MMSegmentation

发布 *0.29.0*

MMSegmentation Authors

2022 年 10 月 24 日

开始你的第一步

1 依赖 1

2 安装 3
- 2.1 最佳实践 . . . 3
- 2.2 验证安装 . . . 4
- 2.3 自定义安装 . . . 5
- 2.4 故障解决 . . . 7

3 准备数据集 9
- 3.1 Cityscapes . . . 12
- 3.2 Pascal VOC . . . 12
- 3.3 ADE20K . . . 13
- 3.4 Pascal Context . . . 13
- 3.5 CHASE DB1 . . . 13
- 3.6 DRIVE . . . 13
- 3.7 HRF . . . 14
- 3.8 STARE . . . 14
- 3.9 Dark Zurich . . . 14
- 3.10 Nighttime Driving . . . 14
- 3.11 LoveDA . . . 14
- 3.12 ISPRS Potsdam . . . 15
- 3.13 ISPRS Vaihingen . . . 15
- 3.14 iSAID . . . 16

4 标准与模型库 17
- 4.1 共同设定 . . . 17
- 4.2 基线 . . . 18
- 4.3 速度标定 . . . 21

5 模型库统计数据 **23**

6 训练一个模型 **27**
6.1 使用单台机器训练 28
6.2 使用多台机器训练 29
6.3 使用 slurm 管理任务 30

7 使用预训练模型推理 **33**
7.1 测试一个数据集 33

8 教程 1: 学习配置文件 **37**
8.1 配置文件的结构 37
8.2 配置文件命名风格 38
8.3 PSPNet 的一个例子 38
8.4 FAQ 44

9 教程 2: 自定义数据集 **47**
9.1 通过重新组织数据来定制数据集 47
9.2 通过混合数据去定制数据集 48

10 教程 3: 自定义数据流程 **53**
10.1 数据流程的设计 53
10.2 拓展和使用自定义的流程 56

11 教程 4: 自定义模型 **57**
11.1 自定义优化器 (optimizer) 57
11.2 定制优化器的构造器 (optimizer constructor) 58
11.3 开发和增加新的组件（Module） 58

12 教程 5: 训练技巧 **63**
12.1 主干网络和解码头组件使用不同的学习率 (Learning Rate, LR) 63
12.2 在线难样本挖掘 (Online Hard Example Mining, OHEM) 64
12.3 类别平衡损失 (Class Balanced Loss) 64
12.4 同时使用多种损失函数 (Multiple Losses) 64
12.5 在损失函数中忽略特定的 label 类别 65

13 教程 6: 自定义运行设定 **67**
13.1 自定义优化设定 67
13.2 自定义训练计划表 70
13.3 自定义工作流 (workflow) 70
13.4 自定义钩 (hooks) 71

14 常用工具 **73**
14.1 计算参数量（params）和计算量（FLOPs）(试验性) 73
14.2 发布模型 74

14.3 导出 ONNX (试验性) . . . 74
14.4 评估 ONNX 模型 . . . 75
14.5 导出 TorchScript (试验性) . . . 76
14.6 导出 TensorRT (试验性) . . . 77

15 其他内容 **79**
15.1 打印完整的配置文件 . . . 79
15.2 对训练日志 (training logs) 画图 . . . 79
15.3 转换其他仓库的权重 . . . 80

16 模型服务 **81**
16.1 1. 将 model 从 MMSegmentation 转换到 TorchServe . . . 81
16.2 2. 构建 `mmseg-serve` 容器镜像 (docker image) . . . 81
16.3 3. 运行 `mmseg-serve` . . . 82
16.4 4. 测试部署 . . . 82

17 模型集成 **85**
17.1 使用方法 . . . 85
17.2 各个参数的描述: . . . 85
17.3 模型集成结果 . . . 86

18 常见问题解答（FAQ） **87**
18.1 安装 . . . 87
18.2 如何获知模型训练时需要的显卡数量 . . . 88
18.3 auxiliary head 是什么 . . . 88
18.4 为什么日志文件没有被创建 . . . 88
18.5 运行测试脚本时如何输出绘制分割掩膜的图像 . . . 88

19 English **89**

20 简体中文 **91**

21 mmseg.apis **93**

22 mmseg.core **97**
22.1 seg . . . 97
22.2 evaluation . . . 98
22.3 utils . . . 101

23 mmseg.datasets **103**
23.1 datasets . . . 103
23.2 pipelines . . . 114

24 mmseg.models **125**
24.1 segmentors . . . 125

24.2 backbones 129
24.3 decode_heads 160
24.4 losses 175

25 Indices and tables **185**

Python 模块索引 **187**

索引 **189**

CHAPTER 1

依赖

在本节中，我们将演示如何用 PyTorch 准备一个环境。

MMSegmentation 可以在 Linux、Windows 和 MacOS 上运行。它需要 Python 3.6 以上，CUDA 9.2 以上和 PyTorch 1.3 以上。

注解： 如果您对 PyTorch 有经验并且已经安装了它，请跳到下一节。否则，您可以按照以下步骤进行准备。

第一步从官方网站下载并安装 Miniconda。

第二步创建并激活一个 conda 环境。

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

第三步按照官方说明安装 PyTorch。

在 GPU 平台上：

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

在 CPU 平台上：

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

CHAPTER 2

安装

我们建议用户遵循我们的最佳实践来安装 MMSegmentation，同时整个过程是高度可定制的。更多信息见自定义安装部分。

2.1 最佳实践

第一步使用 MIM 安装 MMCV

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

第二步安装 MMSegmentation

根据具体需求，我们支持两种安装模式：

- 从源码安装（推荐）：如果基于 MMSegmentation 框架开发自己的任务，需要添加新的功能，比如新的模型或是数据集，或者使用我们提供的各种工具。
- *作为 Python 包安装*：只是希望调用 MMSegmentation 的接口，或者在自己的项目中导入 MMSegmentation 中的模块。

2.1.1 从源码安装

```
git clone https://github.com/open-mmlab/mmsegmentation.git
cd mmsegmentation
pip install -v -e .
# "-v " 指详细说明，或更多的输出
# "-e" 表示在可编辑模式下安装项目，因此对代码所做的任何本地修改都会生效，从而无需重新安装。
```

2.1.2 作为 Python 包安装

```
pip install mmsegmentation
```

2.2 验证安装

为了验证 MMSegmentation 是否安装正确，我们提供了一些示例代码来执行模型推理。

第一步我们需要下载配置文件和模型权重文件。

```
mim download mmsegmentation --config pspnet_r50-d8_512x1024_40k_cityscapes --dest .
```

下载将需要几秒钟或更长时间，这取决于你的网络环境。完成后，你会在当前文件夹中发现两个文件 `pspnet_r50-d8_512x1024_40k_cityscapes.py` 和 `pspnet_r50-d8_512x1024_40k_cityscapes_20200605_003338-2966598c.pth`。

第二步验证推理示例

如果您是**从源码安装**的 MMSegmentation，那么直接运行以下命令进行验证：

```
python demo/image_demo.py demo/demo.png pspnet_r50-d8_512x1024_40k_cityscapes.py␣
↪pspnet_r50-d8_512x1024_40k_cityscapes_20200605_003338-2966598c.pth --device cpu --
↪out-file result.jpg
```

你会在你的当前文件夹中看到一个新的图像 `result.jpg`，其中的分割掩膜覆盖在所有对象上。

如果您是**作为 PyThon 包安装**，那么可以打开您的 Python 解释器，复制并粘贴如下代码：

```
from mmseg.apis import inference_segmentor, init_segmentor
import mmcv

config_file = 'pspnet_r50-d8_512x1024_40k_cityscapes.py'
checkpoint_file = 'pspnet_r50-d8_512x1024_40k_cityscapes_20200605_003338-2966598c.pth'

# 通过配置文件和模型权重文件构建模型
```

(下页继续)

(续上页)

```
model = init_segmentor(config_file, checkpoint_file, device='cuda:0')

# 对单张图片进行推理并展示结果
img = 'test.jpg'  # or img = mmcv.imread(img), which will only load it once
result = inference_segmentor(model, img)
# 在新窗口中可视化推理结果
model.show_result(img, result, show=True)
# 或将可视化结果存储在文件中
# 你可以修改 opacity 在 (0,1] 之间的取值来改变绘制好的分割图的透明度
model.show_result(img, result, out_file='result.jpg', opacity=0.5)

# 对视频进行推理并展示结果
video = mmcv.VideoReader('video.mp4')
for frame in video:
   result = inference_segmentor(model, frame)
   model.show_result(frame, result, wait_time=1)
```

你可以修改上面的代码来测试一张图片或一段视频，这两种方式都可以验证安装是否成功。

2.3 自定义安装

2.3.1 CUDA 版本

在安装 PyTorch 时，您需要指定 CUDA 的版本。如果您不清楚应该选择哪一个，请遵循我们的建议。

- 对于 Ampere 架构的 NVIDIA GPU，例如 GeForce 30 系列以及 NVIDIA A100，CUDA 11 是必需的。
- 对于更早的 NVIDIA GPU，CUDA 11 是向后兼容 (backward compatible) 的，但 CUDA 10.2 能够提供更好的兼容性，也更加轻量。

请确保您的 GPU 驱动版本满足最低的版本需求，参阅这张表。

注解： 如果按照我们的最佳实践进行安装，CUDA 运行时库就足够了，因为我们提供相关 CUDA 代码的预编译，您不需要进行本地编译。但如果您希望从源码进行 MMCV 的编译，或是进行其他 CUDA 算子的开发，那么就必须安装完整的 CUDA 工具链，参见 NVIDIA 官网，另外还需要确保该 CUDA 工具链的版本与 PyTorch 安装时的配置相匹配（如用 `conda install` 安装 PyTorch 时指定的 cudatoolkit 版本）。

2.3.2 不使用 MIM 安装 MMCV

MMCV 包含 C++ 和 CUDA 扩展，因此其对 PyTorch 的依赖比较复杂。MIM 会自动解析这些依赖，选择合适的 MMCV 预编译包，使安装更简单，但它并不是必需的。

要使用 pip 而不是 MIM 来安装 MMCV，请遵照 MMCV 安装指南。它需要您用指定 url 的形式手动指定对应的 PyTorch 和 CUDA 版本。

举个例子，如下命令将会安装基于 PyTorch 1.10.x 和 CUDA 11.3 编译的 mmcv-full。

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

2.3.3 在 CPU 环境中安装

MMPose 可以仅在 CPU 环境中安装，在 CPU 模式下，您可以完成训练（需要 MMCV 版本 >= 1.4.4）、测试和模型推理等所有操作。

2.3.4 在 Google Colab 中安装

Google Colab 通常已经包含了 PyTorch 环境，因此我们只需要安装 MMCV 和 MMPose 即可，命令如下：

第一步使用 MIM 安装 MMCV

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

第二步从源码安装 MMSegmentation

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

第三步验证

```
import mmseg
print(mmseg.__version__)
# 预期输出：0.24.1 或其他版本号
```

注解： 在 Jupyter 中，感叹号 ! 用于执行外部命令，而 %cd 是一个魔术命令，用于切换 Python 的工作路径。

2.3.5 通过 Docker 使用 MMSegmentation

我们提供了一个Dockerfile来构建一个镜像。请确保你的docker 版本 >=19.03。

```
# build an image with PyTorch 1.11, CUDA 11.3
# If you prefer other versions, just modified the Dockerfile
docker build -t mmsegmentation docker/
```

用以下命令运行 Docker 镜像:

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

2.4 故障解决

如果你在安装过程中遇到一些问题，请先查看*FAQ*页面。

如果没有找到解决方案，你也可以在 GitHub 上打开一个问题。

CHAPTER 3

准备数据集

推荐用软链接，将数据集根目录链接到 `$MMSEGMENTATION/data` 里。如果您的文件夹结构是不同的，您也许可以试着修改配置文件里对应的路径。

```
mmsegmentation
├── mmseg
├── tools
├── configs
├── data
│   ├── cityscapes
│   │   ├── leftImg8bit
│   │   │   ├── train
│   │   │   ├── val
│   │   ├── gtFine
│   │   │   ├── train
│   │   │   ├── val
│   ├── VOCdevkit
│   │   ├── VOC2012
│   │   │   ├── JPEGImages
│   │   │   ├── SegmentationClass
│   │   │   ├── ImageSets
│   │   │   │   ├── Segmentation
│   │   ├── VOC2010
│   │   │   ├── JPEGImages
│   │   │   ├── SegmentationClassContext
```

(下页继续)

(续上页)

```
│   │   │   ├── ImageSets
│   │   │   │   ├── SegmentationContext
│   │   │   │   │   ├── train.txt
│   │   │   │   │   ├── val.txt
│   │   │   ├── trainval_merged.json
│   │   ├── VOCaug
│   │   │   ├── dataset
│   │   │   │   ├── cls
│   ├── ade
│   │   ├── ADEChallengeData2016
│   │   │   ├── annotations
│   │   │   │   ├── training
│   │   │   │   ├── validation
│   │   │   ├── images
│   │   │   │   ├── training
│   │   │   │   ├── validation
│   ├── CHASE_DB1
│   │   ├── images
│   │   │   ├── training
│   │   │   ├── validation
│   │   ├── annotations
│   │   │   ├── training
│   │   │   ├── validation
│   ├── DRIVE
│   │   ├── images
│   │   │   ├── training
│   │   │   ├── validation
│   │   ├── annotations
│   │   │   ├── training
│   │   │   ├── validation
│   ├── HRF
│   │   ├── images
│   │   │   ├── training
│   │   │   ├── validation
│   │   ├── annotations
│   │   │   ├── training
│   │   │   ├── validation
│   ├── STARE
│   │   ├── images
│   │   │   ├── training
│   │   │   ├── validation
│   │   ├── annotations
│   │   │   ├── training
```

(下页继续)

(续上页)

```
│   │   │   ├── validation
│   ├── dark_zurich
│   │   ├── gps
│   │   │   ├── val
│   │   │   └── val_ref
│   │   ├── gt
│   │   │   └── val
│   │   ├── LICENSE.txt
│   │   ├── lists_file_names
│   │   │   ├── val_filenames.txt
│   │   │   └── val_ref_filenames.txt
│   │   ├── README.md
│   │   └── rgb_anon
│   │   |   ├── val
│   │   |   └── val_ref
│   ├── NighttimeDrivingTest
|   |   ├── gtCoarse_daytime_trainvaltest
|   |   │   └── test
|   |   │       └── night
|   |   └── leftImg8bit
|   |   |   └── test
|   |   |       └── night
│   ├── loveDA
│   │   ├── img_dir
│   │   │   ├── train
│   │   │   ├── val
│   │   │   ├── test
│   │   ├── ann_dir
│   │   │   ├── train
│   │   │   ├── val
│   ├── potsdam
│   │   ├── img_dir
│   │   │   ├── train
│   │   │   ├── val
│   │   ├── ann_dir
│   │   │   ├── train
│   │   │   ├── val
│   ├── vaihingen
│   │   ├── img_dir
│   │   │   ├── train
│   │   │   ├── val
│   │   ├── ann_dir
│   │   │   ├── train
```

(下页继续)

(续上页)

```
│   │   │   ├── val
│   ├── iSAID
│   │   ├── img_dir
│   │   │   ├── train
│   │   │   ├── val
│   │   │   ├── test
│   │   ├── ann_dir
│   │   │   ├── train
│   │   │   ├── val
```

3.1 Cityscapes

注册成功后，数据集可以在 这里 下载。

通常情况下，`**labelTrainIds.png` 被用来训练 cityscapes。基于 cityscapesscripts, 我们提供了一个 脚本, 去生成 `**labelTrainIds.png`。

```
# --nproc 8 意味着有 8 个进程用来转换，它也可以被忽略。
python tools/convert_datasets/cityscapes.py data/cityscapes --nproc 8
```

3.2 Pascal VOC

Pascal VOC 2012 可以在 这里 下载。此外，许多最近在 Pascal VOC 数据集上的工作都会利用增广的数据，它们可以在 这里 找到。

如果您想使用增广后的 VOC 数据集，请运行下面的命令来将数据增广的标注转成正确的格式。

```
# --nproc 8 意味着有 8 个进程用来转换，它也可以被忽略。
python tools/convert_datasets/voc_aug.py data/VOCdevkit data/VOCdevkit/VOCaug --nproc␣
↪8
```

关于如何拼接数据集 (concatenate) 并一起训练它们，更多细节请参考 拼接连接数据集 。

3.3 ADE20K

ADE20K 的训练集和验证集可以在 这里 下载。您还可以在 这里 下载验证集。

3.4 Pascal Context

Pascal Context 的训练集和验证集可以在 这里 下载。注册成功后，您还可以在 这里 下载验证集。

为了从原始数据集里切分训练集和验证集，您可以在 这里 下载 trainval_merged.json。

如果您想使用 Pascal Context 数据集，请安装 细节 然后再运行如下命令来把标注转换成正确的格式。

```
python tools/convert_datasets/pascal_context.py data/VOCdevkit data/VOCdevkit/VOC2010/
↪trainval_merged.json
```

3.5 CHASE DB1

CHASE DB1 的训练集和验证集可以在 这里 下载。

为了将 CHASE DB1 数据集转换成 MMSegmentation 的格式，您需要运行如下命令：

```
python tools/convert_datasets/chase_db1.py /path/to/CHASEDB1.zip
```

这个脚本将自动生成正确的文件夹结构。

3.6 DRIVE

DRIVE 的训练集和验证集可以在 这里 下载。在此之前，您需要注册一个账号，当前 '1st_manual' 并未被官方提供，因此需要您从其他地方获取。

为了将 DRIVE 数据集转换成 MMSegmentation 格式，您需要运行如下命令：

```
python tools/convert_datasets/drive.py /path/to/training.zip /path/to/test.zip
```

这个脚本将自动生成正确的文件夹结构。

3.7 HRF

首先，下载 healthy.zip glaucoma.zip, diabetic_retinopathy.zip, healthy_manualsegm.zip, glaucoma_manualsegm.zip 以及 diabetic_retinopathy_manualsegm.zip 。

为了将 HRF 数据集转换成 MMSegmentation 格式，您需要运行如下命令：

```
python tools/convert_datasets/hrf.py /path/to/healthy.zip /path/to/healthy_manualsegm.
↪zip /path/to/glaucoma.zip /path/to/glaucoma_manualsegm.zip /path/to/diabetic_
↪retinopathy.zip /path/to/diabetic_retinopathy_manualsegm.zip
```

这个脚本将自动生成正确的文件夹结构。

3.8 STARE

首先，下载 stare-images.tar, labels-ah.tar 和 labels-vk.tar 。

为了将 STARE 数据集转换成 MMSegmentation 格式，您需要运行如下命令：

```
python tools/convert_datasets/stare.py /path/to/stare-images.tar /path/to/labels-ah.
↪tar /path/to/labels-vk.tar
```

这个脚本将自动生成正确的文件夹结构。

3.9 Dark Zurich

因为我们只支持在此数据集上测试模型，所以您只需下载验证集 。

3.10 Nighttime Driving

因为我们只支持在此数据集上测试模型，所以您只需下载测试集 。

3.11 LoveDA

可以从 Google Drive 里下载 LoveDA 数据集 。

或者它还可以从 zenodo 下载, 您需要运行如下命令:

```
# Download Train.zip
wget https://zenodo.org/record/5706578/files/Train.zip
# Download Val.zip
wget https://zenodo.org/record/5706578/files/Val.zip
# Download Test.zip
wget https://zenodo.org/record/5706578/files/Test.zip
```

对于 LoveDA 数据集，请运行以下命令下载并重新组织数据集

```
python tools/convert_datasets/loveda.py /path/to/loveDA
```

请参照 这里 来使用训练好的模型去预测 LoveDA 测试集并且提交到官网。

关于 LoveDA 的更多细节可以在这里 找到。

3.12 ISPRS Potsdam

Potsdam 数据集是一个有着 2D 语义分割内容标注的城市遥感数据集。数据集可以从挑战主页 获得。需要其中的‘2_Ortho_RGB.zip’和‘5_Labels_all_noBoundary.zip’。

对于 Potsdam 数据集，请运行以下命令下载并重新组织数据集

```
python tools/convert_datasets/potsdam.py /path/to/potsdam
```

使用我们默认的配置，将生成 3456 张图片的训练集和 2016 张图片的验证集。

3.13 ISPRS Vaihingen

Vaihingen 数据集是一个有着 2D 语义分割内容标注的城市遥感数据集。

数据集可以从挑战 主页. 需要其中的‘ISPRS_semantic_labeling_Vaihingen.zip’和‘ISPRS_semantic_labeling_Vaihingen_ground_truth_eroded_COMPLETE.zip’。

对于 Vaihingen 数据集，请运行以下命令下载并重新组织数据集

```
python tools/convert_datasets/vaihingen.py /path/to/vaihingen
```

使用我们默认的配置 (`clip_size`=512, `stride_size`=256)，将生成 344 张图片的训练集和 398 张图片的验证集。

3.14 iSAID

iSAID 数据集 (训练集/验证集/测试集) 的图像可以从 DOTA-v1.0 下载.

iSAID 数据集 (训练集/验证集) 的注释可以从 iSAID 下载.

该数据集是一个大规模的实例分割 (也可以用于语义分割) 的遥感数据集.

下载后，在数据集转换前，您需要将数据集文件夹调整成如下格式.

```
│   ├── iSAID
│   │   ├── train
│   │   │   ├── images
│   │   │   │   ├── part1.zip
│   │   │   │   ├── part2.zip
│   │   │   │   ├── part3.zip
│   │   │   ├── Semantic_masks
│   │   │   │   ├── images.zip
│   │   ├── val
│   │   │   ├── images
│   │   │   │   ├── part1.zip
│   │   │   ├── Semantic_masks
│   │   │   │   ├── images.zip
│   │   ├── test
│   │   │   ├── images
│   │   │   │   ├── part1.zip
│   │   │   │   ├── part2.zip
```

```
python tools/convert_datasets/isaid.py /path/to/iSAID
```

使用我们默认的配置 (`patch_width`=896, `patch_height`=896, `overlap_area`=384)，将生成 33978 张图片的训练集和 11644 张图片的验证集。

CHAPTER 4

标准与模型库

4.1 共同设定

- 我们默认使用 4 卡分布式训练
- 所有 PyTorch 风格的 ImageNet 预训练网络由我们自己训练，和 论文 保持一致。我们的 ResNet 网络是基于 ResNetV1c 的变种，在这里输入层的 7x7 卷积被 3 个 3x3 取代
- 为了在不同的硬件上保持一致，我们以 `torch.cuda.max_memory_allocated()` 的最大值作为 GPU 占用率，同时设置 `torch.backends.cudnn.benchmark=False`。注意，这通常比 `nvidia-smi` 显示的要少
- 我们以网络 forward 和后处理的时间加和作为推理时间，除去数据加载时间。我们使用脚本 `tools/benchmark.py` 来获取推理时间，它在 `torch.backends.cudnn.benchmark=False` 的设定下，计算 200 张图片的平均推理时间
- 在框架中，有两种推理模式
  - `slide` 模式（滑动模式）：测试的配置文件字段 `test_cfg` 会是 `dict(mode='slide', crop_size=(769, 769), stride=(513, 513))`. 在这个模式下，从原图中裁剪多个小图分别输入网络中进行推理。小图的大小和小图之间的距离由 `crop_size` 和 `stride` 决定，重合区域会进行平均
  - `whole` 模式（全图模式）：测试的配置文件字段 `test_cfg` 会是 `dict(mode='whole')`. 在这个模式下，全图会被直接输入到网络中进行推理。对于 769x769 下训练的模型，我们默认使用 `slide` 进行推理，其余模型用 `whole` 进行推理

- 对于输入大小为 8x+1（比如 769），我们使用 `align_corners=True`。其余情况，对于输入大小为 8x (比如 512，1024)，我们使用 `align_corners=False`

4.2 基线

4.2.1 FCN

请参考 FCN 获得详细信息。

4.2.2 PSPNet

请参考 PSPNet 获得详细信息。

4.2.3 DeepLabV3

请参考 DeepLabV3 获得详细信息。

4.2.4 PSANet

请参考 PSANet 获得详细信息。

4.2.5 DeepLabV3+

请参考 DeepLabV3+ 获得详细信息。

4.2.6 UPerNet

请参考 UPerNet 获得详细信息。

4.2.7 NonLocal Net

请参考 NonLocal Net 获得详细信息。

4.2.8 EncNet

请参考 EncNet 获得详细信息。

4.2.9 CCNet

请参考 CCNet 获得详细信息。

4.2.10 DANet

请参考 DANet 获得详细信息。

4.2.11 APCNet

请参考 APCNet 获得详细信息。

4.2.12 HRNet

请参考 HRNet 获得详细信息。

4.2.13 GCNet

请参考 GCNet 获得详细信息。

4.2.14 DMNet

请参考 DMNet 获得详细信息。

4.2.15 ANN

请参考 ANN 获得详细信息。

4.2.16 OCRNet

请参考 OCRNet 获得详细信息。

4.2.17 Fast-SCNN

请参考 Fast-SCNN 获得详细信息。

4.2.18 ResNeSt

请参考 ResNeSt 获得详细信息。

4.2.19 Semantic FPN

请参考 Semantic FPN 获得详细信息。

4.2.20 PointRend

请参考 PointRend 获得详细信息。

4.2.21 MobileNetV2

请参考 MobileNetV2 获得详细信息。

4.2.22 MobileNetV3

请参考 MobileNetV3 获得详细信息。

4.2.23 EMANet

请参考 EMANet 获得详细信息。

4.2.24 DNLNet

请参考 DNLNet 获得详细信息。

4.2.25 CGNet

请参考 CGNet 获得详细信息。

4.2.26 Mixed Precision (FP16) Training

请参考 Mixed Precision (FP16) Training 在 BiSeNetV2 训练的样例 获得详细信息。

4.3 速度标定

4.3.1 硬件

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

4.3.2 软件环境

- Python 3.7
- PyTorch 1.5
- CUDA 10.1
- CUDNN 7.6.03
- NCCL 2.4.08

4.3.3 训练速度

为了公平比较，我们全部使用 ResNet-101V1c 进行标定。输入大小为 1024x512，批量样本数为 2。

训练速度如下表，指标为每次迭代的时间，以秒为单位，越低越快。

注意：DeepLabV3+ 的输出步长为 8。

CHAPTER 5

模型库统计数据

- 论文数量: 45
  - ALGORITHM: 34
  - BACKBONE: 11
- 模型数量: 603
  - [ALGORITHM] ANN (16 ckpts)
  - [ALGORITHM] APCNet (12 ckpts)
  - [BACKBONE] BEiT (2 ckpts)
  - [ALGORITHM] BiSeNetV1 (11 ckpts)
  - [ALGORITHM] BiSeNetV2 (4 ckpts)
  - [ALGORITHM] CCNet (16 ckpts)
  - [ALGORITHM] CGNet (2 ckpts)
  - [BACKBONE] ConvNeXt (6 ckpts)
  - [ALGORITHM] DANet (16 ckpts)
  - [ALGORITHM] DeepLabV3 (41 ckpts)
  - [ALGORITHM] DeepLabV3+ (42 ckpts)
  - [ALGORITHM] DMNet (12 ckpts)
  - [ALGORITHM] DNLNet (12 ckpts)

- [ALGORITHM] DPT (1 ckpts)
- [ALGORITHM] EMANet (4 ckpts)
- [ALGORITHM] EncNet (12 ckpts)
- [ALGORITHM] ERFNet (1 ckpts)
- [ALGORITHM] FastFCN (12 ckpts)
- [ALGORITHM] Fast-SCNN (1 ckpts)
- [ALGORITHM] FCN (41 ckpts)
- [ALGORITHM] GCNet (16 ckpts)
- [BACKBONE] HRNet (37 ckpts)
- [ALGORITHM] ICNet (12 ckpts)
- [ALGORITHM] ISANet (16 ckpts)
- [ALGORITHM] K-Net (7 ckpts)
- [BACKBONE] MAE (1 ckpts)
- [BACKBONE] MobileNetV2 (8 ckpts)
- [BACKBONE] MobileNetV3 (4 ckpts)
- [ALGORITHM] NonLocal Net (16 ckpts)
- [ALGORITHM] OCRNet (24 ckpts)
- [ALGORITHM] PointRend (4 ckpts)
- [BACKBONE] PoolFormer (5 ckpts)
- [ALGORITHM] PSANet (16 ckpts)
- [ALGORITHM] PSPNet (54 ckpts)
- [BACKBONE] ResNeSt (8 ckpts)
- [ALGORITHM] SegFormer (13 ckpts)
- [ALGORITHM] Segmenter (5 ckpts)
- [ALGORITHM] Semantic FPN (4 ckpts)
- [ALGORITHM] SETR (7 ckpts)
- [ALGORITHM] STDC (4 ckpts)
- [BACKBONE] Swin Transformer (8 ckpts)
- [BACKBONE] Twins (12 ckpts)
- [ALGORITHM] UNet (25 ckpts)

- [ALGORITHM] UPerNet (22 ckpts)
- [BACKBONE] Vision Transformer (11 ckpts)

CHAPTER 6

训练一个模型

MMSegmentation 可以执行分布式训练和非分布式训练，分别使用 `MMDistributedDataParallel` 和 `MMDataParallel` 命令。

所有的输出（日志 log 和检查点 checkpoints）将被保存到工作路径文件夹里，它可以通过配置文件里的 `work_dir` 指定。

在一定迭代轮次后，我们默认在验证集上评估模型表现。您可以在训练配置文件中添加间隔参数来改变评估间隔。

```
evaluation = dict(interval=4000)  # 每 4000 iterations 评估一次模型的性能
```

*** 重要提示 ***: 在配置文件里的默认学习率是针对 4 卡 GPU 和 2 张图/GPU (此时 batchsize = 4x2 = 8) 来设置的。同样，您也可以使用 8 卡 GPU 和 1 张图/GPU 的设置，因为所有的模型均使用 cross-GPU 的 SyncBN 模式。

我们可以在训练速度和 GPU 显存之间做平衡。当模型或者 Batch Size 比较大的时，可以传递`--cfg-options model.backbone.with_cp=True` , 使用 `with_cp` 来节省显存，但是速度会更慢，因为原先使用 `with_cp` 时，是逐层反向传播 (Back Propagation, BP)，不会保存所有的梯度。

6.1 使用单台机器训练

6.1.1 使用单卡 GPU 训练

```
python tools/train.py ${CONFIG_FILE} [可选参数]
```

如果您想在命令里定义工作文件夹路径，您可以添加一个参数`--work-dir ${工作路径}`。

6.1.2 使用 CPU 训练

如果计算机没有 GPU，那么使用 CPU 训练的流程和使用单 GPU 训练的流程一致。如果计算机有 GPU 但是想使用 CPU，我们仅需要在训练流程开始前禁用 GPU。

```
export CUDA_VISIBLE_DEVICES=-1
```

之后运行单 GPU 训练脚本即可。

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

6.1.3 使用多卡 GPU 训练

```
sh tools/dist_train.sh ${CONFIG_FILE} ${GPUS} [可选参数]
```

可选参数可以为:

- `--no-validate` (**不推荐**): 训练时代码库默认会在每 k 轮迭代后在验证集上进行评估，如果不需评估使用命令 `--no-validate`
- `--work-dir ${工作路径}`: 在配置文件里重写工作路径文件夹
- `--resume-from ${检查点文件}`: 继续使用先前的检查点 (checkpoint) 文件（可以继续训练过程）
- `--load-from ${检查点文件}`: 从一个检查点 (checkpoint) 文件里加载权重（对另一个任务进行精调）
- `--deterministic`: 选择此模式会减慢训练速度，但结果易于复现

`resume-from` 和 `load-from` 的区别:

- `resume-from` 加载出模型权重和优化器状态包括迭代轮数等
- `load-from` 仅加载模型权重，从第 0 轮开始训练

示例:

```
# 模型的权重和日志将会存储在这个路径下： WORK_DIR=work_dirs/pspnet_r50-d8_512x512_80k_ade20k/
# 如果 work_dir 没有被设定，它将会被自动生成
sh tools/dist_train.sh configs/pspnet/pspnet_r50-d8_512x512_80k_ade20k.py 8 --work_
↪dir work_dirs/pspnet_r50-d8_512x512_80k_ade20k/ --deterministic
```

注意: 在训练时，模型的和日志保存在 "work_dirs/" 下的配置文件的相同文件夹结构中。不建议使用自定义的 "work_dirs/"，因为验证脚本可以从配置文件名中推断工作目录。如果你想在其他地方保存模型的权重，请使用符号链接，例如:

```
ln -s ${YOUR_WORK_DIRS} ${MMSEG}/work_dirs
```

6.1.4 在单个机器上启动多个任务

如果您在单个机器上启动多个任务，例如在 8 卡 GPU 的一个机器上有 2 个 4 卡 GPU 的训练任务，您需要特别对每个任务指定不同的端口（默认为 29500）来避免通讯冲突。否则，将会有报错信息 `RuntimeError: Address already in use`。

如果您使用命令 `dist_train.sh` 来启动一个训练任务，您可以在命令行的用环境变量 `PORT` 设置端口:

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

6.2 使用多台机器训练

如果您想使用由 ethernet 连接起来的多台机器，您可以使用以下命令:

在第一台机器上:

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

在第二台机器上:

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

但是，如果您不使用高速网路连接这几台机器的话，训练将会非常慢。

6.3 使用 slurm 管理任务

Slurm 是一个很好的计算集群作业调度系统。在由 Slurm 管理的集群中，可以使用 slurm_train.sh 来进行训练。它同时支持单节点和多节点训练。

在多台机器上训练：

```
[GPUS=${GPUS}] sh tools/slurm_train.sh ${PARTITION} ${JOB_NAME} ${CONFIG_FILE} --work-
↪dir ${WORK_DIR}
```

