mmcv Documentation

发布 *1.6.2*

MMCV Contributors

2022 年 10 月 24 日

介绍与安装

1 介绍 MMCV **3**

2 安装 MMCV **5**

2.1 安装 mmcv-full 5

2.2 安装 mmcv 8

3 从源码编译 MMCV **9**

3.1 编译 mmcv-full 9

3.2 编译 mmcv 15

3.3 在 IPU 机器编译 mmcv 15

4 解读文章汇总 **17**

4.1 MMCV 解读文章 17

4.2 下游算法库解读文章 18

4.3 PyTorch 解读文章 18

4.4 其他 19

5 配置 **21**

5.1 不含重复键值对从基类配置文件继承 22

5.2 含重复键值对从基类配置文件继承 23

5.3 从具有忽略字段的配置文件继承 23

5.4 从多个基类配置文件继承（基类配置文件不应包含相同的键） 23

5.5 从基类引用变量 24

6 注册器 **25**

6.1 什么是注册器 25

6.2 一个简单的例子 26

6.3 自定义构建函数 27

6.4 注册器层结构 28

7 执行器 **31**
7.1 EpochBasedRunner . . . 31
7.2 IterBasedRunner . . . 32
7.3 一个简单例子 . . . 33

8 文件输入输出 **35**
8.1 读取和保存数据 . . . 35
8.2 读取文件并返回列表或字典 . . . 37
8.3 读取和保存权重文件 . . . 39

9 数据处理 **41**
9.1 图像 . . . 41
9.2 视频 . . . 44

10 可视化 **49**

11 卷积神经网络 **51**
11.1 网络层的构建 . . . 51
11.2 模块组件 . . . 52
11.3 Weight initialization . . . 53
11.4 Model Zoo . . . 63

12 算子 **65**

13 辅助函数 **67**
13.1 进度条 . . . 67
13.2 计时器 . . . 68

14 MMCV 中 ONNX 模块简介 (实验性) **71**
14.1 register_extra_symbolics . . . 71

15 MMCV 中的 ONNX Runtime 自定义算子 **73**
15.1 ONNX Runtime 介绍 . . . 73
15.2 ONNX 介绍 . . . 73
15.3 为什么要在 MMCV 中添加 ONNX 自定义算子? . . . 73
15.4 MMCV 已支持的算子 . . . 74
15.5 如何编译 ONNX Runtime 自定义算子? . . . 74
15.6 如何在 python 下使用 ONNX Runtime 对导出的 ONNX 模型做编译 . . . 74
15.7 如何为 MMCV 添加 ONNX Runtime 的自定义算子 . . . 75
15.8 已知问题 . . . 76
15.9 引用 . . . 76

16 ONNX Runtime 自定义算子 **77**
16.1 SoftNMS . . . 79
16.2 RoIAlign . . . 79

16.3 NMS . . . 80
16.4 grid_sampler . . . 80
16.5 CornerPool . . . 81
16.6 cummax . . . 81
16.7 cummin . . . 82
16.8 MMCVModulatedDeformConv2d . . . 82

17 MMCV 中的 TensorRT 自定义算子 (实验性) 83
17.1 介绍 . . . 84
17.2 MMCV 中的 TensorRT 插件列表 . . . 84
17.3 如何编译 MMCV 中的 TensorRT 插件 . . . 84
17.4 创建 TensorRT 推理引擎并在 python 下进行推理 . . . 85
17.5 如何在 MMCV 中添加新的 TensorRT 自定义算子 . . . 86
17.6 已知问题 . . . 87
17.7 引用 . . . 87

18 TensorRT 自定义算子 89
18.1 MMCVRoIAlign . . . 91
18.2 ScatterND . . . 91
18.3 NonMaxSuppression . . . 92
18.4 MMCVDeformConv2d . . . 93
18.5 grid_sampler . . . 93
18.6 cummax . . . 94
18.7 cummin . . . 94
18.8 MMCVInstanceNormalization . . . 95
18.9 MMCVModulatedDeformConv2d . . . 95

19 English 97

20 简体中文 99

21 v1.3.18 101

22 v1.3.11 103

23 常见问题 107
23.1 安装问题 . . . 107
23.2 使用问题 . . . 110

24 贡献代码 111
24.1 工作流 . . . 111
24.2 代码风格 . . . 112

25 拉取请求 113
25.1 什么是拉取请求? . . . 113

25.2 基本的工作流: . . . 113
25.3 具体步骤 . . . 114
25.4 PR 规范 . . . 116

26 fileio **117**

27 image **127**

28 video **147**

29 arraymisc **155**

30 visualization **157**

31 utils **161**

32 cnn **183**

33 runner **205**

34 engine **241**

35 ops **243**

36 Indices and tables **281**

Python 模块索引 **283**

索引 **285**

您可以在页面左下角切换中英文文档。

CHAPTER 1

介绍 MMCV

MMCV 是一个面向计算机视觉的基础库，它支持了很多开源项目，例如：

- MIM: MIM 是 OpenMMlab 项目、算法、模型的统一入口
- MMClassification: OpenMMLab 图像分类工具箱
- MMDetection: OpenMMLab 目标检测工具箱
- MMDetection3D: OpenMMLab 新一代通用 3D 目标检测平台
- MMRotate: OpenMMLab 旋转框检测工具箱与测试基准
- MMSegmentation: OpenMMLab 语义分割工具箱
- MMOCR: OpenMMLab 全流程文字检测识别理解工具箱
- MMPose: OpenMMLab 姿态估计工具箱
- MMHuman3D: OpenMMLab 人体参数化模型工具箱与测试基准
- MMSelfSup: OpenMMLab 自监督学习工具箱与测试基准
- MMRazor: OpenMMLab 模型压缩工具箱与测试基准
- MMFewShot: OpenMMLab 少样本学习工具箱与测试基准
- MMAction2: OpenMMLab 新一代视频理解工具箱
- MMTracking: OpenMMLab 一体化视频目标感知平台
- MMFlow: OpenMMLab 光流估计工具箱与测试基准
- MMEditing: OpenMMLab 图像视频编辑工具箱

- MMGeneration: OpenMMLab 图片视频生成模型工具箱
- MMDeploy: OpenMMLab 模型部署框架

MMCV 提供了以下功能：

- 通用的 IO 接口
- 图像和视频处理
- 图像和标注结果可视化
- 常用小工具（进度条，计时器等）
- 基于 PyTorch 的通用训练框架
- 多种 CNN 网络结构
- 高质量实现的 CPU 和 CUDA 算子

MMCV 支持以下的系统：

- Linux
- Windows
- macOS

欢迎查看文档了解更多特性和用法。

注解： MMCV 需要 Python 3.6 以上版本。

CHAPTER 2

安装 MMCV

MMCV 有两个版本：

- **mmcv-full**: 完整版，包含所有的特性以及丰富的开箱即用的 CPU 和 CUDA 算子。注意，完整版本可能需要更长时间来编译。
- **mmcv**: 精简版，不包含 CPU 和 CUDA 算子但包含其余所有特性和功能，类似 MMCV 1.0 之前的版本。如果你不需要使用算子的话，精简版可以作为一个考虑选项。

警告： 请不要在同一个环境中安装两个版本，否则可能会遇到类似 `ModuleNotFound` 的错误。在安装一个版本之前，需要先卸载另一个。如果 `CUDA 可用`，强烈推荐安装 `mmcv-full`。

2.1 安装 mmcv-full

注解:

- 下述安装步骤仅适用于 Linux 和 Windows 平台，如需在 macOS 平台安装 mmcv-full，请参考源码安装 mmcv-full。
- 如需编译 ONNX Runtime 自定义算子，请参考如何编译 ONNX Runtime 自定义算子
- 如需编译 TensorRT 自定义，请参考如何编译 MMCV 中的 TensorRT 插件

在安装 mmcv-full 之前，请确保 PyTorch 已经成功安装在环境中，可以参考 PyTorch 官方安装文档。可使用以下命令验证

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

如果输出版本信息，则表示 PyTorch 已安装。

2.1.1 使用 mim 安装（推荐）

mim 是 OpenMMLab 项目的包管理工具，使用它可以很方便地安装 mmcv-full。

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

如果发现上述的安装命令没有使用预编译包（以 `.whl` 结尾）而是使用源码包（以 `.tar.gz` 结尾）安装，则有可能是我们没有提供和当前环境的 PyTorch 版本、CUDA 版本相匹配的 mmcv-full 预编译包，此时，你可以*源码安装 mmcv-full*。

