户首先可以通过指定 `_base_ = ../pointpillars/hv_pointpillars_fpn_sbn-all_4x8_2x_nus-3d.py` 来继承基础的 PointPillars 结构，然后修改配置文件中的必要参数以完成继承。

如果你在构建一个与任何现有方法不共享结构的全新方法，可以在 `configs` 文件夹下创建一个新的例如 `xxx_rcnn` 文件夹。

更多细节请参考 MMCV 文档。

21.2 配置文件名称风格

我们遵循以下样式来命名配置文件，并建议贡献者遵循相同的风格。

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

`{xxx}` 是被要求填写的字段而 `[yyy]` 是可选的。

- `{model}`：模型种类，例如 `hv_pointpillars` (Hard Voxelization PointPillars)、`VoteNet` 等。
- `[model setting]`：某些模型的特殊设定。
- `{backbone}`：主干网络种类例如 `regnet-400mf`、`regnet-1.6gf` 等。
- `{neck}`：模型颈部的种类包括 `fpn`、`secfpn` 等。
- `[norm_setting]`：如无特殊声明，默认使用 `bn` (Batch Normalization)，其他类型可以有 `gn` (Group Normalization)、`sbn` (Synchronized Batch Normalization) 等。`gn-head`/`gn-neck` 表示 GN 仅应用于网络的头部或颈部，而 `gn-all` 表示 GN 用于整个模型，例如主干网络、颈部和头部。
- `[misc]`：模型中各式各样的设置/插件，例如 `strong-aug` 意味着在训练过程中使用更强的数据增广策略。
- `[batch_per_gpu x gpu]`：每个 GPU 的样本数和 GPU 数量，默认使用 `4x8`。
- `{schedule}`：训练方案，选项是 `1x`、`2x`、`20e` 等。`1x` 和 `2x` 分别代表训练 12 和 24 轮。`20e` 在级联模型中使用，表示训练 20 轮。对于 `1x`/`2x`，初始学习率在第 8/16 和第 11/22 轮衰减 10 倍；对于 `20e`，初始学习率在第 16 和第 19 轮衰减 10 倍。
- `{dataset}`：数据集，例如 `nus-3d`、`kitti-3d`、`lyft-3d`、`scannet-3d`、`sunrgbd-3d` 等。当某一数据集存在多种设定时，我们也标记下所使用的类别数量，例如 `kitti-3d-3class` 和 `kitti-3d-car` 分别意味着在 KITTI 的所有三类上和单独车这一类上进行训练。

21.3 弃用的 train_cfg/test_cfg

遵循 MMDetection 的做法，我们在配置文件中弃用 `train_cfg` 和 `test_cfg`，请在模型配置中指定它们。原始的配置结构如下：

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

迁移后的配置结构如下：

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

21.4 VoteNet 配置文件示例

```
model = dict(
    type='VoteNet',  # 检测器的类型，更多细节请参考 mmdet3d.models.detectors
    backbone=dict(
        type='PointNet2SASSG',  # 主干网络的类型，更多细节请参考 mmdet3d.models.backbones
        in_channels=4,  # 点云输入通道数
        num_points=(2048, 1024, 512, 256),  # 每个 SA 模块采样的中心点的数量
        radius=(0.2, 0.4, 0.8, 1.2),  # 每个 SA 层的半径
        num_samples=(64, 32, 16, 16),  # 每个 SA 层聚集的点的数量
        sa_channels=((64, 64, 128), (128, 128, 256), (128, 128, 256),
                     (128, 128, 256)),  # SA 模块中每个多层感知器的输出通道数
        fp_channels=((256, 256), (256, 256)),  # FP 模块中每个多层感知器的输出通道数
        norm_cfg=dict(type='BN2d'),  # 归一化层的配置
        sa_cfg=dict(  # 点集抽象 (SA) 模块的配置
            type='PointSAModule',  # SA 模块的类型
            pool_mod='max',  # SA 模块的池化方法 (最大池化或平均池化)
            use_xyz=True,  # 在特征聚合中是否使用 xyz 坐标
            normalize_xyz=True)),  # 在特征聚合中是否使用标准化的 xyz 坐标
    bbox_head=dict(
        type='VoteHead',  # 检测框头的类型，更多细节请参考 mmdet3d.models.dense_heads
        num_classes=18,  # 分类的类别数量
        bbox_coder=dict(
            type='PartialBinBasedBBoxCoder',  # 框编码层的类型，更多细节请参考 mmdet3d.core.
↪bbox.coders
            num_sizes=18,  # 尺寸聚类的数量
            num_dir_bins=1,  # 编码方向角的间隔数
            with_rot=False,  # 框是否带有旋转角度
            mean_sizes=[[0.76966727, 0.8116021, 0.92573744],
                        [1.876858, 1.8425595, 1.1931566],
                        [0.61328, 0.6148609, 0.7182701],
                        [1.3955007, 1.5121545, 0.83443564],
                        [0.97949594, 1.0675149, 0.6329687],
```

(下页继续)

(续上页)

```
                        [0.531663, 0.5955577, 1.7500148],
                        [0.9624706, 0.72462326, 1.1481868],
                        [0.83221924, 1.0490936, 1.6875663],
                        [0.21132214, 0.4206159, 0.5372846],
                        [1.4440073, 1.8970833, 0.26985747],
                        [1.0294262, 1.4040797, 0.87554324],
                        [1.3766412, 0.65521795, 1.6813129],
                        [0.6650819, 0.71111923, 1.298853],
                        [0.41999173, 0.37906948, 1.7513971],
                        [0.59359556, 0.5912492, 0.73919016],
                        [0.50867593, 0.50656086, 0.30136237],
                        [1.1511526, 1.0546296, 0.49706793],
                        [0.47535285, 0.49249494, 0.5802117]]),  # 每一类的平均尺寸，其顺序
与类名顺序相同
        vote_moudule_cfg=dict(  # 投票 (vote) 模块的配置，更多细节请参考 mmdet3d.models.
↪model_utils
            in_channels=256,  # 投票模块的输入通道数
            vote_per_seed=1,  # 对于每个种子点生成的投票数
            gt_per_seed=3,  # 每个种子点的真实标签个数
            conv_channels=(256, 256),  # 卷积通道数
            conv_cfg=dict(type='Conv1d'),  # 卷积配置
            norm_cfg=dict(type='BN1d'),  # 归一化层配置
            norm_feats=True,  # 是否标准化特征
            vote_loss=dict(  # 投票分支的损失函数配置
                type='ChamferDistance',  # 投票分支的损失函数类型
                mode='l1',  # 投票分支的损失函数模式
                reduction='none',  # 设置对损失函数输出的聚合方法
                loss_dst_weight=10.0)),  # 投票分支的目标损失权重
        vote_aggregation_cfg=dict(  # 投票聚合分支的配置
            type='PointSAModule',  # 投票聚合模块的类型
            num_point=256,  # 投票聚合分支中 SA 模块的点的数量
            radius=0.3,  # 投票聚合分支中 SA 模块的半径
            num_sample=16,  # 投票聚合分支中 SA 模块的采样点的数量
            mlp_channels=[256, 128, 128, 128],  # 投票聚合分支中 SA 模块的多层感知器的通道数
            use_xyz=True,  # 是否使用 xyz 坐标
            normalize_xyz=True),  # 是否使用标准化后的 xyz 坐标
        feat_channels=(128, 128),  # 特征卷积的通道数
        conv_cfg=dict(type='Conv1d'),  # 卷积的配置
        norm_cfg=dict(type='BN1d'),  # 归一化层的配置
        objectness_loss=dict(  # 物体性 (objectness) 损失函数的配置
            type='CrossEntropyLoss',  # 损失函数类型
            class_weight=[0.2, 0.8],  # 损失函数对每一类的权重
            reduction='sum',  # 设置损失函数输出的聚合方法
```

(下页继续)

(续上页)

```
            loss_weight=5.0),  # 损失函数权重
        center_loss=dict(  # 中心 (center) 损失函数的配置
            type='ChamferDistance',  # 损失函数类型
            mode='l2',  # 损失函数模式
            reduction='sum',  # 设置损失函数输出的聚合方法
            loss_src_weight=10.0,  # 源损失权重
            loss_dst_weight=10.0),  # 目标损失权重
        dir_class_loss=dict(  # 方向分类损失函数的配置
            type='CrossEntropyLoss',  # 损失函数类型
            reduction='sum',  # 设置损失函数输出的聚合方法
            loss_weight=1.0),  # 损失函数权重
        dir_res_loss=dict(  # 方向残差 (residual) 损失函数的配置
            type='SmoothL1Loss',  # 损失函数类型
            reduction='sum',  # 设置损失函数输出的聚合方法
            loss_weight=10.0),  # 损失函数权重
        size_class_loss=dict(  # 尺寸分类损失函数的配置
            type='CrossEntropyLoss',  # 损失函数类型
            reduction='sum',  # 设置损失函数输出的聚合方法
            loss_weight=1.0),  # 损失函数权重
        size_res_loss=dict(  # 尺寸残差损失函数的配置
            type='SmoothL1Loss',  # 损失函数类型
            reduction='sum',  # 设置损失函数输出的聚合方法
            loss_weight=3.3333333333333335),  # 损失函数权重
        semantic_loss=dict(  # 语义损失函数的配置
            type='CrossEntropyLoss',  # 损失函数类型
            reduction='sum',  # 设置损失函数输出的聚合方法
            loss_weight=1.0)),  # 损失函数权重
    train_cfg = dict(  # VoteNet 训练的超参数配置
        pos_distance_thr=0.3,  # 距离 >= 0.3 阈值的样本将被视为正样本
        neg_distance_thr=0.6,  # 距离 < 0.6 阈值的样本将被视为负样本
        sample_mod='vote'),  # 采样方法的模式
    test_cfg = dict(  # VoteNet 测试的超参数配置
        sample_mod='seed',  # 采样方法的模式
        nms_thr=0.25,  # NMS 中使用的阈值
        score_thr=0.8,  # 剔除框的阈值
        per_class_proposal=False))  # 是否使用逐类提议框 (proposal)
dataset_type = 'ScanNetDataset'  # 数据集类型
data_root = './data/scannet/'  # 数据路径
class_names = ('cabinet', 'bed', 'chair', 'sofa', 'table', 'door', 'window',
               'bookshelf', 'picture', 'counter', 'desk', 'curtain',
               'refrigerator', 'showercurtrain', 'toilet', 'sink', 'bathtub',
               'garbagebin')  # 类的名称
train_pipeline = [  # 训练流水线，更多细节请参考 mmdet3d.datasets.pipelines
```

(下页继续)

(续上页)

```
    dict(
        type='LoadPointsFromFile',  # 第一个流程，用于读取点，更多细节请参考 mmdet3d.
↪datasets.pipelines.indoor_loading
        shift_height=True,  # 是否使用变换高度
        load_dim=6,  # 读取的点的维度
        use_dim=[0, 1, 2]),  # 使用所读取点的哪些维度
    dict(
        type='LoadAnnotations3D',  # 第二个流程，用于读取标注，更多细节请参考 mmdet3d.
↪datasets.pipelines.indoor_loading
        with_bbox_3d=True,  # 是否读取 3D 框
        with_label_3d=True,  # 是否读取 3D 框对应的类别标签
        with_mask_3d=True,  # 是否读取 3D 实例分割掩码
        with_seg_3d=True),  # 是否读取 3D 语义分割掩码
    dict(
        type='PointSegClassMapping',  # 选取有效的类别，更多细节请参考 mmdet3d.datasets.
↪pipelines.point_seg_class_mapping
        valid_cat_ids=(3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 24, 28, 33, 34,
                       36, 39),  # 所有有效类别的编号
        max_cat_id=40),  # 输入语义分割掩码中可能存在的最大类别编号
    dict(type='PointSample',  # 室内点采样，更多细节请参考 mmdet3d.datasets.pipelines.
↪indoor_sample
            num_points=40000),  # 采样的点的数量
    dict(type='IndoorFlipData',  # 数据增广流程，随机翻转点和 3D 框
        flip_ratio_yz=0.5,  # 沿着 yz 平面被翻转的概率
        flip_ratio_xz=0.5),  # 沿着 xz 平面被翻转的概率
    dict(
        type='IndoorGlobalRotScale',  # 数据增广流程，旋转并放缩点和 3D 框，更多细节请参考␣
↪mmdet3d.datasets.pipelines.indoor_augment
        shift_height=True,  # 读取的点是否有高度这一属性
        rot_range=[-0.027777777777777776, 0.027777777777777776],  # 旋转角范围
        scale_range=None),  # 缩放尺寸范围
    dict(
        type='DefaultFormatBundle3D',  # 默认格式打包以收集读取的所有数据，更多细节请参考␣
↪mmdet3d.datasets.pipelines.formatting
        class_names=('cabinet', 'bed', 'chair', 'sofa', 'table', 'door',
                     'window', 'bookshelf', 'picture', 'counter', 'desk',
                     'curtain', 'refrigerator', 'showercurtrain', 'toilet',
                     'sink', 'bathtub', 'garbagebin')),
    dict(
        type='Collect3D',  # 最后一个流程，决定哪些键值对应的数据会被输入给检测器，更多细节请参考␣
↪mmdet3d.datasets.pipelines.formatting
        keys=[
            'points', 'gt_bboxes_3d', 'gt_labels_3d', 'pts_semantic_mask',
```

(下页继续)

(续上页)

```
            'pts_instance_mask'
        ])
]
test_pipeline = [  # 测试流水线，更多细节请参考 mmdet3d.datasets.pipelines
    dict(
        type='LoadPointsFromFile',  # 第一个流程，用于读取点，更多细节请参考 mmdet3d.
↪datasets.pipelines.indoor_loading
        shift_height=True,  # 是否使用变换高度
        load_dim=6,  # 读取的点的维度
        use_dim=[0, 1, 2]),  # 使用所读取点的哪些维度
    dict(type='PointSample',  # 室内点采样，更多细节请参考 mmdet3d.datasets.pipelines.
↪indoor_sample
            num_points=40000),  # 采样的点的数量
    dict(
        type='DefaultFormatBundle3D',  # 默认格式打包以收集读取的所有数据，更多细节请参考␣
↪mmdet3d.datasets.pipelines.formatting
        class_names=('cabinet', 'bed', 'chair', 'sofa', 'table', 'door',
                     'window', 'bookshelf', 'picture', 'counter', 'desk',
                     'curtain', 'refrigerator', 'showercurtrain', 'toilet',
                     'sink', 'bathtub', 'garbagebin')),
    dict(type='Collect3D',  # 最后一个流程，决定哪些键值对应的数据会被输入给检测器，更多细节请参
考 mmdet3d.datasets.pipelines.formatting
        keys=['points'])
]
eval_pipeline = [  # 模型验证或可视化所使用的流水线，更多细节请参考 mmdet3d.datasets.pipelines
    dict(
        type='LoadPointsFromFile',  # 第一个流程，用于读取点，更多细节请参考 mmdet3d.
↪datasets.pipelines.indoor_loading
        shift_height=True,  # 是否使用变换高度
        load_dim=6,  # 读取的点的维度
        use_dim=[0, 1, 2]),  # 使用所读取点的哪些维度
    dict(
        type='DefaultFormatBundle3D',  # 默认格式打包以收集读取的所有数据，更多细节请参考␣
↪mmdet3d.datasets.pipelines.formatting
        class_names=('cabinet', 'bed', 'chair', 'sofa', 'table', 'door',
                     'window', 'bookshelf', 'picture', 'counter', 'desk',
                     'curtain', 'refrigerator', 'showercurtrain', 'toilet',
                     'sink', 'bathtub', 'garbagebin')),
        with_label=False),
    dict(type='Collect3D',  # 最后一个流程，决定哪些键值对应的数据会被输入给检测器，更多细节请参
考 mmdet3d.datasets.pipelines.formatting
        keys=['points'])
]
```

(下页继续)

(续上页)

```
data = dict(
    samples_per_gpu=8,  # 单张 GPU 上的样本数
    workers_per_gpu=4,  # 每张 GPU 上用于读取数据的进程数
    train=dict(  # 训练数据集配置
        type='RepeatDataset',  # 数据集嵌套，更多细节请参考 https://github.com/open-mmlab/
↪mmdetection/blob/master/mmdet/datasets/dataset_wrappers.py
        times=5,  # 重复次数
        dataset=dict(
            type='ScanNetDataset',  # 数据集类型
            data_root='./data/scannet/',  # 数据路径
            ann_file='./data/scannet/scannet_infos_train.pkl',  # 数据标注文件的路径
            pipeline=[  # 流水线，这里传入的就是上面创建的训练流水线变量
                dict(
                    type='LoadPointsFromFile',
                    shift_height=True,
                    load_dim=6,
                    use_dim=[0, 1, 2]),
                dict(
                    type='LoadAnnotations3D',
                    with_bbox_3d=True,
                    with_label_3d=True,
                    with_mask_3d=True,
                    with_seg_3d=True),
                dict(
                    type='PointSegClassMapping',
                    valid_cat_ids=(3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 24,
                                   28, 33, 34, 36, 39),
                    max_cat_id=40),
                dict(type='PointSample', num_points=40000),
                dict(
                    type='IndoorFlipData',
                    flip_ratio_yz=0.5,
                    flip_ratio_xz=0.5),
                dict(
                    type='IndoorGlobalRotScale',
                    shift_height=True,
                    rot_range=[-0.027777777777777776, 0.027777777777777776],
                    scale_range=None),
                dict(
                    type='DefaultFormatBundle3D',
                    class_names=('cabinet', 'bed', 'chair', 'sofa', 'table',
                                 'door', 'window', 'bookshelf', 'picture',
                                 'counter', 'desk', 'curtain', 'refrigerator',
```

(下页继续)

(续上页)

```
                                'showercurtrain', 'toilet', 'sink', 'bathtub',
                                'garbagebin')),
                dict(
                    type='Collect3D',
                    keys=[
                        'points', 'gt_bboxes_3d', 'gt_labels_3d',
                        'pts_semantic_mask', 'pts_instance_mask'
                    ])
            ],
            filter_empty_gt=False,  # 是否过滤掉空的标签框
            classes=('cabinet', 'bed', 'chair', 'sofa', 'table', 'door',
                     'window', 'bookshelf', 'picture', 'counter', 'desk',
                     'curtain', 'refrigerator', 'showercurtrain', 'toilet',
                     'sink', 'bathtub', 'garbagebin'))),  # 类别名称
    val=dict(  # 验证数据集配置
        type='ScanNetDataset',  # 数据集类型
        data_root='./data/scannet/',  # 数据路径
        ann_file='./data/scannet/scannet_infos_val.pkl',  # 数据标注文件的路径
        pipeline=[  # 流水线，这里传入的就是上面创建的测试流水线变量
            dict(
                type='LoadPointsFromFile',
                shift_height=True,
                load_dim=6,
                use_dim=[0, 1, 2]),
            dict(type='PointSample', num_points=40000),
            dict(
                type='DefaultFormatBundle3D',
                class_names=('cabinet', 'bed', 'chair', 'sofa', 'table',
                             'door', 'window', 'bookshelf', 'picture',
                             'counter', 'desk', 'curtain', 'refrigerator',
                             'showercurtrain', 'toilet', 'sink', 'bathtub',
                             'garbagebin')),
            dict(type='Collect3D', keys=['points'])
        ],
        classes=('cabinet', 'bed', 'chair', 'sofa', 'table', 'door', 'window',
                 'bookshelf', 'picture', 'counter', 'desk', 'curtain',
                 'refrigerator', 'showercurtrain', 'toilet', 'sink', 'bathtub',
                 'garbagebin'),  # 类别名称
        test_mode=True),  # 是否开启测试模式
    test=dict(  # 测试数据集配置
        type='ScanNetDataset',  # 数据集类型
        data_root='./data/scannet/',  # 数据路径
        ann_file='./data/scannet/scannet_infos_val.pkl',  # 数据标注文件的路径
```

(下页继续)

(续上页)

```
        pipeline=[  # 流水线，这里传入的就是上面创建的测试流水线变量
            dict(
                type='LoadPointsFromFile',
                shift_height=True,
                load_dim=6,
                use_dim=[0, 1, 2]),
            dict(type='PointSample', num_points=40000),
            dict(
                type='DefaultFormatBundle3D',
                class_names=('cabinet', 'bed', 'chair', 'sofa', 'table',
                             'door', 'window', 'bookshelf', 'picture',
                             'counter', 'desk', 'curtain', 'refrigerator',
                             'showercurtrain', 'toilet', 'sink', 'bathtub',
                             'garbagebin')),
            dict(type='Collect3D', keys=['points'])
        ],
        classes=('cabinet', 'bed', 'chair', 'sofa', 'table', 'door', 'window',
                 'bookshelf', 'picture', 'counter', 'desk', 'curtain',
                 'refrigerator', 'showercurtrain', 'toilet', 'sink', 'bathtub',
                 'garbagebin'),  # 类别名称
        test_mode=True))  # 是否开启测试模式
evaluation = dict(pipeline=[  # 流水线，这里传入的就是上面创建的验证流水线变量
    dict(
        type='LoadPointsFromFile',
        coord_type='DEPTH',
        shift_height=False,
        load_dim=6,
        use_dim=[0, 1, 2]),
    dict(
        type='DefaultFormatBundle3D',
        class_names=('cabinet', 'bed', 'chair', 'sofa', 'table', 'door',
                     'window', 'bookshelf', 'picture', 'counter', 'desk',
                     'curtain', 'refrigerator', 'showercurtrain', 'toilet',
                     'sink', 'bathtub', 'garbagebin'),
        with_label=False),
    dict(type='Collect3D', keys=['points'])
])
lr = 0.008  # 优化器的学习率
optimizer = dict(  # 构建优化器所使用的配置，我们支持所有 PyTorch 中支持的优化器，并且拥有相同的参
数名称
    type='Adam',  # 优化器类型，更多细节请参考 https://github.com/open-mmlab/mmcv/blob/v1.
↪3.7/mmcv/runner/optimizer/default_constructor.py#L12
    lr=0.008)  # 优化器的学习率，用户可以在 PyTorch 文档中查看这些参数的详细使用方法
```

(下页继续)

(续上页)

```
optimizer_config = dict(  # 构建优化器钩子的配置，更多实现细节可参考 https://github.com/open-
↪mmlab/mmcv/blob/v1.3.7/mmcv/runner/hooks/optimizer.py#L22
    grad_clip=dict(  # 梯度裁剪的配置
    max_norm=10,  # 梯度的最大模长
    norm_type=2))  # 所使用的 p-范数的类型，可以设置成 'inf' 则指代无穷范数
lr_config = dict(  # 学习率策略配置，用于注册学习率更新的钩子
    policy='step',  # 学习率调整的策略，支持 CosineAnnealing、Cyclic 等，更多支持的种类请参考␣
↪https://github.com/open-mmlab/mmcv/blob/v1.3.7/mmcv/runner/hooks/lr_updater.py#L9
    warmup=None,  # Warmup 策略，同时也支持 `exp` 和 `constant`
    step=[24, 32])  # 学习率衰减的步数
checkpoint_config = dict(  # 设置保存模型权重钩子的配置，具体实现请参考 https://github.com/
↪open-mmlab/mmcv/blob/master/mmcv/runner/hooks/checkpoint.py
    interval=1)  # 保存模型权重的间隔是 1 轮
log_config = dict(  # 用于注册输出记录信息钩子的配置
    interval=50,  # 输出记录信息的间隔
    hooks=[dict(type='TextLoggerHook'),
           dict(type='TensorboardLoggerHook')])  # 用于记录训练过程的信息记录机制
runner = dict(type='EpochBasedRunner', max_epochs=36) # 程序运行器，将会运行 `workflow`␣
↪`max_epochs` 次
dist_params = dict(backend='nccl')  # 设置分布式训练的配置，通讯端口值也可被设置
log_level = 'INFO'  # 输出记录信息的等级
find_unused_parameters = True  # 是否查找模型中未使用的参数
work_dir = None  # 当前实验存储模型权重和输出信息的路径
load_from = None # 从指定路径读取一个预训练的模型权重，这将不会继续 (resume) 训练
resume_from = None  # 从一个指定路径读入模型权重并继续训练，这意味着训练轮数、优化器状态等都将被读取
workflow = [('train', 1)]  # 要运行的工作流。[('train', 1)] 意味着只有一个名为 'train' 的工作
流，它只会被执行一次。这一工作流依据 `max_epochs` 的值将会训练模型 36 轮。
gpu_ids = range(0, 1)  # 所使用的 GPU 编号
```

21.5 常问问题 (FAQ)

21.5.1 忽略基础配置文件里的部分内容

有时，您也许会需要通过设置 _delete_=True 来忽略基础配置文件里的一些域内容。请参照 mmcv 来获得一些简单的指导。

例如在 MMDetection3D 中，为了改变如下所示 PointPillars FPN 模块的某些配置：

```
model = dict(
    type='MVXFasterRCNN',
    pts_voxel_layer=dict(...),
    pts_voxel_encoder=dict(...),
```

(下页继续)

(续上页)

```
    pts_middle_encoder=dict(...),
    pts_backbone=dict(...),
    pts_neck=dict(
        type='FPN',
        norm_cfg=dict(type='naiveSyncBN2d', eps=1e-3, momentum=0.01),
        act_cfg=dict(type='ReLU'),
        in_channels=[64, 128, 256],
        out_channels=256,
        start_level=0,
        num_outs=3),
    pts_bbox_head=dict(...))
```

FPN 和 SECONDFPN 使用不同的关键词来构建。

```
_base_ = '../_base_/models/hv_pointpillars_fpn_nus.py'
model = dict(
    pts_neck=dict(
        _delete_=True,
        type='SECONDFPN',
        norm_cfg=dict(type='naiveSyncBN2d', eps=1e-3, momentum=0.01),
        in_channels=[64, 128, 256],
        upsample_strides=[1, 2, 4],
        out_channels=[128, 128, 128]),
    pts_bbox_head=dict(...))
```

delete=True 的标识将会使用新的键值覆盖掉 pts_neck 中的所有旧键值。

21.5.2 使用配置文件里的中间变量

配置文件里会使用一些中间变量，例如数据集中的 train_pipeline/test_pipeline。值得注意的是，当修改子配置文件中的中间变量后，用户还需再次将其传入相应字段。例如，我们想在训练和测试中，对 PointPillars 使用多尺度策略 (multi scale strategy)，那么 train_pipeline/test_pipeline 就是我们想要修改的中间变量。

```
_base_ = './nus-3d.py'
train_pipeline = [
    dict(
        type='LoadPointsFromFile',
        load_dim=5,
        use_dim=5,
        file_client_args=file_client_args),
    dict(
        type='LoadPointsFromMultiSweeps',
```

(下页继续)

(续上页)

```
        sweeps_num=10,
        file_client_args=file_client_args),
    dict(type='LoadAnnotations3D', with_bbox_3d=True, with_label_3d=True),
    dict(
        type='GlobalRotScaleTrans',
        rot_range=[-0.3925, 0.3925],
        scale_ratio_range=[0.95, 1.05],
        translation_std=[0, 0, 0]),
    dict(type='RandomFlip3D', flip_ratio_bev_horizontal=0.5),
    dict(type='PointsRangeFilter', point_cloud_range=point_cloud_range),
    dict(type='ObjectRangeFilter', point_cloud_range=point_cloud_range),
    dict(type='ObjectNameFilter', classes=class_names),
    dict(type='PointShuffle'),
    dict(type='DefaultFormatBundle3D', class_names=class_names),
    dict(type='Collect3D', keys=['points', 'gt_bboxes_3d', 'gt_labels_3d'])
]
test_pipeline = [
    dict(
        type='LoadPointsFromFile',
        load_dim=5,
        use_dim=5,
        file_client_args=file_client_args),
    dict(
        type='LoadPointsFromMultiSweeps',
        sweeps_num=10,
        file_client_args=file_client_args),
    dict(
        type='MultiScaleFlipAug3D',
        img_scale=(1333, 800),
        pts_scale_ratio=[0.95, 1.0, 1.05],
        flip=False,
        transforms=[
            dict(
                type='GlobalRotScaleTrans',
                rot_range=[0, 0],
                scale_ratio_range=[1., 1.],
                translation_std=[0, 0, 0]),
            dict(type='RandomFlip3D'),
            dict(
                type='PointsRangeFilter', point_cloud_range=point_cloud_range),
            dict(
                type='DefaultFormatBundle3D',
                class_names=class_names,
```

(下页继续)

(续上页)

```
                with_label=False),
            dict(type='Collect3D', keys=['points'])
        ])
]
data = dict(
    train=dict(pipeline=train_pipeline),
    val=dict(pipeline=test_pipeline),
    test=dict(pipeline=test_pipeline))
```

这里，我们首先定义了新的 train_pipeline/test_pipeline，然后将其传入 data。

CHAPTER 22

教程 2: 自定义数据集

22.1 支持新的数据格式

为了支持新的数据格式，可以通过将新数据转换为现有的数据形式，或者直接将新数据转换为能够被模型直接调用的中间格式。此外，可以通过数据离线转换的方式（在调用脚本进行训练之前完成）或者通过数据在线转换的格式（调用新的数据集并在训练过程中进行数据转换）。在 MMDetection3D 中，对于那些不便于在线读取的数据，我们建议通过离线转换的方法将其转换为 KTIIT 数据集的格式，因此只需要在转换后修改配置文件中的数据标注文件的路径和标注数据所包含类别；对于那些与现有数据格式相似的新数据集，如 Lyft 数据集和 nuScenes 数据集，我们建议直接调用数据转换器和现有的数据集类别信息，在这个过程中，可以考虑通过继承的方式来减少实施数据转换的负担。

22.1.1 将新数据的格式转换为现有数据的格式

对于那些不便于在线读取的数据，最简单的方法是将新数据集的格式转换为现有数据集的格式。

通常来说，我们需要一个数据转换器来重新组织原始数据的格式，并将对应的标注格式转换为 KITTI 数据集的风格；当现有数据集与新数据集存在差异时，可以通过定义一个从现有数据集类继承而来的新数据集类来处理具体的差异；最后，用户需要进一步修改配置文件来调用新的数据集。可以参考如何通过将 Waymo 数据集转换为 KITTI 数据集的风格并进一步训练模型的例子。

22.1.2 将新数据集的格式转换为一种当前可支持的中间格式

如果不想采用将标注格式转为为现有格式的方式，也可以通过以下的方式来完成新数据集的转换。实际上，我们将所支持的所有数据集都转换成 pickle 文件的格式，这些文件整理了所有应用于模型训练和推理的有用的信息。

数据集的标注信息是通过一个字典列表来描述的，每个字典包含对应数据帧的标注信息。下面展示了一个基础例子（应用在 KITTI 数据集上），每一帧包含了几项关键字，如 `image`、`point_cloud`、`calib` 和 `annos` 等。只要能够根据这些信息来直接读取到数据，其原始数据的组织方式就可以不同于现有的数据组织方式。通过这种设计，我们提供一种可替代的方案来自定义数据集。

```
[
    {'image': {'image_idx': 0, 'image_path': 'training/image_2/000000.png', 'image_
↪shape': array([ 370, 1224], dtype=int32)},
     'point_cloud': {'num_features': 4, 'velodyne_path': 'training/velodyne/000000.bin
↪'},
     'calib': {'P0': array([[707.0493,   0.    , 604.0814,   0.    ],
       [  0.    , 707.0493, 180.5066,   0.    ],
       [  0.    ,   0.    ,   1.    ,   0.    ],
       [  0.    ,   0.    ,   0.    ,   1.    ]]),
       'P1': array([[ 707.0493,    0.    ,  604.0814, -379.7842],
       [   0.    ,  707.0493,  180.5066,    0.    ],
       [   0.    ,    0.    ,    1.    ,    0.    ],
       [   0.    ,    0.    ,    0.    ,    1.    ]]),
       'P2': array([[ 7.070493e+02,  0.000000e+00,  6.040814e+02,  4.575831e+01],
       [ 0.000000e+00,  7.070493e+02,  1.805066e+02, -3.454157e-01],
       [ 0.000000e+00,  0.000000e+00,  1.000000e+00,  4.981016e-03],
       [ 0.000000e+00,  0.000000e+00,  0.000000e+00,  1.000000e+00]]),
       'P3': array([[ 7.070493e+02,  0.000000e+00,  6.040814e+02, -3.341081e+02],
       [ 0.000000e+00,  7.070493e+02,  1.805066e+02,  2.330660e+00],
       [ 0.000000e+00,  0.000000e+00,  1.000000e+00,  3.201153e-03],
       [ 0.000000e+00,  0.000000e+00,  0.000000e+00,  1.000000e+00]]),
       'R0_rect': array([[ 0.9999128 ,  0.01009263, -0.00851193,  0.        ],
       [-0.01012729,  0.9999406 , -0.00403767,  0.        ],
       [ 0.00847068,  0.00412352,  0.9999556 ,  0.        ],
       [ 0.        ,  0.        ,  0.        ,  1.        ]]),
       'Tr_velo_to_cam': array([[ 0.00692796, -0.9999722 , -0.00275783, -0.02457729],
       [-0.00116298,  0.00274984, -0.9999955 , -0.06127237],
       [ 0.9999753 ,  0.00693114, -0.0011439 , -0.3321029 ],
       [ 0.        ,  0.        ,  0.        ,  1.        ]]),
       'Tr_imu_to_velo': array([[ 9.999976e-01,  7.553071e-04, -2.035826e-03, -8.
↪086759e-01],
       [-7.854027e-04,  9.998898e-01, -1.482298e-02,  3.195559e-01],
       [ 2.024406e-03,  1.482454e-02,  9.998881e-01, -7.997231e-01],
```

(下页继续)

(续上页)

```
       [ 0.000000e+00,  0.000000e+00,  0.000000e+00,  1.000000e+00]])},
    'annos': {'name': array(['Pedestrian'], dtype='<U10'), 'truncated': array([0.]),
↪'occluded': array([0]), 'alpha': array([-0.2]), 'bbox': array([[712.4 , 143.  , 810.
↪73, 307.92]]), 'dimensions': array([[1.2 , 1.89, 0.48]]), 'location': array([[1.84,␣
↪1.47, 8.41]]), 'rotation_y': array([0.01]), 'score': array([0.]), 'index':␣
↪array([0], dtype=int32), 'group_ids': array([0], dtype=int32), 'difficulty':␣
↪array([0], dtype=int32), 'num_points_in_gt': array([377], dtype=int32)}}
    ...
]
```

在此之上，用户可以通过继承 `Custom3DDataset` 来实现新的数据集类，并重载相关的方法，如 KITTI 数据集和 ScanNet 数据集所示。

22.1.3 自定义数据集的例子

我们在这里提供了一个自定义数据集的例子：

假设已经将标注信息重新组织成一个 pickle 文件格式的字典列表，比如 ScanNet。标注框的标注信息会被存储在 `annotation.pkl` 文件中，其格式如下所示：

```
{'point_cloud': {'num_features': 6, 'lidar_idx': 'scene0000_00'}, 'pts_path': 'points/
↪scene0000_00.bin',
 'pts_instance_mask_path': 'instance_mask/scene0000_00.bin', 'pts_semantic_mask_path
↪': 'semantic_mask/scene0000_00.bin',
 'annos': {'gt_num': 27, 'name': array(['window', 'window', 'table', 'counter',
↪'curtain', 'curtain',
       'desk', 'cabinet', 'sink', 'garbagebin', 'garbagebin',
       'garbagebin', 'sofa', 'refrigerator', 'table', 'table', 'toilet',
       'bed', 'cabinet', 'cabinet', 'cabinet', 'cabinet', 'cabinet',
       'cabinet', 'door', 'door', 'door'], dtype='<U12'),
       'location': array([[ 1.48129511,  3.52074146,  1.85652947],
       [ 2.90395617, -3.48033905,  1.52682471]]),
       'dimensions': array([[1.74445975, 0.23195696, 0.57235193],
       [0.66077662, 0.17072392, 0.67153597]]),
       'gt_boxes_upright_depth': array([
       [ 1.48129511,  3.52074146,  1.85652947,  1.74445975,  0.23195696,
         0.57235193],
       [ 2.90395617, -3.48033905,  1.52682471,  0.66077662,  0.17072392,
         0.67153597]]),
       'index': array([ 0,  1 ], dtype=int32),
       'class': array([ 6,  6 ])}}
```

我们在 `mmdet3d/datasets/my_dataset.py` 中创建了一个新的数据集类来进行数据的加载，如下所示：

```
import numpy as np
from os import path as osp

from mmdet3d.core import show_result
from mmdet3d.core.bbox import DepthInstance3DBoxes
from mmdet.datasets import DATASETS
from .custom_3d import Custom3DDataset


@DATASETS.register_module()
class MyDataset(Custom3DDataset):
    CLASSES = ('cabinet', 'bed', 'chair', 'sofa', 'table', 'door', 'window',
               'bookshelf', 'picture', 'counter', 'desk', 'curtain',
               'refrigerator', 'showercurtrain', 'toilet', 'sink', 'bathtub',
               'garbagebin')

    def __init__(self,
                 data_root,
                 ann_file,
                 pipeline=None,
                 classes=None,
                 modality=None,
                 box_type_3d='Depth',
                 filter_empty_gt=True,
                 test_mode=False):
        super().__init__(
            data_root=data_root,
            ann_file=ann_file,
            pipeline=pipeline,
            classes=classes,
            modality=modality,
            box_type_3d=box_type_3d,
            filter_empty_gt=filter_empty_gt,
            test_mode=test_mode)

    def get_ann_info(self, index):
        # 通过下标来获取标注信息，evalhook 也能够通过此接口来获取标注信息
        info = self.data_infos[index]
        if info['annos']['gt_num'] != 0:
            gt_bboxes_3d = info['annos']['gt_boxes_upright_depth'].astype(
                np.float32)  # k, 6
            gt_labels_3d = info['annos']['class'].astype(np.int64)
        else:
            gt_bboxes_3d = np.zeros((0, 6), dtype=np.float32)
```

(下页继续)

(续上页)

```
            gt_labels_3d = np.zeros((0, ), dtype=np.int64)

        # 转换为目标标注框的结构
        gt_bboxes_3d = DepthInstance3DBoxes(
            gt_bboxes_3d,
            box_dim=gt_bboxes_3d.shape[-1],
            with_yaw=False,
            origin=(0.5, 0.5, 0.5)).convert_to(self.box_mode_3d)

        pts_instance_mask_path = osp.join(self.data_root,
                                          info['pts_instance_mask_path'])
        pts_semantic_mask_path = osp.join(self.data_root,
                                          info['pts_semantic_mask_path'])

        anns_results = dict(
            gt_bboxes_3d=gt_bboxes_3d,
            gt_labels_3d=gt_labels_3d,
            pts_instance_mask_path=pts_instance_mask_path,
            pts_semantic_mask_path=pts_semantic_mask_path)
        return anns_results
```

接着，可以对配置文件进行修改来调用 `MyDataset` 数据集类，如下所示：

```
dataset_A_train = dict(
    type='MyDataset',
    ann_file = 'annotation.pkl',
    pipeline=train_pipeline
)
```

22.1.4 使用数据集包装器来自定义数据集

与 MMDetection 类似，MMDetection3D 也提供了许多数据集包装器来统合数据集或者修改数据集的分布，并应用到模型的训练中。目前 MMDetection3D 支持 3 种数据集包装器

- `RepeatDataset`：简单地重复整个数据集
- `ClassBalancedDataset`：以类别平衡的方式重复数据集
- `ConcatDataset`：拼接多个数据集

22.1.5 重复数据集

我们使用 RepeatDataset 包装器来进行数据集重复的设置，例如，假定当前需要重复的数据集为 Dataset_A，则配置文件应设置成如下所示：

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

22.1.6 类别平衡数据集

我们使用 ClassBalancedDataset 包装器能够基于类别出现的频率进行数据集重复的设置，进行重复的数据集需要实例化函数 self.get_cat_ids(idx)，以支持 ClassBalancedDataset 包装器的正常调用。例如，假定需要以 oversample_thr=1e-3 的设置来定义 Dataset_A 的重复，则对应的配置文件如下所示：

```
dataset_A_train = dict(
        type='ClassBalancedDataset',
        oversample_thr=1e-3,
        dataset=dict(  # 这是 Dataset_A 的原始配置文件
            type='Dataset_A',
            ...
            pipeline=train_pipeline
        )
    )
```

请参考 源码 获取更多细节。

22.1.7 拼接数据集

我们提供 3 种方式来实现数据集的拼接。

1. 如果待拼接的数据集的类别相同，标注文件的不同，此时可通过下面的方式来实现数据集的拼接：

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

(下页继续)

(续上页)

```
    pipeline=train_pipeline
)
```

如果拼接数据集用于测试或者评估，那么这种拼接方式能够对每个数据集进行分开地测试或者评估，若希望对拼接数据集进行整体的测试或者评估，此时需要设置 `separate_eval=False`，如下所示：

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

2. 如果待拼接的数据集完全不相同，此时可通过拼接不同数据集的配置的方式实现数据集的拼接，如下所示：

```
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
    )
```

如果拼接数据集用于测试或者评估，那么这种拼接方式能够对每个数据集进行分开地测试或者评估。

3. 可以通过显示地定义 `ConcatDataset` 来实现数据集的拼接，如下所示：

```
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))
```

其中，separate_eval=False 表示将所有的数据集作为一个整体进行评估。

注意:

1. 当使用选项 separate_eval=False 时，拼接的数据集需要在评估的过程中调用 self.data_infos，由于 COCO 数据集在评估过程中并未完全依赖于 self.data_infos 来获取数据信息，因此 COCO 数据集无法使用 separate_eval=False 选项。此外，我们暂未对将不同类型的数据集进行结合并作为整体进行评估的过程进行测试，因此我们暂时不建议使用上述方法对不同类型的数据集进行整体的评估。
2. 我们暂时不支持对 ClassBalancedDataset 接 RepeatDataset 进行评估，因此也不支持由这两种类型的数据集进行拼接的数据集的评估。

复杂的例子：将 Dataset_A 和 Dataset_B 分别重复 N 次和 M 次，然后将重复数据集进行拼接，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,
```

(下页继续)

(续上页)

```
        dataset_B_train
    ],
    val = dataset_A_val,
    test = dataset_A_test
)
```

22.2 修改数据集的类别

我们可以对现有的数据集的类别名称进行修改，从而实现全部标注的子集标注的训练。例如，如果想要对现有数据集中的三个类别进行训练，可以对现有数据集的类别进行如下的修改，此时数据集将会自动过滤其他类别对应的真实标注框：

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

MMDetection V2.0 也支持从文件中读取数据集的类别，更加符合真实的应用场景。例如，假定 `classes.txt` 包含如下所示的类别名称：

```
person
bicycle
car
```

用户能够将类别文件的路径名写入到配置文件中的类别信息中，此时数据集将会自动地加载该类别文件并将其转换成列表：

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

注意 (与 MMDetection 相关)：

- 在 MMDetection v2.5.0 之前，一旦设置了上述的 classes，数据集将会自动的过滤没有真实标注框的图像，然而却无法通过调整配置文件的方式来取消该行为，这会引起一定的疑惑：当没有设置 classes 的情况下，只有当选项中同时出现 `filter_empty_gt=True` 和 `test_mode=False` 时才会对数据集中没有真实标注框的图像进行过滤。在 MMDetection v2.5.0 之后，我们对图像过滤过程和类别修改过程进行分离，例如：不管配置文件中是否 classes 进行设置，数据集只会在设置 `filter_empty_gt=True` 和

`test_mode=False` 时对没有真实标注框的图像进行过滤。因此，设置 classes 仅会影响用于训练的类别标注信息，用户可以自行决定是否需要对没有真实标注框的图像进行过滤。

- 因为数据集的中间格式仅包含标注框的标签信息，并不包含类别名，因此在使用 `CustomDataset` 时，用户只能够通过离线的方式来过滤没有真实标注框的图像，而无法通过配置文件来实现过滤。
- 设置数据集类别和数据集过滤的特征将在之后进行重构，使得对应的特征更加便于使用。

CHAPTER 23

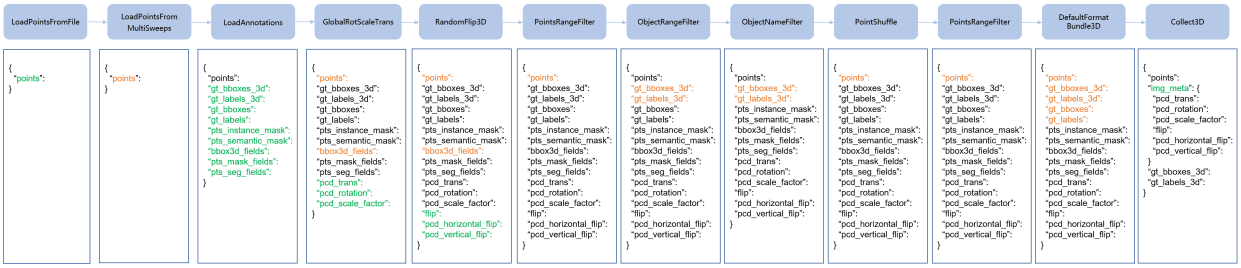
教程 3: 自定义数据预处理流程

23.1 数据预处理流程的设计

遵循一般惯例，我们使用 `Dataset` 和 `DataLoader` 来调用多个进程进行数据的加载。`Dataset` 将会返回与模型前向传播的参数所对应的数据项构成的字典。因为目标检测中的数据的尺寸可能无法保持一致（如点云中点的数量、真实标注框的尺寸等），我们在 MMCV 中引入一个 `DataContainer` 类型，来帮助收集和分发不同尺寸的数据。请参考此处获取更多细节。

数据预处理流程和数据集之间是互相分离的两个部分，通常数据集定义了如何处理标注信息，而数据预处理流程定义了准备数据项字典的所有步骤。数据集预处理流程包含一系列的操作，每个操作将一个字典作为输入，并输出应用于下一个转换的一个新的字典。

我们将在下图中展示一个最经典的数据集预处理流程，其中蓝色框表示预处理流程中的各项操作。随着预处理的进行，每一个操作都会添加新的键值（图中标记为绿色）到输出字典中，或者更新当前存在的键值（图中标记为橙色）。



预处理流程中的各项操作主要分为数据加载、预处理、格式化、测试时的数据增强。

接下来将展示一个用于 PointPillars 模型的数据集预处理流程的例子。

```
train_pipeline = [
    dict(
        type='LoadPointsFromFile',
        load_dim=5,
        use_dim=5,
        file_client_args=file_client_args),
    dict(
        type='LoadPointsFromMultiSweeps',
        sweeps_num=10,
        file_client_args=file_client_args),
    dict(type='LoadAnnotations3D', with_bbox_3d=True, with_label_3d=True),
    dict(
        type='GlobalRotScaleTrans',
        rot_range=[-0.3925, 0.3925],
        scale_ratio_range=[0.95, 1.05],
        translation_std=[0, 0, 0]),
    dict(type='RandomFlip3D', flip_ratio_bev_horizontal=0.5),
    dict(type='PointsRangeFilter', point_cloud_range=point_cloud_range),
    dict(type='ObjectRangeFilter', point_cloud_range=point_cloud_range),
    dict(type='ObjectNameFilter', classes=class_names),
    dict(type='PointShuffle'),
    dict(type='DefaultFormatBundle3D', class_names=class_names),
    dict(type='Collect3D', keys=['points', 'gt_bboxes_3d', 'gt_labels_3d'])
]
test_pipeline = [
    dict(
        type='LoadPointsFromFile',
        load_dim=5,
        use_dim=5,
        file_client_args=file_client_args),
    dict(
        type='LoadPointsFromMultiSweeps',
        sweeps_num=10,
        file_client_args=file_client_args),
    dict(
        type='MultiScaleFlipAug',
        img_scale=(1333, 800),
        pts_scale_ratio=1.0,
        flip=False,
        pcd_horizontal_flip=False,
        pcd_vertical_flip=False,
        transforms=[
            dict(
                type='GlobalRotScaleTrans',
```

(下页继续)

(续上页)

```
                rot_range=[0, 0],
                scale_ratio_range=[1., 1.],
                translation_std=[0, 0, 0]),
            dict(type='RandomFlip3D'),
            dict(
                type='PointsRangeFilter', point_cloud_range=point_cloud_range),
            dict(
                type='DefaultFormatBundle3D',
                class_names=class_names,
                with_label=False),
            dict(type='Collect3D', keys=['points'])
        ])
]
```

对于每项操作，我们将列出相关的被添加/更新/移除的字典项。

23.1.1 数据加载

`LoadPointsFromFile`

- 添加：points

`LoadPointsFromMultiSweeps`

- 更新：points

`LoadAnnotations3D`

- 添加：gt_bboxes_3d, gt_labels_3d, gt_bboxes, gt_labels, pts_instance_mask, pts_semantic_mask, bbox3d_fields, pts_mask_fields, pts_seg_fields

23.1.2 预处理

`GlobalRotScaleTrans`

- 添加：pcd_trans, pcd_rotation, pcd_scale_factor
- 更新：points, *bbox3d_fields

`RandomFlip3D`

- 添加：flip, pcd_horizontal_flip, pcd_vertical_flip
- 更新：points, *bbox3d_fields

`PointsRangeFilter`

- 更新：points

`ObjectRangeFilter`

- 更新：gt_bboxes_3d, gt_labels_3d

`ObjectNameFilter`

- 更新：gt_bboxes_3d, gt_labels_3d

`PointShuffle`

- 更新：points

`PointsRangeFilter`

- 更新：points

23.1.3 格式化

`DefaultFormatBundle3D`

- 更新：points, gt_bboxes_3d, gt_labels_3d, gt_bboxes, gt_labels

`Collect3D`

- 添加：img_meta （由 `meta_keys` 指定的键值构成的 img_meta）
- 移除：所有除 `keys` 指定的键值以外的其他键值

23.1.4 测试时的数据增强

`MultiScaleFlipAug`

- 更新: scale, pcd_scale_factor, flip, flip_direction, pcd_horizontal_flip, pcd_vertical_flip（与这些指定的参数对应的增强后的数据列表）

23.2 扩展并使用自定义数据集预处理方法

1. 在任意文件中写入新的数据集预处理方法，如 `my_pipeline.py`，该预处理方法的输入和输出均为字典

```
from mmdet.datasets import PIPELINES

@PIPELINES.register_module()
class MyTransform:

    def __call__(self, results):
        results['dummy'] = True
        return results
```

2. 导入新的预处理方法类

```
from .my_pipeline import MyTransform
```

3. 在配置文件中使用该数据集预处理方法

```
train_pipeline = [
    dict(
        type='LoadPointsFromFile',
        load_dim=5,
        use_dim=5,
        file_client_args=file_client_args),
    dict(
        type='LoadPointsFromMultiSweeps',
        sweeps_num=10,
        file_client_args=file_client_args),
    dict(type='LoadAnnotations3D', with_bbox_3d=True, with_label_3d=True),
    dict(
        type='GlobalRotScaleTrans',
        rot_range=[-0.3925, 0.3925],
        scale_ratio_range=[0.95, 1.05],
        translation_std=[0, 0, 0]),
    dict(type='RandomFlip3D', flip_ratio_bev_horizontal=0.5),
    dict(type='PointsRangeFilter', point_cloud_range=point_cloud_range),
    dict(type='ObjectRangeFilter', point_cloud_range=point_cloud_range),
    dict(type='ObjectNameFilter', classes=class_names),
    dict(type='MyTransform'),
    dict(type='PointShuffle'),
    dict(type='DefaultFormatBundle3D', class_names=class_names),
    dict(type='Collect3D', keys=['points', 'gt_bboxes_3d', 'gt_labels_3d'])
]
```

CHAPTER 24

教程 4: 自定义模型

我们通常把模型的各个组成成分分成 6 种类型：

- 编码器（encoder）：包括 voxel layer、voxel encoder 和 middle encoder 等进入 backbone 前所使用的基于 voxel 的方法，如 HardVFE 和 PointPillarsScatter。
- 骨干网络（backbone）：通常采用 FCN 网络来提取特征图，如 ResNet 和 SECOND。
- 颈部网络（neck）：位于 backbones 和 heads 之间的组成模块，如 FPN 和 SECONDFPN。
- 检测头（head）：用于特定任务的组成模块，如检测框的预测和掩码的预测。
- RoI 提取器（RoI extractor）：用于从特征图中提取 RoI 特征的组成模块，如 H3DRoIHead 和 PartAggregationROIHead。
- 损失函数（loss）：heads 中用于计算损失函数的组成模块，如 FocalLoss、L1Loss 和 GHMLoss。

24.1 开发新的组成模块

24.1.1 添加新建 encoder

接下来我们以 HardVFE 为例展示如何开发新的组成模块。

1. 定义一个新的 voxel encoder（如 HardVFE：即 DV-SECOND 中所提出的 Voxel 特征提取器）

创建一个新文件 mmdet3d/models/voxel_encoders/voxel_encoder.py：

```
import torch.nn as nn

from ..builder import VOXEL_ENCODERS


@VOXEL_ENCODERS.register_module()
class HardVFE(nn.Module):

    def __init__(self, arg1, arg2):
        pass

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

2. 导入新建模块

用户可以通过添加下面这行代码到 mmdet3d/models/voxel_encoders/__init__.py 中

```
from .voxel_encoder import HardVFE
```

或者添加以下的代码到配置文件中，从而能够在避免修改源码的情况下导入新建模块。

```
custom_imports = dict(
    imports=['mmdet3d.models.voxel_encoders.HardVFE'],
    allow_failed_imports=False)
```

3. 在配置文件中使用 voxel encoder

```
model = dict(
    ...
    voxel_encoder=dict(
        type='HardVFE',
        arg1=xxx,
        arg2=xxx),
    ...
```

24.1.2 添加新建 backbone

接下来我们以 SECOND（Sparsely Embedded Convolutional Detection）为例展示如何开发新的组成模块。

1. 定义一个新的 backbone（如 SECOND）

创建一个新文件 `mmdet3d/models/backbones/second.py`：

```
import torch.nn as nn

from ..builder import BACKBONES


@BACKBONES.register_module()
class SECOND(BaseModule):

    def __init__(self, arg1, arg2):
        pass

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

2. 导入新建模块

用户可以通过添加下面这行代码到 `mmdet3d/models/backbones/__init__.py` 中

```
from .second import SECOND
```

或者添加以下的代码到配置文件中，从而能够在避免修改源码的情况下导入新建模块。

```
custom_imports = dict(
    imports=['mmdet3d.models.backbones.second'],
    allow_failed_imports=False)
```

3. 在配置文件中使用 backbone

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

24.1.3 添加新建 necks

1. 定义一个新的 neck（如 SECONDFPN）

创建一个新文件 mmdet3d/models/necks/second_fpn.py：

```
from ..builder import NECKS

@NECKS.register
class SECONDFPN(BaseModule):

    def __init__(self,
                 in_channels=[128, 128, 256],
                 out_channels=[256, 256, 256],
                 upsample_strides=[1, 2, 4],
                 norm_cfg=dict(type='BN', eps=1e-3, momentum=0.01),
                 upsample_cfg=dict(type='deconv', bias=False),
                 conv_cfg=dict(type='Conv2d', bias=False),
                 use_conv_for_no_stride=False,
                 init_cfg=None):
        pass

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

2. 导入新建模块

用户可以通过添加下面这行代码到 mmdet3D/models/necks/__init__.py 中

```
from .second_fpn import SECONDFPN
```

或者添加以下的代码到配置文件中，从而能够在避免修改源码的情况下导入新建模块。

```
custom_imports = dict(
    imports=['mmdet3d.models.necks.second_fpn'],
    allow_failed_imports=False)
```

3. 在配置文件中使用 neck

```
model = dict(
    ...
    neck=dict(
        type='SECONDFPN',
        in_channels=[64, 128, 256],
        upsample_strides=[1, 2, 4],
        out_channels=[128, 128, 128]),
    ...
```

24.1.4 添加新建 heads

接下来我们以 PartA2 Head 为例展示如何开发新的组成模块。

注意: 此处展示的 PartA2 RoI Head 将应用于双阶段检测器中，对于单阶段检测器，请参考 `mmdet3d/models/dense_heads/` 中所展示的例子。由于这些 heads 简单高效，因此这些 heads 普遍应用在自动驾驶场景下的 3D 检测任务中。

首先，在 `mmdet3d/models/roi_heads/bbox_heads/parta2_bbox_head.py` 中创建一个新的 bbox head。PartA2 RoI Head 实现一个新的 bbox head，并用于目标检测的任务中。为了实现一个新的 bbox head，通常需要在其中实现三个功能，如下所示，有时该模块还需要实现其他相关的功能，如 `loss` 和 `get_targets`。

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

@HEADS.register_module()
class PartA2BboxHead(BaseModule):
    """PartA2 RoI head."""

    def __init__(self,
                 num_classes,
                 seg_in_channels,
                 part_in_channels,
                 seg_conv_channels=None,
                 part_conv_channels=None,
                 merge_conv_channels=None,
                 down_conv_channels=None,
                 shared_fc_channels=None,
                 cls_channels=None,
                 reg_channels=None,
                 dropout_ratio=0.1,
                 roi_feat_size=14,
                 with_corner_loss=True,
```

(下页继续)

(续上页)

```
                bbox_coder=dict(type='DeltaXYZWLHRBBoxCoder'),
                conv_cfg=dict(type='Conv1d'),
                norm_cfg=dict(type='BN1d', eps=1e-3, momentum=0.01),
                loss_bbox=dict(
                    type='SmoothL1Loss', beta=1.0 / 9.0, loss_weight=2.0),
                loss_cls=dict(
                    type='CrossEntropyLoss',
                    use_sigmoid=True,
                    reduction='none',
                    loss_weight=1.0),
                init_cfg=None):
        super(PartA2BboxHead, self).__init__(init_cfg=init_cfg)

    def forward(self, seg_feats, part_feats):
```

其次，如果有必要的话，用户还需要实现一个新的 RoI Head，此处我们从 `Base3DRoIHead` 中继承得到一个新类 `PartAggregationROIHead`，此时我们就能发现 `Base3DRoIHead` 已经实现了下面的功能：

```
from abc import ABCMeta, abstractmethod
from torch import nn as nn


@HEADS.register_module()
class Base3DRoIHead(BaseModule, metaclass=ABCMeta):
    """Base class for 3d RoIHeads."""

    def __init__(self,
                 bbox_head=None,
                 mask_roi_extractor=None,
                 mask_head=None,
                 train_cfg=None,
                 test_cfg=None,
                 init_cfg=None):

    @property
    def with_bbox(self):

    @property
    def with_mask(self):

    @abstractmethod
    def init_weights(self, pretrained):
```

(下页继续)

(续上页)

```
    @abstractmethod
    def init_bbox_head(self):

    @abstractmethod
    def init_mask_head(self):

    @abstractmethod
    def init_assigner_sampler(self):

    @abstractmethod
    def forward_train(self,
                      x,
                      img_metas,
                      proposal_list,
                      gt_bboxes,
                      gt_labels,
                      gt_bboxes_ignore=None,
                      **kwargs):

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

    def aug_test(self, x, proposal_list, img_metas, rescale=False, **kwargs):
        """Test with augmentations.
        If rescale is False, then returned bboxes and masks will fit the scale
        of imgs[0].
        """
        pass
```

接着将会对 bbox_forward 的逻辑进行修改，同时，bbox_forward 还会继承来自 `Base3DRoIHead` 的其他逻辑，在 `mmdet3d/models/roi_heads/part_aggregation_roi_head.py` 中，我们实现了新的 RoI Head，如下所示：

```
from torch.nn import functional as F
```

(下页继续)

(续上页)

```
from mmdet3d.core import AssignResult
from mmdet3d.core.bbox import bbox3d2result, bbox3d2roi
from mmdet.core import build_assigner, build_sampler
from mmdet.models import HEADS
from ..builder import build_head, build_roi_extractor
from .base_3droi_head import Base3DRoIHead


@HEADS.register_module()
class PartAggregationROIHead(Base3DRoIHead):
    """Part aggregation roi head for PartA2.
    Args:
        semantic_head (ConfigDict): Config of semantic head.
        num_classes (int): The number of classes.
        seg_roi_extractor (ConfigDict): Config of seg_roi_extractor.
        part_roi_extractor (ConfigDict): Config of part_roi_extractor.
        bbox_head (ConfigDict): Config of bbox_head.
        train_cfg (ConfigDict): Training config.
        test_cfg (ConfigDict): Testing config.
    """

    def __init__(self,
                 semantic_head,
                 num_classes=3,
                 seg_roi_extractor=None,
                 part_roi_extractor=None,
                 bbox_head=None,
                 train_cfg=None,
                 test_cfg=None,
                 init_cfg=None):
        super(PartAggregationROIHead, self).__init__(
            bbox_head=bbox_head,
            train_cfg=train_cfg,
            test_cfg=test_cfg,
            init_cfg=init_cfg)
        self.num_classes = num_classes
        assert semantic_head is not None
        self.semantic_head = build_head(semantic_head)

        if seg_roi_extractor is not None:
            self.seg_roi_extractor = build_roi_extractor(seg_roi_extractor)
        if part_roi_extractor is not None:
```

(下页继续)

(续上页)

```
            self.part_roi_extractor = build_roi_extractor(part_roi_extractor)

        self.init_assigner_sampler()

    def _bbox_forward(self, seg_feats, part_feats, voxels_dict, rois):
        """Forward function of roi_extractor and bbox_head used in both
        training and testing.
        Args:
            seg_feats (torch.Tensor): Point-wise semantic features.
            part_feats (torch.Tensor): Point-wise part prediction features.
            voxels_dict (dict): Contains information of voxels.
            rois (Tensor): Roi boxes.
        Returns:
            dict: Contains predictions of bbox_head and
                features of roi_extractor.
        """
        pooled_seg_feats = self.seg_roi_extractor(seg_feats,
                                                  voxels_dict['voxel_centers'],
                                                  voxels_dict['coors'][..., 0],
                                                  rois)
        pooled_part_feats = self.part_roi_extractor(
            part_feats, voxels_dict['voxel_centers'],
            voxels_dict['coors'][..., 0], rois)
        cls_score, bbox_pred = self.bbox_head(pooled_seg_feats,
                                              pooled_part_feats)

        bbox_results = dict(
            cls_score=cls_score,
            bbox_pred=bbox_pred,
            pooled_seg_feats=pooled_seg_feats,
            pooled_part_feats=pooled_part_feats)
        return bbox_results
```

此处我们省略了与其他功能相关的细节，请参考 此处 获取更多细节。

最后，用户需要在 mmdet3d/models/bbox_heads/__init__.py 和 mmdet3d/models/roi_heads/__init__.py 中添加新模块，使得对应的注册器能够发现并加载该模块。

此外，用户也可以添加以下的代码到配置文件中，从而实现相同的目标。

```
custom_imports=dict(
    imports=['mmdet3d.models.roi_heads.part_aggregation_roi_head', 'mmdet3d.models.
↪roi_heads.bbox_heads.parta2_bbox_head'])
```

PartAggregationROIHead 的配置文件如下所示：

```
model = dict(
    ...
    roi_head=dict(
        type='PartAggregationROIHead',
        num_classes=3,
        semantic_head=dict(
            type='PointwiseSemanticHead',
            in_channels=16,
            extra_width=0.2,
            seg_score_thr=0.3,
            num_classes=3,
            loss_seg=dict(
                type='FocalLoss',
                use_sigmoid=True,
                reduction='sum',
                gamma=2.0,
                alpha=0.25,
                loss_weight=1.0),
            loss_part=dict(
                type='CrossEntropyLoss', use_sigmoid=True, loss_weight=1.0)),
        seg_roi_extractor=dict(
            type='Single3DRoIAwareExtractor',
            roi_layer=dict(
                type='RoIAwarePool3d',
                out_size=14,
                max_pts_per_voxel=128,
                mode='max')),
        part_roi_extractor=dict(
            type='Single3DRoIAwareExtractor',
            roi_layer=dict(
                type='RoIAwarePool3d',
                out_size=14,
                max_pts_per_voxel=128,
                mode='avg')),
        bbox_head=dict(
            type='PartA2BboxHead',
            num_classes=3,
            seg_in_channels=16,
            part_in_channels=4,
            seg_conv_channels=[64, 64],
            part_conv_channels=[64, 64],
            merge_conv_channels=[128, 128],
            down_conv_channels=[128, 256],
            bbox_coder=dict(type='DeltaXYZWLHRBBoxCoder'),
```

(下页继续)

(续上页)

```
            shared_fc_channels=[256, 512, 512, 512],
            cls_channels=[256, 256],
            reg_channels=[256, 256],
            dropout_ratio=0.1,
            roi_feat_size=14,
            with_corner_loss=True,
            loss_bbox=dict(
                type='SmoothL1Loss',
                beta=1.0 / 9.0,
                reduction='sum',
                loss_weight=1.0),
            loss_cls=dict(
                type='CrossEntropyLoss',
                use_sigmoid=True,
                reduction='sum',
                loss_weight=1.0)))
    ...
    )
```

MMDetection 2.0 支持配置文件之间的继承，使得用户能够更加关注自己的配置文件的修改。PartA2 Head 的第二阶段主要使用新建的 `PartAggregationROIHead` 和 `PartA2BboxHead`，需要根据对应模块的 `__init__` 参数来设置对应的参数。

24.1.5 添加新建 loss

假定用户想要新添一个用于检测框回归的 loss，并命名为 `MyLoss`。为了添加一个新的 loss ，用于需要在 `mmdet3d/models/losses/my_loss.py` 中实现对应的逻辑。装饰器 `weighted_loss` 能够保证对 batch 中每个样本的 loss 进行加权平均。

```
import torch
import torch.nn as nn

from ..builder import LOSSES
from .utils import weighted_loss

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

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

(下页继续)

(续上页)

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

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

接着，用户需要将 loss 添加到 mmdet3d/models/losses/__init__.py：

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

此外，用户也可以添加以下的代码到配置文件中，从而实现相同的目标。

```
custom_imports=dict(
    imports=['mmdet3d.models.losses.my_loss'])
```