这里有一个在 dev 分区上使用 16 块 GPUs 来训练 PSPNet 的例子:

```
GPUS=16 sh tools/slurm_train.sh dev pspr50 configs/pspnet/pspnet_r50-d8_512x1024_40k_
↪cityscapes.py work_dirs/pspnet_r50-d8_512x1024_40k_cityscapes/
```

当使用 `slurm_train.sh` 在一个节点上启动多个任务时，需要指定不同的端口号，这里提供了三种设置:

方式 1：

在 `config1.py` 中设置:

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

在 `config2.py` 中设置:

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

然后就可以使用 config1.py 和 config2.py 启动两个作业:

```
CUDA_VISIBLE_DEVICES=0,1,2,3 GPUS=4 sh tools/slurm_train.sh ${PARTITION} ${JOB_NAME}␣
↪config1.py tmp_work_dir_1
CUDA_VISIBLE_DEVICES=4,5,6,7 GPUS=4 sh tools/slurm_train.sh ${PARTITION} ${JOB_NAME}␣
↪config2.py tmp_work_dir_2
```

方式 2:

您可以设置不同的通信端口，而不需要修改配置文件，但必须设置 "cfg-options"，以覆盖配置文件中的默认端口。

```
CUDA_VISIBLE_DEVICES=0,1,2,3 GPUS=4 sh tools/slurm_train.sh ${PARTITION} ${JOB_NAME}␣
↪config1.py tmp_work_dir_1 --cfg-options dist_params.port=29500
CUDA_VISIBLE_DEVICES=4,5,6,7 GPUS=4 sh tools/slurm_train.sh ${PARTITION} ${JOB_NAME}␣
↪config2.py tmp_work_dir_2 --cfg-options dist_params.port=29501
```

方式 3:

您可以使用环境变量' MASTER_PORT ' 在命令中设置端口:

```
CUDA_VISIBLE_DEVICES=0,1,2,3 GPUS=4 MASTER_PORT=29500 sh tools/slurm_train.sh $
↪{PARTITION} ${JOB_NAME} config1.py tmp_work_dir_1
CUDA_VISIBLE_DEVICES=4,5,6,7 GPUS=4 MASTER_PORT=29501 sh tools/slurm_train.sh $
↪{PARTITION} ${JOB_NAME} config2.py tmp_work_dir_2
```

CHAPTER 7

使用预训练模型推理

我们提供测试脚本来评估完整数据集（Cityscapes, PASCAL VOC, ADE20k 等）上的结果，同时为了使其他项目的整合更容易，也提供一些高级 API。

7.1 测试一个数据集

- 单卡 GPU
- CPU
- 单节点多卡 GPU
- 多节点

您可以使用以下命令来测试一个数据集。

```
# 单卡 GPU 测试
python tools/test.py ${配置文件} ${检查点文件} [--out ${结果文件}] [--eval ${评估指标}] [--
↪show]

# CPU: 如果机器没有 GPU, 则跟上述单卡 GPU 测试一致
# CPU: 如果机器有 GPU, 那么先禁用 GPU 再运行单 GPU 测试脚本
export CUDA_VISIBLE_DEVICES=-1 # 禁用 GPU
python tools/test.py ${配置文件} ${检查点文件} [--out ${结果文件}] [--eval ${评估指标}] [--
↪show]
```

(下页继续)

(续上页)

```
# 多卡 GPU 测试
./tools/dist_test.sh ${配置文件} ${检查点文件} ${GPU 数目} [--out ${结果文件}] [--eval ${评
估指标}]
```

可选参数:

- `RESULT_FILE`: pickle 格式的输出结果的文件名，如果不专门指定，结果将不会被专门保存成文件。（MMseg v0.17 之后，args.out 将只会保存评估时的中间结果或者是分割图的保存路径。）
- `EVAL_METRICS`: 在结果里将被评估的指标。这主要取决于数据集，`mIoU` 对于所有数据集都可获得，像 Cityscapes 数据集可以通过 `cityscapes` 命令来专门评估，就像标准的 `mIoU` 一样。
- `--show`: 如果被指定，分割结果将会在一张图像里画出来并且在另一个窗口展示。它仅仅是用来调试与可视化，并且仅针对单卡 GPU 测试。请确认 GUI 在您的环境里可用，否则您也许会遇到报错 `cannot connect to X server`
- `--show-dir`: 如果被指定，分割结果将会在一张图像里画出来并且保存在指定文件夹里。它仅仅是用来调试与可视化，并且仅针对单卡 GPU 测试。使用该参数时，您的环境不需要 GUI。
- `--eval-options`: 评估时的可选参数，当设置 `efficient_test=True` 时，它将会保存中间结果至本地文件里以节约 CPU 内存。请确认您本地硬盘有足够的存储空间（大于 20GB）。（MMseg v0.17 之后，`efficient_test` 不再生效，我们重构了 test api，通过使用一种渐近式的方式来提升评估和保存结果的效率。）

例子:

假设您已经下载检查点文件至文件夹 `checkpoints/` 里。

1. 测试 PSPNet 并可视化结果。按下任何键会进行到下一张图

```
python tools/test.py configs/pspnet/pspnet_r50-d8_512x1024_40k_cityscapes.py \
    checkpoints/pspnet_r50-d8_512x1024_40k_cityscapes_20200605_003338-2966598c.
↪pth \
    --show
```

2. 测试 PSPNet 并保存画出的图以便于之后的可视化

```
python tools/test.py configs/pspnet/pspnet_r50-d8_512x1024_40k_cityscapes.py \
    checkpoints/pspnet_r50-d8_512x1024_40k_cityscapes_20200605_003338-2966598c.
↪pth \
    --show-dir psp_r50_512x1024_40ki_cityscapes_results
```

3. 在数据集 PASCAL VOC (不保存测试结果) 上测试 PSPNet 并评估 mIoU

```
python tools/test.py configs/pspnet/pspnet_r50-d8_512x1024_20k_voc12aug.py \
    checkpoints/pspnet_r50-d8_512x1024_20k_voc12aug_20200605_003338-c57ef100.pth \
    --eval mAP
```

4. 使用 4 卡 GPU 测试 PSPNet，并且在标准 mIoU 和 cityscapes 指标里评估模型

```
./tools/dist_test.sh configs/pspnet/pspnet_r50-d8_512x1024_40k_cityscapes.py \
    checkpoints/pspnet_r50-d8_512x1024_40k_cityscapes_20200605_003338-2966598c.
↪pth \
    4 --out results.pkl --eval mIoU cityscapes
```

注意：在 cityscapes mIoU 和我们的 mIoU 指标会有一些差异 (~0.1%) 。因为 cityscapes 默认是根据类别样本数的多少进行加权平均，而我们对所有的数据集都是采取直接平均的方法来得到 mIoU。

5. 在 cityscapes 数据集上 4 卡 GPU 测试 PSPNet，并生成 png 文件以便提交给官方评估服务器

首先,在配置文件里添加内容:`configs/pspnet/pspnet_r50-d8_512x1024_40k_cityscapes.py`,

```
data = dict(
    test=dict(
        img_dir='leftImg8bit/test',
        ann_dir='gtFine/test'))
```

随后，进行测试。

```
./tools/dist_test.sh configs/pspnet/pspnet_r50-d8_512x1024_40k_cityscapes.py \
    checkpoints/pspnet_r50-d8_512x1024_40k_cityscapes_20200605_003338-2966598c.
↪pth \
    4 --format-only --eval-options "imgfile_prefix=./pspnet_test_results"
```

您会在文件夹 `./pspnet_test_results` 里得到生成的 png 文件。您也许可以运行 `zip -r results.zip pspnet_test_results/` 并提交 zip 文件给 evaluation server 。

6. 在 Cityscapes 数据集上使用 CPU 高效内存选项来测试 DeeplabV3+ `mIoU` 指标 (没有保存测试结果)

```
python tools/test.py \
configs/deeplabv3plus/deeplabv3plus_r18-d8_512x1024_80k_cityscapes.py \
deeplabv3plus_r18-d8_512x1024_80k_cityscapes_20201226_080942-cff257fe.pth \
--eval-options efficient_test=True \
--eval mIoU
```

使用 `pmap` 可查看 CPU 内存情况，`efficient_test=True` 会使用约 2.25GB 的 CPU 内存，`efficient_test=False` 会使用约 11.06GB 的 CPU 内存。这个可选参数可以节约很多 CPU 内存。（MMseg v0.17 之后, `efficient_test` 参数将不再生效, 我们使用了一种渐近的方式来更加有效快速地评估和保存结果。）

7. 在 LoveDA 数据集上 1 卡 GPU 测试 PSPNet，并生成 png 文件以便提交给官方评估服务器

首先，在配置文件里添加内容：`configs/pspnet/pspnet_r50-d8_512x512_80k_loveda.py`,

```
data = dict(
    test=dict(
        img_dir='img_dir/test',
        ann_dir='ann_dir/test'))
```

随后，进行测试。

```
python ./tools/test.py configs/pspnet/pspnet_r50-d8_512x512_80k_loveda.py \
    checkpoints/pspnet_r50-d8_512x512_80k_loveda_20211104_155728-88610f9f.pth \
    --format-only --eval-options "imgfile_prefix=./pspnet_test_results"
```

您会在文件夹 `./pspnet_test_results` 里得到生成的 png 文件。您也许可以运行 `zip -r -j Results.zip pspnet_test_results/` 并提交 zip 文件给 evaluation server 。

CHAPTER 8

教程 1: 学习配置文件

我们整合了模块和继承设计到我们的配置里，这便于做很多实验。如果您想查看配置文件，您可以运行 `python tools/print_config.py /PATH/TO/CONFIG` 去查看完整的配置文件。您还可以传递参数 `--cfg-options xxx.yyy=zzz` 去查看更新的配置。

8.1 配置文件的结构

在 `config/_base_` 文件夹下面有 4 种基本组件类型：数据集 (dataset)，模型 (model)，训练策略 (schedule) 和运行时的默认设置 (default runtime)。许多方法都可以方便地通过组合这些组件进行实现。这样，像 DeepLabV3, PSPNet 这样的模型可以容易地被构造。被来自 `_base_` 下的组件来构建的配置叫做 *原始配置 (primitive)*。

对于所有在同一个文件夹下的配置文件，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，我们推荐社区贡献者继承已有的方法配置文件。例如，如果一些修改是基于 DeepLabV3，使用者首先首先应该通过指定 `_base_ = ../deeplabv3/deeplabv3_r50_512x1024_40ki_cityscapes.py` 来继承基础 DeepLabV3 结构，再去修改配置文件里其他内容以完成继承。

如果您正在构建一个完整的新模型，它完全没有和已有的方法共享一些结构，您可能需要在 `configs` 下面创建一个文件夹 `xxxnet`。更详细的文档，请参照 mmcv 。

8.2 配置文件命名风格

我们按照下面的风格去命名配置文件，社区贡献者被建议使用同样的风格。

```
{model}_{backbone}_[misc]_[gpu x batch_per_gpu]_{resolution}_{iterations}_{dataset}
```

`{xxx}` 是被要求的文件 `[yyy]` 是可选的。

- `{model}`: 模型种类，例如 `psp`，`deeplabv3` 等等
- `{backbone}`: 主干网络种类，例如 `r50` (ResNet-50)，`x101` (ResNeXt-101)
- `[misc]`: 模型中各式各样的设置/插件，例如 `dconv`，`gcb`，`attention`，`mstrain`
- `[gpu x batch_per_gpu]`: GPU 数目和每个 GPU 的样本数，默认为 `8x2`
- `{iterations}`: 训练迭代轮数，如 `160k`
- `{dataset}`: 数据集，如 `cityscapes`，`voc12aug`，`ade`

8.3 PSPNet 的一个例子

为了帮助使用者熟悉这个流行的语义分割框架的完整配置文件和模块，我们在下面对使用 ResNet50V1c 的 PSPNet 的配置文件做了详细的注释说明。更多的详细使用和其他模块的替代项请参考 API 文档。

```
norm_cfg = dict(type='SyncBN', requires_grad=True)  # 分割框架通常使用 SyncBN
model = dict(
    type='EncoderDecoder',  # 分割器 (segmentor) 的名字
    pretrained='open-mmlab://resnet50_v1c',  # 将被加载的 ImageNet 预训练主干网络
    backbone=dict(
        type='ResNetV1c',  # 主干网络的类别。 可用选项请参考 mmseg/models/backbones/resnet.
↪py
        depth=50,  # 主干网络的深度。通常为 50 和 101。
        num_stages=4,  # 主干网络状态 (stages) 的数目，这些状态产生的特征图作为后续的 head 的输
入。
        out_indices=(0, 1, 2, 3),  # 每个状态产生的特征图输出的索引。
        dilations=(1, 1, 2, 4),  # 每一层 (layer) 的空心率 (dilation rate)。
        strides=(1, 2, 1, 1),  # 每一层 (layer) 的步长 (stride)。
        norm_cfg=dict(  # 归一化层 (norm layer) 的配置项。
            type='SyncBN',  # 归一化层的类别。通常是 SyncBN。
            requires_grad=True),  # 是否训练归一化里的 gamma 和 beta。
        norm_eval=False,  # 是否冻结 BN 里的统计项。
        style='pytorch',  # 主干网络的风格，'pytorch' 意思是步长为 2 的层为 3x3 卷积， 'caffe
↪' 意思是步长为 2 的层为 1x1 卷积。
        contract_dilation=True),  # 当空洞 > 1, 是否压缩第一个空洞层。
    decode_head=dict(
```

(下页继续)

(续上页)

```
        type='PSPHead',  # 解码头 (decode head) 的类别。 可用选项请参考 mmseg/models/
↪decode_heads。
        in_channels=2048,  # 解码头的输入通道数。
        in_index=3,  # 被选择的特征图 (feature map) 的索引。
        channels=512,  # 解码头中间态 (intermediate) 的通道数。
        pool_scales=(1, 2, 3, 6),  # PSPHead 平均池化 (avg pooling) 的规模 (scales)。 细节
请参考文章内容。
        dropout_ratio=0.1,  # 进入最后分类层 (classification layer) 之前的 dropout 比例。
        num_classes=19,  # 分割前景的种类数目。 通常情况下，cityscapes 为 19，VOC 为 21，
ADE20k 为 150。
        norm_cfg=dict(type='SyncBN', requires_grad=True),  # 归一化层的配置项。
        align_corners=False,  # 解码里调整大小 (resize) 的 align_corners 参数。
        loss_decode=dict(  # 解码头 (decode_head) 里的损失函数的配置项。
            type='CrossEntropyLoss',  # 在分割里使用的损失函数的类别。
            use_sigmoid=False,  # 在分割里是否使用 sigmoid 激活。
            loss_weight=1.0)),  # 解码头里损失的权重。
    auxiliary_head=dict(
        type='FCNHead',  # 辅助头 (auxiliary head) 的种类。可用选项请参考 mmseg/models/
↪decode_heads。
        in_channels=1024,  # 辅助头的输入通道数。
        in_index=2,  # 被选择的特征图 (feature map) 的索引。
        channels=256,  # 辅助头中间态 (intermediate) 的通道数。
        num_convs=1,  # FCNHead 里卷积 (convs) 的数目. 辅助头里通常为 1。
        concat_input=False,  # 在分类层 (classification layer) 之前是否连接 (concat) 输入和
卷积的输出。
        dropout_ratio=0.1,  # 进入最后分类层 (classification layer) 之前的 dropout 比例。
        num_classes=19,  # 分割前景的种类数目。 通常情况下，cityscapes 为 19，VOC 为 21，
ADE20k 为 150。
        norm_cfg=dict(type='SyncBN', requires_grad=True),  # 归一化层的配置项。
        align_corners=False,  # 解码里调整大小 (resize) 的 align_corners 参数。
        loss_decode=dict(  # 辅助头 (auxiliary head) 里的损失函数的配置项。
            type='CrossEntropyLoss',  # 在分割里使用的损失函数的类别。
            use_sigmoid=False,  # 在分割里是否使用 sigmoid 激活。
            loss_weight=0.4)))  # 辅助头里损失的权重。默认设置为 0.4。
train_cfg = dict()  # train_cfg 当前仅是一个占位符。
test_cfg = dict(mode='whole')  # 测试模式， 选项是 'whole' 和 'sliding'. 'whole': 整张图像
全卷积 (fully-convolutional) 测试。 'sliding': 图像上做滑动裁剪窗口 (sliding crop window)。
dataset_type = 'CityscapesDataset'  # 数据集类型，这将被用来定义数据集。
data_root = 'data/cityscapes/'  # 数据的根路径。
img_norm_cfg = dict(  # 图像归一化配置，用来归一化输入的图像。
    mean=[123.675, 116.28, 103.53],  # 预训练里用于预训练主干网络模型的平均值。
    std=[58.395, 57.12, 57.375],  # 预训练里用于预训练主干网络模型的标准差。
    to_rgb=True)  # 预训练里用于预训练主干网络的图像的通道顺序。
```

(下页继续)

(续上页)

```
crop_size = (512, 1024)  # 训练时的裁剪大小
train_pipeline = [  # 训练流程
    dict(type='LoadImageFromFile'),  # 第 1 个流程，从文件路径里加载图像。
    dict(type='LoadAnnotations'),  # 第 2 个流程，对于当前图像，加载它的注释信息。
    dict(type='Resize',  # 变化图像和其注释大小的数据增广的流程。
        img_scale=(2048, 1024),  # 图像的最大规模。
        ratio_range=(0.5, 2.0)), # 数据增广的比例范围。
    dict(type='RandomCrop',  # 随机裁剪当前图像和其注释大小的数据增广的流程。
        crop_size=(512, 1024),  # 随机裁剪图像生成 patch 的大小。
        cat_max_ratio=0.75),  # 单个类别可以填充的最大区域的比例。
    dict(
        type='RandomFlip',  # 翻转图像和其注释大小的数据增广的流程。
        flip_ratio=0.5),  # 翻转图像的概率
    dict(type='PhotoMetricDistortion'),  # 光学上使用一些方法扭曲当前图像和其注释的数据增广的流
程。
    dict(
        type='Normalize',  # 归一化当前图像的数据增广的流程。
        mean=[123.675, 116.28, 103.53],  # 这些键与 img_norm_cfg 一致，因为 img_norm_cfg␣
↪被
        std=[58.395, 57.12, 57.375],  # 用作参数。
        to_rgb=True),
    dict(type='Pad',  # 填充当前图像到指定大小的数据增广的流程。
        size=(512, 1024),  # 填充的图像大小。
        pad_val=0,  # 图像的填充值。
        seg_pad_val=255),  # 'gt_semantic_seg' 的填充值。
    dict(type='DefaultFormatBundle'),  # 流程里收集数据的默认格式捆。
    dict(type='Collect',  # 决定数据里哪些键被传递到分割器里的流程。
        keys=['img', 'gt_semantic_seg'])
]
test_pipeline = [
    dict(type='LoadImageFromFile'),  # 第 1 个流程，从文件路径里加载图像。
    dict(
        type='MultiScaleFlipAug',  # 封装测试时数据增广 (test time augmentations)。
        img_scale=(2048, 1024),  # 决定测试时可改变图像的最大规模。用于改变图像大小的流程。
        flip=False,  # 测试时是否翻转图像。
        transforms=[
            dict(type='Resize',  # 使用改变图像大小的数据增广。
                 keep_ratio=True),  # 是否保持宽和高的比例，这里的图像比例设置将覆盖上面的图像规
模大小的设置。
            dict(type='RandomFlip'),  # 考虑到 RandomFlip 已经被添加到流程里，当␣
↪flip=False 时它将不被使用。
            dict(
                type='Normalize',  # 归一化配置项，值来自 img_norm_cfg。
```

(下页继续)

(续上页)

```
                mean=[123.675, 116.28, 103.53],
                std=[58.395, 57.12, 57.375],
                to_rgb=True),
            dict(type='ImageToTensor', # 将图像转为张量
                keys=['img']),
            dict(type='Collect', # 收集测试时必须的键的收集流程。
                keys=['img'])
        ])
]
data = dict(
    samples_per_gpu=2,  # 单个 GPU 的 Batch size
    workers_per_gpu=2,  # 单个 GPU 分配的数据加载线程数
    train=dict(  # 训练数据集配置
        type='CityscapesDataset',  # 数据集的类别, 细节参考自 mmseg/datasets/。
        data_root='data/cityscapes/',  # 数据集的根目录。
        img_dir='leftImg8bit/train',  # 数据集图像的文件夹。
        ann_dir='gtFine/train',  # 数据集注释的文件夹。
        pipeline=[  # 流程, 由之前创建的 train_pipeline 传递进来。
            dict(type='LoadImageFromFile'),
            dict(type='LoadAnnotations'),
            dict(
                type='Resize', img_scale=(2048, 1024), ratio_range=(0.5, 2.0)),
            dict(type='RandomCrop', crop_size=(512, 1024), cat_max_ratio=0.75),
            dict(type='RandomFlip', flip_ratio=0.5),
            dict(type='PhotoMetricDistortion'),
            dict(
                type='Normalize',
                mean=[123.675, 116.28, 103.53],
                std=[58.395, 57.12, 57.375],
                to_rgb=True),
            dict(type='Pad', size=(512, 1024), pad_val=0, seg_pad_val=255),
            dict(type='DefaultFormatBundle'),
            dict(type='Collect', keys=['img', 'gt_semantic_seg'])
        ]),
    val=dict(  # 验证数据集的配置
        type='CityscapesDataset',
        data_root='data/cityscapes/',
        img_dir='leftImg8bit/val',
        ann_dir='gtFine/val',
        pipeline=[  # 由之前创建的 test_pipeline 传递的流程。
            dict(type='LoadImageFromFile'),
            dict(
                type='MultiScaleFlipAug',
```

(下页继续)

(续上页)

```
                img_scale=(2048, 1024),
                flip=False,
                transforms=[
                    dict(type='Resize', keep_ratio=True),
                    dict(type='RandomFlip'),
                    dict(
                        type='Normalize',
                        mean=[123.675, 116.28, 103.53],
                        std=[58.395, 57.12, 57.375],
                        to_rgb=True),
                    dict(type='ImageToTensor', keys=['img']),
                    dict(type='Collect', keys=['img'])
                ])
        ]),
    test=dict(
        type='CityscapesDataset',
        data_root='data/cityscapes/',
        img_dir='leftImg8bit/val',
        ann_dir='gtFine/val',
        pipeline=[
            dict(type='LoadImageFromFile'),
            dict(
                type='MultiScaleFlipAug',
                img_scale=(2048, 1024),
                flip=False,
                transforms=[
                    dict(type='Resize', keep_ratio=True),
                    dict(type='RandomFlip'),
                    dict(
                        type='Normalize',
                        mean=[123.675, 116.28, 103.53],
                        std=[58.395, 57.12, 57.375],
                        to_rgb=True),
                    dict(type='ImageToTensor', keys=['img']),
                    dict(type='Collect', keys=['img'])
                ])
        ]))
log_config = dict(  # 注册日志钩 (register logger hook) 的配置文件。
    interval=50,  # 打印日志的间隔
      hooks=[ # 训练期间执行的钩子
        dict(type='TextLoggerHook', by_epoch=False),
        dict(type='TensorboardLoggerHook', by_epoch=False),
        dict(type='MMSegWandbHook', by_epoch=False, # 还支持 Wandb 记录器，它需要安装␣
↪`wandb`。
```

(下页继续)

(续上页)

```
             init_kwargs={'entity': "OpenMMLab", # 用于登录 wandb 的实体
                          'project': "mmseg", # WandB 中的项目名称
                          'config': cfg_dict}), # 检查 https://docs.wandb.ai/ref/
↪python/init 以获取更多初始化参数
    ])

dist_params = dict(backend='nccl')  # 用于设置分布式训练的参数，端口也同样可被设置。
log_level = 'INFO'  # 日志的级别。
load_from = None  # 从一个给定路径里加载模型作为预训练模型，它并不会消耗训练时间。
resume_from = None  # 从给定路径里恢复检查点 (checkpoints)，训练模式将从检查点保存的轮次开始恢复训练。
workflow = [('train', 1)]  # runner 的工作流程。 [('train', 1)] 意思是只有一个工作流程而且工作流程 'train' 仅执行一次。根据 `runner.max_iters` 工作流程训练模型的迭代轮数为 40000 次。
cudnn_benchmark = True  # 是否是使用 cudnn_benchmark 去加速，它对于固定输入大小的可以提高训练速度。
optimizer = dict(  # 用于构建优化器的配置文件。支持 PyTorch 中的所有优化器，同时它们的参数与 PyTorch 里的优化器参数一致。
    type='SGD',  # 优化器种类，更多细节可参考 https://github.com/open-mmlab/mmcv/blob/
↪master/mmcv/runner/optimizer/default_constructor.py#L13。
    lr=0.01,  # 优化器的学习率，参数的使用细节请参照对应的 PyTorch 文档。
    momentum=0.9,  # 动量 (Momentum)
    weight_decay=0.0005)  # SGD 的衰减权重 (weight decay)。
optimizer_config = dict()  # 用于构建优化器钩 (optimizer hook) 的配置文件，执行细节请参考␣
↪https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/optimizer.py#L8。
lr_config = dict(
    policy='poly',  # 调度流程的策略，同样支持 Step, CosineAnnealing, Cyclic 等. 请从␣
↪https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/lr_updater.py#L9 参考 LrUpdater 的细节。
    power=0.9,  # 多项式衰减 (polynomial decay) 的幂。
    min_lr=0.0001,  # 用来稳定训练的最小学习率。
    by_epoch=False)  # 是否按照每个 epoch 去算学习率。
runner = dict(
    type='IterBasedRunner', # 将使用的 runner 的类别 (例如 IterBasedRunner 或␣
↪EpochBasedRunner)。
    max_iters=40000) # 全部迭代轮数大小，对于 EpochBasedRunner 使用 `max_epochs` 。
checkpoint_config = dict(  # 设置检查点钩子 (checkpoint hook) 的配置文件。执行时请参考␣
↪https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/checkpoint.py。
    by_epoch=False,  # 是否按照每个 epoch 去算 runner。
    interval=4000)  # 保存的间隔
evaluation = dict(  # 构建评估钩 (evaluation hook) 的配置文件。细节请参考 mmseg/core/
↪evaluation/eval_hook.py。
    interval=4000,  # 评估的间歇点
    metric='mIoU')  # 评估的指标
```

(下页继续)

(续上页)

```

```

8.4 FAQ

8.4.1 忽略基础配置文件里的一些域内容。

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

在 MMSegmentation 里，例如为了改变 PSPNet 的主干网络的某些内容：

```
norm_cfg = dict(type='SyncBN', requires_grad=True)
model = dict(
    type='MaskRCNN',
    pretrained='torchvision://resnet50',
    backbone=dict(
        type='ResNetV1c',
        depth=50,
        num_stages=4,
        out_indices=(0, 1, 2, 3),
        dilations=(1, 1, 2, 4),
        strides=(1, 2, 1, 1),
        norm_cfg=norm_cfg,
        norm_eval=False,
        style='pytorch',
        contract_dilation=True),
    decode_head=dict(...),
    auxiliary_head=dict(...))
```

ResNet 和 HRNet 使用不同的关键词去构建。

```
_base_ = '../pspnet/psp_r50_512x1024_40ki_cityscpaes.py'
norm_cfg = dict(type='SyncBN', requires_grad=True)
model = dict(
    pretrained='open-mmlab://msra/hrnetv2_w32',
    backbone=dict(
        _delete_=True,
        type='HRNet',
        norm_cfg=norm_cfg,
        extra=dict(
            stage1=dict(
```

(下页继续)

(续上页)

```
                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)))),
    decode_head=dict(...),
    auxiliary_head=dict(...))
```

`_delete_=True` 将用新的键去替换 `backbone` 域内所有老的键。

8.4.2 使用配置文件里的中间变量

配置文件里会使用一些中间变量，例如数据集里的 `train_pipeline/test_pipeline`。需要注意的是，在子配置文件里修改中间变量时，使用者需要再次传递这些变量给对应的域。例如，我们想改变在训练或测试时，PSPNet 的多尺度策略 (multi scale strategy)，`train_pipeline/test_pipeline` 是我们想要修改的中间变量。

```
_base_ = '../pspnet/psp_r50_512x1024_40ki_cityscapes.py'
crop_size = (512, 1024)
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='LoadAnnotations'),
    dict(type='Resize', img_scale=(2048, 1024), ratio_range=(1.0, 2.0)),  # 改成 [1.,␣
↪2.]
```

(下页继续)

(续上页)

```
    dict(type='RandomCrop', crop_size=crop_size, cat_max_ratio=0.75),
    dict(type='RandomFlip', flip_ratio=0.5),
    dict(type='PhotoMetricDistortion'),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='Pad', size=crop_size, pad_val=0, seg_pad_val=255),
    dict(type='DefaultFormatBundle'),
    dict(type='Collect', keys=['img', 'gt_semantic_seg']),
]
test_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(
        type='MultiScaleFlipAug',
        img_scale=(2048, 1024),
        img_ratios=[0.5, 0.75, 1.0, 1.25, 1.5, 1.75],  # 改成多尺度测试 (multi scale␣
↪testing)。
        flip=False,
        transforms=[
            dict(type='Resize', keep_ratio=True),
            dict(type='RandomFlip'),
            dict(type='Normalize', **img_norm_cfg),
            dict(type='ImageToTensor', keys=['img']),
            dict(type='Collect', keys=['img']),
        ])
]
data = dict(
    train=dict(pipeline=train_pipeline),
    val=dict(pipeline=test_pipeline),
    test=dict(pipeline=test_pipeline))
```

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

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

```
_base_ = '../pspnet/psp_r50_512x1024_40ki_cityscpaes.py'
norm_cfg = dict(type='BN', requires_grad=True)
model = dict(
    backbone=dict(norm_cfg=norm_cfg),
    decode_head=dict(norm_cfg=norm_cfg),
    auxiliary_head=dict(norm_cfg=norm_cfg))
```

CHAPTER 9

教程 2: 自定义数据集

9.1 通过重新组织数据来定制数据集

最简单的方法是将您的数据集进行转化，并组织成文件夹的形式。

如下的文件结构就是一个例子。

```
├── data
│   ├── my_dataset
│   │   ├── img_dir
│   │   │   ├── train
│   │   │   │   ├── xxx{img_suffix}
│   │   │   │   ├── yyy{img_suffix}
│   │   │   │   ├── zzz{img_suffix}
│   │   │   ├── val
│   │   ├── ann_dir
│   │   │   ├── train
│   │   │   │   ├── xxx{seg_map_suffix}
│   │   │   │   ├── yyy{seg_map_suffix}
│   │   │   │   ├── zzz{seg_map_suffix}
│   │   │   ├── val
```

一个训练对将由 img_dir/ann_dir 里同样首缀的文件组成。

如果给定 `split` 参数，只有部分在 img_dir/ann_dir 里的文件会被加载。我们可以对被包括在 split 文本里的

文件指定前缀。

除此以外，一个 split 文本如下所示：

```
xxx
zzz
```

只有

data/my_dataset/img_dir/train/xxx{img_suffix}, data/my_dataset/img_dir/train/zzz{img_suffix}, data/my_dataset/ann_dir/train/xxx{seg_map_suffix}, data/my_dataset/ann_dir/train/zzz{seg_map_suffix} 将被加载。

注意：标注是跟图像同样的形状 (H, W)，其中的像素值的范围是 [0, num_classes - 1]。您也可以使用 pillow 的 'P' 模式去创建包含颜色的标注。

9.2 通过混合数据去定制数据集

MMSegmentation 同样支持混合数据集去训练。当前它支持拼接 (concat), 重复 (repeat) 和多图混合 (multi-image mix) 数据集。

9.2.1 重复数据集

我们使用 RepeatDataset 作为包装 (wrapper) 去重复数据集。例如，假设原始数据集是 Dataset_A，为了重复它，配置文件如下：

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

9.2.2 拼接数据集

有 2 种方式去拼接数据集。

1. 如果您想拼接的数据集是同样的类型，但有不同的标注文件，您可以按如下操作去拼接数据集的配置文件：

    1. 您也许可以拼接两个标注文件夹 `ann_dir`

    ```
    dataset_A_train = dict(
        type='Dataset_A',
        img_dir = 'img_dir',
        ann_dir = ['anno_dir_1', 'anno_dir_2'],
        pipeline=train_pipeline
    )
    ```

    2. 您也可以去拼接两个 `split` 文件列表

    ```
    dataset_A_train = dict(
        type='Dataset_A',
        img_dir = 'img_dir',
        ann_dir = 'anno_dir',
        split = ['split_1.txt', 'split_2.txt'],
        pipeline=train_pipeline
    )
    ```

    3. 您也可以同时拼接 `ann_dir` 文件夹和 `split` 文件列表

    ```
    dataset_A_train = dict(
        type='Dataset_A',
        img_dir = 'img_dir',
        ann_dir = ['anno_dir_1', 'anno_dir_2'],
        split = ['split_1.txt', 'split_2.txt'],
        pipeline=train_pipeline
    )
    ```

    在这样的情况下，`ann_dir_1` 和 `ann_dir_2` 分别对应于 `split_1.txt` 和 `split_2.txt`

2. 如果您想拼接不同的数据集，您可以如下去拼接数据集的配置文件：

```
dataset_A_train = dict()
dataset_B_train = dict()

data = dict(
    imgs_per_gpu=2,
    workers_per_gpu=2,
```

(下页继续)

(续上页)

```
    train = [
        dataset_A_train,
        dataset_B_train
    ],
    val = dataset_A_val,
    test = dataset_A_test
    )
```

一个更复杂的例子如下：分别重复 `Dataset_A` 和 `Dataset_B` N 次和 M 次，然后再去拼接重复后的数据集

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

(下页继续)

(续上页)

```
    test = dataset_A_test
)
```

9.2.3 多图混合集

我们使用 MultiImageMixDataset 作为包装 (wrapper) 去混合多个数据集的图片。MultiImageMixDataset 可以被类似 mosaic 和 mixup 的多图混合数据🄵广使用。