Looking in links: https://download.openmmlab.com/mmcv/dist/cu102/torch1.8.0/index.html Collecting mmcv-full Downloading https://download.openmmlab.com/mmcv/dist/cu102/torch1.8.0/mmcv_full-1.6.1-cp38-cp38-manylinux1_x86_64.whl

Looking in links: https://download.openmmlab.com/mmcv/dist/cu102/torch1.8.0/index.html Collecting mmcv-full==1.6.0 Downloading mmcv-full-1.6.0.tar.gz

如需安装指定版本的 mmcv-full，例如安装 1.6.0 版本的 mmcv-full，可使用以下命令

```
mim install mmcv-full==1.6.0
```

注解： 如果你打算使用 `opencv-python-headless` 而不是 `opencv-python`，例如在一个很小的容器环境或者没有图形用户界面的服务器中，你可以先安装 `opencv-python-headless`，这样在安装 mmcv 依赖的过程中会跳过 `opencv-python`。

另外，如果安装依赖库的时间过长，可以指定 pypi 源

```
mim install mmcv-full -i https://pypi.tuna.tsinghua.edu.cn/simple
```

安装完成后可以运行 check_installation.py 脚本检查 mmcv-full 是否安装成功。

2.1.2 使用 pip 安装

使用以下命令查看 CUDA 和 PyTorch 的版本

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

根据系统的类型、CUDA 版本、PyTorch 版本以及 MMCV 版本选择相应的安装命令

如果在上面的下拉框中没有找到对应的版本，则可能是没有对应 PyTorch 或者 CUDA 或者 mmcv-full 版本的预编译包，此时，你可以源码安装 *mmcv-full*。

注解： PyTorch 在 1.x.0 和 1.x.1 之间通常是兼容的，故 mmcv-full 只提供 1.x.0 的编译包。如果你的 PyTorch 版本是 1.x.1，你可以放心地安装在 1.x.0 版本编译的 mmcv-full。例如，如果你的 PyTorch 版本是 1.8.1，你可以放心选择 1.8.x。

注解： 如果你打算使用 `opencv-python-headless` 而不是 `opencv-python`，例如在一个很小的容器环境或者没有图形用户界面的服务器中，你可以先安装 `opencv-python-headless`，这样在安装 mmcv 依赖的过程中会跳过 `opencv-python`。

另外，如果安装依赖库的时间过长，可以指定 pypi 源

```
pip install mmcv-full -f https://download.openmmlab.com/mmcv/dist/cu111/torch1.9.0/
↪index.html -i https://pypi.tuna.tsinghua.edu.cn/simple
```

安装完成后可以运行 check_installation.py 脚本检查 mmcv-full 是否安装成功。

2.1.3 使用 docker 镜像

先将算法库克隆到本地再构建镜像

```
git clone https://github.com/open-mmlab/mmcv.git && cd mmcv
docker build -t mmcv -f docker/release/Dockerfile .
```

也可以直接使用下面的命令构建镜像

```
docker build -t mmcv https://github.com/open-mmlab/mmcv.git#master:docker/release
```

Dockerfile 默认安装最新的 mmcv-full，如果你想要指定版本，可以使用下面的命令

```
docker image build -t mmcv -f docker/release/Dockerfile --build-arg MMCV=1.5.0 .
```

如果你想要使用其他版本的 PyTorch 和 CUDA，你可以在构建镜像时指定它们的版本。

例如指定 PyTorch 的版本是 1.11，CUDA 的版本是 11.3

```
docker build -t mmcv -f docker/release/Dockerfile \
    --build-arg PYTORCH=1.11.0 \
    --build-arg CUDA=11.3 \
    --build-arg CUDNN=8 \
    --build-arg MMCV=1.5.0 .
```

更多 PyTorch 和 CUDA 镜像可以点击 dockerhub/pytorch 查看。

2.2 安装 mmcv

如果你需要使用和 PyTorch 相关的模块，请确保 PyTorch 已经成功安装在环境中，可以参考 PyTorch 官方安装文档。

```
pip install mmcv
```

CHAPTER 3

从源码编译 MMCV

3.1 编译 mmcv-full

在编译 mmcv-full 之前，请确保 PyTorch 已经成功安装在环境中，可以参考 PyTorch 官方安装文档。可使用以下命令验证

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

注解:

- 如需编译 ONNX Runtime 自定义算子，请参考如何编译 ONNX Runtime 自定义算子
- 如需编译 TensorRT 自定义，请参考如何编译 MMCV 中的 TensorRT 插件

注解:

- 如果克隆代码仓库的速度过慢，可以使用以下命令克隆（注意：gitee 的 mmcv 不一定和 github 的保持一致，因为每天只同步一次）

```
git clone https://gitee.com/open-mmlab/mmcv.git
```

- 如果打算使用 `opencv-python-headless` 而不是 `opencv-python`，例如在一个很小的容器环境或者没有图形用户界面的服务器中，你可以先安装 `opencv-python-headless`，这样在安装 mmcv 依赖的过程中会跳过 `opencv-python`。

- 如果编译过程安装依赖库的时间过长，可以设置 pypi 源

```
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
```

3.1.1 在 Linux 上编译 mmcv-full

| TODO: 视频教程

1. 克隆代码仓库

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

2. 安装 `ninja` 和 `psutil` 以加快编译速度

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

3. 检查 nvcc 的版本（要求大于等于 9.2，如果没有 GPU，可以跳过）

```
nvcc --version
```

上述命令如果输出以下信息，表示 nvcc 的设置没有问题，否则需要设置 CUDA_HOME

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2020 NVIDIA Corporation
Built on Mon_Nov_30_19:08:53_PST_2020
Cuda compilation tools, release 11.2, V11.2.67
Build cuda_11.2.r11.2/compiler.29373293_0
```

注解： 如果想要支持 ROCm，可以参考 AMD ROCm 安装 ROCm。

---

4. 检查 gcc 的版本（要求大于等于 **5.4**）

```
gcc --version
```

5. 开始编译（预估耗时 10 分钟）

```
MMCV_WITH_OPS=1 pip install -e . -v
```

6. 验证安装

```
python .dev_scripts/check_installation.py
```

如果上述命令没有报错，说明安装成功。如有报错，请查看问题解决页面是否已经有解决方案。

如果没有找到解决方案，欢迎提 issue。

3.1.2 在 macOS 上编译 mmcv-full

| TODO: 视频教程

注解： 如果你使用的 mac 是 M1 芯片，请安装 PyTorch 的 nightly 版本，否则会遇到 issues#2218 中的问题。

---

1. 克隆代码仓库

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

2. 安装 `ninja` 和 `psutil` 以加快编译速度

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

3. 开始编译

```
MMCV_WITH_OPS=1 pip install -e .
```

4. 验证安装

```
python .dev_scripts/check_installation.py
```

如果上述命令没有报错，说明安装成功。如有报错，请查看问题解决页面是否已经有解决方案。

如果没有找到解决方案，欢迎提 issue。

3.1.3 在 Windows 上编译 mmcv-full

| TODO: 视频教程

在 Windows 上编译 mmcv-full 比 Linux 复杂，本节将一步步介绍如何在 Windows 上编译 mmcv-full。

依赖项

请先安装以下的依赖项：

- Git：安装期间，请选择 **add git to Path**
- Visual Studio Community 2019：用于编译 C++ 和 CUDA 代码
- Miniconda：包管理工具
- CUDA 10.2：如果只需要 CPU 版本可以不安装 CUDA，安装 CUDA 时，可根据需要进行自定义安装。如果已经安装新版本的显卡驱动，建议取消驱动程序的安装

注解： 如果不清楚如何安装以上依赖，请参考Windows 环境从零安装 mmcv-full。另外，你需要知道如何在 Windows 上设置变量环境，尤其是“PATH”的设置，以下安装过程都会用到。

通用步骤

1. 从 Windows 菜单启动 Anaconda 命令行

   如 Miniconda 安装程序建议，不要使用原始的 `cmd.exe` 或是 `powershell.exe`。命令行有两个版本，一个基于 PowerShell，一个基于传统的 `cmd.exe`。请注意以下说明都是使用的基于 PowerShell

2. 创建一个新的 Conda 环境

```
(base) PS C:\Users\xxx> conda create --name mmcv python=3.7
(base) PS C:\Users\xxx> conda activate mmcv  # 确保做任何操作前先激活环境
```

3. 安装 PyTorch 时，可以根据需要安装支持 CUDA 或不支持 CUDA 的版本