为了使用该 loss，需要对 loss_xxx 域进行修改。因为 MyLoss 主要用于检测框的回归，因此需要在对应的 head 中修改 loss_bbox 域的值。

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

CHAPTER 25

教程 5: 自定义运行时配置

25.1 自定义优化器设置

25.1.1 自定义 PyTorch 支持的优化器

我们已经支持使用所有 PyTorch 实现的优化器，且唯一需要修改的地方就是改变配置文件中的 `optimizer` 字段。举个例子，如果您想使用 `ADAM`（注意到这样可能会使性能大幅下降），您可以这样修改：

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

为了修改模型的学习率，用户只需要修改优化器配置中的 `lr` 字段。用户可以根据 PyTorch 的 API 文档 直接设置参数。

25.1.2 自定义并实现优化器

1. 定义新的优化器

一个自定义优化器可以按照如下过程定义：

假设您想要添加一个叫 `MyOptimizer` 的，拥有参数 `a`，`b` 和 `c` 的优化器，您需要创建一个叫做 `mmdet3d/core/optimizer` 的目录。接下来，应该在目录下某个文件中实现新的优化器，比如 `mmdet3d/core/optimizer/my_optimizer.py`：

```
from mmcv.runner.optimizer import OPTIMIZERS
from torch.optim import Optimizer


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

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

2. 将优化器添加到注册器

为了找到上述定义的优化器模块，该模块首先需要被引入主命名空间。有两种方法实现之：

- 新建 mmdet3d/core/optimizer/__init__.py 文件用于引入。

  新定义的模块应该在 mmdet3d/core/optimizer/__init__.py 中被引入，使得注册器可以找到新模块并注册之：

```
from .my_optimizer import MyOptimizer

__all__ = ['MyOptimizer']
```

您也需要通过添加如下语句在 mmdet3d/core/__init__.py 中引入 optimizer：

```
from .optimizer import *
```

或者在配置中使用 custom_imports 来人工引入新优化器：

```
custom_imports = dict(imports=['mmdet3d.core.optimizer.my_optimizer'], allow_failed_
↪imports=False)
```

模块 mmdet3d.core.optimizer.my_optimizer 会在程序伊始被引入，且 MyOptimizer 类在那时会自动被注册。注意到只有包含 MyOptimizer 类的包应该被引入。mmdet3d.core.optimizer.my_optimizer.MyOptimizer **不能**被直接引入。

事实上，用户可以在这种引入的方法中使用完全不同的文件目录结构，只要保证根目录能在 PYTHONPATH 中被定位。

3. 在配置文件中指定优化器

接下来您可以在配置文件的 optimizer 字段中使用 MyOptimizer。在配置文件中，优化器在 optimizer 字段中以如下方式定义：

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

为了使用您自己的优化器，该字段可以改为：

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

25.1.3 自定义优化器的构造器

部分模型可能会拥有一些参数专属的优化器设置，比如 BatchNorm 层的权重衰减 (weight decay)。用户可以通过自定义优化器的构造器来对那些细粒度的参数进行调优。

```
from mmcv.utils import build_from_cfg

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


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

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

    def __call__(self, model):

        return my_optimizer
```

默认优化器构造器在这里实现。这部分代码也可以用作新优化器构造器的模版。

25.1.4 额外的设置

没有在优化器部分实现的技巧应该通过优化器构造器或者钩子来实现（比如逐参数的学习率设置）。我们列举了一些常用的可以稳定训练过程或者加速训练的设置。我们欢迎提供更多类似设置的 PR 和 issue。

- **使用梯度裁剪 (gradient clip) 来稳定训练过程**：

  一些模型依赖梯度裁剪技术来裁剪训练中的梯度，以稳定训练过程。举例如下：

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

如果您的配置继承了一个已经设置了 optimizer_config 的基础配置，那么您可能需要 _delete_=True 字段来覆盖基础配置中无用的设置。详见配置文件的说明文档。

- **使用动量规划器 (momentum scheduler) 来加速模型收敛**：

  我们支持用动量规划器来根据学习率更改模型的动量，这样可以使模型更快地收敛。动量规划器通常和学习率规划器一起使用，比如说，如下配置文件在 3D 检测中被用于加速模型收敛。更多细节详见 CyclicLrUpdater 和 CyclicMomentumUpdater 的实现。

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

25.2 自定义训练规程

默认情况，我们使用阶梯式学习率衰减的 1 倍训练规程。这会调用 MMCV 中的 StepLRHook。我们在这里支持很多其他学习率规划方案，比如余弦退火和多项式衰减规程。下面是一些样例：

- 多项式衰减规程:

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

- 余弦退火规程:

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

25.3 自定义工作流

工作流是一个（阶段，epoch 数）的列表，用于指定不同阶段运行顺序和运行的 epoch 数。默认情况它被设置为：

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

这意味着，工作流包括训练 1 个 epoch。有时候用户可能想要检查一些模型在验证集上的评估指标（比如损失、准确率）。在这种情况中，我们可以将工作流设置如下：

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

这样，就是交替地运行 1 个 epoch 进行训练，1 个 epoch 进行验证。

请注意:

1. 模型参数在验证期间不会被更新。
2. 配置文件中，`runner` 里的 `max_epochs` 字段只控制训练 epoch 的数量，而不会影响验证工作流。
3. `[('train', 1), ('val', 1)]` 和 `[('train', 1)]` 工作流不会改变 `EvalHook` 的行为，这是因为 `EvalHook` 被 `after_train_epoch` 调用，且验证工作流只会影响通过 `after_val_epoch` 调用的钩子。因此，`[('train', 1), ('val', 1)]` 和 `[('train', 1)]` 的唯一区别就是执行器 (runner) 会在每个训练 epoch 之后在验证集上计算损失。

25.4 自定义钩子

25.4.1 自定义并实现钩子

1. 实现一个新钩子

存在一些情况下用户可能需要实现新钩子。在版本 v2.3.0 之后，MMDetection 支持自定义训练过程中的钩子 (#3395)。因此用户可以直接在 mmdet 中，或者在其基于 mmdet 的代码库中实现钩子并通过更改训练配置来使用钩子。在 v2.3.0 之前，用户需要更改代码以使得训练开始之前钩子已经注册完毕。这里我们给出一个，在 mmdet3d 中创建并使用新钩子的例子。

```
from mmcv.runner import HOOKS, Hook


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

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

(下页继续)

(续上页)

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

    def after_run(self, runner):
        pass

    def before_epoch(self, runner):
        pass

    def after_epoch(self, runner):
        pass

    def before_iter(self, runner):
        pass

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

取决于钩子的功能，用户需要指定钩子在每个训练阶段时的行为，具体包括如下阶段：`before_run`，`after_run`，`before_epoch`，`after_epoch`，`before_iter`，和 `after_iter`。

2. 注册新钩子

接下来我们需要引入 `MyHook`。假设新钩子位于文件 `mmdet3d/core/utils/my_hook.py` 中，有两种方法可以实现之：

- 更改 `mmdet3d/core/utils/__init__.py` 来引入之：

  新定义的模块应在 `mmdet3d/core/utils/__init__.py` 中引入，以使得注册器可以找到新模块并注册之：

```
from .my_hook import MyHook

__all__ = [..., 'MyHook']
```

或者在配置中使用 `custom_imports` 来人为地引入之

```
custom_imports = dict(imports=['mmdet3d.core.utils.my_hook'], allow_failed_
↪imports=False)
```

3. 更改配置文件

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

您可以将字段 `priority` 设置为 `'NORMAL'` 或者 `'HIGHEST'`，来设置钩子的优先级，如下所示：

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

默认情况，在注册阶段钩子的优先级被设置为 `NORMAL`。

25.4.2 使用 MMCV 中实现的钩子

如果钩子已经在 MMCV 中被实现了，您可以直接通过更改配置文件来使用该钩子：

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

25.4.3 更改默认的运行时钩子

有一些常用的钩子并没有通过 `custom_hooks` 注册，它们是：

- 日志配置 (log_config)
- 检查点配置 (checkpoint_config)
- 评估 (evaluation)
- 学习率配置 (lr_config)
- 优化器配置 (optimizer_config)
- 动量配置 (momentum_config)

在这些钩子中，只有日志钩子拥有 `VERY_LOW` 的优先级，其他钩子的优先级均为 `NORMAL`。上述教程已经涉及了如何更改 `optimizer_config`，`momentum_config`，和 `lr_config`。下面我们展示如何在 `log_config`，`checkpoint_config`，和 `evaluation` 上做文章。

检查点配置

MMCV 执行器会使用 checkpoint_config 来初始化 CheckpointHook。

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

用户可以设置 max_keep_ckpts 来保存一定少量的检查点，或者用 save_optimizer 来决定是否保存优化器的状态。更多参数的细节详见这里。

日志配置

log_config 将多个日志钩子封装在一起，并允许设置日志记录间隔。现在 MMCV 支持 WandbLoggerHook，MlflowLoggerHook，和 TensorboardLoggerHook。更详细的使用方法请移步 MMCV 文档。

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

评估配置

evaluation 的配置会被用于初始化 EvalHook。除了 interval 字段，其他参数，比如 metric，会被传递给 dataset.evaluate()。

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

CHAPTER 26

教程 6: 坐标系

26.1 概述

MMDetection3D 使用 3 种不同的坐标系。3D 目标检测领域中不同坐标系的存在是非常有必要的，因为对于各种 3D 数据采集设备来说，如激光雷达、深度相机等，使用的坐标系是不一致的，不同的 3D 数据集也遵循不同的数据格式。早期的工作，比如 SECOND、VoteNet 将原始数据转换为另一种格式，形成了一些后续工作也遵循的约定，使得不同坐标系之间的转换变得更加复杂。

尽管数据集和采集设备多种多样，但是通过总结 3D 目标检测的工作线，我们可以将坐标系大致分为三类:

- 相机坐标系–大多数相机的坐标系，在该坐标系中 y 轴正方向指向地面，x 轴正方向指向右侧，z 轴正方向指向前方。

- 激光雷达坐标系–众多激光雷达的坐标系，在该坐标系中 z 轴负方向指向地面，x 轴正方向指向前方，y 轴正方向指向左侧。

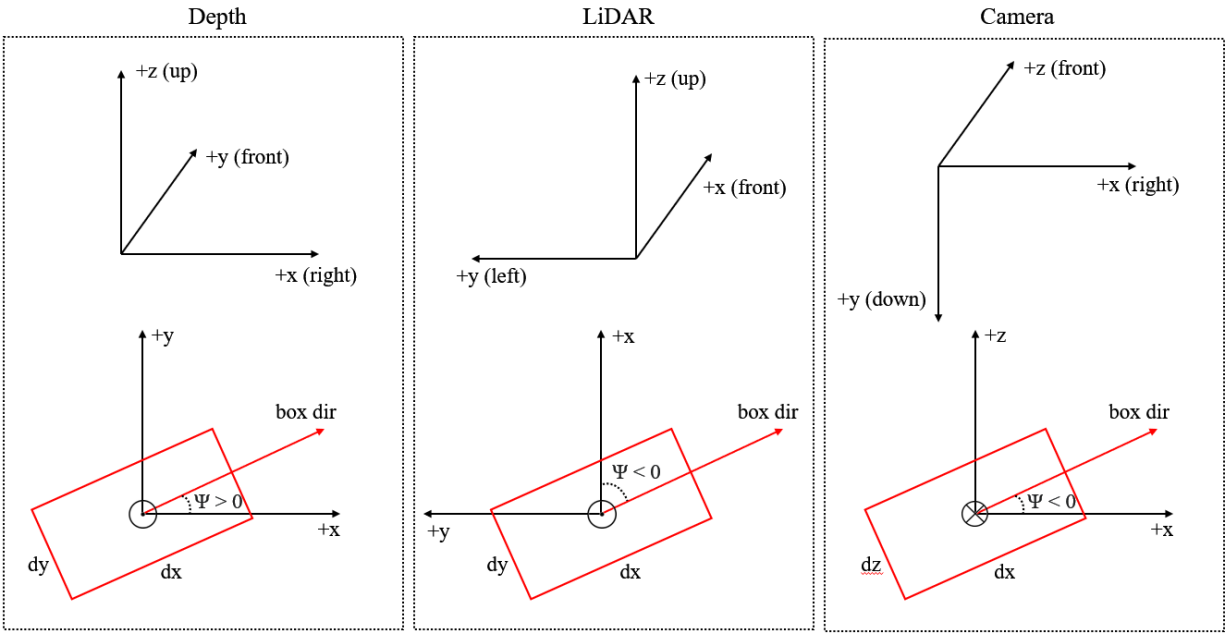
```
                z 上    x 前
                 ^     ^
                 |    /
                 |   /
                 |  /
                 |/
y 左    <------ 0 ------ 右
```

- 深度坐标系–VoteNet、H3DNet 等模型使用的坐标系，在该坐标系中 z 轴负方向指向地面，x 轴正方向指向右侧，y 轴正方向指向前方。

```
              z 上    y 前
               ^     ^
               |    /
               |   /
               |  /
               |/
左    ------ 0 ------> x 右
```

该教程中的坐标系定义实际上**不仅仅是定义三个轴**。对于形如 (x, y, z, dx, dy, dz, r) 的框来说，我们的坐标系也定义了如何解释框的尺寸 (dx, dy, dz) 和转向角 (yaw) 角度 r。

三个坐标系的图示如下：



上面三张图是 3D 坐标系，下面三张图是鸟瞰图。

以后我们将坚持使用本教程中定义的三个坐标系。

26.2 转向角 (yaw) 的定义

请参考维基百科了解转向角的标准定义。在目标检测中，我们选择一个轴作为重力轴，并在垂直于重力轴的平面 Π 上选取一个参考方向，那么参考方向的转向角为 0，在 Π 上的其他方向有非零的转向角，其角度取决于其与参考方向的角度。

目前，对于所有支持的数据集，标注不包括俯仰角 (pitch) 和滚动角 (roll)，这意味着我们在预测框和计算框之间的重叠时只需考虑转向角 (yaw)。

在 MMDetection3D 中，所有坐标系都是右手坐标系，这意味着如果从重力轴的负方向（轴的正方向指向人眼）看，转向角 (yaw) 沿着逆时针方向增加。

下图显示，在右手坐标系中，如果我们设定 x 轴正方向为参考方向，那么 y 轴正方向的转向角 (yaw) 为 $\frac{\pi}{2}$。

```
                      z 上   y 前 (yaw=0.5*pi)
                       ^     ^
                       |    /
                       |   /
                       |  /
                       | /
                       |/
左 (yaw=pi)     ------ 0 ------> x 右 (yaw=0)
```

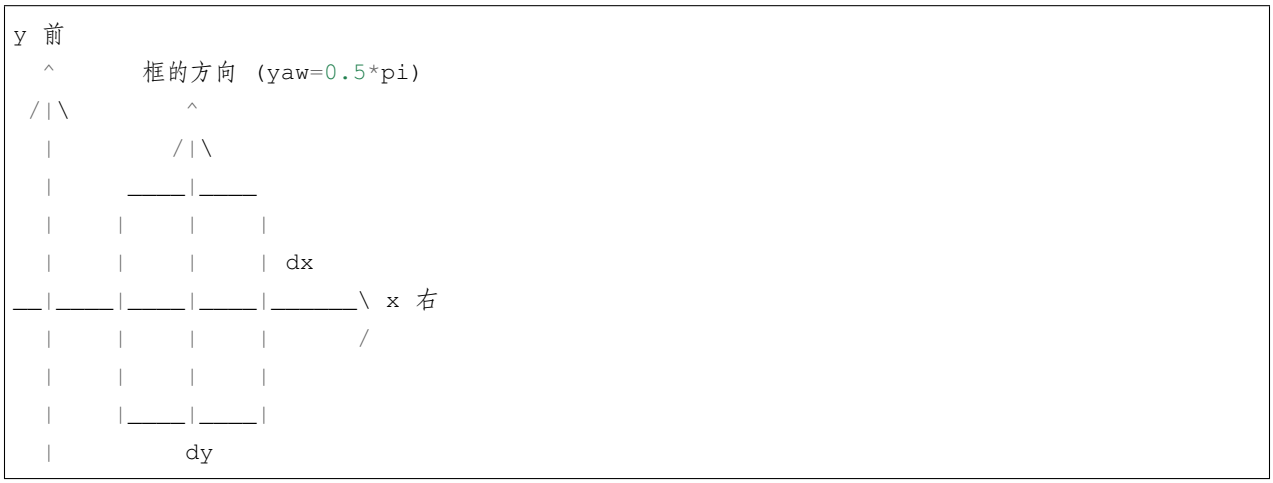
对于一个框来说，其转向角 (yaw) 的值等于其方向减去一个参考方向。在 MMDetection3D 的所有三个坐标系中，参考方向总是 x 轴的正方向，而如果一个框的转向角 (yaw) 为 0，则其方向被定义为与 x 轴平行。框的转向角 (yaw) 的定义如下图所示。

```
  y 前
  ^       框的方向 (yaw=0.5*pi)
 /|\         ^
  |         /|\
  |      ____|____
  |     |    |    |
  |     |    |    |
__|_____|____|____|_______\ x 右
  |     |    |    |       /
  |     |    |    |
  |     |____|____|
  |
```

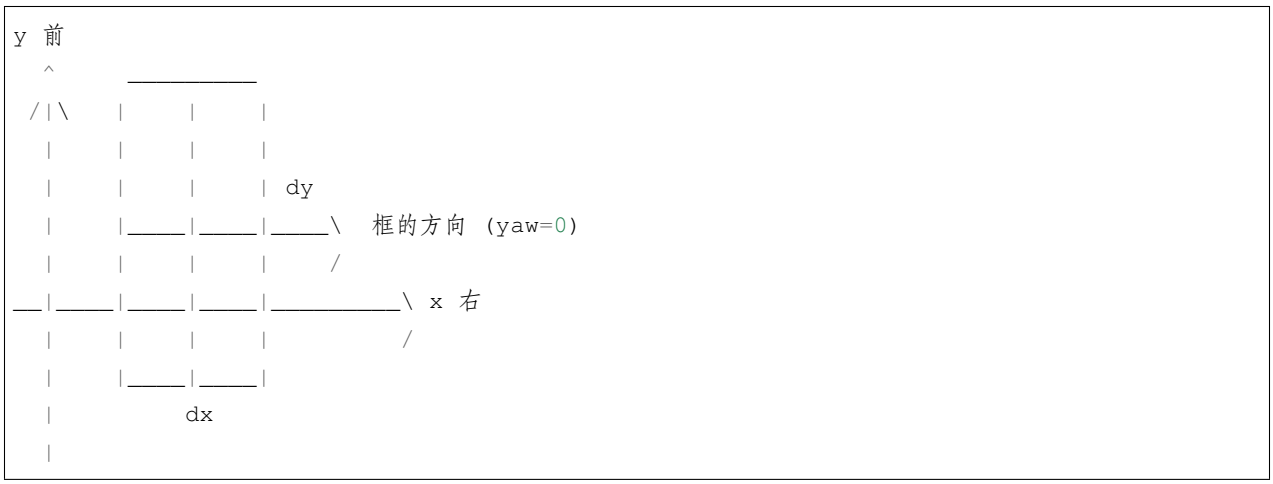
26.3 框尺寸的定义

框尺寸的定义与转向角 (yaw) 的定义是分不开的。在上一节中，我们提到如果一个框的转向角 (yaw) 为 0，它的方向就被定义为与 x 轴平行。那么自然地，一个框对应于 x 轴的尺寸应该是 dx。但是，这在某些数据集中并非总是如此（我们稍后会解决这个问题）。

下图展示了 x 轴和 dx，y 轴和 dy 对应的含义。



注意框的方向总是和 dx 边平行。

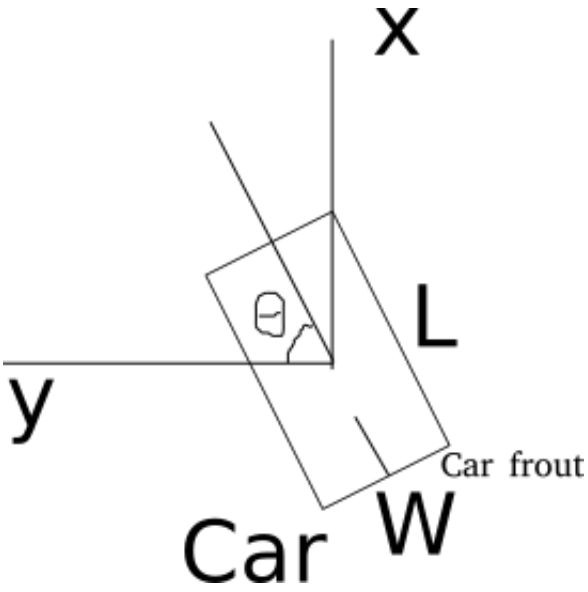


26.4 与支持的数据集的原始坐标系的关系

26.4.1 KITTI

KITTI 数据集的原始标注是在相机坐标系下的，详见 get_label_anno。在 MMDetection3D 中，为了在 KITTI 数据集上训练基于激光雷达的模型，首先将数据从相机坐标系转换到激光雷达坐标，详见 get_ann_info。对于训练基于视觉的模型，数据保持在相机坐标系不变。

在 SECOND 中，框的激光雷达坐标系定义如下（鸟瞰图）：



对于每个框来说，尺寸为 (w, l, h)，转向角 (yaw) 的参考方向为 y 轴正方向。更多细节请参考代码库。

我们的激光雷达坐标系有两处改变：

- 转向角 (yaw) 被定义为右手而非左手，从而保持一致性；
- 框的尺寸为 (l, w, h) 而非 (w, l, h)，由于在 KITTI 数据集中 w 对应 dy，l 对应 dx。

26.4.2 Waymo

我们使用 Waymo 数据集的 KITTI 格式数据。因此，在我们的实现中 KITTI 和 Waymo 也共用相同的坐标系。

26.4.3 NuScenes

NuScenes 提供了一个评估工具包，其中每个框都被包装成一个 `Box` 实例。`Box` 的坐标系不同于我们的激光雷达坐标系，在 `Box` 坐标系中，前两个表示框尺寸的元素分别对应 (dy, dx) 或者 (w, l)，和我们的表示方法相反。更多细节请参考 NuScenes 教程。

读者可以参考 NuScenes 开发工具，了解 NuScenes 框 的定义和 NuScenes 评估的过程。

26.4.4 Lyft

就涉及坐标系而言，Lyft 和 NuScenes 共用相同的数据格式。

请参考官方网站获取更多信息。

26.4.5 ScanNet

ScanNet 的原始数据不是点云而是网格，需要在我们的深度坐标系下进行采样得到点云数据。对于 ScanNet 检测任务，框的标注是轴对齐的，并且转向角 (yaw) 始终是 0。因此，我们的深度坐标系中转向角 (yaw) 的方向对 ScanNet 没有影响。

26.4.6 SUN RGB-D

SUN RGB-D 的原始数据不是点云而是 RGB-D 图像。我们通过反投影，可以得到每张图像对应的点云，其在我们的深度坐标系下。但是，数据集的标注并不在我们的系统中，所以需要进行转换。

将原始标注转换为我们的深度坐标系下的标注的转换过程请参考 sunrgbd_data_utils.py。

26.4.7 S3DIS

在我们的实现中，S3DIS 与 ScanNet 共用相同的坐标系。然而 S3DIS 是一个仅限于分割任务的数据集，因此没有标注是坐标系敏感的。

26.5 例子

26.5.1 框（在不同坐标系间）的转换

以相机坐标系和激光雷达坐标系间的转换为例：

首先，对于点和框的中心点，坐标转换前后满足下列关系：

- $x_{LiDAR} = z_{camera}$
- $y_{LiDAR} = -x_{camera}$
- $z_{LiDAR} = -y_{camera}$

然后，框的尺寸转换前后满足下列关系：

- $dx_{LiDAR} = dx_{camera}$
- $dy_{LiDAR} = dz_{camera}$
- $dz_{LiDAR} = dy_{camera}$

最后，转向角 (yaw) 也应该被转换：

- $r_{LiDAR} = -\frac{\pi}{2} - r_{camera}$

详见此处代码了解更多细节。

26.5.2 鸟瞰图

如果 3D 框是 (x, y, z, dx, dy, dz, r)，相机坐标系下框的鸟瞰图是 $(x, z, dx, dz, -r)$。转向角 (yaw) 符号取反是因为相机坐标系重力轴的正方向指向地面。

详见此处代码了解更多细节。

26.5.3 框的旋转

我们将各种框的旋转设定为绕着重力轴逆时针旋转。因此，为了旋转一个 3D 框，我们首先需要计算新的框的中心，然后将旋转角度添加到转向角 (yaw)。

详见此处代码了解更多细节。

26.6 常见问题

26.6.1 Q1: 与框相关的算子是否适用于所有坐标系类型？

否。例如，用于 RoI-Aware Pooling 的算子只适用于深度坐标系和激光雷达坐标系下的框。由于如果从上方看，旋转是顺时针的，所以 KITTI 数据集这里的评估函数仅适用于相机坐标系下的框。

对于每个和框相关的算子，我们注明了其所适用的框类型。

26.6.2 Q2: 在每个坐标系中，三个轴是否分别准确地指向右侧、前方和地面？

否。例如在 KITTI 中，从相机坐标系转换为激光雷达坐标系时，我们需要一个校准矩阵。

26.6.3 Q3: 框中转向角 (yaw) 2π 的相位差如何影响评估？

对于交并比 (IoU) 计算，转向角 (yaw) 有 2π 的相位差的两个框是相同的，所以不会影响评估。

对于角度预测评估，例如 NuScenes 中的 NDS 指标和 KITTI 中的 AOS 指标，会先对预测框的角度进行标准化，因此 2π 的相位差不会改变结果。

26.6.4 Q4: 框中转向角 (yaw) π 的相位差如何影响评估？

对于交并比 (IoU) 计算，转向角 (yaw) 有 π 的相位差的两个框是相同的，所以不会影响评估。

然而，对于角度预测评估，这会导致完全相反的方向。

考虑一辆汽车，转向角 (yaw) 是汽车前部方向与 x 轴正方向之间的夹角。如果我们将该角度增加 π，车前部将变成车后部。

对于某些类别，例如障碍物，前后没有区别，因此 π 的相位差不会对角度预测分数产生影响。

CHAPTER 27

教程 7: 后端支持

我们支持不同的文件客户端后端：磁盘、Ceph 和 LMDB 等。下面是修改配置使之从 Ceph 加载和保存数据的示例。

27.1 从 Ceph 读取数据和标注文件

我们支持从 Ceph 加载数据和生成的标注信息文件（pkl 和 json）：

```
# set file client backends as Ceph
file_client_args = dict(
    backend='petrel',
    path_mapping=dict({
        './data/nuscenes/':
        's3://openmmlab/datasets/detection3d/nuscenes/', # replace the path with your␣
↪data path on Ceph
        'data/nuscenes/':
        's3://openmmlab/datasets/detection3d/nuscenes/' # replace the path with your␣
↪data path on Ceph
    }))

db_sampler = dict(
    data_root=data_root,
    info_path=data_root + 'kitti_dbinfos_train.pkl',
    rate=1.0,
```

(下页继续)

(续上页)

```
    prepare=dict(filter_by_difficulty=[-1], filter_by_min_points=dict(Car=5)),
    sample_groups=dict(Car=15),
    classes=class_names,
    # set file client for points loader to load training data
    points_loader=dict(
        type='LoadPointsFromFile',
        coord_type='LIDAR',
        load_dim=4,
        use_dim=4,
        file_client_args=file_client_args),
    # set file client for data base sampler to load db info file
    file_client_args=file_client_args)

train_pipeline = [
    # set file client for loading training data
    dict(type='LoadPointsFromFile', coord_type='LIDAR', load_dim=4, use_dim=4, file_
↪client_args=file_client_args),
    # set file client for loading training data annotations
    dict(type='LoadAnnotations3D', with_bbox_3d=True, with_label_3d=True, file_client_
↪args=file_client_args),
    dict(type='ObjectSample', db_sampler=db_sampler),
    dict(
        type='ObjectNoise',
        num_try=100,
        translation_std=[0.25, 0.25, 0.25],
        global_rot_range=[0.0, 0.0],
        rot_range=[-0.15707963267, 0.15707963267]),
    dict(type='RandomFlip3D', flip_ratio_bev_horizontal=0.5),
    dict(
        type='GlobalRotScaleTrans',
        rot_range=[-0.78539816, 0.78539816],
        scale_ratio_range=[0.95, 1.05]),
    dict(type='PointsRangeFilter', point_cloud_range=point_cloud_range),
    dict(type='ObjectRangeFilter', point_cloud_range=point_cloud_range),
    dict(type='PointShuffle'),
    dict(type='DefaultFormatBundle3D', class_names=class_names),
    dict(type='Collect3D', keys=['points', 'gt_bboxes_3d', 'gt_labels_3d'])
]
test_pipeline = [
    # set file client for loading validation/testing data
    dict(type='LoadPointsFromFile', coord_type='LIDAR', load_dim=4, use_dim=4, file_
↪client_args=file_client_args),
    dict(
```

(下页继续)

(续上页)

```
        type='MultiScaleFlipAug3D',
        img_scale=(1333, 800),
        pts_scale_ratio=1,
        flip=False,
        transforms=[
            dict(
                type='GlobalRotScaleTrans',
                rot_range=[0, 0],
                scale_ratio_range=[1., 1.],
                translation_std=[0, 0, 0]),
            dict(type='RandomFlip3D'),
            dict(
                type='PointsRangeFilter', point_cloud_range=point_cloud_range),
            dict(
                type='DefaultFormatBundle3D',
                class_names=class_names,
                with_label=False),
            dict(type='Collect3D', keys=['points'])
        ])
]

data = dict(
    # set file client for loading training info files (.pkl)
    train=dict(
        type='RepeatDataset',
        times=2,
        dataset=dict(pipeline=train_pipeline, classes=class_names, file_client_
↪args=file_client_args)),
    # set file client for loading validation info files (.pkl)
    val=dict(pipeline=test_pipeline, classes=class_names,file_client_args=file_client_
↪args),
    # set file client for loading testing info files (.pkl)
    test=dict(pipeline=test_pipeline, classes=class_names, file_client_args=file_
↪client_args))
```

27.2 从 Ceph 读取预训练模型

```
model = dict(
    pts_backbone=dict(
        _delete_=True,
        type='NoStemRegNet',
        arch='regnetx_1.6gf',
        init_cfg=dict(
            type='Pretrained', checkpoint='s3://openmmlab/checkpoints/mmdetection3d/
↪regnetx_1.6gf'), # replace the path with your pretrained model path on Ceph
        ...
```

27.3 从 Ceph 读取模型权重文件

```
# replace the path with your checkpoint path on Ceph
load_from = 's3://openmmlab/checkpoints/mmdetection3d/v0.1.0_models/pointpillars/hv_
↪pointpillars_secfpn_6x8_160e_kitti-3d-car/hv_pointpillars_secfpn_6x8_160e_kitti-3d-
↪car_20200620_230614-77663cd6.pth'
resume_from = None
workflow = [('train', 1)]
```

27.4 保存模型权重文件至 Ceph

```
# checkpoint saving
# replace the path with your checkpoint saving path on Ceph
checkpoint_config = dict(interval=1, max_keep_ckpts=2, out_dir='s3://openmmlab/
↪mmdetection3d')
```

27.5 EvalHook 保存最优模型权重文件至 Ceph

```
# replace the path with your checkpoint saving path on Ceph
evaluation = dict(interval=1, save_best='bbox', out_dir='s3://openmmlab/mmdetection3d
↪')
```

27.6 训练日志保存至 Ceph

训练后的训练日志会备份到指定的 Ceph 路径。

```
log_config = dict(
    interval=50,
    hooks=[
        dict(type='TextLoggerHook', out_dir='s3://openmmlab/mmdetection3d'),
    ])
```

您还可以通过设置 `keep_local = False` 备份到指定的 Ceph 路径后删除本地训练日志。

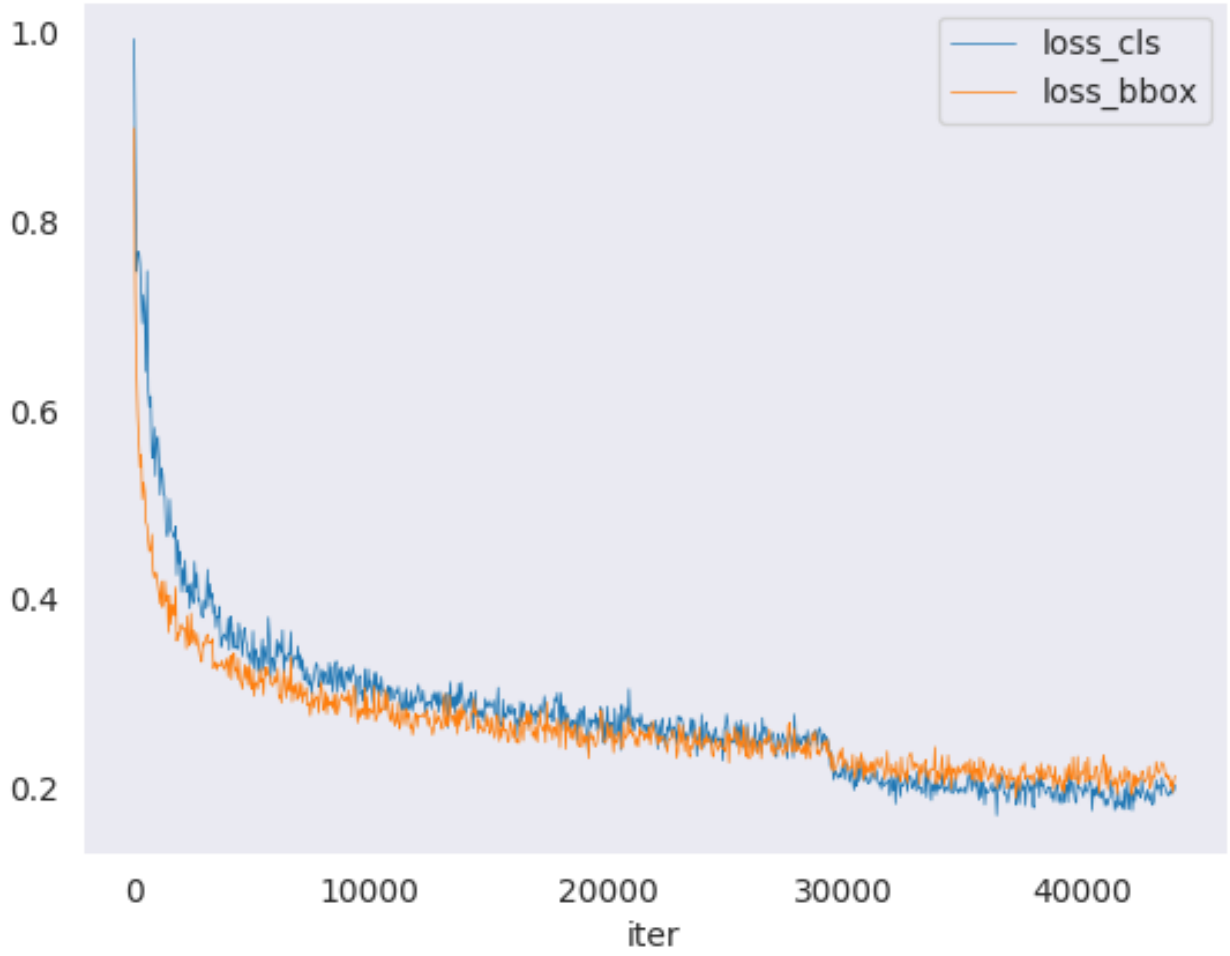
```
log_config = dict(
    interval=50,
    hooks=[
        dict(type='TextLoggerHook', out_dir='s3://openmmlab/mmdetection3d', keep_
↪local=False),
    ])
```

我们在 `tools/` 文件夹路径下提供了许多有用的工具。

CHAPTER 28

日志分析

给定一个训练的日志文件，您可以绘制出 loss/mAP 曲线。首先需要运行 `pip install seaborn` 安装依赖包。




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

注意: 如果您想绘制的指标是在验证阶段计算得到的，您需要添加一个标志 `--mode eval` ，如果您每经过一个 `${INTERVAL}` 的间隔进行评估，您需要增加一个参数 `--interval ${INTERVAL}`。

示例：

- 绘制出某次运行的分类 loss。

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

- 绘制出某次运行的分类和回归 loss，并且保存图片为 pdf 格式。

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

- 在同一张图片中比较两次运行的 bbox mAP。

```
# 根据 Car_3D_moderate_strict 在 KITTI 上评估 PartA2 和 second。
python tools/analysis_tools/analyze_logs.py plot_curve tools/logs/PartA2.log.json␣
↪tools/logs/second.log.json --keys KITTI/Car_3D_moderate_strict --legend PartA2␣
↪second --mode eval --interval 1
# 根据 Car_3D_moderate_strict 在 KITTI 上分别对车和 3 类评估 PointPillars。
python tools/analysis_tools/analyze_logs.py plot_curve tools/logs/pp-3class.log.
↪json tools/logs/pp.log.json --keys KITTI/Car_3D_moderate_strict --legend pp-
↪3class pp --mode eval --interval 2
```

您也能计算平均训练速度。

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

预期输出应该如下所示。

```
-----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 29

可视化

29.1 结果

为了观察模型的预测结果，您可以运行下面的指令

```
python tools/test.py ${CONFIG_FILE} ${CKPT_PATH} --show --show-dir ${SHOW_DIR}
```

在运行这个指令后，所有的绘制结果包括输入数据，以及在输入数据基础上可视化的网络输出和真值（例如：3D 单模态检测任务中的 ***_points.obj 和 ***_pred.obj），将会被保存在 ${SHOW_DIR}。

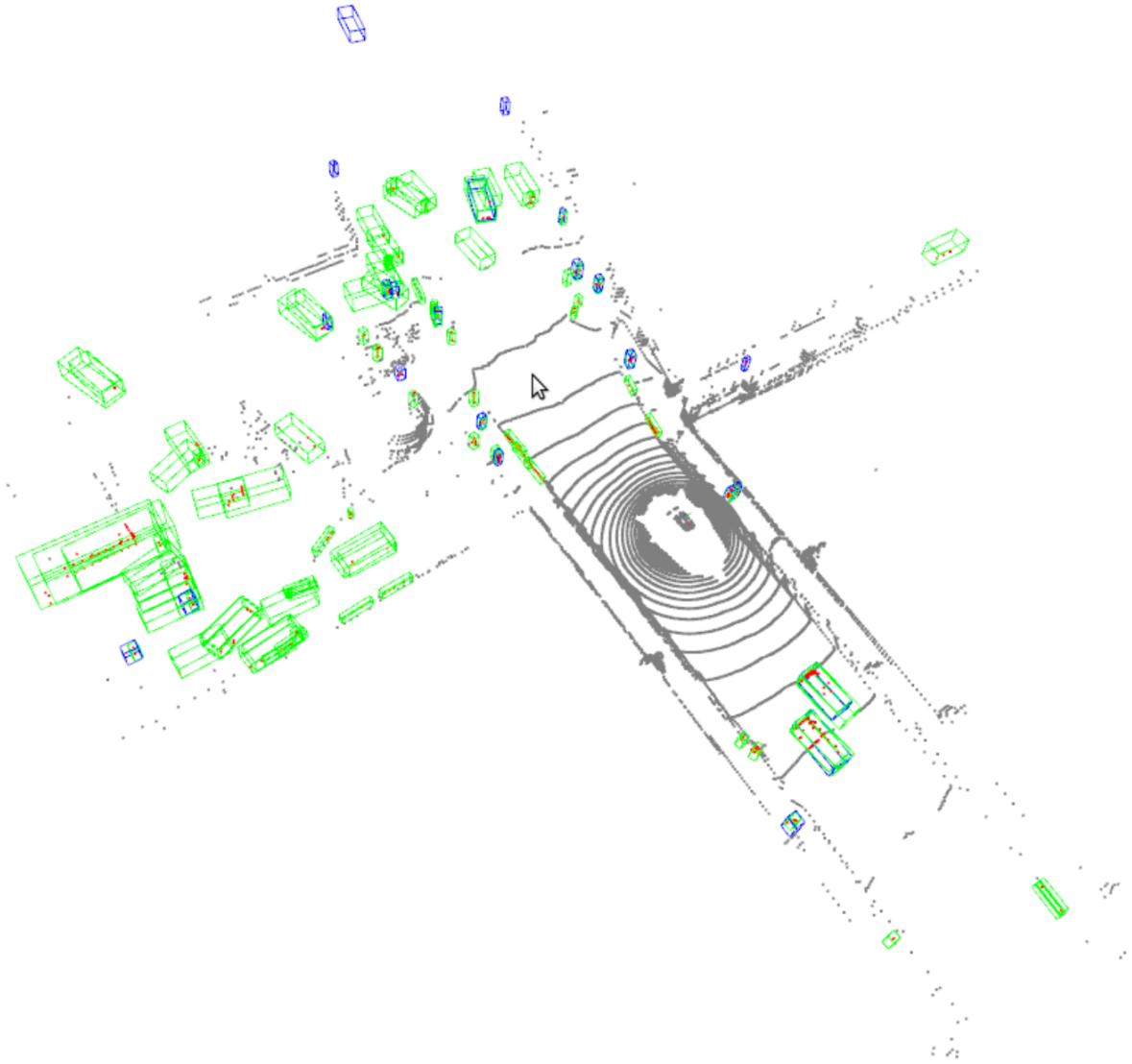
要在评估期间看见预测结果，您可以运行下面的指令

```
python tools/test.py ${CONFIG_FILE} ${CKPT_PATH} --eval 'mAP' --eval-options
↪'show=True' 'out_dir=${SHOW_DIR}'
```

在运行这个指令后，您将会在 ${SHOW_DIR} 获得输入数据、可视化在输入上的网络输出和真值标签（例如：在多模态检测任务中的 ***_points.obj，***_pred.obj，***_gt.obj，***_img.png 和 ***_pred.png ）。当 show 被激活，Open3D 将会被用来在线可视化结果。当您在没有 GUI 的远程服务器上运行测试的时候，无法进行在线可视化，您可以设定 show=False 将输出结果保存在 {SHOW_DIR}。

至于离线可视化，您将有两个选择。利用 Open3D 后端可视化结果，您可以运行下面的指令

```
python tools/misc/visualize_results.py ${CONFIG_FILE} --result ${RESULTS_PATH} --show-
↪dir ${SHOW_DIR}
```



或者您可以使用 3D 可视化软件，例如 MeshLab 来打开这些在 `${SHOW_DIR}` 目录下的文件，从而查看 3D 检测输出。具体来说，打开 `***_points.obj` 查看输入点云，打开 `***_pred.obj` 查看预测的 3D 边界框。这允许推理和结果生成在远程服务器中完成，用户可以使用 GUI 在他们的主机上打开它们。

注意：可视化接口有一些不稳定，我们将计划和 MMDetection 一起重构这一部分。

29.2 数据集

我们也提供脚本用来可视化数据集，而无需推理。您可以使用 `tools/misc/browse_dataset.py` 来在线显示载入的数据和真值标签，并且保存进磁盘。现在我们支持所有数据集上的单模态 3D 检测和 3D 分割，支持 KITTI 和 SUN RGB-D 数据集上的多模态 3D 检测，同时支持 nuScenes 数据集上的单目 3D 检测。为了浏览 KITTI 数据集，您可以运行下面的指令

```
python tools/misc/browse_dataset.py configs/_base_/datasets/kitti-3d-3class.py --task␣
↪det --output-dir ${OUTPUT_DIR} --online
```

注意：一旦指定 `--output-dir`，当按下 open3d 窗口的 `_ESC_`，用户指定的视图图像将被保存。如果您没有显示器，您可以移除 `--online` 标志，从而仅仅保存可视化结果并且进行离线浏览。

为了验证数据的一致性和数据增强的效果，您还可以使用以下命令添加 `--aug` 标志来可视化数据增强后的数据：

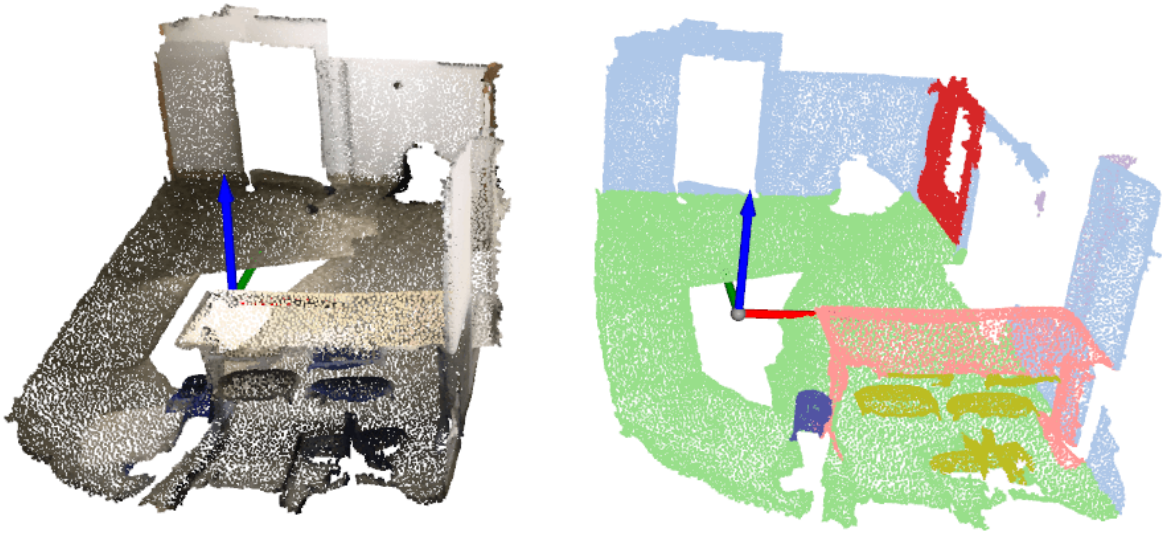
```
python tools/misc/browse_dataset.py configs/_base_/datasets/kitti-3d-3class.py --task␣
↪det --aug --output-dir ${OUTPUT_DIR} --online
```

如果您还想显示 2D 图像以及投影的 3D 边界框，则需要找到支持多模态数据加载的配置文件，然后将 `--task` 参数更改为 `multi_modality-det`。一个例子如下所示

```
python tools/misc/browse_dataset.py configs/mvxnet/dv_mvx-fpn_second_secfpn_adamw_2x8_
↪80e_kitti-3d-3class.py --task multi_modality-det --output-dir ${OUTPUT_DIR} --online
```

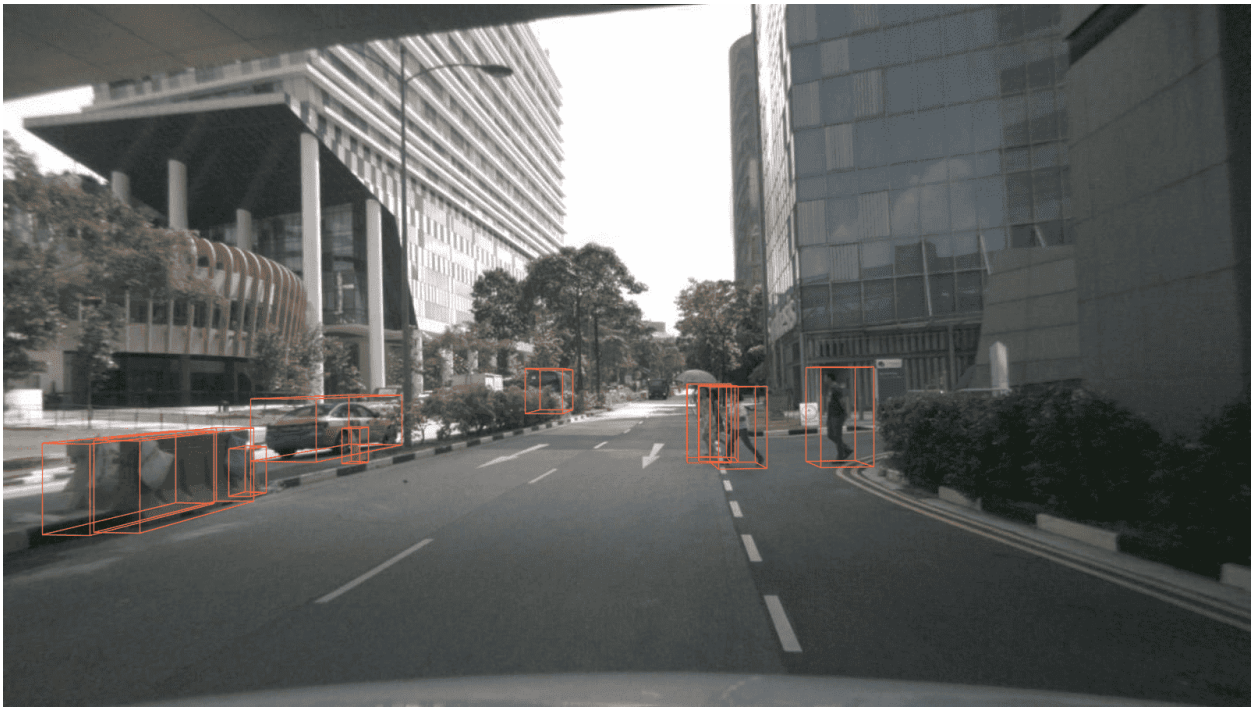
您可以简单的使用不同的配置文件，浏览不同的数据集，例如：在 3D 语义分割任务中可视化 ScanNet 数据集

```
python tools/misc/browse_dataset.py configs/_base_/datasets/scannet_seg-3d-20class.py␣
↪--task seg --output-dir ${OUTPUT_DIR} --online
```



在单目 3D 检测任务中浏览 nuScenes 数据集

```
python tools/misc/browse_dataset.py configs/_base_/datasets/nus-mono3d.py --task mono-
↪det --output-dir ${OUTPUT_DIR} --online
```



CHAPTER 30

模型部署

Note: 此工具仍然处于试验阶段，目前只有 SECOND 支持用 `TorchServe` 部署，我们将会在未来支持更多的模型。

为了使用 `TorchServe` 部署 `MMDetection3D` 模型，您可以遵循以下步骤:

30.1 1. 将模型从 MMDetection3D 转换到 TorchServe

```
python tools/deployment/mmdet3d2torchserve.py ${CONFIG_FILE} ${CHECKPOINT_FILE} \
--output-folder ${MODEL_STORE} \
--model-name ${MODEL_NAME}
```

Note: ${MODEL_STORE} 需要为文件夹的绝对路径。

30.2 2. 构建 `mmdet3d-serve` 镜像

```
docker build -t mmdet3d-serve:latest docker/serve/
```

30.3 3. 运行 mmdet3d-serve

查看官网文档来 使用 docker 运行 TorchServe。

为了在 GPU 上运行，您需要安装 nvidia-docker。您可以忽略 --gpus 参数，从而在 CPU 上运行。

例子:

```
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 \
mmdet3d-serve:latest
```

阅读文档 关于 Inference (8080), Management (8081) and Metrics (8082) 接口。

30.4 4. 测试部署

您可以使用 test_torchserver.py 进行部署，同时比较 torchserver 和 pytorch 的结果。

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

例子:

```
python tools/deployment/test_torchserver.py demo/data/kitti/kitti_000008.bin configs/
↪second/hv_second_secfpn_6x8_80e_kitti-3d-car.py checkpoints/hv_second_secfpn_6x8_
↪80e_kitti-3d-car_20200620_230238-393f000c.pth second
```

CHAPTER 31

模型复杂度

您可以使用 MMDetection 中的 `tools/analysis_tools/get_flops.py` 这个脚本文件，基于 flops-counter.pytorch 计算一个给定模型的计算量 (FLOPS) 和参数量 (params)。

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

您将会得到如下的结果：

```
==============================
Input shape: (4000, 4)
Flops: 5.78 GFLOPs
Params: 953.83 k
==============================
```

注意: 此工具仍然处于试验阶段，我们不能保证数值是绝对正确的。您可以将结果用于简单的比较，但在写技术文档报告或者论文之前您需要再次确认一下。

1. 计算量 (FLOPs) 和输入形状有关，但是参数量 (params) 则和输入形状无关。默认的输入形状为 (1, 40000, 4)。
2. 一些运算操作不计入计算量 (FLOPs)，比如说像 GN 和定制的运算操作，详细细节请参考 `mmcv.cnn.get_model_complexity_info()`。
3. 我们现在仅仅支持单模态输入（点云或者图片）的单阶段模型的计算量 (FLOPs) 计算，我们将会在未来支持两阶段和多模态模型的计算。

CHAPTER 32

模型转换

32.1 RegNet 模型转换到 MMDetection

`tools/model_converters/regnet2mmdet.py` 将 pycls 预训练 RegNet 模型中的键转换为 MMDetection 风格。

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

32.2 Detectron ResNet 转换到 Pytorch

MMDetection 中的 `tools/detectron2pytorch.py` 能够把原始的 detectron 中预训练的 ResNet 模型的键转换为 PyTorch 风格。

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

32.3 准备要发布的模型

`tools/model_converters/publish_model.py` 帮助用户准备他们用于发布的模型。

在您上传一个模型到云服务器 (AWS) 之前，您需要做以下几步：

1. 将模型权重转换为 CPU 张量
2. 删除记录优化器状态 (optimizer states) 的相关信息
3. 计算检查点 (checkpoint) 文件的哈希编码 (hash id) 并且把哈希编码加到文件名里

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

例如，

```
python tools/model_converters/publish_model.py work_dirs/faster_rcnn/latest.pth␣
↪faster_rcnn_r50_fpn_1x_20190801.pth
```

最终的输出文件名将会是 `faster_rcnn_r50_fpn_1x_20190801-{hash id}.pth`。

CHAPTER 33

数据集转换

tools/data_converter/ 包含转换数据集为其他格式的一些工具。其中大多数转换数据集为基于 pickle 的信息文件，比如 KITTI，nuscense 和 lyft。Waymo 转换器被用来重新组织 waymo 原始数据为 KITTI 风格。用户能够参考它们了解我们转换数据格式的方法。将它们修改为 nuImages 转换器等脚本也很方便。

为了转换 nuImages 数据集为 COCO 格式，请使用下面的指令：

```
python -u tools/data_converter/nuimage_converter.py --data-root ${DATA_ROOT} --
↪version ${VERSIONS} \
                                                    --out-dir ${OUT_DIR} --nproc $
↪{NUM_WORKERS} --extra-tag ${TAG}
```

- --data-root: 数据集的根目录，默认为 ./data/nuimages。
- --version: 数据集的版本，默认为 v1.0-mini。要获取完整数据集，请使用 --version v1.0-train v1.0-val v1.0-mini。
- --out-dir: 注释和语义掩码的输出目录，默认为 ./data/nuimages/annotations/。
- --nproc: 数据准备的进程数，默认为 4。由于图片是并行处理的，更大的进程数目能够减少准备时间。
- --extra-tag: 注释的额外标签，默认为 nuimages。这可用于将不同时间处理的不同注释分开以供研究。

更多的数据准备细节参考 doc，nuImages 数据集的细节参考 README。

CHAPTER 34

其他内容

34.1 打印完整的配置文件

tools/misc/print_config.py 逐字打印整个配置文件，展开所有的导入。

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

CHAPTER 35

基准测试

这里我们对 MMDetection3D 和其他开源 3D 目标检测代码库中模型的训练速度和测试速度进行了基准测试。

35.1 配置

- 硬件：8 NVIDIA Tesla V100 (32G) GPUs, Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz
- 软件：Python 3.7, CUDA 10.1, cuDNN 7.6.5, PyTorch 1.3, numba 0.48.0.
- 模型：由于不同代码库所实现的模型种类有所不同，在基准测试中我们选择了 SECOND、PointPillars、Part-A2 和 VoteNet 几种模型，分别与其他代码库中的相应模型实现进行了对比。
- 度量方法：我们使用整个训练过程中的平均吞吐量作为度量方法，并跳过每个 epoch 的前 50 次迭代以消除训练预热的影响。

35.2 主要结果

对于模型的训练速度（样本/秒），我们将 MMDetection3D 与其他实现了相同模型的代码库进行了对比。结果如下所示，表格内的数字越大，代表模型的训练速度越快。代码库中不支持的模型使用 × 进行标识。

35.3 测试细节

35.3.1 为了计算速度所做的修改

- **MMDetection3D**：我们尝试使用与其他代码库中尽可能相同的配置，具体配置细节见 基准测试配置。
- **Det3D**：为了与 Det3D 进行比较，我们使用了 commit 519251e 所对应的代码版本。
- **OpenPCDet**：为了与 OpenPCDet 进行比较，我们使用了 commit b32fbddb 所对应的代码版本。

  为了计算训练速度，我们在 `./tools/train_utils/train_utils.py` 文件中添加了用于记录运行时间的代码。我们对每个 epoch 的训练速度进行计算，并报告所有 epoch 的平均速度。

```
diff --git a/tools/train_utils/train_utils.py b/tools/train_utils/train_utils.py
index 91f21dd..021359d 100644
--- a/tools/train_utils/train_utils.py
+++ b/tools/train_utils/train_utils.py
@@ -2,6 +2,7 @@ import torch
 import os
 import glob
 import tqdm
+import datetime
 from torch.nn.utils import clip_grad_norm_


@@ -13,7 +14,10 @@ def train_one_epoch(model, optimizer, train_loader, model_func,
↪ lr_scheduler, ac
     if rank == 0:
         pbar = tqdm.tqdm(total=total_it_each_epoch, leave=leave_pbar, desc='train
↪', dynamic_ncols=True)

+    start_time = None
     for cur_it in range(total_it_each_epoch):
+        if cur_it > 49 and start_time is None:
+            start_time = datetime.datetime.now()
         try:
             batch = next(dataloader_iter)
         except StopIteration:
@@ -55,9 +59,11 @@ def train_one_epoch(model, optimizer, train_loader, model_func,
↪ lr_scheduler, ac
                 tb_log.add_scalar('learning_rate', cur_lr, accumulated_iter)
                 for key, val in tb_dict.items():
                     tb_log.add_scalar('train_' + key, val, accumulated_iter)
+    endtime = datetime.datetime.now()
+    speed = (endtime - start_time).seconds / (total_it_each_epoch - 50)
```

(下页继续)

(续上页)

```
     if rank == 0:
         pbar.close()
-    return accumulated_iter
+    return accumulated_iter, speed


 def train_model(model, optimizer, train_loader, model_func, lr_scheduler, optim_
↪cfg,
@@ -65,6 +71,7 @@ def train_model(model, optimizer, train_loader, model_func, lr_
↪scheduler, optim_
                 lr_warmup_scheduler=None, ckpt_save_interval=1, max_ckpt_save_
↪num=50,
                 merge_all_iters_to_one_epoch=False):
     accumulated_iter = start_iter
+    speeds = []
     with tqdm.trange(start_epoch, total_epochs, desc='epochs', dynamic_
↪ncols=True, leave=(rank == 0)) as tbar:
         total_it_each_epoch = len(train_loader)
         if merge_all_iters_to_one_epoch:
@@ -82,7 +89,7 @@ def train_model(model, optimizer, train_loader, model_func, lr_
↪scheduler, optim_
                 cur_scheduler = lr_warmup_scheduler
             else:
                 cur_scheduler = lr_scheduler
-            accumulated_iter = train_one_epoch(
+            accumulated_iter, speed = train_one_epoch(
                 model, optimizer, train_loader, model_func,
                 lr_scheduler=cur_scheduler,
                 accumulated_iter=accumulated_iter, optim_cfg=optim_cfg,
@@ -91,7 +98,7 @@ def train_model(model, optimizer, train_loader, model_func, lr_
↪scheduler, optim_
                 total_it_each_epoch=total_it_each_epoch,
                 dataloader_iter=dataloader_iter
             )
-
+            speeds.append(speed)
             # save trained model
             trained_epoch = cur_epoch + 1
             if trained_epoch % ckpt_save_interval == 0 and rank == 0:
@@ -107,6 +114,8 @@ def train_model(model, optimizer, train_loader, model_func,␣
↪lr_scheduler, optim_
                 save_checkpoint(
                     checkpoint_state(model, optimizer, trained_epoch,␣
↪accumulated_iter), filename=ckpt_name,
```

(下页继续)

(续上页)

```
                    )
+               print(speed)
+    print(f'*******{sum(speeds) / len(speeds)}******')


 def model_state_to_cpu(model_state):
```

35.3.2 VoteNet

- **MMDetection3D**：在 v0.1.0 版本下，执行如下命令：

```
./tools/dist_train.sh configs/votenet/votenet_16x8_sunrgbd-3d-10class.py 8 --no-
↪validate
```

- **votenet**：在 commit 2f6d6d3 版本下，执行如下命令：

```
python train.py --dataset sunrgbd --batch_size 16
```

然后执行如下命令，对测试速度进行评估：

```
python eval.py --dataset sunrgbd --checkpoint_path log_sunrgbd/checkpoint.tar --
↪batch_size 1 --dump_dir eval_sunrgbd --cluster_sampling seed_fps --use_3d_nms --
↪use_cls_nms --per_class_proposal
```

注意，为了计算推理速度，我们对 `eval.py` 进行了修改。

```
diff --git a/eval.py b/eval.py
  index c0b2886..04921e9 100644
  --- a/eval.py
  +++ b/eval.py
  @@ -10,6 +10,7 @@ import os
   import sys
   import numpy as np
   from datetime import datetime
  +import time
   import argparse
   import importlib
   import torch
  @@ -28,7 +29,7 @@ parser.add_argument('--checkpoint_path', default=None, help=
↪'Model checkpoint pa
   parser.add_argument('--dump_dir', default=None, help='Dump dir to save sample␣
↪outputs [default: None]')
   parser.add_argument('--num_point', type=int, default=20000, help='Point Number␣
↪[default: 20000]')
```

(下页继续)

(续上页)

```
   parser.add_argument('--num_target', type=int, default=256, help='Point Number␣
↪[default: 256]')
  -parser.add_argument('--batch_size', type=int, default=8, help='Batch Size␣
↪during training [default: 8]')
  +parser.add_argument('--batch_size', type=int, default=1, help='Batch Size␣
↪during training [default: 8]')
   parser.add_argument('--vote_factor', type=int, default=1, help='Number of␣
↪votes generated from each seed [default: 1]')
   parser.add_argument('--cluster_sampling', default='vote_fps', help='Sampling␣
↪strategy for vote clusters: vote_fps, seed_fps, random [default: vote_fps]')
   parser.add_argument('--ap_iou_thresholds', default='0.25,0.5', help='A list of␣
↪AP IoU thresholds [default: 0.25,0.5]')
  @@ -132,6 +133,7 @@ CONFIG_DICT = {'remove_empty_box': (not FLAGS.faster_eval),
↪'use_3d_nms': FLAGS.
   # ------------------------------------------------------------------␣
↪GLOBAL CONFIG END

   def evaluate_one_epoch():
  +    time_list = list()
       stat_dict = {}
       ap_calculator_list = [APCalculator(iou_thresh, DATASET_CONFIG.class2type) \
           for iou_thresh in AP_IOU_THRESHOLDS]
  @@ -144,6 +146,8 @@ def evaluate_one_epoch():

           # Forward pass
           inputs = {'point_clouds': batch_data_label['point_clouds']}
  +        torch.cuda.synchronize()
  +        start_time = time.perf_counter()
           with torch.no_grad():
               end_points = net(inputs)

  @@ -161,6 +165,12 @@ def evaluate_one_epoch():

           batch_pred_map_cls = parse_predictions(end_points, CONFIG_DICT)
           batch_gt_map_cls = parse_groundtruths(end_points, CONFIG_DICT)
  +        torch.cuda.synchronize()
  +        elapsed = time.perf_counter() - start_time
  +        time_list.append(elapsed)
  +
  +        if len(time_list==200):
  +            print("average inference time: %4f"%(sum(time_list[5:])/len(time_
↪list[5:])))
           for ap_calculator in ap_calculator_list:
               ap_calculator.step(batch_pred_map_cls, batch_gt_map_cls)
```

35.3.3 PointPillars-car

- **MMDetection3D**：在 v0.1.0 版本下，执行如下命令：

```
./tools/dist_train.sh configs/benchmark/hv_pointpillars_secfpn_3x8_100e_det3d_
↪kitti-3d-car.py 8 --no-validate
```

- **Det3D**：在 commit 519251e 版本下，使用 `kitti_point_pillars_mghead_syncbn.py` 并执行如下命令：

```
./tools/scripts/train.sh --launcher=slurm --gpus=8
```

注意，为了训练 PointPillars，我们对 `train.sh` 进行了修改。

```
diff --git a/tools/scripts/train.sh b/tools/scripts/train.sh
index 3a93f95..461e0ea 100755
--- a/tools/scripts/train.sh
+++ b/tools/scripts/train.sh
@@ -16,9 +16,9 @@ then
 fi

 # Voxelnet
-python -m torch.distributed.launch --nproc_per_node=8 ./tools/train.py examples/
↪second/configs/  kitti_car_vfev3_spmiddlefhd_rpn1_mghead_syncbn.py --work_dir=
↪$SECOND_WORK_DIR
+# python -m torch.distributed.launch --nproc_per_node=8 ./tools/train.py␣
↪examples/second/configs/  kitti_car_vfev3_spmiddlefhd_rpn1_mghead_syncbn.py --
↪work_dir=$SECOND_WORK_DIR
 # python -m torch.distributed.launch --nproc_per_node=8 ./tools/train.py␣
↪examples/cbgs/configs/  nusc_all_vfev3_spmiddleresnetfhd_rpn2_mghead_syncbn.py -
↪-work_dir=$NUSC_CBGS_WORK_DIR
 # python -m torch.distributed.launch --nproc_per_node=8 ./tools/train.py␣
↪examples/second/configs/  lyft_all_vfev3_spmiddleresnetfhd_rpn2_mghead_syncbn.
↪py --work_dir=$LYFT_CBGS_WORK_DIR