MultiImageMixDataset 与 Mosaic 数据🄵广一起使用的例子:

```
train_pipeline = [
    dict(type='RandomMosaic', prob=1),
    dict(type='Resize', img_scale=(1024, 512), keep_ratio=True),
    dict(type='RandomFlip', prob=0.5),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='DefaultFormatBundle'),
    dict(type='Collect', keys=['img', 'gt_semantic_seg']),
]

train_dataset = dict(
    type='MultiImageMixDataset',
    dataset=dict(
        classes=classes,
        palette=palette,
        type=dataset_type,
        reduce_zero_label=False,
        img_dir=data_root + "images/train",
        ann_dir=data_root + "annotations/train",
        pipeline=[
            dict(type='LoadImageFromFile'),
            dict(type='LoadAnnotations'),
        ]
    ),
    pipeline=train_pipeline
)
```

CHAPTER 10

教程 3: 自定义数据流程

10.1 数据流程的设计

按照通常的惯例，我们使用 `Dataset` 和 `DataLoader` 做多线程的数据加载。`Dataset` 返回一个数据内容的字典，里面对应于模型前传方法的各个参数。因为在语义分割中，输入的图像数据具有不同的大小，我们在 MMCV 里引入一个新的 `DataContainer` 类别去帮助收集和分发不同大小的输入数据。

更多细节，请查看这里 。

数据的准备流程和数据集是解耦的。通常一个数据集定义了如何处理标注数据（annotations）信息，而一个数据流程定义了准备一个数据字典的所有步骤。一个流程包括了一系列操作，每个操作里都把一个字典作为输入，然后再输出一个新的字典给下一个变换操作。

这些操作可分为数据加载 (data loading)，预处理 (pre-processing)，格式变化 (formatting) 和测试时数据增强 (test-time augmentation)。

下面的例子就是 PSPNet 的一个流程：

```
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
crop_size = (512, 1024)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='LoadAnnotations'),
    dict(type='Resize', img_scale=(2048, 1024), ratio_range=(0.5, 2.0)),
    dict(type='RandomCrop', crop_size=crop_size, cat_max_ratio=0.75),
```

(下页继续)

(续上页)

```
    dict(type='RandomFlip', flip_ratio=0.5),
    dict(type='PhotoMetricDistortion'),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='Pad', size=crop_size, pad_val=0, seg_pad_val=255),
    dict(type='DefaultFormatBundle'),
    dict(type='Collect', keys=['img', 'gt_semantic_seg']),
]
test_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(
        type='MultiScaleFlipAug',
        img_scale=(2048, 1024),
        # img_ratios=[0.5, 0.75, 1.0, 1.25, 1.5, 1.75],
        flip=False,
        transforms=[
            dict(type='Resize', keep_ratio=True),
            dict(type='RandomFlip'),
            dict(type='Normalize', **img_norm_cfg),
            dict(type='ImageToTensor', keys=['img']),
            dict(type='Collect', keys=['img']),
        ])
]
```

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

10.1.1 数据加载 Data loading

`LoadImageFromFile`

- 增加: img, img_shape, ori_shape

`LoadAnnotations`

- 增加: gt_semantic_seg, seg_fields

10.1.2 预处理 Pre-processing

`Resize`

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

`RandomFlip`

- 增加: flip

- 更新: img, *seg_fields

`Pad`

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

`RandomCrop`

- 更新: img, pad_shape, *seg_fields

`Normalize`

- 增加: img_norm_cfg
- 更新: img

`SegRescale`

- 更新: gt_semantic_seg

`PhotoMetricDistortion`

- 更新: img

10.1.3 格式 Formatting

`ToTensor`

- 更新: 由 `keys` 指定

`ImageToTensor`

- 更新: 由 `keys` 指定

`Transpose`

- 更新: 由 `keys` 指定

`ToDataContainer`

- 更新: 由 `keys` 指定

`DefaultFormatBundle`

- 更新: img, gt_semantic_seg

`Collect`

- 增加: img_meta (the keys of img_meta is specified by `meta_keys`)
- 移除: all other keys except for those specified by `keys`

10.1.4 测试时数据增强 Test time augmentation

`MultiScaleFlipAug`

10.2 拓展和使用自定义的流程

1. 在任何一个文件里写一个新的流程，例如 `my_pipeline.py`，它以一个字典作为输入并且输出一个字典

```python
from mmseg.datasets import PIPELINES

@PIPELINES.register_module()
class MyTransform:

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

2. 导入一个新类

```python
from .my_pipeline import MyTransform
```

3. 在配置文件里使用它

```python
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
crop_size = (512, 1024)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='LoadAnnotations'),
    dict(type='Resize', img_scale=(2048, 1024), ratio_range=(0.5, 2.0)),
    dict(type='RandomCrop', crop_size=crop_size, cat_max_ratio=0.75),
    dict(type='RandomFlip', flip_ratio=0.5),
    dict(type='PhotoMetricDistortion'),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='Pad', size=crop_size, pad_val=0, seg_pad_val=255),
    dict(type='MyTransform'),
    dict(type='DefaultFormatBundle'),
    dict(type='Collect', keys=['img', 'gt_semantic_seg']),
]
```

CHAPTER 11

教程 4: 自定义模型

11.1 自定义优化器 (optimizer)

假设您想增加一个新的叫 MyOptimizer 的优化器，它的参数分别为 a, b, 和 c。您首先需要在一个文件里实现这个新的优化器，例如在 mmseg/core/optimizer/my_optimizer.py 里面：

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


@OPTIMIZERS.register_module
class MyOptimizer(Optimizer):

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

然后增加这个模块到 mmseg/core/optimizer/__init__.py 里面，这样注册器 (registry) 将会发现这个新的模块并添加它：

```
from .my_optimizer import MyOptimizer
```

之后您可以在配置文件的 optimizer 域里使用 MyOptimizer，如下所示，在配置文件里，优化器被 optimizer 域所定义：

```
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，域可以被修改为：

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

我们已经支持了 PyTorch 自带的全部优化器，唯一修改的地方是在配置文件里的 `optimizer` 域。例如，如果您想使用 `ADAM`，尽管数值表现会掉点，还是可以如下修改：

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

使用者可以直接按照 PyTorch 文档教程 去设置参数。

11.2 定制优化器的构造器 (optimizer constructor)

对于优化，一些模型可能会有一些特别定义的参数，例如批归一化 (BatchNorm) 层里面的权重衰减 (weight decay)。使用者可以通过定制优化器的构造器来微调这些细粒度的优化器参数。

```
from mmcv.utils import build_from_cfg

from mmcv.runner import OPTIMIZER_BUILDERS
from .cocktail_optimizer import CocktailOptimizer


@OPTIMIZER_BUILDERS.register_module
class CocktailOptimizerConstructor(object):

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

    def __call__(self, model):

        return my_optimizer
```

11.3 开发和增加新的组件（Module）

MMSegmentation 里主要有 2 种组件：

- 主干网络 (backbone): 通常是卷积网络的堆叠，来做特征提取，例如 ResNet, HRNet
- 解码头 (decoder head): 用于语义分割图的解码的组件（得到分割结果）

11.3.1 添加新的主干网络

这里我们以 MobileNet 为例，展示如何增加新的主干组件：

1. 创建一个新的文件 `mmseg/models/backbones/mobilenet.py`

```
import torch.nn as nn

from ..builder import BACKBONES


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

    def __init__(self, arg1, arg2):
        pass

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

    def init_weights(self, pretrained=None):
        pass
```

2. 在 `mmseg/models/backbones/__init__.py` 里面导入模块

```
from .mobilenet import MobileNet
```

3. 在您的配置文件里使用它

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

11.3.2 增加新的解码头 (decoder head) 组件

在 MMSegmentation 里面，对于所有的分割头，我们提供一个基类解码头 BaseDecodeHead 。所有新建的解码头都应该继承它。这里我们以 PSPNet 为例，展示如何开发和增加一个新的解码头组件：

首先，在 `mmseg/models/decode_heads/psp_head.py` 里添加一个新的解码头。PSPNet 中实现了一个语义分割的解码头。为了实现一个解码头，我们只需要在新构造的解码头中实现如下的 3 个函数：

```
@HEADS.register_module()
class PSPHead(BaseDecodeHead):

    def __init__(self, pool_scales=(1, 2, 3, 6), **kwargs):
        super(PSPHead, self).__init__(**kwargs)

    def init_weights(self):

    def forward(self, inputs):
```

接着，使用者需要在 mmseg/models/decode_heads/__init__.py 里面添加这个模块，这样对应的注册器 (registry) 可以查找并加载它们。

PSPNet 的配置文件如下所示：

```
norm_cfg = dict(type='SyncBN', requires_grad=True)
model = dict(
    type='EncoderDecoder',
    pretrained='pretrain_model/resnet50_v1c_trick-2cccc1ad.pth',
    backbone=dict(
        type='ResNetV1c',
        depth=50,
        num_stages=4,
        out_indices=(0, 1, 2, 3),
        dilations=(1, 1, 2, 4),
        strides=(1, 2, 1, 1),
        norm_cfg=norm_cfg,
        norm_eval=False,
        style='pytorch',
        contract_dilation=True),
    decode_head=dict(
        type='PSPHead',
        in_channels=2048,
        in_index=3,
        channels=512,
        pool_scales=(1, 2, 3, 6),
        dropout_ratio=0.1,
        num_classes=19,
        norm_cfg=norm_cfg,
        align_corners=False,
        loss_decode=dict(
            type='CrossEntropyLoss', use_sigmoid=False, loss_weight=1.0)))
```

11.3.3 增加新的损失函数

假设您想添加一个新的损失函数 MyLoss 到语义分割解码器里。为了添加一个新的损失函数，使用者需要在 mmseg/models/losses/my_loss.py 里面去实现它。weighted_loss 可以对计算损失时的每个样本做加权。

```python
import torch
import torch.nn as nn

from ..builder import LOSSES
from .utils import weighted_loss

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

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

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

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

然后使用者需要在 mmseg/models/losses/__init__.py 里面添加它：

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

为了使用它，修改 loss_xxx 域。之后您需要在解码头组件里修改 loss_decode 域。loss_weight 可以

被用来对不同的损失函数做加权。

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

CHAPTER 12

教程 5: 训练技巧

MMSegmentation 支持如下训练技巧：

12.1 主干网络和解码头组件使用不同的学习率 (Learning Rate, LR)

在语义分割里，一些方法会让解码头组件的学习率大于主干网络的学习率，这样可以获得更好的表现或更快的收敛。

在 MMSegmentation 里面，您也可以在配置文件里添加如下行来让解码头组件的学习率是主干组件的 10 倍。

```
optimizer=dict(
    paramwise_cfg = dict(
        custom_keys={
            'head': dict(lr_mult=10.)}))
```

通过这种修改，任何被分组到 `'head'` 的参数的学习率都将乘以 10。您也可以参照 MMCV 文档 获取更详细的信息。

12.2 在线难样本挖掘 (Online Hard Example Mining, OHEM)

对于训练时采样，我们在 这里 做了像素采样器。如下例子是使用 PSPNet 训练并采用 OHEM 策略的配置：

```
_base_ = './pspnet_r50-d8_512x1024_40k_cityscapes.py'
model=dict(
    decode_head=dict(
        sampler=dict(type='OHEMPixelSampler', thresh=0.7, min_kept=100000)) )
```

通过这种方式，只有置信分数在 0.7 以下的像素值点会被拿来训练。在训练时我们至少要保留 100000 个像素值点。如果 `thresh` 并未被指定，前 `min_kept` 个损失的像素值点才会被选择。

12.3 类别平衡损失 (Class Balanced Loss)

对于不平衡类别分布的数据集，您也许可以改变每个类别的损失权重。这里以 cityscapes 数据集为例：

```
_base_ = './pspnet_r50-d8_512x1024_40k_cityscapes.py'
model=dict(
    decode_head=dict(
        loss_decode=dict(
            type='CrossEntropyLoss', use_sigmoid=False, loss_weight=1.0,
            # DeepLab 对 cityscapes 使用这种权重
            class_weight=[0.8373, 0.9180, 0.8660, 1.0345, 1.0166, 0.9969, 0.9754,
                          1.0489, 0.8786, 1.0023, 0.9539, 0.9843, 1.1116, 0.9037,
                          1.0865, 1.0955, 1.0865, 1.1529, 1.0507])))
```

`class_weight` 将被作为 `weight` 参数，传递给 `CrossEntropyLoss`。详细信息请参照 PyTorch 文档 。

12.4 同时使用多种损失函数 (Multiple Losses)

对于训练时损失函数的计算，我们目前支持多个损失函数同时使用。以 `unet` 使用 `DRIVE` 数据集训练为例，使用 `CrossEntropyLoss` 和 `DiceLoss` 的 `1:3` 的加权和作为损失函数。配置文件写为:

```
_base_ = './fcn_unet_s5-d16_64x64_40k_drive.py'
model = dict(
    decode_head=dict(loss_decode=[dict(type='CrossEntropyLoss', loss_name='loss_ce',␣
↪loss_weight=1.0),
            dict(type='DiceLoss', loss_name='loss_dice', loss_weight=3.0)]),
    auxiliary_head=dict(loss_decode=[dict(type='CrossEntropyLoss', loss_name='loss_ce
↪',loss_weight=1.0),
            dict(type='DiceLoss', loss_name='loss_dice', loss_weight=3.0)]),
    )
```

通过这种方式，确定训练过程中损失函数的权重 loss_weight 和在训练日志里的名字 loss_name。

注意：loss_name 的名字必须带有 loss_ 前缀，这样它才能被包括在反传的图里。

12.5 在损失函数中忽略特定的 label 类别

默认设置 avg_non_ignore=False，即每个像素都用来计算损失函数。尽管其中的一些像素属于需要被忽略的类别。

对于训练时损失函数的计算，我们目前支持使用 avg_non_ignore 和 ignore_index 来忽略 label 特定的类别。这样损失函数将只在非忽略类别像素中求平均值，会获得更好的表现。这里是相关 PR。以 unet 使用 Cityscapes 数据集训练为例，在计算损失函数时，忽略 label 为 0 的背景，并且仅在不被忽略的像素上计算均值。配置文件写为:

```python
_base_ = './fcn_unet_s5-d16_4x4_512x1024_160k_cityscapes.py'
model = dict(
    decode_head=dict(
        ignore_index=0,
        loss_decode=dict(
            type='CrossEntropyLoss', use_sigmoid=False, loss_weight=1.0, avg_non_
↪ignore=True),
    auxiliary_head=dict(
        ignore_index=0,
        loss_decode=dict(
            type='CrossEntropyLoss', use_sigmoid=False, loss_weight=1.0, avg_non_
↪ignore=True)),
    ))
```

通过这种方式，确定训练过程中损失函数的权重 loss_weight 和在训练日志里的名字 loss_name。

注意：loss_name 的名字必须带有 loss_ 前缀，这样它才能被包括在反传的图里。

CHAPTER 13

教程 6: 自定义运行设定

13.1 自定义优化设定

13.1.1 自定义 PyTorch 支持的优化器

我们已经支持 PyTorch 自带的所有优化器，唯一需要修改的地方是在配置文件里的 `optimizer` 域里面。例如，如果您想使用 `ADAM` (注意如下操作可能会让模型表现下降)，可以使用如下修改：

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

为了修改模型的学习率，使用者仅需要修改配置文件里 optimizer 的 `lr` 即可。使用者可以参照 PyTorch 的 API 文档 直接设置参数。

13.1.2 自定义自己实现的优化器

1. 定义一个新的优化器

一个自定义的优化器可以按照如下去定义：

假如您想增加一个叫做 `MyOptimizer` 的优化器，它的参数分别有 `a`, `b`, 和 `c`。您需要创建一个叫 `mmseg/core/optimizer` 的新文件夹。然后再在文件，即 `mmseg/core/optimizer/my_optimizer.py` 里面去实现这个新优化器：

```python
from .registry import OPTIMIZERS
from torch.optim import Optimizer


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

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

2. 增加优化器到注册表 (registry)

为了让上述定义的模块被框架发现，首先这个模块应该被导入到主命名空间 (main namespace) 里。有两种方式可以实现它。

- 修改 `mmseg/core/optimizer/__init__.py` 来导入它

  新的被定义的模块应该被导入到 `mmseg/core/optimizer/__init__.py` 这样注册表将会发现新的模块并添加它

```python
from .my_optimizer import MyOptimizer
```

- 在配置文件里使用 `custom_imports` 去手动导入它

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

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

事实上，使用者完全可以用另一个按这样导入方法的文件夹结构，只要模块的根路径已经被添加到 `PYTHONPATH` 里面。

3. 在配置文件里定义优化器

之后您可以在配置文件的 `optimizer` 域里面使用 `MyOptimizer` 在配置文件里，优化器被定义在 `optimizer` 域里，如下所示：

```python
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，这个域可以被改成：

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

13.1.3 自定义优化器的构造器 (constructor)

有些模型可能需要在优化器里有一些特别参数的设置，例如批归一化层 (BatchNorm layers) 的权重衰减 (weight decay)。使用者可以通过自定义优化器的构造器去微调这些细粒度参数。

```
from mmcv.utils import build_from_cfg

from mmcv.runner.optimizer import OPTIMIZER_BUILDERS, OPTIMIZERS
from mmseg.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
```

默认的优化器构造器的实现可以参照 这里 ，它也可以被用作新的优化器构造器的模板。

13.1.4 额外的设置

优化器没有实现的一些技巧应该通过优化器构造器 (optimizer constructor) 或者钩子 (hook) 去实现，如设置基于参数的学习率 (parameter-wise learning rates)。我们列出一些常见的设置，它们可以稳定或加速模型的训练。如果您有更多的设置，欢迎在 PR 和 issue 里面提交。

- **使用梯度截断 (gradient clip) 去稳定训练:**

  一些模型需要梯度截断去稳定训练过程，如下所示

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

  如果您的配置继承自已经设置了 `optimizer_config` 的基础配置 (base config)，您可能需要 `_delete_=True` 来重写那些不需要的设置。更多细节请参照 配置文件文档 。

- **使用动量计划表 (momentum schedule) 去加速模型收敛:**

我们支持动量计划表去让模型基于学习率修改动量，这样可能让模型收敛地更快。动量计划表经常和学习率计划表 (LR scheduler) 一起使用，例如如下配置文件就在 3D 检测里经常使用以加速收敛。更多细节请参考 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,
)
```

13.2 自定义训练计划表

我们根据默认的训练迭代步数 40k/80k 来设置学习率，这在 MMCV 里叫做 `PolyLrUpdaterHook` 。我们也支持许多其他的学习率计划表：这里 ，例如 `CosineAnnealing` 和 `Poly` 计划表。下面是一些例子：

- 步计划表 Step schedule:

```
lr_config = dict(policy='step', step=[9, 10])
```

- 余弦退火计划表 ConsineAnnealing schedule:

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

13.3 自定义工作流 (workflow)

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

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

意思是训练是跑 1 个 epoch。有时候使用者可能想检查模型在验证集上的一些指标（如损失 loss，精确性 accuracy），我们可以这样设置工作流：

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

于是 1 个 epoch 训练，1 个 epoch 验证将交替运行。

注意:

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

13.4 自定义钩 (hooks)

13.4.1 使用 MMCV 实现的钩子 (hooks)

如果钩子已经在 MMCV 里被实现，如下所示，您可以直接修改配置文件来使用钩子：

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

13.4.2 修改默认的运行时间钩子 (runtime hooks)

以下的常用的钩子没有被 `custom_hooks` 注册：

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

在这些钩子里，只有 logger hook 有 `VERY_LOW` 优先级，其他的优先级都是 `NORMAL`。上述提及的教程已经包括了如何修改 `optimizer_config`，`momentum_config` 和 `lr_config`。这里我们展示我们如何处理 `log_config`，`checkpoint_config` 和 `evaluation`。

检查点配置文件 (Checkpoint config)

MMCV runner 将使用 checkpoint_config 去初始化 CheckpointHook.

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

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

日志配置文件 (Log config)

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

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

评估配置文件 (Evaluation config)

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

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

CHAPTER 14

常用工具

除了训练和测试的脚本，我们在 tools/ 文件夹路径下还提供许多有用的工具。

14.1 计算参数量（params）和计算量（FLOPs）(试验性)

我们基于 flops-counter.pytorch 提供了一个用于计算给定模型参数量和计算量的脚本。

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

您将得到如下的结果：

```
==============================
Input shape: (3, 2048, 1024)
Flops: 1429.68 GMac
Params: 48.98 M
==============================
```

注意: 这个工具仍然是试验性的，我们无法保证数字是正确的。您可以拿这些结果做简单的实验的对照，在写技术文档报告或者论文前您需要再次确认一下。

(1) 计算量与输入的形状有关，而参数量与输入的形状无关，默认的输入形状是 (1, 3, 1280, 800)；(2) 一些运算操作，如 GN 和其他定制的运算操作没有加入到计算量的计算中。

14.2 发布模型

在您上传一个模型到云服务器之前，您需要做以下几步：(1) 将模型权重转成 CPU 张量；(2) 删除记录优化器状态 (optimizer states) 的相关信息；(3) 计算检查点文件 (checkpoint file) 的哈希编码（hash id）并且将哈希编码加到文件名中。

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

例如，

```
python tools/publish_model.py work_dirs/pspnet/latest.pth psp_r50_hszhao_200ep.pth
```

最终输出文件将是 `psp_r50_512x1024_40ki_cityscapes-{hash id}.pth`。

14.3 导出 ONNX (试验性)

我们提供了一个脚本来导出模型到 ONNX 格式。被转换的模型可以通过工具 Netron 来可视化。除此以外，我们同样支持对 PyTorch 和 ONNX 模型的输出结果做对比。

```
python tools/pytorch2onnx.py \
    ${CONFIG_FILE} \
    --checkpoint ${CHECKPOINT_FILE} \
    --output-file ${ONNX_FILE} \
    --input-img ${INPUT_IMG} \
    --shape ${INPUT_SHAPE} \
    --rescale-shape ${RESCALE_SHAPE} \
    --show \
    --verify \
    --dynamic-export \
    --cfg-options \
      model.test_cfg.mode="whole"
```

各个参数的描述:

- `config` : 模型配置文件的路径
- `--checkpoint` : 模型检查点文件的路径
- `--output-file`: 输出的 ONNX 模型的路径。如果没有专门指定，它默认是 `tmp.onnx`
- `--input-img` : 用来转换和可视化的一张输入图像的路径
- `--shape`: 模型的输入张量的高和宽。如果没有专门指定，它将被设置成 `test_pipeline` 的 `img_scale`
- `--rescale-shape`: 改变输出的形状。设置这个值来避免 OOM，它仅在 `slide` 模式下可以用

- `--show`: 是否打印输出模型的结构。如果没有被专门指定，它将被设置成 `False`
- `--verify`: 是否验证一个输出模型的正确性 (correctness)。如果没有被专门指定，它将被设置成 `False`
- `--dynamic-export`: 是否导出形状变化的输入与输出的 ONNX 模型。如果没有被专门指定，它将被设置成 `False`
- `--cfg-options`: 更新配置选项

注意: 这个工具仍然是试验性的，目前一些自定义操作还没有被支持

14.4 评估 ONNX 模型

我们提供 `tools/deploy_test.py` 去评估不同后端的 ONNX 模型。

14.4.1 先决条件

- 安装 onnx 和 onnxruntime-gpu

  ```
  pip install onnx onnxruntime-gpu
  ```

- 参考 如何在 MMCV 里构建 tensorrt 插件 安装 TensorRT (可选)

14.4.2 使用方法

```
python tools/deploy_test.py \
    ${CONFIG_FILE} \
    ${MODEL_FILE} \
    ${BACKEND} \
    --out ${OUTPUT_FILE} \
    --eval ${EVALUATION_METRICS} \
    --show \
    --show-dir ${SHOW_DIRECTORY} \
    --cfg-options ${CFG_OPTIONS} \
    --eval-options ${EVALUATION_OPTIONS} \
    --opacity ${OPACITY} \
```

各个参数的描述:

- `config`: 模型配置文件的路径
- `model`: 被转换的模型文件的路径
- `backend`: 推理的后端，可选项：`onnxruntime`，`tensorrt`
- `--out`: 输出结果成 pickle 格式文件的路径

- `--format-only` : 不评估直接给输出结果的格式。通常用在当您想把结果输出成一些测试服务器需要的特定格式时。如果没有被专门指定，它将被设置成 `False`。注意这个参数是用 `--eval` 来 **手动添加**
- `--eval`: 评估指标，取决于每个数据集的要求，例如 "mIoU" 是大多数据集的指标而 "cityscapes" 仅针对 Cityscapes 数据集。注意这个参数是用 `--format-only` 来 **手动添加**
- `--show`: 是否展示结果
- `--show-dir`: 涂上结果的图像被保存的文件夹的路径
- `--cfg-options`: 重写配置文件里的一些设置，`xxx=yyy` 格式的键值对将被覆盖到配置文件里
- `--eval-options`: 自定义的评估的选项，`xxx=yyy` 格式的键值对将成为 `dataset.evaluate()` 函数的参数变量
- `--opacity`: 涂上结果的分割图的透明度，范围在 (0, 1] 之间

14.4.3 结果和模型

注意: TensorRT 仅在使用 `whole mode` 测试模式时的配置文件里可用。

14.5 导出 TorchScript (试验性)

我们同样提供一个脚本去把模型导出成 TorchScript 格式。您可以使用 pytorch C++ API LibTorch 去推理训练好的模型。被转换的模型能被像 Netron 的工具来可视化。此外，我们还支持 PyTorch 和 TorchScript 模型的输出结果的比较。

```
python tools/pytorch2torchscript.py \
    ${CONFIG_FILE} \
    --checkpoint ${CHECKPOINT_FILE} \
    --output-file ${ONNX_FILE}
    --shape ${INPUT_SHAPE}
    --verify \
    --show
```

各个参数的描述:

- `config` : pytorch 模型的配置文件的路径
- `--checkpoint` : pytorch 模型的检查点文件的路径
- `--output-file`: TorchScript 模型输出的路径，如果没有被专门指定，它将被设置成 `tmp.pt`
- `--input-img` : 用来转换和可视化的输入图像的路径
- `--shape`: 模型的输入张量的宽和高。如果没有被专门指定，它将被设置成 `512 512`
- `--show`: 是否打印输出模型的追踪图 (traced graph)，如果没有被专门指定，它将被设置成 `False`

- `--verify`: 是否验证一个输出模型的正确性 (correctness)，如果没有被专门指定，它将被设置成 `False`

注意: 目前仅支持 PyTorch>=1.8.0 版本

注意: 这个工具仍然是试验性的，一些自定义操作符目前还不被支持

例子:

- 导出 PSPNet 在 cityscapes 数据集上的 pytorch 模型

```
python tools/pytorch2torchscript.py configs/pspnet/pspnet_r50-d8_512x1024_40k_
↪cityscapes.py \
--checkpoint checkpoints/pspnet_r50-d8_512x1024_40k_cityscapes_20200605_003338-
↪2966598c.pth \
--output-file checkpoints/pspnet_r50-d8_512x1024_40k_cityscapes_20200605_003338-
↪2966598c.pt \
--shape 512 1024
```

14.6 导出 TensorRT (试验性)

一个导出 ONNX 模型成 TensorRT 格式的脚本

先决条件

- 按照 ONNXRuntime in mmcv 和 TensorRT plugin in mmcv ，用 ONNXRuntime 自定义运算 (custom ops) 和 TensorRT 插件安装 `mmcv-full`
- 使用 pytorch2onnx 将模型从 PyTorch 转成 ONNX

使用方法

```
python ${MMSEG_PATH}/tools/onnx2tensorrt.py \
    ${CFG_PATH} \
    ${ONNX_PATH} \
    --trt-file ${OUTPUT_TRT_PATH} \
    --min-shape ${MIN_SHAPE} \
    --max-shape ${MAX_SHAPE} \
    --input-img ${INPUT_IMG} \
    --show \
    --verify
```

各个参数的描述:

- `config` : 模型的配置文件
- `model` : 输入的 ONNX 模型的路径
- `--trt-file` : 输出的 TensorRT 引擎的路径
- `--max-shape` : 模型的输入的最大形状

- `--min-shape` : 模型的输入的最小形状
- `--fp16` : 做 fp16 模型转换
- `--workspace-size` : 在 GiB 里的最大工作空间大小 (Max workspace size)
- `--input-img` : 用来可视化的图像
- `--show` : 做结果的可视化
- `--dataset` : Palette provider, 默认为 `CityscapesDataset`
- `--verify` : 验证 ONNXRuntime 和 TensorRT 的输出
- `--verbose` : 当创建 TensorRT 引擎时，是否详细做信息日志。默认为 False

注意: 仅在全图测试模式 (whole mode) 下测试过

CHAPTER 15

其他内容

15.1 打印完整的配置文件

`tools/print_config.py` 会逐字逐句的打印整个配置文件，展开所有的导入。

```
python tools/print_config.py \
  ${CONFIG} \
  --graph \
  --cfg-options ${OPTIONS [OPTIONS...]} \
```

各个参数的描述:

- `config` : pytorch 模型的配置文件的路径
- `--graph` : 是否打印模型的图 (models graph)
- `--cfg-options`: 自定义替换配置文件的选项

15.2 对训练日志 (training logs) 画图

`tools/analyze_logs.py` 会画出给定的训练日志文件的 loss/mIoU 曲线，首先需要 `pip install seaborn` 安装依赖包。

```
python tools/analyze_logs.py xxx.log.json [--keys ${KEYS}] [--legend ${LEGEND}] [--
↪backend ${BACKEND}] [--style ${STYLE}] [--out ${OUT_FILE}]
```

示例:

- 对 mIoU, mAcc, aAcc 指标画图

```
python tools/analyze_logs.py log.json --keys mIoU mAcc aAcc --legend mIoU mAcc␣
↪aAcc
```

- 对 loss 指标画图

```
python tools/analyze_logs.py log.json --keys loss --legend loss
```

15.3 转换其他仓库的权重

`tools/model_converters/` 提供了若干个预训练权重转换脚本，支持将其他仓库的预训练权重的 key 转换为与 MMSegmentation 相匹配的 key。

15.3.1 ViT Swin MiT Transformer 模型

- ViT

`tools/model_converters/vit2mmseg.py` 将 timm 预训练模型转换到 MMSegmentation。

```
python tools/model_converters/vit2mmseg.py ${SRC} ${DST}
```

- Swin

  `tools/model_converters/swin2mmseg.py` 将官方预训练模型转换到 MMSegmentation。

  ```
  python tools/model_converters/swin2mmseg.py ${SRC} ${DST}
  ```

- SegFormer

  `tools/model_converters/mit2mmseg.py` 将官方预训练模型转换到 MMSegmentation。

  ```
  python tools/model_converters/mit2mmseg.py ${SRC} ${DST}
  ```

CHAPTER 16

模型服务

为了用 TorchServe 服务 MMSegmentation 的模型，您可以遵循如下流程:

16.1 1. 将 model 从 MMSegmentation 转换到 TorchServe

```
python tools/mmseg2torchserve.py ${CONFIG_FILE} ${CHECKPOINT_FILE} \
--output-folder ${MODEL_STORE} \
--model-name ${MODEL_NAME}
```

注意: ${MODEL_STORE} 需要设置为某个文件夹的绝对路径

16.2 2. 构建 mmseg-serve 容器镜像 (docker image)

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

16.3 3. 运行 mmseg-serve

请查阅官方文档: 使用容器运行 TorchServe

为了在 GPU 环境下使用, 您需要安装 nvidia-docker. 若在 CPU 环境下使用，您可以忽略添加 --gpus 参数。

示例:

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

阅读关于推理 (8080), 管理 (8081) 和指标 (8082) APIs 的 文档 。

16.4 4. 测试部署

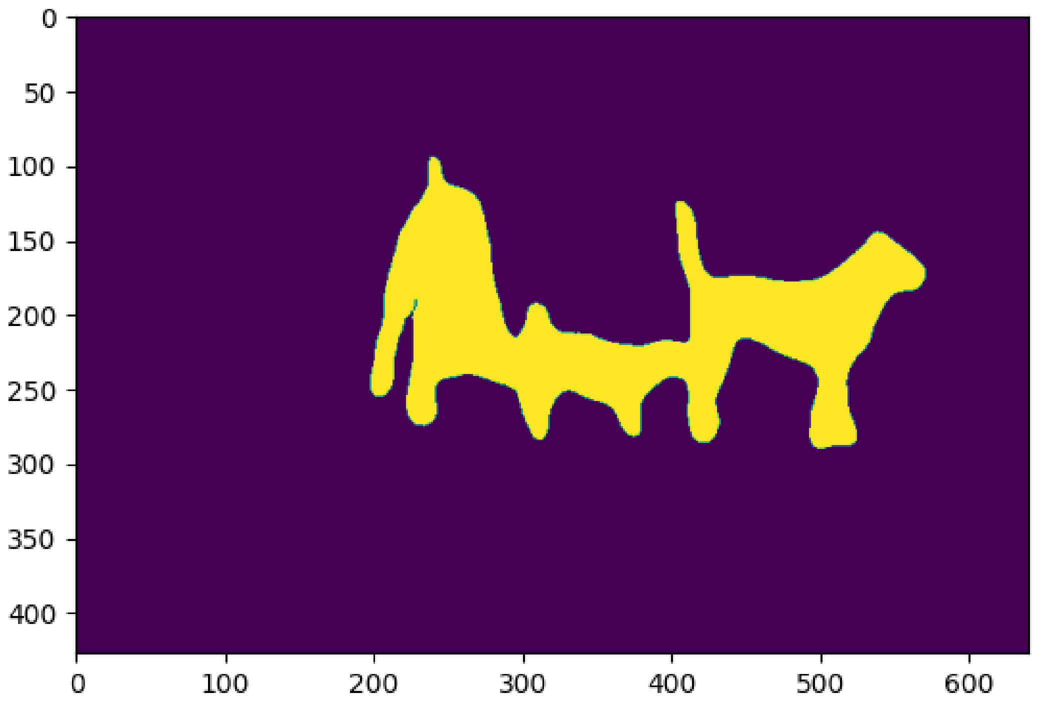
```
curl -O https://raw.githubusercontent.com/open-mmlab/mmsegmentation/master/resources/
↪3dogs.jpg
curl http://127.0.0.1:8080/predictions/${MODEL_NAME} -T 3dogs.jpg -o 3dogs_mask.png
```

得到的响应将是一个 “.png” 的分割掩码.