```
# CUDA version
(mmcv) PS C:\Users\xxx> conda install pytorch torchvision cudatoolkit=10.2 -c␣
↪pytorch
# CPU version
(mmcv) PS C:\Users\xxx> conda install install pytorch torchvision cpuonly -c␣
↪pytorch
```

4. 克隆代码仓库

```
(mmcv) PS C:\Users\xxx> git clone https://github.com/open-mmlab/mmcv.git
(mmcv) PS C:\Users\xxx> cd mmcv
```

5. 安装 `ninja` 和 `psutil` 以加快编译速度

```
(mmcv) PS C:\Users\xxx\mmcv> pip install -r requirements/optional.txt
```

6. 安装 mmcv 依赖

```
(mmcv) PS C:\Users\xxx\mmcv> pip install -r requirements/runtime.txt
```

7. 设置 MSVC 编译器

   设置环境变量。添加 `C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.27.29110\bin\Hostx86\x64` 到 PATH，则 `cl.exe` 可以在命令行中运行，如下所示。

```
(mmcv) PS C:\Users\xxx\mmcv> cl
Microsoft (R) C/C++ Optimizing  Compiler Version 19.27.29111 for x64
Copyright (C) Microsoft Corporation.   All rights reserved.

usage: cl [ option... ] filename... [ / link linkoption... ]
```

   为了兼容性，我们使用 x86-hosted 以及 x64-targeted 版本，即路径中的 `Hostx86\x64` 。

   因为 PyTorch 将解析 `cl.exe` 的输出以检查其版本，只有 utf-8 将会被识别，你可能需要将系统语言更改为英语。控制面板 -> 地区-> 管理-> 非 Unicode 来进行语言转换。

编译与安装 mmcv-full

mmcv-full 有两个版本：

- 只包含 CPU 算子的版本

  编译 CPU 算子，但只有 x86 将会被编译，并且编译版本只能在 CPU only 情况下运行

- 既包含 CPU 算子，又包含 CUDA 算子的版本

  同时编译 CPU 和 CUDA 算子，`ops` 模块的 x86 与 CUDA 的代码都可以被编译。同时编译的版本可以在 CUDA 上调用 GPU

CPU 版本

1. 设置环境变量

```
(mmcv) PS C:\Users\xxx\mmcv> $env:MMCV_WITH_OPS = 1
```

2. 编译安装

```
(mmcv) PS C:\Users\xxx\mmcv> python setup.py build_ext  # 如果成功, cl 将被启动用于编译算子
(mmcv) PS C:\Users\xxx\mmcv> python setup.py develop  # 安装
```

GPU 版本

1. 设置环境变量

```
(mmcv) PS C:\Users\xxx\mmcv> $env:MMCV_WITH_OPS = 1
```

2. 检查 CUDA_PATH 或者 CUDA_HOME 环境变量已经存在在 envs 之中

```
(mmcv) PS C:\Users\xxx\mmcv> ls env:

Name                           Value
----                           -----
CUDA_PATH                      C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\
↪v10.2
CUDA_PATH_V10_1                C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\
↪v10.1
CUDA_PATH_V10_2                C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\
↪v10.2
```

如果没有，你可以按照下面的步骤设置

```
(mmcv) PS C:\Users\xxx\mmcv> $env:CUDA_HOME = "C:\Program Files\NVIDIA GPU␣
↪Computing Toolkit\CUDA\v10.2"
# 或者
(mmcv) PS C:\Users\xxx\mmcv> $env:CUDA_HOME = $env:CUDA_PATH_V10_2   # CUDA_PATH_
↪V10_2 已经在环境变量中
```

3. 设置 CUDA 的目标架构

```
# 这里需要改成你的显卡对应的目标架构
(mmcv) PS C:\Users\xxx\mmcv> $env:TORCH_CUDA_ARCH_LIST="7.5"
```

注解： 可以点击 cuda-gpus 查看 GPU 的计算能力，也可以通过 CUDA 目录下的 deviceQuery.exe 工具查看

```
(mmcv) PS C:\Users\xxx\mmcv> &"C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\
↪v10.2\extras\demo_suite\deviceQuery.exe"
Device 0: "NVIDIA GeForce GTX 1660 SUPER"
CUDA Driver Version / Runtime Version          11.7 / 11.1
CUDA Capability Major/Minor version number:    7.5
```

上面的 7.5 表示目标架构。注意：需把上面命令的 v10.2 换成你的 CUDA 版本。

---

4. 编译安装

```
(mmcv) PS C:\Users\xxx\mmcv> python setup.py build_ext  # 如果成功，cl 将被启动用于编译算子
(mmcv) PS C:\Users\xxx\mmcv> python setup.py develop # 安装
```

注解： 如果你的 PyTorch 版本是 1.6.0，你可能会遇到一些 issue 提到的错误，你可以参考这个 pull request 修改本地环境的 PyTorch 源代码

验证安装

```
(mmcv) PS C:\Users\xxx\mmcv> python .dev_scripts/check_installation.py
```

如果上述命令没有报错，说明安装成功。如有报错，请查看问题解决页面是否已经有解决方案。如果没有找到解决方案，欢迎提 issue。

3.2 编译 mmcv

如果你需要使用和 PyTorch 相关的模块，请确保 PyTorch 已经成功安装在环境中，可以参考 PyTorch 官方安装文档。

1. 克隆代码仓库

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

2. 开始编译

```
pip install -e . -v
```

3. 验证安装

```
python -c 'import mmcv;print(mmcv.__version__)'
```

3.3 在 IPU 机器编译 mmcv

首先你需要有可用的 IPU 云机器，可以查看这里。

3.3.1 选项 1: 使用 Docker

1. 拉取镜像

```
docker pull graphcore/pytorch
```

2. 编译 mmcv

3.3.2 选项 2: 使用 SDK

1. 编译 mmcv
2. 参考 IPU PyTorch document 安装 sdk。

CHAPTER 4

解读文章汇总

这篇文章汇总了 OpenMMLab 解读的部分文章（更多文章和视频见 OpenMMLabCourse），如果您有推荐的文章（不一定是 OpenMMLab 发布的文章，可以是自己写的文章），非常欢迎提 Pull Request 添加到这里。

4.1 MMCV 解读文章

4.1.1 框架解读

- MMCV 核心组件分析 (一)：整体概述
- MMCV 核心组件分析 (二)：FileHandler
- MMCV 核心组件分析 (三): FileClient
- MMCV 核心组件分析 (四): Config
- MMCV 核心组件分析 (五): Registry
- MMCV 核心组件分析 (六): Hook
- MMCV 核心组件分析 (七): Runner
- MMCV Hook 食用指南
- PyTorch & MMCV Dispatcher 机制解析

4.1.2 工具解读

- 训练可视化工具哪款是你的菜？MMCV 一行代码随你挑

4.1.3 安装指南

- 久等了！Windows 平台 MMCV 的预编译包终于来了！
- Windows 环境从零安装 mmcv-full

4.1.4 知乎问答

- 深度学习科研，如何高效进行代码和实验管理?
- 深度学习方面的科研工作中的实验代码有什么规范和写作技巧？如何妥善管理实验数据?

4.2 下游算法库解读文章

- MMDetection

4.3 PyTorch 解读文章

- PyTorch1.11 亮点一览：TorchData、functorch、DDP 静态图
- PyTorch1.12 亮点一览：DataPipe + TorchArrow 新的数据加载与处理范式
- PyTorch 源码解读之 nn.Module：核心网络模块接口详解
- PyTorch 源码解读之 torch.autograd：梯度计算详解
- PyTorch 源码解读之 torch.utils.data：解析数据处理全流程
- PyTorch 源码解读之 torch.optim：优化算法接口详解
- PyTorch 源码解读之 DP & DDP：模型并行和分布式训练解析
- PyTorch 源码解读之 BN & SyncBN：BN 与多卡同步 BN 详解
- PyTorch 源码解读之 torch.cuda.amp: 自动混合精度详解
- PyTorch 源码解读之 cpp_extension：揭秘 C++/CUDA 算子实现和调用全流程
- PyTorch 源码解读之即时编译篇
- PyTorch 源码解读之分布式训练了解一下?
- PyTorch 源码解读之 torch.serialization & torch.hub

4.4 其他

- 困扰我 48 小时的深拷贝，今天终于…
- 拿什么拯救我的 4G 显卡
- 是谁偷偷动了我的 logger
- 三句话，让 logger 言听计从
- Logging 不为人知的二三事
- Type Hints 入门教程，让代码更加规范整洁
- 手把手教你如何高效地在 MMCV 中贡献算子
- OpenMMLab 支持 IPU 训练芯片
- 基于 MMCV 走上开源大佬之路?

CHAPTER 5

配置

`Config` 类用于操作配置文件，它支持从多种文件格式中加载配置，包括 **python**, **json** 和 **yaml**。它提供了类似字典对象的接口来获取和设置值。

以配置文件 `test.py` 为例