 # PointPillars
-# python -m torch.distributed.launch --nproc_per_node=8 ./tools/train.py ./
↪examples/point_pillars/configs/  original_pp_mghead_syncbn_kitti.py --work_dir=
↪$PP_WORK_DIR
+python -m torch.distributed.launch --nproc_per_node=8 ./tools/train.py ./
↪examples/point_pillars/configs/  kitti_point_pillars_mghead_syncbn.py
```

35.3.4 PointPillars-3class

- **MMDetection3D**：在 v0.1.0 版本下，执行如下命令：

```
./tools/dist_train.sh configs/benchmark/hv_pointpillars_secfpn_4x8_80e_pcdet_
↪kitti-3d-3class.py 8 --no-validate
```

- **OpenPCDet**：在 commit b32fbddb 版本下，执行如下命令：

```
cd tools
sh scripts/slurm_train.sh ${PARTITION} ${JOB_NAME} 8  --cfg_file ./cfgs/kitti_
↪models/pointpillar.yaml --batch_size 32  --workers 32 --epochs 80
```

35.3.5 SECOND

基准测试中的 SECOND 指在 second.Pytorch 首次被实现的 SECONDv1.5。Det3D 实现的 SECOND 中，使用了自己实现的 Multi-Group Head，因此无法将它的速度与其他代码库进行对比。

- **MMDetection3D**：在 v0.1.0 版本下，执行如下命令：

```
./tools/dist_train.sh configs/benchmark/hv_second_secfpn_4x8_80e_pcdet_kitti-3d-
↪3class.py 8 --no-validate
```

- **OpenPCDet**：在 commit b32fbddb 版本下，执行如下命令：

```
cd tools
sh ./scripts/slurm_train.sh ${PARTITION} ${JOB_NAME} 8  --cfg_file ./cfgs/kitti_
↪models/second.yaml --batch_size 32  --workers 32 --epochs 80
```

35.3.6 Part-A2

- **MMDetection3D**：在 v0.1.0 版本下，执行如下命令：

```
./tools/dist_train.sh configs/benchmark/hv_PartA2_secfpn_4x8_cyclic_80e_pcdet_
↪kitti-3d-3class.py 8 --no-validate
```

- **OpenPCDet**：在 commit b32fbddb 版本下，执行如下命令以进行模型训练：

```
cd tools
sh ./scripts/slurm_train.sh ${PARTITION} ${JOB_NAME} 8  --cfg_file ./cfgs/kitti_
↪models/PartA2.yaml --batch_size 32 --workers 32 --epochs 80
```

CHAPTER 36

常见问题解答

我们列出了一些用户和开发者在开发过程中会遇到的常见问题以及对应的解决方案，如果您发现了任何频繁出现的问题，请随时扩充本列表，非常欢迎您提出的任何解决方案。如果您在环境配置、模型训练等工作中遇到任何的问题，请使用问题模板来创建相应的 issue, 并将所需的所有信息填入到问题模板中，我们会尽快解决您的问题。

36.1 MMCV/MMDet/MMDet3D Installation

- 跟 MMCV, MMDetection, MMSegmentation 和 MMDetection3D 相关的编译问题; “ConvWS is already registered in conv layer”； “AssertionError: MMCV==xxx is used but incompatible. Please install mmcv>=xxx, <=xxx.”

MMDetection3D 需要的 MMCV, MMDetection 和 MMSegmentation 的版本列在了下面。请安装正确版本的 MMCV、MMDetection 和 MMSegmentation 以避免相关的安装问题。

- 如果您在 `import open3d` 时遇到下面的问题:

  ```
  OSError: /lib/x86_64-linux-gnu/libm.so.6: version 'GLIBC_2.27' not found
  ```

  请将 open3d 的版本降级至 0.9.0.0，因为最新版 open3d 需要‘GLIBC_2.27’文件的支持，Ubuntu 16.04 系统中缺失该文件，且该文件仅存在于 Ubuntu 18.04 及之后的系统中。

- 如果您在 `import pycocotools` 时遇到版本错误的问题，这是由于 nuscenes-devkit 需要安装 pycocotools，然而 mmdet 依赖于 mmpycocotools，当前的解决方案如下所示，我们将会在之后全面支持 pycocotools：

```
pip uninstall pycocotools mmpycocotools
pip install mmpycocotools
```

注意：我们已经在 0.13.0 及之后的版本中全面支持 pycocotools。

- 如果您在导入 pycocotools 相关包时遇到下面的问题：

  ```
  ValueError: numpy.ndarray size changed, may indicate binary
  incompatibility. Expected 88 from C header, got 80 from PyObject
  ```

  请将 pycocotools 的版本降级至 2.0.1，这是由于最新版本的 pycocotools 与 numpy < 1.20.0 不兼容。或者通过下面的方式从源码进行编译来安装最新版本的 pycocotools：

  ```
  pip install -e "git+https://github.com/cocodataset/cocoapi#egg=pycocotools&subdirectory
  ```

  或者

  ```
  pip install -e "git+https://github.com/ppwwyyxx/cocoapi#egg=pycocotools&subdirectory=P
  ```

36.2 如何标注点云？

MMDetection3D 不支持点云标注。我们提供一些开源的标注工具供参考：

- SUSTechPOINTS
- LATTE

此外，我们改进了 LATTE 以便更方便的标注。更多的细节请参考 这里。

CHAPTER 37

0.16.0

CHAPTER 38

mmdet3d.core

38.1 anchor

class mmdet3d.core.anchor.**AlignedAnchor3DRangeGenerator**(*align_corner=False*, ***kwargs*)

Aligned 3D Anchor Generator by range.

This anchor generator uses a different manner to generate the positions of anchors' centers from *Anchor3DRangeGenerator*.

注解： The *align* means that the anchor's center is aligned with the voxel grid, which is also the feature grid. The previous implementation of *Anchor3DRangeGenerator* does not generate the anchors' center according to the voxel grid. Rather, it generates the center by uniformly distributing the anchors inside the minimum and maximum anchor ranges according to the feature map sizes. However, this makes the anchors center does not match the feature grid. The *AlignedAnchor3DRangeGenerator* add + 1 when using the feature map sizes to obtain the corners of the voxel grid. Then it shifts the coordinates to the center of voxel grid and use the left up corner to distribute anchors.

参数 **anchor_corner** (*bool, optional*) – Whether to align with the corner of the voxel grid. By default it is False and the anchor's center will be the same as the corresponding voxel's center, which is also the center of the corresponding greature grid. Defaults to False.

anchors_single_range(*feature_size*, *anchor_range*, *scale*, *sizes=[[3.9, 1.6, 1.56]]*, *rotations=[0, 1.5707963]*, *device='cuda'*)

Generate anchors in a single range.

参数

- **feature_size** (*list[float] | tuple[float]*) –Feature map size. It is either a list of a tuple of [D, H, W](in order of z, y, and x).
- **anchor_range** (*torch.Tensor | list[float]*) –Range of anchors with shape [6]. The order is consistent with that of anchors, i.e., (x_min, y_min, z_min, x_max, y_max, z_max).
- **scale** (*float | int*) –The scale factor of anchors.
- **sizes** (*list[list] | np.ndarray | torch.Tensor, optional*) –Anchor size with shape [N, 3], in order of x, y, z. Defaults to [[3.9, 1.6, 1.56]].
- **rotations** (*list[float] | np.ndarray | torch.Tensor, optional*) –Rotations of anchors in a single feature grid. Defaults to [0, 1.5707963].
- **device** (*str, optional*) –Devices that the anchors will be put on. Defaults to 'cuda'.

返回

Anchors with shape [*feature_size, num_sizes, num_rots, 7].

返回类型 torch.Tensor

class mmdet3d.core.anchor.**AlignedAnchor3DRangeGeneratorPerCls**(***kwargs*)

3D Anchor Generator by range for per class.

This anchor generator generates anchors by the given range for per class. Note that feature maps of different classes may be different.

参数 **kwargs** (*dict*) –Arguments are the same as those in *AlignedAnchor3DRangeGenerator*.

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

Generate grid anchors in multiple feature levels.

参数

- **featmap_sizes** (*list[tuple]*) –List of feature map sizes for different classes in a single feature level.
- **device** (*str, optional*) –Device where the anchors will be put on. Defaults to 'cuda'.

返回

Anchors in multiple feature levels. Note that in this anchor generator, we currently only support single feature level. The sizes of each tensor should be [num_sizes/ranges*num_rots*featmap_size, box_code_size].

返回类型 list[list[torch.Tensor]]

multi_cls_grid_anchors(*featmap_sizes*, *scale*, *device='cuda'*)

Generate grid anchors of a single level feature map for multi-class with different feature map sizes.

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

参数

- **featmap_sizes** (*list[tuple]*) –List of feature map sizes for different classes in a single feature level.
- **scale** (*float*) –Scale factor of the anchors in the current level.
- **device** (*str, optional*) –Device the tensor will be put on. Defaults to 'cuda'.

返回 Anchors in the overall feature map.

返回类型 torch.Tensor

class mmdet3d.core.anchor.**Anchor3DRangeGenerator**(*ranges*, *sizes=[[3.9, 1.6, 1.56]]*, *scales=[1]*, *rotations=[0, 1.5707963]*, *custom_values=()*, *reshape_out=True*, *size_per_range=True*)

3D Anchor Generator by range.

This anchor generator generates anchors by the given range in different feature levels. Due the convention in 3D detection, different anchor sizes are related to different ranges for different categories. However we find this setting does not effect the performance much in some datasets, e.g., nuScenes.

参数

- **ranges** (*list[list[float]]*) –Ranges of different anchors. The ranges are the same across different feature levels. But may vary for different anchor sizes if size_per_range is True.
- **sizes** (*list[list[float]], optional*) –3D sizes of anchors. Defaults to [[3.9, 1.6, 1.56]].
- **scales** (*list[int], optional*) –Scales of anchors in different feature levels. Defaults to [1].
- **rotations** (*list[float], optional*) –Rotations of anchors in a feature grid. Defaults to [0, 1.5707963].
- **custom_values** (*tuple[float], optional*) –Customized values of that anchor. For example, in nuScenes the anchors have velocities. Defaults to ().

- **reshape_out** (*bool, optional*) –Whether to reshape the output into (N x 4). Defaults to True.
- **size_per_range** (*bool, optional*) –Whether to use separate ranges for different sizes. If size_per_range is True, the ranges should have the same length as the sizes, if not, it will be duplicated. Defaults to True.

anchors_single_range (*feature_size*, *anchor_range*, *scale=1*, *sizes=[[3.9, 1.6, 1.56]]*, *rotations=[0, 1.5707963]*, *device='cuda'*)

Generate anchors in a single range.

参数

- **feature_size** (*list[float] | tuple[float]*) –Feature map size. It is either a list of a tuple of [D, H, W](in order of z, y, and x).
- **anchor_range** (*torch.Tensor | list[float]*) –Range of anchors with shape [6]. The order is consistent with that of anchors, i.e., (x_min, y_min, z_min, x_max, y_max, z_max).
- **scale** (*float | int, optional*) –The scale factor of anchors. Defaults to 1.
- **sizes** (*list[list] | np.ndarray | torch.Tensor, optional*) –Anchor size with shape [N, 3], in order of x, y, z. Defaults to [[3.9, 1.6, 1.56]].
- **rotations** (*list[float] | np.ndarray | torch.Tensor, optional*) –Rotations of anchors in a single feature grid. Defaults to [0, 1.5707963].
- **device** (*str*) –Devices that the anchors will be put on. Defaults to 'cuda'.

返回

Anchors with shape [*feature_size, num_sizes, num_rots, 7].

返回类型 torch.Tensor

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

Generate grid anchors in multiple feature levels.

参数

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

返回

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

返回类型 list[torch.Tensor]

property num_base_anchors

Total number of base anchors in a feature grid.

Type list[int]

property num_levels

Number of feature levels that the generator is applied to.

Type int

single_level_grid_anchors(*featmap_size*, *scale*, *device='cuda'*)

Generate grid anchors of a single level feature map.

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

参数

- **featmap_size** (*tuple[int]*) –Size of the feature map.
- **scale** (*float*) –Scale factor of the anchors in the current level.
- **device** (*str, optional*) –Device the tensor will be put on. Defaults to 'cuda'.

返回 Anchors in the overall feature map.

返回类型 torch.Tensor

38.2 bbox

class mmdet3d.core.bbox.**AssignResult**(*num_gts*, *gt_inds*, *max_overlaps*, *labels=None*)

Stores assignments between predicted and truth boxes.

num_gts

the number of truth boxes considered when computing this assignment

Type int

gt_inds

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

Type LongTensor

max_overlaps

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

Type FloatTensor

labels

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

Type None | LongTensor

示例

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

add_gt_(*gt_labels*)

Add ground truth as assigned results.

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

get_extra_property(*key*)

Get user-defined property.

property info

a dictionary of info about the object

Type dict

property num_preds

the number of predictions in this assignment

Type int

classmethod random(***kwargs*)

Create random AssignResult for tests or debugging.

参数

- **num_preds** –number of predicted boxes
- **num_gts** –number of true boxes
- **p_ignore** (*float*) –probability of a predicted box assigned to an ignored truth
- **p_assigned** (*float*) –probability of a predicted box not being assigned
- **p_use_label** (*float | bool*) –with labels or not

- **rng** (*None | int | numpy.random.RandomState*) –seed or state

返回 Randomly generated assign results.

返回类型 *AssignResult*

示例

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

set_extra_property(*key*, *value*)

Set user-defined new property.

class mmdet3d.core.bbox.**AxisAlignedBboxOverlaps3D**

Axis-aligned 3D Overlaps (IoU) Calculator.

class mmdet3d.core.bbox.**BaseAssigner**

Base assigner that assigns boxes to ground truth boxes.

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

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

class mmdet3d.core.bbox.**BaseInstance3DBoxes**(*tensor*, *box_dim=7*, *with_yaw=True*, *origin=(0.5, 0.5, 0)*)

Base class for 3D Boxes.

注解: The box is bottom centered, i.e. the relative position of origin in the box is (0.5, 0.5, 0).

参数

- **tensor** (*torch.Tensor | np.ndarray | list*) –a N x box_dim matrix.
- **box_dim** (*int*) –Number of the dimension of a box. Each row is (x, y, z, x_size, y_size, z_size, yaw). Defaults to 7.
- **with_yaw** (*bool*) –Whether the box is with yaw rotation. If False, the value of yaw will be set to 0 as minmax boxes. Defaults to True.
- **origin** (*tuple[float], optional*) –Relative position of the box origin. Defaults to (0.5, 0.5, 0). This will guide the box be converted to (0.5, 0.5, 0) mode.

tensor

Float matrix of N x box_dim.

Type torch.Tensor

box_dim

Integer indicating the dimension of a box. Each row is (x, y, z, x_size, y_size, z_size, yaw, …).

Type int

with_yaw

If True, the value of yaw will be set to 0 as minmax boxes.

Type bool

property bev

2D BEV box of each box with rotation in XYWHR format, in shape (N, 5).

Type torch.Tensor

property bottom_center

A tensor with center of each box in shape (N, 3).

Type torch.Tensor

property bottom_height

torch.Tensor: A vector with bottom's height of each box in shape (N,).

classmethod cat(*boxes_list*)

Concatenate a list of Boxes into a single Boxes.

参数 boxes_list (list[*BaseInstance3DBoxes*]) –List of boxes.

返回 The concatenated Boxes.

返回类型 *BaseInstance3DBoxes*

property center

Calculate the center of all the boxes.

注解： In MMDetection3D's convention, the bottom center is usually taken as the default center.

The relative position of the centers in different kinds of boxes are different, e.g., the relative center of a boxes is (0.5, 1.0, 0.5) in camera and (0.5, 0.5, 0) in lidar. It is recommended to use `bottom_center` or `gravity_center` for clearer usage.

返回 A tensor with center of each box in shape (N, 3).

返回类型 torch.Tensor

clone()

Clone the Boxes.

返回

Box object with the same properties as self.

返回类型 *BaseInstance3DBoxes*

abstract convert_to(*dst*, *rt_mat=None*)

Convert self to `dst` mode.

参数

- **dst** (*Box3DMode*) –The target Box mode.
- **rt_mat** (*np.ndarray | torch.Tensor, optional*) –The rotation and translation matrix between different coordinates. Defaults to None. The conversion from *src* coordinates to *dst* coordinates usually comes along the change of sensors, e.g., from camera to LiDAR. This requires a transformation matrix.

返回

The converted box of the same type in the *dst* mode.

返回类型 *BaseInstance3DBoxes*

property corners

torch.Tensor: a tensor with 8 corners of each box in shape (N, 8, 3).

property device

The device of the boxes are on.

Type str

property dims

Size dimensions of each box in shape (N, 3).

Type torch.Tensor

abstract flip(*bev_direction='horizontal'*)

Flip the boxes in BEV along given BEV direction.

参数 **bev_direction** (*str, optional*) –Direction by which to flip. Can be chosen from 'horizontal' and 'vertical'. Defaults to 'horizontal'.

property gravity_center

A tensor with center of each box in shape (N, 3).

Type torch.Tensor

property height

A vector with height of each box in shape (N,).

Type torch.Tensor

classmethod height_overlaps(*boxes1*, *boxes2*, *mode='iou'*)

Calculate height overlaps of two boxes.

注解： This function calculates the height overlaps between boxes1 and boxes2, boxes1 and boxes2 should be in the same type.

参数

- **boxes1** (*BaseInstance3DBoxes*) –Boxes 1 contain N boxes.
- **boxes2** (*BaseInstance3DBoxes*) –Boxes 2 contain M boxes.
- **mode** (*str, optional*) –Mode of IoU calculation. Defaults to 'iou'.

返回 Calculated iou of boxes.

返回类型 torch.Tensor

in_range_3d(*box_range*)

Check whether the boxes are in the given range.

参数 box_range (*list | torch.Tensor*) –The range of box (x_min, y_min, z_min, x_max, y_max, z_max)

注解： In the original implementation of SECOND, checking whether a box in the range checks whether the points are in a convex polygon, we try to reduce the burden for simpler cases.

返回

A binary vector indicating whether each box is inside the reference range.

返回类型 torch.Tensor

in_range_bev(*box_range*)

Check whether the boxes are in the given range.

参数 box_range (*list | torch.Tensor*) –the range of box (x_min, y_min, x_max, y_max)

注解： The original implementation of SECOND checks whether boxes in a range by checking whether the points are in a convex polygon, we reduce the burden for simpler cases.

返回 Whether each box is inside the reference range.

返回类型 torch.Tensor

limit_yaw(*offset=0.5*, *period=3.141592653589793*)

Limit the yaw to a given period and offset.

参数

- **offset** (*float, optional*) –The offset of the yaw. Defaults to 0.5.
- **period** (*float, optional*) –The expected period. Defaults to np.pi.

property nearest_bev

A tensor of 2D BEV box of each box without rotation.

Type torch.Tensor

new_box(*data*)

Create a new box object with data.

The new box and its tensor has the similar properties as self and self.tensor, respectively.

参数 **data** (*torch.Tensor | numpy.array | list*) –Data to be copied.

返回

A new bbox object with data, the object's other properties are similar to `self`.

返回类型 *BaseInstance3DBoxes*

nonempty(*threshold=0.0*)

Find boxes that are non-empty.

A box is considered empty, if either of its side is no larger than threshold.

参数 **threshold** (*float, optional*) –The threshold of minimal sizes. Defaults to 0.0.

返回

A binary vector which represents whether each box is empty (False) or non-empty (True).

返回类型 torch.Tensor

classmethod overlaps(*boxes1*, *boxes2*, *mode='iou'*)

Calculate 3D overlaps of two boxes.

注解： This function calculates the overlaps between `boxes1` and `boxes2`, `boxes1` and `boxes2` should be in the same type.

参数

- **boxes1** (*BaseInstance3DBoxes*) –Boxes 1 contain N boxes.
- **boxes2** (*BaseInstance3DBoxes*) –Boxes 2 contain M boxes.

- **mode** (*str, optional*) –Mode of iou calculation. Defaults to ‘iou’.

返回 Calculated 3D overlaps of the boxes.

返回类型 torch.Tensor

points_in_boxes_all(*points*, *boxes_override=None*)

Find all boxes in which each point is.

参数

- **points** (*torch.Tensor*) –Points in shape (1, M, 3) or (M, 3), 3 dimensions are (x, y, z) in LiDAR or depth coordinate.
- **boxes_override** (*torch.Tensor, optional*) –Boxes to override *self.tensor*. Defaults to None.

返回

A tensor indicating whether a point is in a box, in shape (M, T). T is the number of boxes. Denote this tensor as A, if the m^th point is in the t^th box, then *A[m, t] == 1*, elsewise *A[m, t] == 0*.

返回类型 torch.Tensor

points_in_boxes_part(*points*, *boxes_override=None*)

Find the box in which each point is.

参数

- **points** (*torch.Tensor*) –Points in shape (1, M, 3) or (M, 3), 3 dimensions are (x, y, z) in LiDAR or depth coordinate.
- **boxes_override** (*torch.Tensor, optional*) –Boxes to override *self.tensor*. Defaults to None.

返回

The index of the first box that each point is in, in shape (M,). Default value is -1 (if the point is not enclosed by any box).

返回类型 torch.Tensor

注解： If a point is enclosed by multiple boxes, the index of the first box will be returned.

---

abstract rotate(*angle*, *points=None*)

Rotate boxes with points (optional) with the given angle or rotation matrix.

参数

- **angle** (*float | torch.Tensor | np.ndarray*) –Rotation angle or rotation matrix.
- **(torch.Tensor | numpy.ndarray |** (*points*) –BasePoints, optional): Points to rotate. Defaults to None.

scale(*scale_factor*)

Scale the box with horizontal and vertical scaling factors.

参数 **scale_factors** (*float*) –Scale factors to scale the boxes.

to(*device*)

Convert current boxes to a specific device.

参数 **device** (str | torch.device) –The name of the device.

返回

A new boxes object on the specific device.

返回类型 *BaseInstance3DBoxes*

property top_height

torch.Tensor: A vector with the top height of each box in shape (N,).

translate(*trans_vector*)

Translate boxes with the given translation vector.

参数 **trans_vector** (*torch.Tensor*) –Translation vector of size (1, 3).

property volume

A vector with volume of each box.

Type torch.Tensor

property yaw

A vector with yaw of each box in shape (N,).

Type torch.Tensor

class mmdet3d.core.bbox.**BaseSampler**(*num*, *pos_fraction*, *neg_pos_ub=- 1*, *add_gt_as_proposals=True*, ***kwargs*)

Base class of samplers.

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

Sample positive and negative bboxes.

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

参数

- **assign_result** (*AssignResult*) –Bbox assigning results.
- **bboxes** (*Tensor*) –Boxes to be sampled from.

- **gt_bboxes** (*Tensor*) –Ground truth bboxes.
- **gt_labels** (*Tensor, optional*) –Class labels of ground truth bboxes.

返回 Sampling result.

返回类型 *SamplingResult*

示例

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

class mmdet3d.core.bbox.**BboxOverlaps3D**(*coordinate*)

3D IoU Calculator.

参数 **coordinate** (*str*) –The coordinate system, valid options are ‘camera’, ‘lidar’, and ‘depth’.

class mmdet3d.core.bbox.**BboxOverlapsNearest3D**(*coordinate='lidar'*)

Nearest 3D IoU Calculator.

注解： This IoU calculator first finds the nearest 2D boxes in bird eye view (BEV), and then calculates the 2D IoU using `bbox_overlaps()`.

参数 **coordinate** (*str*) – ‘camera’, ‘lidar’, or ‘depth’ coordinate system.

class mmdet3d.core.bbox.**Box3DMode**(*value*)

Enum of different ways to represent a box.

Coordinates in LiDAR:

```
            up z
               ^   x front
               |  /
```

(下页继续)

(续上页)

```
               | /
left y <------ 0
```

The relative coordinate of bottom center in a LiDAR box is (0.5, 0.5, 0), and the yaw is around the z axis, thus the rotation axis=2.

Coordinates in camera:

```
        z front
       /
      /
     0 ------> x right
     |
     |
     v
down y
```

The relative coordinate of bottom center in a CAM box is [0.5, 1.0, 0.5], and the yaw is around the y axis, thus the rotation axis=1.

Coordinates in Depth mode:

```
up z
   ^   y front
   |  /
   | /
   0 ------> x right
```

The relative coordinate of bottom center in a DEPTH box is (0.5, 0.5, 0), and the yaw is around the z axis, thus the rotation axis=2.

static convert(*box*, *src*, *dst*, *rt_mat=None*, *with_yaw=True*)

Convert boxes from *src* mode to *dst* mode.

参数

- **(tuple | list | np.ndarray |** (*box*) –torch.Tensor | *BaseInstance3DBoxes*): Can be a k-tuple, k-list or an Nxk array/tensor, where k = 7.
- **src** (*Box3DMode*) –The src Box mode.
- **dst** (*Box3DMode*) –The target Box mode.
- **rt_mat** (*np.ndarray | torch.Tensor, optional*) –The rotation and translation matrix between different coordinates. Defaults to None. The conversion from *src* coordinates to *dst* coordinates usually comes along the change of sensors, e.g., from camera to LiDAR. This requires a transformation matrix.

- **with_yaw** (*bool, optional*) –If *box* is an instance of *BaseInstance3DBoxes*, whether or not it has a yaw angle. Defaults to True.

返回

(**tuple | list | np.ndarray | torch.Tensor |** *BaseInstance3DBoxes*): The converted box of the same type.

class mmdet3d.core.bbox.**CameraInstance3DBoxes**(*tensor*, *box_dim=7*, *with_yaw=True*, *origin=(0.5, 1.0, 0.5)*)

3D boxes of instances in CAM coordinates.

Coordinates in camera:

```
        z front (yaw=-0.5*pi)
       /
      /
     0 ------> x right (yaw=0)
     |
     |
     v
down y
```

The relative coordinate of bottom center in a CAM box is (0.5, 1.0, 0.5), and the yaw is around the y axis, thus the rotation axis=1. The yaw is 0 at the positive direction of x axis, and decreases from the positive direction of x to the positive direction of z.

tensor

Float matrix in shape (N, box_dim).

Type torch.Tensor

box_dim

Integer indicating the dimension of a box Each row is (x, y, z, x_size, y_size, z_size, yaw, …).

Type int

with_yaw

If True, the value of yaw will be set to 0 as axis-aligned boxes tightly enclosing the original boxes.

Type bool

property bev

2D BEV box of each box with rotation in XYWHR format, in shape (N, 5).

Type torch.Tensor

property bottom_height

torch.Tensor: A vector with bottom’s height of each box in shape (N,).

convert_to(*dst*, *rt_mat=None*)

Convert self to `dst` mode.

参数

- **dst** (*Box3DMode*) –The target Box mode.
- **rt_mat** (*np.ndarray | torch.Tensor, optional*) –The rotation and translation matrix between different coordinates. Defaults to None. The conversion from `src` coordinates to `dst` coordinates usually comes along the change of sensors, e.g., from camera to LiDAR. This requires a transformation matrix.

返回 The converted box of the same type in the `dst` mode.

返回类型 *BaseInstance3DBoxes*

property corners

Coordinates of corners of all the boxes in shape (N, 8, 3).

Convert the boxes to in clockwise order, in the form of (x0y0z0, x0y0z1, x0y1z1, x0y1z0, x1y0z0, x1y0z1, x1y1z1, x1y1z0)

```
             front z
                  /
                 /
   (x0, y0, z1) + -----------  + (x1, y0, z1)
                /|            / |
               / |           /  |
(x0, y0, z0) + ----------- +   + (x1, y1, z1)
             |  /      .   |  /
             | / origin    | /
(x0, y1, z0) + ----------- + -------> x right
             |             (x1, y1, z0)
             |
             v
        down y
```

Type torch.Tensor

flip(*bev_direction='horizontal'*, *points=None*)

Flip the boxes in BEV along given BEV direction.

In CAM coordinates, it flips the x (horizontal) or z (vertical) axis.

参数

- **bev_direction** (*str*) –Flip direction (horizontal or vertical).

- **points** (torch.Tensor | np.ndarray | BasePoints, optional) –Points to flip. Defaults to None.

返回 Flipped points.

返回类型 torch.Tensor, numpy.ndarray or None

property gravity_center

A tensor with center of each box in shape (N, 3).

Type torch.Tensor

property height

A vector with height of each box in shape (N,).

Type torch.Tensor

classmethod height_overlaps(*boxes1*, *boxes2*, *mode='iou'*)

Calculate height overlaps of two boxes.

This function calculates the height overlaps between boxes1 and boxes2, where boxes1 and boxes2 should be in the same type.

参数

- **boxes1** (*CameraInstance3DBoxes*) –Boxes 1 contain N boxes.
- **boxes2** (*CameraInstance3DBoxes*) –Boxes 2 contain M boxes.
- **mode** (*str, optional*) –Mode of iou calculation. Defaults to 'iou'.

返回 Calculated iou of boxes' heights.

返回类型 torch.Tensor

property local_yaw

torch.Tensor: A vector with local yaw of each box in shape (N,). local_yaw equals to alpha in kitti, which is commonly used in monocular 3D object detection task, so only *CameraInstance3DBoxes* has the property.

points_in_boxes_all(*points*, *boxes_override=None*)

Find all boxes in which each point is.

参数

- **points** (*torch.Tensor*) –Points in shape (1, M, 3) or (M, 3), 3 dimensions are (x, y, z) in LiDAR or depth coordinate.
- **boxes_override** (*torch.Tensor, optional*) –Boxes to override 'self.tensor '. Defaults to None.

返回

The index of all boxes in which each point is, in shape (B, M, T).

返回类型 torch.Tensor

points_in_boxes_part (*points*, *boxes_override=None*)

Find the box in which each point is.

参数

- **points** (*torch.Tensor*) –Points in shape (1, M, 3) or (M, 3), 3 dimensions are (x, y, z) in LiDAR or depth coordinate.
- **boxes_override** (*torch.Tensor, optional*) –Boxes to override ‘self.tensor ‘. Defaults to None.

返回

The index of the box in which each point is, in shape (M,). Default value is -1 (if the point is not enclosed by any box).

返回类型 torch.Tensor

rotate (*angle*, *points=None*)

Rotate boxes with points (optional) with the given angle or rotation matrix.

参数

- **angle** (*float | torch.Tensor | np.ndarray*) –Rotation angle or rotation matrix.
- **points** (torch.Tensor | np.ndarray | BasePoints, optional) –Points to rotate. Defaults to None.

返回

When points is None, the function returns None, otherwise it returns the rotated points and the rotation matrix rot_mat_T.

返回类型 tuple or None

property top_height

torch.Tensor: A vector with the top height of each box in shape (N,).

class mmdet3d.core.bbox.**CombinedSampler** (*pos_sampler*, *neg_sampler*, ***kwargs*)

A sampler that combines positive sampler and negative sampler.

class mmdet3d.core.bbox.**Coord3DMode** (*value*)

Enum of different ways to represent a box and point cloud.

Coordinates in LiDAR:

```
            up z
               ^   x front
```

(下页继续)

(续上页)

```
               |  /
               | /
left y <------ 0
```

The relative coordinate of bottom center in a LiDAR box is (0.5, 0.5, 0), and the yaw is around the z axis, thus the rotation axis=2.

Coordinates in camera:

```
        z front
       /
      /
     0 ------> x right
     |
     |
     v
down y
```

The relative coordinate of bottom center in a CAM box is [0.5, 1.0, 0.5], and the yaw is around the y axis, thus the rotation axis=1.

Coordinates in Depth mode:

```
up z
   ^   y front
   |  /
   | /
   0 ------> x right
```

The relative coordinate of bottom center in a DEPTH box is (0.5, 0.5, 0), and the yaw is around the z axis, thus the rotation axis=2.

static convert(*input*, *src*, *dst*, *rt_mat=None*, *with_yaw=True*, *is_point=True*)

Convert boxes or points from *src* mode to *dst* mode.

参数

- **(tuple | list | np.ndarray | torch.Tensor |** (*input*) – *BaseInstance3DBoxes* | BasePoints): Can be a k-tuple, k-list or an Nxk array/tensor, where k = 7.
- **src** (*Box3DMode* | *Coord3DMode*) –The source mode.
- **dst** (*Box3DMode* | *Coord3DMode*) –The target mode.
- **rt_mat** (*np.ndarray | torch.Tensor, optional*) –The rotation and translation matrix between different coordinates. Defaults to None. The conversion from *src*

coordinates to *dst* coordinates usually comes along the change of sensors, e.g., from camera to LiDAR. This requires a transformation matrix.

- **with_yaw** (*bool*) –If *box* is an instance of *BaseInstance3DBoxes*, whether or not it has a yaw angle. Defaults to True.
- **is_point** (*bool*) –If *input* is neither an instance of *BaseInstance3DBoxes* nor an instance of BasePoints, whether or not it is point data. Defaults to True.

返回

(tuple | list | np.ndarray | torch.Tensor | *BaseInstance3DBoxes* **|** BasePoints): The converted box of the same type.

static convert_box(*box*, *src*, *dst*, *rt_mat=None*, *with_yaw=True*)

Convert boxes from *src* mode to *dst* mode.

参数

- **(tuple | list | np.ndarray |** (*box*) –torch.Tensor | *BaseInstance3DBoxes*): Can be a k-tuple, k-list or an Nxk array/tensor, where k = 7.
- **src** (*Box3DMode*) –The src Box mode.
- **dst** (*Box3DMode*) –The target Box mode.
- **rt_mat** (*np.ndarray | torch.Tensor, optional*) –The rotation and translation matrix between different coordinates. Defaults to None. The conversion from *src* coordinates to *dst* coordinates usually comes along the change of sensors, e.g., from camera to LiDAR. This requires a transformation matrix.
- **with_yaw** (*bool*) –If *box* is an instance of *BaseInstance3DBoxes*, whether or not it has a yaw angle. Defaults to True.

返回

(tuple | list | np.ndarray | torch.Tensor | *BaseInstance3DBoxes*): The converted box of the same type.

static convert_point(*point*, *src*, *dst*, *rt_mat=None*)

Convert points from *src* mode to *dst* mode.

参数

- **(tuple | list | np.ndarray |** (*point*) –torch.Tensor | BasePoints): Can be a k-tuple, k-list or an Nxk array/tensor.
- **src** (CoordMode) –The src Point mode.
- **dst** (CoordMode) –The target Point mode.

- **rt_mat** (*np.ndarray | torch.Tensor, optional*) –The rotation and translation matrix between different coordinates. Defaults to None. The conversion from *src* coordinates to *dst* coordinates usually comes along the change of sensors, e.g., from camera to LiDAR. This requires a transformation matrix.

返回 The converted point of the same type.

返回类型 (tuple | list | np.ndarray | torch.Tensor | BasePoints)

class mmdet3d.core.bbox.**DeltaXYZWLHRBBoxCoder**(*code_size=7*)

Bbox Coder for 3D boxes.

参数 **code_size** (*int*) –The dimension of boxes to be encoded.

static decode(*anchors*, *deltas*)

Apply transformation *deltas* (dx, dy, dz, dx_size, dy_size, dz_size, dr, dv*) to *boxes*.

参数

- **anchors** (*torch.Tensor*) –Parameters of anchors with shape (N, 7).
- **deltas** (*torch.Tensor*) –Encoded boxes with shape (N, 7+n) [x, y, z, x_size, y_size, z_size, r, velo*].

返回 Decoded boxes.

返回类型 torch.Tensor

static encode(*src_boxes*, *dst_boxes*)

Get box regression transformation deltas (dx, dy, dz, dx_size, dy_size, dz_size, dr, dv*) that can be used to transform the *src_boxes* into the *target_boxes*.

参数

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

返回 Box transformation deltas.

返回类型 torch.Tensor

class mmdet3d.core.bbox.**DepthInstance3DBoxes**(*tensor*, *box_dim=7*, *with_yaw=True*, *origin=(0.5, 0.5, 0)*)

3D boxes of instances in Depth coordinates.

Coordinates in Depth:

```
up z    y front (yaw=-0.5*pi)
   ^   ^
   |  /
   | /
   0 ------> x right (yaw=0)
```

The relative coordinate of bottom center in a Depth box is (0.5, 0.5, 0), and the yaw is around the z axis, thus the rotation axis=2. The yaw is 0 at the positive direction of x axis, and decreases from the positive direction of x to the positive direction of y. Also note that rotation of DepthInstance3DBoxes is counterclockwise, which is reverse to the definition of the yaw angle (clockwise).

A refactor is ongoing to make the three coordinate systems easier to understand and convert between each other.

tensor

Float matrix of N x box_dim.

Type torch.Tensor

box_dim

Integer indicates the dimension of a box Each row is (x, y, z, x_size, y_size, z_size, yaw, …).

Type int

with_yaw

If True, the value of yaw will be set to 0 as minmax boxes.

Type bool

convert_to(*dst*, *rt_mat=None*)

Convert self to `dst` mode.

参数

- **dst** (*Box3DMode*) –The target Box mode.
- **rt_mat** (*np.ndarray | torch.Tensor, optional*) –The rotation and translation matrix between different coordinates. Defaults to None. The conversion from `src` coordinates to `dst` coordinates usually comes along the change of sensors, e.g., from camera to LiDAR. This requires a transformation matrix.

返回 The converted box of the same type in the `dst` mode.

返回类型 *DepthInstance3DBoxes*

property corners

Coordinates of corners of all the boxes in shape (N, 8, 3).

Convert the boxes to corners in clockwise order, in form of `(x0y0z0, x0y0z1, x0y1z1, x0y1z0, x1y0z0, x1y0z1, x1y1z1, x1y1z0)`

```
                           up z
           front y           ^
                /            |
               /             |
 (x0, y1, z1) + -----------  + (x1, y1, z1)
              /|            / |
             / |           /  |
```

(下页继续)

(续上页)

```
(x0, y0, z1) + ----------- +   + (x1, y1, z0)
             |  /      .   |  /
             | / origin    | /
(x0, y0, z0) + ----------- + --------> right x
                           (x1, y0, z0)
```

Type torch.Tensor

enlarged_box(*extra_width*)

Enlarge the length, width and height boxes.

参数 **extra_width** (*float | torch.Tensor*) –Extra width to enlarge the box.

返回 Enlarged boxes.

返回类型 *DepthInstance3DBoxes*

flip(*bev_direction='horizontal'*, *points=None*)

Flip the boxes in BEV along given BEV direction.

In Depth coordinates, it flips x (horizontal) or y (vertical) axis.

参数

- **bev_direction** (*str, optional*) –Flip direction (horizontal or vertical). Defaults to 'horizontal'.
- **points** (torch.Tensor | np.ndarray | BasePoints, optional) –Points to flip. Defaults to None.

返回 Flipped points.

返回类型 torch.Tensor, numpy.ndarray or None

get_surface_line_center()

Compute surface and line center of bounding boxes.

返回 Surface and line center of bounding boxes.

返回类型 torch.Tensor

property gravity_center

A tensor with center of each box in shape (N, 3).

Type torch.Tensor

rotate(*angle*, *points=None*)

Rotate boxes with points (optional) with the given angle or rotation matrix.

参数

- **angle** (*float | torch.Tensor | np.ndarray*) –Rotation angle or rotation matrix.
- **points** (torch.Tensor | np.ndarray | BasePoints, optional) –Points to rotate. Defaults to None.

返回

When points is None, the function returns None, otherwise it returns the rotated points and the rotation matrix rot_mat_T.

返回类型 tuple or None

class mmdet3d.core.bbox.**InstanceBalancedPosSampler**(*num*, *pos_fraction*, *neg_pos_ub=- 1*, *add_gt_as_proposals=True*, ***kwargs*)

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

class mmdet3d.core.bbox.**IoUBalancedNegSampler**(*num*, *pos_fraction*, *floor_thr=- 1*, *floor_fraction=0*, *num_bins=3*, ***kwargs*)

IoU Balanced Sampling.

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

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

参数

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

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

Sample according to the iou interval.

参数

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

返回 Indices of samples

返回类型 np.ndarray

class mmdet3d.core.bbox.**LiDARInstance3DBoxes**(*tensor*, *box_dim=7*, *with_yaw=True*, *origin=(0.5, 0.5, 0)*)

3D boxes of instances in LIDAR coordinates.

Coordinates in LiDAR:

```
                        up z    x front (yaw=0)
                           ^   ^
                           |  /
                           | /
(yaw=0.5*pi) left y <------ 0
```

The relative coordinate of bottom center in a LiDAR box is (0.5, 0.5, 0), and the yaw is around the z axis, thus the rotation axis=2. The yaw is 0 at the positive direction of x axis, and increases from the positive direction of x to the positive direction of y.

A refactor is ongoing to make the three coordinate systems easier to understand and convert between each other.

tensor

Float matrix of N x box_dim.

Type torch.Tensor

box_dim

Integer indicating the dimension of a box. Each row is (x, y, z, x_size, y_size, z_size, yaw, ⋯).

Type int

with_yaw

If True, the value of yaw will be set to 0 as minmax boxes.

Type bool

convert_to(*dst*, *rt_mat=None*)

Convert self to `dst` mode.

参数

- **dst** (*Box3DMode*) –the target Box mode
- **rt_mat** (*np.ndarray | torch.Tensor, optional*) –The rotation and translation matrix between different coordinates. Defaults to None. The conversion from `src` coordinates to `dst` coordinates usually comes along the change of sensors, e.g., from camera to LiDAR. This requires a transformation matrix.

返回 The converted box of the same type in the `dst` mode.

返回类型 *BaseInstance3DBoxes*

property corners

Coordinates of corners of all the boxes in shape (N, 8, 3).

Convert the boxes to corners in clockwise order, in form of `(x0y0z0, x0y0z1, x0y1z1, x0y1z0, x1y0z0, x1y0z1, x1y1z1, x1y1z0)`

```
                                up z
                front x           ^
                     /            |
                    /             |
      (x1, y0, z1) + -----------  + (x1, y1, z1)
                  /|            / |
                 / |           /  |
   (x0, y0, z1) + ----------- +   + (x1, y1, z0)
                |  /      .   |  /
                | / origin    | /
left y<-------- + ----------- + (x0, y1, z0)
    (x0, y0, z0)
```

Type torch.Tensor

enlarged_box(*extra_width*)

Enlarge the length, width and height boxes.

参数 **extra_width** (*float | torch.Tensor*) –Extra width to enlarge the box.

返回 Enlarged boxes.

返回类型 *LiDARInstance3DBoxes*

flip(*bev_direction='horizontal'*, *points=None*)

Flip the boxes in BEV along given BEV direction.

In LIDAR coordinates, it flips the y (horizontal) or x (vertical) axis.

参数

- **bev_direction** (*str*) –Flip direction (horizontal or vertical).
- **points** (torch.Tensor | np.ndarray | `BasePoints`, optional) –Points to flip. Defaults to None.

返回 Flipped points.

返回类型 torch.Tensor, numpy.ndarray or None

property gravity_center

A tensor with center of each box in shape (N, 3).

Type torch.Tensor

rotate(*angle*, *points=None*)

Rotate boxes with points (optional) with the given angle or rotation matrix.

参数

- **angles** (*float | torch.Tensor | np.ndarray*) –Rotation angle or rotation matrix.
- **points** (torch.Tensor | np.ndarray | BasePoints, optional) –Points to rotate. Defaults to None.

返回

When points is None, the function returns None, otherwise it returns the rotated points and the rotation matrix rot_mat_T.

返回类型 tuple or None

class mmdet3d.core.bbox.**MaxIoUAssigner**(*pos_iou_thr*, *neg_iou_thr*, *min_pos_iou=0.0*, *gt_max_assign_all=True*, *ignore_iof_thr=- 1*, *ignore_wrt_candidates=True*, *match_low_quality=True*, *gpu_assign_thr=- 1*, *iou_calculator={'type': 'BboxOverlaps2D'}*)

Assign a corresponding gt bbox or background to each bbox.

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

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

参数

- **pos_iou_thr** (*float*) –IoU threshold for positive bboxes.
- **neg_iou_thr** (*float or tuple*) –IoU threshold for negative bboxes.
- **min_pos_iou** (*float*) –Minimum iou for a bbox to be considered as a positive bbox. Positive samples can have smaller IoU than pos_iou_thr due to the 4th step (assign max IoU sample to each gt). *min_pos_iou* is set to avoid assigning bboxes that have extremely small iou with GT as positive samples. It brings about 0.3 mAP improvements in 1x schedule but does not affect the performance of 3x schedule. More comparisons can be found in PR #7464.
- **gt_max_assign_all** (*bool*) –Whether to assign all bboxes with the same highest overlap with some gt to that gt.
- **ignore_iof_thr** (*float*) –IoF threshold for ignoring bboxes (if *gt_bboxes_ignore* is specified). Negative values mean not ignoring any bboxes.
- **ignore_wrt_candidates** (*bool*) –Whether to compute the iof between *bboxes* and *gt_bboxes_ignore*, or the contrary.

- **match_low_quality** (*bool*) –Whether to allow low quality matches. This is usually allowed for RPN and single stage detectors, but not allowed in the second stage. Details are demonstrated in Step 4.
- **gpu_assign_thr** (*int*) –The upper bound of the number of GT for GPU assign. When the number of gt is above this threshold, will assign on CPU device. Negative values mean not assign on CPU.

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

Assign gt to bboxes.

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

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

参数

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

返回 The assign result.

返回类型 *AssignResult*

示例

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

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

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

参数

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

返回 The assign result.

返回类型 *AssignResult*

class mmdet3d.core.bbox.**PseudoSampler**(***kwargs*)

A pseudo sampler that does not do sampling actually.

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

Directly returns the positive and negative indices of samples.

参数

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

返回 sampler results

返回类型 *SamplingResult*

class mmdet3d.core.bbox.**RandomSampler**(*num*, *pos_fraction*, *neg_pos_ub=- 1*, *add_gt_as_proposals=True*, ***kwargs*)

Random sampler.

参数

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

random_choice(*gallery*, *num*)

Random select some elements from the gallery.

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

参数

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

返回 sampled indices.

返回类型 Tensor or ndarray

class mmdet3d.core.bbox.**SamplingResult**(*pos_inds*, *neg_inds*, *bboxes*, *gt_bboxes*, *assign_result*, *gt_flags*)

Bbox sampling result.

示例

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

property bboxes

concatenated positive and negative boxes

Type torch.Tensor

property info

Returns a dictionary of info about the object.

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

参数

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

返回 Randomly generated sampling result.

返回类型 *SamplingResult*

示例

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

to(*device*)

Change the device of the data inplace.

示例

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

mmdet3d.core.bbox.**axis_aligned_bbox_overlaps_3d**(*bboxes1*, *bboxes2*, *mode='iou'*, *is_aligned=False*, *eps=1e-06*)

Calculate overlap between two set of axis aligned 3D bboxes. If `is_aligned` is `False`, then calculate the overlaps between each bbox of bboxes1 and bboxes2, otherwise the overlaps between each aligned pair of bboxes1 and bboxes2.

参数

- **bboxes1** (*Tensor*) –shape (B, m, 6) in <x1, y1, z1, x2, y2, z2> format or empty.
- **bboxes2** (*Tensor*) –shape (B, n, 6) in <x1, y1, z1, x2, y2, z2> format or empty. B indicates the batch dim, in shape (B1, B2, ···, Bn). If `is_aligned` is `True`, then m and n must be equal.
- **mode** (*str*) –"iou" (intersection over union) or "giou" (generalized intersection over union).
- **is_aligned** (*bool, optional*) –If True, then m and n must be equal. Defaults to False.
- **eps** (*float, optional*) –A value added to the denominator for numerical stability. Defaults to 1e-6.

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

返回类型 Tensor

示例

```
>>> bboxes1 = torch.FloatTensor([
>>>     [0, 0, 0, 10, 10, 10],
>>>     [10, 10, 10, 20, 20, 20],
>>>     [32, 32, 32, 38, 40, 42],
>>> ])
>>> bboxes2 = torch.FloatTensor([
>>>     [0, 0, 0, 10, 20, 20],
>>>     [0, 10, 10, 10, 19, 20],
>>>     [10, 10, 10, 20, 20, 20],
>>> ])
>>> overlaps = axis_aligned_bbox_overlaps_3d(bboxes1, bboxes2)
>>> assert overlaps.shape == (3, 3)
>>> overlaps = bbox_overlaps(bboxes1, bboxes2, is_aligned=True)
>>> assert overlaps.shape == (3, )
```

示例