您可以按照如下方法可视化输出:

```
import matplotlib.pyplot as plt
import mmcv
plt.imshow(mmcv.imread("3dogs_mask.png", "grayscale"))
plt.show()
```

看到的东西将会和下图类似:



然后您可以使用 `test_torchserve.py` 比较 torchserve 和 pytorch 的结果，并将它们可视化。

```
python tools/torchserve/test_torchserve.py ${IMAGE_FILE} ${CONFIG_FILE} ${CHECKPOINT_
↪FILE} ${MODEL_NAME}
[--inference-addr ${INFERENCE_ADDR}] [--result-image ${RESULT_IMAGE}] [--device $
↪{DEVICE}]
```

示例：

```
python tools/torchserve/test_torchserve.py \
demo/demo.png \
configs/fcn/fcn_r50-d8_512x1024_40k_cityscapes.py \
checkpoint/fcn_r50-d8_512x1024_40k_cityscapes_20200604_192608-efe53f0d.pth \
fcn
```

CHAPTER 17

模型集成

我们提供了 tools/model_ensemble.py 完成对多个模型的预测概率进行集成的脚本

17.1 使用方法

```
python tools/model_ensemble.py \
  --config ${CONFIG_FILE1} ${CONFIG_FILE2} ... \
  --checkpoint ${CHECKPOINT_FILE1} ${CHECKPOINT_FILE2} ...\
  --aug-test \
  --out ${OUTPUT_DIR}\
  --gpus ${GPU_USED}\
```

17.2 各个参数的描述:

- --config: 集成模型的配置文件的路径
- --checkpoint: 集成模型的权重文件的路径
- --aug-test: 是否使用翻转和多尺度预测
- --out: 模型集成结果的保存文件夹路径
- --gpus: 模型集成使用的 gpu-id

17.3 模型集成结果

- 模型集成会对每一张输入，形状为 `[H, W]`，产生一张未渲染的分割掩膜文件 (segmentation mask)，形状为 `[H, W]`，分割掩膜中的每个像素点的值代表该位置分割后的像素类别.
- 模型集成结果的文件名会采用和 `Ground Truth` 一致的文件命名，如 `Ground Truth` 文件名称为 `1.png`，则模型集成结果文件也会被命名为 `1.png`，并放置在`--out` 指定的文件夹中.

CHAPTER 18

常见问题解答（FAQ）

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

18.1 安装

兼容的 MMSegmentation 和 MMCV 版本如下。请安装正确版本的 MMCV 以避免安装问题。

如果你安装了 mmcv，你需要先运行 `pip uninstall mmcv`。如果 mmcv 和 mmcv-full 都安装了，会出现 "ModuleNotFoundError"。

- "No module named 'mmcv.ops'"; "No module named 'mmcv._ext'".
    1. 使用 `pip uninstall mmcv` 卸载环境中现有的 mmcv。
    2. 按照安装说明安装 mmcv-full。

18.2 如何获知模型训练时需要的显卡数量

- 看模型的 config 文件的命名。可以参考学习配置文件中的配置文件命名风格部分。比如，对于名字为 `segformer_mit-b0_8x1_1024x1024_160k_cityscapes.py` 的 config 文件，`8x1` 代表训练其对应的模型需要的卡数为 8，每张卡中的 batch size 为 1。
- 看模型的 log 文件。点开该模型的 log 文件，并在其中搜索 `nGPU`，在 `nGPU` 后的数字个数即训练时所需的卡数。比如，在 log 文件中搜索 `nGPU` 得到 `nGPU 0,1,2,3,4,5,6,7` 的记录，则说明训练该模型需要使用八张卡。

18.3 auxiliary head 是什么

简单来说，这是一个提高准确率的深度监督技术。在训练阶段，`decode_head` 用于输出语义分割的结果，`auxiliary_head` 只是增加了一个辅助损失，其产生的分割结果对你的模型结果没有影响，仅在训练中起作用。你可以阅读这篇论文了解更多信息。

18.4 为什么日志文件没有被创建

在训练脚本中，我们在第 167 行调用 `get_root_logger` 方法，然后 mmseg 的 `get_root_logger` 方法调用 mmcv 的 `get_logger`，mmcv 将返回在 'mmsegmentation/tools/train.py' 中使用参数 `log_file` 初始化的同一个 logger。在训练期间只存在一个用 `log_file` 初始化的 logger。

参　考：https://github.com/open-mmlab/mmcv/blob/21bada32560c7ed7b15b017dc763d862789e29a8/mmcv/utils/logging.py#L9-L16

如果你发现日志文件没有被创建，可以检查 `mmcv.utils.get_logger` 是否在其他地方被调用。

18.5 运行测试脚本时如何输出绘制分割掩膜的图像

在测试脚本中，我们提供了 `show-dir` 参数来控制是否输出绘制的图像。用户可以运行以下命令:

```
python tools/test.py {config} {checkpoint} --show-dir {/path/to/save/image} --opacity␣
↪1
```

CHAPTER 19

English

CHAPTER 20

简体中文

CHAPTER 21

mmseg.apis

mmseg.apis.**get_root_logger**(*log_file=None*, *log_level=20*)

Get the root logger.

The logger will be initialized if it has not been initialized. By default a StreamHandler will be added. If *log_file* is specified, a FileHandler will also be added. The name of the root logger is the top-level package name, e.g., "mmseg".

参数

- **log_file** (*str | None*) –The log filename. If specified, a FileHandler will be added to the root logger.
- **log_level** (*int*) –The root logger level. Note that only the process of rank 0 is affected, while other processes will set the level to "Error" and be silent most of the time.

返回 The root logger.

返回类型 logging.Logger

mmseg.apis.**inference_segmentor**(*model*, *imgs*)

Inference image(s) with the segmentor.

参数

- **model** (*nn.Module*) –The loaded segmentor.
- **imgs** (*str/ndarray or list[str/ndarray]*) –Either image files or loaded images.

返回 The segmentation result.

返回类型 (list[Tensor])

`mmseg.apis.`**`init_random_seed`**(*seed=None*, *device='cuda'*)

Initialize random seed.

If the seed is not set, the seed will be automatically randomized, and then broadcast to all processes to prevent some potential bugs. :param seed: The seed. Default to None. :type seed: int, Optional :param device: The device where the seed will be put on.

Default to 'cuda'.

返回 Seed to be used.

返回类型 int

`mmseg.apis.`**`init_segmentor`**(*config*, *checkpoint=None*, *device='cuda:0'*)

Initialize a segmentor from config file.

参数

- **config** (str or `mmcv.Config`) –Config file path or the config object.
- **checkpoint** (*str, optional*) –Checkpoint path. If left as None, the model will not load any weights.
- **device** (*str, optional*) –0'. Use 'cpu' for loading model on CPU.

返回 The constructed segmentor.

返回类型 nn.Module

`mmseg.apis.`**`multi_gpu_test`**(*model*, *data_loader*, *tmpdir=None*, *gpu_collect=False*, *efficient_test=False*, *pre_eval=False*, *format_only=False*, *format_args={}*)

Test model with multiple gpus by progressive mode.

This method tests model with multiple gpus and collects the results under two different modes: gpu and cpu modes. By setting 'gpu_collect=True' it encodes results to gpu tensors and use gpu communication for results collection. On cpu mode it saves the results on different gpus to 'tmpdir' and collects them by the rank 0 worker.

参数

- **model** (*nn.Module*) –Model to be tested.
- **data_loader** (*utils.data.Dataloader*) –Pytorch data loader.
- **tmpdir** (*str*) –Path of directory to save the temporary results from different gpus under cpu mode. The same path is used for efficient test. Default: None.
- **gpu_collect** (*bool*) –Option to use either gpu or cpu to collect results. Default: False.
- **efficient_test** (*bool*) –Whether save the results as local numpy files to save CPU memory during evaluation. Mutually exclusive with pre_eval and format_results. Default: False.

- **pre_eval** (*bool*) –Use dataset.pre_eval() function to generate pre_results for metric evaluation. Mutually exclusive with efficient_test and format_results. Default: False.
- **format_only** (*bool*) –Only format result for results commit. Mutually exclusive with pre_eval and efficient_test. Default: False.
- **format_args** (*dict*) –The args for format_results. Default: {}.

返回 list of evaluation pre-results or list of save file names.

返回类型 list

`mmseg.apis.`**`set_random_seed`**(*seed*, *deterministic=False*)

Set random seed.

参数

- **seed** (*int*) –Seed to be used.
- **deterministic** (*bool*) –Whether to set the deterministic option for CUDNN backend, i.e., set *torch.backends.cudnn.deterministic* to True and *torch.backends.cudnn.benchmark* to False. Default: False.

`mmseg.apis.`**`show_result_pyplot`**(*model*, *img*, *result*, *palette=None*, *fig_size=(15, 10)*, *opacity=0.5*, *title="*, *block=True*, *out_file=None*)

Visualize the segmentation results on the image.

参数

- **model** (*nn.Module*) –The loaded segmentor.
- **img** (*str or np.ndarray*) –Image filename or loaded image.
- **result** (*list*) –The segmentation result.
- **palette** (*list[list[int]]] | None*) –The palette of segmentation map. If None is given, random palette will be generated. Default: None
- **fig_size** (*tuple*) –Figure size of the pyplot figure.
- **opacity** (*float*) –Opacity of painted segmentation map. Default 0.5. Must be in (0, 1] range.
- **title** (*str*) –The title of pyplot figure. Default is ".
- **block** (*bool*) –Whether to block the pyplot figure. Default is True.
- **out_file** (*str or None*) –The path to write the image. Default: None.

`mmseg.apis.`**`single_gpu_test`**(*model*, *data_loader*, *show=False*, *out_dir=None*, *efficient_test=False*, *opacity=0.5*, *pre_eval=False*, *format_only=False*, *format_args={}*)

Test with single GPU by progressive mode.

参数

- **model** (*nn.Module*) –Model to be tested.
- **data_loader** (*utils.data.Dataloader*) –Pytorch data loader.
- **show** (*bool*) –Whether show results during inference. Default: False.
- **out_dir** (*str, optional*) –If specified, the results will be dumped into the directory to save output results.
- **efficient_test** (*bool*) –Whether save the results as local numpy files to save CPU memory during evaluation. Mutually exclusive with pre_eval and format_results. Default: False.
- **opacity** (*float*) –Opacity of painted segmentation map. Default 0.5. Must be in (0, 1] range.
- **pre_eval** (*bool*) –Use dataset.pre_eval() function to generate pre_results for metric evaluation. Mutually exclusive with efficient_test and format_results. Default: False.
- **format_only** (*bool*) –Only format result for results commit. Mutually exclusive with pre_eval and efficient_test. Default: False.
- **format_args** (*dict*) –The args for format_results. Default: {}.

返回 list of evaluation pre-results or list of save file names.

返回类型 list

mmseg.apis.**train_segmentor**(*model*, *dataset*, *cfg*, *distributed=False*, *validate=False*, *timestamp=None*, *meta=None*)

Launch segmentor training.

CHAPTER 22

mmseg.core

22.1 seg

class mmseg.core.seg.**BasePixelSampler**(***kwargs*)

Base class of pixel sampler.

abstract sample(*seg_logit*, *seg_label*)

Placeholder for sample function.

class mmseg.core.seg.**OHEMPixelSampler**(*context*, *thresh=None*, *min_kept=100000*)

Online Hard Example Mining Sampler for segmentation.

参数

- **context** (*nn.Module*) –The context of sampler, subclass of `BaseDecodeHead`.
- **thresh** (*float, optional*) –The threshold for hard example selection. Below which, are prediction with low confidence. If not specified, the hard examples will be pixels of top `min_kept` loss. Default: None.
- **min_kept** (*int, optional*) –The minimum number of predictions to keep. Default: 100000.

sample(*seg_logit*, *seg_label*)

Sample pixels that have high loss or with low prediction confidence.

参数

- **seg_logit** (*torch.Tensor*) –segmentation logits, shape (N, C, H, W)

- **seg_label** (*torch.Tensor*) –segmentation label, shape (N, 1, H, W)

返回 segmentation weight, shape (N, H, W)

返回类型 torch.Tensor

`mmseg.core.seg.`**`build_pixel_sampler`**(*cfg*, ***default_args*)

Build pixel sampler for segmentation map.

22.2 evaluation

`class` `mmseg.core.evaluation.`**`DistEvalHook`**(**args*, *by_epoch=False*, *efficient_test=False*, *pre_eval=False*, ***kwargs*)

Distributed EvalHook, with efficient test support.

参数

- **by_epoch** (*bool*) –Determine perform evaluation by epoch or by iteration. If set to True, it will perform by epoch. Otherwise, by iteration. Default: False.
- **efficient_test** (*bool*) –Whether save the results as local numpy files to save CPU memory during evaluation. Default: False.
- **pre_eval** (*bool*) –Whether to use progressive mode to evaluate model. Default: False.

返回 The prediction results.

返回类型 list

`class` `mmseg.core.evaluation.`**`EvalHook`**(**args*, *by_epoch=False*, *efficient_test=False*, *pre_eval=False*, ***kwargs*)

Single GPU EvalHook, with efficient test support.

参数

- **by_epoch** (*bool*) –Determine perform evaluation by epoch or by iteration. If set to True, it will perform by epoch. Otherwise, by iteration. Default: False.
- **efficient_test** (*bool*) –Whether save the results as local numpy files to save CPU memory during evaluation. Default: False.
- **pre_eval** (*bool*) –Whether to use progressive mode to evaluate model. Default: False.

返回 The prediction results.

返回类型 list

`mmseg.core.evaluation.`**`eval_metrics`**(*results*, *gt_seg_maps*, *num_classes*, *ignore_index*, *metrics=['mIoU']*, *nan_to_num=None*, *label_map={}*, *reduce_zero_label=False*, *beta=1*)

Calculate evaluation metrics :param results: List of prediction segmentation maps or list of prediction result filenames.

参数

- **gt_seg_maps** (*list[ndarray] | list[str] | Iterables*) –list of ground truth segmentation maps or list of label filenames.
- **num_classes** (*int*) –Number of categories.
- **ignore_index** (*int*) –Index that will be ignored in evaluation.
- **metrics** (*list[str] | str*) –Metrics to be evaluated, ‘mIoU’ and ‘mDice’.
- **nan_to_num** (*int, optional*) –If specified, NaN values will be replaced by the numbers defined by the user. Default: None.
- **label_map** (*dict*) –Mapping old labels to new labels. Default: dict().
- **reduce_zero_label** –Whether ignore zero label. Default: False.

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

Get class names of a dataset.

mmseg.core.evaluation.**get_palette**(*dataset*)

Get class palette (RGB) of a dataset.

mmseg.core.evaluation.**intersect_and_union**(*pred_label*, *label*, *num_classes*, *ignore_index*, *label_map={}*, *reduce_zero_label=False*)

Calculate intersection and Union.

参数

- **pred_label** (*ndarray | str*) –Prediction segmentation map or predict result filename.
- **label** (*ndarray | str*) –Ground truth segmentation map or label filename.
- **num_classes** (*int*) –Number of categories.
- **ignore_index** (*int*) –Index that will be ignored in evaluation.
- **label_map** (*dict*) –Mapping old labels to new labels. The parameter will work only when label is str. Default: dict().
- **reduce_zero_label** –Whether ignore zero label. The parameter will work only when label is str. Default: False.

mmseg.core.evaluation.**mean_dice**(*results*, *gt_seg_maps*, *num_classes*, *ignore_index*, *nan_to_num=None*, *label_map={}*, *reduce_zero_label=False*)

Calculate Mean Dice (mDice)

参数

- **results** (*list[ndarray] | list[str]*) –List of prediction segmentation maps or list of prediction result filenames.

- **gt_seg_maps** (*list[ndarray] | list[str]*) –list of ground truth segmentation maps or list of label filenames.
- **num_classes** (*int*) –Number of categories.
- **ignore_index** (*int*) –Index that will be ignored in evaluation.
- **nan_to_num** (*int, optional*) –If specified, NaN values will be replaced by the numbers defined by the user. Default: None.
- **label_map** (*dict*) –Mapping old labels to new labels. Default: dict().
- **reduce_zero_label** –Whether ignore zero label. Default: False.

mmseg.core.evaluation.**mean_fscore**(*results*, *gt_seg_maps*, *num_classes*, *ignore_index*, *nan_to_num=None*, *label_map={}*, *reduce_zero_label=False*, *beta=1*)

Calculate Mean F-Score (mFscore)

参数

- **results** (*list[ndarray] | list[str]*) –List of prediction segmentation maps or list of prediction result filenames.
- **gt_seg_maps** (*list[ndarray] | list[str]*) –list of ground truth segmentation maps or list of label filenames.
- **num_classes** (*int*) –Number of categories.
- **ignore_index** (*int*) –Index that will be ignored in evaluation.
- **nan_to_num** (*int, optional*) –If specified, NaN values will be replaced by the numbers defined by the user. Default: None.
- **label_map** (*dict*) –Mapping old labels to new labels. Default: dict().
- **reduce_zero_label** (*bool*) –Whether ignore zero label. Default: False.
- **beta** –Determines the weight of recall in the combined score. Default: False.

mmseg.core.evaluation.**mean_iou**(*results*, *gt_seg_maps*, *num_classes*, *ignore_index*, *nan_to_num=None*, *label_map={}*, *reduce_zero_label=False*)

Calculate Mean Intersection and Union (mIoU)

参数

- **results** (*list[ndarray] | list[str]*) –List of prediction segmentation maps or list of prediction result filenames.
- **gt_seg_maps** (*list[ndarray] | list[str]*) –list of ground truth segmentation maps or list of label filenames.
- **num_classes** (*int*) –Number of categories.

- **ignore_index** (*int*) –Index that will be ignored in evaluation.
- **nan_to_num** (*int, optional*) –If specified, NaN values will be replaced by the numbers defined by the user. Default: None.
- **label_map** (*dict*) –Mapping old labels to new labels. Default: dict().
- **reduce_zero_label** –Whether ignore zero label. Default: False.

`mmseg.core.evaluation.`**pre_eval_to_metrics**(*pre_eval_results*, *metrics=['mIoU']*, *nan_to_num=None*, *beta=1*)

Convert pre-eval results to metrics.

参数

- **pre_eval_results** (*list[tuple[torch.Tensor]]*) –per image eval results for computing evaluation metric
- **metrics** (*list[str] | str*) –Metrics to be evaluated, 'mIoU' and 'mDice'.
- **nan_to_num** –If specified, NaN values will be replaced by the numbers defined by the user. Default: None.

22.3 utils

`mmseg.core.utils.`**add_prefix**(*inputs*, *prefix*)

Add prefix for dict.

参数

- **inputs** (*dict*) –The input dict with str keys.
- **prefix** (*str*) –The prefix to add.

返回 The dict with keys updated with `prefix`.

返回类型 dict

`mmseg.core.utils.`**sync_random_seed**(*seed=None*, *device='cuda'*)

Make sure different ranks share the same seed. All workers must call this function, otherwise it will deadlock. This method is generally used in *DistributedSampler*, because the seed should be identical across all processes in the distributed group.

In distributed sampling, different ranks should sample non-overlapped data in the dataset. Therefore, this function is used to make sure that each rank shuffles the data indices in the same order based on the same seed. Then different ranks could use different indices to select non-overlapped data from the same data list.

参数

- **seed** (*int, Optional*) –The seed. Default to None.
- **device** (*str*) –The device where the seed will be put on. Default to 'cuda'.

返回 Seed to be used.

返回类型 int

CHAPTER 23

mmseg.datasets

23.1 datasets

class mmseg.datasets.**ADE20KDataset**(***kwargs*)

ADE20K dataset.

In segmentation map annotation for ADE20K, 0 stands for background, which is not included in 150 categories. `reduce_zero_label` is fixed to True. The `img_suffix` is fixed to ‘.jpg’ and `seg_map_suffix` is fixed to ‘.png’.

format_results(*results*, *imgfile_prefix*, *to_label_id=True*, *indices=None*)

Format the results into dir (standard format for ade20k evaluation).

参数

- **results** (*list*) –Testing results of the dataset.
- **imgfile_prefix** (*str | None*) –The prefix of images files. It includes the file path and the prefix of filename, e.g., “a/b/prefix”.
- **to_label_id** (*bool*) –whether convert output to label_id for submission. Default: False
- **indices** (*list[int], optional*) –Indices of input results, if not set, all the indices of the dataset will be used. Default: None.

返回

(result_files, tmp_dir), result_files is a list containing

the image paths, tmp_dir is the temporal directory created for saving json/png files when img_prefix is not specified.

返回类型 tuple

results2img (*results*, *imgfile_prefix*, *to_label_id*, *indices=None*)

Write the segmentation results to images.

参数

- **results** (*list[ndarray]*) –Testing results of the dataset.
- **imgfile_prefix** (*str*) –The filename prefix of the png files. If the prefix is "somepath/xxx", the png files will be named "somepath/xxx.png".
- **to_label_id** (*bool*) –whether convert output to label_id for submission.
- **indices** (*list[int], optional*) –Indices of input results, if not set, all the indices of the dataset will be used. Default: None.

返回 str]: result txt files which contains corresponding semantic segmentation images.

返回类型 list[str

class mmseg.datasets.**COCOStuffDataset** (***kwargs*)

COCO-Stuff dataset.

In segmentation map annotation for COCO-Stuff, Train-IDs of the 10k version are from 1 to 171, where 0 is the ignore index, and Train-ID of COCO Stuff 164k is from 0 to 170, where 255 is the ignore index. So, they are all 171 semantic categories. `reduce_zero_label` is set to True and False for the 10k and 164k versions, respectively. The `img_suffix` is fixed to '.jpg', and `seg_map_suffix` is fixed to '.png'.

class mmseg.datasets.**ChaseDB1Dataset** (***kwargs*)

Chase_db1 dataset.

In segmentation map annotation for Chase_db1, 0 stands for background, which is included in 2 categories. `reduce_zero_label` is fixed to False. The `img_suffix` is fixed to '.png' and `seg_map_suffix` is fixed to '_1stHO.png'.

class mmseg.datasets.**CityscapesDataset** (*img_suffix='_leftImg8bit.png'*, *seg_map_suffix='_gtFine_labelTrainIds.png'*, ***kwargs*)

Cityscapes dataset.

The `img_suffix` is fixed to '_leftImg8bit.png' and `seg_map_suffix` is fixed to '_gtFine_labelTrainIds.png' for Cityscapes dataset.

evaluate (*results*, *metric='mIoU'*, *logger=None*, *imgfile_prefix=None*)

Evaluation in Cityscapes/default protocol.

参数

- **results** (*list*) –Testing results of the dataset.

- **metric** (*str | list[str]*) –Metrics to be evaluated.
- **logger** (*logging.Logger | None | str*) –Logger used for printing related information during evaluation. Default: None.
- **imgfile_prefix** (*str | None*) –The prefix of output image file, for cityscapes evaluation only. It includes the file path and the prefix of filename, e.g., "a/b/prefix". If results are evaluated with cityscapes protocol, it would be the prefix of output png files. The output files would be png images under folder "a/b/prefix/xxx.png", where "xxx" is the image name of cityscapes. If not specified, a temp file will be created for evaluation. Default: None.

返回 Cityscapes/default metrics.

返回类型 dict[str, float]

format_results (*results*, *imgfile_prefix*, *to_label_id=True*, *indices=None*)

Format the results into dir (standard format for Cityscapes evaluation).

参数

- **results** (*list*) –Testing results of the dataset.
- **imgfile_prefix** (*str*) –The prefix of images files. It includes the file path and the prefix of filename, e.g., "a/b/prefix".
- **to_label_id** (*bool*) –whether convert output to label_id for submission. Default: False
- **indices** (*list[int], optional*) –Indices of input results, if not set, all the indices of the dataset will be used. Default: None.

返回

(result_files, tmp_dir), result_files is a list containing the image paths, tmp_dir is the temporal directory created for saving json/png files when img_prefix is not specified.

返回类型 tuple

results2img (*results*, *imgfile_prefix*, *to_label_id*, *indices=None*)

Write the segmentation results to images.

参数

- **results** (*list[ndarray]*) –Testing results of the dataset.
- **imgfile_prefix** (*str*) –The filename prefix of the png files. If the prefix is "somepath/xxx", the png files will be named "somepath/xxx.png".
- **to_label_id** (*bool*) –whether convert output to label_id for submission.
- **indices** (*list[int], optional*) –Indices of input results, if not set, all the indices of the dataset will be used. Default: None.

返回 str]: result txt files which contains corresponding semantic segmentation images.

返回类型 list[str

class `mmseg.datasets.`**`ConcatDataset`**(*datasets*, *separate_eval=True*)

A wrapper of concatenated dataset.

Same as `torch.utils.data.dataset.ConcatDataset`, but support evaluation and formatting results

参数

- **`datasets`** (list[`Dataset`]) –A list of datasets.
- **`separate_eval`** (*`bool`*) –Whether to evaluate the concatenated dataset results separately, Defaults to True.

`evaluate`(*results*, *logger=None*, ***kwargs*)

Evaluate the results.

参数

- **`results`** (*`list[tuple[torch.Tensor]] | list[str]]`*) –per image pre_eval results or predict segmentation map for computing evaluation metric.
- **`logger`** (*`logging.Logger | str | None`*) –Logger used for printing related information during evaluation. Default: None.

返回

float]: evaluate results of the total dataset or each separate

dataset if *self.separate_eval=True*.

返回类型 dict[str

`format_results`(*results*, *imgfile_prefix*, *indices=None*, ***kwargs*)

format result for every sample of ConcatDataset.

`get_dataset_idx_and_sample_idx`(*indice*)

Return dataset and sample index when given an indice of ConcatDataset.

参数 **`indice`** (*`int`*) –indice of sample in ConcatDataset

返回 the index of sub dataset the sample belong to int: the index of sample in its corresponding subset

返回类型 int

`pre_eval`(*preds*, *indices*)

do pre eval for every sample of ConcatDataset.

class mmseg.datasets.**CustomDataset**(*pipeline*, *img_dir*, *img_suffix='.jpg'*, *ann_dir=None*, *seg_map_suffix='.png'*, *split=None*, *data_root=None*, *test_mode=False*, *ignore_index=255*, *reduce_zero_label=False*, *classes=None*, *palette=None*, *gt_seg_map_loader_cfg=None*, *file_client_args={'backend': 'disk'}*)

Custom dataset for semantic segmentation. An example of file structure is as followed.

```
├── data
│   ├── my_dataset
│   │   ├── img_dir
│   │   │   ├── train
│   │   │   │   ├── xxx{img_suffix}
│   │   │   │   ├── yyy{img_suffix}
│   │   │   │   ├── zzz{img_suffix}
│   │   │   ├── val
│   │   ├── ann_dir
│   │   │   ├── train
│   │   │   │   ├── xxx{seg_map_suffix}
│   │   │   │   ├── yyy{seg_map_suffix}
│   │   │   │   ├── zzz{seg_map_suffix}
│   │   │   ├── val
```

The img/gt_semantic_seg pair of CustomDataset should be of the same except suffix. A valid img/gt_semantic_seg filename pair should be like `xxx{img_suffix}` and `xxx{seg_map_suffix}` (extension is also included in the suffix). If split is given, then `xxx` is specified in txt file. Otherwise, all files in `img_dir/``and ``ann_dir` will be loaded. Please refer to `docs/en/tutorials/new_dataset.md` for more details.

参数

- **pipeline** (*list[dict]*) –Processing pipeline
- **img_dir** (*str*) –Path to image directory
- **img_suffix** (*str*) –Suffix of images. Default: '.jpg'
- **ann_dir** (*str, optional*) –Path to annotation directory. Default: None
- **seg_map_suffix** (*str*) –Suffix of segmentation maps. Default: '.png'
- **split** (*str, optional*) –Split txt file. If split is specified, only file with suffix in the splits will be loaded. Otherwise, all images in img_dir/ann_dir will be loaded. Default: None
- **data_root** (*str, optional*) –Data root for img_dir/ann_dir. Default: None.
- **test_mode** (*bool*) –If test_mode=True, gt wouldn't be loaded.
- **ignore_index** (*int*) –The label index to be ignored. Default: 255
- **reduce_zero_label** (*bool*) –Whether to mark label zero as ignored. Default: False

- **classes** (*str | Sequence[str], optional*) –Specify classes to load. If is None, `cls.CLASSES` will be used. Default: None.
- **palette** (*Sequence[Sequence[int]]] | np.ndarray | None*) –The palette of segmentation map. If None is given, and self.PALETTE is None, random palette will be generated. Default: None
- **gt_seg_map_loader_cfg** (*dict, optional*) –build LoadAnnotations to load gt for evaluation, load from disk by default. Default: None.
- **file_client_args** (*dict*) –Arguments to instantiate a FileClient. See `mmcv.fileio.FileClient` for details. Defaults to `dict(backend='disk')`.

evaluate (*results*, *metric='mIoU'*, *logger=None*, *gt_seg_maps=None*, ***kwargs*)

Evaluate the dataset.

参数

- **results** (*list[tuple[torch.Tensor]] | list[str]*) –per image pre_eval results or predict segmentation map for computing evaluation metric.
- **metric** (*str | list[str]*) –Metrics to be evaluated. ‘mIoU’, ‘mDice’ and ‘mFscore’ are supported.
- **logger** (*logging.Logger | None | str*) –Logger used for printing related information during evaluation. Default: None.
- **gt_seg_maps** (*generator[ndarray]*) –Custom gt seg maps as input, used in ConcatDataset

返回 Default metrics.

返回类型 dict[str, float]

format_results (*results*, *imgfile_prefix*, *indices=None*, ***kwargs*)

Place holder to format result to dataset specific output.

get_ann_info (*idx*)

Get annotation by index.

参数 idx (*int*) –Index of data.

返回 Annotation info of specified index.

返回类型 dict

get_classes_and_palette (*classes=None*, *palette=None*)

Get class names of current dataset.

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.

- **palette** (*Sequence[Sequence[int]] | np.ndarray | None*) –The palette of segmentation map. If None is given, random palette will be generated. Default: None

get_gt_seg_map_by_idx(*index*)

Get one ground truth segmentation map for evaluation.

get_gt_seg_maps(*efficient_test=None*)

Get ground truth segmentation maps for evaluation.

load_annotations(*img_dir*, *img_suffix*, *ann_dir*, *seg_map_suffix*, *split*)

Load annotation from directory.

参数

- **img_dir** (*str*) –Path to image directory
- **img_suffix** (*str*) –Suffix of images.
- **ann_dir** (*str|None*) –Path to annotation directory.
- **seg_map_suffix** (*str|None*) –Suffix of segmentation maps.
- **split** (*str|None*) –Split txt file. If split is specified, only file with suffix in the splits will be loaded. Otherwise, all images in img_dir/ann_dir will be loaded. Default: None

返回 All image info of dataset.

返回类型 list[dict]

pre_eval(*preds*, *indices*)

Collect eval result from each iteration.

参数

- **preds** (*list[torch.Tensor] | torch.Tensor*) –the segmentation logit after argmax, shape (N, H, W).
- **indices** (*list[int] | int*) –the prediction related ground truth indices.

返回

(area_intersect, area_union, area_prediction, area_ground_truth).

返回类型 list[torch.Tensor]

pre_pipeline(*results*)

Prepare results dict for pipeline.

prepare_test_img(*idx*)

Get testing data after pipeline.

参数 **`idx`** (*`int`*) –Index of data.

返回

Testing data after pipeline with new keys introduced by pipeline.

返回类型 dict

`prepare_train_img`(*idx*)

Get training data and annotations after pipeline.

参数 **`idx`** (*`int`*) –Index of data.

返回

Training data and annotation after pipeline with new keys introduced by pipeline.

返回类型 dict

`class` `mmseg.datasets.`**`DRIVEDataset`**(***kwargs*)

DRIVE dataset.

In segmentation map annotation for DRIVE, 0 stands for background, which is included in 2 categories. `reduce_zero_label` is fixed to False. The `img_suffix` is fixed to '.png' and `seg_map_suffix` is fixed to '_manual1.png'.

`class` `mmseg.datasets.`**`DarkZurichDataset`**(***kwargs*)

DarkZurichDataset dataset.

`class` `mmseg.datasets.`**`HRFDataset`**(***kwargs*)

HRF dataset.

In segmentation map annotation for HRF, 0 stands for background, which is included in 2 categories. `reduce_zero_label` is fixed to False. The `img_suffix` is fixed to '.png' and `seg_map_suffix` is fixed to '.png'.

`class` `mmseg.datasets.`**`ISPRSDataset`**(***kwargs*)

ISPRS dataset.

In segmentation map annotation for LoveDA, 0 is the ignore index. `reduce_zero_label` should be set to True. The `img_suffix` and `seg_map_suffix` are both fixed to '.png'.

`class` `mmseg.datasets.`**`LoveDADataset`**(***kwargs*)

LoveDA dataset.

In segmentation map annotation for LoveDA, 0 is the ignore index. `reduce_zero_label` should be set to True. The `img_suffix` and `seg_map_suffix` are both fixed to '.png'.

`format_results`(*results*, *imgfile_prefix*, *indices=None*)

Format the results into dir (standard format for LoveDA evaluation).