```
a = 1
b = dict(b1=[0, 1, 2], b2=None)
c = (1, 2)
d = 'string'
```

加载与使用配置文件

```
>>> cfg = Config.fromfile('test.py')
>>> print(cfg)
>>> dict(a=1,
...      b=dict(b1=[0, 1, 2], b2=None),
...      c=(1, 2),
...      d='string')
```

对于所有格式的配置文件，都支持一些预定义变量。它会将 `{{ var }}` 替换为实际值。

目前支持以下四个预定义变量：

`{{ fileDirname }}` - 当前打开文件的目录名，例如 /home/your-username/your-project/folder

`{{ fileBasename }}` - 当前打开文件的文件名，例如 file.ext

`{{ fileBasenameNoExtension }}` - 当前打开文件不包含扩展名的文件名，例如 file

`{{ fileExtname }}` - 当前打开文件的扩展名，例如.ext

这些变量名引用自 VS Code。

这里是一个带有预定义变量的配置文件的例子。

config_a.py

```
a = 1
b = './work_dir/{{ fileBasenameNoExtension }}'
c = '{{ fileExtname }}'
```

```
>>> cfg = Config.fromfile('./config_a.py')
>>> print(cfg)
>>> dict(a=1,
...      b='./work_dir/config_a',
...      c='.py')
```

对于所有格式的配置文件，都支持继承。为了重用其他配置文件的字段，需要指定 `_base_='./config_a.py'` 或者一个包含配置文件的列表 `_base_=['./config_a.py', './config_b.py']`。

这里有 4 个配置继承关系的例子。

config_a.py 作为基类配置文件

```
a = 1
b = dict(b1=[0, 1, 2], b2=None)
```

5.1 不含重复键值对从基类配置文件继承

config_b.py

```
_base_ = './config_a.py'
c = (1, 2)
d = 'string'
```

```
>>> cfg = Config.fromfile('./config_b.py')
>>> print(cfg)
>>> dict(a=1,
...      b=dict(b1=[0, 1, 2], b2=None),
...      c=(1, 2),
...      d='string')
```

在 config_b.py 里的新字段与在 config_a.py 里的旧字段拼接

5.2 含重复键值对从基类配置文件继承

config_c.py

```
_base_ = './config_a.py'
b = dict(b2=1)
c = (1, 2)
```

```
>>> cfg = Config.fromfile('./config_c.py')
>>> print(cfg)
>>> dict(a=1,
...       b=dict(b1=[0, 1, 2], b2=1),
...       c=(1, 2))
```

在基类配置文件：config_a 里的 b.b2=None 被配置文件：config_c.py 里的 b.b2=1 替代。

5.3 从具有忽略字段的配置文件继承

config_d.py

```
_base_ = './config_a.py'
b = dict(_delete_=True, b2=None, b3=0.1)
c = (1, 2)
```

```
>>> cfg = Config.fromfile('./config_d.py')
>>> print(cfg)
>>> dict(a=1,
...       b=dict(b2=None, b3=0.1),
...       c=(1, 2))
```

您还可以设置 _delete_=True 忽略基类配置文件中的某些字段。所有在 b 中的旧键 b1, b2, b3 将会被新键 b2, b3 所取代。

5.4 从多个基类配置文件继承（基类配置文件不应包含相同的键）

config_e.py

```
c = (1, 2)
d = 'string'
```

config_f.py

```
_base_ = ['./config_a.py', './config_e.py']
```

```
>>> cfg = Config.fromfile('./config_f.py')
>>> print(cfg)
>>> dict(a=1,
...      b=dict(b1=[0, 1, 2], b2=None),
...      c=(1, 2),
...      d='string')
```

5.5 从基类引用变量

您可以使用以下语法引用在基类中定义的变量。

base.py

```
item1 = 'a'
item2 = dict(item3 = 'b')
```

config_g.py

```
_base_ = ['./base.py']
item = dict(a = {{ _base_.item1 }}, b = {{ _base_.item2.item3 }})
```

```
>>> cfg = Config.fromfile('./config_g.py')
>>> print(cfg.pretty_text)
item1 = 'a'
item2 = dict(item3='b')
item = dict(a='a', b='b')
```

CHAPTER 6

注册器

MMCV 使用 注册器 来管理具有相似功能的不同模块, 例如, 检测器中的主干网络、头部、和模型颈部。在 OpenMMLab 家族中的绝大部分开源项目使用注册器去管理数据集和模型的模块，例如 MMDetection, MMDetection3D, MMClassification, MMEditing 等。

注解： 在 v1.5.1 版本开始支持注册函数的功能。

6.1 什么是注册器

在 MMCV 中，注册器可以看作类或函数到字符串的映射。一个注册器中的类或函数通常有相似的接口，但是可以实现不同的算法或支持不同的数据集。借助注册器，用户可以通过使用相应的字符串查找类或函数，并根据他们的需要实例化对应模块或调用函数获取结果。一个典型的案例是，OpenMMLab 中的大部分开源项目的配置系统，这些系统通过配置文件来使用注册器创建钩子、执行器、模型和数据集。可以在这里找到注册器接口使用文档。

使用 `registry`（注册器）管理代码库中的模型，需要以下三个步骤。

1. 创建一个构建方法（可选，在大多数情况下您可以只使用默认方法）
2. 创建注册器
3. 使用此注册器来管理模块

`Registry`（注册器）的参数 `build_func`（构建函数）用来自定义如何实例化类的实例或如何调用函数获取结果，默认使用 这里 实现的 `build_from_cfg`。

6.2 一个简单的例子

这里是一个使用注册器管理包中模块的简单示例。您可以在 OpenMMLab 开源项目中找到更多实例。

假设我们要实现一系列数据集转换器（Dataset Converter），用于将不同格式的数据转换为标准数据格式。我们先创建一个名为 converters 的目录作为包，在包中我们创建一个文件来实现构建器（builder），命名为 converters/builder.py，如下

```
from mmcv.utils import Registry
# 创建转换器（converter）的注册器（registry）
CONVERTERS = Registry('converter')
```

然后我们在包中可以实现不同的转换器（converter），其可以为类或函数。例如，在 converters/converter1.py 中实现 Converter1，在 converters/converter2.py 中实现 converter2。

```
# converter1.py
from .builder import CONVERTERS

# 使用注册器管理模块
@CONVERTERS.register_module()
class Converter1(object):
    def __init__(self, a, b):
        self.a = a
        self.b = b
```

```
# converter2.py
from .builder import CONVERTERS
from .converter1 import Converter1

# 使用注册器管理模块
@CONVERTERS.register_module()
def converter2(a, b):
    return Converter1(a, b)
```

使用注册器管理模块的关键步骤是，将实现的模块注册到注册表 CONVERTERS 中。通过 @CONVERTERS.register_module() 装饰所实现的模块，字符串到类或函数之间的映射就可以由 CONVERTERS 构建和维护，如下所示：

通过这种方式，就可以通过 CONVERTERS 建立字符串与类或函数之间的映射，如下所示：

```
'Converter1' -> <class 'Converter1'>
'converter2' -> <function 'converter2'>
```

注解： 只有模块所在的文件被导入时，注册机制才会被触发，所以您需要在某处导入该文件。更多详情请查

看 https://github.com/open-mmlab/mmdetection/issues/5974。

如果模块被成功注册了，你可以通过配置文件使用这个转换器（converter），如下所示：

```
converter1_cfg = dict(type='Converter1', a=a_value, b=b_value)
converter2_cfg = dict(type='converter2', a=a_value, b=b_value)
converter1 = CONVERTERS.build(converter1_cfg)
# returns the calling result
result = CONVERTERS.build(converter2_cfg)
```

6.3 自定义构建函数

假设我们想自定义 converters 的构建流程，我们可以实现一个自定义的 build_func （构建函数）并将其传递到注册器中。