```
>>> empty = torch.empty(0, 6)
>>> nonempty = torch.FloatTensor([[0, 0, 0, 10, 9, 10]])
>>> assert tuple(bbox_overlaps(empty, nonempty).shape) == (0, 1)
>>> assert tuple(bbox_overlaps(nonempty, empty).shape) == (1, 0)
>>> assert tuple(bbox_overlaps(empty, empty).shape) == (0, 0)
```

mmdet3d.core.bbox.**bbox3d2result**(*bboxes*, *scores*, *labels*, *attrs=None*)

Convert detection results to a list of numpy arrays.

参数

- **bboxes** (*torch.Tensor*) –Bounding boxes with shape (N, 5).
- **labels** (*torch.Tensor*) –Labels with shape (N,).
- **scores** (*torch.Tensor*) –Scores with shape (N,).
- **attrs** (*torch.Tensor, optional*) –Attributes with shape (N,). Defaults to None.

返回

Bounding box results in cpu mode.

- boxes_3d (torch.Tensor): 3D boxes.
- scores (torch.Tensor): Prediction scores.
- labels_3d (torch.Tensor): Box labels.
- attrs_3d (torch.Tensor, optional): Box attributes.

返回类型 dict[str, torch.Tensor]

mmdet3d.core.bbox.**bbox3d2roi**(*bbox_list*)

Convert a list of bounding boxes to roi format.

参数 **bbox_list** (*list[torch.Tensor]*) –A list of bounding boxes corresponding to a batch of images.

返回

Region of interests in shape (n, c), where the channels are in order of [batch_ind, x, y …].

返回类型 torch.Tensor

mmdet3d.core.bbox.**bbox3d_mapping_back**(*bboxes*, *scale_factor*, *flip_horizontal*, *flip_vertical*)

Map bboxes from testing scale to original image scale.

参数

- **bboxes** (*BaseInstance3DBoxes*) –Boxes to be mapped back.
- **scale_factor** (*float*) –Scale factor.
- **flip_horizontal** (*bool*) –Whether to flip horizontally.
- **flip_vertical** (*bool*) –Whether to flip vertically.

返回 Boxes mapped back.

返回类型 *BaseInstance3DBoxes*

mmdet3d.core.bbox.**bbox_overlaps_3d**(*bboxes1*, *bboxes2*, *mode='iou'*, *coordinate='camera'*)

Calculate 3D IoU using cuda implementation.

注解: This function calculates the IoU of 3D boxes based on their volumes. IoU calculator *BboxOverlaps3D* uses this function to calculate the actual IoUs of boxes.

参数

- **bboxes1** (*torch.Tensor*) –with shape (N, 7+C), (x, y, z, x_size, y_size, z_size, ry, v*).
- **bboxes2** (*torch.Tensor*) –with shape (M, 7+C), (x, y, z, x_size, y_size, z_size, ry, v*).
- **mode** (*str*) – "iou" (intersection over union) or iof (intersection over foreground).
- **coordinate** (*str*) – 'camera' or 'lidar' coordinate system.

返回

Bbox overlaps results of bboxes1 and bboxes2 with shape (M, N) (aligned mode is not supported currently).

返回类型 torch.Tensor

mmdet3d.core.bbox.**bbox_overlaps_nearest_3d**(*bboxes1*, *bboxes2*, *mode='iou'*, *is_aligned=False*, *coordinate='lidar'*)

Calculate nearest 3D IoU.

注解： This function first finds the nearest 2D boxes in bird eye view (BEV), and then calculates the 2D IoU using `bbox_overlaps()`. This IoU calculator *BboxOverlapsNearest3D* uses this function to calculate IoUs of boxes.

If `is_aligned` is `False`, then it calculates the ious between each bbox of bboxes1 and bboxes2, otherwise the ious between each aligned pair of bboxes1 and bboxes2.

参数

- **bboxes1** (*torch.Tensor*) –with shape (N, 7+C), (x, y, z, x_size, y_size, z_size, ry, v*).
- **bboxes2** (*torch.Tensor*) –with shape (M, 7+C), (x, y, z, x_size, y_size, z_size, ry, v*).
- **mode** (*str*) –"iou" (intersection over union) or iof (intersection over foreground).
- **is_aligned** (*bool*) –Whether the calculation is aligned

返回

If is_aligned is True, return ious between bboxes1 and bboxes2 with shape (M, N). If `is_aligned` is `False`, return shape is M.

返回类型 torch.Tensor

mmdet3d.core.bbox.**get_box_type**(*box_type*)

Get the type and mode of box structure.

参数 **box_type** (*str*) –The type of box structure. The valid value are "LiDAR", "Camera", or "Depth".

引发 **ValueError** –A ValueError is raised when *box_type* does not belong to the three valid types.

返回 Box type and box mode.

返回类型 tuple

mmdet3d.core.bbox.**limit_period**(*val*, *offset=0.5*, *period=3.141592653589793*)

Limit the value into a period for periodic function.

参数

- **val** (*torch.Tensor | np.ndarray*) –The value to be converted.
- **offset** (*float, optional*) –Offset to set the value range. Defaults to 0.5.
- **period** (*[type], optional*) –Period of the value. Defaults to np.pi.

返回

Value in the range of [-offset * period, (1-offset) * period]

返回类型 (torch.Tensor | np.ndarray)

mmdet3d.core.bbox.**mono_cam_box2vis**(*cam_box*)

This is a post-processing function on the bboxes from Mono-3D task. If we want to perform projection visualization, we need to:

1. rotate the box along x-axis for np.pi / 2 (roll)
2. change orientation from local yaw to global yaw
3. convert yaw by (np.pi / 2 - yaw)

After applying this function, we can project and draw it on 2D images.

参数 **cam_box** (*CameraInstance3DBoxes*) –3D bbox in camera coordinate system before conversion. Could be gt bbox loaded from dataset or network prediction output.

返回 Box after conversion.

返回类型 *CameraInstance3DBoxes*

mmdet3d.core.bbox.**points_cam2img**(*points_3d*, *proj_mat*, *with_depth=False*)

Project points in camera coordinates to image coordinates.

参数

- **points_3d** (*torch.Tensor | np.ndarray*) –Points in shape (N, 3)
- **proj_mat** (*torch.Tensor | np.ndarray*) –Transformation matrix between coordinates.
- **with_depth** (*bool, optional*) –Whether to keep depth in the output. Defaults to False.

返回

Points in image coordinates, with shape [N, 2] if *with_depth=False*, else [N, 3].

返回类型 (torch.Tensor | np.ndarray)

mmdet3d.core.bbox.**points_img2cam**(*points*, *cam2img*)

Project points in image coordinates to camera coordinates.

参数

- **points** (*torch.Tensor*) –2.5D points in 2D images, [N, 3], 3 corresponds with x, y in the image and depth.
- **cam2img** (*torch.Tensor*) –Camera intrinsic matrix. The shape can be [3, 3], [3, 4] or [4, 4].

返回

points in 3D space. [N, 3], 3 corresponds with x, y, z in 3D space.

返回类型 torch.Tensor

mmdet3d.core.bbox.**xywhr2xyxyr**(*boxes_xywhr*)

Convert a rotated boxes in XYWHR format to XYXYR format.

参数 **boxes_xywhr** (*torch.Tensor | np.ndarray*) –Rotated boxes in XYWHR format.

返回 Converted boxes in XYXYR format.

返回类型 (torch.Tensor | np.ndarray)

38.3 evaluation

mmdet3d.core.evaluation.**indoor_eval**(*gt_annos*, *dt_annos*, *metric*, *label2cat*, *logger=None*, *box_type_3d=None*, *box_mode_3d=None*)

Indoor Evaluation.

Evaluate the result of the detection.

参数

- **gt_annos** (*list[dict]*) –Ground truth annotations.
- **dt_annos** (*list[dict]*) –Detection annotations. the dict includes the following keys
  - labels_3d (torch.Tensor): Labels of boxes.
  - **boxes_3d (BaseInstance3DBoxes):** 3D bounding boxes in Depth coordinate.
  - scores_3d (torch.Tensor): Scores of boxes.
- **metric** (*list[float]*) –IoU thresholds for computing average precisions.
- **label2cat** (*dict*) –Map from label to category.
- **logger** (*logging.Logger | str, optional*) –The way to print the mAP summary. See *mmdet.utils.print_log()* for details. Default: None.

返回 Dict of results.

返回类型 dict[str, float]

mmdet3d.core.evaluation.**instance_seg_eval**(*gt_semantic_masks*, *gt_instance_masks*, *pred_instance_masks*, *pred_instance_labels*, *pred_instance_scores*, *valid_class_ids*, *class_labels*, *options=None*, *logger=None*)

Instance Segmentation Evaluation.

Evaluate the result of the instance segmentation.

参数

- **gt_semantic_masks** (*list[torch.Tensor]*) –Ground truth semantic masks.

- **gt_instance_masks** (*list[torch.Tensor]*) –Ground truth instance masks.
- **pred_instance_masks** (*list[torch.Tensor]*) –Predicted instance masks.
- **pred_instance_labels** (*list[torch.Tensor]*) –Predicted instance labels.
- **pred_instance_scores** (*list[torch.Tensor]*) –Predicted instance labels.
- **valid_class_ids** (*tuple[int]*) –Ids of valid categories.
- **class_labels** (*tuple[str]*) –Names of valid categories.
- **options** (*dict, optional*) –Additional options. Keys may contain: *overlaps*, *min_region_sizes*, *distance_threshes*, *distance_confs*. Default: None.
- **logger** (*logging.Logger | str, optional*) –The way to print the mAP summary. See *mmdet.utils.print_log()* for details. Default: None.

返回 Dict of results.

返回类型 dict[str, float]

mmdet3d.core.evaluation.**kitti_eval**(*gt_annos*, *dt_annos*, *current_classes*, *eval_types=['bbox', 'bev', '3d']*)

KITTI evaluation.

参数

- **gt_annos** (*list[dict]*) –Contain gt information of each sample.
- **dt_annos** (*list[dict]*) –Contain detected information of each sample.
- **current_classes** (*list[str]*) –Classes to evaluation.
- **eval_types** (*list[str], optional*) –Types to eval. Defaults to [‘bbox’, ‘bev’, ‘3d’].

返回 String and dict of evaluation results.

返回类型 tuple

mmdet3d.core.evaluation.**kitti_eval_coco_style**(*gt_annos*, *dt_annos*, *current_classes*)

coco style evaluation of kitti.

参数

- **gt_annos** (*list[dict]*) –Contain gt information of each sample.
- **dt_annos** (*list[dict]*) –Contain detected information of each sample.
- **current_classes** (*list[str]*) –Classes to evaluation.

返回 Evaluation results.

返回类型 string

mmdet3d.core.evaluation.**lyft_eval**(*lyft*, *data_root*, *res_path*, *eval_set*, *output_dir*, *logger=None*)

Evaluation API for Lyft dataset.

参数

- **lyft** (LyftDataset) –Lyft class in the sdk.
- **data_root** (*str*) –Root of data for reading splits.
- **res_path** (*str*) –Path of result json file recording detections.
- **eval_set** (*str*) –Name of the split for evaluation.
- **output_dir** (*str*) –Output directory for output json files.
- **logger** (*logging.Logger | str, optional*) –Logger used for printing related information during evaluation. Default: None.

返回 The evaluation results.

返回类型 dict[str, float]

mmdet3d.core.evaluation.**seg_eval**(*gt_labels*, *seg_preds*, *label2cat*, *ignore_index*, *logger=None*)

Semantic Segmentation Evaluation.

Evaluate the result of the Semantic Segmentation.

参数

- **gt_labels** (*list[torch.Tensor]*) –Ground truth labels.
- **seg_preds** (*list[torch.Tensor]*) –Predictions.
- **label2cat** (*dict*) –Map from label to category name.
- **ignore_index** (*int*) –Index that will be ignored in evaluation.
- **logger** (*logging.Logger | str, optional*) –The way to print the mAP summary. See *mmdet.utils.print_log()* for details. Default: None.

返回 Dict of results.

返回类型 dict[str, float]

38.4 visualizer

mmdet3d.core.visualizer.**show_multi_modality_result**(*img*, *gt_bboxes*, *pred_bboxes*, *proj_mat*, *out_dir*, *filename*, *box_mode='lidar'*, *img_metas=None*, *show=False*, *gt_bbox_color=(61, 102, 255)*, *pred_bbox_color=(241, 101, 72)*)

Convert multi-modality detection results into 2D results.

Project the predicted 3D bbox to 2D image plane and visualize them.

参数

- **img** (*np.ndarray*) –The numpy array of image in cv2 fashion.
- **gt_bboxes** (BaseInstance3DBoxes) –Ground truth boxes.
- **pred_bboxes** (BaseInstance3DBoxes) –Predicted boxes.
- **proj_mat** (*numpy.array, shape=[4, 4]*) –The projection matrix according to the camera intrinsic parameters.
- **out_dir** (*str*) –Path of output directory.
- **filename** (*str*) –Filename of the current frame.
- **box_mode** (*str, optional*) –Coordinate system the boxes are in. Should be one of 'depth', 'lidar' and 'camera'. Defaults to 'lidar'.
- **img_metas** (*dict, optional*) –Used in projecting depth bbox. Defaults to None.
- **show** (*bool, optional*) –Visualize the results online. Defaults to False.
- **gt_bbox_color** (*str or tuple(int), optional*) –Color of bbox lines. The tuple of color should be in BGR order. Default: (255, 102, 61).
- **pred_bbox_color** (*str or tuple(int), optional*) –Color of bbox lines. The tuple of color should be in BGR order. Default: (72, 101, 241).

mmdet3d.core.visualizer.**show_result**(*points*, *gt_bboxes*, *pred_bboxes*, *out_dir*, *filename*, *show=False*, *snapshot=False*, *pred_labels=None*)

Convert results into format that is directly readable for meshlab.

参数

- **points** (*np.ndarray*) –Points.
- **gt_bboxes** (*np.ndarray*) –Ground truth boxes.
- **pred_bboxes** (*np.ndarray*) –Predicted boxes.
- **out_dir** (*str*) –Path of output directory
- **filename** (*str*) –Filename of the current frame.
- **show** (*bool, optional*) –Visualize the results online. Defaults to False.
- **snapshot** (*bool, optional*) –Whether to save the online results. Defaults to False.
- **pred_labels** (*np.ndarray, optional*) –Predicted labels of boxes. Defaults to None.

mmdet3d.core.visualizer.**show_seg_result**(*points*, *gt_seg*, *pred_seg*, *out_dir*, *filename*, *palette*, *ignore_index=None*, *show=False*, *snapshot=False*)

Convert results into format that is directly readable for meshlab.

参数

- **points** (*np.ndarray*) –Points.
- **gt_seg** (*np.ndarray*) –Ground truth segmentation mask.
- **pred_seg** (*np.ndarray*) –Predicted segmentation mask.
- **out_dir** (*str*) –Path of output directory
- **filename** (*str*) –Filename of the current frame.
- **palette** (*np.ndarray*) –Mapping between class labels and colors.
- **ignore_index** (*int, optional*) –The label index to be ignored, e.g. unannotated points. Defaults to None.
- **show** (*bool, optional*) –Visualize the results online. Defaults to False.
- **snapshot** (*bool, optional*) –Whether to save the online results. Defaults to False.

38.5 voxel

class mmdet3d.core.voxel.**VoxelGenerator**(*voxel_size*, *point_cloud_range*, *max_num_points*, *max_voxels=20000*)

Voxel generator in numpy implementation.

参数

- **voxel_size** (*list[float]*) –Size of a single voxel
- **point_cloud_range** (*list[float]*) –Range of points
- **max_num_points** (*int*) –Maximum number of points in a single voxel
- **max_voxels** (*int, optional*) –Maximum number of voxels. Defaults to 20000.

generate(*points*)

Generate voxels given points.

property grid_size

The size of grids.

Type np.ndarray

property max_num_points_per_voxel

Maximum number of points per voxel.

Type int

property point_cloud_range

Range of point cloud.

Type list[float]

property voxel_size

Size of a single voxel.

Type list[float]

mmdet3d.core.voxel.**build_voxel_generator**(*cfg*, ***kwargs*)

Builder of voxel generator.

38.6 post_processing

mmdet3d.core.post_processing.**aligned_3d_nms**(*boxes*, *scores*, *classes*, *thresh*)

3D NMS for aligned boxes.

参数

- **boxes** (*torch.Tensor*) –Aligned box with shape [n, 6].
- **scores** (*torch.Tensor*) –Scores of each box.
- **classes** (*torch.Tensor*) –Class of each box.
- **thresh** (*float*) –IoU threshold for nms.

返回 Indices of selected boxes.

返回类型 torch.Tensor

mmdet3d.core.post_processing.**box3d_multiclass_nms**(*mlvl_bboxes*, *mlvl_bboxes_for_nms*, *mlvl_scores*, *score_thr*, *max_num*, *cfg*, *mlvl_dir_scores=None*, *mlvl_attr_scores=None*, *mlvl_bboxes2d=None*)

Multi-class NMS for 3D boxes. The IoU used for NMS is defined as the 2D IoU between BEV boxes.

参数

- **mlvl_bboxes** (*torch.Tensor*) –Multi-level boxes with shape (N, M). M is the dimensions of boxes.
- **mlvl_bboxes_for_nms** (*torch.Tensor*) –Multi-level boxes with shape (N, 5) ([x1, y1, x2, y2, ry]). N is the number of boxes. The coordinate system of the BEV boxes is counterclockwise.
- **mlvl_scores** (*torch.Tensor*) –Multi-level boxes with shape (N, C + 1). N is the number of boxes. C is the number of classes.
- **score_thr** (*float*) –Score threshold to filter boxes with low confidence.
- **max_num** (*int*) –Maximum number of boxes will be kept.
- **cfg** (*dict*) –Configuration dict of NMS.

- **mlvl_dir_scores** (*torch.Tensor, optional*) –Multi-level scores of direction classifier. Defaults to None.
- **mlvl_attr_scores** (*torch.Tensor, optional*) –Multi-level scores of attribute classifier. Defaults to None.
- **mlvl_bboxes2d**(*torch.Tensor, optional*) –Multi-level 2D bounding boxes. Defaults to None.

返回

Return results after nms, including 3D bounding boxes, scores, labels, direction scores, attribute scores (optional) and 2D bounding boxes (optional).

返回类型 tuple[torch.Tensor]

mmdet3d.core.post_processing.**circle_nms**(*dets*, *thresh*, *post_max_size=83*)

Circular NMS.

An object is only counted as positive if no other center with a higher confidence exists within a radius r using a bird-eye view distance metric.

参数

- **dets** (*torch.Tensor*) –Detection results with the shape of [N, 3].
- **thresh** (*float*) –Value of threshold.
- **post_max_size** (*int, optional*) –Max number of prediction to be kept. Defaults to 83.

返回 Indexes of the detections to be kept.

返回类型 torch.Tensor

mmdet3d.core.post_processing.**merge_aug_bboxes**(*aug_bboxes*, *aug_scores*, *img_metas*, *rcnn_test_cfg*)

Merge augmented detection bboxes and scores.

参数

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

返回 (bboxes, scores)

返回类型 tuple

mmdet3d.core.post_processing.**merge_aug_bboxes_3d**(*aug_results*, *img_metas*, *test_cfg*)

Merge augmented detection 3D bboxes and scores.

参数

- **aug_results** (*list[dict]*) –The dict of detection results. The dict contains the following keys
  - boxes_3d (BaseInstance3DBoxes): Detection bbox.
  - scores_3d (torch.Tensor): Detection scores.
  - labels_3d (torch.Tensor): Predicted box labels.
- **img_metas** (*list[dict]*) –Meta information of each sample.
- **test_cfg** (*dict*) –Test config.

返回

Bounding boxes results in cpu mode, containing merged results.

- boxes_3d (BaseInstance3DBoxes): Merged detection bbox.
- scores_3d (torch.Tensor): Merged detection scores.
- labels_3d (torch.Tensor): Merged predicted box labels.

返回类型 dict

mmdet3d.core.post_processing.**merge_aug_masks**(*aug_masks*, *img_metas*, *rcnn_test_cfg*, *weights=None*)

Merge augmented mask prediction.

参数

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

返回 (bboxes, scores)

返回类型 tuple

mmdet3d.core.post_processing.**merge_aug_proposals**(*aug_proposals*, *img_metas*, *cfg*)

Merge augmented proposals (multiscale, flip, etc.)

参数

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

- **cfg** (*dict*) –rpn test config.

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

返回类型 Tensor

mmdet3d.core.post_processing.**merge_aug_scores**(*aug_scores*)

Merge augmented bbox scores.

mmdet3d.core.post_processing.**multiclass_nms**(*multi_bboxes*, *multi_scores*, *score_thr*, *nms_cfg*, *max_num=- 1*, *score_factors=None*, *return_inds=False*)

NMS for multi-class bboxes.

参数

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

返回

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

返回类型 tuple

mmdet3d.core.post_processing.**nms_bev**(*boxes*, *scores*, *thresh*, *pre_max_size=None*, *post_max_size=None*)

NMS function GPU implementation (for BEV boxes). The overlap of two boxes for IoU calculation is defined as the exact overlapping area of the two boxes. In this function, one can also set `pre_max_size` and `post_max_size`.

参数

- **boxes** (*torch.Tensor*) –Input boxes with the shape of [N, 5] ([x1, y1, x2, y2, ry]).
- **scores** (*torch.Tensor*) –Scores of boxes with the shape of [N].

- **thresh** (*float*) –Overlap threshold of NMS.
- **pre_max_size** (*int, optional*) –Max size of boxes before NMS. Default: None.
- **post_max_size** (*int, optional*) –Max size of boxes after NMS. Default: None.

返回 Indexes after NMS.

返回类型 torch.Tensor

mmdet3d.core.post_processing.**nms_normal_bev**(*boxes*, *scores*, *thresh*)

Normal NMS function GPU implementation (for BEV boxes). The overlap of two boxes for IoU calculation is defined as the exact overlapping area of the two boxes WITH their yaw angle set to 0.

参数

- **boxes** (*torch.Tensor*) –Input boxes with shape (N, 5).
- **scores** (*torch.Tensor*) –Scores of predicted boxes with shape (N).
- **thresh** (*float*) –Overlap threshold of NMS.

返回 Remaining indices with scores in descending order.

返回类型 torch.Tensor

CHAPTER 39

mmdet3d.datasets

class mmdet3d.datasets.**AffineResize**(*img_scale*, *down_ratio*, *bbox_clip_border=True*)

Get the affine transform matrices to the target size.

Different from `RandomAffine` in MMDetection, this class can calculate the affine transform matrices while resizing the input image to a fixed size. The affine transform matrices include: 1) matrix transforming original image to the network input image size. 2) matrix transforming original image to the network output feature map size.

参数

- **img_scale** (*tuple*) –Images scales for resizing.
- **down_ratio** (*int*) –The down ratio of feature map. Actually the arg should be >= 1.
- **bbox_clip_border** (*bool, optional*) –Whether clip the objects outside the border of the image. Defaults to True.

class mmdet3d.datasets.**BackgroundPointsFilter**(*bbox_enlarge_range*)

Filter background points near the bounding box.

参数 **bbox_enlarge_range** (*tuple[float], float*) –Bbox enlarge range.

class mmdet3d.datasets.**Custom3DDataset**(*data_root*, *ann_file*, *pipeline=None*, *classes=None*, *modality=None*, *box_type_3d='LiDAR'*, *filter_empty_gt=True*, *test_mode=False*, *file_client_args={'backend': 'disk'}*)

Customized 3D dataset.

This is the base dataset of SUNRGB-D, ScanNet, nuScenes, and KITTI dataset.

[

{ ‘sample_idx’ :

‘lidar_points’ : { ‘lidar_path’ : velodyne_path,

},

‘annos’ : { ‘box_type_3d’ : (str) ‘LiDAR/Camera/Depth’

‘gt_bboxes_3d’ : <np.ndarray> (n, 7) ‘gt_names’ : [list] ….

}

‘calib’ : { …..} ‘images’ : { …..}

}

]

参数

- **data_root** (*str*) –Path of dataset root.
- **ann_file** (*str*) –Path of annotation file.
- **pipeline** (*list[dict], optional*) –Pipeline used for data processing. Defaults to None.
- **classes** (*tuple[str], optional*) –Classes used in the dataset. Defaults to None.
- **modality** (*dict, optional*) –Modality to specify the sensor data used as input. Defaults to None.
- **box_type_3d** (*str, optional*) –Type of 3D box of this dataset. Based on the *box_type_3d*, the dataset will encapsulate the box to its original format then converted them to *box_type_3d*. Defaults to ‘LiDAR’. Available options includes
  - ’ LiDAR’ : Box in LiDAR coordinates.
  - ’ Depth’ : Box in depth coordinates, usually for indoor dataset.
  - ’ Camera’ : Box in camera coordinates.
- **filter_empty_gt** (*bool, optional*) –Whether to filter empty GT. Defaults to True.
- **test_mode** (*bool, optional*) –Whether the dataset is in test mode. Defaults to False.

evaluate(*results*, *metric=None*, *iou_thr=(0.25, 0.5)*, *logger=None*, *show=False*, *out_dir=None*, *pipeline=None*)

Evaluate.

Evaluation in indoor protocol.

参数

- **results** (*list[dict]*) –List of results.
- **metric** (*str | list[str], optional*) –Metrics to be evaluated. Defaults to None.
- **iou_thr** (*list[float]*) –AP IoU thresholds. Defaults to (0.25, 0.5).
- **logger** (*logging.Logger | str, optional*) –Logger used for printing related information during evaluation. Defaults to None.
- **show** (*bool, optional*) –Whether to visualize. Default: False.
- **out_dir** (*str, optional*) –Path to save the visualization results. Default: None.
- **pipeline** (*list[dict], optional*) –raw data loading for showing. Default: None.

返回 Evaluation results.

返回类型 dict

format_results(*outputs*, *pklfile_prefix=None*, *submission_prefix=None*)

Format the results to pkl file.

参数

- **outputs** (*list[dict]*) –Testing results of the dataset.
- **pklfile_prefix** (*str*) –The prefix of pkl files. It includes the file path and the prefix of filename, e.g., "a/b/prefix". If not specified, a temp file will be created. Default: None.

返回

(outputs, tmp_dir), outputs is the detection results, tmp_dir is the temporal directory created for saving json files when `jsonfile_prefix` is not specified.

返回类型 tuple

get_ann_info(*index*)

Get annotation info according to the given index.

参数 **index** (*int*) –Index of the annotation data to get.

返回

Annotation information consists of the following keys:

- **gt_bboxes_3d (LiDARInstance3DBoxes):** 3D ground truth bboxes
- gt_labels_3d (np.ndarray): Labels of ground truths.
- gt_names (list[str]): Class names of ground truths.

返回类型 dict

classmethod get_classes(*classes=None*)

Get class names of current dataset.

参数 **classes** (*Sequence[str] | str*) –If classes is None, use default CLASSES defined by builtin dataset. If classes is a string, take it as a file name. The file contains the name of classes where each line contains one class name. If classes is a tuple or list, override the CLASSES defined by the dataset.

返回 A list of class names.

返回类型 list[str]

get_data_info(*index*)

Get data info according to the given index.

参数 **index** (*int*) –Index of the sample data to get.

返回

Data information that will be passed to the data preprocessing pipelines. It includes the following keys:

- sample_idx (str): Sample index.
- pts_filename (str): Filename of point clouds.
- file_name (str): Filename of point clouds.
- ann_info (dict): Annotation info.

返回类型 dict

load_annotations(*ann_file*)

Load annotations from ann_file.

参数 **ann_file** (*str*) –Path of the annotation file.

返回 List of annotations.

返回类型 list[dict]

pre_pipeline(*results*)

Initialization before data preparation.

参数 **results** (*dict*) –Dict before data preprocessing.

- img_fields (list): Image fields.
- bbox3d_fields (list): 3D bounding boxes fields.
- pts_mask_fields (list): Mask fields of points.
- pts_seg_fields (list): Mask fields of point segments.
- bbox_fields (list): Fields of bounding boxes.

- mask_fields (list): Fields of masks.
- seg_fields (list): Segment fields.
- box_type_3d (str): 3D box type.
- box_mode_3d (str): 3D box mode.

prepare_test_data(*index*)

Prepare data for testing.

参数 **index** (*int*) –Index for accessing the target data.

返回 Testing data dict of the corresponding index.

返回类型 dict

prepare_train_data(*index*)

Training data preparation.

参数 **index** (*int*) –Index for accessing the target data.

返回 Training data dict of the corresponding index.

返回类型 dict

class mmdet3d.datasets.**Custom3DSegDataset**(*data_root*, *ann_file*, *pipeline=None*, *classes=None*, *palette=None*, *modality=None*, *test_mode=False*, *ignore_index=None*, *scene_idxs=None*, *file_client_args={'backend': 'disk'}*)

Customized 3D dataset for semantic segmentation task.

This is the base dataset of ScanNet and S3DIS dataset.

参数

- **data_root** (*str*) –Path of dataset root.
- **ann_file** (*str*) –Path of annotation file.
- **pipeline** (*list[dict], optional*) –Pipeline used for data processing. Defaults to None.
- **classes** (*tuple[str], optional*) –Classes used in the dataset. Defaults to None.
- **palette** (*list[list[int]], optional*) –The palette of segmentation map. Defaults to None.
- **modality** (*dict, optional*) –Modality to specify the sensor data used as input. Defaults to None.
- **test_mode** (*bool, optional*) –Whether the dataset is in test mode. Defaults to False.

- **ignore_index** (*int, optional*) –The label index to be ignored, e.g. unannotated points. If None is given, set to len(self.CLASSES) to be consistent with PointSegClassMapping function in pipeline. Defaults to None.
- **scene_idxs** (*np.ndarray | str, optional*) –Precomputed index to load data. For scenes with many points, we may sample it several times. Defaults to None.

evaluate(*results*, *metric=None*, *logger=None*, *show=False*, *out_dir=None*, *pipeline=None*)

Evaluate.

Evaluation in semantic segmentation protocol.

参数

- **results** (*list[dict]*) –List of results.
- **metric** (*str | list[str]*) –Metrics to be evaluated.
- **logger** (*logging.Logger | str, optional*) –Logger used for printing related information during evaluation. Defaults to None.
- **show** (*bool, optional*) –Whether to visualize. Defaults to False.
- **out_dir** (*str, optional*) –Path to save the visualization results. Defaults to None.
- **pipeline** (*list[dict], optional*) –raw data loading for showing. Default: None.

返回 Evaluation results.

返回类型 dict

format_results(*outputs*, *pklfile_prefix=None*, *submission_prefix=None*)

Format the results to pkl file.

参数

- **outputs** (*list[dict]*) –Testing results of the dataset.
- **pklfile_prefix** (*str*) –The prefix of pkl files. It includes the file path and the prefix of filename, e.g., "a/b/prefix". If not specified, a temp file will be created. Default: None.

返回

(outputs, tmp_dir), outputs is the detection results, tmp_dir is the temporal directory created for saving json files when `jsonfile_prefix` is not specified.

返回类型 tuple

get_classes_and_palette(*classes=None*, *palette=None*)

Get class names of current dataset.

This function is taken from MMSegmentation.

参数

- **classes** (*Sequence[str] | str*) –If classes is None, use default CLASSES defined by builtin dataset. If classes is a string, take it as a file name. The file contains the name of classes where each line contains one class name. If classes is a tuple or list, override the CLASSES defined by the dataset. Defaults to None.
- **palette** (*Sequence[Sequence[int]] | np.ndarray*) –The palette of segmentation map. If None is given, random palette will be generated. Defaults to None.

get_data_info(*index*)

Get data info according to the given index.

参数 **index** (*int*) –Index of the sample data to get.

返回

Data information that will be passed to the data preprocessing pipelines. It includes the following keys:

- sample_idx (str): Sample index.
- pts_filename (str): Filename of point clouds.
- file_name (str): Filename of point clouds.
- ann_info (dict): Annotation info.

返回类型 dict

get_scene_idxs(*scene_idxs*)

Compute scene_idxs for data sampling.

We sample more times for scenes with more points.

load_annotations(*ann_file*)

Load annotations from ann_file.

参数 **ann_file** (*str*) –Path of the annotation file.

返回 List of annotations.

返回类型 list[dict]

pre_pipeline(*results*)

Initialization before data preparation.

参数 **results** (*dict*) –Dict before data preprocessing.

- img_fields (list): Image fields.
- pts_mask_fields (list): Mask fields of points.
- pts_seg_fields (list): Mask fields of point segments.
- mask_fields (list): Fields of masks.
- seg_fields (list): Segment fields.

prepare_test_data(*index*)

Prepare data for testing.

参数 **index** (*int*) –Index for accessing the target data.

返回 Testing data dict of the corresponding index.

返回类型 dict

prepare_train_data(*index*)

Training data preparation.

参数 **index** (*int*) –Index for accessing the target data.

返回 Training data dict of the corresponding index.

返回类型 dict

class mmdet3d.datasets.**GlobalAlignment**(*rotation_axis*)

Apply global alignment to 3D scene points by rotation and translation.

参数 **rotation_axis** (*int*) –Rotation axis for points and bboxes rotation.

注解:

We do not record the applied rotation and translation as in GlobalRotScaleTrans. Because usually, we do not need to reverse the alignment step.

For example, ScanNet 3D detection task uses aligned ground-truth bounding boxes for evaluation.

---

class mmdet3d.datasets.**GlobalRotScaleTrans**(*rot_range=[- 0.78539816, 0.78539816]*, *scale_ratio_range=[0.95, 1.05]*, *translation_std=[0, 0, 0]*, *shift_height=False*)

Apply global rotation, scaling and translation to a 3D scene.

参数

- **rot_range** (*list[float], optional*) –Range of rotation angle. Defaults to [-0.78539816, 0.78539816] (close to [-pi/4, pi/4]).
- **scale_ratio_range** (*list[float], optional*) –Range of scale ratio. Defaults to [0.95, 1.05].
- **translation_std** (*list[float], optional*) –The standard deviation of translation noise applied to a scene, which is sampled from a gaussian distribution whose standard deviation is set by `translation_std`. Defaults to [0, 0, 0]
- **shift_height** (*bool, optional*) –Whether to shift height. (the fourth dimension of indoor points) when scaling. Defaults to False.

class mmdet3d.datasets.**IndoorPatchPointSample**(*num_points*, *block_size=1.5*, *sample_rate=None*, *ignore_index=None*, *use_normalized_coord=False*, *num_try=10*, *enlarge_size=0.2*, *min_unique_num=None*, *eps=0.01*)

Indoor point sample within a patch. Modified from PointNet++.

Sampling data to a certain number for semantic segmentation.

参数

- **num_points** (*int*) –Number of points to be sampled.
- **block_size** (*float, optional*) –Size of a block to sample points from. Defaults to 1.5.
- **sample_rate** (*float, optional*) –Stride used in sliding patch generation. This parameter is unused in *IndoorPatchPointSample* and thus has been deprecated. We plan to remove it in the future. Defaults to None.
- **ignore_index** (*int, optional*) –Label index that won't be used for the segmentation task. This is set in PointSegClassMapping as neg_cls. If not None, will be used as a patch selection criterion. Defaults to None.
- **use_normalized_coord** (*bool, optional*) –Whether to use normalized xyz as additional features. Defaults to False.
- **num_try** (*int, optional*) –Number of times to try if the patch selected is invalid. Defaults to 10.
- **enlarge_size** (*float, optional*) –Enlarge the sampled patch to [-block_size / 2 - enlarge_size, block_size / 2 + enlarge_size] as an augmentation. If None, set it as 0. Defaults to 0.2.
- **min_unique_num** (*int, optional*) –Minimum number of unique points the sampled patch should contain. If None, use PointNet++'s method to judge uniqueness. Defaults to None.
- **eps** (*float, optional*) –A value added to patch boundary to guarantee points coverage. Defaults to 1e-2.

注解:

This transform should only be used in the training process of point cloud segmentation tasks. For the sliding patch generation and inference process in testing, please refer to the *slide_inference* function of *EncoderDecoder3D* class.

---

class mmdet3d.datasets.**IndoorPointSample**(**args*, ***kwargs*)

Indoor point sample.

Sampling data to a certain number. NOTE: IndoorPointSample is deprecated in favor of PointSample

参数 **num_points** (*int*) –Number of points to be sampled.

class mmdet3d.datasets.**KittiDataset**(*data_root*, *ann_file*, *split*, *pts_prefix='velodyne'*, *pipeline=None*, *classes=None*, *modality=None*, *box_type_3d='LiDAR'*, *filter_empty_gt=True*, *test_mode=False*, *pcd_limit_range=[0, -40, - 3, 70.4, 40, 0.0]*, ***kwargs*)

KITTI Dataset.

This class serves as the API for experiments on the KITTI Dataset.

参数

- **data_root** (*str*) –Path of dataset root.
- **ann_file** (*str*) –Path of annotation file.
- **split** (*str*) –Split of input data.
- **pts_prefix** (*str, optional*) –Prefix of points files. Defaults to ‘velodyne’.
- **pipeline** (*list[dict], optional*) –Pipeline used for data processing. Defaults to None.
- **classes** (*tuple[str], optional*) –Classes used in the dataset. Defaults to None.
- **modality** (*dict, optional*) –Modality to specify the sensor data used as input. Defaults to None.
- **box_type_3d** (*str, optional*) –Type of 3D box of this dataset. Based on the *box_type_3d*, the dataset will encapsulate the box to its original format then converted them to *box_type_3d*. Defaults to ‘LiDAR’ in this dataset. Available options includes
  - ’LiDAR’: Box in LiDAR coordinates.
  - ’Depth’: Box in depth coordinates, usually for indoor dataset.
  - ’Camera’: Box in camera coordinates.
- **filter_empty_gt** (*bool, optional*) –Whether to filter empty GT. Defaults to True.
- **test_mode** (*bool, optional*) –Whether the dataset is in test mode. Defaults to False.
- **pcd_limit_range** (*list, optional*) –The range of point cloud used to filter invalid predicted boxes. Default: [0, -40, -3, 70.4, 40, 0.0].

bbox2result_kitti(*net_outputs*, *class_names*, *pklfile_prefix=None*, *submission_prefix=None*)

Convert 3D detection results to kitti format for evaluation and test submission.

参数

- **net_outputs** (*list[np.ndarray]*) –List of array storing the inferenced bounding boxes and scores.
- **class_names** (*list[String]*) –A list of class names.
- **pklfile_prefix** (*str*) –The prefix of pkl file.
- **submission_prefix** (*str*) –The prefix of submission file.

返回 A list of dictionaries with the kitti format.

返回类型 list[dict]

bbox2result_kitti2d(*net_outputs*, *class_names*, *pklfile_prefix=None*, *submission_prefix=None*)

Convert 2D detection results to kitti format for evaluation and test submission.

参数

- **net_outputs** (*list[np.ndarray]*) –List of array storing the inferenced bounding boxes and scores.
- **class_names** (*list[String]*) –A list of class names.
- **pklfile_prefix** (*str*) –The prefix of pkl file.
- **submission_prefix** (*str*) –The prefix of submission file.

返回 A list of dictionaries have the kitti format

返回类型 list[dict]

convert_valid_bboxes(*box_dict*, *info*)

Convert the predicted boxes into valid ones.

参数

- **box_dict** (*dict*) –Box dictionaries to be converted.
  - boxes_3d (LiDARInstance3DBoxes): 3D bounding boxes.
  - scores_3d (torch.Tensor): Scores of boxes.
  - labels_3d (torch.Tensor): Class labels of boxes.
- **info** (*dict*) –Data info.

返回

Valid predicted boxes.

- bbox (np.ndarray): 2D bounding boxes.
- **box3d_camera (np.ndarray): 3D bounding boxes in** camera coordinate.
- **box3d_lidar (np.ndarray): 3D bounding boxes in** LiDAR coordinate.
- scores (np.ndarray): Scores of boxes.

- label_preds (np.ndarray): Class label predictions.
- sample_idx (int): Sample index.

返回类型 dict

drop_arrays_by_name(*gt_names*, *used_classes*)

Drop irrelevant ground truths by name.

参数

- **gt_names** (*list[str]*) –Names of ground truths.
- **used_classes** (*list[str]*) –Classes of interest.

返回 Indices of ground truths that will be dropped.

返回类型 np.ndarray

evaluate(*results*, *metric=None*, *logger=None*, *pklfile_prefix=None*, *submission_prefix=None*, *show=False*, *out_dir=None*, *pipeline=None*)

Evaluation in KITTI protocol.

参数

- **results** (*list[dict]*) –Testing results of the dataset.
- **metric** (*str | list[str], optional*) –Metrics to be evaluated. Default: None.
- **logger** (*logging.Logger | str, optional*) –Logger used for printing related information during evaluation. Default: None.
- **pklfile_prefix** (*str, optional*) –The prefix of pkl files, including the file path and the prefix of filename, e.g., "a/b/prefix". If not specified, a temp file will be created. Default: None.
- **submission_prefix** (*str, optional*) –The prefix of submission data. If not specified, the submission data will not be generated. Default: None.
- **show** (*bool, optional*) –Whether to visualize. Default: False.
- **out_dir** (*str, optional*) –Path to save the visualization results. Default: None.
- **pipeline** (*list[dict], optional*) –raw data loading for showing. Default: None.

返回 Results of each evaluation metric.

返回类型 dict[str, float]

format_results(*outputs*, *pklfile_prefix=None*, *submission_prefix=None*)

Format the results to pkl file.

参数

- **outputs** (*list[dict]*) –Testing results of the dataset.

- **pklfile_prefix** (*str*) –The prefix of pkl files. It includes the file path and the prefix of filename, e.g., "a/b/prefix" . If not specified, a temp file will be created. Default: None.
- **submission_prefix** (*str*) –The prefix of submitted files. It includes the file path and the prefix of filename, e.g., "a/b/prefix" . If not specified, a temp file will be created. Default: None.

返回

(result_files, tmp_dir), result_files is a dict containing the json filepaths, tmp_dir is the temporal directory created for saving json files when jsonfile_prefix is not specified.

返回类型 tuple

get_ann_info(*index*)

Get annotation info according to the given index.

参数 **index** (*int*) –Index of the annotation data to get.

返回

annotation information consists of the following keys:

- **gt_bboxes_3d (LiDARInstance3DBoxes):** 3D ground truth bboxes.
- gt_labels_3d (np.ndarray): Labels of ground truths.
- gt_bboxes (np.ndarray): 2D ground truth bboxes.
- gt_labels (np.ndarray): Labels of ground truths.
- gt_names (list[str]): Class names of ground truths.
- **difficulty (int): Difficulty defined by KITTI.** 0, 1, 2 represent xxxxx respectively.

返回类型 dict

get_data_info(*index*)

Get data info according to the given index.

参数 **index** (*int*) –Index of the sample data to get.

返回

Data information that will be passed to the data preprocessing pipelines. It includes the following keys:

- sample_idx (str): Sample index.
- pts_filename (str): Filename of point clouds.
- img_prefix (str): Prefix of image files.
- img_info (dict): Image info.

- **lidar2img (list[np.ndarray], optional): Transformations** from lidar to different cameras.
- ann_info (dict): Annotation info.

返回类型 dict

keep_arrays_by_name(*gt_names*, *used_classes*)

Keep useful ground truths by name.

参数

- **gt_names** (*list[str]*) –Names of ground truths.
- **used_classes** (*list[str]*) –Classes of interest.

返回 Indices of ground truths that will be keeped.

返回类型 np.ndarray

remove_dontcare(*ann_info*)

Remove annotations that do not need to be cared.

参数 **ann_info** (*dict*) –Dict of annotation infos. The `'DontCare'` annotations will be removed according to ann_file['name'].

返回 Annotations after filtering.

返回类型 dict

show(*results*, *out_dir*, *show=True*, *pipeline=None*)

Results visualization.

参数

- **results** (*list[dict]*) –List of bounding boxes results.
- **out_dir** (*str*) –Output directory of visualization result.
- **show** (*bool*) –Whether to visualize the results online. Default: False.
- **pipeline** (*list[dict], optional*) –raw data loading for showing. Default: None.

class mmdet3d.datasets.**KittiMonoDataset**(*data_root*, *info_file*, *ann_file*, *pipeline*, *load_interval=1*, *with_velocity=False*, *eval_version=None*, *version=None*, ***kwargs*)

Monocular 3D detection on KITTI Dataset.

参数

- **data_root** (*str*) –Path of dataset root.
- **info_file** (*str*) –Path of info file.

- **load_interval** (*int, optional*) –Interval of loading the dataset. It is used to uniformly sample the dataset. Defaults to 1.
- **with_velocity** (*bool, optional*) –Whether include velocity prediction into the experiments. Defaults to False.
- **eval_version** (*str, optional*) –Configuration version of evaluation. Defaults to None.
- **version** (*str, optional*) –Dataset version. Defaults to None.
- **kwargs** (*dict*) –Other arguments are the same of NuScenesMonoDataset.

bbox2result_kitti(*net_outputs*, *class_names*, *pklfile_prefix=None*, *submission_prefix=None*)

Convert 3D detection results to kitti format for evaluation and test submission.

参数

- **net_outputs** (*list[np.ndarray]*) –List of array storing the inferenced bounding boxes and scores.
- **class_names** (*list[String]*) –A list of class names.
- **pklfile_prefix** (*str*) –The prefix of pkl file.
- **submission_prefix** (*str*) –The prefix of submission file.

返回 A list of dictionaries with the kitti format.

返回类型 list[dict]

bbox2result_kitti2d(*net_outputs*, *class_names*, *pklfile_prefix=None*, *submission_prefix=None*)

Convert 2D detection results to kitti format for evaluation and test submission.

参数

- **net_outputs** (*list[np.ndarray]*) –List of array storing the inferenced bounding boxes and scores.
- **class_names** (*list[String]*) –A list of class names.
- **pklfile_prefix** (*str*) –The prefix of pkl file.
- **submission_prefix** (*str*) –The prefix of submission file.

返回 A list of dictionaries have the kitti format

返回类型 list[dict]

convert_valid_bboxes(*box_dict*, *info*)

Convert the predicted boxes into valid ones.

参数

- **box_dict** (*dict*) –Box dictionaries to be converted. - boxes_3d (CameraInstance3DBoxes): 3D bounding boxes. - scores_3d (torch.Tensor): Scores of boxes. - labels_3d (torch.Tensor): Class labels of boxes.
- **info** (*dict*) –Data info.

返回

Valid predicted boxes.

- bbox (np.ndarray): 2D bounding boxes.
- **box3d_camera (np.ndarray): 3D bounding boxes in** camera coordinate.
- scores (np.ndarray): Scores of boxes.
- label_preds (np.ndarray): Class label predictions.
- sample_idx (int): Sample index.

返回类型 dict

evaluate(*results*, *metric=None*, *logger=None*, *pklfile_prefix=None*, *submission_prefix=None*, *show=False*, *out_dir=None*, *pipeline=None*)

Evaluation in KITTI protocol.

参数

- **results** (*list[dict]*) –Testing results of the dataset.
- **metric** (*str | list[str], optional*) –Metrics to be evaluated. Defaults to None.
- **logger** (*logging.Logger | str, optional*) –Logger used for printing related information during evaluation. Default: None.
- **pklfile_prefix** (*str, optional*) –The prefix of pkl files, including the file path and the prefix of filename, e.g., "a/b/prefix". If not specified, a temp file will be created. Default: None.
- **submission_prefix** (*str, optional*) –The prefix of submission data. If not specified, the submission data will not be generated.
- **show** (*bool, optional*) –Whether to visualize. Default: False.
- **out_dir** (*str, optional*) –Path to save the visualization results. Default: None.
- **pipeline** (*list[dict], optional*) –raw data loading for showing. Default: None.

返回 Results of each evaluation metric.

返回类型 dict[str, float]

format_results(*outputs*, *pklfile_prefix=None*, *submission_prefix=None*)

Format the results to pkl file.

参数

- **outputs** (*list[dict]*) –Testing results of the dataset.
- **pklfile_prefix** (*str*) –The prefix of pkl files. It includes the file path and the prefix of filename, e.g., "a/b/prefix". If not specified, a temp file will be created. Default: None.
- **submission_prefix** (*str*) –The prefix of submitted files. It includes the file path and the prefix of filename, e.g., "a/b/prefix". If not specified, a temp file will be created. Default: None.

返回

(result_files, tmp_dir), result_files is a dict containing the json filepaths, tmp_dir is the temporal directory created for saving json files when jsonfile_prefix is not specified.

返回类型 tuple

class mmdet3d.datasets.**LoadAnnotations3D**(*with_bbox_3d=True*, *with_label_3d=True*, *with_attr_label=False*, *with_mask_3d=False*, *with_seg_3d=False*, *with_bbox=False*, *with_label=False*, *with_mask=False*, *with_seg=False*, *with_bbox_depth=False*, *poly2mask=True*, *seg_3d_dtype=<class 'numpy.int64'>*, *file_client_args={'backend': 'disk'}*)

Load Annotations3D.

Load instance mask and semantic mask of points and encapsulate the items into related fields.

参数

- **with_bbox_3d** (*bool, optional*) –Whether to load 3D boxes. Defaults to True.
- **with_label_3d** (*bool, optional*) –Whether to load 3D labels. Defaults to True.
- **with_attr_label** (*bool, optional*) –Whether to load attribute label. Defaults to False.
- **with_mask_3d** (*bool, optional*) –Whether to load 3D instance masks. for points. Defaults to False.
- **with_seg_3d** (*bool, optional*) –Whether to load 3D semantic masks. for points. Defaults to False.
- **with_bbox** (*bool, optional*) –Whether to load 2D boxes. Defaults to False.
- **with_label** (*bool, optional*) –Whether to load 2D labels. Defaults to False.

- **with_mask** (*bool, optional*) –Whether to load 2D instance masks. Defaults to False.
- **with_seg** (*bool, optional*) –Whether to load 2D semantic masks. Defaults to False.
- **with_bbox_depth** (*bool, optional*) –Whether to load 2.5D boxes. Defaults to False.
- **poly2mask** (*bool, optional*) –Whether to convert polygon annotations to bitmasks. Defaults to True.
- **seg_3d_dtype** (*dtype, optional*) –Dtype of 3D semantic masks. Defaults to int64
- **file_client_args** (*dict*) –Config dict of file clients, refer to https://github.com/open-mmlab/mmcv/blob/master/mmcv/fileio/file_client.py for more details.

class mmdet3d.datasets.**LoadPointsFromDict**(*coord_type*, *load_dim=6*, *use_dim=[0, 1, 2]*, *shift_height=False*, *use_color=False*, *file_client_args={'backend': 'disk'}*)

Load Points From Dict.

class mmdet3d.datasets.**LoadPointsFromFile**(*coord_type*, *load_dim=6*, *use_dim=[0, 1, 2]*, *shift_height=False*, *use_color=False*, *file_client_args={'backend': 'disk'}*)

Load Points From File.

Load points from file.

参数

- **coord_type** (*str*) –The type of coordinates of points cloud. Available options includes: - 'LIDAR': Points in LiDAR coordinates. - 'DEPTH': Points in depth coordinates, usually for indoor dataset. - 'CAMERA': Points in camera coordinates.
- **load_dim** (*int, optional*) –The dimension of the loaded points. Defaults to 6.
- **use_dim** (*list[int], optional*) –Which dimensions of the points to use. Defaults to [0, 1, 2]. For KITTI dataset, set use_dim=4 or use_dim=[0, 1, 2, 3] to use the intensity dimension.
- **shift_height** (*bool, optional*) –Whether to use shifted height. Defaults to False.
- **use_color** (*bool, optional*) –Whether to use color features. Defaults to False.
- **file_client_args** (*dict, optional*) –Config dict of file clients, refer to https://github.com/open-mmlab/mmcv/blob/master/mmcv/fileio/file_client.py for more details. Defaults to dict(backend='disk').

class mmdet3d.datasets.**LoadPointsFromMultiSweeps**(*sweeps_num=10*, *load_dim=5*, *use_dim=[0, 1, 2, 4]*, *time_dim=4*, *file_client_args={'backend': 'disk'}*, *pad_empty_sweeps=False*, *remove_close=False*, *test_mode=False*)

Load points from multiple sweeps.

This is usually used for nuScenes dataset to utilize previous sweeps.

参数

- **sweeps_num** (*int, optional*) –Number of sweeps. Defaults to 10.
- **load_dim** (*int, optional*) –Dimension number of the loaded points. Defaults to 5.
- **use_dim** (*list[int], optional*) –Which dimension to use. Defaults to [0, 1, 2, 4].
- **time_dim** (*int, optional*) –Which dimension to represent the timestamps of each points. Defaults to 4.
- **file_client_args** (*dict, optional*) –Config dict of file clients, refer to https://github.com/open-mmlab/mmcv/blob/master/mmcv/fileio/file_client.py for more details. Defaults to dict(backend='disk').
- **pad_empty_sweeps** (*bool, optional*) –Whether to repeat keyframe when sweeps is empty. Defaults to False.
- **remove_close** (*bool, optional*) –Whether to remove close points. Defaults to False.
- **test_mode** (*bool, optional*) –If *test_mode=True*, it will not randomly sample sweeps but select the nearest N frames. Defaults to False.

class mmdet3d.datasets.**LyftDataset**(*ann_file*, *pipeline=None*, *data_root=None*, *classes=None*, *load_interval=1*, *modality=None*, *box_type_3d='LiDAR'*, *filter_empty_gt=True*, *test_mode=False*, ***kwargs*)

Lyft Dataset.

This class serves as the API for experiments on the Lyft Dataset.

Please refer to https://www.kaggle.com/c/3d-object-detection-for-autonomous-vehicles/data for data downloading.

参数

- **ann_file** (*str*) –Path of annotation file.
- **pipeline** (*list[dict], optional*) –Pipeline used for data processing. Defaults to None.
- **data_root** (*str*) –Path of dataset root.

- **classes** (*tuple[str], optional*) –Classes used in the dataset. Defaults to None.
- **load_interval** (*int, optional*) –Interval of loading the dataset. It is used to uniformly sample the dataset. Defaults to 1.
- **modality** (*dict, optional*) –Modality to specify the sensor data used as input. Defaults to None.
- **box_type_3d** (*str, optional*) –Type of 3D box of this dataset. Based on the *box_type_3d*, the dataset will encapsulate the box to its original format then converted them to *box_type_3d*. Defaults to 'LiDAR' in this dataset. Available options includes
  - 'LiDAR': Box in LiDAR coordinates.
  - 'Depth': Box in depth coordinates, usually for indoor dataset.
  - 'Camera': Box in camera coordinates.
- **filter_empty_gt** (*bool, optional*) –Whether to filter empty GT. Defaults to True.
- **test_mode** (*bool, optional*) –Whether the dataset is in test mode. Defaults to False.

evaluate(*results*, *metric='bbox'*, *logger=None*, *jsonfile_prefix=None*, *csv_savepath=None*, *result_names=['pts_bbox']*, *show=False*, *out_dir=None*, *pipeline=None*)

Evaluation in Lyft protocol.

参数

- **results** (*list[dict]*) –Testing results of the dataset.
- **metric** (*str | list[str], optional*) –Metrics to be evaluated. Default: 'bbox'.
- **logger** (*logging.Logger | str, optional*) –Logger used for printing related information during evaluation. Default: None.
- **jsonfile_prefix** (*str, optional*) –The prefix of json files including the file path and the prefix of filename, e.g., "a/b/prefix". If not specified, a temp file will be created. Default: None.
- **csv_savepath** (*str, optional*) –The path for saving csv files. It includes the file path and the csv filename, e.g., "a/b/filename.csv". If not specified, the result will not be converted to csv file.
- **result_names** (*list[str], optional*) –Result names in the metric prefix. Default: ['pts_bbox'].
- **show** (*bool, optional*) –Whether to visualize. Default: False.
- **out_dir** (*str, optional*) –Path to save the visualization results. Default: None.

- **pipeline** (*list[dict], optional*) –raw data loading for showing. Default: None.

返回 Evaluation results.

返回类型 dict[str, float]

format_results(*results*, *jsonfile_prefix=None*, *csv_savepath=None*)

Format the results to json (standard format for COCO evaluation).

参数

- **results** (*list[dict]*) –Testing results of the dataset.
- **jsonfile_prefix** (*str*) –The prefix of json files. It includes the file path and the prefix of filename, e.g., "a/b/prefix". If not specified, a temp file will be created. Default: None.
- **csv_savepath** (*str*) –The path for saving csv files. It includes the file path and the csv filename, e.g., "a/b/filename.csv". If not specified, the result will not be converted to csv file.

返回

Returns (result_files, tmp_dir), where *result_files* is a dict containing the json filepaths, *tmp_dir* is the temporal directory created for saving json files when *jsonfile_prefix* is not specified.

返回类型 tuple

get_ann_info(*index*)

Get annotation info according to the given index.

参数 **index** (*int*) –Index of the annotation data to get.

返回

Annotation information consists of the following keys:

- **gt_bboxes_3d (LiDARInstance3DBoxes):** 3D ground truth bboxes.
- gt_labels_3d (np.ndarray): Labels of ground truths.
- gt_names (list[str]): Class names of ground truths.

返回类型 dict

get_data_info(*index*)

Get data info according to the given index.

参数 **index** (*int*) –Index of the sample data to get.

返回

Data information that will be passed to the data preprocessing pipelines. It includes the following keys:

- sample_idx (str): sample index
- pts_filename (str): filename of point clouds
- sweeps (list[dict]): infos of sweeps
- timestamp (float): sample timestamp
- img_filename (str, optional): image filename
- **lidar2img (list[np.ndarray], optional): transformations** from lidar to different cameras
- ann_info (dict): annotation info

返回类型 dict

json2csv(*json_path*, *csv_savepath*)

Convert the json file to csv format for submission.

参数

- **json_path** (*str*) –Path of the result json file.
- **csv_savepath** (*str*) –Path to save the csv file.

load_annotations(*ann_file*)

Load annotations from ann_file.

参数 **ann_file** (*str*) –Path of the annotation file.

返回 List of annotations sorted by timestamps.

返回类型 list[dict]

show(*results*, *out_dir*, *show=False*, *pipeline=None*)

Results visualization.

参数

- **results** (*list[dict]*) –List of bounding boxes results.
- **out_dir** (*str*) –Output directory of visualization result.
- **show** (*bool*) –Whether to visualize the results online. Default: False.
- **pipeline** (*list[dict], optional*) –raw data loading for showing. Default: None.

class mmdet3d.datasets.**MultiViewWrapper**(*transforms*, *process_fields={'img_fields': ['img']}*, *collected_keys=[]*)

Wrap transformation from single-view into multi-view.

The wrapper processes the images from multi-view one by one. For each image, it constructs a pseudo dict according to the keys specified by the ‘process_fields’ parameter. After the transformation is finished, desired information can be collected by specifying the keys in the ‘collected_keys’ parameter. Multi-view images share the same transformation parameters but do not share the same magnitude when a random transformation is conducted.

参数

- **transforms** (*list[dict]*) –A list of dict specifying the transformations for the monocular situation.
- **process_fields** (*dict*) –Desired keys that the transformations should be conducted on. Default to dict(img_fields=[‘img’]).
- **collected_keys** (*list[str]*) –Collect information in transformation like rotate angles, crop roi, and flip state.

class mmdet3d.datasets.**NormalizePointsColor**(*color_mean*)

Normalize color of points.

参数 **color_mean** (*list[float]*) –Mean color of the point cloud.

class mmdet3d.datasets.**NuScenesDataset**(*ann_file*, *pipeline=None*, *data_root=None*, *classes=None*, *load_interval=1*, *with_velocity=True*, *modality=None*, *box_type_3d='LiDAR'*, *filter_empty_gt=True*, *test_mode=False*, *eval_version='detection_cvpr_2019'*, *use_valid_flag=False*)

NuScenes Dataset.

This class serves as the API for experiments on the NuScenes Dataset.

Please refer to NuScenes Dataset for data downloading.

参数

- **ann_file** (*str*) –Path of annotation file.
- **pipeline** (*list[dict], optional*) –Pipeline used for data processing. Defaults to None.
- **data_root** (*str*) –Path of dataset root.
- **classes** (*tuple[str], optional*) –Classes used in the dataset. Defaults to None.
- **load_interval** (*int, optional*) –Interval of loading the dataset. It is used to uniformly sample the dataset. Defaults to 1.
- **with_velocity** (*bool, optional*) –Whether include velocity prediction into the experiments. Defaults to True.
- **modality** (*dict, optional*) –Modality to specify the sensor data used as input. Defaults to None.

- **box_type_3d** (*str, optional*) –Type of 3D box of this dataset. Based on the *box_type_3d*, the dataset will encapsulate the box to its original format then converted them to *box_type_3d*. Defaults to ‘LiDAR’ in this dataset. Available options includes. - ‘LiDAR’: Box in LiDAR coordinates. - ‘Depth’: Box in depth coordinates, usually for indoor dataset. - ‘Camera’: Box in camera coordinates.
- **filter_empty_gt** (*bool, optional*) –Whether to filter empty GT. Defaults to True.
- **test_mode** (*bool, optional*) –Whether the dataset is in test mode. Defaults to False.
- **eval_version** (*bool, optional*) –Configuration version of evaluation. Defaults to ‘detection_cvpr_2019’.
- **use_valid_flag** (*bool, optional*) –Whether to use *use_valid_flag* key in the info file as mask to filter gt_boxes and gt_names. Defaults to False.

evaluate(*results*, *metric='bbox'*, *logger=None*, *jsonfile_prefix=None*, *result_names=['pts_bbox']*, *show=False*, *out_dir=None*, *pipeline=None*)

Evaluation in nuScenes protocol.

参数

- **results** (*list[dict]*) –Testing results of the dataset.
- **metric** (*str | list[str], optional*) –Metrics to be evaluated. Default: ‘bbox’.
- **logger** (*logging.Logger | str, optional*) –Logger used for printing related information during evaluation. Default: None.
- **jsonfile_prefix** (*str, optional*) –The prefix of json files including the file path and the prefix of filename, e.g., “a/b/prefix”. If not specified, a temp file will be created. Default: None.
- **show** (*bool, optional*) –Whether to visualize. Default: False.
- **out_dir** (*str, optional*) –Path to save the visualization results. Default: None.
- **pipeline** (*list[dict], optional*) –raw data loading for showing. Default: None.

返回 Results of each evaluation metric.

返回类型 dict[str, float]

format_results(*results*, *jsonfile_prefix=None*)

Format the results to json (standard format for COCO evaluation).

参数

- **results** (*list[dict]*) –Testing results of the dataset.

- **jsonfile_prefix** (*str*) –The prefix of json files. It includes the file path and the prefix of filename, e.g., "a/b/prefix". If not specified, a temp file will be created. Default: None.

返回

Returns (result_files, tmp_dir), where *result_files* is a dict containing the json filepaths, *tmp_dir* is the temporal directory created for saving json files when *jsonfile_prefix* is not specified.

返回类型 tuple

get_ann_info(*index*)

Get annotation info according to the given index.

参数 **index** (*int*) –Index of the annotation data to get.

返回

Annotation information consists of the following keys:

- **gt_bboxes_3d (LiDARInstance3DBoxes):** 3D ground truth bboxes
- gt_labels_3d (np.ndarray): Labels of ground truths.
- gt_names (list[str]): Class names of ground truths.

返回类型 dict

get_cat_ids(*idx*)

Get category distribution of single scene.

参数 **idx** (*int*) –Index of the data_info.

返回

for each category, if the current scene contains such boxes, store a list containing idx, otherwise, store empty list.

返回类型 dict[list]

get_data_info(*index*)

Get data info according to the given index.

参数 **index** (*int*) –Index of the sample data to get.

返回

Data information that will be passed to the data preprocessing pipelines. It includes the following keys:

- sample_idx (str): Sample index.
- pts_filename (str): Filename of point clouds.
- sweeps (list[dict]): Infos of sweeps.

- timestamp (float): Sample timestamp.
- img_filename (str, optional): Image filename.
- **lidar2img (list[np.ndarray], optional): Transformations** from lidar to different cameras.
- ann_info (dict): Annotation info.

返回类型 dict

load_annotations(*ann_file*)

Load annotations from ann_file.

参数 **ann_file** (*str*) –Path of the annotation file.

返回 List of annotations sorted by timestamps.

返回类型 list[dict]

show(*results*, *out_dir*, *show=False*, *pipeline=None*)

Results visualization.

参数

- **results** (*list[dict]*) –List of bounding boxes results.
- **out_dir** (*str*) –Output directory of visualization result.
- **show** (*bool*) –Whether to visualize the results online. Default: False.
- **pipeline** (*list[dict], optional*) –raw data loading for showing. Default: None.

class mmdet3d.datasets.**NuScenesMonoDataset**(*data_root*, *ann_file*, *pipeline*, *load_interval=1*, *with_velocity=True*, *modality=None*, *box_type_3d='Camera'*, *eval_version='detection_cvpr_2019'*, *use_valid_flag=False*, *version='v1.0-trainval'*, *classes=None*, *img_prefix=''*, *seg_prefix=None*, *proposal_file=None*, *test_mode=False*, *filter_empty_gt=True*, *file_client_args={'backend': 'disk'}*)

Monocular 3D detection on NuScenes Dataset.

This class serves as the API for experiments on the NuScenes Dataset.

Please refer to NuScenes Dataset for data downloading.

参数

- **ann_file** (*str*) –Path of annotation file.
- **data_root** (*str*) –Path of dataset root.

- **load_interval** (*int, optional*) –Interval of loading the dataset. It is used to uniformly sample the dataset. Defaults to 1.
- **with_velocity** (*bool, optional*) –Whether include velocity prediction into the experiments. Defaults to True.
- **modality** (*dict, optional*) –Modality to specify the sensor data used as input. Defaults to None.
- **box_type_3d** (*str, optional*) –Type of 3D box of this dataset. Based on the *box_type_3d*, the dataset will encapsulate the box to its original format then converted them to *box_type_3d*. Defaults to 'Camera' in this class. Available options includes. - 'LiDAR' : Box in LiDAR coordinates. - 'Depth' : Box in depth coordinates, usually for indoor dataset. - 'Camera' : Box in camera coordinates.
- **eval_version** (*str, optional*) –Configuration version of evaluation. Defaults to 'detection_cvpr_2019'.
- **use_valid_flag** (*bool, optional*) –Whether to use *use_valid_flag* key in the info file as mask to filter gt_boxes and gt_names. Defaults to False.
- **version** (*str, optional*) –Dataset version. Defaults to 'v1.0-trainval'.

evaluate (*results*, *metric='bbox'*, *logger=None*, *jsonfile_prefix=None*, *result_names=['img_bbox']*, *show=False*, *out_dir=None*, *pipeline=None*)

Evaluation in nuScenes protocol.

参数

- **results** (*list[dict]*) –Testing results of the dataset.
- **metric** (*str | list[str], optional*) –Metrics to be evaluated. Default: 'bbox'.
- **logger** (*logging.Logger | str, optional*) –Logger used for printing related information during evaluation. Default: None.
- **jsonfile_prefix** (*str*) –The prefix of json files. It includes the file path and the prefix of filename, e.g., "a/b/prefix". If not specified, a temp file will be created. Default: None.
- **result_names** (*list[str], optional*) –Result names in the metric prefix. Default: ['img_bbox'].
- **show** (*bool, optional*) –Whether to visualize. Default: False.
- **out_dir** (*str, optional*) –Path to save the visualization results. Default: None.
- **pipeline** (*list[dict], optional*) –raw data loading for showing. Default: None.

返回 Results of each evaluation metric.

返回类型 dict[str, float]

format_results(*results*, *jsonfile_prefix=None*, ***kwargs*)

Format the results to json (standard format for COCO evaluation).

参数

- **results** (*list[tuple | numpy.ndarray]*) –Testing results of the dataset.
- **jsonfile_prefix** (*str*) –The prefix of json files. It includes the file path and the prefix of filename, e.g., "a/b/prefix". If not specified, a temp file will be created. Default: None.

返回

(result_files, tmp_dir), result_files is a dict containing the json filepaths, tmp_dir is the temporal directory created for saving json files when jsonfile_prefix is not specified.

返回类型 tuple

get_attr_name(*attr_idx*, *label_name*)

Get attribute from predicted index.

This is a workaround to predict attribute when the predicted velocity is not reliable. We map the predicted attribute index to the one in the attribute set. If it is consistent with the category, we will keep it. Otherwise, we will use the default attribute.

参数

- **attr_idx** (*int*) –Attribute index.
- **label_name** (*str*) –Predicted category name.

返回 Predicted attribute name.

返回类型 str

pre_pipeline(*results*)

Initialization before data preparation.

参数 **results** (*dict*) –Dict before data preprocessing.

- img_fields (list): Image fields.
- bbox3d_fields (list): 3D bounding boxes fields.
- pts_mask_fields (list): Mask fields of points.
- pts_seg_fields (list): Mask fields of point segments.
- bbox_fields (list): Fields of bounding boxes.
- mask_fields (list): Fields of masks.
- seg_fields (list): Segment fields.

- box_type_3d (str): 3D box type.
- box_mode_3d (str): 3D box mode.

show(*results*, *out_dir*, *show=False*, *pipeline=None*)

Results visualization.

参数

- **results** (*list[dict]*) –List of bounding boxes results.
- **out_dir** (*str*) –Output directory of visualization result.
- **show** (*bool*) –Whether to visualize the results online. Default: False.
- **pipeline** (*list[dict], optional*) –raw data loading for showing. Default: None.

class mmdet3d.datasets.**ObjectNameFilter**(*classes*)

Filter GT objects by their names.

参数 **classes** (*list[str]*) –List of class names to be kept for training.

class mmdet3d.datasets.**ObjectNoise**(*translation_std=[0.25, 0.25, 0.25]*, *global_rot_range=[0.0, 0.0]*, *rot_range=[- 0.15707963267, 0.15707963267]*, *num_try=100*)

Apply noise to each GT objects in the scene.

参数

- **translation_std** (*list[float], optional*) –Standard deviation of the distribution where translation noise are sampled from. Defaults to [0.25, 0.25, 0.25].
- **global_rot_range** (*list[float], optional*) –Global rotation to the scene. Defaults to [0.0, 0.0].
- **rot_range** (*list[float], optional*) –Object rotation range. Defaults to [-0.15707963267, 0.15707963267].
- **num_try** (*int, optional*) –Number of times to try if the noise applied is invalid. Defaults to 100.

class mmdet3d.datasets.**ObjectRangeFilter**(*point_cloud_range*)

Filter objects by the range.

参数 **point_cloud_range** (*list[float]*) –Point cloud range.

class mmdet3d.datasets.**ObjectSample**(*db_sampler*, *sample_2d=False*, *use_ground_plane=False*)

Sample GT objects to the data.

参数

- **db_sampler** (*dict*) –Config dict of the database sampler.

- **sample_2d** (*bool*) –Whether to also paste 2D image patch to the images This should be true when applying multi-modality cut-and-paste. Defaults to False.
- **use_ground_plane** (*bool*) –Whether to use gound plane to adjust the 3D labels.

static remove_points_in_boxes(*points*, *boxes*)

Remove the points in the sampled bounding boxes.

参数

- **points** (BasePoints) –Input point cloud array.
- **boxes** (*np.ndarray*) –Sampled ground truth boxes.

返回 Points with those in the boxes removed.

返回类型 np.ndarray

class mmdet3d.datasets.**PointSample**(*num_points*, *sample_range=None*, *replace=False*)

Point sample.

Sampling data to a certain number.

参数

- **num_points** (*int*) –Number of points to be sampled.
- **sample_range** (*float, optional*) –The range where to sample points. If not None, the points with depth larger than *sample_range* are prior to be sampled. Defaults to None.
- **replace** (*bool, optional*) –Whether the sampling is with or without replacement. Defaults to False.

class mmdet3d.datasets.**PointShuffle**

Shuffle input points.

class mmdet3d.datasets.**PointsRangeFilter**(*point_cloud_range*)

Filter points by the range.

参数 **point_cloud_range** (*list[float]*) –Point cloud range.

class mmdet3d.datasets.**RandomDropPointsColor**(*drop_ratio=0.2*)

Randomly set the color of points to all zeros.

Once this transform is executed, all the points' color will be dropped. Refer to PAConv for more details.

参数 **drop_ratio** (*float, optional*) –The probability of dropping point colors. Defaults to 0.2.

class mmdet3d.datasets.**RandomFlip3D**(*sync_2d=True*, *flip_ratio_bev_horizontal=0.0*, *flip_ratio_bev_vertical=0.0*, ***kwargs*)

Flip the points & bbox.

If the input dict contains the key "flip", then the flag will be used, otherwise it will be randomly decided by a ratio specified in the init method.

参数

- **sync_2d** (*bool, optional*) –Whether to apply flip according to the 2D images. If True, it will apply the same flip as that to 2D images. If False, it will decide whether to flip randomly and independently to that of 2D images. Defaults to True.
- **flip_ratio_bev_horizontal** (*float, optional*) –The flipping probability in horizontal direction. Defaults to 0.0.
- **flip_ratio_bev_vertical** (*float, optional*) –The flipping probability in vertical direction. Defaults to 0.0.

random_flip_data_3d(*input_dict*, *direction='horizontal'*)

Flip 3D data randomly.

参数

- **input_dict** (*dict*) –Result dict from loading pipeline.
- **direction** (*str, optional*) –Flip direction. Default: 'horizontal'.

返回

Flipped results, 'points', 'bbox3d_fields' keys are updated in the result dict.

返回类型 dict

class mmdet3d.datasets.**RandomJitterPoints**(*jitter_std=[0.01, 0.01, 0.01]*, *clip_range=[- 0.05, 0.05]*)

Randomly jitter point coordinates.

Different from the global translation in GlobalRotScaleTrans, here we apply different noises to each point in a scene.

参数

- **jitter_std** (*list[float]*) –The standard deviation of jittering noise. This applies random noise to all points in a 3D scene, which is sampled from a gaussian distribution whose standard deviation is set by `jitter_std`. Defaults to [0.01, 0.01, 0.01]
- **clip_range** (*list[float]*) –Clip the randomly generated jitter noise into this range. If None is given, don't perform clipping. Defaults to [-0.05, 0.05]

注解:

This transform should only be used in point cloud segmentation tasks because we don't transform ground-truth bboxes accordingly.

For similar transform in detection task, please refer to *ObjectNoise*.

---

class mmdet3d.datasets.**RandomRotate**(*range*, ***kwargs*)

Randomly rotate images.

The ratation angle is selected uniformly within the interval specified by the 'range' parameter.

参数 **range** (*tuple[float]*) –Define the range of random rotation. (angle_min, angle_max) in angle.

class mmdet3d.datasets.**RandomShiftScale**(*shift_scale*, *aug_prob*)

Random shift scale.

Different from the normal shift and scale function, it doesn't directly shift or scale image. It can record the shift and scale infos into loading pipelines. It's designed to be used with AffineResize together.

参数

- **shift_scale** (*tuple[float]*) –Shift and scale range.
- **aug_prob** (*float*) –The shifting and scaling probability.

class mmdet3d.datasets.**RangeLimitedRandomCrop**(*relative_x_offset_range=(0.0, 1.0)*, *relative_y_offset_range=(0.0, 1.0)*, ***kwargs*)

Randomly crop image-view objects under a limitation of range.

参数

- **relative_x_offset_range** (*tuple[float]*) –Relative range of random crop in x direction. (x_min, x_max) in [0, 1.0]. Default to (0.0, 1.0).
- **relative_y_offset_range** (*tuple[float]*) –Relative range of random crop in y direction. (y_min, y_max) in [0, 1.0]. Default to (0.0, 1.0).

class mmdet3d.datasets.**S3DISDataset**(*data_root*, *ann_file*, *pipeline=None*, *classes=None*, *modality=None*, *box_type_3d='Depth'*, *filter_empty_gt=True*, *test_mode=False*, **kwargs*)

S3DIS Dataset for Detection Task.

This class is the inner dataset for S3DIS. Since S3DIS has 6 areas, we often train on 5 of them and test on the remaining one. The one for test is Area_5 as suggested in GSDN. To concatenate 5 areas during training *mmdet.datasets.dataset_wrappers.ConcatDataset* should be used.

参数

- **data_root** (*str*) –Path of dataset root.
- **ann_file** (*str*) –Path of annotation file.
- **pipeline** (*list[dict], optional*) –Pipeline used for data processing. Defaults to None.

- **classes** (*tuple[str], optional*) –Classes used in the dataset. Defaults to None.
- **modality** (*dict, optional*) –Modality to specify the sensor data used as input. Defaults to None.
- **box_type_3d** (*str, optional*) –Type of 3D box of this dataset. Based on the *box_type_3d*, the dataset will encapsulate the box to its original format then converted them to *box_type_3d*. Defaults to 'Depth' in this dataset. Available options includes
  - 'LiDAR': Box in LiDAR coordinates.
  - 'Depth': Box in depth coordinates, usually for indoor dataset.
  - 'Camera': Box in camera coordinates.
- **filter_empty_gt** (*bool, optional*) –Whether to filter empty GT. Defaults to True.
- **test_mode** (*bool, optional*) –Whether the dataset is in test mode. Defaults to False.

get_ann_info(*index*)

Get annotation info according to the given index.

参数 **index** (*int*) –Index of the annotation data to get.

返回

annotation information consists of the following keys:

- **gt_bboxes_3d (DepthInstance3DBoxes):** 3D ground truth bboxes
- gt_labels_3d (np.ndarray): Labels of ground truths.
- pts_instance_mask_path (str): Path of instance masks.
- pts_semantic_mask_path (str): Path of semantic masks.

返回类型 dict

get_data_info(*index*)

Get data info according to the given index.

参数 **index** (*int*) –Index of the sample data to get.

返回

Data information that will be passed to the data preprocessing pipelines. It includes the following keys:

- pts_filename (str): Filename of point clouds.
- file_name (str): Filename of point clouds.
- ann_info (dict): Annotation info.

返回类型 dict

class mmdet3d.datasets.**S3DISSegDataset**(*data_root*, *ann_files*, *pipeline=None*, *classes=None*, *palette=None*, *modality=None*, *test_mode=False*, *ignore_index=None*, *scene_idxs=None*, ***kwargs*)

S3DIS Dataset for Semantic Segmentation Task.

This class serves as the API for experiments on the S3DIS Dataset. It wraps the provided datasets of different areas. We don't use *mmdet.datasets.dataset_wrappers.ConcatDataset* because we need to concat the *scene_idxs* of different areas.

Please refer to the google form for data downloading.

参数

- **data_root** (*str*) –Path of dataset root.
- **ann_files** (*list[str]*) –Path of several annotation files.
- **pipeline** (*list[dict], optional*) –Pipeline used for data processing. Defaults to None.
- **classes** (*tuple[str], optional*) –Classes used in the dataset. Defaults to None.
- **palette** (*list[list[int]], optional*) –The palette of segmentation map. Defaults to None.
- **modality** (*dict, optional*) –Modality to specify the sensor data used as input. Defaults to None.
- **test_mode** (*bool, optional*) –Whether the dataset is in test mode. Defaults to False.
- **ignore_index** (*int, optional*) –The label index to be ignored, e.g. unannotated points. If None is given, set to len(self.CLASSES). Defaults to None.
- **scene_idxs** (*list[np.ndarray] | list[str], optional*) –Precomputed index to load data. For scenes with many points, we may sample it several times. Defaults to None.

concat_data_infos(*data_infos*)

Concat data_infos from several datasets to form self.data_infos.

参数 **data_infos** (*list[list[dict]]*) –

concat_scene_idxs(*scene_idxs*)

Concat scene_idxs from several datasets to form self.scene_idxs.

Needs to manually add offset to scene_idxs[1, 2, ···].

参数 **scene_idxs** (*list[np.ndarray]*) –

class mmdet3d.datasets.**SUNRGBDDataset**(*data_root*, *ann_file*, *pipeline=None*, *classes=None*, *modality={'use_camera': True, 'use_lidar': True}*, *box_type_3d='Depth'*, *filter_empty_gt=True*, *test_mode=False*, ***kwargs*)

SUNRGBD Dataset.

This class serves as the API for experiments on the SUNRGBD Dataset.

See the download page for data downloading.

参数

- **data_root** (*str*) –Path of dataset root.
- **ann_file** (*str*) –Path of annotation file.
- **pipeline** (*list[dict], optional*) –Pipeline used for data processing. Defaults to None.
- **classes** (*tuple[str], optional*) –Classes used in the dataset. Defaults to None.
- **modality** (*dict, optional*) –Modality to specify the sensor data used as input. Defaults to None.
- **box_type_3d** (*str, optional*) –Type of 3D box of this dataset. Based on the *box_type_3d*, the dataset will encapsulate the box to its original format then converted them to *box_type_3d*. Defaults to 'Depth' in this dataset. Available options includes
  - 'LiDAR': Box in LiDAR coordinates.
  - 'Depth': Box in depth coordinates, usually for indoor dataset.
  - 'Camera': Box in camera coordinates.
- **filter_empty_gt** (*bool, optional*) –Whether to filter empty GT. Defaults to True.
- **test_mode** (*bool, optional*) –Whether the dataset is in test mode. Defaults to False.

evaluate(*results*, *metric=None*, *iou_thr=(0.25, 0.5)*, *iou_thr_2d=(0.5)*, *logger=None*, *show=False*, *out_dir=None*, *pipeline=None*)

Evaluate.

Evaluation in indoor protocol.

参数

- **results** (*list[dict]*) –List of results.
- **metric** (*str | list[str], optional*) –Metrics to be evaluated. Default: None.

- **iou_thr** (*list[float], optional*) –AP IoU thresholds for 3D evaluation. Default: (0.25, 0.5).
- **iou_thr_2d** (*list[float], optional*) –AP IoU thresholds for 2D evaluation. Default: (0.5,).
- **show** (*bool, optional*) –Whether to visualize. Default: False.
- **out_dir** (*str, optional*) –Path to save the visualization results. Default: None.
- **pipeline** (*list[dict], optional*) –raw data loading for showing. Default: None.

返回 Evaluation results.

返回类型 dict

get_ann_info(*index*)

Get annotation info according to the given index.

参数 **index** (*int*) –Index of the annotation data to get.

返回

annotation information consists of the following keys:

- **gt_bboxes_3d (DepthInstance3DBoxes):** 3D ground truth bboxes
- gt_labels_3d (np.ndarray): Labels of ground truths.
- pts_instance_mask_path (str): Path of instance masks.
- pts_semantic_mask_path (str): Path of semantic masks.

返回类型 dict

get_data_info(*index*)

Get data info according to the given index.

参数 **index** (*int*) –Index of the sample data to get.

返回

Data information that will be passed to the data preprocessing pipelines. It includes the following keys:

- sample_idx (str): Sample index.
- pts_filename (str, optional): Filename of point clouds.
- file_name (str, optional): Filename of point clouds.
- img_prefix (str, optional): Prefix of image files.
- img_info (dict, optional): Image info.
- calib (dict, optional): Camera calibration info.

- ann_info (dict): Annotation info.

返回类型 dict

show(*results*, *out_dir*, *show=True*, *pipeline=None*)

Results visualization.

参数

- **results** (*list[dict]*) –List of bounding boxes results.
- **out_dir** (*str*) –Output directory of visualization result.
- **show** (*bool*) –Visualize the results online.
- **pipeline** (*list[dict], optional*) –raw data loading for showing. Default: None.

class mmdet3d.datasets.**ScanNetDataset**(*data_root*, *ann_file*, *pipeline=None*, *classes=None*, *modality={'use_camera': False, 'use_depth': True}*, *box_type_3d='Depth'*, *filter_empty_gt=True*, *test_mode=False*, ***kwargs*)

ScanNet Dataset for Detection Task.

This class serves as the API for experiments on the ScanNet Dataset.

Please refer to the github repo for data downloading.

参数

- **data_root** (*str*) –Path of dataset root.
- **ann_file** (*str*) –Path of annotation file.
- **pipeline** (*list[dict], optional*) –Pipeline used for data processing. Defaults to None.
- **classes** (*tuple[str], optional*) –Classes used in the dataset. Defaults to None.
- **modality** (*dict, optional*) –Modality to specify the sensor data used as input. Defaults to None.
- **box_type_3d** (*str, optional*) –Type of 3D box of this dataset. Based on the *box_type_3d*, the dataset will encapsulate the box to its original format then converted them to *box_type_3d*. Defaults to ‘Depth’ in this dataset. Available options includes
  - ’LiDAR’: Box in LiDAR coordinates.
  - ’Depth’: Box in depth coordinates, usually for indoor dataset.
  - ’Camera’: Box in camera coordinates.
- **filter_empty_gt** (*bool, optional*) –Whether to filter empty GT. Defaults to True.

- **test_mode** (*bool, optional*) –Whether the dataset is in test mode. Defaults to False.

get_ann_info(*index*)

Get annotation info according to the given index.

参数 **index** (*int*) –Index of the annotation data to get.

返回

annotation information consists of the following keys:

- **gt_bboxes_3d (DepthInstance3DBoxes):** 3D ground truth bboxes
- gt_labels_3d (np.ndarray): Labels of ground truths.
- pts_instance_mask_path (str): Path of instance masks.
- pts_semantic_mask_path (str): Path of semantic masks.
- **axis_align_matrix (np.ndarray): Transformation matrix for** global scene alignment.

返回类型 dict

get_data_info(*index*)

Get data info according to the given index.

参数 **index** (*int*) –Index of the sample data to get.

返回

Data information that will be passed to the data preprocessing pipelines. It includes the following keys:

- sample_idx (str): Sample index.
- pts_filename (str): Filename of point clouds.
- file_name (str): Filename of point clouds.
- img_prefix (str, optional): Prefix of image files.
- img_info (dict, optional): Image info.
- ann_info (dict): Annotation info.

返回类型 dict

prepare_test_data(*index*)

Prepare data for testing.

We should take axis_align_matrix from self.data_infos since we need to align point clouds.

参数 **index** (*int*) –Index for accessing the target data.

返回 Testing data dict of the corresponding index.

返回类型 dict

show(*results*, *out_dir*, *show=True*, *pipeline=None*)

Results visualization.

参数

- **results** (*list[dict]*) –List of bounding boxes results.
- **out_dir** (*str*) –Output directory of visualization result.
- **show** (*bool*) –Visualize the results online.
- **pipeline** (*list[dict], optional*) –raw data loading for showing. Default: None.

class mmdet3d.datasets.**ScanNetInstanceSegDataset**(*data_root*, *ann_file*, *pipeline=None*, *classes=None*, *palette=None*, *modality=None*, *test_mode=False*, *ignore_index=None*, *scene_idxs=None*, *file_client_args={'backend': 'disk'}*)

evaluate(*results*, *metric=None*, *options=None*, *logger=None*, *show=False*, *out_dir=None*, *pipeline=None*)

Evaluation in instance segmentation protocol.

参数

- **results** (*list[dict]*) –List of results.
- **metric** (*str | list[str]*) –Metrics to be evaluated.
- **options** (*dict, optional*) –options for instance_seg_eval.
- **logger** (*logging.Logger | None | str*) –Logger used for printing related information during evaluation. Defaults to None.
- **show** (*bool, optional*) –Whether to visualize. Defaults to False.
- **out_dir** (*str, optional*) –Path to save the visualization results. Defaults to None.
- **pipeline** (*list[dict], optional*) –raw data loading for showing. Default: None.

返回 Evaluation results.

返回类型 dict

get_ann_info(*index*)

Get annotation info according to the given index.

参数 **index** (*int*) –Index of the annotation data to get.

返回

annotation information consists of the following keys:

- pts_semantic_mask_path (str): Path of semantic masks.
- pts_instance_mask_path (str): Path of instance masks.

返回类型 dict

get_classes_and_palette(*classes=None*, *palette=None*)

Get class names of current dataset. Palette is simply ignored for instance segmentation.

参数

- **classes** (*Sequence[str] | str | None*) –If classes is None, use default CLASSES defined by builtin dataset. If classes is a string, take it as a file name. The file contains the name of classes where each line contains one class name. If classes is a tuple or list, override the CLASSES defined by the dataset. Defaults to None.
- **palette** (*Sequence[Sequence[int]] | np.ndarray | None*) –The palette of segmentation map. If None is given, random palette will be generated. Defaults to None.

class mmdet3d.datasets.**ScanNetSegDataset**(*data_root*, *ann_file*, *pipeline=None*, *classes=None*, *palette=None*, *modality=None*, *test_mode=False*, *ignore_index=None*, *scene_idxs=None*, ***kwargs*)

ScanNet Dataset for Semantic Segmentation Task.

This class serves as the API for experiments on the ScanNet Dataset.

Please refer to the github repo for data downloading.

参数

- **data_root** (*str*) –Path of dataset root.
- **ann_file** (*str*) –Path of annotation file.
- **pipeline** (*list[dict], optional*) –Pipeline used for data processing. Defaults to None.
- **classes** (*tuple[str], optional*) –Classes used in the dataset. Defaults to None.
- **palette** (*list[list[int]], optional*) –The palette of segmentation map. Defaults to None.
- **modality** (*dict, optional*) –Modality to specify the sensor data used as input. Defaults to None.
- **test_mode** (*bool, optional*) –Whether the dataset is in test mode. Defaults to False.

- **ignore_index** (*int, optional*) –The label index to be ignored, e.g. unannotated points. If None is given, set to len(self.CLASSES). Defaults to None.
- **scene_idxs** (*np.ndarray | str, optional*) –Precomputed index to load data. For scenes with many points, we may sample it several times. Defaults to None.

format_results(*results*, *txtfile_prefix=None*)

Format the results to txt file. Refer to ScanNet documentation.

参数

- **outputs** (*list[dict]*) –Testing results of the dataset.
- **txtfile_prefix** (*str*) –The prefix of saved files. It includes the file path and the prefix of filename, e.g., "a/b/prefix". If not specified, a temp file will be created. Default: None.

返回

(outputs, tmp_dir), outputs is the detection results, tmp_dir is the temporal directory created for saving submission files when `submission_prefix` is not specified.

返回类型 tuple

get_ann_info(*index*)

Get annotation info according to the given index.

参数 **index** (*int*) –Index of the annotation data to get.

返回

annotation information consists of the following keys:

- pts_semantic_mask_path (str): Path of semantic masks.

返回类型 dict

get_scene_idxs(*scene_idxs*)

Compute scene_idxs for data sampling.

We sample more times for scenes with more points.

show(*results*, *out_dir*, *show=True*, *pipeline=None*)

Results visualization.

参数

- **results** (*list[dict]*) –List of bounding boxes results.
- **out_dir** (*str*) –Output directory of visualization result.
- **show** (*bool*) –Visualize the results online.
- **pipeline** (*list[dict], optional*) –raw data loading for showing. Default: None.

class mmdet3d.datasets.**SemanticKITTIDataset**(*data_root*, *ann_file*, *pipeline=None*, *classes=None*, *modality=None*, *box_type_3d='Lidar'*, *filter_empty_gt=False*, *test_mode=False*)

SemanticKITTI Dataset.

This class serves as the API for experiments on the SemanticKITTI Dataset Please refer to <http://www.semantic-kitti.org/dataset.html>'_ for data downloading

参数

- **data_root** (*str*) –Path of dataset root.
- **ann_file** (*str*) –Path of annotation file.
- **pipeline** (*list[dict], optional*) –Pipeline used for data processing. Defaults to None.
- **classes** (*tuple[str], optional*) –Classes used in the dataset. Defaults to None.
- **modality** (*dict, optional*) –Modality to specify the sensor data used as input. Defaults to None.
- **box_type_3d** (*str, optional*) –NO 3D box for this dataset. You can choose any type Based on the *box_type_3d*, the dataset will encapsulate the box to its original format then converted them to *box_type_3d*. Defaults to 'LiDAR' in this dataset. Available options includes
  - 'LiDAR': Box in LiDAR coordinates.
  - 'Depth': Box in depth coordinates, usually for indoor dataset.
  - 'Camera': Box in camera coordinates.
- **filter_empty_gt** (*bool, optional*) –Whether to filter empty GT. Defaults to True.
- **test_mode** (*bool, optional*) –Whether the dataset is in test mode. Defaults to False.

get_ann_info(*index*)

Get annotation info according to the given index.

参数 **index** (*int*) –Index of the annotation data to get.

返回

annotation information consists of the following keys:

- pts_semantic_mask_path (str): Path of semantic masks.

返回类型 dict

get_data_info(*index*)

Get data info according to the given index. :param index: Index of the sample data to get. :type index: int

返回

Data information that will be passed to the data preprocessing pipelines. It includes the following keys: - sample_idx (str): Sample index. - pts_filename (str): Filename of point clouds. - file_name (str): Filename of point clouds. - ann_info (dict): Annotation info.

返回类型 dict

class mmdet3d.datasets.**VoxelBasedPointSampler**(*cur_sweep_cfg*, *prev_sweep_cfg=None*, *time_dim=3*)

Voxel based point sampler.

Apply voxel sampling to multiple sweep points.

参数

- **cur_sweep_cfg** (*dict*) –Config for sampling current points.
- **prev_sweep_cfg** (*dict*) –Config for sampling previous points.
- **time_dim** (*int*) –Index that indicate the time dimension for input points.

class mmdet3d.datasets.**WaymoDataset**(*data_root*, *ann_file*, *split*, *pts_prefix='velodyne'*, *pipeline=None*, *classes=None*, *modality=None*, *box_type_3d='LiDAR'*, *filter_empty_gt=True*, *test_mode=False*, *load_interval=1*, *pcd_limit_range=[- 85, - 85, - 5, 85, 85, 5]*, ***kwargs*)

Waymo Dataset.

This class serves as the API for experiments on the Waymo Dataset.

Please refer to '<https://waymo.com/open/download/>'_for data downloading. It is recommended to symlink the dataset root to $MMDETECTION3D/data and organize them as the doc shows.

参数

- **data_root** (*str*) –Path of dataset root.
- **ann_file** (*str*) –Path of annotation file.
- **split** (*str*) –Split of input data.
- **pts_prefix** (*str, optional*) –Prefix of points files. Defaults to 'velodyne'.
- **pipeline** (*list[dict], optional*) –Pipeline used for data processing. Defaults to None.
- **classes** (*tuple[str], optional*) –Classes used in the dataset. Defaults to None.
- **modality** (*dict, optional*) –Modality to specify the sensor data used as input. Defaults to None.

- **box_type_3d** (*str, optional*) –Type of 3D box of this dataset. Based on the *box_type_3d*, the dataset will encapsulate the box to its original format then converted them to *box_type_3d*. Defaults to ‘LiDAR’ in this dataset. Available options includes
  - ’LiDAR’: box in LiDAR coordinates
  - ’Depth’: box in depth coordinates, usually for indoor dataset
  - ’Camera’: box in camera coordinates
- **filter_empty_gt** (*bool, optional*) –Whether to filter empty GT. Defaults to True.
- **test_mode** (*bool, optional*) –Whether the dataset is in test mode. Defaults to False.
- **pcd_limit_range** (*list(float), optional*) –The range of point cloud used to filter invalid predicted boxes. Default: [-85, -85, -5, 85, 85, 5].

bbox2result_kitti(*net_outputs*, *class_names*, *pklfile_prefix=None*, *submission_prefix=None*)

Convert results to kitti format for evaluation and test submission.

参数

- **net_outputs** (*List[np.ndarray]*) –list of array storing the bbox and score
- **class_nanes** (*List[String]*) –A list of class names
- **pklfile_prefix** (*str*) –The prefix of pkl file.
- **submission_prefix** (*str*) –The prefix of submission file.

返回 A list of dict have the kitti 3d format

返回类型 List[dict]

convert_valid_bboxes(*box_dict*, *info*)

Convert the boxes into valid format.

参数

- **box_dict** (*dict*) –Bounding boxes to be converted.
  - boxes_3d (:obj:LiDARInstance3DBoxes): 3D bounding boxes.
  - scores_3d (np.ndarray): Scores of predicted boxes.
  - labels_3d (np.ndarray): Class labels of predicted boxes.
- **info** (*dict*) –Dataset information dictionary.

返回

Valid boxes after conversion.

- bbox (np.ndarray): 2D bounding boxes (in camera 0).

- box3d_camera (np.ndarray): 3D boxes in camera coordinates.
- box3d_lidar (np.ndarray): 3D boxes in lidar coordinates.
- scores (np.ndarray): Scores of predicted boxes.
- label_preds (np.ndarray): Class labels of predicted boxes.
- sample_idx (np.ndarray): Sample index.

返回类型 dict

evaluate(*results*, *metric='waymo'*, *logger=None*, *pklfile_prefix=None*, *submission_prefix=None*, *show=False*, *out_dir=None*, *pipeline=None*)

Evaluation in KITTI protocol.

参数

- **results** (*list[dict]*) –Testing results of the dataset.
- **metric** (*str | list[str], optional*) –Metrics to be evaluated. Default: 'waymo'. Another supported metric is 'kitti'.
- **logger** (*logging.Logger | str, optional*) –Logger used for printing related information during evaluation. Default: None.
- **pklfile_prefix** (*str, optional*) –The prefix of pkl files including the file path and the prefix of filename, e.g., "a/b/prefix". If not specified, a temp file will be created. Default: None.
- **submission_prefix** (*str, optional*) –The prefix of submission data. If not specified, the submission data will not be generated.
- **show** (*bool, optional*) –Whether to visualize. Default: False.
- **out_dir** (*str, optional*) –Path to save the visualization results. Default: None.
- **pipeline** (*list[dict], optional*) –raw data loading for showing. Default: None.

返回 float]: results of each evaluation metric

返回类型 dict[str

format_results(*outputs*, *pklfile_prefix=None*, *submission_prefix=None*, *data_format='waymo'*)

Format the results to pkl file.

参数

- **outputs** (*list[dict]*) –Testing results of the dataset.
- **pklfile_prefix** (*str*) –The prefix of pkl files. It includes the file path and the prefix of filename, e.g., "a/b/prefix". If not specified, a temp file will be created. Default: None.

- **submission_prefix** (*str*) –The prefix of submitted files. It includes the file path and the prefix of filename, e.g., "a/b/prefix". If not specified, a temp file will be created. Default: None.
- **data_format** (*str, optional*) –Output data format. Default: 'waymo'. Another supported choice is 'kitti'.

返回

(result_files, tmp_dir), result_files is a dict containing the json filepaths, tmp_dir is the temporal directory created for saving json files when jsonfile_prefix is not specified.

返回类型 tuple

get_data_info(*index*)

Get data info according to the given index.

参数 **index** (*int*) –Index of the sample data to get.

返回

Standard input_dict consists of the data information.

- sample_idx (str): sample index
- pts_filename (str): filename of point clouds
- img_prefix (str): prefix of image files
- img_info (dict): image info
- **lidar2img (list[np.ndarray], optional): transformations from** lidar to different cameras
- ann_info (dict): annotation info

返回类型 dict

mmdet3d.datasets.**build_dataloader**(*dataset*, *samples_per_gpu*, *workers_per_gpu*, *num_gpus=1*, *dist=True*, *shuffle=True*, *seed=None*, *runner_type='EpochBasedRunner'*, *persistent_workers=False*, *class_aware_sampler=None*, ***kwargs*)

Build PyTorch DataLoader.

In distributed training, each GPU/process has a dataloader. In non-distributed training, there is only one dataloader for all GPUs.

参数

- **dataset** (*Dataset*) –A PyTorch dataset.
- **samples_per_gpu** (*int*) –Number of training samples on each GPU, i.e., batch size of each GPU.

- **workers_per_gpu** (*int*) –How many subprocesses to use for data loading for each GPU.
- **num_gpus** (*int*) –Number of GPUs. Only used in non-distributed training.
- **dist** (*bool*) –Distributed training/test or not. Default: True.
- **shuffle** (*bool*) –Whether to shuffle the data at every epoch. Default: True.
- **seed** (*int, Optional*) –Seed to be used. Default: None.
- **runner_type** (*str*) –Type of runner. Default: *EpochBasedRunner*
- **persistent_workers** (*bool*) –If True, the data loader will not shutdown the worker processes after a dataset has been consumed once. This allows to maintain the workers *Dataset* instances alive. This argument is only valid when PyTorch>=1.7.0. Default: False.
- **class_aware_sampler** (*dict*) –Whether to use *ClassAwareSampler* during training. Default: None.
- **kwargs** –any keyword argument to be used to initialize DataLoader

返回 A PyTorch dataloader.

返回类型 DataLoader

mmdet3d.datasets.**get_loading_pipeline**(*pipeline*)

Only keep loading image, points and annotations related configuration.

参数 **pipeline** (list[dict] | list[Pipeline]) –Data pipeline configs or list of pipeline functions.

返回

The new pipeline list with only keep loading image, points and annotations related configuration.

返回类型 list[dict] | list[Pipeline])

实际案例