参数

- **`results`** (*`list`*) –Testing results of the dataset.

- **imgfile_prefix** (*str*) –The prefix of images files. It includes the file path and the prefix of filename, e.g., "a/b/prefix".
- **indices** (*list[int], optional*) –Indices of input results, if not set, all the indices of the dataset will be used. Default: None.

返回

(result_files, tmp_dir), result_files is a list containing the image paths, tmp_dir is the temporal directory created for saving json/png files when img_prefix is not specified.

返回类型 tuple

results2img (*results*, *imgfile_prefix*, *indices=None*)

Write the segmentation results to images.

参数

- **results** (*list[ndarray]*) –Testing results of the dataset.
- **imgfile_prefix** (*str*) –The filename prefix of the png files. If the prefix is "somepath/xxx", the png files will be named "somepath/xxx.png".
- **indices** (*list[int], optional*) –Indices of input results, if not set, all the indices of the dataset will be used. Default: None.

返回 str]: result txt files which contains corresponding semantic segmentation images.

返回类型 list[str

class mmseg.datasets.**MultiImageMixDataset** (*dataset*, *pipeline*, *skip_type_keys=None*)

A wrapper of multiple images mixed dataset.

Suitable for training on multiple images mixed data augmentation like mosaic and mixup. For the augmentation pipeline of mixed image data, the *get_indexes* method needs to be provided to obtain the image indexes, and you can set *skip_flags* to change the pipeline running process.

参数

- **dataset** (*CustomDataset*) –The dataset to be mixed.
- **pipeline** (*Sequence[dict]*) –Sequence of transform object or config dict to be composed.
- **skip_type_keys** (*list[str], optional*) –Sequence of type string to be skip pipeline. Default to None.

update_skip_type_keys (*skip_type_keys*)

Update skip_type_keys.

It is called by an external hook.

参数 skip_type_keys (*list[str], optional*) –Sequence of type string to be skip pipeline.

class mmseg.datasets.**NightDrivingDataset**(***kwargs*)

NightDrivingDataset dataset.

class mmseg.datasets.**PascalContextDataset**(*split*, ***kwargs*)

PascalContext dataset.

In segmentation map annotation for PascalContext, 0 stands for background, which is included in 60 categories. `reduce_zero_label` is fixed to False. The `img_suffix` is fixed to '.jpg' and `seg_map_suffix` is fixed to '.png'.

参数 **split** (*str*) –Split txt file for PascalContext.

class mmseg.datasets.**PascalContextDataset59**(*split*, ***kwargs*)

PascalContext dataset.

In segmentation map annotation for PascalContext, 0 stands for background, which is included in 60 categories. `reduce_zero_label` is fixed to False. The `img_suffix` is fixed to '.jpg' and `seg_map_suffix` is fixed to '.png'.

参数 **split** (*str*) –Split txt file for PascalContext.

class mmseg.datasets.**PascalVOCDataset**(*split*, ***kwargs*)

Pascal VOC dataset.

参数 **split** (*str*) –Split txt file for Pascal VOC.

class mmseg.datasets.**PotsdamDataset**(***kwargs*)

ISPRS Potsdam dataset.

In segmentation map annotation for Potsdam dataset, 0 is the ignore index. `reduce_zero_label` should be set to True. The `img_suffix` and `seg_map_suffix` are both fixed to '.png'.

class mmseg.datasets.**RepeatDataset**(*dataset*, *times*)

A wrapper of repeated dataset.

The length of repeated dataset will be *times* larger than the original dataset. This is useful when the data loading time is long but the dataset is small. Using RepeatDataset can reduce the data loading time between epochs.

参数

- **dataset** (Dataset) –The dataset to be repeated.
- **times** (*int*) –Repeat times.

class mmseg.datasets.**STAREDataset**(***kwargs*)

STARE dataset.

In segmentation map annotation for STARE, 0 stands for background, which is included in 2 categories. `reduce_zero_label` is fixed to False. The `img_suffix` is fixed to '.png' and `seg_map_suffix` is fixed to '.ah.png'.

`mmseg.datasets.`**`build_dataloader`**(*dataset*, *samples_per_gpu*, *workers_per_gpu*, *num_gpus=1*, *dist=True*, *shuffle=True*, *seed=None*, *drop_last=False*, *pin_memory=True*, *persistent_workers=True*, ***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 | None`*) –Seed to be used. Default: None.
- **`drop_last`** (*`bool`*) –Whether to drop the last incomplete batch in epoch. Default: False
- **`pin_memory`** (*`bool`*) –Whether to use pin_memory in DataLoader. Default: True
- **`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. The argument also has effect in PyTorch>=1.7.0. Default: True
- **`kwargs`** –any keyword argument to be used to initialize DataLoader

返回 A PyTorch dataloader.

返回类型 DataLoader

`mmseg.datasets.`**`build_dataset`**(*cfg*, *default_args=None*)

Build datasets.

`class` `mmseg.datasets.`**`iSAIDDataset`**(***kwargs*)

iSAID: A Large-scale Dataset for Instance Segmentation in Aerial Images In segmentation map annotation for iSAID dataset, which is included in 16 categories. `reduce_zero_label` is fixed to False. The `img_suffix` is fixed to '.png' and `seg_map_suffix` is fixed to '_manual1.png'.

`load_annotations`(*img_dir*, *img_suffix*, *ann_dir*, *seg_map_suffix=None*, *split=None*)

Load annotation from directory.

参数

- **`img_dir`** (*`str`*) –Path to image directory

- **img_suffix** (*str*) –Suffix of images.
- **ann_dir** (*str | None*) –Path to annotation directory.
- **seg_map_suffix** (*str | None*) –Suffix of segmentation maps.
- **split** (*str | None*) –Split txt file. If split is specified, only file with suffix in the splits will be loaded. Otherwise, all images in img_dir/ann_dir will be loaded. Default: None

返回 All image info of dataset.

返回类型 list[dict]

23.2 pipelines

class mmseg.datasets.pipelines.**AdjustGamma**(*gamma=1.0*)

Using gamma correction to process the image.

参数 **gamma** (*float or int*) –Gamma value used in gamma correction. Default: 1.0.

class mmseg.datasets.pipelines.**CLAHE**(*clip_limit=40.0*, *tile_grid_size=(8, 8)*)

Use CLAHE method to process the image.

See *ZUIDERVELD,K. Contrast Limited Adaptive Histogram Equalization[J]. Graphics Gems, 1994:474-485.* for more information.

参数

- **clip_limit** (*float*) –Threshold for contrast limiting. Default: 40.0.
- **tile_grid_size** (*tuple[int]*) –Size of grid for histogram equalization. Input image will be divided into equally sized rectangular tiles. It defines the number of tiles in row and column. Default: (8, 8).

class mmseg.datasets.pipelines.**Collect**(*keys*, *meta_keys=('filename', 'ori_filename', 'ori_shape', 'img_shape', 'pad_shape', 'scale_factor', 'flip', 'flip_direction', 'img_norm_cfg')*)

Collect data from the loader relevant to the specific task.

This is usually the last stage of the data loader pipeline. Typically keys is set to some subset of "img", "gt_semantic_seg".

The "img_meta" item is always populated. The contents of the "img_meta" dictionary depends on "meta_keys". By default this includes:

- **"img_shape": shape of the image input to the network as a tuple** (h, w, c). Note that images may be zero padded on the bottom/right if the batch tensor is larger than this shape.
- "scale_factor": a float indicating the preprocessing scale
- "flip": a boolean indicating if image flip transform was used

- “filename”: path to the image file
- “ori_shape”: original shape of the image as a tuple (h, w, c)
- “pad_shape”: image shape after padding
- **“img_norm_cfg”: a dict of normalization information:**
  - mean - per channel mean subtraction
  - std - per channel std divisor
  - to_rgb - bool indicating if bgr was converted to rgb

参数

- **keys** (*Sequence[str]*) –Keys of results to be collected in `data`.
- **meta_keys** (*Sequence[str], optional*) –Meta keys to be converted to `mmcv.DataContainer` and collected in `data[img_metas]`. Default: (`filename`, `ori_filename`, `ori_shape`, `img_shape`, `pad_shape`, `scale_factor`, `flip`, `flip_direction`, `img_norm_cfg`)

class `mmseg.datasets.pipelines.`**Compose**(*transforms*)

Compose multiple transforms sequentially.

参数 transforms (*Sequence[dict | callable]*) –Sequence of transform object or config dict to be composed.

class `mmseg.datasets.pipelines.`**ImageToTensor**(*keys*)

Convert image to `torch.Tensor` by given keys.

The dimension order of input image is (H, W, C). The pipeline will convert it to (C, H, W). If only 2 dimension (H, W) is given, the output would be (1, H, W).

参数 keys (*Sequence[str]*) –Key of images to be converted to Tensor.

class `mmseg.datasets.pipelines.`**LoadAnnotations**(*reduce_zero_label=False, file_client_args={'backend': 'disk'}, imdecode_backend='pillow'*)

Load annotations for semantic segmentation.

参数

- **reduce_zero_label** (*bool*) –Whether reduce all label value by 1. Usually used for datasets where 0 is background label. Default: False.
- **file_client_args** (*dict*) –Arguments to instantiate a FileClient. See `mmcv.fileio.FileClient` for details. Defaults to `dict(backend='disk')`.
- **imdecode_backend** (*str*) –Backend for `mmcv.imdecode()`. Default: ‘pillow’

class mmseg.datasets.pipelines.**LoadImageFromFile**(*to_float32=False*, *color_type='color'*, *file_client_args={'backend': 'disk'}*, *imdecode_backend='cv2'*)

Load an image from file.

Required keys are "img_prefix" and "img_info" (a dict that must contain the key "filename"). Added or updated keys are "filename" , "img" , "img_shape" , "ori_shape" (same as *img_shape*), "pad_shape" (same as *img_shape*), "scale_factor" (1.0) and "img_norm_cfg" (means=0 and stds=1).

参数

- **to_float32** (*bool*) –Whether to convert the loaded image to a float32 numpy array. If set to False, the loaded image is an uint8 array. Defaults to False.
- **color_type** (*str*) –The flag argument for `mmcv.imfrombytes()`. Defaults to 'color' .
- **file_client_args** (*dict*) –Arguments to instantiate a FileClient. See `mmcv.fileio.FileClient` for details. Defaults to `dict(backend='disk')`.
- **imdecode_backend** (*str*) –Backend for `mmcv.imdecode()`. Default: 'cv2'

class mmseg.datasets.pipelines.**MultiScaleFlipAug**(*transforms*, *img_scale*, *img_ratios=None*, *flip=False*, *flip_direction='horizontal'*)

Test-time augmentation with multiple scales and flipping.

An example configuration is as followed:

```
img_scale=(2048, 1024),
img_ratios=[0.5, 1.0],
flip=True,
transforms=[
    dict(type='Resize', keep_ratio=True),
    dict(type='RandomFlip'),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='Pad', size_divisor=32),
    dict(type='ImageToTensor', keys=['img']),
    dict(type='Collect', keys=['img']),
]
```

After MultiScaleFLipAug with above configuration, the results are wrapped into lists of the same length as followed:

```
dict(
    img=[...],
    img_shape=[...],
    scale=[(1024, 512), (1024, 512), (2048, 1024), (2048, 1024)]
    flip=[False, True, False, True]
```

(下页继续)

(续上页)

```
    ...
)
```

参数

- **transforms** (*list[dict]*) –Transforms to apply in each augmentation.
- **img_scale** (*None | tuple | list[tuple]*) –Images scales for resizing.
- **img_ratios** (*float | list[float]*) –Image ratios for resizing
- **flip** (*bool*) –Whether apply flip augmentation. Default: False.
- **flip_direction** (*str | list[str]*) –Flip augmentation directions, options are "horizontal" and "vertical". If flip_direction is list, multiple flip augmentations will be applied. It has no effect when flip == False. Default: "horizontal".

class mmseg.datasets.pipelines.**Normalize**(*mean*, *std*, *to_rgb=True*)

Normalize the image.

Added key is "img_norm_cfg".

参数

- **mean** (*sequence*) –Mean values of 3 channels.
- **std** (*sequence*) –Std values of 3 channels.
- **to_rgb** (*bool*) –Whether to convert the image from BGR to RGB, default is true.

class mmseg.datasets.pipelines.**Pad**(*size=None*, *size_divisor=None*, *pad_val=0*, *seg_pad_val=255*)

Pad the image & mask.

There are two padding modes: (1) pad to a fixed size and (2) pad to the minimum size that is divisible by some number. Added keys are "pad_shape", "pad_fixed_size", "pad_size_divisor",

参数

- **size** (*tuple, optional*) –Fixed padding size.
- **size_divisor** (*int, optional*) –The divisor of padded size.
- **pad_val** (*float, optional*) –Padding value. Default: 0.
- **seg_pad_val** (*float, optional*) –Padding value of segmentation map. Default: 255.

class mmseg.datasets.pipelines.**PhotoMetricDistortion**(*brightness_delta=32*, *contrast_range=(0.5, 1.5)*, *saturation_range=(0.5, 1.5)*, *hue_delta=18*)

Apply photometric distortion to image sequentially, every transformation is applied with a probability of 0.5. The position of random contrast is in second or second to last.

1. random brightness
2. random contrast (mode 0)
3. convert color from BGR to HSV
4. random saturation
5. random hue
6. convert color from HSV to BGR
7. random contrast (mode 1)

参数

- **brightness_delta** (*int*) –delta of brightness.
- **contrast_range** (*tuple*) –range of contrast.
- **saturation_range** (*tuple*) –range of saturation.
- **hue_delta** (*int*) –delta of hue.

brightness (*img*)

Brightness distortion.

contrast (*img*)

Contrast distortion.

convert (*img*, *alpha=1*, *beta=0*)

Multiple with alpha and add beat with clip.

hue (*img*)

Hue distortion.

saturation (*img*)

Saturation distortion.

class mmseg.datasets.pipelines.**RGB2Gray** (*out_channels=None*, *weights=(0.299, 0.587, 0.114)*)

Convert RGB image to grayscale image.

This transform calculate the weighted mean of input image channels with `weights` and then expand the channels to `out_channels`. When `out_channels` is None, the number of output channels is the same as input channels.

参数

- **out_channels** (*int*) –Expected number of output channels after transforming. Default: None.
- **weights** (*tuple[float]*) –The weights to calculate the weighted mean. Default: (0.299, 0.587, 0.114).

class mmseg.datasets.pipelines.**RandomCrop**(*crop_size*, *cat_max_ratio=1.0*, *ignore_index=255*)

Random crop the image & seg.

参数

- **crop_size** (*tuple*) –Expected size after cropping, (h, w).
- **cat_max_ratio** (*float*) –The maximum ratio that single category could occupy.

crop(*img*, *crop_bbox*)

Crop from `img`

get_crop_bbox(*img*)

Randomly get a crop bounding box.

class mmseg.datasets.pipelines.**RandomCutOut**(*prob*, *n_holes*, *cutout_shape=None*, *cutout_ratio=None*, *fill_in=(0, 0, 0)*, *seg_fill_in=None*)

CutOut operation.

Randomly drop some regions of image used in Cutout. :param prob: cutout probability. :type prob: float :param n_holes: Number of regions to be dropped.

If it is given as a list, number of holes will be randomly selected from the closed interval [*n_holes[0]*, *n_holes[1]*].

参数

- **cutout_shape** (*tuple[int, int] | list[tuple[int, int]]*) –The candidate shape of dropped regions. It can be *tuple[int, int]* to use a fixed cutout shape, or *list[tuple[int, int]]* to randomly choose shape from the list.
- **cutout_ratio** (*tuple[float, float] | list[tuple[float, float]]*) –The candidate ratio of dropped regions. It can be *tuple[float, float]* to use a fixed ratio or *list[tuple[float, float]]* to randomly choose ratio from the list. Please note that *cutout_shape* and *cutout_ratio* cannot be both given at the same time.
- **fill_in** (*tuple[float, float, float] | tuple[int, int, int]*) –The value of pixel to fill in the dropped regions. Default: (0, 0, 0).
- **seg_fill_in** (*int*) –The labels of pixel to fill in the dropped regions. If seg_fill_in is None, skip. Default: None.

class mmseg.datasets.pipelines.**RandomFlip**(*prob=None*, *direction='horizontal'*)

Flip the image & seg.

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

- **prob** (*float, optional*) –The flipping probability. Default: None.
- **direction** (*str, optional*) –The flipping direction. Options are ‘horizontal’ and ‘vertical’. Default: ‘horizontal’.

class mmseg.datasets.pipelines.**RandomMosaic**(*prob*, *img_scale=(640, 640)*, *center_ratio_range=(0.5, 1.5)*, *pad_val=0*, *seg_pad_val=255*)

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

```
                     mosaic transform
                        center_x
            +------------------------------+
            |       pad        |  pad      |
            |      +-----------+           |
            |      |           |           |
            |      |  image1   |--------+  |
            |      |           |        |  |
            |      |           | image2 |  |
center_y    |----+-------------+-----------|
            |    |   cropped   |           |
            |pad |   image3    |  image4   |
            |    |             |           |
            +----|-------------+-----------+
                 |             |
                 +-------------+

The mosaic transform steps are as follows:
    1. Choose the mosaic center as the intersections of 4 images
    2. Get the left top image according to the index, and randomly
       sample another 3 images from the custom dataset.
    3. Sub image will be cropped if image is larger than mosaic patch
```

参数

- **prob** (*float*) –mosaic probability.
- **img_scale** (*Sequence[int]*) –Image size after mosaic pipeline of a single image. The size of the output image is four times that of a single image. The output image comprises 4 single images. Default: (640, 640).
- **center_ratio_range** (*Sequence[float]*) –Center ratio range of mosaic output. Default: (0.5, 1.5).
- **pad_val** (*int*) –Pad value. Default: 0.
- **seg_pad_val** (*int*) –Pad value of segmentation map. Default: 255.

get_indexes(*dataset*)

Call function to collect indexes.

参数 dataset (MultiImageMixDataset) –The dataset.

返回 indexes.

返回类型 list

class mmseg.datasets.pipelines.**RandomRotate**(*prob*, *degree*, *pad_val=0*, *seg_pad_val=255*, *center=None*, *auto_bound=False*)

Rotate the image & seg.

参数

- **prob** (*float*) –The rotation probability.
- **degree** (*float, tuple[float]*) –Range of degrees to select from. If degree is a number instead of tuple like (min, max), the range of degree will be (-degree, +degree)
- **pad_val** (*float, optional*) –Padding value of image. Default: 0.
- **seg_pad_val** (*float, optional*) –Padding value of segmentation map. Default: 255.
- **center** (*tuple[float], optional*) –Center point (w, h) of the rotation in the source image. If not specified, the center of the image will be used. Default: None.
- **auto_bound** (*bool*) –Whether to adjust the image size to cover the whole rotated image. Default: False

class mmseg.datasets.pipelines.**Rerange**(*min_value=0*, *max_value=255*)

Rerange the image pixel value.

参数

- **min_value** (*float or int*) –Minimum value of the reranged image. Default: 0.
- **max_value** (*float or int*) –Maximum value of the reranged image. Default: 255.

class mmseg.datasets.pipelines.**Resize**(*img_scale=None*, *multiscale_mode='range'*, *ratio_range=None*, *keep_ratio=True*, *min_size=None*)

Resize images & seg.

This transform resizes the input image to some scale. If the input dict contains the key "scale", then the scale in the input dict is used, otherwise the specified scale in the init method is used.

`img_scale` can be None, a tuple (single-scale) or a list of tuple (multi-scale). There are 4 multiscale modes:

- `ratio_range is not None`:

1. **When img_scale is None, img_scale is the shape of image in results** (img_scale = results['img'].shape[:2]) and the image is resized based on the original size. (mode 1)

2. **When img_scale is a tuple (single-scale), randomly sample a ratio from** the ratio range and multiply it with the image scale. (mode 2)

- `ratio_range is None and multiscale_mode == "range"`: randomly sample a

scale from the a range. (mode 3)

- `ratio_range is None and multiscale_mode == "value"`: randomly sample a

scale from multiple scales. (mode 4)

参数

- **img_scale** (*tuple or list[tuple]*) –Images scales for resizing. Default:None.
- **multiscale_mode** (*str*) –Either "range" or "value". Default: 'range'
- **ratio_range** (*tuple[float]*) –(min_ratio, max_ratio). Default: None
- **keep_ratio** (*bool*) –Whether to keep the aspect ratio when resizing the image. Default: True
- **min_size** (*int, optional*) –The minimum size for input and the shape of the image and seg map will not be less than `min_size`. As the shape of model input is fixed like 'SETR' and 'BEiT'. Following the setting in these models, resized images must be bigger than the crop size in `slide_inference`. Default: None

static random_sample(*img_scales*)

Randomly sample an img_scale when `multiscale_mode=='range'`.

参数 img_scales (*list[tuple]*) –Images scale range for sampling. There must be two tuples in img_scales, which specify the lower and upper bound of image scales.

返回

Returns a tuple (img_scale, None), where `img_scale` is sampled scale and None is just a placeholder to be consistent with *random_select()*.

返回类型 (tuple, None)

static random_sample_ratio(*img_scale, ratio_range*)

Randomly sample an img_scale when `ratio_range` is specified.

A ratio will be randomly sampled from the range specified by `ratio_range`. Then it would be multiplied with `img_scale` to generate sampled scale.

参数

- **img_scale** (*tuple*) –Images scale base to multiply with ratio.
- **ratio_range** (*tuple[float]*) –The minimum and maximum ratio to scale the `img_scale`.

返回

Returns a tuple `(scale, None)`, where `scale` is sampled ratio multiplied with `img_scale` and None is just a placeholder to be consistent with *`random_select()`*.

返回类型 (tuple, None)

`static random_select`(*img_scales*)

Randomly select an img_scale from given candidates.

参数 `img_scales` (*`list[tuple]`*) –Images scales for selection.

返回

Returns a tuple `(img_scale, scale_dix)`, where `img_scale` is the selected image scale and `scale_idx` is the selected index in the given candidates.

返回类型 (tuple, int)

`class` `mmseg.datasets.pipelines.`**`SegRescale`**(*scale_factor=1*)

Rescale semantic segmentation maps.

参数 `scale_factor` (*`float`*) –The scale factor of the final output.

`class` `mmseg.datasets.pipelines.`**`ToDataContainer`**(*fields=({'key': 'img', 'stack': True}, {'key': 'gt_semantic_seg'})*)

Convert results to `mmcv.DataContainer` by given fields.

参数 `fields` (*`Sequence[dict]`*) –Each field is a dict like `dict(key='xxx', **kwargs)`. The `key` in result will be converted to `mmcv.DataContainer` with `**kwargs`. Default: `(dict(key='img', stack=True), dict(key='gt_semantic_seg'))`.

`class` `mmseg.datasets.pipelines.`**`ToTensor`**(*keys*)

Convert some results to `torch.Tensor` by given keys.

参数 `keys` (*`Sequence[str]`*) –Keys that need to be converted to Tensor.

`class` `mmseg.datasets.pipelines.`**`Transpose`**(*keys, order*)

Transpose some results by given keys.

参数

- **`keys`** (*`Sequence[str]`*) –Keys of results to be transposed.
- **`order`** (*`Sequence[int]`*) –Order of transpose.

`mmseg.datasets.pipelines.`**`to_tensor`**(*data*)

Convert objects of various python types to `torch.Tensor`.

Supported types are: `numpy.ndarray`, `torch.Tensor`, `Sequence`, `int` and `float`.

参数 `data`(*`torch.Tensor | numpy.ndarray | Sequence | int | float`*)–Data to be converted.

CHAPTER 24

mmseg.models

24.1 segmentors

class mmseg.models.segmentors.**BaseSegmentor**(*init_cfg=None*)

Base class for segmentors.

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

Placeholder for augmentation test.

abstract encode_decode(*img, img_metas*)

Placeholder for encode images with backbone and decode into a semantic segmentation map of the same size as input.

abstract extract_feat(*imgs*)

Placeholder for extract features from images.

forward(*img, img_metas, return_loss=True, **kwargs*)

Calls either *forward_train()* or *forward_test()* depending on whether `return_loss` is `True`.

Note this setting will change the expected inputs. When `return_loss=True`, img and img_meta are single-nested (i.e. Tensor and List[dict]), and when `resturn_loss=False`, img and img_meta should be double nested (i.e. List[Tensor], List[List[dict]]), with the outer list indicating test time augmentations.

forward_test(*imgs, img_metas, **kwargs*)

参数

- **imgs** (*List[Tensor]*) –the outer list indicates test-time augmentations and inner Tensor should have a shape NxCxHxW, which contains all images in the batch.
- **img_metas** (*List[List[dict]]*) –the outer list indicates test-time augs (multiscale, flip, etc.) and the inner list indicates images in a batch.

abstract forward_train(*imgs*, *img_metas*, ***kwargs*)

Placeholder for Forward function for training.

show_result(*img*, *result*, *palette=None*, *win_name=''*, *show=False*, *wait_time=0*, *out_file=None*, *opacity=0.5*)

Draw *result* over *img*.

参数

- **img** (*str or Tensor*) –The image to be displayed.
- **result** (*Tensor*) –The semantic segmentation results to draw over *img*.
- **palette** (*list[list[int]]] | np.ndarray | None*) –The palette of segmentation map. If None is given, random palette will be generated. Default: None
- **win_name** (*str*) –The window name.
- **wait_time** (*int*) –Value of waitKey param. Default: 0.
- **show** (*bool*) –Whether to show the image. Default: False.
- **out_file** (*str or None*) –The filename to write the image. Default: None.
- **opacity** (*float*) –Opacity of painted segmentation map. Default 0.5. Must be in (0, 1] range.

返回 Only if not *show* or *out_file*

返回类型 img (Tensor)

abstract simple_test(*img*, *img_meta*, ***kwargs*)

Placeholder for single image test.

train_step(*data_batch*, *optimizer*, ***kwargs*)

The iteration step during training.

This method defines an iteration step during training, except for the back propagation and optimizer updating, which are done in an optimizer hook. Note that in some complicated cases or models, the whole process including back propagation and optimizer updating is also defined in this method, such as GAN.

参数

- **data** (*dict*) –The output of dataloader.
- **optimizer** (`torch.optim.Optimizer` | dict) –The optimizer of runner is passed to `train_step()`. This argument is unused and reserved.

返回

It should contain at least 3 keys: `loss`, `log_vars`, `num_samples`. `loss` is a tensor for back propagation, which can be a weighted sum of multiple losses. `log_vars` contains all the variables to be sent to the logger. `num_samples` indicates the batch size (when the model is DDP, it means the batch size on each GPU), which is used for averaging the logs.

返回类型 dict

`val_step`(*data_batch*, *optimizer=None*, ***kwargs*)

The iteration step during validation.

This method shares the same signature as `train_step()`, but used during val epochs. Note that the evaluation after training epochs is not implemented with this method, but an evaluation hook.

`property with_auxiliary_head`

whether the segmentor has auxiliary head

Type bool

`property with_decode_head`

whether the segmentor has decode head

Type bool

`property with_neck`

whether the segmentor has neck

Type bool

`class` `mmseg.models.segmentors.`**`CascadeEncoderDecoder`**(*num_stages*, *backbone*, *decode_head*, *neck=None*, *auxiliary_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

Cascade Encoder Decoder segmentors.

CascadeEncoderDecoder almost the same as EncoderDecoder, while decoders of CascadeEncoderDecoder are cascaded. The output of previous decoder_head will be the input of next decoder_head.

`encode_decode`(*img*, *img_metas*)

Encode images with backbone and decode into a semantic segmentation map of the same size as input.

`class` `mmseg.models.segmentors.`**`EncoderDecoder`**(*backbone*, *decode_head*, *neck=None*, *auxiliary_head=None*, *train_cfg=None*, *test_cfg=None*, *pretrained=None*, *init_cfg=None*)

Encoder Decoder segmentors.

EncoderDecoder typically consists of backbone, decode_head, auxiliary_head. Note that auxiliary_head is only used for deep supervision during training, which could be dumped during inference.

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

Test with augmentations.

Only rescale=True is supported.

aug_test_logits (*img*, *img_metas*, *rescale=True*)

Test with augmentations.

Return seg_map logits. Only rescale=True is supported.

encode_decode (*img*, *img_metas*)

Encode images with backbone and decode into a semantic segmentation map of the same size as input.

extract_feat (*img*)

Extract features from images.

forward_dummy (*img*)

Dummy forward function.

forward_train (*img*, *img_metas*, *gt_semantic_seg*)

Forward function for training.

参数

- **img** (*Tensor*) –Input images.
- **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 *mmseg/datasets/pipelines/formatting.py:Collect*.
- **gt_semantic_seg** (*Tensor*) –Semantic segmentation masks used if the architecture supports semantic segmentation task.

返回 a dictionary of loss components

返回类型 dict[str, Tensor]

inference (*img*, *img_meta*, *rescale*)

Inference with slide/whole style.

参数

- **img** (*Tensor*) –The input image of shape (N, 3, H, W).
- **img_meta** (*dict*) –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 *mmseg/datasets/pipelines/formatting.py:Collect*.
- **rescale** (*bool*) –Whether rescale back to original shape.

返回 The output segmentation map.

返回类型 Tensor

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

Simple test with single image.

simple_test_logits (*img*, *img_metas*, *rescale=True*)

Test without augmentations.

Return numpy seg_map logits.

slide_inference (*img*, *img_meta*, *rescale*)

Inference by sliding-window with overlap.

If h_crop > h_img or w_crop > w_img, the small patch will be used to decode without padding.

whole_inference (*img*, *img_meta*, *rescale*)

Inference with full image.

24.2 backbones

class mmseg.models.backbones.**BEiT** (*img_size=224*, *patch_size=16*, *in_channels=3*, *embed_dims=768*, *num_layers=12*, *num_heads=12*, *mlp_ratio=4*, *out_indices=- 1*, *qv_bias=True*, *attn_drop_rate=0.0*, *drop_path_rate=0.0*, *norm_cfg={'type': 'LN'}*, *act_cfg={'type': 'GELU'}*, *patch_norm=False*, *final_norm=False*, *num_fcs=2*, *norm_eval=False*, *pretrained=None*, *init_values=0.1*, *init_cfg=None*)

BERT Pre-Training of Image Transformers.

参数

- **img_size** (*int | tuple*) –Input image size. Default: 224.
- **patch_size** (*int*) –The patch size. Default: 16.
- **in_channels** (*int*) –Number of input channels. Default: 3.
- **embed_dims** (*int*) –Embedding dimension. Default: 768.
- **num_layers** (*int*) –Depth of transformer. Default: 12.
- **num_heads** (*int*) –Number of attention heads. Default: 12.
- **mlp_ratio** (*int*) –Ratio of mlp hidden dim to embedding dim. Default: 4.
- **out_indices** (*list | tuple | int*) –Output from which stages. Default: -1.
- **qv_bias** (*bool*) –Enable bias for qv if True. Default: True.
- **attn_drop_rate** (*float*) –The drop out rate for attention layer. Default 0.0
- **drop_path_rate** (*float*) –Stochastic depth rate. Default 0.0.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type='LN')

- **act_cfg** (*dict*) –The activation config for FFNs. Default: dict(type='GELU').
- **patch_norm** (*bool*) –Whether to add a norm in PatchEmbed Block. Default: False.
- **final_norm** (*bool*) –Whether to add a additional layer to normalize final feature map. Default: False.
- **num_fcs** (*int*) –The number of fully-connected layers for FFNs. Default: 2.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **pretrained** (*str, optional*) –Model pretrained path. Default: None.
- **init_values** (*float*) –Initialize the values of BEiTAttention and FFN with learnable scaling.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None.

forward (*inputs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

init_weights ()

Initialize the weights.

resize_rel_pos_embed (*checkpoint*)

Resize relative pos_embed weights.

This function is modified from https://github.com/microsoft/unilm/blob/master/beit/semantic_segmentation/mmcv_custom/checkpoint.py. # noqa: E501 Copyright (c) Microsoft Corporation Licensed under the MIT License :param checkpoint: Key and value of the pretrain model. :type checkpoint: dict

返回

Interpolate the relative pos_embed weights in the pre-train model to the current model size.

返回类型 state_dict (dict)

train (*mode=True*)

Sets the module in training mode.

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

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

返回 self

返回类型 Module

class `mmseg.models.backbones.`**BiSeNetV1**(*backbone_cfg*, *in_channels=3*, *spatial_channels=(64, 64, 64, 128)*, *context_channels=(128, 256, 512)*, *out_indices=(0, 1, 2)*, *align_corners=False*, *out_channels=256*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *init_cfg=None*)

BiSeNetV1 backbone.

This backbone is the implementation of BiSeNet: Bilateral Segmentation Network for Real-time Semantic Segmentation.

参数

- **backbone_cfg** –(dict): Config of backbone of Context Path.
- **in_channels** (*int*) –The number of channels of input image. Default: 3.
- **spatial_channels** (*Tuple[int]*) –Size of channel numbers of various layers in Spatial Path. Default: (64, 64, 64, 128).
- **context_channels** (*Tuple[int]*) –Size of channel numbers of various modules in Context Path. Default: (128, 256, 512).
- **out_indices** (*Tuple[int] | int, optional*) –Output from which stages. Default: (0, 1, 2).
- **align_corners** (*bool, optional*) –The align_corners argument of resize operation in Bilateral Guided Aggregation Layer. Default: False.
- **out_channels** (*int*) –The number of channels of output. It must be the same with *in_channels* of decode_head. Default: 256.

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 mmseg.models.backbones.**BiSeNetV2**(*in_channels=3*, *detail_channels=(64, 64, 128)*, *semantic_channels=(16, 32, 64, 128)*, *semantic_expansion_ratio=6*, *bga_channels=128*, *out_indices=(0, 1, 2, 3, 4)*, *align_corners=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *init_cfg=None*)

BiSeNetV2: Bilateral Network with Guided Aggregation for Real-time Semantic Segmentation.