```
from mmcv.utils import Registry

# 创建一个构建函数
def build_converter(cfg, registry, *args, **kwargs):
    cfg_ = cfg.copy()
    converter_type = cfg_.pop('type')
    if converter_type not in registry:
        raise KeyError(f'Unrecognized converter type {converter_type}')
    else:
        converter_cls = registry.get(converter_type)

    converter = converter_cls(*args, **kwargs, **cfg_)
    return converter

# 创建一个用于转换器 (converters) 的注册器，并传递 (registry) ``build_converter`` 函数
CONVERTERS = Registry('converter', build_func=build_converter)
```

注解： 注：在这个例子中，我们演示了如何使用参数：build_func 自定义构建类的实例的方法。该功能类似于默认的 build_from_cfg。在大多数情况下，默认就足够了。

build_model_from_cfg 也实现了在 nn.Sequential 中构建 PyTorch 模块，你可以直接使用它们。

6.4 注册器层结构

你也可以从多个 OpenMMLab 开源框架中构建模块，例如，你可以把所有 MMClassification 中的主干网络（backbone）用到 MMDetection 的目标检测中，你也可以融合 MMDetection 中的目标检测模型和 MMSegmentation 语义分割模型。

下游代码库中所有 MODELS 注册器都是 MMCV MODELS 注册器的子注册器。基本上，使用以下两种方法从子注册器或相邻兄弟注册器构建模块。

1. 从子注册器中构建

   例如：

   我们在 MMDetection 中定义：

```
from mmcv.utils import Registry
from mmcv.cnn import MODELS as MMCV_MODELS
MODELS = Registry('model', parent=MMCV_MODELS)

@MODELS.register_module()
class NetA(nn.Module):
    def forward(self, x):
        return x
```

   我们在 MMClassification 中定义：

```
from mmcv.utils import Registry
from mmcv.cnn import MODELS as MMCV_MODELS
MODELS = Registry('model', parent=MMCV_MODELS)

@MODELS.register_module()
class NetB(nn.Module):
    def forward(self, x):
        return x + 1
```

   我们可以通过以下代码在 MMDetection 或 MMClassification 中构建两个网络：

```
from mmdet.models import MODELS
net_a = MODELS.build(cfg=dict(type='NetA'))
net_b = MODELS.build(cfg=dict(type='mmcls.NetB'))
```

   或

```
from mmcls.models import MODELS
net_a = MODELS.build(cfg=dict(type='mmdet.NetA'))
net_b = MODELS.build(cfg=dict(type='NetB'))
```

2. 从父注册器中构建

   MMCV 中的共享 `MODELS` 注册器是所有下游代码库的父注册器（根注册器）：

```
from mmcv.cnn import MODELS as MMCV_MODELS
net_a = MMCV_MODELS.build(cfg=dict(type='mmdet.NetA'))
net_b = MMCV_MODELS.build(cfg=dict(type='mmcls.NetB'))
```

CHAPTER 7

执行器

执行器模块负责模型训练过程调度，主要目的是让用户使用更少的代码以及灵活可配置方式开启训练。其具备如下核心特性:

- 支持以 EpochBasedRunner 和 IterBasedRunner 为单位的迭代模式以满足不同场景
- 支持定制工作流以满足训练过程中各状态自由切换，目前支持训练和验证两个工作流。工作流可以简单理解为一个完成的训练和验证迭代过程。
- 配合各类默认和自定义 Hook，对外提供了灵活扩展能力

7.1 EpochBasedRunner

顾名思义，EpochBasedRunner 是指以 epoch 为周期的工作流，例如设置 workflow = [(‘train’, 2), (‘val’, 1)] 表示循环迭代地训练 2 个 epoch，然后验证 1 个 epoch。MMDetection 目标检测框架默认采用的是 EpochBasedRunner。

其抽象逻辑如下所示:

```
# 训练终止条件
while curr_epoch < max_epochs:
    # 遍历用户设置的工作流，例如 workflow = [('train', 2), ('val', 1)]
    for i, flow in enumerate(workflow):
        # mode 是工作流函数，例如 train, epochs 是迭代次数
        mode, epochs = flow
        # 要么调用 self.train()，要么调用 self.val()
```

(下页继续)

(续上页)

```
        epoch_runner = getattr(self, mode)
        # 运行对应工作流函数
        for _ in range(epochs):
            epoch_runner(data_loaders[i], **kwargs)
```

目前支持训练和验证两个工作流，以训练函数为例，其抽象逻辑是：

```
# epoch_runner 目前可以是 train 或者 val
def train(self, data_loader, **kwargs):
    # 遍历 dataset，共返回一个 epoch 的 batch 数据
    for i, data_batch in enumerate(data_loader):
        self.call_hook('before_train_iter')
        # 验证时候 train_mode=False
        self.run_iter(data_batch, train_mode=True, **kwargs)
        self.call_hook('after_train_iter')
   self.call_hook('after_train_epoch')
```

7.2 IterBasedRunner

不同于 EpochBasedRunner，IterBasedRunner 是指以 iter 为周期的工作流，例如设置 workflow = [('train', 2)，('val', 1)] 表示循环迭代的训练 2 个 iter，然后验证 1 个 iter，MMSegmentation 语义分割框架默认采用的是 IterBasedRunner。

其抽象逻辑如下所示：

```
# 虽然是 iter 单位，但是某些场合需要 epoch 信息，由 IterLoader 提供
iter_loaders = [IterLoader(x) for x in data_loaders]
# 训练终止条件
while curr_iter < max_iters:
    # 遍历用户设置的工作流，例如 workflow = [('train', 2), ('val', 1)]
    for i, flow in enumerate(workflow):
        # mode 是工作流函数，例如 train, iters 是迭代次数
        mode, iters = flow
        # 要么调用 self.train()，要么调用 self.val()
        iter_runner = getattr(self, mode)
        # 运行对应工作流函数
        for _ in range(iters):
            iter_runner(iter_loaders[i], **kwargs)
```

目前支持训练和验证两个工作流，以验证函数为例，其抽象逻辑是：

```
# iter_runner 目前可以是 train 或者 val
def val(self, data_loader, **kwargs):
    # 获取 batch 数据，用于一次迭代
    data_batch = next(data_loader)
    self.call_hook('before_val_iter')
    outputs = self.model.val_step(data_batch, self.optimizer, **kwargs)
    self.outputs = outputs
    self.call_hook('after_val_iter')
```

除了上述基础功能外，EpochBasedRunner 和 IterBasedRunner 还提供了 resume 、save_checkpoint 和注册 hook 功能。

7.3 一个简单例子

以最常用的分类任务为例详细说明 runner 的使用方法。开启任何一个训练任务，都需要包括如下步骤：

(1) dataloader、model 和优化器等类初始化

```
# 模型类初始化
model=...
# 优化器类初始化，典型值 cfg.optimizer = dict(type='SGD', lr=0.1, momentum=0.9, weight_
↪decay=0.0001)
optimizer = build_optimizer(model, cfg.optimizer)
# 工作流对应的 dataloader 初始化
data_loaders = [
        build_dataloader(
            ds,
            cfg.data.samples_per_gpu,
            cfg.data.workers_per_gpu,
            ...) for ds in dataset
    ]
```

(2) runner 类初始化

```
runner = build_runner(
    # cfg.runner 典型配置为
    # runner = dict(type='EpochBasedRunner', max_epochs=200)
    cfg.runner,
    default_args=dict(
        model=model,
        batch_processor=None,
        optimizer=optimizer,
        logger=logger))
```

(3) 注册默认训练所必须的 hook，和用户自定义 hook

```
# 注册定制必需的 hook
runner.register_training_hooks(
    # lr 相关配置，典型为
    # lr_config = dict(policy='step', step=[100, 150])
    cfg.lr_config,
    # 优化相关配置，例如 grad_clip 等
    optimizer_config,
    # 权重保存相关配置，典型为
    # checkpoint_config = dict(interval=1)，每个单位都保存权重
    cfg.checkpoint_config,
    # 日志相关配置
    cfg.log_config,
    ...)