```
>>> pipelines = [
...     dict(type='LoadPointsFromFile',
...          coord_type='LIDAR', load_dim=4, use_dim=4),
...     dict(type='LoadImageFromFile'),
...     dict(type='LoadAnnotations3D',
...          with_bbox=True, with_label_3d=True),
...     dict(type='Resize',
...          img_scale=[(640, 192), (2560, 768)], keep_ratio=True),
...     dict(type='RandomFlip3D', flip_ratio_bev_horizontal=0.5),
...     dict(type='PointsRangeFilter',
...          point_cloud_range=point_cloud_range),
```

(下页继续)

(续上页)

```
...     dict(type='ObjectRangeFilter',
...          point_cloud_range=point_cloud_range),
...     dict(type='PointShuffle'),
...     dict(type='Normalize', **img_norm_cfg),
...     dict(type='Pad', size_divisor=32),
...     dict(type='DefaultFormatBundle3D', class_names=class_names),
...     dict(type='Collect3D',
...          keys=['points', 'img', 'gt_bboxes_3d', 'gt_labels_3d'])
...     ]
>>> expected_pipelines = [
...     dict(type='LoadPointsFromFile',
...          coord_type='LIDAR', load_dim=4, use_dim=4),
...     dict(type='LoadImageFromFile'),
...     dict(type='LoadAnnotations3D',
...          with_bbox=True, with_label_3d=True),
...     dict(type='DefaultFormatBundle3D', class_names=class_names),
...     dict(type='Collect3D',
...          keys=['points', 'img', 'gt_bboxes_3d', 'gt_labels_3d'])
...     ]
>>> assert expected_pipelines ==         ...         get_loading_
↪pipeline(pipelines)
```

CHAPTER 40

mmdet3d.models

40.1 detectors

class mmdet3d.models.detectors.**Base3DDetector**(*init_cfg=None*)

Base class for detectors.

forward(*return_loss=True*, ***kwargs*)

Calls either forward_train or forward_test depending on whether return_loss=True.

Note this setting will change the expected inputs. When *return_loss=True*, img and img_metas are single-nested (i.e. torch.Tensor and list[dict]), and when *resturn_loss=False*, img and img_metas should be double nested (i.e. list[torch.Tensor], list[list[dict]]), with the outer list indicating test time augmentations.

forward_test(*points*, *img_metas*, *img=None*, ***kwargs*)

参数

- **points** (*list[torch.Tensor]*) –the outer list indicates test-time augmentations and inner torch.Tensor should have a shape NxC, which contains all points in the batch.
- **img_metas** (*list[list[dict]]*) –the outer list indicates test-time augs (multi-scale, flip, etc.) and the inner list indicates images in a batch
- **img** (*list[torch.Tensor], optional*) –the outer list indicates test-time augmentations and inner torch.Tensor should have a shape NxCxHxW, which contains all images in the batch. Defaults to None.

show_results(*data*, *result*, *out_dir*, *show=False*, *score_thr=None*)

Results visualization.

参数

- **data** (*list[dict]*) –Input points and the information of the sample.
- **result** (*list[dict]*) –Prediction results.
- **out_dir** (*str*) –Output directory of visualization result.
- **show** (*bool, optional*) –Determines whether you are going to show result by open3d. Defaults to False.
- **score_thr** (*float, optional*) –Score threshold of bounding boxes. Default to None.

class mmdet3d.models.detectors.**CenterPoint**(*pts_voxel_layer=None*, *pts_voxel_encoder=None*, *pts_middle_encoder=None*, *pts_fusion_layer=None*, *img_backbone=None*, *pts_backbone=None*, *img_neck=None*, *pts_neck=None*, *pts_bbox_head=None*, *img_roi_head=None*, *img_rpn_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

Base class of Multi-modality VoxelNet.

aug_test(*points*, *img_metas*, *imgs=None*, *rescale=False*)

Test function with augmentaiton.

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

Test function of point cloud branch with augmentaiton.

The function implementation process is as follows:

- step 1: map features back for double-flip augmentation.
- step 2: merge all features and generate boxes.
- step 3: map boxes back for scale augmentation.
- step 4: merge results.

参数

- **feats** (*list[torch.Tensor]*) –Feature of point cloud.
- **img_metas** (*list[dict]*) –Meta information of samples.
- **rescale** (*bool, optional*) –Whether to rescale bboxes. Default: False.

返回

Returned bboxes consists of the following keys:

- boxes_3d (LiDARInstance3DBoxes): Predicted bboxes.
- scores_3d (torch.Tensor): Scores of predicted boxes.
- labels_3d (torch.Tensor): Labels of predicted boxes.

返回类型 dict

extract_pts_feat(*pts*, *img_feats*, *img_metas*)

Extract features of points.

forward_pts_train(*pts_feats*, *gt_bboxes_3d*, *gt_labels_3d*, *img_metas*, *gt_bboxes_ignore=None*)

Forward function for point cloud branch.

参数

- **pts_feats** (*list[torch.Tensor]*) –Features of point cloud branch
- **gt_bboxes_3d** (list[BaseInstance3DBoxes]) –Ground truth boxes for each sample.
- **gt_labels_3d** (*list[torch.Tensor]*) –Ground truth labels for boxes of each sampole
- **img_metas** (*list[dict]*) –Meta information of samples.
- **gt_bboxes_ignore** (*list[torch.Tensor], optional*) –Ground truth boxes to be ignored. Defaults to None.

返回 Losses of each branch.

返回类型 dict

simple_test_pts(*x*, *img_metas*, *rescale=False*)

Test function of point cloud branch.

property with_velocity

Whether the head predicts velocity

Type bool

class mmdet3d.models.detectors.**DynamicMVXFasterRCNN**(***kwargs*)

Multi-modality VoxelNet using Faster R-CNN and dynamic voxelization.

extract_pts_feat(*points*, *img_feats*, *img_metas*)

Extract point features.

voxelize(*points*)

Apply dynamic voxelization to points.

参数 **points** (*list[torch.Tensor]*) –Points of each sample.

返回 Concatenated points and coordinates.

返回类型 tuple[torch.Tensor]

class mmdet3d.models.detectors.**DynamicVoxelNet**(*voxel_layer*, *voxel_encoder*, *middle_encoder*, *backbone*, *neck=None*, *bbox_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

VoxelNet using dynamic voxelization.

extract_feat(*points*, *img_metas*)

Extract features from points.

voxelize(*points*)

Apply dynamic voxelization to points.

参数 **points** (*list[torch.Tensor]*) –Points of each sample.

返回 Concatenated points and coordinates.

返回类型 tuple[torch.Tensor]

class mmdet3d.models.detectors.**FCOSMono3D**(*backbone*, *neck*, *bbox_head*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*)

FCOS3D for monocular 3D object detection.

Currently please refer to our entry on the leaderboard.

class mmdet3d.models.detectors.**GroupFree3DNet**(*backbone*, *bbox_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*)

Group-Free 3D.

aug_test(*points*, *img_metas*, *imgs=None*, *rescale=False*)

Test with augmentation.

forward_train(*points*, *img_metas*, *gt_bboxes_3d*, *gt_labels_3d*, *pts_semantic_mask=None*, *pts_instance_mask=None*, *gt_bboxes_ignore=None*)

Forward of training.

参数

- **points** (*list[torch.Tensor]*) –Points of each batch.
- **img_metas** (*list*) –Image metas.
- **gt_bboxes_3d** (BaseInstance3DBoxes) –gt bboxes of each batch.
- **gt_labels_3d** (*list[torch.Tensor]*) –gt class labels of each batch.
- **pts_semantic_mask** (*list[torch.Tensor]*) –point-wise semantic label of each batch.
- **pts_instance_mask** (*list[torch.Tensor]*) –point-wise instance label of each batch.
- **gt_bboxes_ignore** (*list[torch.Tensor]*) –Specify which bounding.

返回 torch.Tensor]: Losses.

返回类型 dict[str

simple_test(*points*, *img_metas*, *imgs=None*, *rescale=False*)

Forward of testing.

参数

- **points** (*list[torch.Tensor]*) –Points of each sample.
- **img_metas** (*list*) –Image metas.
- **rescale** (*bool*) –Whether to rescale results.

返回 Predicted 3d boxes.

返回类型 list

class mmdet3d.models.detectors.**H3DNet**(*backbone*, *neck=None*, *rpn_head=None*, *roi_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

H3DNet model.

Please refer to the paper

aug_test(*points*, *img_metas*, *imgs=None*, *rescale=False*)

Test with augmentation.

extract_feats(*points*, *img_metas*)

Extract features of multiple samples.

forward_train(*points*, *img_metas*, *gt_bboxes_3d*, *gt_labels_3d*, *pts_semantic_mask=None*, *pts_instance_mask=None*, *gt_bboxes_ignore=None*)

Forward of training.

参数

- **points** (*list[torch.Tensor]*) –Points of each batch.
- **img_metas** (*list*) –Image metas.
- **gt_bboxes_3d** (BaseInstance3DBoxes) –gt bboxes of each batch.
- **gt_labels_3d** (*list[torch.Tensor]*) –gt class labels of each batch.
- **pts_semantic_mask** (*list[torch.Tensor]*) –point-wise semantic label of each batch.
- **pts_instance_mask** (*list[torch.Tensor]*) –point-wise instance label of each batch.
- **gt_bboxes_ignore** (*list[torch.Tensor]*) –Specify which bounding.

返回 Losses.

返回类型 dict

simple_test(*points*, *img_metas*, *imgs=None*, *rescale=False*)

Forward of testing.

参数

- **points** (*list[torch.Tensor]*) –Points of each sample.
- **img_metas** (*list*) –Image metas.
- **rescale** (*bool*) –Whether to rescale results.

返回 Predicted 3d boxes.

返回类型 list

class mmdet3d.models.detectors.**ImVoteNet**(*pts_backbone=None*, *pts_bbox_heads=None*, *pts_neck=None*, *img_backbone=None*, *img_neck=None*, *img_roi_head=None*, *img_rpn_head=None*, *img_mlp=None*, *freeze_img_branch=False*, *fusion_layer=None*, *num_sampled_seed=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

ImVoteNet for 3D detection.

aug_test(*points=None*, *img_metas=None*, *imgs=None*, *bboxes_2d=None*, *rescale=False*, ***kwargs*)

Test function with augmentation, stage 2.

参数

- **points** (*list[list[torch.Tensor]], optional*) –the outer list indicates test-time augmentations and the inner list contains all points in the batch, where each Tensor should have a shape NxC. Defaults to None.
- **img_metas** (*list[list[dict]], optional*) –the outer list indicates test-time augs (multiscale, flip, etc.) and the inner list indicates images in a batch. Defaults to None.
- **imgs** (*list[list[torch.Tensor]], optional*) –the outer list indicates test-time augmentations and inner Tensor should have a shape NxCxHxW, which contains all images in the batch. Defaults to None. Defaults to None.
- **bboxes_2d** (*list[list[torch.Tensor]], optional*) –Provided 2d bboxes, not supported yet. Defaults to None.
- **rescale** (*bool, optional*) –Whether or not rescale bboxes. Defaults to False.

返回 Predicted 3d boxes.

返回类型 list[dict]

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

Test function with augmentation, image network pretrain. May refer to https://github.com/open-mmlab/mmdetection/blob/master/mmdet/models/detectors/two_stage.py.

参数

- **img** (*list[list[torch.Tensor]], optional*) –the outer list indicates test-time augmentations and inner Tensor should have a shape NxCxHxW, which contains all images in the batch. Defaults to None. Defaults to None.
- **img_metas** (*list[list[dict]], optional*) –the outer list indicates test-time augs (multiscale, flip, etc.) and the inner list indicates images in a batch. Defaults to None.
- **rescale** (*bool, optional*) –Whether or not rescale bboxes to the original shape of input image. If rescale is False, then returned bboxes and masks will fit the scale of imgs[0]. Defaults to None.

返回 Predicted 2d boxes.

返回类型 list[list[torch.Tensor]]

extract_bboxes_2d(*img*, *img_metas*, *train=True*, *bboxes_2d=None*, ***kwargs*)

Extract bounding boxes from 2d detector.

参数

- **img** (*torch.Tensor*) –of shape (N, C, H, W) encoding input images. Typically these should be mean centered and std scaled.
- **img_metas** (*list[dict]*) –Image meta info.
- **train** (*bool*) –train-time or not.
- **bboxes_2d** (*list[torch.Tensor]*) –provided 2d bboxes, not supported yet.

返回 a list of processed 2d bounding boxes.

返回类型 list[torch.Tensor]

extract_feat(*imgs*)

Just to inherit from abstract method.

extract_img_feat(*img*)

Directly extract features from the img backbone+neck.

extract_img_feats(*imgs*)

Extract features from multiple images.

参数 **imgs** (*list[torch.Tensor]*) –A list of images. The images are augmented from the same image but in different ways.

返回 Features of different images

返回类型 list[torch.Tensor]

extract_pts_feat(*pts*)

Extract features of points.

extract_pts_feats(*pts*)

Extract features of points from multiple samples.

forward_test(*points=None*, *img_metas=None*, *img=None*, *bboxes_2d=None*, ***kwargs*)

Forwarding of test for image branch pretrain or stage 2 train.

参数

- **points** (*list[list[torch.Tensor]], optional*) –the outer list indicates test-time augmentations and the inner list contains all points in the batch, where each Tensor should have a shape NxC. Defaults to None.
- **img_metas** (*list[list[dict]], optional*) –the outer list indicates test-time augs (multiscale, flip, etc.) and the inner list indicates images in a batch. Defaults to None.
- **img** (*list[list[torch.Tensor]], optional*) –the outer list indicates test-time augmentations and inner Tensor should have a shape NxCxHxW, which contains all images in the batch. Defaults to None. Defaults to None.
- **bboxes_2d** (*list[list[torch.Tensor]], optional*) –Provided 2d bboxes, not supported yet. Defaults to None.

返回 Predicted 2d or 3d boxes.

返回类型 list[list[torch.Tensor]]|list[dict]

forward_train(*points=None*, *img=None*, *img_metas=None*, *gt_bboxes=None*, *gt_labels=None*, *gt_bboxes_ignore=None*, *gt_masks=None*, *proposals=None*, *bboxes_2d=None*, *gt_bboxes_3d=None*, *gt_labels_3d=None*, *pts_semantic_mask=None*, *pts_instance_mask=None*, ***kwargs*)

Forwarding of train for image branch pretrain or stage 2 train.

参数

- **points** (*list[torch.Tensor]*) –Points of each batch.
- **img** (*torch.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 and point cloud meta info dict. For example, keys include ‘ori_shape’, ‘img_norm_cfg’, and ‘transformation_3d_flow’. For details on the values of the keys see *mmdet/datasets/pipelines/formatting.py:Collect*.
- **gt_bboxes** (*list[torch.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[torch.Tensor]*) –class indices for each 2d bounding box.
- **gt_bboxes_ignore** (*list[torch.Tensor]*) –specify which 2d bounding boxes can be ignored when computing the loss.
- **gt_masks** (*torch.Tensor*) –true segmentation masks for each 2d bbox, used if the architecture supports a segmentation task.
- **proposals** –override rpn proposals (2d) with custom proposals. Use when *with_rpn* is False.
- **bboxes_2d** (*list[torch.Tensor]*) –provided 2d bboxes, not supported yet.
- **gt_bboxes_3d** (BaseInstance3DBoxes) –3d gt bboxes.
- **gt_labels_3d** (*list[torch.Tensor]*) –gt class labels for 3d bboxes.
- **pts_semantic_mask** (*list[torch.Tensor]*) –point-wise semantic label of each batch.
- **pts_instance_mask** (*list[torch.Tensor]*) –point-wise instance label of each batch.

返回 a dictionary of loss components.

返回类型 dict[str, torch.Tensor]

freeze_img_branch_params()

Freeze all image branch parameters.

simple_test(*points=None*, *img_metas=None*, *img=None*, *bboxes_2d=None*, *rescale=False*, ***kwargs*)

Test without augmentation, stage 2.

参数

- **points** (*list[torch.Tensor], optional*) –Elements in the list should have a shape NxC, the list indicates all point-clouds in the batch. Defaults to None.
- **img_metas** (*list[dict], optional*) –List indicates images in a batch. Defaults to None.
- **img** (*torch.Tensor, optional*) –Should have a shape NxCxHxW, which contains all images in the batch. Defaults to None.
- **bboxes_2d** (*list[torch.Tensor], optional*) –Provided 2d bboxes, not supported yet. Defaults to None.
- **rescale** (*bool, optional*) –Whether or not rescale bboxes. Defaults to False.

返回 Predicted 3d boxes.

返回类型 list[dict]

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

Test without augmentation, image network pretrain. May refer to https://github.com/open-mmlab/mmdetection/blob/master/mmdet/models/detectors/two_stage.py.

参数

- **img** (*torch.Tensor*) –Should have a shape NxCxHxW, which contains all images in the batch.
- **img_metas** (*list[dict]*) –
- **proposals** (*list[Tensor], optional*) –override rpn proposals with custom proposals. Defaults to None.
- **rescale** (*bool, optional*) –Whether or not rescale bboxes to the original shape of input image. Defaults to False.

返回 Predicted 2d boxes.

返回类型 list[list[torch.Tensor]]

train(*mode=True*)

Overload in order to keep image branch modules in eval mode.

property with_img_backbone

Whether the detector has a 2D image backbone.

Type bool

property with_img_bbox

Whether the detector has a 2D image box head.

Type bool

property with_img_bbox_head

Whether the detector has a 2D image box head (not roi).

Type bool

property with_img_neck

Whether the detector has a neck in image branch.

Type bool

property with_img_roi_head

Whether the detector has a RoI Head in image branch.

Type bool

property with_img_rpn

Whether the detector has a 2D RPN in image detector branch.

Type bool

property with_pts_backbone

Whether the detector has a 3D backbone.

Type bool

property with_pts_bbox

Whether the detector has a 3D box head.

Type bool

property with_pts_neck

Whether the detector has a neck in 3D detector branch.

Type bool

class mmdet3d.models.detectors.**ImVoxelNet**(*backbone*, *neck*, *neck_3d*, *bbox_head*, *n_voxels*, *anchor_generator*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

ImVoxelNet.

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

Test with augmentations.

参数

- **imgs** (*list[torch.Tensor]*) –Input images of shape (N, C_in, H, W).
- **img_metas** (*list*) –Image metas.

返回 Predicted 3d boxes.

返回类型 list[dict]

extract_feat(*img*, *img_metas*)

Extract 3d features from the backbone -> fpn -> 3d projection.

参数

- **img** (*torch.Tensor*) –Input images of shape (N, C_in, H, W).
- **img_metas** (*list*) –Image metas.

返回 of shape (N, C_out, N_x, N_y, N_z)

返回类型 torch.Tensor

forward_test(*img*, *img_metas*, ***kwargs*)

Forward of testing.

参数

- **img** (*torch.Tensor*) –Input images of shape (N, C_in, H, W).
- **img_metas** (*list*) –Image metas.

返回 Predicted 3d boxes.

返回类型 list[dict]

forward_train(*img*, *img_metas*, *gt_bboxes_3d*, *gt_labels_3d*, ***kwargs*)

Forward of training.

参数

- **img** (*torch.Tensor*) –Input images of shape (N, C_in, H, W).
- **img_metas** (*list*) –Image metas.
- **gt_bboxes_3d** (BaseInstance3DBoxes) –gt bboxes of each batch.
- **gt_labels_3d** (*list[torch.Tensor]*) –gt class labels of each batch.

返回 A dictionary of loss components.

返回类型 dict[str, torch.Tensor]

simple_test(*img*, *img_metas*)

Test without augmentations.

参数

- **img** (*torch.Tensor*) –Input images of shape (N, C_in, H, W).
- **img_metas** (*list*) –Image metas.

返回 Predicted 3d boxes.

返回类型 list[dict]

class mmdet3d.models.detectors.**MVXFasterRCNN**(***kwargs*)

Multi-modality VoxelNet using Faster R-CNN.

class mmdet3d.models.detectors.**MVXTwoStageDetector**(*pts_voxel_layer=None*, *pts_voxel_encoder=None*, *pts_middle_encoder=None*, *pts_fusion_layer=None*, *img_backbone=None*, *pts_backbone=None*, *img_neck=None*, *pts_neck=None*, *pts_bbox_head=None*, *img_roi_head=None*, *img_rpn_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

Base class of Multi-modality VoxelNet.

aug_test(*points*, *img_metas*, *imgs=None*, *rescale=False*)

Test function with augmentaiton.

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

Test function of point cloud branch with augmentaiton.

extract_feat(*points*, *img*, *img_metas*)

Extract features from images and points.

extract_feats(*points*, *img_metas*, *imgs=None*)

Extract point and image features of multiple samples.

extract_img_feat(*img*, *img_metas*)

Extract features of images.

extract_pts_feat(*pts*, *img_feats*, *img_metas*)

Extract features of points.

forward_img_train(*x*, *img_metas*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*, *proposals=None*, ***kwargs*)

Forward function for image branch.

This function works similar to the forward function of Faster R-CNN.

参数

- **x** (*list[torch.Tensor]*) –Image features of shape (B, C, H, W) of multiple levels.
- **img_metas** (*list[dict]*) –Meta information of images.
- **gt_bboxes** (*list[torch.Tensor]*) –Ground truth boxes of each image sample.
- **gt_labels** (*list[torch.Tensor]*) –Ground truth labels of boxes.
- **gt_bboxes_ignore** (*list[torch.Tensor], optional*) –Ground truth boxes to be ignored. Defaults to None.
- **proposals** (*list[torch.Tensor], optional*) –Proposals of each sample. Defaults to None.

返回 Losses of each branch.

返回类型 dict

forward_pts_train(*pts_feats*, *gt_bboxes_3d*, *gt_labels_3d*, *img_metas*, *gt_bboxes_ignore=None*)

Forward function for point cloud branch.

参数

- **pts_feats** (*list[torch.Tensor]*) –Features of point cloud branch
- **gt_bboxes_3d** (list[BaseInstance3DBoxes]) –Ground truth boxes for each sample.
- **gt_labels_3d** (*list[torch.Tensor]*) –Ground truth labels for boxes of each sampole
- **img_metas** (*list[dict]*) –Meta information of samples.

- **gt_bboxes_ignore** (*list[torch.Tensor], optional*) –Ground truth boxes to be ignored. Defaults to None.

返回 Losses of each branch.

返回类型 dict

forward_train(*points=None*, *img_metas=None*, *gt_bboxes_3d=None*, *gt_labels_3d=None*, *gt_labels=None*, *gt_bboxes=None*, *img=None*, *proposals=None*, *gt_bboxes_ignore=None*)

Forward training function.

参数

- **points** (*list[torch.Tensor], optional*) –Points of each sample. Defaults to None.
- **img_metas** (*list[dict], optional*) –Meta information of each sample. Defaults to None.
- **gt_bboxes_3d** (list[BaseInstance3DBoxes], optional) –Ground truth 3D boxes. Defaults to None.
- **gt_labels_3d** (*list[torch.Tensor], optional*) –Ground truth labels of 3D boxes. Defaults to None.
- **gt_labels** (*list[torch.Tensor], optional*) –Ground truth labels of 2D boxes in images. Defaults to None.
- **gt_bboxes** (*list[torch.Tensor], optional*) –Ground truth 2D boxes in images. Defaults to None.
- **img** (*torch.Tensor, optional*) –Images of each sample with shape (N, C, H, W). Defaults to None.
- **proposals** (*[list[torch.Tensor], optional*) –Predicted proposals used for training Fast RCNN. Defaults to None.
- **gt_bboxes_ignore** (*list[torch.Tensor], optional*) –Ground truth 2D boxes in images to be ignored. Defaults to None.

返回 Losses of different branches.

返回类型 dict

show_results(*data*, *result*, *out_dir*)

Results visualization.

参数

- **data** (*dict*) –Input points and the information of the sample.
- **result** (*dict*) –Prediction results.
- **out_dir** (*str*) –Output directory of visualization result.

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

Test function without augmentaiton.

simple_test_img(*x*, *img_metas*, *proposals=None*, *rescale=False*)

Test without augmentation.

simple_test_pts(*x*, *img_metas*, *rescale=False*)

Test function of point cloud branch.

simple_test_rpn(*x*, *img_metas*, *rpn_test_cfg*)

RPN test function.

voxelize(*points*)

Apply dynamic voxelization to points.

参数 **points** (*list[torch.Tensor]*) –Points of each sample.

返回

Concatenated points, number of points per voxel, and coordinates.

返回类型 tuple[torch.Tensor]

property with_fusion

Whether the detector has a fusion layer.

Type bool

property with_img_backbone

Whether the detector has a 2D image backbone.

Type bool

property with_img_bbox

Whether the detector has a 2D image box head.

Type bool

property with_img_neck

Whether the detector has a neck in image branch.

Type bool

property with_img_roi_head

Whether the detector has a RoI Head in image branch.

Type bool

property with_img_rpn

Whether the detector has a 2D RPN in image detector branch.

Type bool

property with_img_shared_head

Whether the detector has a shared head in image branch.

Type bool

property with_middle_encoder

Whether the detector has a middle encoder.

Type bool

property with_pts_backbone

Whether the detector has a 3D backbone.

Type bool

property with_pts_bbox

Whether the detector has a 3D box head.

Type bool

property with_pts_neck

Whether the detector has a neck in 3D detector branch.

Type bool

property with_voxel_encoder

Whether the detector has a voxel encoder.

Type bool

class mmdet3d.models.detectors.**MinkSingleStage3DDetector**(*backbone*, *head*, *voxel_size*, *train_cfg=None*, *test_cfg=None*, *init_cfg=None*, *pretrained=None*)

Single stage detector based on MinkowskiEngine GSDN.

参数

- **backbone** (*dict*) –Config of the backbone.
- **head** (*dict*) –Config of the head.
- **voxel_size** (*float*) –Voxel size in meters.
- **train_cfg** (*dict, optional*) –Config for train stage. Defaults to None.
- **test_cfg** (*dict, optional*) –Config for test stage. Defaults to None.
- **init_cfg** (*dict, optional*) –Config for weight initialization. Defaults to None.
- **pretrained** (*str, optional*) –Deprecated initialization parameter. Defaults to None.

aug_test(*points*, *img_metas*, ***kwargs*)

Test with augmentations.

参数

- **points** (*list[list[torch.Tensor]]*) –Points of each sample.
- **img_metas** (*list[dict]*) –Contains scene meta infos.

返回 Predicted 3d boxes.

返回类型 list[dict]

extract_feat(*points*)

Extract features from points.

参数 **points** (*list[Tensor]*) –Raw point clouds.

返回 Voxelized point clouds.

返回类型 SparseTensor

forward_train(*points*, *gt_bboxes_3d*, *gt_labels_3d*, *img_metas*)

Forward of training.

参数

- **points** (*list[Tensor]*) –Raw point clouds.
- **gt_bboxes** (*list[BaseInstance3DBoxes]*) –Ground truth bboxes of each sample.
- **gt_labels** (*list[torch.Tensor]*) –Labels of each sample.
- **img_metas** (*list[dict]*) –Contains scene meta infos.

返回 Centerness, bbox and classification loss values.

返回类型 dict

simple_test(*points*, *img_metas*, **args*, ***kwargs*)

Test without augmentations.

参数

- **points** (*list[torch.Tensor]*) –Points of each sample.
- **img_metas** (*list[dict]*) –Contains scene meta infos.

返回 Predicted 3d boxes.

返回类型 list[dict]

class mmdet3d.models.detectors.**PartA2**(*voxel_layer*, *voxel_encoder*, *middle_encoder*, *backbone*, *neck=None*, *rpn_head=None*, *roi_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

Part-A2 detector.

Please refer to the paper

extract_feat(*points*, *img_metas*)

Extract features from points.

forward_train(*points*, *img_metas*, *gt_bboxes_3d*, *gt_labels_3d*, *gt_bboxes_ignore=None*, *proposals=None*)

Training forward function.

参数

- **points** (*list[torch.Tensor]*) –Point cloud of each sample.
- **img_metas** (*list[dict]*) –Meta information of each sample
- **gt_bboxes_3d** (list[BaseInstance3DBoxes]) –Ground truth boxes for each sample.
- **gt_labels_3d** (*list[torch.Tensor]*) –Ground truth labels for boxes of each sampole
- **gt_bboxes_ignore** (*list[torch.Tensor], optional*) –Ground truth boxes to be ignored. Defaults to None.

返回 Losses of each branch.

返回类型 dict

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

Test function without augmentaiton.

voxelize(*points*)

Apply hard voxelization to points.

class mmdet3d.models.detectors.**PointRCNN**(*backbone*, *neck=None*, *rpn_head=None*, *roi_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

PointRCNN detector.

Please refer to the PointRCNN

参数

- **backbone** (*dict*) –Config dict of detector's backbone.
- **neck** (*dict, optional*) –Config dict of neck. Defaults to None.
- **rpn_head** (*dict, optional*) –Config of RPN head. Defaults to None.

- **roi_head** (*dict, optional*) –Config of ROI head. Defaults to None.
- **train_cfg** (*dict, optional*) –Train configs. Defaults to None.
- **test_cfg** (*dict, optional*) –Test configs. Defaults to None.
- **pretrained** (*str, optional*) –Model pretrained path. Defaults to None.
- **init_cfg** (*dict, optional*) –Config of initialization. Defaults to None.

extract_feat(*points*)

Directly extract features from the backbone+neck.

参数 **points** (*torch.Tensor*) –Input points.

返回 Features from the backbone+neck

返回类型 dict

forward_train(*points*, *img_metas*, *gt_bboxes_3d*, *gt_labels_3d*)

Forward of training.

参数

- **points** (*list[torch.Tensor]*) –Points of each batch.
- **img_metas** (*list[dict]*) –Meta information of each sample.
- **gt_bboxes_3d** (BaseInstance3DBoxes) –gt bboxes of each batch.
- **gt_labels_3d** (*list[torch.Tensor]*) –gt class labels of each batch.

返回 Losses.

返回类型 dict

simple_test(*points*, *img_metas*, *imgs=None*, *rescale=False*)

Forward of testing.

参数

- **points** (*list[torch.Tensor]*) –Points of each sample.
- **img_metas** (*list[dict]*) –Image metas.
- **imgs** (*list[torch.Tensor], optional*) –Images of each sample. Defaults to None.
- **rescale** (*bool, optional*) –Whether to rescale results. Defaults to False.

返回 Predicted 3d boxes.

返回类型 list

class mmdet3d.models.detectors.**SASSD**(*voxel_layer*, *voxel_encoder*, *middle_encoder*, *backbone*, *neck=None*, *bbox_head=None*, *train_cfg=None*, *test_cfg=None*, *init_cfg=None*, *pretrained=None*)

SASSD <https://github.com/skyhehe123/SA-SSD> _ for 3D detection.

aug_test(*points*, *img_metas*, *imgs=None*, *rescale=False*)

Test function with augmentaiton.

extract_feat(*points*, *img_metas=None*, *test_mode=False*)

Extract features from points.

forward_train(*points*, *img_metas*, *gt_bboxes_3d*, *gt_labels_3d*, *gt_bboxes_ignore=None*)

Training forward function.

参数

- **points** (*list[torch.Tensor]*) –Point cloud of each sample.
- **img_metas** (*list[dict]*) –Meta information of each sample
- **gt_bboxes_3d** (list[BaseInstance3DBoxes]) –Ground truth boxes for each sample.
- **gt_labels_3d** (*list[torch.Tensor]*) –Ground truth labels for boxes of each sampole
- **gt_bboxes_ignore** (*list[torch.Tensor], optional*) –Ground truth boxes to be ignored. Defaults to None.

返回 Losses of each branch.

返回类型 dict

simple_test(*points*, *img_metas*, *imgs=None*, *rescale=False*)

Test function without augmentaiton.

voxelize(*points*)

Apply hard voxelization to points.

class mmdet3d.models.detectors.**SMOKEMono3D**(*backbone*, *neck*, *bbox_head*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*)

SMOKE <https://arxiv.org/abs/2002.10111>‘_ for monocular 3D object detection.

class mmdet3d.models.detectors.**SSD3DNet**(*backbone*, *bbox_head=None*, *train_cfg=None*, *test_cfg=None*, *init_cfg=None*, *pretrained=None*)

3DSSDNet model.

https://arxiv.org/abs/2002.10187.pdf

class mmdet3d.models.detectors.**SingleStageMono3DDetector**(*backbone*, *neck=None*, *bbox_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

Base class for monocular 3D 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.

extract_feats(*imgs*)

Directly extract features from the backbone+neck.

forward_train(*img*, *img_metas*, *gt_bboxes*, *gt_labels*, *gt_bboxes_3d*, *gt_labels_3d*, *centers2d*, *depths*, *attr_labels=None*, *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_3d** (*list[Tensor]*) –Each item are the 3D truth boxes for each image in [x, y, z, x_size, y_size, z_size, yaw, vx, vy] format.
- **gt_labels_3d** (*list[Tensor]*) –3D class indices corresponding to each box.
- **centers2d** (*list[Tensor]*) –Projected 3D centers onto 2D images.
- **depths** (*list[Tensor]*) –Depth of projected centers on 2D images.
- **attr_labels** (*list[Tensor], optional*) –Attribute indices corresponding to each box
- **gt_bboxes_ignore** (*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]

show_results(*data*, *result*, *out_dir*, *show=False*, *score_thr=None*)

Results visualization.

参数

- **data** (*list[dict]*) –Input images and the information of the sample.
- **result** (*list[dict]*) –Prediction results.
- **out_dir** (*str*) –Output directory of visualization result.
- **show** (*bool, optional*) –Determines whether you are going to show result by open3d. Defaults to False.
- **TODO** –implement score_thr of single_stage_mono3d.
- **score_thr** (*float, optional*) –Score threshold of bounding boxes. Default to None. Not implemented yet, but it is here for unification.

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 mmdet3d.models.detectors.**VoteNet**(*backbone*, *bbox_head=None*, *train_cfg=None*, *test_cfg=None*, *init_cfg=None*, *pretrained=None*)

VoteNet for 3D detection.

aug_test(*points*, *img_metas*, *imgs=None*, *rescale=False*)

Test with augmentation.

forward_train(*points*, *img_metas*, *gt_bboxes_3d*, *gt_labels_3d*, *pts_semantic_mask=None*, *pts_instance_mask=None*, *gt_bboxes_ignore=None*)

Forward of training.

参数

- **points** (*list[torch.Tensor]*) –Points of each batch.
- **img_metas** (*list*) –Image metas.
- **gt_bboxes_3d** (BaseInstance3DBoxes) –gt bboxes of each batch.

- **gt_labels_3d** (*list[torch.Tensor]*) –gt class labels of each batch.
- **pts_semantic_mask** (*list[torch.Tensor]*) –point-wise semantic label of each batch.
- **pts_instance_mask** (*list[torch.Tensor]*) –point-wise instance label of each batch.
- **gt_bboxes_ignore** (*list[torch.Tensor]*) –Specify which bounding.

返回 Losses.

返回类型 dict

simple_test(*points*, *img_metas*, *imgs=None*, *rescale=False*)

Forward of testing.

参数

- **points** (*list[torch.Tensor]*) –Points of each sample.
- **img_metas** (*list*) –Image metas.
- **rescale** (*bool*) –Whether to rescale results.

返回 Predicted 3d boxes.

返回类型 list

class mmdet3d.models.detectors.**VoxelNet**(*voxel_layer*, *voxel_encoder*, *middle_encoder*, *backbone*, *neck=None*, *bbox_head=None*, *train_cfg=None*, *test_cfg=None*, *init_cfg=None*, *pretrained=None*)

VoxelNet for 3D detection.

aug_test(*points*, *img_metas*, *imgs=None*, *rescale=False*)

Test function with augmentaiton.

extract_feat(*points*, *img_metas=None*)

Extract features from points.

forward_train(*points*, *img_metas*, *gt_bboxes_3d*, *gt_labels_3d*, *gt_bboxes_ignore=None*)

Training forward function.

参数

- **points** (*list[torch.Tensor]*) –Point cloud of each sample.
- **img_metas** (*list[dict]*) –Meta information of each sample
- **gt_bboxes_3d** (list[BaseInstance3DBoxes]) –Ground truth boxes for each sample.
- **gt_labels_3d** (*list[torch.Tensor]*) –Ground truth labels for boxes of each sampole

- **gt_bboxes_ignore** (*list[torch.Tensor], optional*) –Ground truth boxes to be ignored. Defaults to None.

返回 Losses of each branch.

返回类型 dict

simple_test(*points*, *img_metas*, *imgs=None*, *rescale=False*)

Test function without augmentaiton.

voxelize(*points*)

Apply hard voxelization to points.

40.2 backbones

class mmdet3d.models.backbones.**DGCNNBackbone**(*in_channels*, *num_samples=(20, 20, 20)*, *knn_modes=('D-KNN', 'F-KNN', 'F-KNN')*, *radius=(None, None, None)*, *gf_channels=((64, 64), (64, 64), (64))*, *fa_channels=(1024)*, *act_cfg={'type': 'ReLU'}*, *init_cfg=None*)

Backbone network for DGCNN.

参数

- **in_channels** (*int*) –Input channels of point cloud.
- **num_samples** (*tuple[int], optional*) –The number of samples for knn or ball query in each graph feature (GF) module. Defaults to (20, 20, 20).
- **knn_modes** (*tuple[str], optional*) –Mode of KNN of each knn module. Defaults to ('D-KNN', 'F-KNN', 'F-KNN').
- **radius** (*tuple[float], optional*) –Sampling radii of each GF module. Defaults to (None, None, None).
- **gf_channels** (*tuple[tuple[int]], optional*) –Out channels of each mlp in GF module. Defaults to ((64, 64), (64, 64), (64,)).
- **fa_channels** (*tuple[int], optional*) –Out channels of each mlp in FA module. Defaults to (1024,).
- **act_cfg** (*dict, optional*) –Config of activation layer. Defaults to dict(type='ReLU').
- **init_cfg** (*dict, optional*) –Initialization config. Defaults to None.

forward(*points*)

Forward pass.

参数 **points** (*torch.Tensor*) –point coordinates with features, with shape (B, N, in_channels).

返回

Outputs after graph feature (GF) and feature aggregation (FA) modules.

- gf_points (list[torch.Tensor]): Outputs after each GF module.
- fa_points (torch.Tensor): Outputs after FA module.

返回类型 dict[str, list[torch.Tensor]]

class mmdet3d.models.backbones.**DLANet**(*depth*, *in_channels=3*, *out_indices=(0, 1, 2, 3, 4, 5)*, *frozen_stages=- 1*, *norm_cfg=None*, *conv_cfg=None*, *layer_with_level_root=(False, True, True, True)*, *with_identity_root=False*, *pretrained=None*, *init_cfg=None*)

DLA backbone.

参数

- **depth** (*int*) –Depth of DLA. Default: 34.
- **in_channels** (*int, optional*) –Number of input image channels. Default: 3.
- **norm_cfg** (*dict, optional*) –Dictionary to construct and config norm layer. Default: None.
- **conv_cfg** (*dict, optional*) –Dictionary to construct and config conv layer. Default: None.
- **layer_with_level_root** (*list[bool], optional*) –Whether to apply level_root in each DLA layer, this is only used for tree levels. Default: (False, True, True, True).
- **with_identity_root** (*bool, optional*) –Whether to add identity in root layer. Default: False.
- **pretrained** (*str, optional*) –model pretrained path. Default: None.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

class mmdet3d.models.backbones.**HRNet**(*extra*, *in_channels=3*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *norm_eval=True*, *with_cp=False*, *zero_init_residual=False*, *multiscale_output=True*, *pretrained=None*, *init_cfg=None*)

HRNet backbone.

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

参数

- **extra** (*dict*) –Detailed configuration for each stage of HRNet. There must be 4 stages, the configuration for each stage must have 5 keys:
  - num_modules(int): The number of HRModule in this stage.
  - num_branches(int): The number of branches in the HRModule.
  - block(str): The type of convolution block.
  - **num_blocks(tuple): The number of blocks in each branch.** The length must be equal to num_branches.
  - **num_channels(tuple): The number of channels in each branch.** The length must be equal to num_branches.
- **in_channels** (*int*) –Number of input image channels. Default: 3.
- **conv_cfg** (*dict*) –Dictionary to construct and config conv layer.
- **norm_cfg** (*dict*) –Dictionary to construct and config norm layer.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: True.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **zero_init_residual** (*bool*) –Whether to use zero init for last norm layer in resblocks to let them behave as identity. Default: False.
- **multiscale_output** (*bool*) –Whether to output multi-level features produced by multiple branches. If False, only the first level feature will be output. Default: True.
- **pretrained** (*str, optional*) –Model pretrained path. Default: None.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None.

示例

```
>>> from mmdet.models import HRNet
>>> import torch
>>> extra = dict(
>>>     stage1=dict(
>>>         num_modules=1,
>>>         num_branches=1,
>>>         block='BOTTLENECK',
>>>         num_blocks=(4, ),
>>>         num_channels=(64, )),
>>>     stage2=dict(
>>>         num_modules=1,
>>>         num_branches=2,
>>>         block='BASIC',
>>>         num_blocks=(4, 4),
>>>         num_channels=(32, 64)),
>>>     stage3=dict(
>>>         num_modules=4,
>>>         num_branches=3,
>>>         block='BASIC',
>>>         num_blocks=(4, 4, 4),
>>>         num_channels=(32, 64, 128)),
>>>     stage4=dict(
>>>         num_modules=3,
>>>         num_branches=4,
>>>         block='BASIC',
>>>         num_blocks=(4, 4, 4, 4),
>>>         num_channels=(32, 64, 128, 256)))
>>> self = HRNet(extra, in_channels=1)
>>> self.eval()
>>> inputs = torch.rand(1, 1, 32, 32)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 32, 8, 8)
(1, 64, 4, 4)
(1, 128, 2, 2)
(1, 256, 1, 1)
```

forward(*x*)

Forward function.

property norm1

the normalization layer named "norm1"

Type nn.Module

property norm2

the normalization layer named "norm2"

Type nn.Module

train(*mode=True*)

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

class mmdet3d.models.backbones.**MinkResNet**(*depth*, *in_channels*, *num_stages=4*, *pool=True*)

Minkowski ResNet backbone. See 4D Spatio-Temporal ConvNets for more details.

参数

- **depth** (*int*) –Depth of resnet, from {18, 34, 50, 101, 152}.
- **in_channels** (*ont*) –Number of input channels, 3 for RGB.
- **num_stages** (*int, optional*) –Resnet stages. Default: 4.
- **pool** (*bool, optional*) –Add max pooling after first conv if True. Default: True.

forward(*x*)

Forward pass of ResNet.

参数 **x** (*ME.SparseTensor*) –Input sparse tensor.

返回 Output sparse tensors.

返回类型 list[ME.SparseTensor]

class mmdet3d.models.backbones.**MultiBackbone**(*num_streams*, *backbones*, *aggregation_mlp_channels=None*, *conv_cfg={'type': 'Conv1d'}*, *norm_cfg={'eps': 1e-05, 'momentum': 0.01, 'type': 'BN1d'}*, *act_cfg={'type': 'ReLU'}*, *suffixes=('net0', 'net1')*, *init_cfg=None*, *pretrained=None*, ***kwargs*)

MultiBackbone with different configs.

参数

- **num_streams** (*int*) –The number of backbones.
- **backbones** (*list or dict*) –A list of backbone configs.
- **aggregation_mlp_channels** (*list[int]*) –Specify the mlp layers for feature aggregation.
- **conv_cfg** (*dict*) –Config dict of convolutional layers.
- **norm_cfg** (*dict*) –Config dict of normalization layers.
- **act_cfg** (*dict*) –Config dict of activation layers.

- **suffixes** (*list*) –A list of suffixes to rename the return dict for each backbone.

forward(*points*)

Forward pass.

参数 **points** (*torch.Tensor*) –point coordinates with features, with shape (B, N, 3 + input_feature_dim).

返回

Outputs from multiple backbones.

- fp_xyz[suffix] (list[torch.Tensor]): The coordinates of each fp features.
- fp_features[suffix] (list[torch.Tensor]): The features from each Feature Propagate Layers.
- fp_indices[suffix] (list[torch.Tensor]): Indices of the input points.
- hd_feature (torch.Tensor): The aggregation feature from multiple backbones.

返回类型 dict[str, list[torch.Tensor]]

class mmdet3d.models.backbones.**NoStemRegNet**(*arch*, *init_cfg=None*, ***kwargs*)

RegNet backbone without Stem for 3D detection.

More details can be found in paper .

参数

- **arch** (*dict*) –The parameter of RegNets. - w0 (int): Initial width. - wa (float): Slope of width. - wm (float): Quantization parameter to quantize the width. - depth (int): Depth of the backbone. - group_w (int): Width of group. - bot_mul (float): Bottleneck ratio, i.e. expansion of bottleneck.
- **strides** (*Sequence[int]*) –Strides of the first block of each stage.
- **base_channels** (*int*) –Base channels after stem layer.
- **in_channels** (*int*) –Number of input image channels. Normally 3.
- **dilations** (*Sequence[int]*) –Dilation of each stage.
- **out_indices** (*Sequence[int]*) –Output from which stages.
- **style** (*str*) –*pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer.
- **frozen_stages** (*int*) –Stages to be frozen (all param fixed). -1 means not freezing any parameters.
- **norm_cfg** (*dict*) –Dictionary to construct and config norm layer.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only.

- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed.
- **zero_init_residual** (*bool*) –Whether to use zero init for last norm layer in res-blocks to let them behave as identity.

示例