This backbone is the implementation of BiSeNetV2.

参数

- **in_channels** (*int*) –Number of channel of input image. Default: 3.
- **detail_channels** (*Tuple[int], optional*) –Channels of each stage in Detail Branch. Default: (64, 64, 128).
- **semantic_channels** (*Tuple[int], optional*) –Channels of each stage in Semantic Branch. Default: (16, 32, 64, 128). See Table 1 and Figure 3 of paper for more details.
- **semantic_expansion_ratio** (*int, optional*) –The expansion factor expanding channel number of middle channels in Semantic Branch. Default: 6.
- **bga_channels** (*int, optional*) –Number of middle channels in Bilateral Guided Aggregation Layer. Default: 128.
- **out_indices** (*Tuple[int] | int, optional*) –Output from which stages. Default: (0, 1, 2, 3, 4).
- **align_corners** (*bool, optional*) –The align_corners argument of resize operation in Bilateral Guided Aggregation Layer. Default: False.
- **conv_cfg** (*dict | None*) –Config of conv layers. Default: None.
- **norm_cfg** (*dict | None*) –Config of norm layers. Default: dict(type='BN').
- **act_cfg** (*dict*) –Config of activation layers. Default: dict(type='ReLU').
- **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 mmseg.models.backbones.**CGNet**(*in_channels=3*, *num_channels=(32, 64, 128)*, *num_blocks=(3, 21)*, *dilations=(2, 4)*, *reductions=(8, 16)*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *act_cfg={'type': 'PReLU'}*, *norm_eval=False*, *with_cp=False*, *pretrained=None*, *init_cfg=None*)

CGNet backbone.

This backbone is the implementation of A Light-weight Context Guided Network for Semantic Segmentation.

参数

- **in_channels** (*int*) –Number of input image channels. Normally 3.
- **num_channels** (*tuple[int]*) –Numbers of feature channels at each stages. Default: (32, 64, 128).
- **num_blocks** (*tuple[int]*) –Numbers of CG blocks at stage 1 and stage 2. Default: (3, 21).
- **dilations** (*tuple[int]*) –Dilation rate for surrounding context extractors at stage 1 and stage 2. Default: (2, 4).
- **reductions** (*tuple[int]*) –Reductions for global context extractors at stage 1 and stage 2. Default: (8, 16).
- **conv_cfg** (*dict*) –Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type=' BN' , requires_grad=True).
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type=' PReLU').
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **pretrained** (*str, optional*) –model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

train(*mode=True*)

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

class mmseg.models.backbones.**ERFNet**(*in_channels=3*, *enc_downsample_channels=(16, 64, 128)*, *enc_stage_non_bottlenecks=(5, 8)*, *enc_non_bottleneck_dilations=(2, 4, 8, 16)*, *enc_non_bottleneck_channels=(64, 128)*, *dec_upsample_channels=(64, 16)*, *dec_stages_non_bottleneck=(2, 2)*, *dec_non_bottleneck_channels=(64, 16)*, *dropout_ratio=0.1*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *init_cfg=None*)

ERFNet backbone.

This backbone is the implementation of ERFNet: Efficient Residual Factorized ConvNet for Real-time Semantic-Segmentation.

参数

- **in_channels** (*int*) –The number of channels of input image. Default: 3.
- **enc_downsample_channels** (*Tuple[int]*) –Size of channel numbers of various Downsampler block in encoder. Default: (16, 64, 128).
- **enc_stage_non_bottlenecks** (*Tuple[int]*) –Number of stages of Non-bottleneck block in encoder. Default: (5, 8).
- **enc_non_bottleneck_dilations** (*Tuple[int]*) –Dilation rate of each stage of Non-bottleneck block of encoder. Default: (2, 4, 8, 16).
- **enc_non_bottleneck_channels** (*Tuple[int]*) –Size of channel numbers of various Non-bottleneck block in encoder. Default: (64, 128).
- **dec_upsample_channels** (*Tuple[int]*) –Size of channel numbers of various Deconvolution block in decoder. Default: (64, 16).
- **dec_stages_non_bottleneck** (*Tuple[int]*) –Number of stages of Non-bottleneck block in decoder. Default: (2, 2).
- **dec_non_bottleneck_channels** (*Tuple[int]*) –Size of channel numbers of various Non-bottleneck block in decoder. Default: (64, 16).
- **drop_rate** (*float*) –Probability of an element to be zeroed. Default 0.1.

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 `mmseg.models.backbones.`**`FastSCNN`**(*in_channels=3*, *downsample_dw_channels=(32, 48)*, *global_in_channels=64*, *global_block_channels=(64, 96, 128)*, *global_block_strides=(2, 2, 1)*, *global_out_channels=128*, *higher_in_channels=64*, *lower_in_channels=128*, *fusion_out_channels=128*, *out_indices=(0, 1, 2)*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *align_corners=False*, *dw_act_cfg=None*, *init_cfg=None*)

Fast-SCNN Backbone.

This backbone is the implementation of Fast-SCNN: Fast Semantic Segmentation Network.

参数

- **in_channels** (*int*) –Number of input image channels. Default: 3.
- **downsample_dw_channels** (*tuple[int]*) –Number of output channels after the first conv layer & the second conv layer in Learning-To-Downsample (LTD) module. Default: (32, 48).
- **global_in_channels** (*int*) –Number of input channels of Global Feature Extractor(GFE). Equal to number of output channels of LTD. Default: 64.
- **global_block_channels** (*tuple[int]*) –Tuple of integers that describe the output channels for each of the MobileNet-v2 bottleneck residual blocks in GFE. Default: (64, 96, 128).
- **global_block_strides** (*tuple[int]*) –Tuple of integers that describe the strides (downsampling factors) for each of the MobileNet-v2 bottleneck residual blocks in GFE. Default: (2, 2, 1).
- **global_out_channels** (*int*) –Number of output channels of GFE. Default: 128.
- **higher_in_channels** (*int*) –Number of input channels of the higher resolution branch in FFM. Equal to global_in_channels. Default: 64.
- **lower_in_channels** (*int*) –Number of input channels of the lower resolution branch in FFM. Equal to global_out_channels. Default: 128.
- **fusion_out_channels** (*int*) –Number of output channels of FFM. Default: 128.
- **out_indices** (*tuple*) –Tuple of indices of list [higher_res_features, lower_res_features, fusion_output]. Often set to (0,1,2) to enable aux. heads. Default: (0, 1, 2).

- **conv_cfg** (*dict | None*) –Config of conv layers. Default: None
- **norm_cfg** (*dict | None*) –Config of norm layers. Default: dict(type='BN')
- **act_cfg** (*dict*) –Config of activation layers. Default: dict(type='ReLU')
- **align_corners** (*bool*) –align_corners argument of F.interpolate. Default: False
- **dw_act_cfg** (*dict*) –In DepthwiseSeparableConvModule, activation config of depthwise ConvModule. If it is 'default', it will be the same as *act_cfg*. 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 mmseg.models.backbones.**HRNet**(*extra*, *in_channels=3*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *norm_eval=False*, *with_cp=False*, *frozen_stages=- 1*, *zero_init_residual=False*, *multiscale_output=True*, *pretrained=None*, *init_cfg=None*)

HRNet backbone.

This backbone is the implementation of High-Resolution Representations for Labeling Pixels and Regions.

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. Normally 3.
- **conv_cfg** (*dict*) –Dictionary to construct and config conv layer. Default: None.

- **norm_cfg** (*dict*) –Dictionary to construct and config norm layer. Use *BN* by default.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Default: -1.
- **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 mmseg.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,
```

(下页继续)

(续上页)

```
>>>         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 mmseg.models.backbones.**ICNet**(*backbone_cfg*, *in_channels=3*, *layer_channels=(512, 2048)*, *light_branch_middle_channels=32*, *psp_out_channels=512*, *out_channels=(64, 256, 256)*, *pool_scales=(1, 2, 3, 6)*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *align_corners=False*, *init_cfg=None*)

ICNet for Real-Time Semantic Segmentation on High-Resolution Images.

This backbone is the implementation of ICNet.

参数

- **backbone_cfg** (*dict*) –Config dict to build backbone. Usually it is ResNet but it can also be other backbones.
- **in_channels** (*int*) –The number of input image channels. Default: 3.
- **layer_channels** (*Sequence[int]*) –The numbers of feature channels at layer 2 and layer 4 in ResNet. It can also be other backbones. Default: (512, 2048).

- **light_branch_middle_channels** (*int*) –The number of channels of the middle layer in light branch. Default: 32.
- **psp_out_channels** (*int*) –The number of channels of the output of PSP module. Default: 512.
- **out_channels** (*Sequence[int]*) –The numbers of output feature channels at each branches. Default: (64, 256, 256).
- **pool_scales** (*tuple[int]*) –Pooling scales used in Pooling Pyramid Module. Default: (1, 2, 3, 6).
- **conv_cfg** (*dict*) –Dictionary to construct and config conv layer. Default: None.
- **norm_cfg** (*dict*) –Dictionary to construct and config norm layer. Default: dict(type=' BN').
- **act_cfg** (*dict*) –Dictionary to construct and config act layer. Default: dict(type=' ReLU').
- **align_corners** (*bool*) –align_corners argument of F.interpolate. Default: False.
- **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 mmseg.models.backbones.**MAE**(*img_size=224*, *patch_size=16*, *in_channels=3*, *embed_dims=768*, *num_layers=12*, *num_heads=12*, *mlp_ratio=4*, *out_indices=- 1*, *attn_drop_rate=0.0*, *drop_path_rate=0.0*, *norm_cfg={'type': 'LN'}*, *act_cfg={'type': 'GELU'}*, *patch_norm=False*, *final_norm=False*, *num_fcs=2*, *norm_eval=False*, *pretrained=None*, *init_values=0.1*, *init_cfg=None*)

VisionTransformer with support for patch.

参数

- **img_size** (*int | tuple*) –Input image size. Default: 224.
- **patch_size** (*int*) –The patch size. Default: 16.
- **in_channels** (*int*) –Number of input channels. Default: 3.

- **embed_dims** (*int*) –embedding dimension. Default: 768.
- **num_layers** (*int*) –depth of transformer. Default: 12.
- **num_heads** (*int*) –number of attention heads. Default: 12.
- **mlp_ratio** (*int*) –ratio of mlp hidden dim to embedding dim. Default: 4.
- **out_indices** (*list | tuple | int*) –Output from which stages. Default: -1.
- **attn_drop_rate** (*float*) –The drop out rate for attention layer. Default 0.0
- **drop_path_rate** (*float*) –stochastic depth rate. Default 0.0.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type='LN')
- **act_cfg** (*dict*) –The activation config for FFNs. Default: dict(type='GELU').
- **patch_norm** (*bool*) –Whether to add a norm in PatchEmbed Block. Default: False.
- **final_norm** (*bool*) –Whether to add a additional layer to normalize final feature map. Default: False.
- **num_fcs** (*int*) –The number of fully-connected layers for FFNs. Default: 2.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **pretrained** (*str, optional*) –model pretrained path. Default: None.
- **init_values** (*float*) –Initialize the values of Attention and FFN with learnable scaling. Defaults to 0.1.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None.

fix_init_weight()

Rescale the initialization according to layer id.

This function is copied from https://github.com/microsoft/unilm/blob/master/beit/modeling_pretrain.py. # noqa: E501 Copyright (c) Microsoft Corporation Licensed under the MIT License

forward(*inputs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

init_weights()

Initialize the weights.

class mmseg.models.backbones.**MixVisionTransformer**(*in_channels=3*, *embed_dims=64*, *num_stages=4*, *num_layers=[3, 4, 6, 3]*, *num_heads=[1, 2, 4, 8]*, *patch_sizes=[7, 3, 3, 3]*, *strides=[4, 2, 2, 2]*, *sr_ratios=[8, 4, 2, 1]*, *out_indices=(0, 1, 2, 3)*, *mlp_ratio=4*, *qkv_bias=True*, *drop_rate=0.0*, *attn_drop_rate=0.0*, *drop_path_rate=0.0*, *act_cfg={'type': 'GELU'}*, *norm_cfg={'eps': 1e-06, 'type': 'LN'}*, *pretrained=None*, *init_cfg=None*, *with_cp=False*)

The backbone of Segformer.

This backbone is the implementation of SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers. :param in_channels: Number of input channels. Default: 3. :type in_channels: int :param embed_dims: Embedding dimension. Default: 768. :type embed_dims: int :param num_stags: The num of stages. Default: 4. :type num_stags: int :param num_layers: The layer number of each transformer encode

layer. Default: [3, 4, 6, 3].

参数

- **num_heads** (*Sequence[int]*) –The attention heads of each transformer encode layer. Default: [1, 2, 4, 8].
- **patch_sizes** (*Sequence[int]*) –The patch_size of each overlapped patch embedding. Default: [7, 3, 3, 3].
- **strides** (*Sequence[int]*) –The stride of each overlapped patch embedding. Default: [4, 2, 2, 2].
- **sr_ratios** (*Sequence[int]*) –The spatial reduction rate of each transformer encode layer. Default: [8, 4, 2, 1].
- **out_indices** (*Sequence[int] | int*) –Output from which stages. Default: (0, 1, 2, 3).
- **mlp_ratio** (*int*) –ratio of mlp hidden dim to embedding dim. Default: 4.
- **qkv_bias** (*bool*) –Enable bias for qkv if True. Default: True.
- **drop_rate** (*float*) –Probability of an element to be zeroed. Default 0.0
- **attn_drop_rate** (*float*) –The drop out rate for attention layer. Default 0.0
- **drop_path_rate** (*float*) –stochastic depth rate. Default 0.0

- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type=' LN')
- **act_cfg** (*dict*) –The activation config for FFNs. Default: dict(type=' GELU').
- **pretrained** (*str, optional*) –model pretrained path. Default: None.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

init_weights()

Initialize the weights.

class mmseg.models.backbones.**MobileNetV2**(*widen_factor=1.0, strides=(1, 2, 2, 2, 1, 2, 1), dilations=(1, 1, 1, 1, 1, 1, 1), out_indices=(1, 2, 4, 6), frozen_stages=- 1, conv_cfg=None, norm_cfg={'type': 'BN'}, act_cfg={'type': 'ReLU6'}, norm_eval=False, with_cp=False, pretrained=None, init_cfg=None*)

MobileNetV2 backbone.

This backbone is the implementation of MobileNetV2: Inverted Residuals and Linear Bottlenecks.

参数

- **widen_factor** (*float*) –Width multiplier, multiply number of channels in each layer by this amount. Default: 1.0.
- **strides** (*Sequence[int], optional*) –Strides of the first block of each layer. If not specified, default config in `arch_setting` will be used.
- **dilations** (*Sequence[int]*) –Dilation of each layer.
- **out_indices** (*None or Sequence[int]*) –Output from which stages. Default: (7,).
- **frozen_stages** (*int*) –Stages to be frozen (all param fixed). Default: -1, which means not freezing any parameters.

- **conv_cfg** (*dict*) –Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type='BN').
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type='ReLU6').
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **pretrained** (*str, optional*) –model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None

forward(*x*)

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.

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

Stack InvertedResidual blocks to build a layer for MobileNetV2.

参数

- **out_channels** (*int*) –out_channels of block.
- **num_blocks** (*int*) –Number of blocks.
- **stride** (*int*) –Stride of the first block.
- **dilation** (*int*) –Dilation of the first block.
- **expand_ratio** (*int*) –Expand the number of channels of the hidden layer in InvertedResidual by this ratio.

train(*mode=True*)

Sets the module in training mode.

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

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

返回 self

返回类型 Module

`class mmseg.models.backbones.`**`MobileNetV3`**(*arch='small', conv_cfg=None, norm_cfg={'type': 'BN'}, out_indices=(0, 1, 12), frozen_stages=- 1, reduction_factor=1, norm_eval=False, with_cp=False, pretrained=None, init_cfg=None*)

MobileNetV3 backbone.

This backbone is the improved implementation of Searching for MobileNetV3.

参数

- **arch** (*str*) –Architecture of mobilnetv3, from {'small', 'large'}. Default: 'small'.
- **conv_cfg** (*dict*) –Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type='BN').
- **out_indices** (*tuple[int]*) –Output from which layer. Default: (0, 1, 12).
- **frozen_stages** (*int*) –Stages to be frozen (all param fixed). Default: -1, which means not freezing any parameters.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **pretrained** (*str, optional*) –model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None

`forward`(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

`train`(*mode=True*)

Sets the module in training mode.

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

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

返回 self

返回类型 Module

class mmseg.models.backbones.**PCPVT**(*in_channels=3*, *embed_dims=[64, 128, 256, 512]*, *patch_sizes=[4, 2, 2, 2]*, *strides=[4, 2, 2, 2]*, *num_heads=[1, 2, 4, 8]*, *mlp_ratios=[4, 4, 4, 4]*, *out_indices=(0, 1, 2, 3)*, *qkv_bias=False*, *drop_rate=0.0*, *attn_drop_rate=0.0*, *drop_path_rate=0.0*, *norm_cfg={'type': 'LN'}*, *depths=[3, 4, 6, 3]*, *sr_ratios=[8, 4, 2, 1]*, *norm_after_stage=False*, *pretrained=None*, *init_cfg=None*)

The backbone of Twins-PCPVT.

This backbone is the implementation of Twins: Revisiting the Design of Spatial Attention in Vision Transformers.

参数

- **in_channels** (*int*) –Number of input channels. Default: 3.
- **embed_dims** (*list*) –Embedding dimension. Default: [64, 128, 256, 512].
- **patch_sizes** (*list*) –The patch sizes. Default: [4, 2, 2, 2].
- **strides** (*list*) –The strides. Default: [4, 2, 2, 2].
- **num_heads** (*int*) –Number of attention heads. Default: [1, 2, 4, 8].
- **mlp_ratios** (*int*) –Ratio of mlp hidden dim to embedding dim. Default: [4, 4, 4, 4].
- **out_indices** (*tuple[int]*) –Output from which stages. Default: (0, 1, 2, 3).
- **qkv_bias** (*bool*) –Enable bias for qkv if True. Default: False.
- **drop_rate** (*float*) –Probability of an element to be zeroed. Default 0.
- **attn_drop_rate** (*float*) –The drop out rate for attention layer. Default 0.0
- **drop_path_rate** (*float*) –Stochastic depth rate. Default 0.0
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type='LN')
- **depths** (*list*) –Depths of each stage. Default [3, 4, 6, 3]
- **sr_ratios** (*list*) –Kernel_size of conv in each Attn module in Transformer encoder layer. Default: [8, 4, 2, 1].
- **norm_after_stage** (**bool**) –Add extra norm. Default False.
- **init_cfg** (*dict, optional*) –The Config for initialization. Defaults to None.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

ResNeSt backbone.

This backbone is the implementation of ResNeSt: Split-Attention Networks.

参数

- **groups** (*int*) –Number of groups of Bottleneck. Default: 1
- **base_width** (*int*) –Base width of Bottleneck. Default: 4
- **radix** (*int*) –Radix of SpltAtConv2d. Default: 2
- **reduction_factor** (*int*) –Reduction factor of inter_channels in SplitAttentionConv2d. Default: 4.
- **avg_down_stride** (*bool*) –Whether to use average pool for stride in Bottleneck. Default: True.
- **kwargs** (*dict*) –Keyword arguments for ResNet.

make_res_layer(***kwargs*)

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

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

ResNeXt backbone.

This backbone is the implementation of Aggregated Residual Transformations for Deep Neural Networks.

参数

- **depth** (*int*) –Depth of resnet, from {18, 34, 50, 101, 152}.
- **in_channels** (*int*) –Number of input image channels. Normally 3.
- **num_stages** (*int*) –Resnet stages, normally 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.

示例

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

make_res_layer(***kwargs*)

Pack all blocks in a stage into a `ResLayer`

class mmseg.models.backbones.**ResNet**(*depth*, *in_channels=3*, *stem_channels=64*, *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=False*, *dcn=None*, *stage_with_dcn=(False, False, False, False)*, *plugins=None*, *multi_grid=None*, *contract_dilation=False*, *with_cp=False*, *zero_init_residual=True*, *pretrained=None*, *init_cfg=None*)

ResNet backbone.

This backbone is the improved implementation of Deep Residual Learning for Image Recognition.

参数

- **depth** (*int*) –Depth of resnet, from {18, 34, 50, 101, 152}.
- **in_channels** (*int*) –Number of input image channels. Default: 3.
- **stem_channels** (*int*) –Number of stem channels. Default: 64.
- **base_channels** (*int*) –Number of base channels of res layer. Default: 64.
- **num_stages** (*int*) –Resnet stages, normally 4. Default: 4.
- **strides** (*Sequence[int]*) –Strides of the first block of each stage. Default: (1, 2, 2, 2).
- **dilations** (*Sequence[int]*) –Dilation of each stage. Default: (1, 1, 1, 1).
- **out_indices** (*Sequence[int]*) –Output from which stages. Default: (0, 1, 2, 3).
- **style** (*str*) –*pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer. Default: 'pytorch'.
- **deep_stem** (*bool*) –Replace 7x7 conv in input stem with 3 3x3 conv. Default: False.
- **avg_down** (*bool*) –Use AvgPool instead of stride conv when downsampling in the bottle-neck. Default: False.
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Default: -1.
- **conv_cfg** (*dict | None*) –Dictionary to construct and config conv layer. When conv_cfg is None, cfg will be set to dict(type='Conv2d'). Default: None.
- **norm_cfg** (*dict*) –Dictionary to construct and config norm layer. Default: dict(type='BN', requires_grad=True).
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **dcn** (*dict | None*) –Dictionary to construct and config DCN conv layer. When dcn is not None, conv_cfg must be None. Default: None.

- **stage_with_dcn** (*Sequence[bool]*) –Whether to set DCN conv for each stage. The length of stage_with_dcn is equal to num_stages. Default: (False, False, False, False).
- **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: ‘after_conv1’, ‘after_conv2’, ‘after_conv3’.
  - stages (tuple[bool], optional): Stages to apply plugin, length

  should be same as ‘num_stages’. Default: None.
- **multi_grid** (*Sequence[int]|None*) –Multi grid dilation rates of last stage. Default: None.
- **contract_dilation** (*bool*) –Whether contract first dilation of each layer Default: False.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **zero_init_residual** (*bool*) –Whether to use zero init for last norm layer in resblocks to let them behave as identity. Default: True.
- **pretrained** (*str, optional*) –model pretrained path. Default: None.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None.

示例

```
>>> from mmseg.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 : »> 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 mmseg.models.backbones.**ResNetV1c**(***kwargs*)

ResNetV1c variant described in [1]_.

Compared with default ResNet(ResNetV1b), ResNetV1c replaces the 7x7 conv in the input stem with three 3x3 convs. For more details please refer to Bag of Tricks for Image Classification with Convolutional Neural Networks.

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

ResNetV1d variant described in [1]_.

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 `mmseg.models.backbones.`**`STDCContextPathNet`**(*backbone_cfg*, *last_in_channels=(1024, 512)*, *out_channels=128*, *ffm_cfg={'in_channels': 512, 'out_channels': 256, 'scale_factor': 4}*, *upsample_mode='nearest'*, *align_corners=None*, *norm_cfg={'type': 'BN'}*, *init_cfg=None*)

STDCNet with Context Path. The *outs* below is a list of three feature maps from deep to shallow, whose height and width is from small to big, respectively. The biggest feature map of *outs* is outputted for *STDCHead*, where Detail Loss would be calculated by Detail Ground-truth. The other two feature maps are used for Attention Refinement Module, respectively. Besides, the biggest feature map of *outs* and the last output of Attention Refinement Module are concatenated for Feature Fusion Module. Then, this fusion feature map *feat_fuse* would be outputted for *decode_head*. More details please refer to Figure 4 of original paper.

参数

- **backbone_cfg** (*dict*) –Config dict for stdc backbone.
- **last_in_channels** (*tuple(int)*) –two feature maps from stdc backbone. Default: (1024, 512).
- **out_channels** (*int*) –The channels of output feature maps. Default: 128.
- **ffm_cfg** (*dict*) –Config dict for Feature Fusion Module. Default: *dict(in_channels=512, out_channels=256, scale_factor=4)*.
- **upsample_mode** (*str*) –Algorithm used for upsampling: `'nearest'` | `'linear'` | `'bilinear'` | `'bicubic'` | `'trilinear'`. Default: `'nearest'`.
- **align_corners** (*str*) –align_corners argument of F.interpolate. It must be *None* if upsample_mode is `'nearest'`. Default: None.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type='BN').
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None.

返回

The tuple of list of output feature map for auxiliary heads and decoder head.

返回类型 outputs (tuple)

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 `mmseg.models.backbones.`**`STDCNet`**(*stdc_type*, *in_channels*, *channels*, *bottleneck_type*, *norm_cfg*, *act_cfg*, *num_convs=4*, *with_final_conv=False*, *pretrained=None*, *init_cfg=None*)

This backbone is the implementation of Rethinking BiSeNet For Real-time Semantic Segmentation.

参数

- **stdc_type** (*int*) –The type of backbone structure, *STDCNet1* and'STDCNet2' denotes two main backbones in paper, whose FLOPs is 813M and 1446M, respectively.
- **in_channels** (*int*) –The num of input_channels.
- **channels** (*tuple[int]*) –The output channels for each stage.
- **bottleneck_type** (*str*) –The type of STDC Module type, the value must be 'add' or 'cat'.
- **norm_cfg** (*dict*) –Config dict for normalization layer.
- **act_cfg** (*dict*) –The activation config for conv layers.
- **num_convs** (*int*) –Numbers of conv layer at each STDC Module. Default: 4.
- **with_final_conv** (*bool*) –Whether add a conv layer at the Module 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

```
>>> import torch
>>> stdc_type = 'STDCNet1'
>>> in_channels = 3
>>> channels = (32, 64, 256, 512, 1024)
>>> bottleneck_type = 'cat'
>>> inputs = torch.rand(1, 3, 1024, 2048)
>>> self = STDCNet(stdc_type, in_channels,
...                 channels, bottleneck_type).eval()
>>> outputs = self.forward(inputs)
>>> for i in range(len(outputs)):
```

(下页继续)

(续上页)