# 注册用户自定义 hook
# 例如想使用 ema 功能，则可以设置 custom_hooks=[dict(type='EMAHook')]
if cfg.get('custom_hooks', None):
    custom_hooks = cfg.custom_hooks
    for hook_cfg in cfg.custom_hooks:
        hook_cfg = hook_cfg.copy()
        priority = hook_cfg.pop('priority', 'NORMAL')
        hook = build_from_cfg(hook_cfg, HOOKS)
        runner.register_hook(hook, priority=priority)
```

然后可以进行 resume 或者 load_checkpoint 对权重进行加载。

(4) 开启训练流

```
# workflow 典型为 workflow = [('train', 1)]
# 此时就真正开启了训练
runner.run(data_loaders, cfg.workflow)
```

关于 workflow 设置，以 `EpochBasedRunner` 为例，详情如下：

- 假设只想运行训练工作流，则可以设置 workflow = [（‘train’, 1)]，表示只进行迭代训练
- 假设想运行训练和验证工作流，则可以设置 workflow = [（‘train’, 3), (‘val’, 1)]，表示先训练 3 个 epoch ，然后切换到 val 工作流，运行 1 个 epoch，然后循环，直到训练 epoch 次数达到指定值
- 工作流设置还自由定制，例如你可以先验证再训练 workflow = [（‘val’, 1), (‘train’, 1)]

上述代码都已经封装到了各个代码库的 train.py 中，用户只需要设置相应的配置即可，上述流程会自动运行。

CHAPTER 8

文件输入输出

文件输入输出模块提供了两个通用的 API 接口用于读取和保存不同格式的文件。

注解： 在 v1.3.16 及之后的版本中，IO 模块支持从不同后端读取数据并支持将数据至不同后端。更多细节请访问 PR #1330。

8.1 读取和保存数据

mmcv 提供了一个通用的 api 用于读取和保存数据，目前支持的格式有 json、yaml 和 pickle。

8.1.1 从硬盘读取数据或者将数据保存至硬盘

```
import mmcv

# 从文件中读取数据
data = mmcv.load('test.json')
data = mmcv.load('test.yaml')
data = mmcv.load('test.pkl')
# 从文件对象中读取数据
with open('test.json', 'r') as f:
    data = mmcv.load(f, file_format='json')
```

(下页继续)

(续上页)

```
# 将数据序列化为字符串
json_str = mmcv.dump(data, file_format='json')

# 将数据保存至文件 (根据文件名后缀反推文件类型)
mmcv.dump(data, 'out.pkl')

# 将数据保存至文件对象
with open('test.yaml', 'w') as f:
    data = mmcv.dump(data, f, file_format='yaml')
```

8.1.2 从其他后端加载或者保存至其他后端

```
import mmcv

# 从 s3 文件读取数据
data = mmcv.load('s3://bucket-name/test.json')
data = mmcv.load('s3://bucket-name/test.yaml')
data = mmcv.load('s3://bucket-name/test.pkl')

# 将数据保存至 s3 文件 (根据文件名后缀反推文件类型)
mmcv.dump(data, 's3://bucket-name/out.pkl')
```

我们提供了易于拓展的方式以支持更多的文件格式。我们只需要创建一个继承自 `BaseFileHandler` 的文件句柄类并将其注册到 `mmcv` 中即可。句柄类至少需要重写三个方法。

```
import mmcv

# 支持为文件句柄类注册多个文件格式
# @mmcv.register_handler(['txt', 'log'])
@mmcv.register_handler('txt')
class TxtHandler1(mmcv.BaseFileHandler):

    def load_from_fileobj(self, file):
        return file.read()

    def dump_to_fileobj(self, obj, file):
        file.write(str(obj))

    def dump_to_str(self, obj, **kwargs):
        return str(obj)
```

以 `PickleHandler` 为例

```
import pickle

class PickleHandler(mmcv.BaseFileHandler):

    def load_from_fileobj(self, file, **kwargs):
        return pickle.load(file, **kwargs)

    def load_from_path(self, filepath, **kwargs):
        return super(PickleHandler, self).load_from_path(
            filepath, mode='rb', **kwargs)

    def dump_to_str(self, obj, **kwargs):
        kwargs.setdefault('protocol', 2)
        return pickle.dumps(obj, **kwargs)

    def dump_to_fileobj(self, obj, file, **kwargs):
        kwargs.setdefault('protocol', 2)
        pickle.dump(obj, file, **kwargs)

    def dump_to_path(self, obj, filepath, **kwargs):
        super(PickleHandler, self).dump_to_path(
            obj, filepath, mode='wb', **kwargs)
```

8.2 读取文件并返回列表或字典

例如，a.txt 是文本文件，一共有 5 行内容。

```
a
b
c
d
e
```

8.2.1 从硬盘读取

使用 list_from_file 读取 a.txt

```
>>> mmcv.list_from_file('a.txt')
['a', 'b', 'c', 'd', 'e']
>>> mmcv.list_from_file('a.txt', offset=2)
['c', 'd', 'e']
```

(下页继续)

(续上页)

```
>>> mmcv.list_from_file('a.txt', max_num=2)
['a', 'b']
>>> mmcv.list_from_file('a.txt', prefix='/mnt/')
['/mnt/a', '/mnt/b', '/mnt/c', '/mnt/d', '/mnt/e']
```

同样，b.txt 也是文本文件，一共有 3 行内容

```
1 cat
2 dog cow
3 panda
```

使用 dict_from_file 读取 b.txt

```
>>> mmcv.dict_from_file('b.txt')
{'1': 'cat', '2': ['dog', 'cow'], '3': 'panda'}
>>> mmcv.dict_from_file('b.txt', key_type=int)
{1: 'cat', 2: ['dog', 'cow'], 3: 'panda'}
```

8.2.2 从其他后端读取

使用 list_from_file 读取 s3://bucket-name/a.txt

```
>>> mmcv.list_from_file('s3://bucket-name/a.txt')
['a', 'b', 'c', 'd', 'e']
>>> mmcv.list_from_file('s3://bucket-name/a.txt', offset=2)
['c', 'd', 'e']
>>> mmcv.list_from_file('s3://bucket-name/a.txt', max_num=2)
['a', 'b']
>>> mmcv.list_from_file('s3://bucket-name/a.txt', prefix='/mnt/')
['/mnt/a', '/mnt/b', '/mnt/c', '/mnt/d', '/mnt/e']
```

使用 dict_from_file 读取 b.txt

```
>>> mmcv.dict_from_file('s3://bucket-name/b.txt')
{'1': 'cat', '2': ['dog', 'cow'], '3': 'panda'}
>>> mmcv.dict_from_file('s3://bucket-name/b.txt', key_type=int)
{1: 'cat', 2: ['dog', 'cow'], 3: 'panda'}
```

8.3 读取和保存权重文件

8.3.1 从硬盘读取权重文件或者将权重文件保存至硬盘

我们可以通过下面的方式从磁盘读取权重文件或者将权重文件保存至磁盘

```
import torch

filepath1 = '/path/of/your/checkpoint1.pth'
filepath2 = '/path/of/your/checkpoint2.pth'
# 从 filepath1 读取权重文件
checkpoint = torch.load(filepath1)
# 将权重文件保存至 filepath2
torch.save(checkpoint, filepath2)
```

MMCV 提供了很多后端，HardDiskBackend 是其中一个，我们可以通过它来读取或者保存权重文件。

```
import io
from mmcv.fileio.file_client import HardDiskBackend

disk_backend = HardDiskBackend()
with io.BytesIO(disk_backend.get(filepath1)) as buffer:
    checkpoint = torch.load(buffer)
with io.BytesIO() as buffer:
    torch.save(checkpoint, f)
    disk_backend.put(f.getvalue(), filepath2)
```

如果我们想在接口中实现根据文件路径自动选择对应的后端，我们可以使用 FileClient。例如，我们想实现两个方法，分别是读取权重以及保存权重，它们需支持不同类型的文件路径，可以是磁盘路径，也可以是网络路径或者其他路径。

```
from mmcv.fileio.file_client import FileClient

def load_checkpoint(path):
    file_client = FileClient.infer(uri=path)
    with io.BytesIO(file_client.get(path)) as buffer:
        checkpoint = torch.load(buffer)
    return checkpoint

def save_checkpoint(checkpoint, path):
    with io.BytesIO() as buffer:
        torch.save(checkpoint, buffer)
        file_client.put(buffer.getvalue(), path)
```

(下页继续)

(续上页)

```
file_client = FileClient.infer_client(uri=filepath1)
checkpoint = load_checkpoint(filepath1)
save_checkpoint(checkpoint, filepath2)
```

8.3.2 从网络远端读取权重文件

注解: 目前只支持从网络远端读取权重文件，暂不支持将权重文件写入网络远端

```
import io
import torch
from mmcv.fileio.file_client import HTTPBackend, FileClient

filepath = 'http://path/of/your/checkpoint.pth'
checkpoint = torch.utils.model_zoo.load_url(filepath)

http_backend = HTTPBackend()
with io.BytesIO(http_backend.get(filepath)) as buffer:
    checkpoint = torch.load(buffer)

file_client = FileClient.infer_client(uri=filepath)
with io.BytesIO(file_client.get(filepath)) as buffer:
    checkpoint = torch.load(buffer)
```

CHAPTER 9

数据处理

9.1 图像

图像模块提供了一些图像预处理的函数，该模块依赖 opencv 。

9.1.1 读取/保存/显示

使用 imread 和 imwrite 函数可以读取和保存图像。