```
>>> from mmdet3d.models import NoStemRegNet
>>> import torch
>>> self = NoStemRegNet(
        arch=dict(
            w0=88,
            wa=26.31,
            wm=2.25,
            group_w=48,
            depth=25,
            bot_mul=1.0))
>>> self.eval()
>>> inputs = torch.rand(1, 64, 16, 16)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 96, 8, 8)
(1, 192, 4, 4)
(1, 432, 2, 2)
(1, 1008, 1, 1)
```

forward(*x*)

Forward function of backbone.

参数 **x** (*torch.Tensor*) –Features in shape (N, C, H, W).

返回 Multi-scale features.

返回类型 tuple[torch.Tensor]

class mmdet3d.models.backbones.**PointNet2SAMSG**(*in_channels*, *num_points=(2048, 1024, 512, 256)*, *radii=((0.2, 0.4, 0.8), (0.4, 0.8, 1.6), (1.6, 3.2, 4.8))*, *num_samples=((32, 32, 64), (32, 32, 64), (32, 32, 32))*, *sa_channels=(((16, 16, 32), (16, 16, 32), (32, 32, 64)), ((64, 64, 128), (64, 64, 128), (64, 96, 128)), ((128, 128, 256), (128, 192, 256), (128, 256, 256)))*, *aggregation_channels=(64, 128, 256)*, *fps_mods=('D-FPS', 'FS', ('F-FPS', 'D-FPS'))*, *fps_sample_range_lists=(- 1, - 1, (512, - 1))*, *dilated_group=(True, True, True)*, *out_indices=(2)*, *norm_cfg={'type': 'BN2d'}*, *sa_cfg={'normalize_xyz': False, 'pool_mod': 'max', 'type': 'PointSAModuleMSG', 'use_xyz': True}*, *init_cfg=None*)

PointNet2 with Multi-scale grouping.

参数

- **in_channels** (*int*) –Input channels of point cloud.
- **num_points** (*tuple[int]*) –The number of points which each SA module samples.
- **radii** (*tuple[float]*) –Sampling radii of each SA module.
- **num_samples** (*tuple[int]*) –The number of samples for ball query in each SA module.
- **sa_channels** (*tuple[tuple[int]]*) –Out channels of each mlp in SA module.
- **aggregation_channels** (*tuple[int]*) –Out channels of aggregation multi-scale grouping features.
- **fps_mods** (*tuple[int]*) –Mod of FPS for each SA module.
- **fps_sample_range_lists** (*tuple[tuple[int]]*) –The number of sampling points which each SA module samples.
- **dilated_group** (*tuple[bool]*) –Whether to use dilated ball query for
- **out_indices** (*Sequence[int]*) –Output from which stages.
- **norm_cfg** (*dict*) –Config of normalization layer.
- **sa_cfg** (*dict*) –Config of set abstraction module, which may contain the following keys and values:
  - pool_mod (str): Pool method (‘max’ or ‘avg’) for SA modules.
  - use_xyz (bool): Whether to use xyz as a part of features.

- normalize_xyz (bool): Whether to normalize xyz with radii in each SA module.

forward(*points*)

Forward pass.

参数 **points** (*torch.Tensor*) –point coordinates with features, with shape (B, N, 3 + input_feature_dim).

返回

Outputs of the last SA module.

- sa_xyz (torch.Tensor): The coordinates of sa features.
- **sa_features (torch.Tensor): The features from the** last Set Aggregation Layers.
- **sa_indices (torch.Tensor): Indices of the** input points.

返回类型 dict[str, torch.Tensor]

class mmdet3d.models.backbones.**PointNet2SASSG**(*in_channels*, *num_points=(2048, 1024, 512, 256)*, *radius=(0.2, 0.4, 0.8, 1.2)*, *num_samples=(64, 32, 16, 16)*, *sa_channels=((64, 64, 128), (128, 128, 256), (128, 128, 256), (128, 128, 256))*, *fp_channels=((256, 256), (256, 256))*, *norm_cfg={'type': 'BN2d'}*, *sa_cfg={'normalize_xyz': True, 'pool_mod': 'max', 'type': 'PointSAModule', 'use_xyz': True}*, *init_cfg=None*)

PointNet2 with Single-scale grouping.

参数

- **in_channels** (*int*) –Input channels of point cloud.
- **num_points** (*tuple[int]*) –The number of points which each SA module samples.
- **radius** (*tuple[float]*) –Sampling radii of each SA module.
- **num_samples** (*tuple[int]*) –The number of samples for ball query in each SA module.
- **sa_channels** (*tuple[tuple[int]]*) –Out channels of each mlp in SA module.
- **fp_channels** (*tuple[tuple[int]]*) –Out channels of each mlp in FP module.
- **norm_cfg** (*dict*) –Config of normalization layer.
- **sa_cfg** (*dict*) –Config of set abstraction module, which may contain the following keys and values:
  - pool_mod (str): Pool method (‘max’ or ‘avg’) for SA modules.

- use_xyz (bool): Whether to use xyz as a part of features.
- normalize_xyz (bool): Whether to normalize xyz with radii in each SA module.

forward(*points*)

Forward pass.

参数 **points** (*torch.Tensor*) –point coordinates with features, with shape (B, N, 3 + input_feature_dim).

返回

Outputs after SA and FP modules.

- **fp_xyz (list[torch.Tensor]): The coordinates of** each fp features.
- **fp_features (list[torch.Tensor]): The features** from each Feature Propagate Layers.
- **fp_indices (list[torch.Tensor]): Indices of the** input points.

返回类型 dict[str, list[torch.Tensor]]

class mmdet3d.models.backbones.**ResNeXt**(*groups=1*, *base_width=4*, ***kwargs*)

ResNeXt backbone.

参数

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

- **zero_init_residual** (*bool*) –whether to use zero init for last norm layer in resblocks to let them behave as identity.

make_res_layer(***kwargs*)

Pack all blocks in a stage into a `ResLayer`

class mmdet3d.models.backbones.**ResNet**(*depth*, *in_channels=3*, *stem_channels=None*, *base_channels=64*, *num_stages=4*, *strides=(1, 2, 2, 2)*, *dilations=(1, 1, 1, 1)*, *out_indices=(0, 1, 2, 3)*, *style='pytorch'*, *deep_stem=False*, *avg_down=False*, *frozen_stages=- 1*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *norm_eval=True*, *dcn=None*, *stage_with_dcn=(False, False, False, False)*, *plugins=None*, *with_cp=False*, *zero_init_residual=True*, *pretrained=None*, *init_cfg=None*)

ResNet backbone.

参数

- **depth** (*int*) –Depth of resnet, from {18, 34, 50, 101, 152}.
- **stem_channels** (*int | None*) –Number of stem channels. If not specified, it will be the same as *base_channels*. Default: None.
- **base_channels** (*int*) –Number of base channels of res layer. Default: 64.
- **in_channels** (*int*) –Number of input image channels. Default: 3.
- **num_stages** (*int*) –Resnet stages. Default: 4.
- **strides** (*Sequence[int]*) –Strides of the first block of each stage.
- **dilations** (*Sequence[int]*) –Dilation of each stage.
- **out_indices** (*Sequence[int]*) –Output from which stages.
- **style** (*str*) –*pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer.
- **deep_stem** (*bool*) –Replace 7x7 conv in input stem with 3 3x3 conv
- **avg_down** (*bool*) –Use AvgPool instead of stride conv when downsampling in the bottleneck.
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters.
- **norm_cfg** (*dict*) –Dictionary to construct and config norm layer.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only.
- **plugins** (*list[dict]*) –List of plugins for stages, each dict contains:

  - cfg (dict, required): Cfg dict to build plugin.
  - position (str, required): Position inside block to insert plugin, options are 'after_conv1', 'after_conv2', 'after_conv3'.
  - stages (tuple[bool], optional): Stages to apply plugin, length should be same as 'num_stages'.

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

示例

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

forward(*x*)

Forward function.

make_res_layer(***kwargs*)

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

make_stage_plugins(*plugins*, *stage_idx*)

Make plugins for ResNet `stage_idx` th stage.

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

An example of plugins format could be:

实际案例

```
>>> plugins=[
...     dict(cfg=dict(type='xxx', arg1='xxx'),
...          stages=(False, True, True, True),
...          position='after_conv2'),
...     dict(cfg=dict(type='yyy'),
...          stages=(True, True, True, True),
...          position='after_conv3'),
...     dict(cfg=dict(type='zzz', postfix='1'),
...          stages=(True, True, True, True),
...          position='after_conv3'),
...     dict(cfg=dict(type='zzz', postfix='2'),
...          stages=(True, True, True, True),
...          position='after_conv3')
... ]
>>> self = ResNet(depth=18)
>>> stage_plugins = self.make_stage_plugins(plugins, 0)
>>> assert len(stage_plugins) == 3
```

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

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

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

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

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

参数

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

返回 Plugins for current stage

返回类型 list[dict]

property norm1

the normalization layer named “norm1”

Type nn.Module

train(*mode=True*)

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

class mmdet3d.models.backbones.**ResNetV1d**(***kwargs*)

ResNetV1d variant described in Bag of Tricks.

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

class mmdet3d.models.backbones.**SECOND**(*in_channels=128*, *out_channels=[128, 128, 256]*, *layer_nums=[3, 5, 5]*, *layer_strides=[2, 2, 2]*, *norm_cfg={'eps': 0.001, 'momentum': 0.01, 'type': 'BN'}*, *conv_cfg={'bias': False, 'type': 'Conv2d'}*, *init_cfg=None*, *pretrained=None*)

Backbone network for SECOND/PointPillars/PartA2/MVXNet.

参数

- **in_channels** (*int*) –Input channels.
- **out_channels** (*list[int]*) –Output channels for multi-scale feature maps.
- **layer_nums** (*list[int]*) –Number of layers in each stage.
- **layer_strides** (*list[int]*) –Strides of each stage.
- **norm_cfg** (*dict*) –Config dict of normalization layers.
- **conv_cfg** (*dict*) –Config dict of convolutional layers.

forward(*x*)

Forward function.

参数 **x** (*torch.Tensor*) –Input with shape (N, C, H, W).

返回 Multi-scale features.

返回类型 tuple[torch.Tensor]

class mmdet3d.models.backbones.**SSDVGG**(*depth*, *with_last_pool=False*, *ceil_mode=True*, *out_indices=(3, 4)*, *out_feature_indices=(22, 34)*, *pretrained=None*, *init_cfg=None*, *input_size=None*, *l2_norm_scale=None*)

VGG Backbone network for single-shot-detection.

参数

- **depth** (*int*) –Depth of vgg, from {11, 13, 16, 19}.
- **with_last_pool** (*bool*) –Whether to add a pooling layer at the last of the model
- **ceil_mode** (*bool*) –When True, will use *ceil* instead of *floor* to compute the output shape.
- **out_indices** (*Sequence[int]*) –Output from which stages.

- **out_feature_indices** (*Sequence[int]*) –Output from which feature map.
- **pretrained** (*str, optional*) –model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None
- **input_size** (*int, optional*) –Deprecated argumment. Width and height of input, from {300, 512}.
- **l2_norm_scale** (*float, optional*) –Deprecated argumment. L2 normalization layer init scale.

示例

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

forward(*x*)

Forward function.

init_weights(*pretrained=None*)

Initialize the weights.

40.3 necks

class mmdet3d.models.necks.**DLANeck**(*in_channels=[16, 32, 64, 128, 256, 512]*, *start_level=2*, *end_level=5*, *norm_cfg=None*, *use_dcn=True*, *init_cfg=None*)

DLA Neck.

参数

- **in_channels** (*list[int], optional*) –List of input channels of multi-scale feature map.
- **start_level** (*int, optional*) –The scale level where upsampling starts. Default: 2.

- **end_level** (*int, optional*) –The scale level where upsampling ends. Default: 5.
- **norm_cfg** (*dict, optional*) –Config dict for normalization layer. Default: None.
- **use_dcn** (*bool, optional*) –Whether to use dcn in IDAup module. Default: True.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

init_weights()

Initialize the weights.

class mmdet3d.models.necks.**FPN**(*in_channels*, *out_channels*, *num_outs*, *start_level=0*, *end_level=-1*, *add_extra_convs=False*, *relu_before_extra_convs=False*, *no_norm_on_lateral=False*, *conv_cfg=None*, *norm_cfg=None*, *act_cfg=None*, *upsample_cfg={'mode': 'nearest'}*, *init_cfg={'distribution': 'uniform', 'layer': 'Conv2d', 'type': 'Xavier'}*)

Feature Pyramid Network.

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

参数

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

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

示例

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

forward(*inputs*)

Forward function.

class mmdet3d.models.necks.**LSSViewTransformer**(*grid_config*, *input_size*, *downsample*, *in_channels*, *out_channels*, *accelerate=False*, *max_voxel_points=300*)

Lift-Splat-Shoot view transformer.

Please refer to the paper

参数

- **grid_config** (*dict*) –Config of grid alone each axis in format of (lower_bound, upper_bound, interval). axis in {x,y,z,depth}.
- **input_size** (*tuple(int)*) –Size of input images in format of (height, width).
- **downsample** (*int*) –Down sample factor from the input size to the feature size.

- **in_channels** (*int*) –Channels of input feature.
- **out_channels** (*int*) –Channels of transformed feature.
- **accelerate** (*bool*) –Whether the view transformation is conducted with acceleration. Note: the intrinsic and extrinsic of cameras should be constant when 'accelerate' is set true.
- **max_voxel_points** (*int*) –Specify the maximum point number in a single voxel during the acceleration.

create_frustum(*depth_cfg*, *input_size*, *downsample*)

Generate the frustum template for each image.

参数

- **depth_cfg** (*tuple(float)*) –Config of grid alone depth axis in format of (lower_bound, upper_bound, interval).
- **input_size** (*tuple(int)*) –Size of input images in format of (height, width).
- **downsample** (*int*) –Down sample scale factor from the input size to the feature size.

create_grid_infos(*x*, *y*, *z*, ***kwargs*)

Generate the grid information including the lower bound, interval, and size.

参数

- **x** (*tuple(float)*) –Config of grid alone x axis in format of (lower_bound, upper_bound, interval).
- **y** (*tuple(float)*) –Config of grid alone y axis in format of (lower_bound, upper_bound, interval).
- **z** (*tuple(float)*) –Config of grid alone z axis in format of (lower_bound, upper_bound, interval).
- ****kwargs** –Container for other potential parameters

forward(*input*)

Transform image-view feature into bird-eye-view feature.

参数 **input** (*list(torch.tensor)*) –of (image-view feature, rots, trans, intrins, post_rots, post_trans)

返回 Bird-eye-view feature in shape (B, C, H_BEV, W_BEV)

返回类型 torch.tensor

get_lidar_coor(*rots*, *trans*, *cam2imgs*, *post_rots*, *post_trans*)

Calculate the locations of the frustum points in the lidar coordinate system.

参数

- **rots** (*torch.Tensor*) –Rotation from camera coordinate system to lidar coordinate system in shape (B, N_cams, 3, 3).
- **trans** (*torch.Tensor*) –Translation from camera coordinate system to lidar coordinate system in shape (B, N_cams, 3).
- **cam2imgs** (*torch.Tensor*) –Camera intrinsic matrixes in shape (B, N_cams, 3, 3).
- **post_rots** (*torch.Tensor*) –Rotation in camera coordinate system in shape (B, N_cams, 3, 3). It is derived from the image view augmentation.
- **post_trans** (*torch.Tensor*) –Translation in camera coordinate system derived from image view augmentation in shape (B, N_cams, 3).

返回

Point coordinates in shape (B, N_cams, D, ownsample, 3)

返回类型 torch.tensor

init_acceleration(*coor*, *x*)

Pre-compute the necessary information in acceleration including the index of points in the final feature.

参数

- **coor** (*torch.tensor*) –Coordinate of points in lidar space in shape (B, N_cams, D, H, W, 3).
- **x** (*torch.tensor*) –Feature of points in shape (B, N_cams, D, H, W, C).

voxel_pooling(*coor*, *x*)

Generate bird-eye-view features with the pseudo point cloud.

参数

- **coor** (*torch.tensor*) –Coordinate of points in the lidar space in shape (B, N_cams, D, H, W, 3).
- **x** (*torch.tensor*) –Feature of points in shape (B, N_cams, D, H, W, C).

返回 Bird-eye-view features in shape (B, C, H_BEV, W_BEV).

返回类型 torch.tensor

voxel_pooling_accelerated(*x*)

Conducting voxel pooling in accelerated mode.

参数 **x** (*torch.tensor*) –The feature of the volumes in shape (B, N_cams, D, H, W, C).

返回 Bird-eye-view features in shape (B, C, H_BEV, W_BEV).

返回类型 torch.tensor

voxel_pooling_prepare(*coor*, *x*)

Data preparation for voxel pooling.

参数

- **coor** (*torch.tensor*) –Coordinate of points in the lidar space in shape (B, N, D, H, W, 3).
- **x** (*torch.tensor*) –Feature of points in shape (B, N_cams, D, H, W, C).

返回

Feature of points in shape (N_Points, C); Coordinate of points in the voxel space in shape (N_Points, 3); Rank of the voxel that a point is belong to in shape (N_Points); Reserved index of points in the input point queue in shape (N_Points).

返回类型 tuple[torch.tensor]

class mmdet3d.models.necks.**OutdoorImVoxelNeck**(*in_channels*, *out_channels*)

Neck for ImVoxelNet outdoor scenario.

参数

- **in_channels** (*int*) –Input channels of multi-scale feature map.
- **out_channels** (*int*) –Output channels of multi-scale feature map.

forward(*x*)

Forward function.

参数 **x** (*torch.Tensor*) –of shape (N, C_in, N_x, N_y, N_z).

返回 of shape (N, C_out, N_y, N_x).

返回类型 list[torch.Tensor]

init_weights()

Initialize weights of neck.

class mmdet3d.models.necks.**PointNetFPNeck**(*fp_channels*, *init_cfg=None*)

PointNet FP Module used in PointRCNN.

Refer to the official code.

```
sa_n ----------------------------------------
                                              |
... ----------------------------------         |
                                    |         |
sa_1 -------------                  |         |
                 |                  |         |
sa_0 -> fp_0 -> fp_module ->fp_1 -> ... -> fp_module -> fp_n
```

sa_n including sa_xyz (torch.Tensor) and sa_features (torch.Tensor) fp_n including fp_xyz (torch.Tensor) and fp_features (torch.Tensor)

参数

- **fp_channels** (*tuple[tuple[int]]*) –Tuple of mlp channels in FP modules.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None

forward(*feat_dict*)

Forward pass.

参数 **feat_dict** (*dict*) –Feature dict from backbone.

返回

Outputs of the Neck.

- fp_xyz (torch.Tensor): The coordinates of fp features.
- **fp_features (torch.Tensor): The features from the last** feature propagation layers.

返回类型 dict[str, torch.Tensor]

class mmdet3d.models.necks.**SECONDFPN**(*in_channels=[128, 128, 256]*, *out_channels=[256, 256, 256]*, *upsample_strides=[1, 2, 4]*, *norm_cfg={'eps': 0.001, 'momentum': 0.01, 'type': 'BN'}*, *upsample_cfg={'bias': False, 'type': 'deconv'}*, *conv_cfg={'bias': False, 'type': 'Conv2d'}*, *use_conv_for_no_stride=False*, *init_cfg=None*)

FPN used in SECOND/PointPillars/PartA2/MVXNet.

参数

- **in_channels** (*list[int]*) –Input channels of multi-scale feature maps.
- **out_channels** (*list[int]*) –Output channels of feature maps.
- **upsample_strides** (*list[int]*) –Strides used to upsample the feature maps.
- **norm_cfg** (*dict*) –Config dict of normalization layers.
- **upsample_cfg** (*dict*) –Config dict of upsample layers.
- **conv_cfg** (*dict*) –Config dict of conv layers.
- **use_conv_for_no_stride** (*bool*) –Whether to use conv when stride is 1.

forward(*x*)

Forward function.

参数 **x** (*torch.Tensor*) –4D Tensor in (N, C, H, W) shape.

返回 Multi-level feature maps.

返回类型 list[torch.Tensor]

40.4 dense_heads

class mmdet3d.models.dense_heads.**Anchor3DHead**(*num_classes*, *in_channels*, *train_cfg*, *test_cfg*, *feat_channels=256*, *use_direction_classifier=True*, *anchor_generator={'custom_values': [], 'range': [0, - 39.68, - 1.78, 69.12, 39.68, - 1.78], 'reshape_out': False, 'rotations': [0, 1.57], 'sizes': [[3.9, 1.6, 1.56]], 'strides': [2], 'type': 'Anchor3DRangeGenerator'}*, *assigner_per_size=False*, *assign_per_class=False*, *diff_rad_by_sin=True*, *dir_offset=-1.5707963267948966*, *dir_limit_offset=0*, *bbox_coder={'type': 'DeltaXYZWLHRBBoxCoder'}*, *loss_cls={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, *loss_bbox={'beta': 0.1111111111111111, 'loss_weight': 2.0, 'type': 'SmoothL1Loss'}*, *loss_dir={'loss_weight': 0.2, 'type': 'CrossEntropyLoss'}*, *init_cfg=None*)

Anchor head for SECOND/PointPillars/MVXNet/PartA2.

参数

- **num_classes** (*int*) –Number of classes.
- **in_channels** (*int*) –Number of channels in the input feature map.
- **train_cfg** (*dict*) –Train configs.
- **test_cfg** (*dict*) –Test configs.
- **feat_channels** (*int*) –Number of channels of the feature map.
- **use_direction_classifier** (*bool*) –Whether to add a direction classifier.
- **anchor_generator** (*dict*) –Config dict of anchor generator.
- **assigner_per_size** (*bool*) –Whether to do assignment for each separate anchor size.
- **assign_per_class** (*bool*) –Whether to do assignment for each class.
- **diff_rad_by_sin** (*bool*) –Whether to change the difference into sin difference for box regression loss.

- **dir_offset** (*float | int*) –The offset of BEV rotation angles. (TODO: may be moved into box coder)
- **dir_limit_offset** (*float | int*) –The limited range of BEV rotation angles. (TODO: may be moved into box coder)
- **bbox_coder** (*dict*) –Config dict of box coders.
- **loss_cls** (*dict*) –Config of classification loss.
- **loss_bbox** (*dict*) –Config of localization loss.
- **loss_dir** (*dict*) –Config of direction classifier loss.

static add_sin_difference(*boxes1*, *boxes2*)

Convert the rotation difference to difference in sine function.

参数

- **boxes1** (*torch.Tensor*) –Original Boxes in shape (NxC), where C>=7 and the 7th dimension is rotation dimension.
- **boxes2** (*torch.Tensor*) –Target boxes in shape (NxC), where C>=7 and the 7th dimension is rotation dimension.

返回

boxes1 and boxes2 whose 7th dimensions are changed.

返回类型 tuple[torch.Tensor]

forward(*feats*)

Forward pass.

参数 **feats** (*list[torch.Tensor]*) –Multi-level features, e.g., features produced by FPN.

返回

Multi-level class score, bbox and direction predictions.

返回类型 tuple[list[torch.Tensor]]

forward_single(*x*)

Forward function on a single-scale feature map.

参数 **x** (*torch.Tensor*) –Input features.

返回

Contain score of each class, bbox regression and direction classification predictions.

返回类型 tuple[torch.Tensor]

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

Get anchors according to feature map sizes.

参数

- **featmap_sizes** (*list[tuple]*) –Multi-level feature map sizes.
- **input_metas** (*list[dict]*) –contain pcd and img’s meta info.
- **device** (*str*) –device of current module.

返回

Anchors of each image, valid flags of each image.

返回类型 list[list[torch.Tensor]]

get_bboxes(*cls_scores*, *bbox_preds*, *dir_cls_preds*, *input_metas*, *cfg=None*, *rescale=False*)

Get bboxes of anchor head.

参数

- **cls_scores** (*list[torch.Tensor]*) –Multi-level class scores.
- **bbox_preds** (*list[torch.Tensor]*) –Multi-level bbox predictions.
- **dir_cls_preds** (*list[torch.Tensor]*) –Multi-level direction class predictions.
- **input_metas** (*list[dict]*) –Contain pcd and img’s meta info.
- **cfg** (ConfigDict) –Training or testing config.
- **rescale** (*list[torch.Tensor]*) –Whether th rescale bbox.

返回 Prediction resultes of batches.

返回类型 list[tuple]

get_bboxes_single(*cls_scores*, *bbox_preds*, *dir_cls_preds*, *mlvl_anchors*, *input_meta*, *cfg=None*, *rescale=False*)

Get bboxes of single branch.

参数

- **cls_scores** (*torch.Tensor*) –Class score in single batch.
- **bbox_preds** (*torch.Tensor*) –Bbox prediction in single batch.
- **dir_cls_preds** (*torch.Tensor*) –Predictions of direction class in single batch.
- **mlvl_anchors** (*List[torch.Tensor]*) –Multi-level anchors in single batch.
- **input_meta** (*list[dict]*) –Contain pcd and img’s meta info.
- **cfg** (ConfigDict) –Training or testing config.
- **rescale** (*list[torch.Tensor]*) –whether th rescale bbox.

返回

Contain predictions of single batch.

- bboxes (BaseInstance3DBoxes): Predicted 3d bboxes.
- scores (torch.Tensor): Class score of each bbox.
- labels (torch.Tensor): Label of each bbox.

返回类型 tuple

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

Calculate losses.

参数

- **cls_scores** (*list[torch.Tensor]*) –Multi-level class scores.
- **bbox_preds** (*list[torch.Tensor]*) –Multi-level bbox predictions.
- **dir_cls_preds** (*list[torch.Tensor]*) –Multi-level direction class predictions.
- **gt_bboxes** (list[BaseInstance3DBoxes]) –Gt bboxes of each sample.
- **gt_labels** (*list[torch.Tensor]*) –Gt labels of each sample.
- **input_metas** (*list[dict]*) –Contain pcd and img's meta info.
- **gt_bboxes_ignore** (*list[torch.Tensor]*) –Specify which bounding boxes to ignore.

返回

Classification, bbox, and direction losses of each level.

- loss_cls (list[torch.Tensor]): Classification losses.
- loss_bbox (list[torch.Tensor]): Box regression losses.
- **loss_dir (list[torch.Tensor]): Direction classification** losses.

返回类型 dict[str, list[torch.Tensor]]

loss_single(*cls_score*, *bbox_pred*, *dir_cls_preds*, *labels*, *label_weights*, *bbox_targets*, *bbox_weights*, *dir_targets*, *dir_weights*, *num_total_samples*)

Calculate loss of Single-level results.

参数

- **cls_score** (*torch.Tensor*) –Class score in single-level.
- **bbox_pred** (*torch.Tensor*) –Bbox prediction in single-level.
- **dir_cls_preds** (*torch.Tensor*) –Predictions of direction class in single-level.
- **labels** (*torch.Tensor*) –Labels of class.

- **label_weights** (*torch.Tensor*) –Weights of class loss.
- **bbox_targets** (*torch.Tensor*) –Targets of bbox predictions.
- **bbox_weights** (*torch.Tensor*) –Weights of bbox loss.
- **dir_targets** (*torch.Tensor*) –Targets of direction predictions.
- **dir_weights** (*torch.Tensor*) –Weights of direction loss.
- **num_total_samples** (*int*) –The number of valid samples.

返回

Losses of class, bbox and direction, respectively.

返回类型 tuple[torch.Tensor]

class mmdet3d.models.dense_heads.**AnchorFreeMono3DHead**(*num_classes*, *in_channels*, *feat_channels=256*, *stacked_convs=4*, *strides=(4, 8, 16, 32, 64)*, *dcn_on_last_conv=False*, *conv_bias='auto'*, *background_label=None*, *use_direction_classifier=True*, *diff_rad_by_sin=True*, *dir_offset=0*, *dir_limit_offset=0*, *loss_cls={'alpha': 0.25, 'gamma': 2.0, 'loss_weight': 1.0, 'type': 'FocalLoss', 'use_sigmoid': True}*, *loss_bbox={'beta': 0.1111111111111111, 'loss_weight': 1.0, 'type': 'SmoothL1Loss'}*, *loss_dir={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_sigmoid': False}*, *loss_attr={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_sigmoid': False}*, *bbox_code_size=9*, *pred_attrs=False*, *num_attrs=9*, *pred_velo=False*, *pred_bbox2d=False*, *group_reg_dims=(2, 1, 3, 1, 2)*, *cls_branch=(128, 64)*, *reg_branch=((128, 64), (128, 64), (64), (64), ())*, *dir_branch=(64)*, *attr_branch=(64)*, *conv_cfg=None*, *norm_cfg=None*, *train_cfg=None*, *test_cfg=None*, *init_cfg=None*)

Anchor-free head for monocular 3D object detection.

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, optional*) –Number of hidden channels. Used in child classes. Defaults to 256.
- **stacked_convs** (*int, optional*) –Number of stacking convs of the head.
- **strides** (*tuple, optional*) –Downsample factor of each feature map.
- **dcn_on_last_conv** (*bool, optional*) –If true, use dcn in the last layer of towers. Default: False.

- **conv_bias** (*bool | str, optional*) –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'.
- **background_label** (*int, optional*) –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.
- **use_direction_classifier** (*bool, optional*) –Whether to add a direction classifier.
- **diff_rad_by_sin** (*bool, optional*) –Whether to change the difference into sin difference for box regression loss. Defaults to True.
- **dir_offset** (*float, optional*) –Parameter used in direction classification. Defaults to 0.
- **dir_limit_offset** (*float, optional*) –Parameter used in direction classification. Defaults to 0.
- **loss_cls** (*dict, optional*) –Config of classification loss.
- **loss_bbox** (*dict, optional*) –Config of localization loss.
- **loss_dir** (*dict, optional*) –Config of direction classifier loss.
- **loss_attr** (*dict, optional*) –Config of attribute classifier loss, which is only active when *pred_attrs=True*.
- **bbox_code_size** (*int, optional*) –Dimensions of predicted bounding boxes.
- **pred_attrs** (*bool, optional*) –Whether to predict attributes. Defaults to False.
- **num_attrs** (*int, optional*) –The number of attributes to be predicted. Default: 9.
- **pred_velo** (*bool, optional*) –Whether to predict velocity. Defaults to False.
- **pred_bbox2d** (*bool, optional*) –Whether to predict 2D boxes. Defaults to False.
- **group_reg_dims** (*tuple[int], optional*) –The dimension of each regression target group. Default: (2, 1, 3, 1, 2).
- **cls_branch** (*tuple[int], optional*) –Channels for classification branch. Default: (128, 64).
- **reg_branch** (*tuple[tuple], optional*) –Channels for regression branch. Default: (

  (128, 64), # offset (128, 64), # depth (64,), # size (64,), # rot () # velo

  ),
- **dir_branch** (*tuple[int], optional*) –Channels for direction classification branch. Default: (64,).

- **attr_branch** (*tuple[int], optional*) –Channels for classification branch. Default: (64,).
- **conv_cfg** (*dict, optional*) –Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict, optional*) –Config dict for normalization layer. Default: None.
- **train_cfg** (*dict, optional*) –Training config of anchor head.
- **test_cfg** (*dict, optional*) –Testing config of anchor head.

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

Usually contain classification scores, bbox predictions, and direction class predictions. cls_scores (list[Tensor]): Box scores for each scale level,

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

bbox_preds (list[Tensor]): Box energies / deltas for each scale level, each is a 4D-tensor, the channel number is num_points * bbox_code_size.

dir_cls_preds (list[Tensor]): Box scores for direction class predictions on each scale level, each is a 4D-tensor, the channel number is num_points * 2. (bin = 2)

attr_preds (list[Tensor]): Attribute scores for each scale level, each is a 4D-tensor, the channel number is num_points * num_attrs.

返回类型 tuple

forward_single(*x*)

Forward features of a single scale level.

参数 **x** (*Tensor*) –FPN feature maps of the specified stride.

返回

Scores for each class, bbox predictions, direction class, and attributes, features after classification and regression conv layers, some models needs these features like FCOS.

返回类型 tuple

abstract get_bboxes(*cls_scores*, *bbox_preds*, *dir_cls_preds*, *attr_preds*, *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 * bbox_code_size, H, W)
- **dir_cls_preds** (*list[Tensor]*) –Box scores for direction class predictions on each scale level, each is a 4D-tensor, the channel number is num_points * 2. (bin = 2)
- **attr_preds** (*list[Tensor]*) –Attribute scores for each scale level Has shape (N, num_points * num_attrs, 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

get_points(*featmap_sizes*, *dtype*, *device*, *flatten=False*)

Get points according to feature map sizes.

参数

- **featmap_sizes** (*list[tuple]*) –Multi-level feature map sizes.
- **dtype** (*torch.dtype*) –Type of points.
- **device** (*torch.device*) –Device of points.

返回 points of each image.

返回类型 tuple

abstract get_targets(*points*, *gt_bboxes_list*, *gt_labels_list*, *gt_bboxes_3d_list*, *gt_labels_3d_list*, *centers2d_list*, *depths_list*, *attr_labels_list*)

Compute regression, classification and centerss 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,).
- **gt_bboxes_3d_list** (*list[Tensor]*) –3D Ground truth bboxes of each image, each has shape (num_gt, bbox_code_size).
- **gt_labels_3d_list** (*list[Tensor]*) –3D Ground truth labels of each box, each has shape (num_gt,).

- **centers2d_list** (*list[Tensor]*) –Projected 3D centers onto 2D image, each has shape (num_gt, 2).
- **depths_list** (*list[Tensor]*) –Depth of projected 3D centers onto 2D image, each has shape (num_gt, 1).
- **attr_labels_list** (*list[Tensor]*) –Attribute labels of each box, each has shape (num_gt,).

init_weights()

Initialize weights of the head.

We currently still use the customized defined init_weights because the default init of DCN triggered by the init_cfg will init conv_offset.weight, which mistakenly affects the training stability.

abstract loss(*cls_scores*, *bbox_preds*, *dir_cls_preds*, *attr_preds*, *gt_bboxes*, *gt_labels*, *gt_bboxes_3d*, *gt_labels_3d*, *centers2d*, *depths*, *attr_labels*, *img_metas*, *gt_bboxes_ignore=None*)

Compute loss of the head.

参数

- **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 * bbox_code_size.
- **dir_cls_preds** (*list[Tensor]*) –Box scores for direction class predictions on each scale level, each is a 4D-tensor, the channel number is num_points * 2. (bin = 2)
- **attr_preds** (*list[Tensor]*) –Box scores for each scale level, each is a 4D-tensor, the channel number is num_points * num_attrs.
- **gt_bboxes** (*list[Tensor]*) –Ground truth bboxes for each image with shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) –class indices corresponding to each box
- **gt_bboxes_3d** (*list[Tensor]*) –3D Ground truth bboxes for each image with shape (num_gts, bbox_code_size).
- **gt_labels_3d** (*list[Tensor]*) –3D class indices of each box.
- **centers2d** (*list[Tensor]*) –Projected 3D centers onto 2D images.
- **depths** (*list[Tensor]*) –Depth of projected centers on 2D images.
- **attr_labels** (*list[Tensor], optional*) –Attribute indices corresponding to each box
- **img_metas** (*list[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.

- **gt_bboxes_ignore** (*list[Tensor]*) –specify which bounding boxes can be ignored when computing the loss.

class mmdet3d.models.dense_heads.**BaseConvBboxHead**(*in_channels=0*, *shared_conv_channels=()*, *cls_conv_channels=()*, *num_cls_out_channels=0*, *reg_conv_channels=()*, *num_reg_out_channels=0*, *conv_cfg={'type': 'Conv1d'}*, *norm_cfg={'type': 'BN1d'}*, *act_cfg={'type': 'ReLU'}*, *bias='auto'*, *init_cfg=None*, **args*, ***kwargs*)

More general bbox head, with shared conv layers and two optional separated branches.

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

forward(*feats*)

Forward.

参数 **feats** (*Tensor*) –Input features

返回 Class scores predictions Tensor: Regression predictions

返回类型 Tensor

class mmdet3d.models.dense_heads.**BaseMono3DDenseHead**(*init_cfg=None*)

Base class for Monocular 3D DenseHeads.

forward_train(*x*, *img_metas*, *gt_bboxes*, *gt_labels=None*, *gt_bboxes_3d=None*, *gt_labels_3d=None*, *centers2d=None*, *depths=None*, *attr_labels=None*, *gt_bboxes_ignore=None*, *proposal_cfg=None*, ***kwargs*)

参数

- **x** (*list[Tensor]*) –Features from FPN.
- **img_metas** (*list[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes** (*list[Tensor]*) –Ground truth bboxes of the image, shape (num_gts, 4).
- **gt_labels** (*list[Tensor]*) –Ground truth labels of each box, shape (num_gts,).
- **gt_bboxes_3d** (*list[Tensor]*) –3D ground truth bboxes of the image, shape (num_gts, self.bbox_code_size).

- **gt_labels_3d** (*list[Tensor]*) –3D ground truth labels of each box, shape (num_gts,).
- **centers2d** (*list[Tensor]*) –Projected 3D center of each box, shape (num_gts, 2).
- **depths** (*list[Tensor]*) –Depth of projected 3D center of each box, shape (num_gts,).
- **attr_labels** (*list[Tensor]*) –Attribute labels of each box, shape (num_gts,).
- **gt_bboxes_ignore** (*list[Tensor]*) –Ground truth bboxes to be ignored, shape (num_ignored_gts, 4).
- **proposal_cfg** (*mmcv.Config*) –Test / postprocessing configuration, if None, test_cfg would be used

返回 losses: (dict[str, Tensor]): A dictionary of loss components. proposal_list (list[Tensor]): Proposals of each image.

返回类型 tuple

abstract get_bboxes(***kwargs*)

Transform network output for a batch into bbox predictions.

abstract loss(***kwargs*)

Compute losses of the head.

class mmdet3d.models.dense_heads.**CenterHead**(*in_channels=[128]*, *tasks=None*, *train_cfg=None*, *test_cfg=None*, *bbox_coder=None*, *common_heads={}*, *loss_cls={'reduction': 'mean', 'type': 'GaussianFocalLoss'}*, *loss_bbox={'loss_weight': 0.25, 'reduction': 'none', 'type': 'L1Loss'}*, *separate_head={'final_kernel': 3, 'init_bias': - 2.19, 'type': 'SeparateHead'}*, *share_conv_channel=64*, *num_heatmap_convs=2*, *conv_cfg={'type': 'Conv2d'}*, *norm_cfg={'type': 'BN2d'}*, *bias='auto'*, *norm_bbox=True*, *init_cfg=None*)

CenterHead for CenterPoint.

参数

- **in_channels** (*list[int] | int, optional*) –Channels of the input feature map. Default: [128].
- **tasks** (*list[dict], optional*) –Task information including class number and class names. Default: None.
- **train_cfg** (*dict, optional*) –Train-time configs. Default: None.

- **test_cfg** (*dict, optional*) –Test-time configs. Default: None.
- **bbox_coder** (*dict, optional*) –Bbox coder configs. Default: None.
- **common_heads** (*dict, optional*) –Conv information for common heads. Default: dict().
- **loss_cls** (*dict, optional*) –Config of classification loss function. Default: dict(type='GaussianFocalLoss', reduction='mean').
- **loss_bbox** (*dict, optional*) –Config of regression loss function. Default: dict(type='L1Loss', reduction='none').
- **separate_head** (*dict, optional*) –Config of separate head. Default: dict(type='SeparateHead', init_bias=-2.19, final_kernel=3)
- **share_conv_channel** (*int, optional*) –Output channels for share_conv layer. Default: 64.
- **num_heatmap_convs** (*int, optional*) –Number of conv layers for heatmap conv layer. Default: 2.
- **conv_cfg** (*dict, optional*) –Config of conv layer. Default: dict(type='Conv2d')
- **norm_cfg** (*dict, optional*) –Config of norm layer. Default: dict(type='BN2d').
- **bias** (*str, optional*) –Type of bias. Default: 'auto'.

forward(*feats*)

Forward pass.

参数 **feats** (*list[torch.Tensor]*) –Multi-level features, e.g., features produced by FPN.

返回 Output results for tasks.

返回类型 tuple(list[dict])

forward_single(*x*)

Forward function for CenterPoint.

参数 **x** (*torch.Tensor*) –Input feature map with the shape of [B, 512, 128, 128].

返回 Output results for tasks.

返回类型 list[dict]

get_bboxes(*preds_dicts*, *img_metas*, *img=None*, *rescale=False*)

Generate bboxes from bbox head predictions.

参数

- **preds_dicts** (*tuple[list[dict]]*) –Prediction results.

- **img_metas** (*list[dict]*) –Point cloud and image's meta info.

返回 Decoded bbox, scores and labels after nms.

返回类型 list[dict]

get_targets(*gt_bboxes_3d*, *gt_labels_3d*)

Generate targets.

How each output is transformed:

Each nested list is transposed so that all same-index elements in each sub-list (1, ···, N) become the new sub-lists.

[[a0, a1, a2, ···], [b0, b1, b2, ···], ···] ==> [[a0, b0, ···], [a1, b1, ···], [a2, b2, ···]]

The new transposed nested list is converted into a list of N tensors generated by concatenating tensors in the new sub-lists.

[tensor0, tensor1, tensor2, ···]

参数

- **gt_bboxes_3d** (list[LiDARInstance3DBoxes]) –Ground truth gt boxes.
- **gt_labels_3d** (*list[torch.Tensor]*) –Labels of boxes.

返回

tuple[list[torch.Tensor]: Tuple of target including the following results in order.

- list[torch.Tensor]: Heatmap scores.
- list[torch.Tensor]: Ground truth boxes.
- **list[torch.Tensor]: Indexes indicating the** position of the valid boxes.
- **list[torch.Tensor]: Masks indicating which** boxes are valid.

返回类型 Returns

get_targets_single(*gt_bboxes_3d*, *gt_labels_3d*)

Generate training targets for a single sample.

参数

- **gt_bboxes_3d** (LiDARInstance3DBoxes) –Ground truth gt boxes.
- **gt_labels_3d** (*torch.Tensor*) –Labels of boxes.

返回

Tuple of target including the following results in order.

- list[torch.Tensor]: Heatmap scores.
- list[torch.Tensor]: Ground truth boxes.

- **list[torch.Tensor]: Indexes indicating the position** of the valid boxes.
- **list[torch.Tensor]: Masks indicating which boxes** are valid.

返回类型 tuple[list[torch.Tensor]]

get_task_detections(*num_class_with_bg*, *batch_cls_preds*, *batch_reg_preds*, *batch_cls_labels*, *img_metas*)

Rotate nms for each task.

参数

- **num_class_with_bg** (*int*) –Number of classes for the current task.
- **batch_cls_preds** (*list[torch.Tensor]*) –Prediction score with the shape of [N].
- **batch_reg_preds** (*list[torch.Tensor]*) –Prediction bbox with the shape of [N, 9].
- **batch_cls_labels** (*list[torch.Tensor]*) –Prediction label with the shape of [N].
- **img_metas** (*list[dict]*) –Meta information of each sample.

返回

torch.Tensor]]: contains the following keys:

-bboxes (torch.Tensor): Prediction bboxes after nms with the shape of [N, 9].

-scores (torch.Tensor): Prediction scores after nms with the shape of [N].

-labels (torch.Tensor): Prediction labels after nms with the shape of [N].

返回类型 list[dict[str

loss(*gt_bboxes_3d*, *gt_labels_3d*, *preds_dicts*, ***kwargs*)

Loss function for CenterHead.

参数

- **gt_bboxes_3d** (list[LiDARInstance3DBoxes]) –Ground truth gt boxes.
- **gt_labels_3d** (*list[torch.Tensor]*) –Labels of boxes.
- **preds_dicts** (*dict*) –Output of forward function.

返回 torch.Tensor]: Loss of heatmap and bbox of each task.

返回类型 dict[str

class mmdet3d.models.dense_heads.**FCAF3DHead**(*n_classes*, *in_channels*, *out_channels*, *n_reg_outs*, *voxel_size*, *pts_prune_threshold*, *pts_assign_threshold*, *pts_center_threshold*, *center_loss={'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, *bbox_loss={'type': 'AxisAlignedIoULoss'}*, *cls_loss={'type': 'FocalLoss'}*, *train_cfg=None*, *test_cfg=None*, *init_cfg=None*)

Bbox head of FCAF3D. Actually here we store both the sparse 3D FPN and a head. The neck and the head can not be simply separated as pruning score on the i-th level of FPN requires classification scores from i+1-th level of the head.

参数

- **n_classes** (*int*) –Number of classes.
- **in_channels** (*tuple[int]*) –Number of channels in input tensors.
- **out_channels** (*int*) –Number of channels in the neck output tensors.
- **n_reg_outs** (*int*) –Number of regression layer channels.
- **voxel_size** (*float*) –Voxel size in meters.
- **pts_prune_threshold** (*int*) –Pruning threshold on each feature level.
- **pts_assign_threshold** (*int*) –Box to location assigner parameter. Assigner selects the maximum feature level with more locations inside the box than pts_assign_threshold.
- **pts_center_threshold** (*int*) –Box to location assigner parameter. After feature level for the box is determined, assigner selects pts_center_threshold locations closest to the box center.
- **center_loss** (*dict, optional*) –Config of centerness loss.
- **bbox_loss** (*dict, optional*) –Config of bbox loss.
- **cls_loss** (*dict, optional*) –Config of classification loss.
- **train_cfg** (*dict, optional*) –Config for train stage. Defaults to None.
- **test_cfg** (*dict, optional*) –Config for test stage. Defaults to None.
- **init_cfg** (*dict, optional*) –Config for weight initialization. Defaults to None.

forward(*x*)

Forward pass.

参数 **x** (*list[Tensor]*) –Features from the backbone.

返回 Predictions of the head.

返回类型 list[list[Tensor]]

forward_test(*x*, *input_metas*)

Forward pass of the test stage.

参数

- **x** (*list[SparseTensor]*) –Features from the backbone.
- **input_metas** (*list[dict]*) –Contains scene meta info for each sample.

返回 bboxes, scores and labels for each sample.

返回类型 list[list[Tensor]]

forward_train(*x*, *gt_bboxes*, *gt_labels*, *input_metas*)

Forward pass of the train stage.

参数

- **x** (*list[SparseTensor]*) –Features from the backbone.
- **gt_bboxes** (list[BaseInstance3DBoxes]) –Ground truth bboxes of each sample.
- **gt_labels** (*list[torch.Tensor]*) –Labels of each sample.
- **input_metas** (*list[dict]*) –Contains scene meta info for each sample.

返回 Centerness, bbox and classification loss values.

返回类型 dict

init_weights()

Initialize weights.

class mmdet3d.models.dense_heads.**FCOSMono3DHead**(*regress_ranges=((- 1, 48), (48, 96), (96, 192), (192, 384), (384, 100000000.0))*, *center_sampling=True*, *center_sample_radius=1.5*, *norm_on_bbox=True*, *centerness_on_reg=True*, *centerness_alpha=2.5*, *loss_cls={'alpha': 0.25, 'gamma': 2.0, 'loss_weight': 1.0, 'type': 'FocalLoss', 'use_sigmoid': True}*, *loss_bbox={'beta': 0.1111111111111111, 'loss_weight': 1.0, 'type': 'SmoothL1Loss'}*, *loss_dir={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_sigmoid': False}*, *loss_attr={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_sigmoid': False}*, *loss_centerness={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, *bbox_coder={'code_size': 9, 'type': 'FCOS3DBBoxCoder'}*, *norm_cfg={'num_groups': 32, 'requires_grad': True, 'type': 'GN'}*, *centerness_branch=(64)*, *init_cfg=None*, ***kwargs*)

Anchor-free head used in FCOS3D.

参数

- **num_classes** (*int*) –Number of categories excluding the background category.
- **in_channels** (*int*) –Number of channels in the input feature map.
- **regress_ranges** (*tuple[tuple[int, int]], optional*) –Regress range of multiple level points.
- **center_sampling** (*bool, optional*) –If true, use center sampling. Default: True.
- **center_sample_radius** (*float, optional*) –Radius of center sampling. Default: 1.5.
- **norm_on_bbox** (*bool, optional*) –If true, normalize the regression targets with FPN strides. Default: True.
- **centerness_on_reg** (*bool, optional*) –If true, position centerness on the regress branch. Please refer to https://github.com/tianzhi0549/FCOS/issues/89#issuecomment-516877042. Default: True.
- **centerness_alpha** (*int, optional*) –Parameter used to adjust the intensity at-

tenuation from the center to the periphery. Default: 2.5.

- **loss_cls** (*dict, optional*) –Config of classification loss.
- **loss_bbox** (*dict, optional*) –Config of localization loss.
- **loss_dir** (*dict, optional*) –Config of direction classification loss.
- **loss_attr** (*dict, optional*) –Config of attribute classification loss.
- **loss_centerness** (*dict, optional*) –Config of centerness loss.
- **norm_cfg** (*dict, optional*) –dictionary to construct and config norm layer. Default: norm_cfg=dict(type='GN', num_groups=32, requires_grad=True).
- **centerness_branch** (*tuple[int], optional*) –Channels for centerness branch. Default: (64,).

static add_sin_difference(*boxes1*, *boxes2*)

Convert the rotation difference to difference in sine function.

参数

- **boxes1** (*torch.Tensor*) –Original Boxes in shape (NxC), where C>=7 and the 7th dimension is rotation dimension.
- **boxes2** (*torch.Tensor*) –Target boxes in shape (NxC), where C>=7 and the 7th dimension is rotation dimension.

返回

boxes1 and boxes2 whose 7th dimensions are changed.

返回类型 tuple[torch.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

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 * bbox_code_size.

dir_cls_preds (list[Tensor]): Box scores for direction class predictions on each scale level, each is a 4D-tensor, the channel number is num_points * 2. (bin = 2).

attr_preds (list[Tensor]): Attribute scores for each scale level, each is a 4D-tensor, the channel number is num_points * num_attrs.

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 and direction class predictions, centerness predictions of input feature maps.

返回类型 tuple

get_bboxes(*cls_scores*, *bbox_preds*, *dir_cls_preds*, *attr_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)
- **dir_cls_preds** (*list[Tensor]*) –Box scores for direction class predictions on each scale level, each is a 4D-tensor, the channel number is num_points * 2. (bin = 2)
- **attr_preds** (*list[Tensor]*) –Attribute scores for each scale level Has shape (N, num_points * num_attrs, 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, 5) tensor, where the first 4 columns are bounding box positions (tl_x, tl_y, br_x, br_y) and the 5-th column is a score between 0 and 1. The second item is a (n,) tensor where each item is the predicted class label of the corresponding box.

返回类型 list[tuple[Tensor, Tensor]]

static get_direction_target(*reg_targets*, *dir_offset=0*, *dir_limit_offset=0.0*, *num_bins=2*, *one_hot=True*)

Encode direction to 0 ~ num_bins-1.

参数

- **reg_targets** (*torch.Tensor*) –Bbox regression targets.
- **dir_offset** (*int, optional*) –Direction offset. Default to 0.
- **dir_limit_offset** (*float, optional*) –Offset to set the direction range. Default to 0.0.
- **num_bins** (*int, optional*) –Number of bins to divide 2*PI. Default to 2.
- **one_hot** (*bool, optional*) –Whether to encode as one hot. Default to True.

返回 Encoded direction targets.

返回类型 torch.Tensor

get_targets(*points*, *gt_bboxes_list*, *gt_labels_list*, *gt_bboxes_3d_list*, *gt_labels_3d_list*, *centers2d_list*, *depths_list*, *attr_labels_list*)

Compute regression, classification and centerss 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,).
- **gt_bboxes_3d_list** (*list[Tensor]*) –3D Ground truth bboxes of each image, each has shape (num_gt, bbox_code_size).
- **gt_labels_3d_list** (*list[Tensor]*) –3D Ground truth labels of each box, each has shape (num_gt,).
- **centers2d_list** (*list[Tensor]*) –Projected 3D centers onto 2D image, each has shape (num_gt, 2).

- **depths_list** (*list[Tensor]*) –Depth of projected 3D centers onto 2D image, each has shape (num_gt, 1).
- **attr_labels_list** (*list[Tensor]*) –Attribute 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.

返回类型 tuple

init_weights()

Initialize weights of the head.

We currently still use the customized init_weights because the default init of DCN triggered by the init_cfg will init conv_offset.weight, which mistakenly affects the training stability.

loss(*cls_scores*, *bbox_preds*, *dir_cls_preds*, *attr_preds*, *centernesses*, *gt_bboxes*, *gt_labels*, *gt_bboxes_3d*, *gt_labels_3d*, *centers2d*, *depths*, *attr_labels*, *img_metas*, *gt_bboxes_ignore=None*)

Compute loss of the head.

参数

- **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 * bbox_code_size.
- **dir_cls_preds** (*list[Tensor]*) –Box scores for direction class predictions on each scale level, each is a 4D-tensor, the channel number is num_points * 2. (bin = 2)
- **attr_preds** (*list[Tensor]*) –Attribute scores for each scale level, each is a 4D-tensor, the channel number is num_points * num_attrs.
- **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
- **gt_bboxes_3d** (*list[Tensor]*) –3D boxes ground truth with shape of (num_gts, code_size).
- **gt_labels_3d** (*list[Tensor]*) –same as gt_labels

- **centers2d** (*list[Tensor]*) –2D centers on the image with shape of (num_gts, 2).
- **depths** (*list[Tensor]*) –Depth ground truth with shape of (num_gts,).
- **attr_labels** (*list[Tensor]*) –Attributes indices of each box.
- **img_metas** (*list[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*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]

static pts2Dto3D(*points*, *view*)

参数

- **points** (*torch.Tensor*) –points in 2D images, [N, 3], 3 corresponds with x, y in the image and depth.
- **view** (*np.ndarray*) –camera intrinsic, [3, 3]

返回

points in 3D space. [N, 3], 3 corresponds with x, y, z in 3D space.

返回类型 torch.Tensor

class mmdet3d.models.dense_heads.**FreeAnchor3DHead**(*pre_anchor_topk=50*, *bbox_thr=0.6*, *gamma=2.0*, *alpha=0.5*, *init_cfg=None*, ***kwargs*)

FreeAnchor head for 3D detection.

注解: This implementation is directly modified from the mmdet implementation. We find it also works on 3D detection with minor modification, i.e., different hyper-parameters and a additional direction classifier.

参数

- **pre_anchor_topk** (*int*) –Number of boxes that be token in each bag.
- **bbox_thr** (*float*) –The threshold of the saturated linear function. It is usually the same with the IoU threshold used in NMS.
- **gamma** (*float*) –Gamma parameter in focal loss.
- **alpha** (*float*) –Alpha parameter in focal loss.

- **kwargs** (*dict*) –Other arguments are the same as those in *Anchor3DHead*.

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

Calculate loss of FreeAnchor head.

参数

- **cls_scores** (*list[torch.Tensor]*) –Classification scores of different samples.
- **bbox_preds** (*list[torch.Tensor]*) –Box predictions of different samples
- **dir_cls_preds** (*list[torch.Tensor]*) –Direction predictions of different samples
- **gt_bboxes** (list[BaseInstance3DBoxes]) –Ground truth boxes.
- **gt_labels** (*list[torch.Tensor]*) –Ground truth labels.
- **input_metas** (*list[dict]*) –List of input meta information.
- **gt_bboxes_ignore** (list[BaseInstance3DBoxes], optional) –Ground truth boxes that should be ignored. Defaults to None.

返回

Loss items.

- positive_bag_loss (torch.Tensor): Loss of positive samples.
- negative_bag_loss (torch.Tensor): Loss of negative samples.

返回类型 dict[str, torch.Tensor]

negative_bag_loss(*cls_prob*, *box_prob*)

Generate negative bag loss.

参数

- **cls_prob** (*torch.Tensor*) –Classification probability of negative samples.
- **box_prob** (*torch.Tensor*) –Bounding box probability of negative samples.

返回 Loss of negative samples.

返回类型 torch.Tensor

positive_bag_loss(*matched_cls_prob*, *matched_box_prob*)

Generate positive bag loss.

参数

- **matched_cls_prob** (*torch.Tensor*) –Classification probability of matched positive samples.

- **matched_box_prob** (*torch.Tensor*) –Bounding box probability of matched positive samples.

返回 Loss of positive samples.

返回类型 torch.Tensor

class mmdet3d.models.dense_heads.**GroupFree3DHead**(*num_classes*, *in_channels*, *bbox_coder*, *num_decoder_layers*, *transformerlayers*, *decoder_self_posembeds={'input_channel': 6, 'num_pos_feats': 288, 'type': 'ConvBNPositionalEncoding'}*, *decoder_cross_posembeds={'input_channel': 3, 'num_pos_feats': 288, 'type': 'ConvBNPositionalEncoding'}*, *train_cfg=None*, *test_cfg=None*, *num_proposal=128*, *pred_layer_cfg=None*, *size_cls_agnostic=True*, *gt_per_seed=3*, *sampling_objectness_loss=None*, *objectness_loss=None*, *center_loss=None*, *dir_class_loss=None*, *dir_res_loss=None*, *size_class_loss=None*, *size_res_loss=None*, *size_reg_loss=None*, *semantic_loss=None*, *init_cfg=None*)

Bbox head of Group-Free 3D.

参数

- **num_classes** (*int*) –The number of class.
- **in_channels** (*int*) –The dims of input features from backbone.
- **bbox_coder** (BaseBBoxCoder) –Bbox coder for encoding and decoding boxes.
- **num_decoder_layers** (*int*) –The number of transformer decoder layers.
- **transformerlayers** (*dict*) –Config for transformer decoder.
- **train_cfg** (*dict*) –Config for training.
- **test_cfg** (*dict*) –Config for testing.
- **num_proposal** (*int*) –The number of initial sampling candidates.
- **pred_layer_cfg** (*dict*) –Config of classfication and regression prediction layers.
- **size_cls_agnostic** (*bool*) –Whether the predicted size is class-agnostic.
- **gt_per_seed** (*int*) –the number of candidate instance each point belongs to.
- **sampling_objectness_loss** (*dict*) –Config of initial sampling objectness loss.

- **objectness_loss** (*dict*) –Config of objectness loss.
- **center_loss** (*dict*) –Config of center loss.
- **dir_class_loss** (*dict*) –Config of direction classification loss.
- **dir_res_loss** (*dict*) –Config of direction residual regression loss.
- **size_class_loss** (*dict*) –Config of size classification loss.
- **size_res_loss** (*dict*) –Config of size residual regression loss.
- **size_reg_loss** (*dict*) –Config of class-agnostic size regression loss.
- **semantic_loss** (*dict*) –Config of point-wise semantic segmentation loss.

forward(*feat_dict*, *sample_mod*)

Forward pass.

注解： The forward of GroupFree3DHead is divided into 2 steps:

1. Initial object candidates sampling.
2. Iterative object box prediction by transformer decoder.

参数

- **feat_dict** (*dict*) –Feature dict from backbone.
- **sample_mod** (*str*) –sample mode for initial candidates sampling.

返回 Predictions of GroupFree3D head.

返回类型 results (dict)

get_bboxes(*points*, *bbox_preds*, *input_metas*, *rescale=False*, *use_nms=True*)

Generate bboxes from GroupFree3D head predictions.

参数

- **points** (*torch.Tensor*) –Input points.
- **bbox_preds** (*dict*) –Predictions from GroupFree3D head.
- **input_metas** (*list[dict]*) –Point cloud and image’s meta info.
- **rescale** (*bool*) –Whether to rescale bboxes.
- **use_nms** (*bool*) –Whether to apply NMS, skip nms postprocessing while using GroupFree3D head in rpn stage.

返回 Bounding boxes, scores and labels.

返回类型 list[tuple[torch.Tensor]]

get_targets(*points*, *gt_bboxes_3d*, *gt_labels_3d*, *pts_semantic_mask=None*, *pts_instance_mask=None*, *bbox_preds=None*, *max_gt_num=64*)

Generate targets of GroupFree3D head.

参数

- **points** (*list[torch.Tensor]*) –Points of each batch.
- **gt_bboxes_3d** (list[BaseInstance3DBoxes]) –Ground truth bboxes of each batch.
- **gt_labels_3d** (*list[torch.Tensor]*) –Labels of each batch.
- **pts_semantic_mask** (*list[torch.Tensor]*) –Point-wise semantic label of each batch.
- **pts_instance_mask** (*list[torch.Tensor]*) –Point-wise instance label of each batch.
- **bbox_preds** (*torch.Tensor*) –Bounding box predictions of vote head.
- **max_gt_num** (*int*) –Max number of GTs for single batch.

返回 Targets of GroupFree3D head.

返回类型 tuple[torch.Tensor]

get_targets_single(*points*, *gt_bboxes_3d*, *gt_labels_3d*, *pts_semantic_mask=None*, *pts_instance_mask=None*, *max_gt_nums=None*, *seed_points=None*, *seed_indices=None*, *candidate_indices=None*, *seed_points_obj_topk=4*)

Generate targets of GroupFree3D head for single batch.

参数

- **points** (*torch.Tensor*) –Points of each batch.
- **gt_bboxes_3d** (BaseInstance3DBoxes) –Ground truth boxes of each batch.
- **gt_labels_3d** (*torch.Tensor*) –Labels of each batch.
- **pts_semantic_mask** (*torch.Tensor*) –Point-wise semantic label of each batch.
- **pts_instance_mask** (*torch.Tensor*) –Point-wise instance label of each batch.
- **max_gt_nums** (*int*) –Max number of GTs for single batch.
- **seed_points** (*torch.Tensor*) –Coordinates of seed points.
- **seed_indices** (*torch.Tensor*) –Indices of seed points.
- **candidate_indices** (*torch.Tensor*) –Indices of object candidates.
- **seed_points_obj_topk** (*int*) –k value of k-Closest Points Sampling.

返回 Targets of GroupFree3D head.

返回类型 tuple[torch.Tensor]

init_weights()

Initialize weights of transformer decoder in GroupFree3DHead.

loss(*bbox_preds*, *points*, *gt_bboxes_3d*, *gt_labels_3d*, *pts_semantic_mask=None*, *pts_instance_mask=None*, *img_metas=None*, *gt_bboxes_ignore=None*, *ret_target=False*)

Compute loss.

参数

- **bbox_preds** (*dict*) –Predictions from forward of vote head.
- **points** (*list[torch.Tensor]*) –Input points.
- **gt_bboxes_3d** (list[BaseInstance3DBoxes]) –Ground truth bboxes of each sample.
- **gt_labels_3d** (*list[torch.Tensor]*) –Labels of each sample.
- **pts_semantic_mask** (*list[torch.Tensor]*) –Point-wise semantic mask.
- **pts_instance_mask** (*list[torch.Tensor]*) –Point-wise instance mask.
- **img_metas** (*list[dict]*) –Contain pcd and img's meta info.
- **gt_bboxes_ignore** (*list[torch.Tensor]*) –Specify which bounding.
- **ret_target** (*Bool*) –Return targets or not.

返回 Losses of GroupFree3D.

返回类型 dict

multiclass_nms_single(*obj_scores*, *sem_scores*, *bbox*, *points*, *input_meta*)

Multi-class nms in single batch.

参数

- **obj_scores** (*torch.Tensor*) –Objectness score of bounding boxes.
- **sem_scores** (*torch.Tensor*) –semantic class score of bounding boxes.
- **bbox** (*torch.Tensor*) –Predicted bounding boxes.
- **points** (*torch.Tensor*) –Input points.
- **input_meta** (*dict*) –Point cloud and image's meta info.

返回 Bounding boxes, scores and labels.

返回类型 tuple[torch.Tensor]

class mmdet3d.models.dense_heads.**MonoFlexHead**(*num_classes*, *in_channels*, *use_edge_fusion*, *edge_fusion_inds*, *edge_heatmap_ratio*, *filter_outside_objs=True*, *loss_cls={'loss_weight': 1.0, 'type': 'GaussianFocalLoss'}*, *loss_bbox={'loss_weight': 0.1, 'type': 'IoULoss'}*, *loss_dir={'loss_weight': 0.1, 'type': 'MultiBinLoss'}*, *loss_keypoints={'loss_weight': 0.1, 'type': 'L1Loss'}*, *loss_dims={'loss_weight': 0.1, 'type': 'L1Loss'}*, *loss_offsets2d={'loss_weight': 0.1, 'type': 'L1Loss'}*, *loss_direct_depth={'loss_weight': 0.1, 'type': 'L1Loss'}*, *loss_keypoints_depth={'loss_weight': 0.1, 'type': 'L1Loss'}*, *loss_combined_depth={'loss_weight': 0.1, 'type': 'L1Loss'}*, *loss_attr=None*, *bbox_coder={'code_size': 7, 'type': 'MonoFlexCoder'}*, *norm_cfg={'type': 'BN'}*, *init_cfg=None*, *init_bias=- 2.19*, ***kwargs*)

MonoFlex head used in MonoFlex