```
...     print(f'outputs[{i}].shape = {outputs[i].shape}')
outputs[0].shape = torch.Size([1, 256, 128, 256])
outputs[1].shape = torch.Size([1, 512, 64, 128])
outputs[2].shape = torch.Size([1, 1024, 32, 64])
```

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 mmseg.models.backbones.**SVT**(*in_channels=3*, *embed_dims=[64, 128, 256]*, *patch_sizes=[4, 2, 2, 2]*, *strides=[4, 2, 2, 2]*, *num_heads=[1, 2, 4]*, *mlp_ratios=[4, 4, 4]*, *out_indices=(0, 1, 2, 3)*, *qkv_bias=False*, *drop_rate=0.0*, *attn_drop_rate=0.0*, *drop_path_rate=0.2*, *norm_cfg={'type': 'LN'}*, *depths=[4, 4, 4]*, *sr_ratios=[4, 2, 1]*, *windiow_sizes=[7, 7, 7]*, *norm_after_stage=True*, *pretrained=None*, *init_cfg=None*)

The backbone of Twins-SVT.

This backbone is the implementation of Twins: Revisiting the Design of Spatial Attention in Vision Transformers.

参数

- **in_channels** (*int*) –Number of input channels. Default: 3.
- **embed_dims** (*list*) –Embedding dimension. Default: [64, 128, 256, 512].
- **patch_sizes** (*list*) –The patch sizes. Default: [4, 2, 2, 2].
- **strides** (*list*) –The strides. Default: [4, 2, 2, 2].
- **num_heads** (*int*) –Number of attention heads. Default: [1, 2, 4].
- **mlp_ratios** (*int*) –Ratio of mlp hidden dim to embedding dim. Default: [4, 4, 4].
- **out_indices** (*tuple[int]*) –Output from which stages. Default: (0, 1, 2, 3).
- **qkv_bias** (*bool*) –Enable bias for qkv if True. Default: False.
- **drop_rate** (*float*) –Dropout rate. Default 0.
- **attn_drop_rate** (*float*) –Dropout ratio of attention weight. Default 0.0
- **drop_path_rate** (*float*) –Stochastic depth rate. Default 0.2.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type=' LN')

- **depths** (*list*) –Depths of each stage. Default [4, 4, 4].
- **sr_ratios** (*list*) –Kernel_size of conv in each Attn module in Transformer encoder layer. Default: [4, 2, 1].
- **windiow_sizes** (*list*) –Window size of LSA. Default: [7, 7, 7],
- **input_features_slice** (**bool**) –Input features need slice. Default: False.
- **norm_after_stage** (**bool**) –Add extra norm. Default False.
- **strides** –Strides in patch-Embedding modules. Default: (2, 2, 2)
- **init_cfg** (*dict, optional*) –The Config for initialization. Defaults to None.

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

Swin Transformer backbone.

This backbone is the implementation of Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. Inspiration from https://github.com/microsoft/Swin-Transformer.

参数

- **pretrain_img_size** (*int | tuple[int]*) –The size of input image when pretrain. Defaults: 224.
- **in_channels** (*int*) –The num of input channels. Defaults: 3.
- **embed_dims** (*int*) –The feature dimension. Default: 96.
- **patch_size** (*int | tuple[int]*) –Patch size. Default: 4.
- **window_size** (*int*) –Window size. Default: 7.
- **mlp_ratio** (*int | float*) –Ratio of mlp hidden dim to embedding dim. Default: 4.
- **depths** (*tuple[int]*) –Depths of each Swin Transformer stage. Default: (2, 2, 6, 2).
- **num_heads** (*tuple[int]*) –Parallel attention heads of each Swin Transformer stage. Default: (3, 6, 12, 24).
- **strides** (*tuple[int]*) –The patch merging or patch embedding stride of each Swin Transformer stage. (In swin, we set kernel size equal to stride.) Default: (4, 2, 2, 2).

- **out_indices** (*tuple[int]*) –Output from which stages. Default: (0, 1, 2, 3).
- **qkv_bias** (*bool, optional*) –If True, add a learnable bias to query, key, value. Default: True
- **qk_scale** (*float | None, optional*) –Override default qk scale of head_dim ** -0.5 if set. Default: None.
- **patch_norm** (*bool*) –If add a norm layer for patch embed and patch merging. Default: True.
- **drop_rate** (*float*) –Dropout rate. Defaults: 0.
- **attn_drop_rate** (*float*) –Attention dropout rate. Default: 0.
- **drop_path_rate** (*float*) –Stochastic depth rate. Defaults: 0.1.
- **use_abs_pos_embed** (*bool*) –If True, add absolute position embedding to the patch embedding. Defaults: False.
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type=' LN').
- **norm_cfg** (*dict*) –Config dict for normalization layer at output of backone. Defaults: dict(type=' LN').
- **with_cp** (*bool, optional*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **pretrained** (*str, optional*) –model pretrained path. Default: None.
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters.
- **init_cfg** (*dict, optional*) –The Config for initialization. Defaults to None.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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.

train(*mode=True*)

Convert the model into training mode while keep layers freezed.

class mmseg.models.backbones.**TIMMBackbone**(*model_name*, *features_only=True*, *pretrained=True*, *checkpoint_path=''*, *in_channels=3*, *init_cfg=None*, ***kwargs*)

Wrapper to use backbones from timm library. More details can be found in timm .

参数

- **model_name** (*str*) –Name of timm model to instantiate.
- **pretrained** (*bool*) –Load pretrained weights if True.
- **checkpoint_path** (*str*) –Path of checkpoint to load after model is initialized.
- **in_channels** (*int*) –Number of input image channels. Default: 3.
- **init_cfg** (*dict, optional*) –Initialization config dict
- ****kwargs** –Other timm & model specific arguments.

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 mmseg.models.backbones.**UNet**(*in_channels=3*, *base_channels=64*, *num_stages=5*, *strides=(1, 1, 1, 1, 1)*, *enc_num_convs=(2, 2, 2, 2, 2)*, *dec_num_convs=(2, 2, 2, 2)*, *downsamples=(True, True, True, True)*, *enc_dilations=(1, 1, 1, 1, 1)*, *dec_dilations=(1, 1, 1, 1)*, *with_cp=False*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *upsample_cfg={'type': 'InterpConv'}*, *norm_eval=False*, *dcn=None*, *plugins=None*, *pretrained=None*, *init_cfg=None*)

UNet backbone.

This backbone is the implementation of U-Net: Convolutional Networks for Biomedical Image Segmentation.

参数

- **in_channels** (*int*) –Number of input image channels. Default" 3.
- **base_channels** (*int*) –Number of base channels of each stage. The output channels of the first stage. Default: 64.
- **num_stages** (*int*) –Number of stages in encoder, normally 5. Default: 5.
- **strides** (*Sequence[int 1 | 2]*) –Strides of each stage in encoder. len(strides) is equal to num_stages. Normally the stride of the first stage in encoder is 1. If strides[i]=2, it

uses stride convolution to downsample in the correspondence encoder stage. Default: (1, 1, 1, 1, 1).

- **enc_num_convs** (*Sequence[int]*) –Number of convolutional layers in the convolution block of the correspondence encoder stage. Default: (2, 2, 2, 2, 2).
- **dec_num_convs** (*Sequence[int]*) –Number of convolutional layers in the convolution block of the correspondence decoder stage. Default: (2, 2, 2, 2).
- **downsamples** (*Sequence[int]*) –Whether use MaxPool to downsample the feature map after the first stage of encoder (stages: [1, num_stages)). If the correspondence encoder stage use stride convolution (strides[i]=2), it will never use MaxPool to downsample, even downsamples[i-1]=True. Default: (True, True, True, True).
- **enc_dilations** (*Sequence[int]*) –Dilation rate of each stage in encoder. Default: (1, 1, 1, 1, 1).
- **dec_dilations** (*Sequence[int]*) –Dilation rate of each stage in decoder. Default: (1, 1, 1, 1).
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **conv_cfg** (*dict | None*) –Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict | None*) –Config dict for normalization layer. Default: dict(type='BN').
- **act_cfg** (*dict | None*) –Config dict for activation layer in ConvModule. Default: dict(type='ReLU').
- **upsample_cfg** (*dict*) –The upsample config of the upsample module in decoder. Default: dict(type='InterpConv').
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **dcn** (*bool*) –Use deformable convolution in convolutional layer or not. Default: None.
- **plugins** (*dict*) –plugins for convolutional layers. Default: None.
- **pretrained** (*str, optional*) –model pretrained path. Default: None
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None

Notice: The input image size should be divisible by the whole downsample rate of the encoder. More detail of the whole downsample rate can be found in UNet._check_input_divisible.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

train(*mode=True*)

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

class mmseg.models.backbones.**VisionTransformer**(*img_size=224, patch_size=16, in_channels=3, embed_dims=768, num_layers=12, num_heads=12, mlp_ratio=4, out_indices=- 1, qkv_bias=True, drop_rate=0.0, attn_drop_rate=0.0, drop_path_rate=0.0, with_cls_token=True, output_cls_token=False, norm_cfg={'type': 'LN'}, act_cfg={'type': 'GELU'}, patch_norm=False, final_norm=False, interpolate_mode='bicubic', num_fcs=2, norm_eval=False, with_cp=False, pretrained=None, init_cfg=None*)

Vision Transformer.

This backbone is the implementation of An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.

参数

- **img_size** (*int | tuple*) –Input image size. Default: 224.
- **patch_size** (*int*) –The patch size. Default: 16.
- **in_channels** (*int*) –Number of input channels. Default: 3.
- **embed_dims** (*int*) –embedding dimension. Default: 768.
- **num_layers** (*int*) –depth of transformer. Default: 12.
- **num_heads** (*int*) –number of attention heads. Default: 12.
- **mlp_ratio** (*int*) –ratio of mlp hidden dim to embedding dim. Default: 4.
- **out_indices** (*list | tuple | int*) –Output from which stages. Default: -1.
- **qkv_bias** (*bool*) –enable bias for qkv if True. Default: True.
- **drop_rate** (*float*) –Probability of an element to be zeroed. Default 0.0
- **attn_drop_rate** (*float*) –The drop out rate for attention layer. Default 0.0
- **drop_path_rate** (*float*) –stochastic depth rate. Default 0.0

- **with_cls_token** (*bool*) –Whether concatenating class token into image tokens as transformer input. Default: True.
- **output_cls_token** (*bool*) –Whether output the cls_token. If set True, *with_cls_token* must be True. Default: False.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type=' LN')
- **act_cfg** (*dict*) –The activation config for FFNs. Default: dict(type=' GELU').
- **patch_norm** (*bool*) –Whether to add a norm in PatchEmbed Block. Default: False.
- **final_norm** (*bool*) –Whether to add a additional layer to normalize final feature map. Default: False.
- **interpolate_mode** (*str*) –Select the interpolate mode for position embeding vector resize. Default: bicubic.
- **num_fcs** (*int*) –The number of fully-connected layers for FFNs. Default: 2.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **pretrained** (*str, optional*) –model pretrained path. Default: None.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None.

forward(*inputs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

init_weights()

Initialize the weights.

static resize_pos_embed(*pos_embed*, *input_shpae*, *pos_shape*, *mode*)

Resize pos_embed weights.

Resize pos_embed using bicubic interpolate method. :param pos_embed: Position embedding weights. :type pos_embed: torch.Tensor :param input_shpae: Tuple for (downsampled input image height,

downsampled input image width).

参数

- **pos_shape** (*tuple*) –The resolution of downsampled origin training image.
- **mode** (*str*) –Algorithm used for upsampling: `'nearest'` | `'linear'` | `'bilinear'` | `'bicubic'` | `'trilinear'`. Default: `'nearest'`

返回 The resized pos_embed of shape [B, L_new, C]

返回类型 torch.Tensor

train (*mode=True*)

Sets the module in training mode.

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

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

返回 self

返回类型 Module

24.3 decode_heads

class mmseg.models.decode_heads.**ANNHead** (*project_channels*, *query_scales=(1)*, *key_pool_scales=(1, 3, 6, 8)*, ***kwargs*)

Asymmetric Non-local Neural Networks for Semantic Segmentation.

This head is the implementation of ANNNet.

参数

- **project_channels** (*int*) –Projection channels for Nonlocal.
- **query_scales** (*tuple[int]*) –The scales of query feature map. Default: (1,)
- **key_pool_scales** (*tuple[int]*) –The pooling scales of key feature map. Default: (1, 3, 6, 8).

forward (*inputs*)

Forward function.

class mmseg.models.decode_heads.**APCHead** (*pool_scales=(1, 2, 3, 6)*, *fusion=True*, ***kwargs*)

Adaptive Pyramid Context Network for Semantic Segmentation.

This head is the implementation of APCNet.

参数

- **pool_scales** (*tuple[int]*) –Pooling scales used in Adaptive Context Module. Default: (1, 2, 3, 6).
- **fusion** (*bool*) –Add one conv to fuse residual feature.

forward (*inputs*)

Forward function.

class mmseg.models.decode_heads.**ASPPHead** (*dilations=(1, 6, 12, 18)*, ***kwargs*)

Rethinking Atrous Convolution for Semantic Image Segmentation.

This head is the implementation of DeepLabV3.

参数 **dilations** (*tuple[int]*) –Dilation rates for ASPP module. Default: (1, 6, 12, 18).

forward (*inputs*)

Forward function.

class mmseg.models.decode_heads.**CCHead** (*recurrence=2*, ***kwargs*)

CCNet: Criss-Cross Attention for Semantic Segmentation.

This head is the implementation of CCNet.

参数 **recurrence** (*int*) –Number of recurrence of Criss Cross Attention module. Default: 2.

forward (*inputs*)

Forward function.

class mmseg.models.decode_heads.**DAHead** (*pam_channels*, ***kwargs*)

Dual Attention Network for Scene Segmentation.

This head is the implementation of DANet.

参数 **pam_channels** (*int*) –The channels of Position Attention Module(PAM).

cam_cls_seg (*feat*)

CAM feature classification.

forward (*inputs*)

Forward function.

forward_test (*inputs*, *img_metas*, *test_cfg*)

Forward function for testing, only `pam_cam` is used.

losses (*seg_logit*, *seg_label*)

Compute `pam_cam`, `pam`, `cam` loss.

pam_cls_seg (*feat*)

PAM feature classification.

class mmseg.models.decode_heads.**DMHead** (*filter_sizes=(1, 3, 5, 7)*, *fusion=False*, ***kwargs*)

Dynamic Multi-scale Filters for Semantic Segmentation.

This head is the implementation of DMNet.

参数

- **filter_sizes** (*tuple[int]*) –The size of generated convolutional filters used in Dynamic Convolutional Module. Default: (1, 3, 5, 7).
- **fusion** (*bool*) –Add one conv to fuse DCM output feature.

forward (*inputs*)

Forward function.

class mmseg.models.decode_heads.**DNLHead** (*reduction=2*, *use_scale=True*, *mode='embedded_gaussian'*, *temperature=0.05*, ***kwargs*)

Disentangled Non-Local Neural Networks.

This head is the implementation of DNLNet.

参数

- **reduction** (*int*) –Reduction factor of projection transform. Default: 2.
- **use_scale** (*bool*) –Whether to scale pairwise_weight by sqrt(1/inter_channels). Default: False.
- **mode** (*str*) –The nonlocal mode. Options are ‘embedded_gaussian’, ‘dot_product’. Default: ‘embedded_gaussian.’.
- **temperature** (*float*) –Temperature to adjust attention. Default: 0.05

forward (*inputs*)

Forward function.

class mmseg.models.decode_heads.**DPTHead** (*embed_dims=768*, *post_process_channels=[96, 192, 384, 768]*, *readout_type='ignore'*, *patch_size=16*, *expand_channels=False*, *act_cfg={'type': 'ReLU'}*, *norm_cfg={'type': 'BN'}*, ***kwargs*)

Vision Transformers for Dense Prediction.

This head is implemented of DPT.

参数

- **embed_dims** (*int*) –The embed dimension of the ViT backbone. Default: 768.
- **post_process_channels** (*List*) –Out channels of post process conv layers. Default: [96, 192, 384, 768].
- **readout_type** (*str*) –Type of readout operation. Default: ‘ignore’.
- **patch_size** (*int*) –The patch size. Default: 16.
- **expand_channels** (*bool*) –Whether expand the channels in post process block. Default: False.

- **act_cfg** (*dict*) –The activation config for residual conv unit. Default dict(type=' ReLU').
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type=' BN').

forward (*inputs*)
Placeholder of forward function.

class mmseg.models.decode_heads.**DepthwiseSeparableASPPHead** (*c1_in_channels*, *c1_channels*, ***kwargs*)

Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation.

This head is the implementation of DeepLabV3+.

参数

- **c1_in_channels** (*int*) –The input channels of c1 decoder. If is 0, the no decoder will be used.
- **c1_channels** (*int*) –The intermediate channels of c1 decoder.

forward (*inputs*)
Forward function.

class mmseg.models.decode_heads.**DepthwiseSeparableFCNHead** (*dw_act_cfg=None*, ***kwargs*)

Depthwise-Separable Fully Convolutional Network for Semantic Segmentation.

This head is implemented according to Fast-SCNN: Fast Semantic Segmentation Network.

参数

- **in_channels** (*int*) –Number of output channels of FFM.
- **channels** (*int*) –Number of middle-stage channels in the decode head.
- **concat_input** (*bool*) –Whether to concatenate original decode input into the result of several consecutive convolution layers. Default: True.
- **num_classes** (*int*) –Used to determine the dimension of final prediction tensor.
- **in_index** (*int*) –Correspond with ' out_indices' in FastSCNN backbone.
- **norm_cfg** (*dict | None*) –Config of norm layers.
- **align_corners** (*bool*) –align_corners argument of F.interpolate. Default: False.
- **loss_decode** (*dict*) –Config of loss type and some relevant additional options.
- **dw_act_cfg** (*dict*) –Activation config of depthwise ConvModule. If it is ' default' , it will be the same as *act_cfg*. Default: None.

class mmseg.models.decode_heads.**EMAHead** (*ema_channels*, *num_bases*, *num_stages*, *concat_input=True*, *momentum=0.1*, ***kwargs*)

Expectation Maximization Attention Networks for Semantic Segmentation.

This head is the implementation of EMANet.

参数

- **ema_channels** (*int*) –EMA module channels
- **num_bases** (*int*) –Number of bases.
- **num_stages** (*int*) –Number of the EM iterations.
- **concat_input** (*bool*) –Whether concat the input and output of convs before classification layer. Default: True
- **momentum** (*float*) –Momentum to update the base. Default: 0.1.

forward (*inputs*)

Forward function.

class mmseg.models.decode_heads.**EncHead** (*num_codes=32*, *use_se_loss=True*, *add_lateral=False*, *loss_se_decode={'loss_weight': 0.2, 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, ***kwargs*)

Context Encoding for Semantic Segmentation.

This head is the implementation of EncNet.

参数

- **num_codes** (*int*) –Number of code words. Default: 32.
- **use_se_loss** (*bool*) –Whether use Semantic Encoding Loss (SE-loss) to regularize the training. Default: True.
- **add_lateral** (*bool*) –Whether use lateral connection to fuse features. Default: False.
- **loss_se_decode** (*dict*) –Config of decode loss. Default: dict(type=' CrossEntropyLoss' , use_sigmoid=True).

forward (*inputs*)

Forward function.

forward_test (*inputs*, *img_metas*, *test_cfg*)

Forward function for testing, ignore se_loss.

losses (*seg_logit*, *seg_label*)

Compute segmentation and semantic encoding loss.

class mmseg.models.decode_heads.**FCNHead** (*num_convs=2*, *kernel_size=3*, *concat_input=True*, *dilation=1*, ***kwargs*)

Fully Convolution Networks for Semantic Segmentation.

This head is implemented of FCNNet.

参数

- **num_convs** (*int*) –Number of convs in the head. Default: 2.
- **kernel_size** (*int*) –The kernel size for convs in the head. Default: 3.
- **concat_input** (*bool*) –Whether concat the input and output of convs before classification layer.
- **dilation** (*int*) –The dilation rate for convs in the head. Default: 1.

forward (*inputs*)

Forward function.

class mmseg.models.decode_heads.**FPNHead** (*feature_strides*, ***kwargs*)

Panoptic Feature Pyramid Networks.

This head is the implementation of Semantic FPN.

参数 feature_strides (*tuple[int]*) –The strides for input feature maps. stack_lateral. All strides suppose to be power of 2. The first one is of largest resolution.

forward (*inputs*)

Placeholder of forward function.

class mmseg.models.decode_heads.**GCHead** (*ratio=0.25*, *pooling_type='att'*, *fusion_types=('channel_add')*, ***kwargs*)

GCNet: Non-local Networks Meet Squeeze-Excitation Networks and Beyond.

This head is the implementation of GCNet.

参数

- **ratio** (*float*) –Multiplier of channels ratio. Default: 1/4.
- **pooling_type** (*str*) –The pooling type of context aggregation. Options are ‘att’, ‘avg’. Default: ‘avg’.
- **fusion_types** (*tuple[str]*) –The fusion type for feature fusion. Options are ‘channel_add’, ‘channel_mul’. Default: (‘channel_add’,)

forward (*inputs*)

Forward function.

class mmseg.models.decode_heads.**ISAHead** (*isa_channels*, *down_factor=(8, 8)*, ***kwargs*)

Interlaced Sparse Self-Attention for Semantic Segmentation.

This head is the implementation of ISA.

参数

- **isa_channels** (*int*) –The channels of ISA Module.
- **down_factor** (*tuple[int]*) –The local group size of ISA.

forward (*inputs*)

Forward function.

class mmseg.models.decode_heads.**IterativeDecodeHead** (*num_stages*, *kernel_generate_head*, *kernel_update_head*, ***kwargs*)

K-Net: Towards Unified Image Segmentation.

This head is the implementation of **'K-Net: <https://arxiv.org/abs/2106.14855>'_**.

参数

- **num_stages** (*int*) –The number of stages (kernel update heads) in IterativeDecodeHead. Default: 3.
- **kernel_generate_head** –(dict): Config of kernel generate head which generate mask predictions, dynamic kernels and class predictions for next kernel update heads.
- **kernel_update_head** (*dict*) –Config of kernel update head which refine dynamic kernels and class predictions iteratively.

forward (*inputs*)

Forward function.

losses (*seg_logit*, *seg_label*)

Compute segmentation loss.

class mmseg.models.decode_heads.**KernelUpdateHead** (*num_classes=150*, *num_ffn_fcs=2*, *num_heads=8*, *num_mask_fcs=3*, *feedforward_channels=2048*, *in_channels=256*, *out_channels=256*, *dropout=0.0*, *act_cfg={'inplace': True, 'type': 'ReLU'}*, *ffn_act_cfg={'inplace': True, 'type': 'ReLU'}*, *conv_kernel_size=1*, *feat_transform_cfg=None*, *kernel_init=False*, *with_ffn=True*, *feat_gather_stride=1*, *mask_transform_stride=1*, *kernel_updator_cfg={'act_cfg': {'inplace': True, 'type': 'ReLU'}, 'feat_channels': 64, 'in_channels': 256, 'norm_cfg': {'type': 'LN'}, 'out_channels': 256, 'type': 'DynamicConv'}*)

Kernel Update Head in K-Net.

参数

- **num_classes** (*int*) –Number of classes. Default: 150.
- **num_ffn_fcs** (*int*) –The number of fully-connected layers in FFNs. Default: 2.

- **num_heads** (*int*) –The number of parallel attention heads. Default: 8.
- **num_mask_fcs** (*int*) –The number of fully connected layers for mask prediction. Default: 3.
- **feedforward_channels** (*int*) –The hidden dimension of FFNs. Defaults: 2048.
- **in_channels** (*int*) –The number of channels of input feature map. Default: 256.
- **out_channels** (*int*) –The number of output channels. Default: 256.
- **dropout** (*float*) –The Probability of an element to be zeroed in MultiheadAttention and FFN. Default 0.0.
- **act_cfg** (*dict*) –Config of activation layers. Default: dict(type=' ReLU').
- **ffn_act_cfg** (*dict*) –Config of activation layers in FFN. Default: dict(type=' ReLU').
- **conv_kernel_size** (*int*) –The kernel size of convolution in Kernel Update Head for dynamic kernel updation. Default: 1.
- **feat_transform_cfg** (*dict | None*) –Config of feature transform. Default: None.
- **kernel_init** (*bool*) –Whether initiate mask kernel in mask head. Default: False.
- **with_ffn** (*bool*) –Whether add FFN in kernel update head. Default: True.
- **feat_gather_stride** (*int*) –Stride of convolution in feature transform. Default: 1.
- **mask_transform_stride** (*int*) –Stride of mask transform. Default: 1.
- **kernel_updator_cfg** (*dict*) –Config of kernel updator. Default: dict(

  type=' DynamicConv' , in_channels=256, feat_channels=64, out_channels=256, act_cfg=dict(type=' ReLU' , inplace=True), norm_cfg=dict(type=' LN')).

forward (*x*, *proposal_feat*, *mask_preds*, *mask_shape=None*)

Forward function of Dynamic Instance Interactive Head.

参数

- **x** (*Tensor*) –Feature map from FPN with shape (batch_size, feature_dimensions, H , W).
- **proposal_feat** (*Tensor*) –Intermediate feature get from diihead in last stage, has shape (batch_size, num_proposals, feature_dimensions)
- **mask_preds** (*Tensor*) –mask prediction from the former stage in shape (batch_size, num_proposals, H, W).

返回 The first tensor is predicted mask with shape (N, num_classes, H, W), the second tensor is dynamic kernel with shape (N, num_classes, channels, K, K).

返回类型 Tuple

init_weights()

Use xavier initialization for all weight parameter and set classification head bias as a specific value when use focal loss.

class mmseg.models.decode_heads.**KernelUpdator**(*in_channels=256*, *feat_channels=64*, *out_channels=None*, *gate_sigmoid=True*, *gate_norm_act=False*, *activate_out=False*, *norm_cfg={'type': 'LN'}*, *act_cfg={'inplace': True, 'type': 'ReLU'}*)

Dynamic Kernel Updator in Kernel Update Head.

参数

- **in_channels** (*int*) –The number of channels of input feature map. Default: 256.
- **feat_channels** (*int*) –The number of middle-stage channels in the kernel updator. Default: 64.
- **out_channels** (*int*) –The number of output channels.
- **gate_sigmoid** (*bool*) –Whether use sigmoid function in gate mechanism. Default: True.
- **gate_norm_act** (*bool*) –Whether add normalization and activation layer in gate mechanism. Default: False.
- **activate_out** –Whether add activation after gate mechanism. Default: False.
- **norm_cfg** (*dict | None*) –Config of norm layers. Default: dict(type=' LN').
- **act_cfg** (*dict*) –Config of activation layers. Default: dict(type=' ReLU').

forward(*update_feature*, *input_feature*)

Forward function of KernelUpdator.

参数

- **update_feature** (*torch.Tensor*) –Feature map assembled from each group. It would be reshaped with last dimension shape: *self.in_channels*.
- **input_feature** (*torch.Tensor*) –Intermediate feature with shape: (N, num_classes, conv_kernel_size**2, channels).

返回 The output tensor of shape (N*C1/C2, K*K, C2), where N is the number of classes, C1 and C2 are the feature map channels of KernelUpdateHead and KernelUpdator, respectively.

返回类型 Tensor

class mmseg.models.decode_heads.**LRASPPHead**(*branch_channels=(32, 64)*, ***kwargs*)

Lite R-ASPP (LRASPP) head is proposed in Searching for MobileNetV3.

This head is the improved implementation of Searching for MobileNetV3.

参数 **branch_channels** (*tuple[int]*) –The number of output channels in every each branch. Default: (32, 64).

forward (*inputs*)

Forward function.

class mmseg.models.decode_heads.**NLHead** (*reduction=2*, *use_scale=True*, *mode='embedded_gaussian'*, ***kwargs*)

Non-local Neural Networks.

This head is the implementation of NLNet.

参数

- **reduction** (*int*) –Reduction factor of projection transform. Default: 2.
- **use_scale** (*bool*) –Whether to scale pairwise_weight by sqrt(1/inter_channels). Default: True.
- **mode** (*str*) –The nonlocal mode. Options are ‘embedded_gaussian’, ‘dot_product’. Default: ‘embedded_gaussian.’.

forward (*inputs*)

Forward function.

class mmseg.models.decode_heads.**OCRHead** (*ocr_channels*, *scale=1*, ***kwargs*)

Object-Contextual Representations for Semantic Segmentation.

This head is the implementation of OCRNet.

参数

- **ocr_channels** (*int*) –The intermediate channels of OCR block.
- **scale** (*int*) –The scale of probability map in SpatialGatherModule in Default: 1.

forward (*inputs*, *prev_output*)

Forward function.

class mmseg.models.decode_heads.**PSAHead** (*mask_size*, *psa_type='bi-direction'*, *compact=False*, *shrink_factor=2*, *normalization_factor=1.0*, *psa_softmax=True*, ***kwargs*)

Point-wise Spatial Attention Network for Scene Parsing.

This head is the implementation of PSANet.

参数

- **mask_size** (*tuple[int]*) –The PSA mask size. It usually equals input size.
- **psa_type** (*str*) –The type of psa module. Options are ‘collect’, ‘distribute’, ‘bi-direction’. Default: ‘bi-direction’
- **compact** (*bool*) –Whether use compact map for ‘collect’ mode. Default: True.

- **shrink_factor** (*int*) –The downsample factors of psa mask. Default: 2.
- **normalization_factor** (*float*) –The normalize factor of attention.
- **psa_softmax** (*bool*) –Whether use softmax for attention.

forward (*inputs*)

Forward function.

class mmseg.models.decode_heads.**PSPHead** (*pool_scales=(1, 2, 3, 6)*, ***kwargs*)

Pyramid Scene Parsing Network.

This head is the implementation of PSPNet.

参数 **pool_scales** (*tuple[int]*) –Pooling scales used in Pooling Pyramid Module. Default: (1, 2, 3, 6).

forward (*inputs*)

Forward function.

class mmseg.models.decode_heads.**PointHead** (*num_fcs=3*, *coarse_pred_each_layer=True*, *conv_cfg={'type': 'Conv1d'}*, *norm_cfg=None*, *act_cfg={'inplace': False, 'type': 'ReLU'}*, ***kwargs*)

A mask point head use in PointRend.

This head is implemented of PointRend: Image Segmentation as Rendering. `PointHead` use shared multi-layer perceptron (equivalent to nn.Conv1d) to predict the logit of input points. The fine-grained feature and coarse feature will be concatenate together for predication.

参数

- **num_fcs** (*int*) –Number of fc layers in the head. Default: 3.
- **in_channels** (*int*) –Number of input channels. Default: 256.
- **fc_channels** (*int*) –Number of fc channels. Default: 256.
- **num_classes** (*int*) –Number of classes for logits. Default: 80.
- **class_agnostic** (*bool*) –Whether use class agnostic classification. If so, the output channels of logits will be 1. Default: False.
- **coarse_pred_each_layer** (*bool*) –Whether concatenate coarse feature with the output of each fc layer. Default: True.
- **conv_cfg** (*dict | None*) –Dictionary to construct and config conv layer. Default: dict(type=' Conv1d'))
- **norm_cfg** (*dict | None*) –Dictionary to construct and config norm layer. Default: None.
- **loss_point** (*dict*) –Dictionary to construct and config loss layer of point head. Default: dict(type=' CrossEntropyLoss' , use_mask=True, loss_weight=1.0).

cls_seg (*feat*)

Classify each pixel with fc.

forward (*fine_grained_point_feats*, *coarse_point_feats*)

Placeholder of forward function.

forward_test (*inputs*, *prev_output*, *img_metas*, *test_cfg*)

Forward function for testing.

参数

- **inputs** (*list[Tensor]*) –List of multi-level img features.
- **prev_output** (*Tensor*) –The output of previous decode head.
- **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 *mmseg/datasets/pipelines/formatting.py:Collect*.
- **test_cfg** (*dict*) –The testing config.

返回 Output segmentation map.

返回类型 Tensor

forward_train (*inputs*, *prev_output*, *img_metas*, *gt_semantic_seg*, *train_cfg*)

Forward function for training. :param inputs: List of multi-level img features. :type inputs: list[Tensor] :param prev_output: The output of previous decode head. :type prev_output: Tensor :param img_metas: 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 *mmseg/datasets/pipelines/formatting.py:Collect*.

参数

- **gt_semantic_seg** (*Tensor*) –Semantic segmentation masks used if the architecture supports semantic segmentation task.
- **train_cfg** (*dict*) –The training config.

返回 a dictionary of loss components

返回类型 dict[str, Tensor]

get_points_test (*seg_logits*, *uncertainty_func*, *cfg*)

Sample points for testing.

Find `num_points` most uncertain points from `uncertainty_map`.

参数

- **seg_logits** (*Tensor*) –A tensor of shape (batch_size, num_classes, height, width) for class-specific or class-agnostic prediction.
- **uncertainty_func** (*func*) –uncertainty calculation function.
- **cfg** (*dict*) –Testing config of point head.

返回

A tensor of shape (batch_size, num_points) that contains indices from [0, height x width) of the most uncertain points.

point_coords (Tensor): A tensor of shape (batch_size, num_points, 2) that contains [0, 1] x [0, 1] normalized coordinates of the most uncertain points from the `height x width` grid .

返回类型 point_indices (Tensor)

get_points_train (*seg_logits*, *uncertainty_func*, *cfg*)

Sample points for training.

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

参数

- **seg_logits** (*Tensor*) –Semantic segmentation logits, shape (batch_size, num_classes, height, width).
- **uncertainty_func** (*func*) –uncertainty calculation function.
- **cfg** (*dict*) –Training config of point head.

返回

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

返回类型 point_coords (Tensor)

losses (*point_logits*, *point_label*)

Compute segmentation loss.

class mmseg.models.decode_heads.**SETRMLAHead** (*mla_channels=128*, *up_scale=4*, ***kwargs*)

Multi level feature aggretation head of SETR.

MLA head of SETR.

参数

- **mlahead_channels** (*int*) –Channels of conv-conv-4x of multi-level feature aggregation. Default: 128.
- **up_scale** (*int*) –The scale factor of interpolate. Default:4.

forward(*inputs*)

Placeholder of forward function.

class mmseg.models.decode_heads.**SETRUPHead**(*norm_layer={'eps': 1e-06, 'requires_grad': True, 'type': 'LN'}*, *num_convs=1*, *up_scale=4*, *kernel_size=3*, *init_cfg=[{'type': 'Constant', 'val': 1.0, 'bias': 0, 'layer': 'LayerNorm'}, {'type': 'Normal', 'std': 0.01, 'override': {'name': 'conv_seg'}}]*, ***kwargs*)

Naive upsampling head and Progressive upsampling head of SETR.

Naive or PUP head of SETR.

参数

- **norm_layer** (*dict*) –Config dict for input normalization. Default: norm_layer=dict(type=' LN ' , eps=1e-6, requires_grad=True).
- **num_convs** (*int*) –Number of decoder convolutions. Default: 1.
- **up_scale** (*int*) –The scale factor of interpolate. Default:4.
- **kernel_size** (*int*) –The kernel size of convolution when decoding feature information from backbone. Default: 3.
- **init_cfg** (*dict | list[dict] | None*) –Initialization config dict. Default: dict(

  type=' Constant ' , val=1.0, bias=0, layer=' LayerNorm ').

forward(*x*)

Placeholder of forward function.

class mmseg.models.decode_heads.**STDCHead**(*boundary_threshold=0.1*, ***kwargs*)

This head is the implementation of Rethinking BiSeNet For Real-time Semantic Segmentation.

参数 **boundary_threshold** (*float*) –The threshold of calculating boundary. Default: 0.1.

losses(*seg_logit*, *seg_label*)

Compute Detail Aggregation Loss.

class mmseg.models.decode_heads.**SegformerHead**(*interpolate_mode='bilinear'*, ***kwargs*)

The all mlp Head of segformer.

This head is the implementation of *Segformer <https://arxiv.org/abs/2105.15203> _*.

参数 **interpolate_mode** –The interpolate mode of MLP head upsample operation. Default: ' bilinear ' .

forward(*inputs*)

Placeholder of forward function.

class mmseg.models.decode_heads.**SegmenterMaskTransformerHead**(*in_channels*, *num_layers*, *num_heads*, *embed_dims*, *mlp_ratio=4*, *drop_path_rate=0.1*, *drop_rate=0.0*, *attn_drop_rate=0.0*, *num_fcs=2*, *qkv_bias=True*, *act_cfg={'type': 'GELU'}*, *norm_cfg={'type': 'LN'}*, *init_std=0.02*, ***kwargs*)

Segmenter: Transformer for Semantic Segmentation.

This head is the implementation of **'Segmenter: <https://arxiv.org/abs/2105.05633>'_**.

参数

- **backbone_cfg** –(dict): Config of backbone of Context Path.
- **in_channels** (*int*) –The number of channels of input image.
- **num_layers** (*int*) –The depth of transformer.
- **num_heads** (*int*) –The number of attention heads.
- **embed_dims** (*int*) –The number of embedding dimension.
- **mlp_ratio** (*int*) –ratio of mlp hidden dim to embedding dim. Default: 4.
- **drop_path_rate** (*float*) –stochastic depth rate. Default 0.1.
- **drop_rate** (*float*) –Probability of an element to be zeroed. Default 0.0
- **attn_drop_rate** (*float*) –The drop out rate for attention layer. Default 0.0
- **num_fcs** (*int*) –The number of fully-connected layers for FFNs. Default: 2.
- **qkv_bias** (*bool*) –Enable bias for qkv if True. Default: True.
- **act_cfg** (*dict*) –The activation config for FFNs. Default: dict(type='GELU').
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type='LN')
- **init_std** (*float*) –The value of std in weight initialization. Default: 0.02.

forward(*inputs*)

Placeholder of forward function.

init_weights()

Initialize the weights.

class mmseg.models.decode_heads.**UPerHead**(*pool_scales=(1, 2, 3, 6)*, ***kwargs*)

Unified Perceptual Parsing for Scene Understanding.

This head is the implementation of UPerNet.

参数 **pool_scales** (*tuple[int]*) –Pooling scales used in Pooling Pyramid Module applied on the last feature. Default: (1, 2, 3, 6).

forward (*inputs*)
Forward function.

psp_forward (*inputs*)
Forward function of PSP module.

24.4 losses

class mmseg.models.losses.**Accuracy** (*topk=(1)*, *thresh=None*, *ignore_index=None*)
Accuracy calculation module.

forward (*pred*, *target*)
Forward function to calculate accuracy.

参数

- **pred** (*torch.Tensor*) –Prediction of models.
- **target** (*torch.Tensor*) –Target for each prediction.

返回 The accuracies under different topk criterions.

返回类型 tuple[float]

class mmseg.models.losses.**CrossEntropyLoss** (*use_sigmoid=False*, *use_mask=False*, *reduction='mean'*, *class_weight=None*, *loss_weight=1.0*, *loss_name='loss_ce'*, *avg_non_ignore=False*)
CrossEntropyLoss.

参数

- **use_sigmoid** (*bool, optional*) –Whether the prediction uses sigmoid of softmax. Defaults to False.
- **use_mask** (*bool, optional*) –Whether to use mask cross entropy loss. Defaults to False.
- **reduction** (*str, optional*) –. Defaults to ‘mean’. Options are “none”, “mean” and “sum”.
- **class_weight** (*list[float] | str, optional*) –Weight of each class. If in str format, read them from a file. Defaults to None.
- **loss_weight** (*float, optional*) –Weight of the loss. Defaults to 1.0.

- **loss_name** (*str, optional*) –Name of the loss item. If you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. Defaults to 'loss_ce'.
- **avg_non_ignore** (*bool*) –The flag decides to whether the loss is only averaged over non-ignored targets. Default: False. *New in version 0.23.0.*

extra_repr()

Extra repr.

forward(*cls_score*, *label*, *weight=None*, *avg_factor=None*, *reduction_override=None*, *ignore_index=- 100*, ***kwargs*)

Forward function.

property loss_name

Loss Name.

This function must be implemented and will return the name of this loss function. This name will be used to combine different loss items by simple sum operation. In addition, if you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name.

返回 The name of this loss item.

返回类型 str

class mmseg.models.losses.**DiceLoss**(*smooth=1*, *exponent=2*, *reduction='mean'*, *class_weight=None*, *loss_weight=1.0*, *ignore_index=255*, *loss_name='loss_dice'*, ***kwargs*)

DiceLoss.

This loss is proposed in V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation.

参数

- **smooth** (*float*) –A float number to smooth loss, and avoid NaN error. Default: 1
- **exponent** (*float*) –An float number to calculate denominator value: sum{x^exponent} + sum{y^exponent}. Default: 2.
- **reduction** (*str, optional*) –The method used to reduce the loss. Options are "none", "mean" and "sum". This parameter only works when per_image is True. Default: 'mean'.
- **class_weight** (*list[float] | str, optional*) –Weight of each class. If in str format, read them from a file. Defaults to None.
- **loss_weight** (*float, optional*) –Weight of the loss. Default to 1.0.
- **ignore_index** (*int | None*) –The label index to be ignored. Default: 255.
- **loss_name** (*str, optional*) –Name of the loss item. If you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. Defaults to 'loss_dice'

.

forward (*pred*, *target*, *avg_factor=None*, *reduction_override=None*, ***kwargs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

property loss_name

Loss Name.

This function must be implemented and will return the name of this loss function. This name will be used to combine different loss items by simple sum operation. In addition, if you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. :returns: The name of this loss item. :rtype: str

class `mmseg.models.losses.`**FocalLoss** (*use_sigmoid=True*, *gamma=2.0*, *alpha=0.5*, *reduction='mean'*, *class_weight=None*, *loss_weight=1.0*, *loss_name='loss_focal'*)

forward (*pred*, *target*, *weight=None*, *avg_factor=None*, *reduction_override=None*, *ignore_index=255*, ***kwargs*)

Forward function.

参数

- **pred** (*torch.Tensor*) –The prediction with shape (N, C) where C = number of classes, or (N, C, d_1, d_2, ⋯, d_K) with K≥1 in the case of K-dimensional loss.
- **target** (*torch.Tensor*) –The ground truth. If containing class indices, shape (N) where each value is 0≤targets[i]≤C−1, or (N, d_1, d_2, ⋯, d_K) with K≥1 in the case of K-dimensional loss. If containing class probabilities, same shape as the input.
- **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".
- **ignore_index** (*int, optional*) –The label index to be ignored. Default: 255

返回 The calculated loss

返回类型 torch.Tensor

property loss_name

Loss Name.

This function must be implemented and will return the name of this loss function. This name will be used to combine different loss items by simple sum operation. In addition, if you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. :returns: The name of this loss item. :rtype: str

class mmseg.models.losses.**LovaszLoss**(*loss_type='multi_class'*, *classes='present'*, *per_image=False*, *reduction='mean'*, *class_weight=None*, *loss_weight=1.0*, *loss_name='loss_lovasz'*)

LovaszLoss.

This loss is proposed in The Lovasz-Softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks.

参数

- **loss_type** (*str, optional*) –Binary or multi-class loss. Default: ‘multi_class’. Options are “binary” and “multi_class”.
- **classes** (*str | list[int], optional*) –Classes chosen to calculate loss. ‘all’ for all classes, ‘present’ for classes present in labels, or a list of classes to average. Default: ‘present’.
- **per_image** (*bool, optional*) –If per_image is True, compute the loss per image instead of per batch. Default: False.
- **reduction** (*str, optional*) –The method used to reduce the loss. Options are “none”, “mean” and “sum”. This parameter only works when per_image is True. Default: ‘mean’.
- **class_weight** (*list[float] | str, optional*) –Weight of each class. If in str format, read them from a file. Defaults to None.
- **loss_weight** (*float, optional*) –Weight of the loss. Defaults to 1.0.
- **loss_name** (*str, optional*) –Name of the loss item. If you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. Defaults to ‘loss_lovasz’.

forward(*cls_score*, *label*, *weight=None*, *avg_factor=None*, *reduction_override=None*, ***kwargs*)

Forward function.

property loss_name

Loss Name.

This function must be implemented and will return the name of this loss function. This name will be used to combine different loss items by simple sum operation. In addition, if you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. :returns: The name of this loss item. :rtype: str

class `mmseg.models.losses.`**`TverskyLoss`**(*smooth=1*, *class_weight=None*, *loss_weight=1.0*, *ignore_index=255*, *alpha=0.3*, *beta=0.7*, *loss_name='loss_tversky'*)

TverskyLoss. This loss is proposed in 'Tversky loss function for image segmentation using 3D fully convolutional deep networks.

<https://arxiv.org/abs/1706.05721>'_. :param smooth: A float number to smooth loss, and avoid NaN error.

Default: 1.

参数

- **class_weight** (*list[float] | str, optional*) –Weight of each class. If in str format, read them from a file. Defaults to None.
- **loss_weight** (*float, optional*) –Weight of the loss. Default to 1.0.
- **ignore_index** (*int | None*) –The label index to be ignored. Default: 255.
- **alpha** (*float, in [0, 1]*) –The coefficient of false positives. Default: 0.3.
- **beta** (*float, in [0, 1]*) –The coefficient of false negatives. Default: 0.7. Note: alpha + beta = 1.
- **loss_name** (*str, optional*) –Name of the loss item. If you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. Defaults to 'loss_tversky'.

forward(*pred*, *target*, ***kwargs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

property loss_name

Loss Name.

This function must be implemented and will return the name of this loss function. This name will be used to combine different loss items by simple sum operation. In addition, if you want this loss item to be included into the backward graph, *loss_* must be the prefix of the name. :returns: The name of this loss item. :rtype: str `mmseg.models.losses.`**`accuracy`**(*pred*, *target*, *topk=1*, *thresh=None*, *ignore_index=None*)

Calculate accuracy according to the prediction and target.

参数

- **`pred`** (*`torch.Tensor`*) –The model prediction, shape (N, num_class, …)
- **`target`** (*`torch.Tensor`*) –The target of each prediction, shape (N, , …)
- **`ignore_index`** (*`int | None`*) –The label index to be ignored. Default: None
- **`topk`** (*`int | tuple[int], optional`*) –If the predictions in `topk` matches the target, the predictions will be regarded as correct ones. Defaults to 1.
- **`thresh`** (*`float, optional`*) –If not None, predictions with scores under this threshold are considered incorrect. Default to None.

返回

If the input `topk` is a single integer, the function will return a single float as accuracy. If `topk` is a tuple containing multiple integers, the function will return a tuple containing accuracies of each `topk` number.

返回类型 float | tuple[float]

`mmseg.models.losses.`**`binary_cross_entropy`**(*pred*, *label*, *weight=None*, *reduction='mean'*, *avg_factor=None*, *class_weight=None*, *ignore_index=-100*, *avg_non_ignore=False*, ***kwargs*)

Calculate the binary CrossEntropy loss.

参数

- **`pred`** (*`torch.Tensor`*) –The prediction with shape (N, 1).
- **`label`** (*`torch.Tensor`*) –The learning label of the prediction. Note: In bce loss, label < 0 is invalid.
- **`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`*) –The label index to be ignored. Default: -100.
- **`avg_non_ignore`** (*`bool`*) –The flag decides to whether the loss is only averaged over non-ignored targets. Default: False. *New in version 0.23.0.*

返回 The calculated loss

返回类型 torch.Tensor

mmseg.models.losses.**cross_entropy**(*pred*, *label*, *weight=None*, *class_weight=None*, *reduction='mean'*, *avg_factor=None*, *ignore_index=- 100*, *avg_non_ignore=False*)

cross_entropy. The wrapper function for `F.cross_entropy()`

参数

- **pred** (*torch.Tensor*) –The prediction with shape (N, 1).
- **label** (*torch.Tensor*) –The learning label of the prediction.
- **weight** (*torch.Tensor, optional*) –Sample-wise loss weight. Default: None.
- **class_weight** (*list[float], optional*) –The weight for each class. Default: None.
- **reduction** (*str, optional*) –The method used to reduce the loss. Options are 'none', 'mean' and 'sum'. Default: 'mean'.
- **avg_factor** (*int, optional*) –Average factor that is used to average the loss. Default: None.
- **ignore_index** (*int*) –Specifies a target value that is ignored and does not contribute to the input gradients. When `avg_non_ignore `` is ``True`, and the `reduction` is `''mean''`, the loss is averaged over non-ignored targets. Defaults: -100.
- **avg_non_ignore** (*bool*) –The flag decides to whether the loss is only averaged over non-ignored targets. Default: False. *New in version 0.23.0.*

mmseg.models.losses.**mask_cross_entropy**(*pred*, *target*, *label*, *reduction='mean'*, *avg_factor=None*, *class_weight=None*, *ignore_index=None*, ***kwargs*)

Calculate the CrossEntropy loss for masks.

参数

- **pred** (*torch.Tensor*) –The prediction with shape (N, C), C is the number of classes.
- **target** (*torch.Tensor*) –The learning label of the prediction.
- **label** (*torch.Tensor*) –`label` indicates the class label of the mask' corresponding object. This will be used to select the mask in the of the class which the object belongs to when the mask prediction if not class-agnostic.
- **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** (*None*) –Placeholder, to be consistent with other loss. Default: None.

返回 The calculated loss

返回类型 torch.Tensor

mmseg.models.losses.**reduce_loss**(*loss*, *reduction*)

Reduce loss as specified.

参数

- **loss** (*Tensor*) –Elementwise loss tensor.
- **reduction** (*str*) –Options are "none", "mean" and "sum".

返回 Reduced loss tensor.

返回类型 Tensor

mmseg.models.losses.**weight_reduce_loss**(*loss*, *weight=None*, *reduction='mean'*, *avg_factor=None*)

Apply element-wise weight and reduce loss.

参数

- **loss** (*Tensor*) –Element-wise loss.
- **weight** (*Tensor*) –Element-wise weights.
- **reduction** (*str*) –Same as built-in losses of PyTorch.
- **avg_factor** (*float*) –Average factor when computing the mean of losses.

返回 Processed loss values.

返回类型 Tensor

mmseg.models.losses.**weighted_loss**(*loss_func*)

Create a weighted version of a given loss function.

To use this decorator, the loss function must have the signature like *loss_func(pred, target, **kwargs)*. The function only needs to compute element-wise loss without any reduction. This decorator will add weight and reduction arguments to the function. The decorated function will have the signature like *loss_func(pred, target, weight=None, reduction='mean', avg_factor=None, **kwargs)*.

Example