```
import mmcv

img = mmcv.imread('test.jpg')
img = mmcv.imread('test.jpg', flag='grayscale')
img_ = mmcv.imread(img)  # 相当于什么也没做
mmcv.imwrite(img, 'out.jpg')
```

从二进制中读取图像

```
with open('test.jpg', 'rb') as f:
    data = f.read()
img = mmcv.imfrombytes(data)
```

显示图像文件或已读取的图像

```
mmcv.imshow('tests/data/color.jpg')

for i in range(10):
    img = np.random.randint(256, size=(100, 100, 3), dtype=np.uint8)
    mmcv.imshow(img, win_name='test image', wait_time=200)
```

9.1.2 色彩空间转换

支持的转换函数：

- bgr2gray
- gray2bgr
- bgr2rgb
- rgb2bgr
- bgr2hsv
- hsv2bgr

```
img = mmcv.imread('tests/data/color.jpg')
img1 = mmcv.bgr2rgb(img)
img2 = mmcv.rgb2gray(img1)
img3 = mmcv.bgr2hsv(img)
```

9.1.3 缩放

有三种缩放图像的方法。所有以 `imresize_*` 开头的函数都有一个 `return_scale` 参数，如果该参数为 `False` ，函数的返回值只有调整之后的图像，否则是一个元组 `(resized_img, scale)` 。

```
# 缩放图像至给定的尺寸
mmcv.imresize(img, (1000, 600), return_scale=True)

# 缩放图像至与给定的图像同样的尺寸
mmcv.imresize_like(img, dst_img, return_scale=False)

# 以一定的比例缩放图像
mmcv.imrescale(img, 0.5)

# 缩放图像至最长的边不大于 1000、最短的边不大于 800 并且没有改变图像的长宽比
mmcv.imrescale(img, (1000, 800))
```

9.1.4 旋转

我们可以使用 imrotate 旋转图像一定的角度。旋转的中心需要指定，默认值是原始图像的中心。有两种旋转的模式，一种保持图像的尺寸不变，因此旋转后原始图像中的某些部分会被裁剪，另一种是扩大图像的尺寸进而保留完整的原始图像。

```
img = mmcv.imread('tests/data/color.jpg')

# 顺时针旋转图像 30 度
img_ = mmcv.imrotate(img, 30)

# 逆时针旋转图像 90 度
img_ = mmcv.imrotate(img, -90)

# 顺时针旋转图像 30 度并且缩放图像为原始图像的 1.5 倍
img_ = mmcv.imrotate(img, 30, scale=1.5)

# 以坐标 (100, 100) 为中心顺时针旋转图像 30 度
img_ = mmcv.imrotate(img, 30, center=(100, 100))

# 顺时针旋转图像 30 度并扩大图像的尺寸
img_ = mmcv.imrotate(img, 30, auto_bound=True)
```

9.1.5 翻转

我们可以使用 imflip 翻转图像。

```
img = mmcv.imread('tests/data/color.jpg')

# 水平翻转图像
mmcv.imflip(img)

# 垂直翻转图像
mmcv.imflip(img, direction='vertical')
```

9.1.6 裁剪

imcrop 可以裁剪图像的一个或多个区域，每个区域用左上角和右下角坐标表示，形如 (x1, y1, x2, y2)

```
import mmcv
import numpy as np
```

(下页继续)

(续上页)

```
img = mmcv.imread('tests/data/color.jpg')

# 裁剪区域 (10, 10, 100, 120)
bboxes = np.array([10, 10, 100, 120])
patch = mmcv.imcrop(img, bboxes)

# 裁剪两个区域，分别是 (10, 10, 100, 120) 和 (0, 0, 50, 50)
bboxes = np.array([[10, 10, 100, 120], [0, 0, 50, 50]])
patches = mmcv.imcrop(img, bboxes)

# 裁剪两个区域并且缩放区域 1.2 倍
patches = mmcv.imcrop(img, bboxes, scale=1.2)
```

9.1.7 填充

impad and impad_to_multiple 可以用给定的值将图像填充至给定的尺寸。

```
img = mmcv.imread('tests/data/color.jpg')

# 用给定值将图像填充至 (1000, 1200)
img_ = mmcv.impad(img, shape=(1000, 1200), pad_val=0)

# 用给定值分别填充图像的 3 个通道至 (1000, 1200)
img_ = mmcv.impad(img, shape=(1000, 1200), pad_val=(100, 50, 200))

# 用给定值填充图像的左、右、上、下四条边
img_ = mmcv.impad(img, padding=(10, 20, 30, 40), pad_val=0)

# 用 3 个值分别填充图像的左、右、上、下四条边的 3 个通道
img_ = mmcv.impad(img, padding=(10, 20, 30, 40), pad_val=(100, 50, 200))

# 将图像的四条边填充至能够被给定值整除
img_ = mmcv.impad_to_multiple(img, 32)
```

9.2 视频

视频模块提供了以下的功能：

- 一个 VideoReader 类，具有友好的 API 接口可以读取和转换视频
- 一些编辑视频的方法，包括 cut ， concat ， resize
- 光流的读取/保存/变换

9.2.1 VideoReader

VideoReader 类提供了和序列一样的接口去获取视频帧。该类会缓存所有被访问过的帧。

```
video = mmcv.VideoReader('test.mp4')

# 获取基本的信息
print(len(video))
print(video.width, video.height, video.resolution, video.fps)

# 遍历所有的帧
for frame in video:
    print(frame.shape)

# 读取下一帧
img = video.read()

# 使用索引获取帧
img = video[100]

# 获取指定范围的帧
img = video[5:10]
```

将视频切成帧并保存至给定目录或者从给定目录中生成视频。

```
# 将视频切成帧并保存至目录
video = mmcv.VideoReader('test.mp4')
video.cvt2frames('out_dir')

# 从给定目录中生成视频
mmcv.frames2video('out_dir', 'test.avi')
```

9.2.2 编辑函数

有几个用于编辑视频的函数，这些函数是对 ffmpeg 的封装。

```
# 裁剪视频
mmcv.cut_video('test.mp4', 'clip1.mp4', start=3, end=10, vcodec='h264')

# 将多个视频拼接成一个视频
mmcv.concat_video(['clip1.mp4', 'clip2.mp4'], 'joined.mp4', log_level='quiet')

# 将视频缩放至给定的尺寸
mmcv.resize_video('test.mp4', 'resized1.mp4', (360, 240))
```

(下页继续)

(续上页)

```
# 将视频缩放至给定的倍率
mmcv.resize_video('test.mp4', 'resized2.mp4', ratio=2)
```

9.2.3 光流

mmcv 提供了以下用于操作光流的函数：

- 读取/保存
- 可视化
- 流变换

我们提供了两种将光流 dump 到文件的方法，分别是非压缩和压缩的方法。非压缩的方法直接将浮点数值的光流保存至二进制文件，虽然光流无损但文件会比较大。而压缩的方法先量化光流至 0-255 整形数值再保存为 jpeg 图像。光流的 x 维度和 y 维度会被拼接到图像中。

1. 读取/保存

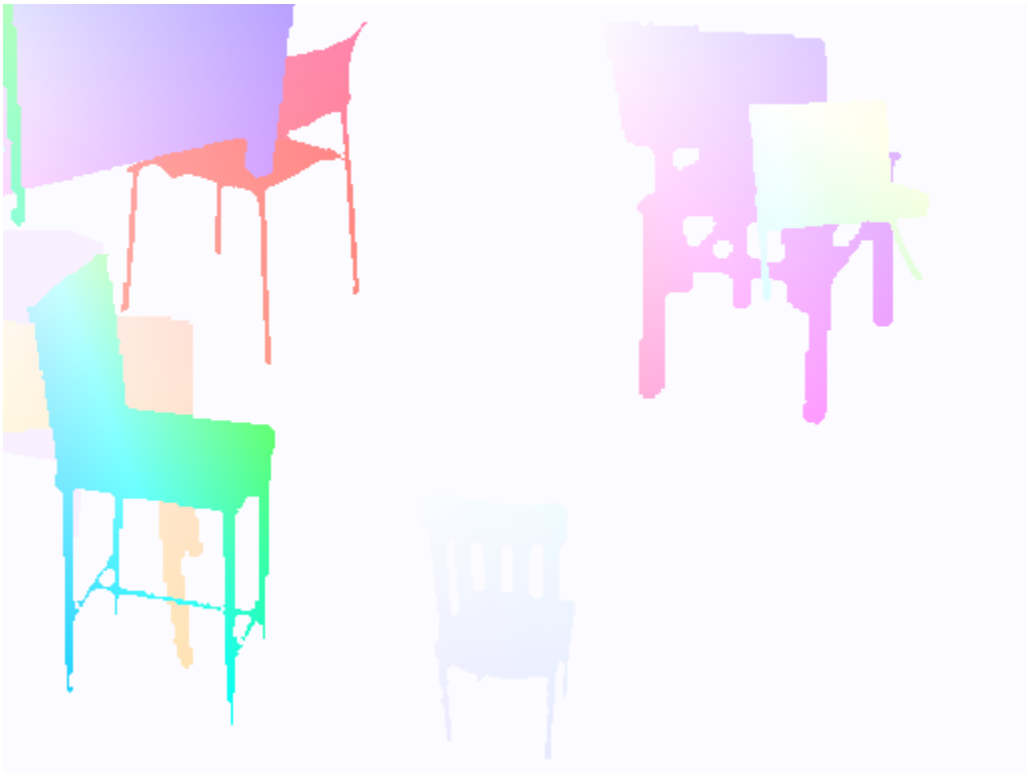
```
flow = np.random.rand(800, 600, 2).astype(np.float32)
# 保存光流到 flo 文件 (~3.7M)
mmcv.flowwrite(flow, 'uncompressed.flo')
# 保存光流为 jpeg 图像 (~230K)，图像的尺寸为 (800, 1200)
mmcv.flowwrite(flow, 'compressed.jpg', quantize=True, concat_axis=1)