```
        / --> 3 x 3 conv --> 1 x 1 conv --> [edge fusion] --> cls
        |
        | --> 3 x 3 conv --> 1 x 1 conv --> 2d bbox
        |
        | --> 3 x 3 conv --> 1 x 1 conv --> [edge fusion] --> 2d offsets
        |
        | --> 3 x 3 conv --> 1 x 1 conv -->  keypoints offsets
        |
        | --> 3 x 3 conv --> 1 x 1 conv -->  keypoints uncertainty
feature
        | --> 3 x 3 conv --> 1 x 1 conv -->  keypoints uncertainty
        |
        | --> 3 x 3 conv --> 1 x 1 conv -->   3d dimensions
        |
        |                  |--- 1 x 1 conv -->  ori cls
        | --> 3 x 3 conv --|
        |                  |--- 1 x 1 conv -->  ori offsets
        |
        | --> 3 x 3 conv --> 1 x 1 conv -->  depth
        |
        \ --> 3 x 3 conv --> 1 x 1 conv -->  depth uncertainty
```

参数

- **use_edge_fusion** (*bool*) –Whether to use edge fusion module while feature extraction.
- **edge_fusion_inds** (*list[tuple]*) –Indices of feature to use edge fusion.
- **edge_heatmap_ratio** (*float*) –Ratio of generating target heatmap.
- **filter_outside_objs** (*bool, optional*) –Whether to filter the outside objects. Default: True.
- **loss_cls** (*dict, optional*) –Config of classification loss. Default: loss_cls=dict(type='GaussionFocalLoss', loss_weight=1.0).
- **loss_bbox** (*dict, optional*) –Config of localization loss. Default: loss_bbox=dict(type='IOULoss', loss_weight=10.0).
- **loss_dir** (*dict, optional*) –Config of direction classification loss. Default: dict(type='MultibinLoss', loss_weight=0.1).
- **loss_keypoints** (*dict, optional*) –Config of keypoints loss. Default: dict(type='L1Loss', loss_weight=0.1).
- **loss_dims** –(dict, optional): Config of dimensions loss. Default: dict(type='L1Loss', loss_weight=0.1).
- **loss_offsets2d** –(dict, optional): Config of offsets2d loss. Default: dict(type='L1Loss', loss_weight=0.1).
- **loss_direct_depth** –(dict, optional): Config of directly regression depth loss. Default: dict(type='L1Loss', loss_weight=0.1).
- **loss_keypoints_depth** –(dict, optional): Config of keypoints decoded depth loss. Default: dict(type='L1Loss', loss_weight=0.1).
- **loss_combined_depth** –(dict, optional): Config of combined depth loss. Default: dict(type='L1Loss', loss_weight=0.1).
- **loss_attr** (*dict, optional*) –Config of attribute classification loss. In MonoFlex, Default: None.
- **bbox_coder** (*dict, optional*) –Bbox coder for encoding and decoding boxes. Default: dict(type='MonoFlexCoder', code_size=7).
- **norm_cfg** (*dict, optional*) –Dictionary to construct and config norm layer. Default: norm_cfg=dict(type='GN', num_groups=32, requires_grad=True).
- **init_cfg** (*dict*) –Initialization config dict. Default: None.

decode_heatmap(*cls_score*, *reg_pred*, *input_metas*, *cam2imgs*, *topk=100*, *kernel=3*)

Transform outputs into detections raw bbox predictions.

参数

- **class_score** (*Tensor*) –Center predict heatmap, shape (B, num_classes, H, W).
- **reg_pred** (*Tensor*) –Box regression map. shape (B, channel, H , W).
- **input_metas** (*List[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.
- **cam2imgs** (*Tensor*) –Camera intrinsic matrix. shape (N, 4, 4)
- **topk** (*int, optional*) –Get top k center keypoints from heatmap. Default 100.
- **kernel** (*int, optional*) –Max pooling kernel for extract local maximum pixels. Default 3.

返回

Decoded output of SMOKEHead, containing

the following Tensors:

- **batch_bboxes (Tensor): Coords of each 3D box.** shape (B, k, 7)
- **batch_scores (Tensor): Scores of each 3D box.** shape (B, k)
- **batch_topk_labels (Tensor): Categories of each 3D box.** shape (B, k)

返回类型 tuple[torch.Tensor]

forward(*feats*, *input_metas*)

Forward features from the upstream network.

参数

- **feats** (*list[Tensor]*) –Features from the upstream network, each is a 4D-tensor.
- **input_metas** (*list[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.

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 * bbox_code_size.

返回类型 tuple

forward_single(*x*, *input_metas*)

Forward features of a single scale level.

参数

- **x** (*Tensor*) –Feature maps from a specific FPN feature level.

- **input_metas** (*list[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.

返回 Scores for each class, bbox predictions.

返回类型 tuple

forward_train(*x*, *input_metas*, *gt_bboxes*, *gt_labels*, *gt_bboxes_3d*, *gt_labels_3d*, *centers2d*, *depths*, *attr_labels*, *gt_bboxes_ignore*, *proposal_cfg*, ***kwargs*)

参数

- **x** (*list[Tensor]*) –Features from FPN.
- **input_metas** (*list[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes** (*list[Tensor]*) –Ground truth bboxes of the image, shape (num_gts, 4).
- **gt_labels** (*list[Tensor]*) –Ground truth labels of each box, shape (num_gts,).
- **gt_bboxes_3d** (*list[Tensor]*) –3D ground truth bboxes of the image, shape (num_gts, self.bbox_code_size).
- **gt_labels_3d** (*list[Tensor]*) –3D ground truth labels of each box, shape (num_gts,).
- **centers2d** (*list[Tensor]*) –Projected 3D center of each box, shape (num_gts, 2).
- **depths** (*list[Tensor]*) –Depth of projected 3D center of each box, shape (num_gts,).
- **attr_labels** (*list[Tensor]*) –Attribute labels of each box, shape (num_gts,).
- **gt_bboxes_ignore** (*list[Tensor]*) –Ground truth bboxes to be ignored, shape (num_ignored_gts, 4).
- **proposal_cfg** (*mmcv.Config*) –Test / postprocessing configuration, if None, test_cfg would be used

返回 losses: (dict[str, Tensor]): A dictionary of loss components. proposal_list (list[Tensor]): Proposals of each image.

返回类型 tuple

get_bboxes(*cls_scores*, *bbox_preds*, *input_metas*)

Generate bboxes from bbox head predictions.

参数

- **cls_scores** (*list[Tensor]*) –Box scores for each scale level.

- **bbox_preds** (*list[Tensor]*) –Box regression for each scale.
- **input_metas** (*list[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.
- **rescale** (*bool*) –If True, return boxes in original image space.

返回 Each item in result_list is 4-tuple.

返回类型 list[tuple[CameraInstance3DBoxes, Tensor, Tensor, None]]

get_predictions(*pred_reg*, *labels3d*, *centers2d*, *reg_mask*, *batch_indices*, *input_metas*, *downsample_ratio*)

Prepare predictions for computing loss.

参数

- **pred_reg** (*Tensor*) –Box regression map. shape (B, channel, H , W).
- **labels3d** (*Tensor*) –Labels of each 3D box. shape (B * max_objs,)
- **centers2d** (*Tensor*) –Coords of each projected 3D box center on image. shape (N, 2)
- **reg_mask** (*Tensor*) –Indexes of the existence of the 3D box. shape (B * max_objs,)
- **batch_indices** (*Tenosr*) –Batch indices of the 3D box. shape (N, 3)
- **input_metas** (*list[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.
- **downsample_ratio** (*int*) –The stride of feature map.

返回 The predictions for computing loss.

返回类型 dict

get_targets(*gt_bboxes_list*, *gt_labels_list*, *gt_bboxes_3d_list*, *gt_labels_3d_list*, *centers2d_list*, *depths_list*, *feat_shape*, *img_shape*, *input_metas*)

Get training targets for batch images. “

Args:

gt_bboxes_list (list[Tensor]): Ground truth bboxes of each image, shape (num_gt, 4).

gt_labels_list (list[Tensor]): Ground truth labels of each box, shape (num_gt,).

gt_bboxes_3d_list (list[CameraInstance3DBoxes]): 3D Ground truth bboxes of each image, shape (num_gt, bbox_code_size).

gt_labels_3d_list (list[Tensor]): 3D Ground truth labels of each box, shape (num_gt,).

centers2d_list (list[Tensor]): Projected 3D centers onto 2D image, shape (num_gt, 2).

depths_list (list[Tensor]): Depth of projected 3D centers onto 2D image, each has shape (num_gt, 1).

feat_shape (tuple[int]): Feature map shape with value, shape (B, _, H, W).

img_shape (tuple[int]): Image shape in [h, w] format. input_metas (list[dict]): Meta information of each image, e.g.,

image size, scaling factor, etc.

Returns:

tuple[Tensor, dict]: The Tensor value is the targets of

center heatmap, the dict has components below:

- **base_centers2d_target (Tensor): Coords of each projected 3D box** center on image. shape (B * max_objs, 2), [dtype: int]
- **labels3d (Tensor): Labels of each 3D box.** shape (N,)
- **reg_mask (Tensor): Mask of the existence of the 3D box.** shape (B * max_objs,)
- **batch_indices (Tensor): Batch id of the 3D box.** shape (N,)
- **depth_target (Tensor): Depth target of each 3D box.** shape (N,)
- **keypoints2d_target (Tensor): Keypoints of each projected 3D box** on image. shape (N, 10, 2)
- **keypoints_mask (Tensor): Keypoints mask of each projected 3D** box on image. shape (N, 10)
- **keypoints_depth_mask (Tensor): Depths decoded from keypoints** of each 3D box. shape (N, 3)
- **orientations_target (Tensor): Orientation (encoded local yaw)** target of each 3D box. shape (N,)
- **offsets2d_target (Tensor): Offsets target of each projected** 3D box. shape (N, 2)
- **dimensions_target (Tensor): Dimensions target of each 3D box.** shape (N, 3)
- downsample_ratio (int): The stride of feature map.

init_weights()

Initialize weights.

loss(*cls_scores*, *bbox_preds*, *gt_bboxes*, *gt_labels*, *gt_bboxes_3d*, *gt_labels_3d*, *centers2d*, *depths*, *attr_labels*, *input_metas*, *gt_bboxes_ignore=None*)

Compute loss of the head.

参数

- **cls_scores** (*list[Tensor]*) –Box scores for each scale level. shape (num_gt, 4).
- **bbox_preds** (*list[Tensor]*) –Box dims is a 4D-tensor, the channel number is bbox_code_size. shape (B, 7, H, W).
- **gt_bboxes** (*list[Tensor]*) –Ground truth bboxes for each image. shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) –Class indices corresponding to each box. shape (num_gts,).
- **gt_bboxes_3d** (list[CameraInstance3DBoxes]) –3D boxes ground truth. it is the flipped gt_bboxes
- **gt_labels_3d** (*list[Tensor]*) –Same as gt_labels.
- **centers2d** (*list[Tensor]*) –2D centers on the image. shape (num_gts, 2).
- **depths** (*list[Tensor]*) –Depth ground truth. shape (num_gts,).
- **attr_labels** (*list[Tensor]*) –Attributes indices of each box. In kitti it' s None.
- **input_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]

class mmdet3d.models.dense_heads.**PGDHead**(*use_depth_classifier=True*, *use_onlyreg_proj=False*, *weight_dim=- 1*, *weight_branch=((256))*, *depth_branch=(64)*, *depth_range=(0, 70)*, *depth_unit=10*, *division='uniform'*, *depth_bins=8*, *loss_depth={'beta': 0.1111111111111111, 'loss_weight': 1.0, 'type': 'SmoothL1Loss'}*, *loss_bbox2d={'beta': 0.1111111111111111, 'loss_weight': 1.0, 'type': 'SmoothL1Loss'}*, *loss_consistency={'loss_weight': 1.0, 'type': 'GIoULoss'}*, *pred_bbox2d=True*, *pred_keypoints=False*, *bbox_coder={'base_depths': ((28.01, 16.32)), 'base_dims': ((0.8, 1.73, 0.6), (1.76, 1.73, 0.6), (3.9, 1.56, 1.6)), 'code_size': 7, 'type': 'PGDBBoxCoder'}*, ***kwargs*)

Anchor-free head used in PGD.

参数

- **use_depth_classifer** (*bool, optional*) –Whether to use depth classifier. Defaults to True.
- **use_only_reg_proj** (*bool, optional*) –Whether to use only direct regressed depth in the re-projection (to make the network easier to learn). Defaults to False.
- **weight_dim** (*int, optional*) –Dimension of the location-aware weight map. Defaults to -1.
- **weight_branch** (*tuple[tuple[int]], optional*) –Feature map channels of the convolutional branch for weight map. Defaults to ((256,),).
- **depth_branch** (*tuple[int], optional*) –Feature map channels of the branch for probabilistic depth estimation. Defaults to (64,),
- **depth_range** (*tuple[float], optional*) –Range of depth estimation. Defaults to (0, 70),
- **depth_unit** (*int, optional*) –Unit of depth range division. Defaults to 10.
- **division** (*str, optional*) –Depth division method. Options include ‘uniform’, ‘linear’, ‘log’, ‘loguniform’. Defaults to ‘uniform’.
- **depth_bins** (*int, optional*) –Discrete bins of depth division. Defaults to 8.
- **loss_depth** (*dict, optional*) –Depth loss. Defaults to dict(type=’SmoothL1Loss’, beta=1.0 / 9.0, loss_weight=1.0).
- **loss_bbox2d** (*dict, optional*) –Loss for 2D box estimation. Defaults to dict(type=’SmoothL1Loss’, beta=1.0 / 9.0, loss_weight=1.0).
- **loss_consistency** (*dict, optional*) –Consistency loss. Defaults to dict(type=’GIoULoss’, loss_weight=1.0),
- **pred_velo** (*bool, optional*) –Whether to predict velocity. Defaults to False.
- **pred_bbox2d** (*bool, optional*) –Whether to predict 2D bounding boxes. Defaults to True.
- **pred_keypoints** (*bool, optional*) –Whether to predict keypoints. Defaults to False,
- **bbox_coder** (*dict, optional*) –Bounding box coder. Defaults to dict(type=’PGDBBoxCoder’, base_depths=((28.01, 16.32),), base_dims=((0.8, 1.73, 0.6), (1.76, 1.73, 0.6), (3.9, 1.56, 1.6)), code_size=7).

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

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 * bbox_code_size.

dir_cls_preds (list[Tensor]): Box scores for direction class predictions on each scale level, each is a 4D-tensor, the channel number is num_points * 2. (bin = 2).

weight (list[Tensor]): Location-aware weight maps on each scale level, each is a 4D-tensor, the channel number is num_points * 1.

depth_cls_preds (list[Tensor]): Box scores for depth class predictions on each scale level, each is a 4D-tensor, the channel number is num_points * self.num_depth_cls.

attr_preds (list[Tensor]): Attribute scores for each scale level, each is a 4D-tensor, the channel number is num_points * num_attrs.

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 and direction class predictions, depth class predictions, location-aware weights, attribute and centerness predictions of input feature maps.

返回类型 tuple

get_bboxes(*cls_scores*, *bbox_preds*, *dir_cls_preds*, *depth_cls_preds*, *weights*, *attr_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)
- **dir_cls_preds** (*list[Tensor]*) –Box scores for direction class predictions on each scale level, each is a 4D-tensor, the channel number is num_points * 2. (bin = 2)
- **depth_cls_preds** (*list[Tensor]*) –Box scores for direction class predictions on each scale level, each is a 4D-tensor, the channel number is num_points * self.num_depth_cls.
- **weights** (*list[Tensor]*) –Location-aware weights for each scale level, each is a 4D-tensor, the channel number is num_points * self.weight_dim.
- **attr_preds** (*list[Tensor]*) –Attribute scores for each scale level Has shape (N, num_points * num_attrs, 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, optional*) –Test / postprocessing configuration, if None, test_cfg would be used. Defaults to None.
- **rescale** (*bool, optional*) –If True, return boxes in original image space. Defaults to None.

返回

Each item in result_list is a tuple, which consists of predicted 3D boxes, scores, labels, attributes and 2D boxes (if necessary).

返回类型 list[tuple[Tensor]]

get_pos_predictions(*bbox_preds*, *dir_cls_preds*, *depth_cls_preds*, *weights*, *attr_preds*, *centernesses*, *pos_inds*, *img_metas*)

Flatten predictions and get positive ones.

参数

- **bbox_preds** (*list[Tensor]*) –Box energies / deltas for each scale level, each is a 4D-tensor, the channel number is num_points * bbox_code_size.
- **dir_cls_preds** (*list[Tensor]*) –Box scores for direction class predictions on each scale level, each is a 4D-tensor, the channel number is num_points * 2. (bin = 2)
- **depth_cls_preds** (*list[Tensor]*) –Box scores for direction class predictions on each scale level, each is a 4D-tensor, the channel number is num_points * self.num_depth_cls.

- **attr_preds** (*list[Tensor]*) –Attribute scores for each scale level, each is a 4D-tensor, the channel number is num_points * num_attrs.
- **centernesses** (*list[Tensor]*) –Centerness for each scale level, each is a 4D-tensor, the channel number is num_points * 1.
- **pos_inds** (*Tensor*) –Index of foreground points from flattened tensors.
- **img_metas** (*list[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.

返回

Box predictions, direction classes, probabilistic depth maps, location-aware weight maps, attributes and centerness predictions.

返回类型 tuple[Tensor]

get_proj_bbox2d(*bbox_preds*, *pos_dir_cls_preds*, *labels_3d*, *bbox_targets_3d*, *pos_points*, *pos_inds*, *img_metas*, *pos_depth_cls_preds=None*, *pos_weights=None*, *pos_cls_scores=None*, *with_kpts=False*)

Decode box predictions and get projected 2D attributes.

参数

- **bbox_preds** (*list[Tensor]*) –Box predictions for each scale level, each is a 4D-tensor, the channel number is num_points * bbox_code_size.
- **pos_dir_cls_preds** (*Tensor*) –Box scores for direction class predictions of positive boxes on all the scale levels in shape (num_pos_points, 2).
- **labels_3d** (*list[Tensor]*) –3D box category labels for each scale level, each is a 4D-tensor.
- **bbox_targets_3d** (*list[Tensor]*) –3D box targets for each scale level, each is a 4D-tensor, the channel number is num_points * bbox_code_size.
- **pos_points** (*Tensor*) –Foreground points.
- **pos_inds** (*Tensor*) –Index of foreground points from flattened tensors.
- **img_metas** (*list[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.
- **pos_depth_cls_preds** (*Tensor, optional*) –Probabilistic depth map of positive boxes on all the scale levels in shape (num_pos_points, self.num_depth_cls). Defaults to None.
- **pos_weights** (*Tensor, optional*) –Location-aware weights of positive boxes in shape (num_pos_points, self.weight_dim). Defaults to None.
- **pos_cls_scores** (*Tensor, optional*) –Classification scores of positive boxes in shape (num_pos_points, self.num_classes). Defaults to None.

- **with_kpts** (*bool, optional*) –Whether to output keypoints targets. Defaults to False.

返回

Exterior 2D boxes from projected 3D boxes, predicted 2D boxes and keypoint targets (if necessary).

返回类型 tuple[Tensor]

get_targets(*points*, *gt_bboxes_list*, *gt_labels_list*, *gt_bboxes_3d_list*, *gt_labels_3d_list*, *centers2d_list*, *depths_list*, *attr_labels_list*)

Compute regression, classification and centerss 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,).
- **gt_bboxes_3d_list** (*list[Tensor]*) –3D Ground truth bboxes of each image, each has shape (num_gt, bbox_code_size).
- **gt_labels_3d_list** (*list[Tensor]*) –3D Ground truth labels of each box, each has shape (num_gt,).
- **centers2d_list** (*list[Tensor]*) –Projected 3D centers onto 2D image, each has shape (num_gt, 2).
- **depths_list** (*list[Tensor]*) –Depth of projected 3D centers onto 2D image, each has shape (num_gt, 1).
- **attr_labels_list** (*list[Tensor]*) –Attribute 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.

返回类型 tuple

init_weights()

Initialize weights of the head.

We currently still use the customized defined init_weights because the default init of DCN triggered by the init_cfg will init conv_offset.weight, which mistakenly affects the training stability.

loss(*cls_scores*, *bbox_preds*, *dir_cls_preds*, *depth_cls_preds*, *weights*, *attr_preds*, *centernesses*, *gt_bboxes*, *gt_labels*, *gt_bboxes_3d*, *gt_labels_3d*, *centers2d*, *depths*, *attr_labels*, *img_metas*, *gt_bboxes_ignore=None*)

Compute loss of the head.

参数

- **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 * bbox_code_size.
- **dir_cls_preds** (*list[Tensor]*) –Box scores for direction class predictions on each scale level, each is a 4D-tensor, the channel number is num_points * 2. (bin = 2)
- **depth_cls_preds** (*list[Tensor]*) –Box scores for direction class predictions on each scale level, each is a 4D-tensor, the channel number is num_points * self.num_depth_cls.
- **weights** (*list[Tensor]*) –Location-aware weights for each scale level, each is a 4D-tensor, the channel number is num_points * self.weight_dim.
- **attr_preds** (*list[Tensor]*) –Attribute scores for each scale level, each is a 4D-tensor, the channel number is num_points * num_attrs.
- **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
- **gt_bboxes_3d** (*list[Tensor]*) –3D boxes ground truth with shape of (num_gts, code_size).
- **gt_labels_3d** (*list[Tensor]*) –same as gt_labels
- **centers2d** (*list[Tensor]*) –2D centers on the image with shape of (num_gts, 2).
- **depths** (*list[Tensor]*) –Depth ground truth with shape of (num_gts,).
- **attr_labels** (*list[Tensor]*) –Attributes indices of each box.
- **img_metas** (*list[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.
- **gt_bboxes_ignore** (*list[Tensor]*) –specify which bounding boxes can be ignored when computing the loss. Defaults to None.

返回 A dictionary of loss components.

返回类型 dict[str, Tensor]

class mmdet3d.models.dense_heads.**PartA2RPNHead**(*num_classes*, *in_channels*, *train_cfg*, *test_cfg*, *feat_channels=256*, *use_direction_classifier=True*, *anchor_generator={'custom_values': [], 'range': [0, - 39.68, - 1.78, 69.12, 39.68, - 1.78], 'reshape_out': False, 'rotations': [0, 1.57], 'sizes': [[3.9, 1.6, 1.56]], 'strides': [2], 'type': 'Anchor3DRangeGenerator'}*, *assigner_per_size=False*, *assign_per_class=False*, *diff_rad_by_sin=True*, *dir_offset=- 1.5707963267948966*, *dir_limit_offset=0*, *bbox_coder={'type': 'DeltaXYZWLHRBBoxCoder'}*, *loss_cls={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, *loss_bbox={'beta': 0.1111111111111111, 'loss_weight': 2.0, 'type': 'SmoothL1Loss'}*, *loss_dir={'loss_weight': 0.2, 'type': 'CrossEntropyLoss'}*, *init_cfg=None*)

RPN head for PartA2.

注解: The main difference between the PartA2 RPN head and the Anchor3DHead lies in their output during inference. PartA2 RPN head further returns the original classification score for the second stage since the bbox head in RoI head does not do classification task.

Different from RPN heads in 2D detectors, this RPN head does multi-class classification task and uses FocalLoss like the SECOND and PointPillars do. But this head uses class agnostic nms rather than multi-class nms.

参数

- **num_classes** (*int*) –Number of classes.
- **in_channels** (*int*) –Number of channels in the input feature map.
- **train_cfg** (*dict*) –Train configs.
- **test_cfg** (*dict*) –Test configs.
- **feat_channels** (*int*) –Number of channels of the feature map.
- **use_direction_classifier** (*bool*) –Whether to add a direction classifier.
- **anchor_generator** (*dict*) –Config dict of anchor generator.

- **assigner_per_size** (*bool*) –Whether to do assignment for each separate anchor size.
- **assign_per_class** (*bool*) –Whether to do assignment for each class.
- **diff_rad_by_sin** (*bool*) –Whether to change the difference into sin difference for box regression loss.
- **dir_offset** (*float | int*) –The offset of BEV rotation angles (TODO: may be moved into box coder)
- **dir_limit_offset** (*float | int*) –The limited range of BEV rotation angles. (TODO: may be moved into box coder)
- **bbox_coder** (*dict*) –Config dict of box coders.
- **loss_cls** (*dict*) –Config of classification loss.
- **loss_bbox** (*dict*) –Config of localization loss.
- **loss_dir** (*dict*) –Config of direction classifier loss.

class_agnostic_nms(*mlvl_bboxes*, *mlvl_bboxes_for_nms*, *mlvl_max_scores*, *mlvl_label_pred*, *mlvl_cls_score*, *mlvl_dir_scores*, *score_thr*, *max_num*, *cfg*, *input_meta*)

Class agnostic nms for single batch.

参数

- **mlvl_bboxes** (*torch.Tensor*) –Bboxes from Multi-level.
- **mlvl_bboxes_for_nms** (*torch.Tensor*) –Bboxes for nms (bev or minmax boxes) from Multi-level.
- **mlvl_max_scores** (*torch.Tensor*) –Max scores of Multi-level bbox.
- **mlvl_label_pred** (*torch.Tensor*) –Class predictions of Multi-level bbox.
- **mlvl_cls_score** (*torch.Tensor*) –Class scores of Multi-level bbox.
- **mlvl_dir_scores** (*torch.Tensor*) –Direction scores of Multi-level bbox.
- **score_thr** (*int*) –Score threshold.
- **max_num** (*int*) –Max number of bboxes after nms.
- **cfg** (ConfigDict) –Training or testing config.
- **input_meta** (*dict*) –Contain pcd and img’ s meta info.

返回

Predictions of single batch. Contain the keys:

- boxes_3d (BaseInstance3DBoxes): Predicted 3d bboxes.
- scores_3d (torch.Tensor): Score of each bbox.

- labels_3d (torch.Tensor): Label of each bbox.
- cls_preds (torch.Tensor): Class score of each bbox.

返回类型 dict

get_bboxes_single(*cls_scores*, *bbox_preds*, *dir_cls_preds*, *mlvl_anchors*, *input_meta*, *cfg*, *rescale=False*)

Get bboxes of single branch.

参数

- **cls_scores** (*torch.Tensor*) –Class score in single batch.
- **bbox_preds** (*torch.Tensor*) –Bbox prediction in single batch.
- **dir_cls_preds** (*torch.Tensor*) –Predictions of direction class in single batch.
- **mlvl_anchors** (*List[torch.Tensor]*) –Multi-level anchors in single batch.
- **input_meta** (*list[dict]*) –Contain pcd and img's meta info.
- **cfg** (ConfigDict) –Training or testing config.
- **rescale** (*list[torch.Tensor]*) –whether th rescale bbox.

返回

Predictions of single batch containing the following keys:

- boxes_3d (BaseInstance3DBoxes): Predicted 3d bboxes.
- scores_3d (torch.Tensor): Score of each bbox.
- labels_3d (torch.Tensor): Label of each bbox.
- cls_preds (torch.Tensor): Class score of each bbox.

返回类型 dict

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

Calculate losses.

参数

- **cls_scores** (*list[torch.Tensor]*) –Multi-level class scores.
- **bbox_preds** (*list[torch.Tensor]*) –Multi-level bbox predictions.
- **dir_cls_preds** (*list[torch.Tensor]*) –Multi-level direction class predictions.
- **gt_bboxes** (list[BaseInstance3DBoxes]) –Ground truth boxes of each sample.
- **gt_labels** (*list[torch.Tensor]*) –Labels of each sample.
- **input_metas** (*list[dict]*) –Point cloud and image's meta info.
- **gt_bboxes_ignore** (*list[torch.Tensor]*) –Specify which bounding.

返回

Classification, bbox, and direction losses of each level.

- loss_rpn_cls (list[torch.Tensor]): Classification losses.
- loss_rpn_bbox (list[torch.Tensor]): Box regression losses.
- **loss_rpn_dir (list[torch.Tensor]): Direction classification** losses.

返回类型 dict[str, list[torch.Tensor]]

class mmdet3d.models.dense_heads.**PointRPNHead**(*num_classes*, *train_cfg*, *test_cfg*, *pred_layer_cfg=None*, *enlarge_width=0.1*, *cls_loss=None*, *bbox_loss=None*, *bbox_coder=None*, *init_cfg=None*)

RPN module for PointRCNN.

参数

- **num_classes** (*int*) –Number of classes.
- **train_cfg** (*dict*) –Train configs.
- **test_cfg** (*dict*) –Test configs.
- **pred_layer_cfg** (*dict, optional*) –Config of classification and regression prediction layers. Defaults to None.
- **enlarge_width** (*float, optional*) –Enlarge bbox for each side to ignore close points. Defaults to 0.1.
- **cls_loss** (*dict, optional*) –Config of direction classification loss. Defaults to None.
- **bbox_loss** (*dict, optional*) –Config of localization loss. Defaults to None.
- **bbox_coder** (*dict, optional*) –Config dict of box coders. Defaults to None.
- **init_cfg** (*dict, optional*) –Config of initialization. Defaults to None.

class_agnostic_nms(*obj_scores*, *sem_scores*, *bbox*, *points*, *input_meta*)

Class agnostic nms.

参数

- **obj_scores** (*torch.Tensor*) –Objectness score of bounding boxes.
- **sem_scores** (*torch.Tensor*) –Semantic class score of bounding boxes.
- **bbox** (*torch.Tensor*) –Predicted bounding boxes.

返回 Bounding boxes, scores and labels.

返回类型 tuple[torch.Tensor]

forward(*feat_dict*)

Forward pass.

参数 **feat_dict** (*dict*) –Feature dict from backbone.

返回

Predicted boxes and classification scores.

返回类型 tuple[list[torch.Tensor]]

get_bboxes(*points*, *bbox_preds*, *cls_preds*, *input_metas*, *rescale=False*)

Generate bboxes from RPN head predictions.

参数

- **points** (*torch.Tensor*) –Input points.
- **bbox_preds** (*dict*) –Regression predictions from PointRCNN head.
- **cls_preds** (*dict*) –Class scores predictions from PointRCNN head.
- **input_metas** (*list[dict]*) –Point cloud and image’s meta info.
- **rescale** (*bool, optional*) –Whether to rescale bboxes. Defaults to False.

返回 Bounding boxes, scores and labels.

返回类型 list[tuple[torch.Tensor]]

get_targets(*points*, *gt_bboxes_3d*, *gt_labels_3d*)

Generate targets of PointRCNN RPN head.

参数

- **points** (*list[torch.Tensor]*) –Points of each batch.
- **gt_bboxes_3d** (list[BaseInstance3DBoxes]) –Ground truth bboxes of each batch.
- **gt_labels_3d** (*list[torch.Tensor]*) –Labels of each batch.

返回 Targets of PointRCNN RPN head.

返回类型 tuple[torch.Tensor]

get_targets_single(*points*, *gt_bboxes_3d*, *gt_labels_3d*)

Generate targets of PointRCNN RPN head for single batch.

参数

- **points** (*torch.Tensor*) –Points of each batch.
- **gt_bboxes_3d** (BaseInstance3DBoxes) –Ground truth boxes of each batch.
- **gt_labels_3d** (*torch.Tensor*) –Labels of each batch.

返回 Targets of ssd3d head.

返回类型 tuple[torch.Tensor]

loss(*bbox_preds*, *cls_preds*, *points*, *gt_bboxes_3d*, *gt_labels_3d*, *img_metas=None*)

Compute loss.

参数

- **bbox_preds** (*dict*) –Predictions from forward of PointRCNN RPN_Head.
- **cls_preds** (*dict*) –Classification from forward of PointRCNN RPN_Head.
- **points** (*list[torch.Tensor]*) –Input points.
- **gt_bboxes_3d** (list[BaseInstance3DBoxes]) –Ground truth bboxes of each sample.
- **gt_labels_3d** (*list[torch.Tensor]*) –Labels of each sample.
- **img_metas** (*list[dict], Optional*) –Contain pcd and img' s meta info. Defaults to None.

返回 Losses of PointRCNN RPN module.

返回类型 dict

class mmdet3d.models.dense_heads.**SMOKEMono3DHead**(*num_classes*, *in_channels*, *dim_channel*, *ori_channel*, *bbox_coder*, *loss_cls={'loss_weight': 1.0, 'type': 'GaussionFocalLoss'}*, *loss_bbox={'loss_weight': 0.1, 'type': 'L1Loss'}*, *loss_dir=None*, *loss_attr=None*, *norm_cfg={'num_groups': 32, 'requires_grad': True, 'type': 'GN'}*, *init_cfg=None*, ***kwargs*)

Anchor-free head used in SMOKE