```
>>> import torch
>>> @weighted_loss
>>> def l1_loss(pred, target):
>>>     return (pred - target).abs()
```

```
>>> pred = torch.Tensor([0, 2, 3])
>>> target = torch.Tensor([1, 1, 1])
>>> weight = torch.Tensor([1, 0, 1])
>>> l1_loss(pred, target)
tensor(1.3333)
>>> l1_loss(pred, target, weight)
tensor(1.)
>>> l1_loss(pred, target, reduction='none')
tensor([1., 1., 2.])
>>> l1_loss(pred, target, weight, avg_factor=2)
tensor(1.5000)
```

CHAPTER 25

Indices and tables

- genindex
- search

Python 模块索引

m

mmseg.apis, 93

mmseg.core.evaluation, 98

mmseg.core.seg, 97

mmseg.core.utils, 101

mmseg.datasets, 103

mmseg.datasets.pipelines, 114

mmseg.models.backbones, 129

mmseg.models.decode_heads, 160

mmseg.models.losses, 175

mmseg.models.segmentors, 125

索引

A

Accuracy (*mmseg.models.losses* 中的类), 175
accuracy() (在 *mmseg.models.losses* 模块中), 179
add_prefix() (在 *mmseg.core.utils* 模块中), 101
ADE20KDataset (*mmseg.datasets* 中的类), 103
AdjustGamma (*mmseg.datasets.pipelines* 中的类), 114
ANNHead (*mmseg.models.decode_heads* 中的类), 160
APCHead (*mmseg.models.decode_heads* 中的类), 160
ASPPHead (*mmseg.models.decode_heads* 中的类), 161
aug_test() (*mmseg.models.segmentors.BaseSegmentor* 方法), 125
aug_test() (*mmseg.models.segmentors.EncoderDecoder* 方法), 127
aug_test_logits() (*mmseg.models.segmentors.EncoderDecoder* 方法), 128

B

BasePixelSampler (*mmseg.core.seg* 中的类), 97
BaseSegmentor (*mmseg.models.segmentors* 中的类), 125
BEiT (*mmseg.models.backbones* 中的类), 129
binary_cross_entropy() (在 *mmseg.models.losses* 模块中), 180
BiSeNetV1 (*mmseg.models.backbones* 中的类), 131
BiSeNetV2 (*mmseg.models.backbones* 中的类), 131
brightness() (*mmseg.datasets.pipelines.PhotoMetricDistortion* 方法), 118
build_dataloader() (在 *mmseg.datasets* 模块中), 112
build_dataset() (在 *mmseg.datasets* 模块中), 113
build_pixel_sampler() (在 *mmseg.core.seg* 模块中), 98

C

cam_cls_seg() (*mmseg.models.decode_heads.DAHead* 方法), 161
CascadeEncoderDecoder (*mmseg.models.segmentors* 中的类), 127
CCHead (*mmseg.models.decode_heads* 中的类), 161
CGNet (*mmseg.models.backbones* 中的类), 132
ChaseDB1Dataset (*mmseg.datasets* 中的类), 104
CityscapesDataset (*mmseg.datasets* 中的类), 104
CLAHE (*mmseg.datasets.pipelines* 中的类), 114
cls_seg() (*mmseg.models.decode_heads.PointHead* 方法), 170
COCOStuffDataset (*mmseg.datasets* 中的类), 104
Collect (*mmseg.datasets.pipelines* 中的类), 114
Compose (*mmseg.datasets.pipelines* 中的类), 115
ConcatDataset (*mmseg.datasets* 中的类), 106
contrast() (*mmseg.datasets.pipelines.PhotoMetricDistortion* 方法), 118
convert() (*mmseg.datasets.pipelines.PhotoMetricDistortion* 方法), 118
crop() (*mmseg.datasets.pipelines.RandomCrop* 方法), 119
cross_entropy() (在 *mmseg.models.losses* 模块中), 181

CrossEntropyLoss (*mmseg.models.losses 中的类*), 175
CustomDataset (*mmseg.datasets 中的类*), 106

D

DAHead (*mmseg.models.decode_heads 中的类*), 161
DarkZurichDataset (*mmseg.datasets 中的类*), 110
DepthwiseSeparableASPPHead (*mmseg.models.decode_heads 中的类*), 163
DepthwiseSeparableFCNHead (*mmseg.models.decode_heads 中的类*), 163
DiceLoss (*mmseg.models.losses 中的类*), 176
DistEvalHook (*mmseg.core.evaluation 中的类*), 98
DMHead (*mmseg.models.decode_heads 中的类*), 161
DNLHead (*mmseg.models.decode_heads 中的类*), 162
DPTHead (*mmseg.models.decode_heads 中的类*), 162
DRIVEDataset (*mmseg.datasets 中的类*), 110

E

EMAHead (*mmseg.models.decode_heads 中的类*), 163
EncHead (*mmseg.models.decode_heads 中的类*), 164
encode_decode() (*mmseg.models.segmentors.BaseSegmentor 方法*), 125
encode_decode() (*mmseg.models.segmentors.CascadeEncoderDecoder 方法*), 127
encode_decode() (*mmseg.models.segmentors.EncoderDecoder 方法*), 128
EncoderDecoder (*mmseg.models.segmentors 中的类*), 127
ERFNet (*mmseg.models.backbones 中的类*), 134
eval_metrics() (*在 mmseg.core.evaluation 模块中*), 98
EvalHook (*mmseg.core.evaluation 中的类*), 98
evaluate() (*mmseg.datasets.CityscapesDataset 方法*), 104
evaluate() (*mmseg.datasets.ConcatDataset 方法*), 106
evaluate() (*mmseg.datasets.CustomDataset 方法*), 108
extra_repr() (*mmseg.models.losses.CrossEntropyLoss 方法*), 176
extract_feat() (*mmseg.models.segmentors.BaseSegmentor 方法*), 125
extract_feat() (*mmseg.models.segmentors.EncoderDecoder 方法*), 128

F

FastSCNN (*mmseg.models.backbones 中的类*), 135
FCNHead (*mmseg.models.decode_heads 中的类*), 164
fix_init_weight() (*mmseg.models.backbones.MAE 方法*), 140
FocalLoss (*mmseg.models.losses 中的类*), 177
format_results() (*mmseg.datasets.ADE20KDataset 方法*), 103
format_results() (*mmseg.datasets.CityscapesDataset 方法*), 105
format_results() (*mmseg.datasets.ConcatDataset 方法*), 106
format_results() (*mmseg.datasets.CustomDataset 方法*), 108
format_results() (*mmseg.datasets.LoveDADataset 方法*), 110
forward() (*mmseg.models.backbones.BEiT 方法*), 130
forward() (*mmseg.models.backbones.BiSeNetV1 方法*), 131
forward() (*mmseg.models.backbones.BiSeNetV2 方法*), 132
forward() (*mmseg.models.backbones.CGNet 方法*), 133
forward() (*mmseg.models.backbones.ERFNet 方法*), 134
forward() (*mmseg.models.backbones.FastSCNN 方法*), 136
forward() (*mmseg.models.backbones.HRNet 方法*), 138
forward() (*mmseg.models.backbones.ICNet 方法*), 139
forward() (*mmseg.models.backbones.MAE 方法*), 140
forward() (*mmseg.models.backbones.MixVisionTransformer 方法*), 142
forward() (*mmseg.models.backbones.MobileNetV2 方法*), 143
forward() (*mmseg.models.backbones.MobileNetV3 方*

法), 144
forward() (*mmseg.models.backbones.PCPVT 方法*), 145
forward() (*mmseg.models.backbones.ResNet 方法*), 149
forward() (*mmseg.models.backbones.STDCContextPathNet 方法*), 151
forward() (*mmseg.models.backbones.STDCNet 方法*), 153
forward() (*mmseg.models.backbones.SwinTransformer 方法*), 155
forward() (*mmseg.models.backbones.TIMMBackbone 方法*), 156
forward() (*mmseg.models.backbones.UNet 方法*), 157
forward() (*mmseg.models.backbones.VisionTransformer 方法*), 159
forward() (*mmseg.models.decode_heads.ANNHead 方法*), 160
forward() (*mmseg.models.decode_heads.APCHead 方法*), 161
forward() (*mmseg.models.decode_heads.ASPPHead 方法*), 161
forward() (*mmseg.models.decode_heads.CCHead 方法*), 161
forward() (*mmseg.models.decode_heads.DAHead 方法*), 161
forward() (*mmseg.models.decode_heads.DepthwiseSeparableASPPHead 方法*), 163
forward() (*mmseg.models.decode_heads.DMHead 方法*), 162
forward() (*mmseg.models.decode_heads.DNLHead 方法*), 162
forward() (*mmseg.models.decode_heads.DPTHead 方法*), 163
forward() (*mmseg.models.decode_heads.EMAHead 方法*), 164
forward() (*mmseg.models.decode_heads.EncHead 方法*), 164
forward() (*mmseg.models.decode_heads.FCNHead 方法*), 165
forward() (*mmseg.models.decode_heads.FPNHead 方法*), 165
forward() (*mmseg.models.decode_heads.GCHead 方法*), 165
forward() (*mmseg.models.decode_heads.ISAHead 方法*), 165
forward() (*mmseg.models.decode_heads.IterativeDecodeHead 方法*), 166
forward() (*mmseg.models.decode_heads.KernelUpdateHead 方法*), 167
forward() (*mmseg.models.decode_heads.KernelUpdator 方法*), 168
forward() (*mmseg.models.decode_heads.LRASPPHead 方法*), 169
forward() (*mmseg.models.decode_heads.NLHead 方法*), 169
forward() (*mmseg.models.decode_heads.OCRHead 方法*), 169
forward() (*mmseg.models.decode_heads.PointHead 方法*), 171
forward() (*mmseg.models.decode_heads.PSAHead 方法*), 170
forward() (*mmseg.models.decode_heads.PSPHead 方法*), 170
forward() (*mmseg.models.decode_heads.SegformerHead 方法*), 173
forward() (*mmseg.models.decode_heads.SegmenterMaskTransformerHead 方法*), 174
forward() (*mmseg.models.decode_heads.SETRMLAHead 方法*), 172
forward() (*mmseg.models.decode_heads.SETRUPHead 方法*), 173
forward() (*mmseg.models.decode_heads.UPerHead 方法*), 175
forward() (*mmseg.models.losses.Accuracy 方法*), 175
forward() (*mmseg.models.losses.CrossEntropyLoss 方法*), 176
forward() (*mmseg.models.losses.DiceLoss 方法*), 177
forward() (*mmseg.models.losses.FocalLoss 方法*), 177
forward() (*mmseg.models.losses.LovaszLoss 方法*), 178
forward() (*mmseg.models.losses.TverskyLoss 方法*), 179
forward() (*mmseg.models.segmentors.BaseSegmentor*

方法), 125
forward_dummy() (*mmseg.models.segmentors.EncoderDecoder* 方法), 128
forward_test() (*mmseg.models.decode_heads.DAHead* 方法), 161
forward_test() (*mmseg.models.decode_heads.EncHead* 方法), 164
forward_test() (*mmseg.models.decode_heads.PointHead* 方法), 171
forward_test() (*mmseg.models.segmentors.BaseSegmentor* 方法), 125
forward_train() (*mmseg.models.decode_heads.PointHead* 方法), 171
forward_train() (*mmseg.models.segmentors.BaseSegmentor* 方法), 126
forward_train() (*mmseg.models.segmentors.EncoderDecoder* 方法), 128
FPNHead (*mmseg.models.decode_heads* 中的类), 165

G

GCHead (*mmseg.models.decode_heads* 中的类), 165
get_ann_info() (*mmseg.datasets.CustomDataset* 方法), 108
get_classes() (在 *mmseg.core.evaluation* 模块中), 99
get_classes_and_palette() (*mmseg.datasets.CustomDataset* 方法), 108
get_crop_bbox() (*mmseg.datasets.pipelines.RandomCrop* 方法), 119
get_dataset_idx_and_sample_idx() (*mmseg.datasets.ConcatDataset* 方法), 106
get_gt_seg_map_by_idx() (*mmseg.datasets.CustomDataset* 方法), 109
get_gt_seg_maps() (*mmseg.datasets.CustomDataset* 方法), 109
get_indexes() (*mmseg.datasets.pipelines.RandomMosaic* 方法), 121
get_palette() (在 *mmseg.core.evaluation* 模块中), 99
get_points_test() (*mmseg.models.decode_heads.PointHead* 方法), 171
get_points_train() (*mmseg.models.decode_heads.PointHead* 方法), 172
get_root_logger() (在 *mmseg.apis* 模块中), 93

H

HRFDataset (*mmseg.datasets* 中的类), 110
HRNet (*mmseg.models.backbones* 中的类), 136
hue() (*mmseg.datasets.pipelines.PhotoMetricDistortion* 方法), 118

I

ICNet (*mmseg.models.backbones* 中的类), 138
ImageToTensor (*mmseg.datasets.pipelines* 中的类), 115
inference() (*mmseg.models.segmentors.EncoderDecoder* 方法), 128
inference_segmentor() (在 *mmseg.apis* 模块中), 93
init_random_seed() (在 *mmseg.apis* 模块中), 94
init_segmentor() (在 *mmseg.apis* 模块中), 94
init_weights() (*mmseg.models.backbones.BEiT* 方法), 130
init_weights() (*mmseg.models.backbones.MAE* 方法), 140
init_weights() (*mmseg.models.backbones.MixVisionTransformer* 方法), 142
init_weights() (*mmseg.models.backbones.PCPVT* 方法), 146
init_weights() (*mmseg.models.backbones.SwinTransformer* 方

法), 155

init_weights() (*mmseg.models.backbones.VisionTransformer* 方法), 159

init_weights() (*mmseg.models.decode_heads.KernelUpdateHead* 方法), 167

init_weights() (*mmseg.models.decode_heads.SegmenterMaskTransformerHead* 方法), 174

intersect_and_union() (在 *mmseg.core.evaluation* 模块中), 99

ISAHead (*mmseg.models.decode_heads* 中的类), 165

iSAIDDataset (*mmseg.datasets* 中的类), 113

ISPRSDataset (*mmseg.datasets* 中的类), 110

IterativeDecodeHead (*mmseg.models.decode_heads* 中的类), 166

K

KernelUpdateHead (*mmseg.models.decode_heads* 中的类), 166

KernelUpdator (*mmseg.models.decode_heads* 中的类), 168

L

load_annotations() (*mmseg.datasets.CustomDataset* 方法), 109

load_annotations() (*mmseg.datasets.iSAIDDataset* 方法), 113

LoadAnnotations (*mmseg.datasets.pipelines* 中的类), 115

LoadImageFromFile (*mmseg.datasets.pipelines* 中的类), 115

loss_name (*mmseg.models.losses.CrossEntropyLoss property*), 176

loss_name (*mmseg.models.losses.DiceLoss property*), 177

loss_name (*mmseg.models.losses.FocalLoss property*), 178

loss_name (*mmseg.models.losses.LovaszLoss property*), 178

loss_name (*mmseg.models.losses.TverskyLoss property*), 179

losses() (*mmseg.models.decode_heads.DAHead* 方法), 161

losses() (*mmseg.models.decode_heads.EncHead* 方法), 164

losses() (*mmseg.models.decode_heads.IterativeDecodeHead* 方法), 166

losses() (*mmseg.models.decode_heads.PointHead* 方法), 172

losses() (*mmseg.models.decode_heads.STDCHead* 方法), 173

LovaszLoss (*mmseg.models.losses* 中的类), 178

LoveDADataset (*mmseg.datasets* 中的类), 110

LRASPPHead (*mmseg.models.decode_heads* 中的类), 168

M

MAE (*mmseg.models.backbones* 中的类), 139

make_layer() (*mmseg.models.backbones.MobileNetV2* 方法), 143

make_res_layer() (*mmseg.models.backbones.ResNeSt* 方法), 146

make_res_layer() (*mmseg.models.backbones.ResNet* 方法), 149

make_res_layer() (*mmseg.models.backbones.ResNeXt* 方法), 147

make_stage_plugins() (*mmseg.models.backbones.ResNet* 方法), 150

mask_cross_entropy() (在 *mmseg.models.losses* 模块中), 181

mean_dice() (在 *mmseg.core.evaluation* 模块中), 99

mean_fscore() (在 *mmseg.core.evaluation* 模块中), 100

mean_iou() (在 *mmseg.core.evaluation* 模块中), 100

MixVisionTransformer (*mmseg.models.backbones* 中的类), 141

mmseg.apis
  模块, 93

mmseg.core.evaluation
  模块, 98

mmseg.core.seg

模块, 97
mmseg.core.utils
模块, 101
mmseg.datasets
模块, 103
mmseg.datasets.pipelines
模块, 114
mmseg.models.backbones
模块, 129
mmseg.models.decode_heads
模块, 160
mmseg.models.losses
模块, 175
mmseg.models.segmentors
模块, 125
MobileNetV2 (*mmseg.models.backbones 中的类*), 142
MobileNetV3 (*mmseg.models.backbones 中的类*), 144
multi_gpu_test() (*在 mmseg.apis 模块中*), 94
MultiImageMixDataset (*mmseg.datasets 中的类*), 111
MultiScaleFlipAug (*mmseg.datasets.pipelines 中的类*), 116

N

NightDrivingDataset (*mmseg.datasets 中的类*), 111
NLHead (*mmseg.models.decode_heads 中的类*), 169
norm1 (*mmseg.models.backbones.HRNet property*), 138
norm1 (*mmseg.models.backbones.ResNet property*), 150
norm2 (*mmseg.models.backbones.HRNet property*), 138
Normalize (*mmseg.datasets.pipelines 中的类*), 117

O

OCRHead (*mmseg.models.decode_heads 中的类*), 169
OHEMPixelSampler (*mmseg.core.seg 中的类*), 97

P

Pad (*mmseg.datasets.pipelines 中的类*), 117
pam_cls_seg() (*mmseg.models.decode_heads.DAHead 方法*), 161
PascalContextDataset (*mmseg.datasets 中的类*), 112
PascalContextDataset59 (*mmseg.datasets 中的类*), 112
PascalVOCDataset (*mmseg.datasets 中的类*), 112
PCPVT (*mmseg.models.backbones 中的类*), 145
PhotoMetricDistortion (*mmseg.datasets.pipelines 中的类*), 117
PointHead (*mmseg.models.decode_heads 中的类*), 170
PotsdamDataset (*mmseg.datasets 中的类*), 112
pre_eval() (*mmseg.datasets.ConcatDataset 方法*), 106
pre_eval() (*mmseg.datasets.CustomDataset 方法*), 109
pre_eval_to_metrics() (*在 mmseg.core.evaluation 模块中*), 101
pre_pipeline() (*mmseg.datasets.CustomDataset 方法*), 109
prepare_test_img() (*mmseg.datasets.CustomDataset 方法*), 109
prepare_train_img() (*mmseg.datasets.CustomDataset 方法*), 110
PSAHead (*mmseg.models.decode_heads 中的类*), 169
psp_forward() (*mmseg.models.decode_heads.UPerHead 方法*), 175
PSPHead (*mmseg.models.decode_heads 中的类*), 170

R

random_sample() (*mmseg.datasets.pipelines.Resize 静态方法*), 122
random_sample_ratio() (*mmseg.datasets.pipelines.Resize 静态方法*), 122
random_select() (*mmseg.datasets.pipelines.Resize 静态方法*), 123
RandomCrop (*mmseg.datasets.pipelines 中的类*), 118
RandomCutOut (*mmseg.datasets.pipelines 中的类*), 119
RandomFlip (*mmseg.datasets.pipelines 中的类*), 119
RandomMosaic (*mmseg.datasets.pipelines 中的类*), 120
RandomRotate (*mmseg.datasets.pipelines 中的类*), 121
reduce_loss() (*在 mmseg.models.losses 模块中*), 182
RepeatDataset (*mmseg.datasets 中的类*), 112
Rerange (*mmseg.datasets.pipelines 中的类*), 121
Resize (*mmseg.datasets.pipelines 中的类*), 121
resize_pos_embed() (*mmseg.models.backbones.VisionTransformer 静*

态方法), 159
resize_rel_pos_embed() (*mmseg.models.backbones.BEiT* 方法), 130
ResNeSt (*mmseg.models.backbones* 中的类), 146
ResNet (*mmseg.models.backbones* 中的类), 147
ResNetV1c (*mmseg.models.backbones* 中的类), 150
ResNetV1d (*mmseg.models.backbones* 中的类), 150
ResNeXt (*mmseg.models.backbones* 中的类), 146
results2img() (*mmseg.datasets.ADE20KDataset* 方法), 104
results2img() (*mmseg.datasets.CityscapesDataset* 方法), 105
results2img() (*mmseg.datasets.LoveDADataset* 方法), 111
RGB2Gray (*mmseg.datasets.pipelines* 中的类), 118

S

sample() (*mmseg.core.seg.BasePixelSampler* 方法), 97
sample() (*mmseg.core.seg.OHEMPixelSampler* 方法), 97
saturation() (*mmseg.datasets.pipelines.PhotoMetricDistortion* 方法), 118
SegformerHead (*mmseg.models.decode_heads* 中的类), 173
SegmenterMaskTransformerHead (*mmseg.models.decode_heads* 中的类), 173
SegRescale (*mmseg.datasets.pipelines* 中的类), 123
set_random_seed() (在 *mmseg.apis* 模块中), 95
SETRMLAHead (*mmseg.models.decode_heads* 中的类), 172
SETRUPHead (*mmseg.models.decode_heads* 中的类), 173
show_result() (*mmseg.models.segmentors.BaseSegmentor* 方法), 126
show_result_pyplot() (在 *mmseg.apis* 模块中), 95
simple_test() (*mmseg.models.segmentors.BaseSegmentor* 方法), 126
simple_test() (*mmseg.models.segmentors.EncoderDecoder* 方法), 128
simple_test_logits() (*mmseg.models.segmentors.EncoderDecoder* 方法), 129
single_gpu_test() (在 *mmseg.apis* 模块中), 95
slide_inference() (*mmseg.models.segmentors.EncoderDecoder* 方法), 129
STAREDataset (*mmseg.datasets* 中的类), 112
STDCContextPathNet (*mmseg.models.backbones* 中的类), 151
STDCHead (*mmseg.models.decode_heads* 中的类), 173
STDCNet (*mmseg.models.backbones* 中的类), 152
SVT (*mmseg.models.backbones* 中的类), 153
SwinTransformer (*mmseg.models.backbones* 中的类), 154
sync_random_seed() (在 *mmseg.core.utils* 模块中), 101

T

TIMMBackbone (*mmseg.models.backbones* 中的类), 155
to_tensor() (在 *mmseg.datasets.pipelines* 模块中), 123
ToDataContainer (*mmseg.datasets.pipelines* 中的类), 123
ToTensor (*mmseg.datasets.pipelines* 中的类), 123
train() (*mmseg.models.backbones.BEiT* 方法), 130
train() (*mmseg.models.backbones.CGNet* 方法), 134
train() (*mmseg.models.backbones.HRNet* 方法), 138
train() (*mmseg.models.backbones.MobileNetV2* 方法), 143
train() (*mmseg.models.backbones.MobileNetV3* 方法), 144
train() (*mmseg.models.backbones.ResNet* 方法), 150
train() (*mmseg.models.backbones.SwinTransformer* 方法), 155
train() (*mmseg.models.backbones.UNet* 方法), 158
train() (*mmseg.models.backbones.VisionTransformer* 方法), 160
train_segmentor() (在 *mmseg.apis* 模块中), 96
train_step() (*mm-*

seg.models.segmentors.BaseSegmentor 方 法), 126

Transpose (*mmseg.datasets.pipelines* 中的类), 123

TverskyLoss (*mmseg.models.losses* 中的类), 179

U

UNet (*mmseg.models.backbones* 中的类), 156

update_skip_type_keys() (*mmseg.datasets.MultiImageMixDataset* 方 法), 111

UPerHead (*mmseg.models.decode_heads* 中的类), 174

V

val_step() (*mmseg.models.segmentors.BaseSegmentor* 方法), 127

VisionTransformer (*mmseg.models.backbones* 中的类), 158

W

weight_reduce_loss() (在 *mmseg.models.losses* 模块中), 182

weighted_loss() (在 *mmseg.models.losses* 模块中), 182

whole_inference() (*mmseg.models.segmentors.EncoderDecoder* 方法), 129

with_auxiliary_head (*mmseg.models.segmentors.BaseSegmentor property*), 127

with_decode_head (*mmseg.models.segmentors.BaseSegmentor property*), 127

with_neck (*mmseg.models.segmentors.BaseSegmentor property*), 127

�

模块
- mmseg.apis, 93
- mmseg.core.evaluation, 98
- mmseg.core.seg, 97
- mmseg.core.utils, 101
- mmseg.datasets, 103
- mmseg.datasets.pipelines, 114
- mmseg.models.backbones, 129
- mmseg.models.decode_heads, 160
- mmseg.models.losses, 175
- mmseg.models.segmentors, 125