```
        /-----> 3*3 conv -----> 1*1 conv -----> cls
feature
        \-----> 3*3 conv -----> 1*1 conv -----> reg
```

参数

- **num_classes** (*int*) –Number of categories excluding the background category.
- **in_channels** (*int*) –Number of channels in the input feature map.
- **dim_channel** (*list[int]*) –indices of dimension offset preds in regression heatmap channels.

- **ori_channel** (*list[int]*) –indices of orientation offset pred in regression heatmap channels.
- **bbox_coder** (CameraInstance3DBoxes) –Bbox coder for encoding and decoding boxes.
- **loss_cls** (*dict, optional*) –Config of classification loss. Default: loss_cls=dict(type='GaussionFocalLoss', loss_weight=1.0).
- **loss_bbox** (*dict, optional*) –Config of localization loss. Default: loss_bbox=dict(type='L1Loss', loss_weight=10.0).
- **loss_dir** (*dict, optional*) –Config of direction classification loss. In SMOKE, Default: None.
- **loss_attr** (*dict, optional*) –Config of attribute classification loss. In SMOKE, Default: None.
- **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*) –Initialization config dict. Default: None.

decode_heatmap(*cls_score*, *reg_pred*, *img_metas*, *cam2imgs*, *trans_mats*, *topk=100*, *kernel=3*)

Transform outputs into detections raw bbox predictions.

参数

- **class_score** (*Tensor*) –Center predict heatmap, shape (B, num_classes, H, W).
- **reg_pred** (*Tensor*) –Box regression map. shape (B, channel, H , W).
- **img_metas** (*List[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.
- **cam2imgs** (*Tensor*) –Camera intrinsic matrixs. shape (B, 4, 4)
- **trans_mats** (*Tensor*) –Transformation matrix from original image to feature map. shape: (batch, 3, 3)
- **topk** (*int*) –Get top k center keypoints from heatmap. Default 100.
- **kernel** (*int*) –Max pooling kernel for extract local maximum pixels. Default 3.

返回

Decoded output of SMOKEHead, containing

the following Tensors:

- **batch_bboxes (Tensor): Coords of each 3D box.** shape (B, k, 7)

- **batch_scores (Tensor): Scores of each 3D box.** shape (B, k)
- **batch_topk_labels (Tensor): Categories of each 3D box.** shape (B, k)

返回类型 tuple[torch.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

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 * bbox_code_size.

返回类型 tuple

forward_single(*x*)

Forward features of a single scale level.

参数 **x** (*Tensor*) –Input feature map.

返回 Scores for each class, bbox of input feature maps.

返回类型 tuple

get_bboxes(*cls_scores*, *bbox_preds*, *img_metas*, *rescale=None*)

Generate bboxes from bbox head predictions.

参数

- **cls_scores** (*list[Tensor]*) –Box scores for each scale level.
- **bbox_preds** (*list[Tensor]*) –Box regression for each scale.
- **img_metas** (*list[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.
- **rescale** (*bool*) –If True, return boxes in original image space.

返回 Each item in result_list is 4-tuple.

返回类型 list[tuple[CameraInstance3DBoxes, Tensor, Tensor, None]]

get_predictions(*labels3d*, *centers2d*, *gt_locations*, *gt_dimensions*, *gt_orientations*, *indices*, *img_metas*, *pred_reg*)

Prepare predictions for computing loss.

参数

- **labels3d** (*Tensor*) –Labels of each 3D box. shape (B, max_objs,)

- **centers2d** (*Tensor*) –Coords of each projected 3D box center on image. shape (B * max_objs, 2)
- **gt_locations** (*Tensor*) –Coords of each 3D box' s location. shape (B * max_objs, 3)
- **gt_dimensions** (*Tensor*) –Dimensions of each 3D box. shape (N, 3)
- **gt_orientations** (*Tensor*) –Orientation(yaw) of each 3D box. shape (N, 1)
- **indices** (*Tensor*) –Indices of the existence of the 3D box. shape (B * max_objs,)
- **img_metas** (*list[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.
- **pre_reg** (*Tensor*) –Box regression map. shape (B, channel, H , W).

返回

the dict has components below: - bbox3d_yaws (`CameraInstance3DBoxes`):

bbox calculated using pred orientations.

- **bbox3d_dims (`CameraInstance3DBoxes`):** bbox calculated using pred dimensions.
- **bbox3d_locs (`CameraInstance3DBoxes`):** bbox calculated using pred locations.

返回类型 dict

get_targets(*gt_bboxes*, *gt_labels*, *gt_bboxes_3d*, *gt_labels_3d*, *centers2d*, *feat_shape*, *img_shape*, *img_metas*)

Get training targets for batch images.

参数

- **gt_bboxes** (*list[Tensor]*) –Ground truth bboxes of each image, shape (num_gt, 4).
- **gt_labels** (*list[Tensor]*) –Ground truth labels of each box, shape (num_gt,).
- **gt_bboxes_3d** (list[`CameraInstance3DBoxes`]) –3D Ground truth bboxes of each image, shape (num_gt, bbox_code_size).
- **gt_labels_3d** (*list[Tensor]*) –3D Ground truth labels of each box, shape (num_gt,).
- **centers2d** (*list[Tensor]*) –Projected 3D centers onto 2D image, shape (num_gt, 2).
- **feat_shape** (*tuple[int]*) –Feature map shape with value, shape (B, _, H, W).
- **img_shape** (*tuple[int]*) –Image shape in [h, w] format.

- **img_metas** (*list[dict]*) –Meta information of each image, e.g., image size, scaling factor, etc.

返回

The Tensor value is the targets of

center heatmap, the dict has components below:

- **gt_centers2d (Tensor): Coords of each projected 3D box** center on image. shape (B * max_objs, 2)
- **gt_labels3d (Tensor): Labels of each 3D box.** shape (B, max_objs,)
- **indices (Tensor): Indices of the existence of the 3D box.** shape (B * max_objs,)
- **affine_indices (Tensor): Indices of the affine of the 3D box.** shape (N,)
- **gt_locs (Tensor): Coords of each 3D box's location.** shape (N, 3)
- **gt_dims (Tensor): Dimensions of each 3D box.** shape (N, 3)
- **gt_yaws (Tensor): Orientation(yaw) of each 3D box.** shape (N, 1)
- **gt_cors (Tensor): Coords of the corners of each 3D box.** shape (N, 8, 3)

返回类型 tuple[Tensor, dict]

loss(*cls_scores*, *bbox_preds*, *gt_bboxes*, *gt_labels*, *gt_bboxes_3d*, *gt_labels_3d*, *centers2d*, *depths*, *attr_labels*, *img_metas*, *gt_bboxes_ignore=None*)

Compute loss of the head.

参数

- **cls_scores** (*list[Tensor]*) –Box scores for each scale level. shape (num_gt, 4).
- **bbox_preds** (*list[Tensor]*) –Box dims is a 4D-tensor, the channel number is bbox_code_size. shape (B, 7, H, W).
- **gt_bboxes** (*list[Tensor]*) –Ground truth bboxes for each image. shape (num_gts, 4) in [tl_x, tl_y, br_x, br_y] format.
- **gt_labels** (*list[Tensor]*) –Class indices corresponding to each box. shape (num_gts,).
- **gt_bboxes_3d** (list[CameraInstance3DBoxes]) –3D boxes ground truth. it is the flipped gt_bboxes
- **gt_labels_3d** (*list[Tensor]*) –Same as gt_labels.
- **centers2d** (*list[Tensor]*) –2D centers on the image. shape (num_gts, 2).
- **depths** (*list[Tensor]*) –Depth ground truth. shape (num_gts,).

- **attr_labels** (*list[Tensor]*) –Attributes indices of each box. In kitti it's None.
- **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]

class mmdet3d.models.dense_heads.**SSD3DHead**(*num_classes*, *bbox_coder*, *in_channels=256*, *train_cfg=None*, *test_cfg=None*, *vote_module_cfg=None*, *vote_aggregation_cfg=None*, *pred_layer_cfg=None*, *conv_cfg={'type': 'Conv1d'}*, *norm_cfg={'type': 'BN1d'}*, *act_cfg={'type': 'ReLU'}*, *objectness_loss=None*, *center_loss=None*, *dir_class_loss=None*, *dir_res_loss=None*, *size_res_loss=None*, *corner_loss=None*, *vote_loss=None*, *init_cfg=None*)

Bbox head of 3DSSD.

参数

- **num_classes** (*int*) –The number of class.
- **bbox_coder** (BaseBBoxCoder) –Bbox coder for encoding and decoding boxes.
- **in_channels** (*int*) –The number of input feature channel.
- **train_cfg** (*dict*) –Config for training.
- **test_cfg** (*dict*) –Config for testing.
- **vote_module_cfg** (*dict*) –Config of VoteModule for point-wise votes.
- **vote_aggregation_cfg** (*dict*) –Config of vote aggregation layer.
- **pred_layer_cfg** (*dict*) –Config of classfication and regression prediction layers.
- **conv_cfg** (*dict*) –Config of convolution in prediction layer.
- **norm_cfg** (*dict*) –Config of BN in prediction layer.
- **act_cfg** (*dict*) –Config of activation in prediction layer.
- **objectness_loss** (*dict*) –Config of objectness loss.
- **center_loss** (*dict*) –Config of center loss.
- **dir_class_loss** (*dict*) –Config of direction classification loss.

- **dir_res_loss** (*dict*) –Config of direction residual regression loss.
- **size_res_loss** (*dict*) –Config of size residual regression loss.
- **corner_loss** (*dict*) –Config of bbox corners regression loss.
- **vote_loss** (*dict*) –Config of candidate points regression loss.

get_bboxes(*points*, *bbox_preds*, *input_metas*, *rescale=False*)

Generate bboxes from 3DSSD head predictions.

参数

- **points** (*torch.Tensor*) –Input points.
- **bbox_preds** (*dict*) –Predictions from sdd3d head.
- **input_metas** (*list[dict]*) –Point cloud and image's meta info.
- **rescale** (*bool*) –Whether to rescale bboxes.

返回 Bounding boxes, scores and labels.

返回类型 list[tuple[torch.Tensor]]

get_targets(*points*, *gt_bboxes_3d*, *gt_labels_3d*, *pts_semantic_mask=None*, *pts_instance_mask=None*, *bbox_preds=None*)

Generate targets of ssd3d head.

参数

- **points** (*list[torch.Tensor]*) –Points of each batch.
- **gt_bboxes_3d** (list[BaseInstance3DBoxes]) –Ground truth bboxes of each batch.
- **gt_labels_3d** (*list[torch.Tensor]*) –Labels of each batch.
- **pts_semantic_mask** (*list[torch.Tensor]*) –Point-wise semantic label of each batch.
- **pts_instance_mask** (*list[torch.Tensor]*) –Point-wise instance label of each batch.
- **bbox_preds** (*torch.Tensor*) –Bounding box predictions of ssd3d head.

返回 Targets of ssd3d head.

返回类型 tuple[torch.Tensor]

get_targets_single(*points*, *gt_bboxes_3d*, *gt_labels_3d*, *pts_semantic_mask=None*, *pts_instance_mask=None*, *aggregated_points=None*, *seed_points=None*)

Generate targets of ssd3d head for single batch.

参数

- **points** (*torch.Tensor*) –Points of each batch.
- **gt_bboxes_3d** (BaseInstance3DBoxes) –Ground truth boxes of each batch.
- **gt_labels_3d** (*torch.Tensor*) –Labels of each batch.
- **pts_semantic_mask** (*torch.Tensor*) –Point-wise semantic label of each batch.
- **pts_instance_mask** (*torch.Tensor*) –Point-wise instance label of each batch.
- **aggregated_points** (*torch.Tensor*) –Aggregated points from candidate points layer.
- **seed_points** (*torch.Tensor*) –Seed points of candidate points.

返回 Targets of ssd3d head.

返回类型 tuple[torch.Tensor]

loss(*bbox_preds*, *points*, *gt_bboxes_3d*, *gt_labels_3d*, *pts_semantic_mask=None*, *pts_instance_mask=None*, *img_metas=None*, *gt_bboxes_ignore=None*)

Compute loss.

参数

- **bbox_preds** (*dict*) –Predictions from forward of SSD3DHead.
- **points** (*list[torch.Tensor]*) –Input points.
- **gt_bboxes_3d** (list[BaseInstance3DBoxes]) –Ground truth bboxes of each sample.
- **gt_labels_3d** (*list[torch.Tensor]*) –Labels of each sample.
- **pts_semantic_mask** (*list[torch.Tensor]*) –Point-wise semantic mask.
- **pts_instance_mask** (*list[torch.Tensor]*) –Point-wise instance mask.
- **img_metas** (*list[dict]*) –Contain pcd and img’s meta info.
- **gt_bboxes_ignore** (*list[torch.Tensor]*) –Specify which bounding.

返回 Losses of 3DSSD.

返回类型 dict

multiclass_nms_single(*obj_scores*, *sem_scores*, *bbox*, *points*, *input_meta*)

Multi-class nms in single batch.

参数

- **obj_scores** (*torch.Tensor*) –Objectness score of bounding boxes.
- **sem_scores** (*torch.Tensor*) –Semantic class score of bounding boxes.
- **bbox** (*torch.Tensor*) –Predicted bounding boxes.

- **points** (*torch.Tensor*) –Input points.
- **input_meta** (*dict*) –Point cloud and image’s meta info.

返回 Bounding boxes, scores and labels.

返回类型 tuple[torch.Tensor]

class mmdet3d.models.dense_heads.**ShapeAwareHead**(*tasks*, *assign_per_class=True*, *init_cfg=None*, ***kwargs*)

Shape-aware grouping head for SSN.

参数

- **tasks** (*dict*) –Shape-aware groups of multi-class objects.
- **assign_per_class** (*bool, optional*) –Whether to do assignment for each class. Default: True.
- **kwargs** (*dict*) –Other arguments are the same as those in *Anchor3DHead*.

forward_single(*x*)

Forward function on a single-scale feature map.

参数 **x** (*torch.Tensor*) –Input features.

返回

Contain score of each class, bbox regression and direction classification predictions.

返回类型 tuple[torch.Tensor]

get_bboxes(*cls_scores*, *bbox_preds*, *dir_cls_preds*, *input_metas*, *cfg=None*, *rescale=False*)

Get bboxes of anchor head.

参数

- **cls_scores** (*list[torch.Tensor]*) –Multi-level class scores.
- **bbox_preds** (*list[torch.Tensor]*) –Multi-level bbox predictions.
- **dir_cls_preds** (*list[torch.Tensor]*) –Multi-level direction class predictions.
- **input_metas** (*list[dict]*) –Contain pcd and img’s meta info.
- **cfg** (ConfigDict, optional) –Training or testing config. Default: None.
- **rescale** (*list[torch.Tensor], optional*) –Whether to rescale bbox. Default: False.

返回 Prediction resultes of batches.

返回类型 list[tuple]

get_bboxes_single(*cls_scores*, *bbox_preds*, *dir_cls_preds*, *mlvl_anchors*, *input_meta*, *cfg=None*, *rescale=False*)

Get bboxes of single branch.

参数

- **cls_scores** (*torch.Tensor*) –Class score in single batch.
- **bbox_preds** (*torch.Tensor*) –Bbox prediction in single batch.
- **dir_cls_preds** (*torch.Tensor*) –Predictions of direction class in single batch.
- **mlvl_anchors** (*List[torch.Tensor]*) –Multi-level anchors in single batch.
- **input_meta** (*list[dict]*) –Contain pcd and img's meta info.
- **cfg** (ConfigDict) –Training or testing config.
- **rescale** (*list[torch.Tensor], optional*) –whether to rescale bbox. Default: False.

返回

Contain predictions of single batch.

- bboxes (BaseInstance3DBoxes): Predicted 3d bboxes.
- scores (torch.Tensor): Class score of each bbox.
- labels (torch.Tensor): Label of each bbox.

返回类型 tuple

init_weights()

Initialize the weights.

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

Calculate losses.

参数

- **cls_scores** (*list[torch.Tensor]*) –Multi-level class scores.
- **bbox_preds** (*list[torch.Tensor]*) –Multi-level bbox predictions.
- **dir_cls_preds** (*list[torch.Tensor]*) –Multi-level direction class predictions.
- **gt_bboxes** (list[BaseInstance3DBoxes]) –Gt bboxes of each sample.
- **gt_labels** (*list[torch.Tensor]*) –Gt labels of each sample.
- **input_metas** (*list[dict]*) –Contain pcd and img's meta info.
- **gt_bboxes_ignore** (*list[torch.Tensor]*) –Specify which bounding.

返回

Classification, bbox, and direction losses of each level.

- loss_cls (list[torch.Tensor]): Classification losses.
- loss_bbox (list[torch.Tensor]): Box regression losses.
- **loss_dir (list[torch.Tensor]): Direction classification** losses.

返回类型 dict[str, list[torch.Tensor]]

loss_single(*cls_score*, *bbox_pred*, *dir_cls_preds*, *labels*, *label_weights*, *bbox_targets*, *bbox_weights*, *dir_targets*, *dir_weights*, *num_total_samples*)

Calculate loss of Single-level results.

参数

- **cls_score** (*torch.Tensor*) –Class score in single-level.
- **bbox_pred** (*torch.Tensor*) –Bbox prediction in single-level.
- **dir_cls_preds** (*torch.Tensor*) –Predictions of direction class in single-level.
- **labels** (*torch.Tensor*) –Labels of class.
- **label_weights** (*torch.Tensor*) –Weights of class loss.
- **bbox_targets** (*torch.Tensor*) –Targets of bbox predictions.
- **bbox_weights** (*torch.Tensor*) –Weights of bbox loss.
- **dir_targets** (*torch.Tensor*) –Targets of direction predictions.
- **dir_weights** (*torch.Tensor*) –Weights of direction loss.
- **num_total_samples** (*int*) –The number of valid samples.

返回

Losses of class, bbox and direction, respectively.

返回类型 tuple[torch.Tensor]

class mmdet3d.models.dense_heads.**VoteHead**(*num_classes*, *bbox_coder*, *train_cfg=None*, *test_cfg=None*, *vote_module_cfg=None*, *vote_aggregation_cfg=None*, *pred_layer_cfg=None*, *conv_cfg={'type': 'Conv1d'}*, *norm_cfg={'type': 'BN1d'}*, *objectness_loss=None*, *center_loss=None*, *dir_class_loss=None*, *dir_res_loss=None*, *size_class_loss=None*, *size_res_loss=None*, *semantic_loss=None*, *iou_loss=None*, *init_cfg=None*)

Bbox head of Votenet.

参数

- **num_classes** (*int*) –The number of class.

- **bbox_coder** (BaseBBoxCoder) –Bbox coder for encoding and decoding boxes.
- **train_cfg** (*dict*) –Config for training.
- **test_cfg** (*dict*) –Config for testing.
- **vote_module_cfg** (*dict*) –Config of VoteModule for point-wise votes.
- **vote_aggregation_cfg** (*dict*) –Config of vote aggregation layer.
- **pred_layer_cfg** (*dict*) –Config of classfication and regression prediction layers.
- **conv_cfg** (*dict*) –Config of convolution in prediction layer.
- **norm_cfg** (*dict*) –Config of BN in prediction layer.
- **objectness_loss** (*dict*) –Config of objectness loss.
- **center_loss** (*dict*) –Config of center loss.
- **dir_class_loss** (*dict*) –Config of direction classification loss.
- **dir_res_loss** (*dict*) –Config of direction residual regression loss.
- **size_class_loss** (*dict*) –Config of size classification loss.
- **size_res_loss** (*dict*) –Config of size residual regression loss.
- **semantic_loss** (*dict*) –Config of point-wise semantic segmentation loss.

forward(*feat_dict*, *sample_mod*)

Forward pass.

注解: The forward of VoteHead is divided into 4 steps:

1. Generate vote_points from seed_points.
2. Aggregate vote_points.
3. Predict bbox and score.
4. Decode predictions.

参数

- **feat_dict** (*dict*) –Feature dict from backbone.
- **sample_mod** (*str*) –Sample mode for vote aggregation layer. valid modes are "vote" , "seed" , "random" and "spec" .

返回 Predictions of vote head.

返回类型 dict

get_bboxes(*points*, *bbox_preds*, *input_metas*, *rescale=False*, *use_nms=True*)

Generate bboxes from vote head predictions.

参数

- **points** (*torch.Tensor*) –Input points.
- **bbox_preds** (*dict*) –Predictions from vote head.
- **input_metas** (*list[dict]*) –Point cloud and image's meta info.
- **rescale** (*bool*) –Whether to rescale bboxes.
- **use_nms** (*bool*) –Whether to apply NMS, skip nms postprocessing while using vote head in rpn stage.

返回 Bounding boxes, scores and labels.

返回类型 list[tuple[torch.Tensor]]

get_targets(*points*, *gt_bboxes_3d*, *gt_labels_3d*, *pts_semantic_mask=None*, *pts_instance_mask=None*, *bbox_preds=None*)

Generate targets of vote head.

参数

- **points** (*list[torch.Tensor]*) –Points of each batch.
- **gt_bboxes_3d** (list[BaseInstance3DBoxes]) –Ground truth bboxes of each batch.
- **gt_labels_3d** (*list[torch.Tensor]*) –Labels of each batch.
- **pts_semantic_mask** (*list[torch.Tensor]*) –Point-wise semantic label of each batch.
- **pts_instance_mask** (*list[torch.Tensor]*) –Point-wise instance label of each batch.
- **bbox_preds** (*torch.Tensor*) –Bounding box predictions of vote head.

返回 Targets of vote head.

返回类型 tuple[torch.Tensor]

get_targets_single(*points*, *gt_bboxes_3d*, *gt_labels_3d*, *pts_semantic_mask=None*, *pts_instance_mask=None*, *aggregated_points=None*)

Generate targets of vote head for single batch.

参数

- **points** (*torch.Tensor*) –Points of each batch.
- **gt_bboxes_3d** (BaseInstance3DBoxes) –Ground truth boxes of each batch.
- **gt_labels_3d** (*torch.Tensor*) –Labels of each batch.

- **pts_semantic_mask** (*torch.Tensor*) –Point-wise semantic label of each batch.
- **pts_instance_mask** (*torch.Tensor*) –Point-wise instance label of each batch.
- **aggregated_points** (*torch.Tensor*) –Aggregated points from vote aggregation layer.

返回 Targets of vote head.

返回类型 tuple[torch.Tensor]

loss(*bbox_preds*, *points*, *gt_bboxes_3d*, *gt_labels_3d*, *pts_semantic_mask=None*, *pts_instance_mask=None*, *img_metas=None*, *gt_bboxes_ignore=None*, *ret_target=False*)

Compute loss.

参数

- **bbox_preds** (*dict*) –Predictions from forward of vote head.
- **points** (*list[torch.Tensor]*) –Input points.
- **gt_bboxes_3d** (list[BaseInstance3DBoxes]) –Ground truth bboxes of each sample.
- **gt_labels_3d** (*list[torch.Tensor]*) –Labels of each sample.
- **pts_semantic_mask** (*list[torch.Tensor]*) –Point-wise semantic mask.
- **pts_instance_mask** (*list[torch.Tensor]*) –Point-wise instance mask.
- **img_metas** (*list[dict]*) –Contain pcd and img's meta info.
- **gt_bboxes_ignore** (*list[torch.Tensor]*) –Specify which bounding.
- **ret_target** (*Bool*) –Return targets or not.

返回 Losses of Votenet.

返回类型 dict

multiclass_nms_single(*obj_scores*, *sem_scores*, *bbox*, *points*, *input_meta*)

Multi-class nms in single batch.

参数

- **obj_scores** (*torch.Tensor*) –Objectness score of bounding boxes.
- **sem_scores** (*torch.Tensor*) –semantic class score of bounding boxes.
- **bbox** (*torch.Tensor*) –Predicted bounding boxes.
- **points** (*torch.Tensor*) –Input points.
- **input_meta** (*dict*) –Point cloud and image's meta info.

返回 Bounding boxes, scores and labels.

返回类型 tuple[torch.Tensor]

40.5 roi_heads

class mmdet3d.models.roi_heads.**Base3DRoIHead**(*bbox_head=None*, *mask_roi_extractor=None*, *mask_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

Base class for 3d RoIHeads.

aug_test(*x*, *proposal_list*, *img_metas*, *rescale=False*, ***kwargs*)

Test with augmentations.

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

abstract forward_train(*x*, *img_metas*, *proposal_list*, *gt_bboxes*, *gt_labels*, *gt_bboxes_ignore=None*, ***kwargs*)

Forward function during training.

参数

- **x** (*dict*) –Contains features from the first stage.
- **img_metas** (*list[dict]*) –Meta info of each image.
- **proposal_list** (*list[dict]*) –Proposal information from rpn.
- **gt_bboxes** (list[BaseInstance3DBoxes]) –GT bboxes of each sample. The bboxes are encapsulated by 3D box structures.
- **gt_labels** (*list[torch.LongTensor]*) –GT labels of each sample.
- **gt_bboxes_ignore** (*list[torch.Tensor], optional*) –Ground truth boxes to be ignored.

返回 Losses from each head.

返回类型 dict[str, torch.Tensor]

abstract init_assigner_sampler()

Initialize assigner and sampler.

abstract init_bbox_head()

Initialize the box head.

abstract init_mask_head()

Initialize maek head.

simple_test(*x*, *proposal_list*, *img_metas*, *proposals=None*, *rescale=False*, ***kwargs*)

Test without augmentation.

property with_bbox

whether the RoIHead has box head

Type bool

property with_mask

whether the RoIHead has mask head

Type bool

class mmdet3d.models.roi_heads.**H3DBboxHead**(*num_classes*, *suface_matching_cfg*, *line_matching_cfg*, *bbox_coder*, *train_cfg=None*, *test_cfg=None*, *gt_per_seed=1*, *num_proposal=256*, *feat_channels=(128, 128)*, *primitive_feat_refine_streams=2*, *primitive_refine_channels=[128, 128, 128]*, *upper_thresh=100.0*, *surface_thresh=0.5*, *line_thresh=0.5*, *conv_cfg={'type': 'Conv1d'}*, *norm_cfg={'type': 'BN1d'}*, *objectness_loss=None*, *center_loss=None*, *dir_class_loss=None*, *dir_res_loss=None*, *size_class_loss=None*, *size_res_loss=None*, *semantic_loss=None*, *cues_objectness_loss=None*, *cues_semantic_loss=None*, *proposal_objectness_loss=None*, *primitive_center_loss=None*, *init_cfg=None*)

Bbox head of H3DNet.

参数

- **num_classes** (*int*) –The number of classes.
- **surface_matching_cfg** (*dict*) –Config for surface primitive matching.
- **line_matching_cfg** (*dict*) –Config for line primitive matching.
- **bbox_coder** (BaseBBoxCoder) –Bbox coder for encoding and decoding boxes.
- **train_cfg** (*dict*) –Config for training.
- **test_cfg** (*dict*) –Config for testing.
- **gt_per_seed** (*int*) –Number of ground truth votes generated from each seed point.
- **num_proposal** (*int*) –Number of proposal votes generated.
- **feat_channels** (*tuple[int]*) –Convolution channels of prediction layer.
- **primitive_feat_refine_streams** (*int*) –The number of mlps to refine primitive feature.

- **primitive_refine_channels** (*tuple[int]*) –Convolution channels of prediction layer.
- **upper_thresh** (*float*) –Threshold for line matching.
- **surface_thresh** (*float*) –Threshold for surface matching.
- **line_thresh** (*float*) –Threshold for line matching.
- **conv_cfg** (*dict*) –Config of convolution in prediction layer.
- **norm_cfg** (*dict*) –Config of BN in prediction layer.
- **objectness_loss** (*dict*) –Config of objectness loss.
- **center_loss** (*dict*) –Config of center loss.
- **dir_class_loss** (*dict*) –Config of direction classification loss.
- **dir_res_loss** (*dict*) –Config of direction residual regression loss.
- **size_class_loss** (*dict*) –Config of size classification loss.
- **size_res_loss** (*dict*) –Config of size residual regression loss.
- **semantic_loss** (*dict*) –Config of point-wise semantic segmentation loss.
- **cues_objectness_loss** (*dict*) –Config of cues objectness loss.
- **cues_semantic_loss** (*dict*) –Config of cues semantic loss.
- **proposal_objectness_loss** (*dict*) –Config of proposal objectness loss.
- **primitive_center_loss** (*dict*) –Config of primitive center regression loss.

forward(*feats_dict*, *sample_mod*)

Forward pass.

参数

- **feats_dict** (*dict*) –Feature dict from backbone.
- **sample_mod** (*str*) –Sample mode for vote aggregation layer. valid modes are “vote”, “seed” and “random”.

返回 Predictions of vote head.

返回类型 dict

get_bboxes(*points*, *bbox_preds*, *input_metas*, *rescale=False*, *suffix=''*)

Generate bboxes from vote head predictions.

参数

- **points** (*torch.Tensor*) –Input points.
- **bbox_preds** (*dict*) –Predictions from vote head.

- **input_metas** (*list[dict]*) –Point cloud and image’ s meta info.
- **rescale** (*bool*) –Whether to rescale bboxes.

返回 Bounding boxes, scores and labels.

返回类型 list[tuple[torch.Tensor]]

get_proposal_stage_loss(*bbox_preds*, *size_class_targets*, *size_res_targets*, *dir_class_targets*, *dir_res_targets*, *center_targets*, *mask_targets*, *objectness_targets*, *objectness_weights*, *box_loss_weights*, *valid_gt_weights*, *suffix=''*)

Compute loss for the aggregation module.

参数

- **bbox_preds** (*dict*) –Predictions from forward of vote head.
- **size_class_targets** (*torch.Tensor*) –Ground truth size class of each prediction bounding box.
- **size_res_targets** (*torch.Tensor*) –Ground truth size residual of each prediction bounding box.
- **dir_class_targets** (*torch.Tensor*) –Ground truth direction class of each prediction bounding box.
- **dir_res_targets** (*torch.Tensor*) –Ground truth direction residual of each prediction bounding box.
- **center_targets** (*torch.Tensor*) –Ground truth center of each prediction bounding box.
- **mask_targets** (*torch.Tensor*) –Validation of each prediction bounding box.
- **objectness_targets** (*torch.Tensor*) –Ground truth objectness label of each prediction bounding box.
- **objectness_weights** (*torch.Tensor*) –Weights of objectness loss for each prediction bounding box.
- **box_loss_weights** (*torch.Tensor*) –Weights of regression loss for each prediction bounding box.
- **valid_gt_weights** (*torch.Tensor*) –Validation of each ground truth bounding box.

返回 Losses of aggregation module.

返回类型 dict

get_targets(*points*, *gt_bboxes_3d*, *gt_labels_3d*, *pts_semantic_mask=None*, *pts_instance_mask=None*, *bbox_preds=None*)

Generate targets of proposal module.

参数

- **points** (*list[torch.Tensor]*) –Points of each batch.
- **gt_bboxes_3d** (list[BaseInstance3DBoxes]) –Ground truth bboxes of each batch.
- **gt_labels_3d** (*list[torch.Tensor]*) –Labels of each batch.
- **pts_semantic_mask** (*list[torch.Tensor]*) –Point-wise semantic label of each batch.
- **pts_instance_mask** (*list[torch.Tensor]*) –Point-wise instance label of each batch.
- **bbox_preds** (*torch.Tensor*) –Bounding box predictions of vote head.

返回 Targets of proposal module.

返回类型 tuple[torch.Tensor]

get_targets_single(*points*, *gt_bboxes_3d*, *gt_labels_3d*, *pts_semantic_mask=None*, *pts_instance_mask=None*, *aggregated_points=None*, *pred_surface_center=None*, *pred_line_center=None*, *pred_obj_surface_center=None*, *pred_obj_line_center=None*, *pred_surface_sem=None*, *pred_line_sem=None*)

Generate targets for primitive cues for single batch.

参数

- **points** (*torch.Tensor*) –Points of each batch.
- **gt_bboxes_3d** (BaseInstance3DBoxes) –Ground truth boxes of each batch.
- **gt_labels_3d** (*torch.Tensor*) –Labels of each batch.
- **pts_semantic_mask** (*torch.Tensor*) –Point-wise semantic label of each batch.
- **pts_instance_mask** (*torch.Tensor*) –Point-wise instance label of each batch.
- **aggregated_points** (*torch.Tensor*) –Aggregated points from vote aggregation layer.
- **pred_surface_center** (*torch.Tensor*) –Prediction of surface center.
- **pred_line_center** (*torch.Tensor*) –Prediction of line center.
- **pred_obj_surface_center** (*torch.Tensor*) –Objectness prediction of surface center.
- **pred_obj_line_center** (*torch.Tensor*) –Objectness prediction of line center.
- **pred_surface_sem** (*torch.Tensor*) –Semantic prediction of surface center.

- **pred_line_sem** (*torch.Tensor*) –Semantic prediction of line center.

返回 Targets for primitive cues.

返回类型 tuple[torch.Tensor]

loss(*bbox_preds*, *points*, *gt_bboxes_3d*, *gt_labels_3d*, *pts_semantic_mask=None*, *pts_instance_mask=None*, *img_metas=None*, *rpn_targets=None*, *gt_bboxes_ignore=None*)

Compute loss.

参数

- **bbox_preds** (*dict*) –Predictions from forward of h3d bbox head.
- **points** (*list[torch.Tensor]*) –Input points.
- **gt_bboxes_3d** (list[BaseInstance3DBoxes]) –Ground truth bboxes of each sample.
- **gt_labels_3d** (*list[torch.Tensor]*) –Labels of each sample.
- **pts_semantic_mask** (*list[torch.Tensor]*) –Point-wise semantic mask.
- **pts_instance_mask** (*list[torch.Tensor]*) –Point-wise instance mask.
- **img_metas** (*list[dict]*) –Contain pcd and img's meta info.
- **rpn_targets** (*Tuple*) –Targets generated by rpn head.
- **gt_bboxes_ignore** (*list[torch.Tensor]*) –Specify which bounding.

返回 Losses of H3dnet.

返回类型 dict

multiclass_nms_single(*obj_scores*, *sem_scores*, *bbox*, *points*, *input_meta*)

Multi-class nms in single batch.

参数

- **obj_scores** (*torch.Tensor*) –Objectness score of bounding boxes.
- **sem_scores** (*torch.Tensor*) –semantic class score of bounding boxes.
- **bbox** (*torch.Tensor*) –Predicted bounding boxes.
- **points** (*torch.Tensor*) –Input points.
- **input_meta** (*dict*) –Point cloud and image's meta info.

返回 Bounding boxes, scores and labels.

返回类型 tuple[torch.Tensor]

class mmdet3d.models.roi_heads.**H3DRoIHead**(*primitive_list*, *bbox_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

H3D roi head for H3DNet.

参数

- **primitive_list** (*List*) –Configs of primitive heads.
- **bbox_head** (*ConfigDict*) –Config of bbox_head.
- **train_cfg** (*ConfigDict*) –Training config.
- **test_cfg** (*ConfigDict*) –Testing config.

forward_train(*feats_dict*, *img_metas*, *points*, *gt_bboxes_3d*, *gt_labels_3d*, *pts_semantic_mask*, *pts_instance_mask*, *gt_bboxes_ignore=None*)

Training forward function of PartAggregationROIHead.

参数

- **feats_dict** (*dict*) –Contains features from the first stage.
- **img_metas** (*list[dict]*) –Contain pcd and img's meta info.
- **points** (*list[torch.Tensor]*) –Input points.
- **gt_bboxes_3d** (list[BaseInstance3DBoxes]) –Ground truth bboxes of each sample.
- **gt_labels_3d** (*list[torch.Tensor]*) –Labels of each sample.
- **pts_semantic_mask** (*list[torch.Tensor]*) –Point-wise semantic mask.
- **pts_instance_mask** (*list[torch.Tensor]*) –Point-wise instance mask.
- **gt_bboxes_ignore** (*list[torch.Tensor]*) –Specify which bounding boxes to ignore.

返回 losses from each head.

返回类型 dict

init_assigner_sampler()

Initialize assigner and sampler.

init_bbox_head(*bbox_head*)

Initialize box head.

init_mask_head()

Initialize mask head, skip since H3DROIHead does not have one.

simple_test(*feats_dict*, *img_metas*, *points*, *rescale=False*)

Simple testing forward function of PartAggregationROIHead.

注解: This function assumes that the batch size is 1

参数

- **feats_dict** (*dict*) –Contains features from the first stage.
- **img_metas** (*list[dict]*) –Contain pcd and img' s meta info.
- **points** (*torch.Tensor*) –Input points.
- **rescale** (*bool*) –Whether to rescale results.

返回 Bbox results of one frame.

返回类型 dict

class mmdet3d.models.roi_heads.**PartA2BboxHead**(*num_classes*, *seg_in_channels*, *part_in_channels*, *seg_conv_channels=None*, *part_conv_channels=None*, *merge_conv_channels=None*, *down_conv_channels=None*, *shared_fc_channels=None*, *cls_channels=None*, *reg_channels=None*, *dropout_ratio=0.1*, *roi_feat_size=14*, *with_corner_loss=True*, *bbox_coder={'type': 'DeltaXYZWLHRBBoxCoder'}*, *conv_cfg={'type': 'Conv1d'}*, *norm_cfg={'eps': 0.001, 'momentum': 0.01, 'type': 'BN1d'}*, *loss_bbox={'beta': 0.1111111111111111, 'loss_weight': 2.0, 'type': 'SmoothL1Loss'}*, *loss_cls={'loss_weight': 1.0, 'reduction': 'none', 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, *init_cfg=None*)

PartA2 RoI head.

参数

- **num_classes** (*int*) –The number of classes to prediction.
- **seg_in_channels** (*int*) –Input channels of segmentation convolution layer.
- **part_in_channels** (*int*) –Input channels of part convolution layer.
- **seg_conv_channels** (*list(int)*) –Out channels of each segmentation convolution layer.
- **part_conv_channels** (*list(int)*) –Out channels of each part convolution layer.
- **merge_conv_channels** (*list(int)*) –Out channels of each feature merged convolution layer.
- **down_conv_channels** (*list(int)*) –Out channels of each downsampled convolution layer.
- **shared_fc_channels** (*list(int)*) –Out channels of each shared fc layer.

- **cls_channels** (*list(int)*) –Out channels of each classification layer.
- **reg_channels** (*list(int)*) –Out channels of each regression layer.
- **dropout_ratio** (*float*) –Dropout ratio of classification and regression layers.
- **roi_feat_size** (*int*) –The size of pooled roi features.
- **with_corner_loss** (*bool*) –Whether to use corner loss or not.
- **bbox_coder** (BaseBBoxCoder) –Bbox coder for box head.
- **conv_cfg** (*dict*) –Config dict of convolutional layers
- **norm_cfg** (*dict*) –Config dict of normalization layers
- **loss_bbox** (*dict*) –Config dict of box regression loss.
- **loss_cls** (*dict*) –Config dict of classifacation loss.

forward(*seg_feats*, *part_feats*)

Forward pass.

参数

- **seg_feats** (*torch.Tensor*) –Point-wise semantic features.
- **part_feats** (*torch.Tensor*) –Point-wise part prediction features.

返回 Score of class and bbox predictions.

返回类型 tuple[torch.Tensor]

get_bboxes(*rois*, *cls_score*, *bbox_pred*, *class_labels*, *class_pred*, *img_metas*, *cfg=None*)

Generate bboxes from bbox head predictions.

参数

- **rois** (*torch.Tensor*) –Roi bounding boxes.
- **cls_score** (*torch.Tensor*) –Scores of bounding boxes.
- **bbox_pred** (*torch.Tensor*) –Bounding boxes predictions
- **class_labels** (*torch.Tensor*) –Label of classes
- **class_pred** (*torch.Tensor*) –Score for nms.
- **img_metas** (*list[dict]*) –Point cloud and image’s meta info.
- **cfg** (ConfigDict) –Testing config.

返回 Decoded bbox, scores and labels after nms.

返回类型 list[tuple]

get_corner_loss_lidar(*pred_bbox3d*, *gt_bbox3d*, *delta=1.0*)

Calculate corner loss of given boxes.

参数

- **pred_bbox3d** (*torch.FloatTensor*) –Predicted boxes in shape (N, 7).
- **gt_bbox3d** (*torch.FloatTensor*) –Ground truth boxes in shape (N, 7).
- **delta** (*float, optional*) –huber loss threshold. Defaults to 1.0

返回 Calculated corner loss in shape (N).

返回类型 torch.FloatTensor

get_targets(*sampling_results*, *rcnn_train_cfg*, *concat=True*)

Generate targets.

参数

- **sampling_results** (list[SamplingResult]) –Sampled results from rois.
- **rcnn_train_cfg** (ConfigDict) –Training config of rcnn.
- **concat** (*bool*) –Whether to concatenate targets between batches.

返回 Targets of boxes and class prediction.

返回类型 tuple[torch.Tensor]

init_weights()

Initialize the weights.

loss(*cls_score*, *bbox_pred*, *rois*, *labels*, *bbox_targets*, *pos_gt_bboxes*, *reg_mask*, *label_weights*, *bbox_weights*)

Computing losses.

参数

- **cls_score** (*torch.Tensor*) –Scores of each roi.
- **bbox_pred** (*torch.Tensor*) –Predictions of bboxes.
- **rois** (*torch.Tensor*) –Roi bboxes.
- **labels** (*torch.Tensor*) –Labels of class.
- **bbox_targets** (*torch.Tensor*) –Target of positive bboxes.
- **pos_gt_bboxes** (*torch.Tensor*) –Ground truths of positive bboxes.
- **reg_mask** (*torch.Tensor*) –Mask for positive bboxes.
- **label_weights** (*torch.Tensor*) –Weights of class loss.
- **bbox_weights** (*torch.Tensor*) –Weights of bbox loss.

返回

Computed losses.

- loss_cls (torch.Tensor): Loss of classes.

- loss_bbox (torch.Tensor): Loss of bboxes.
- loss_corner (torch.Tensor): Loss of corners.

返回类型 dict

multi_class_nms(*box_probs*, *box_preds*, *score_thr*, *nms_thr*, *input_meta*, *use_rotate_nms=True*)

Multi-class NMS for box head.

注解: This function has large overlap with the *box3d_multiclass_nms* implemented in *mmdet3d.core.post_processing*. We are considering merging these two functions in the future.

参数

- **box_probs** (*torch.Tensor*) –Predicted boxes probabitilies in shape (N,).
- **box_preds** (*torch.Tensor*) –Predicted boxes in shape (N, 7+C).
- **score_thr** (*float*) –Threshold of scores.
- **nms_thr** (*float*) –Threshold for NMS.
- **input_meta** (*dict*) –Meta information of the current sample.
- **use_rotate_nms** (*bool, optional*) –Whether to use rotated nms. Defaults to True.

返回 Selected indices.

返回类型 torch.Tensor

class mmdet3d.models.roi_heads.**PartAggregationROIHead**(*semantic_head*, *num_classes=3*, *seg_roi_extractor=None*, *part_roi_extractor=None*, *bbox_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

Part aggregation roi head for PartA2.

参数

- **semantic_head** (*ConfigDict*) –Config of semantic head.
- **num_classes** (*int*) –The number of classes.
- **seg_roi_extractor** (*ConfigDict*) –Config of seg_roi_extractor.
- **part_roi_extractor** (*ConfigDict*) –Config of part_roi_extractor.
- **bbox_head** (*ConfigDict*) –Config of bbox_head.
- **train_cfg** (*ConfigDict*) –Training config.

- **test_cfg** (*ConfigDict*) –Testing config.

forward_train(*feats_dict*, *voxels_dict*, *img_metas*, *proposal_list*, *gt_bboxes_3d*, *gt_labels_3d*)

Training forward function of PartAggregationROIHead.

参数

- **feats_dict** (*dict*) –Contains features from the first stage.
- **voxels_dict** (*dict*) –Contains information of voxels.
- **img_metas** (*list[dict]*) –Meta info of each image.
- **proposal_list** (*list[dict]*) –Proposal information from rpn. The dictionary should contain the following keys:
  - boxes_3d (BaseInstance3DBoxes): Proposal bboxes
  - labels_3d (torch.Tensor): Labels of proposals
  - cls_preds (torch.Tensor): Original scores of proposals
- **gt_bboxes_3d** (list[BaseInstance3DBoxes]) –GT bboxes of each sample. The bboxes are encapsulated by 3D box structures.
- **gt_labels_3d** (*list[LongTensor]*) –GT labels of each sample.

返回

losses from each head.

- loss_semantic (torch.Tensor): loss of semantic head
- loss_bbox (torch.Tensor): loss of bboxes

返回类型 dict

init_assigner_sampler()

Initialize assigner and sampler.

init_bbox_head(*bbox_head*)

Initialize box head.

init_mask_head()

Initialize mask head, skip since PartAggregationROIHead does not have one.

simple_test(*feats_dict*, *voxels_dict*, *img_metas*, *proposal_list*, ***kwargs*)

Simple testing forward function of PartAggregationROIHead.

注解: This function assumes that the batch size is 1

参数

- **feats_dict** (*dict*) –Contains features from the first stage.
- **voxels_dict** (*dict*) –Contains information of voxels.
- **img_metas** (*list[dict]*) –Meta info of each image.
- **proposal_list** (*list[dict]*) –Proposal information from rpn.

返回 Bbox results of one frame.

返回类型 dict

property with_semantic

whether the head has semantic branch

Type bool

class mmdet3d.models.roi_heads.**PointRCNNBboxHead**(*num_classes*, *in_channels*, *mlp_channels*, *pred_layer_cfg=None*, *num_points=(128, 32, - 1)*, *radius=(0.2, 0.4, 100)*, *num_samples=(64, 64, 64)*, *sa_channels=((128, 128, 128), (128, 128, 256), (256, 256, 512))*, *bbox_coder={'type': 'DeltaXYZWLHRBBoxCoder'}*, *sa_cfg={'pool_mod': 'max', 'type': 'PointSAModule', 'use_xyz': True}*, *conv_cfg={'type': 'Conv1d'}*, *norm_cfg={'type': 'BN1d'}*, *act_cfg={'type': 'ReLU'}*, *bias='auto'*, *loss_bbox={'beta': 0.1111111111111111, 'loss_weight': 1.0, 'reduction': 'sum', 'type': 'SmoothL1Loss'}*, *loss_cls={'loss_weight': 1.0, 'reduction': 'sum', 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, *with_corner_loss=True*, *init_cfg=None*)

PointRCNN RoI Bbox head.

参数

- **num_classes** (*int*) –The number of classes to prediction.
- **in_channels** (*int*) –
- **mlp_channels** (*list[int]*) –the number of mlp channels
- **pred_layer_cfg** (*dict, optional*) –Config of classfication and regression prediction layers. Defaults to None.
- **num_points** (*tuple, optional*) –The number of points which each SA module samples. Defaults to (128, 32, -1).

- **radius** (*tuple, optional*) –Sampling radius of each SA module. Defaults to (0.2, 0.4, 100).
- **num_samples** (*tuple, optional*) –The number of samples for ball query in each SA module. Defaults to (64, 64, 64).
- **sa_channels** (*tuple, optional*) –Out channels of each mlp in SA module. Defaults to ((128, 128, 128), (128, 128, 256), (256, 256, 512)).
- **bbox_coder** (*dict, optional*) –Config dict of box coders. Defaults to dict(type='DeltaXYZWLHRBBoxCoder').
- **sa_cfg** (*dict, optional*) –Config of set abstraction module, which may contain the following keys and values:
  - pool_mod (str): Pool method ('max' or 'avg') for SA modules.
  - use_xyz (bool): Whether to use xyz as a part of features.
  - normalize_xyz (bool): Whether to normalize xyz with radii in each SA module.

  Defaults to dict(type='PointSAModule', pool_mod='max', use_xyz=True).
- **conv_cfg** (*dict, optional*) –Config dict of convolutional layers. Defaults to dict(type='Conv1d').
- **norm_cfg** (*dict, optional*) –Config dict of normalization layers. Defaults to dict(type='BN1d').
- **act_cfg** (*dict, optional*) –Config dict of activation layers. Defaults to dict(type='ReLU').
- **bias** (*str, optional*) –Type of bias. Defaults to 'auto'.
- **loss_bbox** (*dict, optional*) –Config of regression loss function. Defaults to dict(type='SmoothL1Loss', beta=1.0 / 9.0,

  reduction='sum', loss_weight=1.0).
- **loss_cls** (*dict, optional*) –Config of classification loss function. Defaults to dict(type='CrossEntropyLoss', use_sigmoid=True,

  reduction='sum', loss_weight=1.0).
- **with_corner_loss** (*bool, optional*) –Whether using corner loss. Defaults to True.
- **init_cfg** (*dict, optional*) –Config of initialization. Defaults to None.

forward (*feats*)

Forward pass.

参数 **feats** (*torch.Torch*) –Features from RCNN modules.

返回 Score of class and bbox predictions.

返回类型 tuple[torch.Tensor]

get_bboxes(*rois*, *cls_score*, *bbox_pred*, *class_labels*, *img_metas*, *cfg=None*)

Generate bboxes from bbox head predictions.

参数

- **rois** (*torch.Tensor*) –RoI bounding boxes.
- **cls_score** (*torch.Tensor*) –Scores of bounding boxes.
- **bbox_pred** (*torch.Tensor*) –Bounding boxes predictions
- **class_labels** (*torch.Tensor*) –Label of classes
- **img_metas** (*list[dict]*) –Point cloud and image's meta info.
- **cfg** (ConfigDict, optional) –Testing config. Defaults to None.

返回 Decoded bbox, scores and labels after nms.

返回类型 list[tuple]

get_corner_loss_lidar(*pred_bbox3d*, *gt_bbox3d*, *delta=1.0*)

Calculate corner loss of given boxes.

参数

- **pred_bbox3d** (*torch.FloatTensor*) –Predicted boxes in shape (N, 7).
- **gt_bbox3d** (*torch.FloatTensor*) –Ground truth boxes in shape (N, 7).
- **delta** (*float, optional*) –huber loss threshold. Defaults to 1.0

返回 Calculated corner loss in shape (N).

返回类型 torch.FloatTensor

get_targets(*sampling_results*, *rcnn_train_cfg*, *concat=True*)

Generate targets.

参数

- **sampling_results** (list[SamplingResult]) –Sampled results from rois.
- **rcnn_train_cfg** (ConfigDict) –Training config of rcnn.
- **concat** (*bool, optional*) –Whether to concatenate targets between batches. Defaults to True.

返回 Targets of boxes and class prediction.

返回类型 tuple[torch.Tensor]

init_weights()

Initialize weights of the head.

loss(*cls_score*, *bbox_pred*, *rois*, *labels*, *bbox_targets*, *pos_gt_bboxes*, *reg_mask*, *label_weights*, *bbox_weights*)

Computing losses.

参数

- **cls_score** (*torch.Tensor*) –Scores of each RoI.
- **bbox_pred** (*torch.Tensor*) –Predictions of bboxes.
- **rois** (*torch.Tensor*) –RoI bboxes.
- **labels** (*torch.Tensor*) –Labels of class.
- **bbox_targets** (*torch.Tensor*) –Target of positive bboxes.
- **pos_gt_bboxes** (*torch.Tensor*) –Ground truths of positive bboxes.
- **reg_mask** (*torch.Tensor*) –Mask for positive bboxes.
- **label_weights** (*torch.Tensor*) –Weights of class loss.
- **bbox_weights** (*torch.Tensor*) –Weights of bbox loss.

返回

Computed losses.

- loss_cls (torch.Tensor): Loss of classes.
- loss_bbox (torch.Tensor): Loss of bboxes.
- loss_corner (torch.Tensor): Loss of corners.

返回类型 dict

multi_class_nms(*box_probs*, *box_preds*, *score_thr*, *nms_thr*, *input_meta*, *use_rotate_nms=True*)

Multi-class NMS for box head.

注解: This function has large overlap with the *box3d_multiclass_nms* implemented in *mmdet3d.core.post_processing*. We are considering merging these two functions in the future.

参数

- **box_probs** (*torch.Tensor*) –Predicted boxes probabilities in shape (N,).
- **box_preds** (*torch.Tensor*) –Predicted boxes in shape (N, 7+C).
- **score_thr** (*float*) –Threshold of scores.
- **nms_thr** (*float*) –Threshold for NMS.
- **input_meta** (*dict*) –Meta information of the current sample.

- **use_rotate_nms** (*bool, optional*) –Whether to use rotated nms. Defaults to True.

返回 Selected indices.

返回类型 torch.Tensor

class mmdet3d.models.roi_heads.**PointRCNNRoIHead**(*bbox_head*, *point_roi_extractor*, *train_cfg*, *test_cfg*, *depth_normalizer=70.0*, *pretrained=None*, *init_cfg=None*)

RoI head for PointRCNN.

参数

- **bbox_head** (*dict*) –Config of bbox_head.
- **point_roi_extractor** (*dict*) –Config of RoI extractor.
- **train_cfg** (*dict*) –Train configs.
- **test_cfg** (*dict*) –Test configs.
- **depth_normalizer** (*float, optional*) –Normalize depth feature. Defaults to 70.0.
- **init_cfg** (*dict, optional*) –Config of initialization. Defaults to None.

forward_train(*feats_dict*, *input_metas*, *proposal_list*, *gt_bboxes_3d*, *gt_labels_3d*)

Training forward function of PointRCNNRoIHead.

参数

- **feats_dict** (*dict*) –Contains features from the first stage.
- **imput_metas** (*list[dict]*) –Meta info of each input.
- **proposal_list** (*list[dict]*) –Proposal information from rpn. The dictionary should contain the following keys:
  - boxes_3d (BaseInstance3DBoxes): Proposal bboxes
  - labels_3d (torch.Tensor): Labels of proposals
- **gt_bboxes_3d** (list[BaseInstance3DBoxes]) –GT bboxes of each sample. The bboxes are encapsulated by 3D box structures.
- **gt_labels_3d** (*list[LongTensor]*) –GT labels of each sample.

返回

Losses from RoI RCNN head.

- loss_bbox (torch.Tensor): Loss of bboxes

返回类型 dict

init_assigner_sampler()
Initialize assigner and sampler.

init_bbox_head(*bbox_head*)
Initialize box head.

参数 **bbox_head** (*dict*) –Config dict of RoI Head.

init_mask_head()
Initialize maek head.

simple_test(*feats_dict*, *img_metas*, *proposal_list*, ***kwargs*)
Simple testing forward function of PointRCNNRoIHead.

注解: This function assumes that the batch size is 1

参数

- **feats_dict** (*dict*) –Contains features from the first stage.
- **img_metas** (*list[dict]*) –Meta info of each image.
- **proposal_list** (*list[dict]*) –Proposal information from rpn.

返回 Bbox results of one frame.

返回类型 dict

class mmdet3d.models.roi_heads.**PointwiseSemanticHead**(*in_channels*, *num_classes=3*, *extra_width=0.2*, *seg_score_thr=0.3*, *init_cfg=None*, *loss_seg={'alpha': 0.25, 'gamma': 2.0, 'loss_weight': 1.0, 'reduction': 'sum', 'type': 'FocalLoss', 'use_sigmoid': True}*, *loss_part={'loss_weight': 1.0, 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*)
Semantic segmentation head for point-wise segmentation.

Predict point-wise segmentation and part regression results for PartA2. See paper for more details.

参数

- **in_channels** (*int*) –The number of input channel.
- **num_classes** (*int*) –The number of class.
- **extra_width** (*float*) –Boxes enlarge width.
- **loss_seg** (*dict*) –Config of segmentation loss.

- **loss_part** (*dict*) –Config of part prediction loss.

forward(*x*)

Forward pass.

参数 **x** (*torch.Tensor*) –Features from the first stage.

返回

Part features, segmentation and part predictions.

- seg_preds (torch.Tensor): Segment predictions.
- part_preds (torch.Tensor): Part predictions.
- part_feats (torch.Tensor): Feature predictions.

返回类型 dict

get_targets(*voxels_dict*, *gt_bboxes_3d*, *gt_labels_3d*)

generate segmentation and part prediction targets.

参数

- **voxel_centers** (*torch.Tensor*) –The center of voxels in shape (voxel_num, 3).
- **gt_bboxes_3d** (BaseInstance3DBoxes) –Ground truth boxes in shape (box_num, 7).
- **gt_labels_3d** (*torch.Tensor*) –Class labels of ground truths in shape (box_num).

返回

Prediction targets

- **seg_targets (torch.Tensor): Segmentation targets** with shape [voxel_num].
- **part_targets (torch.Tensor): Part prediction targets** with shape [voxel_num, 3].

返回类型 dict

get_targets_single(*voxel_centers*, *gt_bboxes_3d*, *gt_labels_3d*)

generate segmentation and part prediction targets for a single sample.

参数

- **voxel_centers** (*torch.Tensor*) –The center of voxels in shape (voxel_num, 3).
- **gt_bboxes_3d** (BaseInstance3DBoxes) –Ground truth boxes in shape (box_num, 7).
- **gt_labels_3d** (*torch.Tensor*) –Class labels of ground truths in shape (box_num).

返回

Segmentation targets with shape [voxel_num] part prediction targets with shape [voxel_num, 3]

返回类型 tuple[torch.Tensor]

loss(*semantic_results*, *semantic_targets*)

Calculate point-wise segmentation and part prediction losses.

参数

- **semantic_results** (*dict*) –Results from semantic head.
  - seg_preds: Segmentation predictions.
  - part_preds: Part predictions.
- **semantic_targets** (*dict*) –Targets of semantic results.
  - seg_preds: Segmentation targets.
  - part_preds: Part targets.

返回

Loss of segmentation and part prediction.

- loss_seg (torch.Tensor): Segmentation prediction loss.
- loss_part (torch.Tensor): Part prediction loss.

返回类型 dict

class mmdet3d.models.roi_heads.**PrimitiveHead**(*num_dims*, *num_classes*, *primitive_mode*, *train_cfg=None*, *test_cfg=None*, *vote_module_cfg=None*, *vote_aggregation_cfg=None*, *feat_channels=(128, 128)*, *upper_thresh=100.0*, *surface_thresh=0.5*, *conv_cfg={'type': 'Conv1d'}*, *norm_cfg={'type': 'BN1d'}*, *objectness_loss=None*, *center_loss=None*, *semantic_reg_loss=None*, *semantic_cls_loss=None*, *init_cfg=None*)

Primitive head of H3DNet.

参数

- **num_dims** (*int*) –The dimension of primitive semantic information.
- **num_classes** (*int*) –The number of class.
- **primitive_mode** (*str*) –The mode of primitive module, available mode [‘z’, ‘xy’, ‘line’].

- **bbox_coder** (BaseBBoxCoder) –Bbox coder for encoding and decoding boxes.
- **train_cfg** (*dict*) –Config for training.
- **test_cfg** (*dict*) –Config for testing.
- **vote_module_cfg** (*dict*) –Config of VoteModule for point-wise votes.
- **vote_aggregation_cfg** (*dict*) –Config of vote aggregation layer.
- **feat_channels** (*tuple[int]*) –Convolution channels of prediction layer.
- **upper_thresh** (*float*) –Threshold for line matching.
- **surface_thresh** (*float*) –Threshold for surface matching.
- **conv_cfg** (*dict*) –Config of convolution in prediction layer.
- **norm_cfg** (*dict*) –Config of BN in prediction layer.
- **objectness_loss** (*dict*) –Config of objectness loss.
- **center_loss** (*dict*) –Config of center loss.
- **semantic_loss** (*dict*) –Config of point-wise semantic segmentation loss.

check_dist(*plane_equ*, *points*)

Whether the mean of points to plane distance is lower than thresh.

参数

- **plane_equ** (*torch.Tensor*) –Plane to be checked.
- **points** (*torch.Tensor*) –Points to be checked.

返回 Flag of result.

返回类型 Tuple

check_horizon(*points*)

Check whether is a horizontal plane.

参数 **points** (*torch.Tensor*) –Points of input.

返回 Flag of result.

返回类型 Bool

compute_primitive_loss(*primitive_center*, *primitive_semantic*, *semantic_scores*, *num_proposal*, *gt_primitive_center*, *gt_primitive_semantic*, *gt_sem_cls_label*, *gt_primitive_mask*)

Compute loss of primitive module.

参数

- **primitive_center** (*torch.Tensor*) –Pridictions of primitive center.
- **primitive_semantic** (*torch.Tensor*) –Pridictions of primitive semantic.

- **semantic_scores** (*torch.Tensor*) –Pridictions of primitive semantic scores.
- **num_proposal** (*int*) –The number of primitive proposal.
- **gt_primitive_center** (*torch.Tensor*) –Ground truth of primitive center.
- **gt_votes_sem** (*torch.Tensor*) –Ground truth of primitive semantic.
- **gt_sem_cls_label** (*torch.Tensor*) –Ground truth of primitive semantic class.
- **gt_primitive_mask** (*torch.Tensor*) –Ground truth of primitive mask.

返回 Loss of primitive module.

返回类型 Tuple

forward(*feats_dict*, *sample_mod*)

Forward pass.

参数

- **feats_dict** (*dict*) –Feature dict from backbone.
- **sample_mod** (*str*) –Sample mode for vote aggregation layer. valid modes are "vote", "seed" and "random".

返回 Predictions of primitive head.

返回类型 dict

get_primitive_center(*pred_flag*, *center*)

Generate primitive center from predictions.

参数

- **pred_flag** (*torch.Tensor*) –Scores of primitive center.
- **center** (*torch.Tensor*) –Pridictions of primitive center.

返回 Primitive center and the prediction indices.

返回类型 Tuple

get_targets(*points*, *gt_bboxes_3d*, *gt_labels_3d*, *pts_semantic_mask=None*, *pts_instance_mask=None*, *bbox_preds=None*)

Generate targets of primitive head.

参数

- **points** (*list[torch.Tensor]*) –Points of each batch.
- **gt_bboxes_3d** (list[BaseInstance3DBoxes]) –Ground truth bboxes of each batch.
- **gt_labels_3d** (*list[torch.Tensor]*) –Labels of each batch.

- **pts_semantic_mask** (*list[torch.Tensor]*) –Point-wise semantic label of each batch.
- **pts_instance_mask** (*list[torch.Tensor]*) –Point-wise instance label of each batch.
- **bbox_preds** (*dict*) –Predictions from forward of primitive head.

返回 Targets of primitive head.

返回类型 tuple[torch.Tensor]

get_targets_single(*points*, *gt_bboxes_3d*, *gt_labels_3d*, *pts_semantic_mask=None*, *pts_instance_mask=None*)

Generate targets of primitive head for single batch.

参数

- **points** (*torch.Tensor*) –Points of each batch.
- **gt_bboxes_3d** (BaseInstance3DBoxes) –Ground truth boxes of each batch.
- **gt_labels_3d** (*torch.Tensor*) –Labels of each batch.
- **pts_semantic_mask** (*torch.Tensor*) –Point-wise semantic label of each batch.
- **pts_instance_mask** (*torch.Tensor*) –Point-wise instance label of each batch.

返回 Targets of primitive head.

返回类型 tuple[torch.Tensor]

loss(*bbox_preds*, *points*, *gt_bboxes_3d*, *gt_labels_3d*, *pts_semantic_mask=None*, *pts_instance_mask=None*, *img_metas=None*, *gt_bboxes_ignore=None*)

Compute loss.

参数

- **bbox_preds** (*dict*) –Predictions from forward of primitive head.
- **points** (*list[torch.Tensor]*) –Input points.
- **gt_bboxes_3d** (list[BaseInstance3DBoxes]) –Ground truth bboxes of each sample.
- **gt_labels_3d** (*list[torch.Tensor]*) –Labels of each sample.
- **pts_semantic_mask** (*list[torch.Tensor]*) –Point-wise semantic mask.
- **pts_instance_mask** (*list[torch.Tensor]*) –Point-wise instance mask.
- **img_metas** (*list[dict]*) –Contain pcd and img’s meta info.
- **gt_bboxes_ignore** (*list[torch.Tensor]*) –Specify which bounding.

返回 Losses of Primitive Head.

返回类型 dict

match_point2line(*points*, *corners*, *with_yaw*, *mode='bottom'*)

Match points to corresponding line.

参数

- **points** (*torch.Tensor*) –Points of input.
- **corners** (*torch.Tensor*) –Eight corners of a bounding box.
- **with_yaw** (*Bool*) –Whether the boundind box is with rotation.
- **mode** (*str, optional*) –Specify which line should be matched, available mode are ('bottom', 'top', 'left', 'right'). Defaults to 'bottom'.

返回 Flag of matching correspondence.

返回类型 Tuple

match_point2plane(*plane*, *points*)

Match points to plane.

参数

- **plane** (*torch.Tensor*) –Equation of the plane.
- **points** (*torch.Tensor*) –Points of input.

返回

Distance of each point to the plane and flag of matching correspondence.

返回类型 Tuple

point2line_dist(*points*, *pts_a*, *pts_b*)

Calculate the distance from point to line.

参数

- **points** (*torch.Tensor*) –Points of input.
- **pts_a** (*torch.Tensor*) –Point on the specific line.
- **pts_b** (*torch.Tensor*) –Point on the specific line.

返回 Distance between each point to line.

返回类型 torch.Tensor

primitive_decode_scores(*predictions*, *aggregated_points*)

Decode predicted parts to primitive head.

参数

- **predictions** (*torch.Tensor*) –primitive pridictions of each batch.

- **aggregated_points** (*torch.Tensor*) –The aggregated points of vote stage.

返回

Predictions of primitive head, including center, semantic size and semantic scores.

返回类型 Dict

class mmdet3d.models.roi_heads.**Single3DRoIAwareExtractor**(*roi_layer=None*, *init_cfg=None*)

Point-wise roi-aware Extractor.

Extract Point-wise roi features.

参数 **roi_layer** (*dict*) –The config of roi layer.

build_roi_layers(*layer_cfg*)

Build roi layers using *layer_cfg*

forward(*feats*, *coordinate*, *batch_inds*, *rois*)

Extract point-wise roi features.

参数

- **feats** (*torch.FloatTensor*) –Point-wise features with shape (batch, npoints, channels) for pooling.
- **coordinate** (*torch.FloatTensor*) –Coordinate of each point.
- **batch_inds** (*torch.LongTensor*) –Indicate the batch of each point.
- **rois** (*torch.FloatTensor*) –Roi boxes with batch indices.

返回 Pooled features

返回类型 torch.FloatTensor

class mmdet3d.models.roi_heads.**Single3DRoIPointExtractor**(*roi_layer=None*)

Point-wise roi-aware Extractor.

Extract Point-wise roi features.

参数 **roi_layer** (*dict*) –The config of roi layer.

build_roi_layers(*layer_cfg*)

Build roi layers using *layer_cfg*

forward(*feats*, *coordinate*, *batch_inds*, *rois*)

Extract point-wise roi features.

参数

- **feats** (*torch.FloatTensor*) –Point-wise features with shape (batch, npoints, channels) for pooling.
- **coordinate** (*torch.FloatTensor*) –Coordinate of each point.

- **batch_inds** (*torch.LongTensor*) –Indicate the batch of each point.
- **rois** (*torch.FloatTensor*) –Roi boxes with batch indices.

返回 Pooled features

返回类型 torch.FloatTensor

class mmdet3d.models.roi_heads.**SingleRoIExtractor**(*roi_layer*, *out_channels*, *featmap_strides*, *finest_scale=56*, *init_cfg=None*)

Extract RoI features from a single level feature map.

If there are multiple input feature levels, each RoI is mapped to a level according to its scale. The mapping rule is proposed in FPN.

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

forward(*feats*, *rois*, *roi_scale_factor=None*)

Forward function.

map_roi_levels(*rois*, *num_levels*)

Map rois to corresponding feature levels by scales.

- scale < finest_scale * 2: level 0
- finest_scale * 2 <= scale < finest_scale * 4: level 1
- finest_scale * 4 <= scale < finest_scale * 8: level 2
- scale >= finest_scale * 8: level 3

参数

- **rois** (*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

40.6 fusion_layers

class mmdet3d.models.fusion_layers.**PointFusion**(*img_channels*, *pts_channels*, *mid_channels*, *out_channels*, *img_levels=3*, *coord_type='LIDAR'*, *conv_cfg=None*, *norm_cfg=None*, *act_cfg=None*, *init_cfg=None*, *activate_out=True*, *fuse_out=False*, *dropout_ratio=0*, *aligned=True*, *align_corners=True*, *padding_mode='zeros'*, *lateral_conv=True*)

Fuse image features from multi-scale features.

参数

- **img_channels** (*list[int] | int*) –Channels of image features. It could be a list if the input is multi-scale image features.
- **pts_channels** (*int*) –Channels of point features
- **mid_channels** (*int*) –Channels of middle layers
- **out_channels** (*int*) –Channels of output fused features
- **img_levels** (*int, optional*) –Number of image levels. Defaults to 3.
- **coord_type** (*str*) –'DEPTH' or 'CAMERA' or 'LIDAR'. Defaults to 'LIDAR'.
- **conv_cfg** (*dict, optional*) –Dict config of conv layers of middle layers. Defaults to None.
- **norm_cfg** (*dict, optional*) –Dict config of norm layers of middle layers. Defaults to None.
- **act_cfg** (*dict, optional*) –Dict config of activatation layers. Defaults to None.
- **activate_out** (*bool, optional*) –Whether to apply relu activation to output features. Defaults to True.
- **fuse_out** (*bool, optional*) –Whether apply conv layer to the fused features. Defaults to False.
- **dropout_ratio** (*int, float, optional*) –Dropout ratio of image features to prevent overfitting. Defaults to 0.
- **aligned** (*bool, optional*) –Whether apply aligned feature fusion. Defaults to True.
- **align_corners** (*bool, optional*) –Whether to align corner when sampling features according to points. Defaults to True.

- **padding_mode** (*str, optional*) –Mode used to pad the features of points that do not have corresponding image features. Defaults to ‘zeros’.
- **lateral_conv** (*bool, optional*) –Whether to apply lateral convs to image features. Defaults to True.

forward(*img_feats*, *pts*, *pts_feats*, *img_metas*)

Forward function.

参数

- **img_feats** (*list[torch.Tensor]*) –Image features.
- **pts** –[list[torch.Tensor]]: A batch of points with shape N x 3.
- **pts_feats** (*torch.Tensor*) –A tensor consist of point features of the total batch.
- **img_metas** (*list[dict]*) –Meta information of images.

返回 Fused features of each point.

返回类型 torch.Tensor

obtain_mlvl_feats(*img_feats*, *pts*, *img_metas*)

Obtain multi-level features for each point.

参数

- **img_feats** (*list(torch.Tensor)*) –Multi-scale image features produced by image backbone in shape (N, C, H, W).
- **pts** (*list[torch.Tensor]*) –Points of each sample.
- **img_metas** (*list[dict]*) –Meta information for each sample.

返回 Corresponding image features of each point.

返回类型 torch.Tensor

sample_single(*img_feats*, *pts*, *img_meta*)

Sample features from single level image feature map.

参数

- **img_feats** (*torch.Tensor*) –Image feature map in shape (1, C, H, W).
- **pts** (*torch.Tensor*) –Points of a single sample.
- **img_meta** (*dict*) –Meta information of the single sample.

返回 Single level image features of each point.

返回类型 torch.Tensor

class mmdet3d.models.fusion_layers.**VoteFusion**(*num_classes=10*, *max_imvote_per_pixel=3*)

Fuse 2d features from 3d seeds.

参数

- **num_classes** (*int*) –number of classes.
- **max_imvote_per_pixel** (*int*) –max number of imvotes.

forward(*imgs*, *bboxes_2d_rescaled*, *seeds_3d_depth*, *img_metas*)

Forward function.

参数

- **imgs** (*list[torch.Tensor]*) –Image features.
- **bboxes_2d_rescaled** (*list[torch.Tensor]*) –2D bboxes.
- **seeds_3d_depth** (*torch.Tensor*) –3D seeds.
- **img_metas** (*list[dict]*) –Meta information of images.

返回 Concatenated cues of each point. torch.Tensor: Validity mask of each feature.

返回类型 torch.Tensor

mmdet3d.models.fusion_layers.**apply_3d_transformation**(*pcd*, *coord_type*, *img_meta*, *reverse=False*)

Apply transformation to input point cloud.

参数

- **pcd** (*torch.Tensor*) –The point cloud to be transformed.
- **coord_type** (*str*) – 'DEPTH' or 'CAMERA' or 'LIDAR'.
- **img_meta** (*dict*) –Meta info regarding data transformation.
- **reverse** (*bool*) –Reversed transformation or not.

注解: The elements in img_meta['transformation_3d_flow']: "T" stands for translation; "S" stands for scale; "R" stands for rotation; "HF" stands for horizontal flip; "VF" stands for vertical flip.

返回 The transformed point cloud.

返回类型 torch.Tensor

mmdet3d.models.fusion_layers.**bbox_2d_transform**(*img_meta*, *bbox_2d*, *ori2new*)

Transform 2d bbox according to img_meta.

参数

- **img_meta** (*dict*) –Meta info regarding data transformation.
- **bbox_2d** (*torch.Tensor*) –Shape (…, >4) The input 2d bboxes to transform.
- **ori2new** (*bool*) –Origin img coord system to new or not.

返回 The transformed 2d bboxes.

返回类型 torch.Tensor

mmdet3d.models.fusion_layers.**coord_2d_transform**(*img_meta*, *coord_2d*, *ori2new*)

Transform 2d pixel coordinates according to img_meta.

参数

- **img_meta** (*dict*) –Meta info regarding data transformation.
- **coord_2d** (*torch.Tensor*) –Shape (…, 2) The input 2d coords to transform.
- **ori2new** (*bool*) –Origin img coord system to new or not.

返回 The transformed 2d coordinates.

返回类型 torch.Tensor

40.7 losses

class mmdet3d.models.losses.**AxisAlignedIoULoss**(*reduction='mean'*, *loss_weight=1.0*)

Calculate the IoU loss (1-IoU) of axis aligned bounding boxes.

参数

- **reduction** (*str*) –Method to reduce losses. The valid reduction method are none, sum or mean.
- **loss_weight** (*float, optional*) –Weight of loss. Defaults to 1.0.

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*, ***kwargs*)

Forward function of loss calculation.

参数

- **pred** (*torch.Tensor*) –Bbox predictions with shape […, 6] (x1, y1, z1, x2, y2, z2).
- **target** (*torch.Tensor*) –Bbox targets (gt) with shape […, 6] (x1, y1, z1, x2, y2, z2).
- **weight** (*torch.Tensor | float, optional*) –Weight of loss. Defaults to None.
- **avg_factor** (*int, optional*) –Average factor that is used to average the loss. Defaults to None.
- **reduction_override** (*str, optional*) –Method to reduce losses. The valid reduction method are ‘none’, ‘sum’ or ‘mean’. Defaults to None.

返回 IoU loss between predictions and targets.

返回类型 torch.Tensor

class mmdet3d.models.losses.**ChamferDistance**(*mode='l2'*, *reduction='mean'*, *loss_src_weight=1.0*, *loss_dst_weight=1.0*)

Calculate Chamfer Distance of two sets.

参数

- **mode** (*str*) –Criterion mode to calculate distance. The valid modes are smooth_l1, l1 or l2.
- **reduction** (*str*) –Method to reduce losses. The valid reduction method are none, sum or mean.
- **loss_src_weight** (*float*) –Weight of loss_source.
- **loss_dst_weight** (*float*) –Weight of loss_target.

forward(*source*, *target*, *src_weight=1.0*, *dst_weight=1.0*, *reduction_override=None*, *return_indices=False*, ***kwargs*)

Forward function of loss calculation.

参数

- **source** (*torch.Tensor*) –Source set with shape [B, N, C] to calculate Chamfer Distance.
- **target** (*torch.Tensor*) –Destination set with shape [B, M, C] to calculate Chamfer Distance.
- **src_weight** (*torch.Tensor | float, optional*) –Weight of source loss. Defaults to 1.0.
- **dst_weight** (*torch.Tensor | float, optional*) –Weight of destination loss. Defaults to 1.0.
- **reduction_override** (*str, optional*) –Method to reduce losses. The valid reduction method are 'none', 'sum' or 'mean'. Defaults to None.
- **return_indices** (*bool, optional*) –Whether to return indices. Defaults to False.

返回

If return_indices=True, return losses of source and target with their corresponding indices in the order of (loss_source, loss_target, indices1, indices2). If return_indices=False, return (loss_source, loss_target).

返回类型 tuple[torch.Tensor]

class mmdet3d.models.losses.**FocalLoss**(*use_sigmoid=True*, *gamma=2.0*, *alpha=0.25*, *reduction='mean'*, *loss_weight=1.0*, *activated=False*)

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 mmdet3d.models.losses.**MultiBinLoss**(*reduction='none'*, *loss_weight=1.0*)

Multi-Bin Loss for orientation.

参数

- **reduction** (*str, optional*) –The method to reduce the loss. Options are 'none', 'mean' and 'sum'. Defaults to 'none'.
- **loss_weight** (*float, optional*) –The weight of loss. Defaults to 1.0.

forward(*pred*, *target*, *num_dir_bins*, *reduction_override=None*)

Forward function.

参数

- **pred** (*torch.Tensor*) –The prediction.
- **target** (*torch.Tensor*) –The learning target of the prediction.
- **num_dir_bins** (*int*) –Number of bins to encode direction angle.
- **reduction_override** (*str, optional*) –The reduction method used to override the original reduction method of the loss. Defaults to None.

class mmdet3d.models.losses.**PAConvRegularizationLoss**(*reduction='mean'*, *loss_weight=1.0*)

Calculate correlation loss of kernel weights in PAConv's weight bank.

This is used as a regularization term in PAConv model training.

参数

- **reduction** (*str*) –Method to reduce losses. The reduction is performed among all PAConv modules instead of prediction tensors. The valid reduction method are none, sum or mean.
- **loss_weight** (*float, optional*) –Weight of loss. Defaults to 1.0.

forward (*modules*, *reduction_override=None*, ***kwargs*)

Forward function of loss calculation.

参数

- **modules** (List[nn.Module] | generator) –A list or a python generator of torch.nn.Modules.
- **reduction_override** (*str, optional*) –Method to reduce losses. The valid reduction method are 'none', 'sum' or 'mean'. Defaults to None.

返回 Correlation loss of kernel weights.

返回类型 torch.Tensor

class mmdet3d.models.losses.**RotatedIoU3DLoss** (*reduction='mean'*, *loss_weight=1.0*)

Calculate the IoU loss (1-IoU) of rotated bounding boxes.

参数

- **reduction** (*str*) –Method to reduce losses. The valid reduction method are none, sum or mean.
- **loss_weight** (*float, optional*) –Weight of loss. Defaults to 1.0.

forward (*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*, ***kwargs*)

Forward function of loss calculation.

参数

- **pred** (*torch.Tensor*) –Bbox predictions with shape […, 7] (x, y, z, w, l, h, alpha).
- **target** (*torch.Tensor*) –Bbox targets (gt) with shape […, 7] (x, y, z, w, l, h, alpha).
- **weight** (*torch.Tensor | float, optional*) –Weight of loss. Defaults to None.
- **avg_factor** (*int, optional*) –Average factor that is used to average the loss. Defaults to None.
- **reduction_override** (*str, optional*) –Method to reduce losses. The valid reduction method are 'none', 'sum' or 'mean'. Defaults to None.

返回 IoU loss between predictions and targets.

返回类型 torch.Tensor

class mmdet3d.models.losses.**SmoothL1Loss**(*beta=1.0*, *reduction='mean'*, *loss_weight=1.0*)

Smooth L1 loss.

参数

- **beta** (*float, optional*) –The threshold in the piecewise function. Defaults to 1.0.
- **reduction** (*str, optional*) –The method to reduce the loss. Options are "none", "mean" and "sum". Defaults to "mean".
- **loss_weight** (*float, optional*) –The weight of loss.

forward(*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*, ***kwargs*)

Forward function.

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.

class mmdet3d.models.losses.**UncertainL1Loss**(*alpha=1.0*, *reduction='mean'*, *loss_weight=1.0*)

L1 loss with uncertainty.

参数

- **alpha** (*float, optional*) –The coefficient of log(sigma). Defaults to 1.0.
- **reduction** (*str, optional*) –The method to reduce the loss. Options are 'none', 'mean' and 'sum'. Defaults to 'mean'.
- **loss_weight** (*float, optional*) –The weight of loss. Defaults to 1.0.

forward(*pred*, *target*, *sigma*, *weight=None*, *avg_factor=None*, *reduction_override=None*)

Forward function.

参数

- **pred** (*torch.Tensor*) –The prediction.
- **target** (*torch.Tensor*) –The learning target of the prediction.
- **sigma** (*torch.Tensor*) –The sigma for uncertainty.

- **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 mmdet3d.models.losses.**UncertainSmoothL1Loss**(*alpha=1.0*, *beta=1.0*, *reduction='mean'*, *loss_weight=1.0*)

Smooth L1 loss with uncertainty.

Please refer to PGD and Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics for more details.

参数

- **alpha** (*float, optional*) –The coefficient of log(sigma). Defaults to 1.0.
- **beta** (*float, optional*) –The threshold in the piecewise function. Defaults to 1.0.
- **reduction** (*str, optional*) –The method to reduce the loss. Options are 'none', 'mean' and 'sum'. Defaults to 'mean'.
- **loss_weight** (*float, optional*) –The weight of loss. Defaults to 1.0

forward(*pred*, *target*, *sigma*, *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.
- **sigma** (*torch.Tensor*) –The sigma for uncertainty.
- **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.

mmdet3d.models.losses.**axis_aligned_iou_loss**(*pred*, *target*)

Calculate the IoU loss (1-IoU) of two sets of axis aligned bounding boxes. Note that predictions and targets are one-to-one corresponded.

参数

- **pred** (*torch.Tensor*) –Bbox predictions with shape […, 6] (x1, y1, z1, x2, y2, z2).

- **target** (*torch.Tensor*) –Bbox targets (gt) with shape […, 6] (x1, y1, z1, x2, y2, z2).

返回 IoU loss between predictions and targets.

返回类型 torch.Tensor

mmdet3d.models.losses.**binary_cross_entropy**(*pred*, *label*, *weight=None*, *reduction='mean'*, *avg_factor=None*, *class_weight=None*, *ignore_index=- 100*, *avg_non_ignore=False*)

Calculate the binary CrossEntropy loss.

参数

- **pred** (*torch.Tensor*) –The prediction with shape (N, 1) or (N,). When the shape of pred is (N, 1), label will be expanded to one-hot format, and when the shape of pred is (N,), label will not be expanded to one-hot format.
- **label** (*torch.Tensor*) –The learning label of the prediction, with shape (N,).
- **weight** (*torch.Tensor, optional*) –Sample-wise loss weight.
- **reduction** (*str, optional*) –The method used to reduce the loss. Options are "none", "mean" and "sum".
- **avg_factor** (*int, optional*) –Average factor that is used to average the loss. Defaults to None.
- **class_weight** (*list[float], optional*) –The weight for each class.
- **ignore_index** (*int | None*) –The label index to be ignored. If None, it will be set to default value. Default: -100.
- **avg_non_ignore** (*bool*) –The flag decides to whether the loss is only averaged over non-ignored targets. Default: False.

返回 The calculated loss.

返回类型 torch.Tensor

mmdet3d.models.losses.**chamfer_distance**(*src*, *dst*, *src_weight=1.0*, *dst_weight=1.0*, *criterion_mode='l2'*, *reduction='mean'*)

Calculate Chamfer Distance of two sets.

参数

- **src** (*torch.Tensor*) –Source set with shape [B, N, C] to calculate Chamfer Distance.
- **dst** (*torch.Tensor*) –Destination set with shape [B, M, C] to calculate Chamfer Distance.
- **src_weight** (*torch.Tensor or float*) –Weight of source loss.
- **dst_weight** (*torch.Tensor or float*) –Weight of destination loss.

- **criterion_mode** (*str*) –Criterion mode to calculate distance. The valid modes are smooth_l1, l1 or l2.
- **reduction** (*str*) –Method to reduce losses. The valid reduction method are ‘none’, ‘sum’ or ‘mean’.

返回

Source and Destination loss with the corresponding indices.

- **loss_src (torch.Tensor): The min distance** from source to destination.
- **loss_dst (torch.Tensor): The min distance** from destination to source.
- **indices1 (torch.Tensor): Index the min distance point** for each point in source to destination.
- **indices2 (torch.Tensor): Index the min distance point** for each point in destination to source.

返回类型 tuple

40.8 middle_encoders

class mmdet3d.models.middle_encoders.**PointPillarsScatter**(*in_channels*, *output_shape*)

Point Pillar’s Scatter.

Converts learned features from dense tensor to sparse pseudo image.

参数

- **in_channels** (*int*) –Channels of input features.
- **output_shape** (*list[int]*) –Required output shape of features.

forward(*voxel_features*, *coors*, *batch_size=None*)

Foraward function to scatter features.

forward_batch(*voxel_features*, *coors*, *batch_size*)

Scatter features of single sample.

参数

- **voxel_features** (*torch.Tensor*) –Voxel features in shape (N, C).
- **coors** (*torch.Tensor*) –Coordinates of each voxel in shape (N, 4). The first column indicates the sample ID.
- **batch_size** (*int*) –Number of samples in the current batch.

forward_single(*voxel_features*, *coors*)

Scatter features of single sample.

参数

- **voxel_features** (*torch.Tensor*) –Voxel features in shape (N, C).
- **coors** (*torch.Tensor*) –Coordinates of each voxel. The first column indicates the sample ID.

class mmdet3d.models.middle_encoders.**SparseEncoder**(*in_channels*, *sparse_shape*, *order=('conv', 'norm', 'act')*, *norm_cfg={'eps': 0.001, 'momentum': 0.01, 'type': 'BN1d'}*, *base_channels=16*, *output_channels=128*, *encoder_channels=((16), (32, 32, 32), (64, 64, 64), (64, 64, 64))*, *encoder_paddings=((1), (1, 1, 1), (1, 1, 1), ((0, 1, 1), 1, 1))*, *block_type='conv_module'*)

Sparse encoder for SECOND and Part-A2.

参数

- **in_channels** (*int*) –The number of input channels.
- **sparse_shape** (*list[int]*) –The sparse shape of input tensor.
- **order** (*list[str], optional*) –Order of conv module. Defaults to (‘conv’, ‘norm’, ‘act’).
- **norm_cfg** (*dict, optional*) –Config of normalization layer. Defaults to dict(type=’BN1d’, eps=1e-3, momentum=0.01).
- **base_channels** (*int, optional*) –Out channels for conv_input layer. Defaults to 16.
- **output_channels** (*int, optional*) –Out channels for conv_out layer. Defaults to 128.
- **encoder_channels** (*tuple[tuple[int]], optional*) –Convolutional channels of each encode block. Defaults to ((16,), (32, 32, 32), (64, 64, 64), (64, 64, 64)).
- **encoder_paddings** (*tuple[tuple[int]], optional*) –Paddings of each encode block. Defaults to ((1,), (1, 1, 1), (1, 1, 1), ((0, 1, 1), 1, 1)).
- **block_type** (*str, optional*) –Type of the block to use. Defaults to ‘conv_module’.

forward(*voxel_features*, *coors*, *batch_size*)

Forward of SparseEncoder.

参数

- **voxel_features** (*torch.Tensor*) –Voxel features in shape (N, C).
- **coors** (*torch.Tensor*) –Coordinates in shape (N, 4), the columns in the order of (batch_idx, z_idx, y_idx, x_idx).
- **batch_size** (*int*) –Batch size.

返回 Backbone features.

返回类型 dict

make_encoder_layers(*make_block*, *norm_cfg*, *in_channels*, *block_type='conv_module'*, *conv_cfg={'type': 'SubMConv3d'}*)

make encoder layers using sparse convs.

参数

- **make_block** (*method*) –A bounded function to build blocks.
- **norm_cfg** (*dict[str]*) –Config of normalization layer.
- **in_channels** (*int*) –The number of encoder input channels.
- **block_type** (*str, optional*) –Type of the block to use. Defaults to 'conv_module'.
- **conv_cfg** (*dict, optional*) –Config of conv layer. Defaults to dict(type='SubMConv3d').

返回 The number of encoder output channels.

返回类型 int

class mmdet3d.models.middle_encoders.**SparseEncoderSASSD**(*in_channels*, *sparse_shape*, *order=('conv', 'norm', 'act')*, *norm_cfg={'eps': 0.001, 'momentum': 0.01, 'type': 'BN1d'}*, *base_channels=16*, *output_channels=128*, *encoder_channels=((16), (32, 32, 32), (64, 64, 64), (64, 64, 64))*, *encoder_paddings=((1), (1, 1, 1), (1, 1, 1), ((0, 1, 1), 1, 1))*, *block_type='conv_module'*)

Sparse encoder for SASSD

参数

- **in_channels** (*int*) –The number of input channels.
- **sparse_shape** (*list[int]*) –The sparse shape of input tensor.

- **order** (*list[str], optional*) –Order of conv module. Defaults to (‘conv’, ‘norm’, ‘act’).
- **norm_cfg** (*dict, optional*) –Config of normalization layer. Defaults to dict(type=’BN1d’, eps=1e-3, momentum=0.01).
- **base_channels** (*int, optional*) –Out channels for conv_input layer. Defaults to 16.
- **output_channels** (*int, optional*) –Out channels for conv_out layer. Defaults to 128.
- **encoder_channels** (*tuple[tuple[int]], optional*) –Convolutional channels of each encode block. Defaults to ((16,), (32, 32, 32), (64, 64, 64), (64, 64, 64)).
- **encoder_paddings** (*tuple[tuple[int]], optional*) –Paddings of each encode block. Defaults to ((1,), (1, 1, 1), (1, 1, 1), ((0, 1, 1), 1, 1)).
- **block_type** (*str, optional*) –Type of the block to use. Defaults to ‘conv_module’.

aux_loss(*points*, *point_cls*, *point_reg*, *gt_bboxes*)

Calculate auxiliary loss.

参数

- **points** (*torch.Tensor*) –Mean feature value of the points.
- **point_cls** (*torch.Tensor*) –Classificaion result of the points.
- **point_reg** (*torch.Tensor*) –Regression offsets of the points.
- **gt_bboxes** (list[BaseInstance3DBoxes]) –Ground truth boxes for each sample.

返回 Backbone features.

返回类型 dict

calculate_pts_offsets(*points*, *boxes*)

Find all boxes in which each point is, as well as the offsets from the box centers.

参数

- **points** (*torch.Tensor*) –[M, 3], [x, y, z] in LiDAR/DEPTH coordinate
- **boxes** (*torch.Tensor*) –[T, 7], num_valid_boxes <= T, [x, y, z, x_size, y_size, z_size, rz], (x, y, z) is the bottom center.

返回

Point indices of boxes with the shape of (T, M). Default background = 0. And offsets from the box centers of points, if it belows to the box, with the shape of (M, 3). Default background = 0.

返回类型 tuple[torch.Tensor]

forward(*voxel_features*, *coors*, *batch_size*, *test_mode=False*)

Forward of SparseEncoder.

参数

- **voxel_features** (*torch.Tensor*) –Voxel features in shape (N, C).
- **coors** (*torch.Tensor*) –Coordinates in shape (N, 4), the columns in the order of (batch_idx, z_idx, y_idx, x_idx).
- **batch_size** (*int*) –Batch size.
- **test_mode** (*bool, optional*) –Whether in test mode. Defaults to False.

返回

Backbone features. tuple[torch.Tensor]: Mean feature value of the points,

Classificaion result of the points, Regression offsets of the points.

返回类型 dict

get_auxiliary_targets(*nxyz*, *gt_boxes3d*, *enlarge=1.0*)

Get auxiliary target.

参数

- **nxyz** (*torch.Tensor*) –Mean features of the points.
- **gt_boxes3d** (*torch.Tensor*) –Coordinates in shape (N, 4), the columns in the order of (batch_idx, z_idx, y_idx, x_idx).
- **enlarge** (*int, optional*) –Enlaged scale. Defaults to 1.0.

返回

Label of the points and center offsets of the points.

返回类型 tuple[torch.Tensor]

make_auxiliary_points(*source_tensor*, *target*, *offset=(0.0, - 40.0, - 3.0)*, *voxel_size=(0.05, 0.05, 0.1)*)

Make auxiliary points for loss computation.

参数

- **source_tensor** (*torch.Tensor*) –(M, C) features to be propigated.
- **target** (*torch.Tensor*) –(N, 4) bxyz positions of the target features.
- **offset** (*tuple[float], optional*) –Voxelization offset. Defaults to (0., -40., -3.)
- **voxel_size** (*tuple[float], optional*) –Voxelization size. Defaults to (.05, .05, .1)

返回 (N, C) tensor of the features of the target features.

返回类型 torch.Tensor

class mmdet3d.models.middle_encoders.**SparseUNet**(*in_channels*, *sparse_shape*, *order=('conv', 'norm', 'act')*, *norm_cfg={'eps': 0.001, 'momentum': 0.01, 'type': 'BN1d'}*, *base_channels=16*, *output_channels=128*, *encoder_channels=((16), (32, 32, 32), (64, 64, 64), (64, 64, 64))*, *encoder_paddings=((1), (1, 1, 1), (1, 1, 1), ((0, 1, 1), 1, 1))*, *decoder_channels=((64, 64, 64), (64, 64, 32), (32, 32, 16), (16, 16, 16))*, *decoder_paddings=((1, 0), (1, 0), (0, 0), (0, 1))*, *init_cfg=None*)

SparseUNet for PartA^2.

See the paper for more details.

参数

- **in_channels** (*int*) –The number of input channels.
- **sparse_shape** (*list[int]*) –The sparse shape of input tensor.
- **norm_cfg** (*dict*) –Config of normalization layer.
- **base_channels** (*int*) –Out channels for conv_input layer.
- **output_channels** (*int*) –Out channels for conv_out layer.
- **encoder_channels** (*tuple[tuple[int]]*) –Convolutional channels of each encode block.
- **encoder_paddings** (*tuple[tuple[int]]*) –Paddings of each encode block.
- **decoder_channels** (*tuple[tuple[int]]*) –Convolutional channels of each decode block.
- **decoder_paddings** (*tuple[tuple[int]]*) –Paddings of each decode block.

decoder_layer_forward(*x_lateral*, *x_bottom*, *lateral_layer*, *merge_layer*, *upsample_layer*)

Forward of upsample and residual block.

参数

- **x_lateral** (SparseConvTensor) –Lateral tensor.
- **x_bottom** (SparseConvTensor) –Feature from bottom layer.
- **lateral_layer** (*SparseBasicBlock*) –Convolution for lateral tensor.
- **merge_layer** (*SparseSequential*) –Convolution for merging features.

- **upsample_layer** (*SparseSequential*) –Convolution for upsampling.

返回 Upsampled feature.

返回类型 SparseConvTensor

forward(*voxel_features*, *coors*, *batch_size*)

Forward of SparseUNet.

参数

- **voxel_features** (*torch.float32*) –Voxel features in shape [N, C].
- **coors** (*torch.int32*) –Coordinates in shape [N, 4], the columns in the order of (batch_idx, z_idx, y_idx, x_idx).
- **batch_size** (*int*) –Batch size.

返回 Backbone features.

返回类型 dict[str, torch.Tensor]

make_decoder_layers(*make_block*, *norm_cfg*, *in_channels*)

make decoder layers using sparse convs.

参数

- **make_block** (*method*) –A bounded function to build blocks.
- **norm_cfg** (*dict[str]*) –Config of normalization layer.
- **in_channels** (*int*) –The number of encoder input channels.

返回 The number of encoder output channels.

返回类型 int

make_encoder_layers(*make_block*, *norm_cfg*, *in_channels*)

make encoder layers using sparse convs.

参数

- **make_block** (*method*) –A bounded function to build blocks.
- **norm_cfg** (*dict[str]*) –Config of normalization layer.
- **in_channels** (*int*) –The number of encoder input channels.

返回 The number of encoder output channels.

返回类型 int

static reduce_channel(*x*, *out_channels*)

reduce channel for element-wise addition.

参数

- **x** (SparseConvTensor) –Sparse tensor, x.features are in shape (N, C1).

- **out_channels** (*int*) –The number of channel after reduction.

返回 Channel reduced feature.

返回类型 SparseConvTensor

40.9 model_utils

class mmdet3d.models.model_utils.**EdgeFusionModule**(*out_channels*, *feat_channels*, *kernel_size=3*, *act_cfg={'type': 'ReLU'}*, *norm_cfg={'type': 'BN1d'}*)

Edge Fusion Module for feature map.

参数

- **out_channels** (*int*) –The number of output channels.
- **feat_channels** (*int*) –The number of channels in feature map during edge feature fusion.
- **kernel_size** (*int, optional*) –Kernel size of convolution. Default: 3.
- **act_cfg** (*dict, optional*) –Config of activation. Default: dict(type=' ReLU').
- **norm_cfg** (*dict, optional*) –Config of normalization. Default: dict(type=' BN1d')).

forward(*features*, *fused_features*, *edge_indices*, *edge_lens*, *output_h*, *output_w*)

Forward pass.

参数

- **features** (*torch.Tensor*) –Different representative features for fusion.
- **fused_features** (*torch.Tensor*) –Different representative features to be fused.
- **edge_indices** (*torch.Tensor*) –Batch image edge indices.
- **edge_lens** (*list[int]*) –List of edge length of each image.
- **output_h** (*int*) –Height of output feature map.
- **output_w** (*int*) –Width of output feature map.

返回 Fused feature maps.

返回类型 torch.Tensor

class mmdet3d.models.model_utils.**GroupFree3DMHA**(*embed_dims*, *num_heads*, *attn_drop=0.0*, *proj_drop=0.0*, *dropout_layer={'drop_prob': 0.0, 'type': 'DropOut'}*, *init_cfg=None*, *batch_first=False*, ***kwargs*)

A warpper for torch.nn.MultiheadAttention for GroupFree3D.

This module implements MultiheadAttention with identity connection, and positional encoding used in DETR is also passed as input.

参数

- **embed_dims** (*int*) –The embedding dimension.
- **num_heads** (*int*) –Parallel attention heads. Same as *nn.MultiheadAttention*.
- **attn_drop** (*float, optional*) –A Dropout layer on attn_output_weights. Defaults to 0.0.
- **proj_drop** (*float, optional*) –A Dropout layer. Defaults to 0.0.
- **(obj** (*init_cfg*) –*ConfigDict*, optional): The dropout_layer used when adding the shortcut.
- **(obj** –*mmcv.ConfigDict*, optional): The Config for initialization. Default: None.
- **batch_first** (*bool, optional*) –Key, Query and Value are shape of (batch, n, embed_dim) or (n, batch, embed_dim). Defaults to False.

forward(*query*, *key*, *value*, *identity*, *query_pos=None*, *key_pos=None*, *attn_mask=None*, *key_padding_mask=None*, ***kwargs*)

Forward function for *GroupFree3DMHA*.

**kwargs allow passing a more general data flow when combining with other operations in *transformerlayer*.

参数

- **query** (*Tensor*) –The input query with shape [num_queries, bs, embed_dims]. Same in *nn.MultiheadAttention.forward*.
- **key** (*Tensor*) –The key tensor with shape [num_keys, bs, embed_dims]. Same in *nn.MultiheadAttention.forward*. If None, the `query` will be used.
- **value** (*Tensor*) –The value tensor with same shape as *key*. Same in *nn.MultiheadAttention.forward*. If None, the *key* will be used.
- **identity** (*Tensor*) –This tensor, with the same shape as x, will be used for the identity link. If None, *x* will be used.
- **query_pos** (*Tensor, optional*) –The positional encoding for query, with the same shape as *x*. Defaults to None. If not None, it will be added to *x* before forward function.

- **key_pos** (*Tensor, optional*) –The positional encoding for *key*, with the same shape as *key*. Defaults to None. If not None, it will be added to *key* before forward function. If None, and *query_pos* has the same shape as *key*, then *query_pos* will be used for *key_pos*. Defaults to None.
- **attn_mask** (*Tensor, optional*) –ByteTensor mask with shape [num_queries, num_keys]. Same in *nn.MultiheadAttention.forward*. Defaults to None.
- **key_padding_mask** (*Tensor, optional*) –ByteTensor with shape [bs, num_keys]. Same in *nn.MultiheadAttention.forward*. Defaults to None.

返回 forwarded results with shape [num_queries, bs, embed_dims].

返回类型 Tensor

class mmdet3d.models.model_utils.**VoteModule**(*in_channels*, *vote_per_seed=1*, *gt_per_seed=3*, *num_points=- 1*, *conv_channels=(16, 16)*, *conv_cfg={'type': 'Conv1d'}*, *norm_cfg={'type': 'BN1d'}*, *act_cfg={'type': 'ReLU'}*, *norm_feats=True*, *with_res_feat=True*, *vote_xyz_range=None*, *vote_loss=None*)

Vote module.

Generate votes from seed point features.

参数

- **in_channels** (*int*) –Number of channels of seed point features.
- **vote_per_seed** (*int, optional*) –Number of votes generated from each seed point. Default: 1.
- **gt_per_seed** (*int, optional*) –Number of ground truth votes generated from each seed point. Default: 3.
- **num_points** (*int, optional*) –Number of points to be used for voting. Default: 1.
- **conv_channels** (*tuple[int], optional*) –Out channels of vote generating convolution. Default: (16, 16).
- **conv_cfg** (*dict, optional*) –Config of convolution. Default: dict(type=' Conv1d').
- **norm_cfg** (*dict, optional*) –Config of normalization. Default: dict(type=' BN1d').
- **norm_feats** (*bool, optional*) –Whether to normalize features. Default: True.
- **with_res_feat** (*bool, optional*) –Whether to predict residual features. Default: True.

- **vote_xyz_range** (*list[float], optional*) –The range of points translation. Default: None.
- **vote_loss** (*dict, optional*) –Config of vote loss. Default: None.

forward(*seed_points*, *seed_feats*)

forward.

参数

- **seed_points** (*torch.Tensor*) –Coordinate of the seed points in shape (B, N, 3).
- **seed_feats** (*torch.Tensor*) –Features of the seed points in shape (B, C, N).

返回

- **vote_points: Voted xyz based on the seed points** with shape (B, M, 3), M=num_seed*vote_per_seed.
- **vote_features: Voted features based on the seed points with** shape (B, C, M) where M=num_seed*vote_per_seed, C=vote_feature_dim.

返回类型 tuple[torch.Tensor]

get_loss(*seed_points*, *vote_points*, *seed_indices*, *vote_targets_mask*, *vote_targets*)

Calculate loss of voting module.

参数

- **seed_points** (*torch.Tensor*) –Coordinate of the seed points.
- **vote_points** (*torch.Tensor*) –Coordinate of the vote points.
- **seed_indices** (*torch.Tensor*) –Indices of seed points in raw points.
- **vote_targets_mask** (*torch.Tensor*) –Mask of valid vote targets.
- **vote_targets** (*torch.Tensor*) –Targets of votes.

返回 Weighted vote loss.

返回类型 torch.Tensor

CHAPTER 41

English

CHAPTER 42

简体中文

CHAPTER 43

Indices and tables

- genindex
- search

Python 模块索引

m

索引

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

模块