# 读取光流文件，以下两种方式读取的光流尺寸均为 (800, 600, 2)
flow = mmcv.flowread('uncompressed.flo')
flow = mmcv.flowread('compressed.jpg', quantize=True, concat_axis=1)
```

2. 可视化

使用 mmcv.flowshow() 可视化光流

```
mmcv.flowshow(flow)
```



1. 流变换

```
img1 = mmcv.imread('img1.jpg')
flow = mmcv.flowread('flow.flo')
warpped_img2 = mmcv.flow_warp(img1, flow)
```

img1 (左) and img2 (右)

光流 (img2 -> img1)

变换后的图像和真实图像的差异

CHAPTER 10

可视化

mmcv 可以展示图像以及标注（目前只支持标注框）

```
# 展示图像文件
mmcv.imshow('a.jpg')

# 展示已加载的图像
img = np.random.rand(100, 100, 3)
mmcv.imshow(img)

# 展示带有标注框的图像
img = np.random.rand(100, 100, 3)
bboxes = np.array([[0, 0, 50, 50], [20, 20, 60, 60]])
mmcv.imshow_bboxes(img, bboxes)
```

mmcv 也可以展示特殊的图像，例如光流

```
flow = mmcv.flowread('test.flo')
mmcv.flowshow(flow)
```

CHAPTER 11

卷积神经网络

我们为卷积神经网络提供了一些构建模块，包括层构建、模块组件和权重初始化。

11.1 网络层的构建

在运行实验时，我们可能需要尝试同属一种类型但不同配置的层，但又不希望每次都修改代码。于是我们提供一些层构建方法，可以从字典构建层，字典可以在配置文件中配置，也可以通过命令行参数指定。

11.1.1 用法

一个简单的例子:

```
cfg = dict(type='Conv3d')
layer = build_conv_layer(cfg, in_channels=3, out_channels=8, kernel_size=3)
```

- `build_conv_layer`: 支持的类型包括 Conv1d、Conv2d、Conv3d、Conv (Conv 是 Conv2d 的别名）
- `build_norm_layer`: 支持的类型包括 BN1d、BN2d、BN3d、BN (alias for BN2d)、SyncBN、GN、LN、IN1d、IN2d、IN3d、IN（IN 是 IN2d 的别名）
- `build_activation_layer`: 支持的类型包括 ReLU、LeakyReLU、PReLU、RReLU、ReLU6、ELU、Sigmoid、Tanh、GELU
- `build_upsample_layer`: 支持的类型包括 nearest、bilinear、deconv、pixel_shuffle
- `build_padding_layer`: 支持的类型包括 zero、reflect、replicate

11.1.2 拓展

我们还允许自定义层和算子来扩展构建方法。

1. 编写和注册自己的模块：

```
from mmcv.cnn import UPSAMPLE_LAYERS

@UPSAMPLE_LAYERS.register_module()
class MyUpsample:

    def __init__(self, scale_factor):
        pass

    def forward(self, x):
        pass
```

2. 在某处导入 MyUpsample （例如 __init__.py ）然后使用它：

```
cfg = dict(type='MyUpsample', scale_factor=2)
layer = build_upsample_layer(cfg)
```

11.2 模块组件

我们还提供了常用的模块组件，以方便网络构建。卷积组件 ConvModule 由 convolution、normalization 以及 activation layers 组成，更多细节请参考 ConvModule api。

```
# conv + bn + relu
conv = ConvModule(3, 8, 2, norm_cfg=dict(type='BN'))
# conv + gn + relu
conv = ConvModule(3, 8, 2, norm_cfg=dict(type='GN', num_groups=2))
# conv + relu
conv = ConvModule(3, 8, 2)
# conv
conv = ConvModule(3, 8, 2, act_cfg=None)
# conv + leaky relu
conv = ConvModule(3, 8, 3, padding=1, act_cfg=dict(type='LeakyReLU'))
# bn + conv + relu
conv = ConvModule(
    3, 8, 2, norm_cfg=dict(type='BN'), order=('norm', 'conv', 'act'))
```

11.3 Weight initialization

实现细节可以在 mmcv/cnn/utils/weight_init.py 中找到

在训练过程中，适当的初始化策略有利于加快训练速度或者获得更高的性能。在 MMCV 中，我们提供了一些常用的方法来初始化模块，比如 `nn.Conv2d` 模块。当然，我们也提供了一些高级 API，可用于初始化包含一个或多个模块的模型。

11.3.1 Initialization functions

以函数的方式初始化 `nn.Module` ，例如 `nn.Conv2d` 、`nn.Linear` 等。

我们提供以下初始化方法，

- constant_init

  使用给定常量值初始化模型参数

```
>>> import torch.nn as nn
>>> from mmcv.cnn import constant_init
>>> conv1 = nn.Conv2d(3, 3, 1)
>>> # constant_init(module, val, bias=0)
>>> constant_init(conv1, 1, 0)
>>> conv1.weight
```

- xavier_init

  按照 Understanding the difficulty of training deep feedforward neural networks - Glorot, X. & Bengio, Y. (2010) 描述的方法初始化模型参数

```
>>> import torch.nn as nn
>>> from mmcv.cnn import xavier_init
>>> conv1 = nn.Conv2d(3, 3, 1)
>>> # xavier_init(module, gain=1, bias=0, distribution='normal')
>>> xavier_init(conv1, distribution='normal')
```

- normal_init

  使用正态分布（高斯分布）初始化模型参数

```
>>> import torch.nn as nn
>>> from mmcv.cnn import normal_init
>>> conv1 = nn.Conv2d(3, 3, 1)
>>> # normal_init(module, mean=0, std=1, bias=0)
>>> normal_init(conv1, std=0.01, bias=0)
```

- uniform_init

使用均匀分布初始化模型参数

```
>>> import torch.nn as nn
>>> from mmcv.cnn import uniform_init
>>> conv1 = nn.Conv2d(3, 3, 1)
>>> # uniform_init(module, a=0, b=1, bias=0)
>>> uniform_init(conv1, a=0, b=1)
```

- kaiming_init

  按照 Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification - He, K. et al. (2015) 描述的方法来初始化模型参数。

```
>>> import torch.nn as nn
>>> from mmcv.cnn import kaiming_init
>>> conv1 = nn.Conv2d(3, 3, 1)
>>> # kaiming_init(module, a=0, mode='fan_out', nonlinearity='relu', bias=0,␣
↪distribution='normal')
>>> kaiming_init(conv1)
```

- caffe2_xavier_init

  caffe2 中实现的 `xavier initialization`，对应于 PyTorch 中的 `kaiming_uniform_`

```
>>> import torch.nn as nn
>>> from mmcv.cnn import caffe2_xavier_init
>>> conv1 = nn.Conv2d(3, 3, 1)
>>> # caffe2_xavier_init(module, bias=0)
>>> caffe2_xavier_init(conv1)
```

- bias_init_with_prob

  根据给定的概率初始化 `conv/fc`, 这在 Focal Loss for Dense Object Detection 提出。

```
>>> from mmcv.cnn import bias_init_with_prob
>>> # bias_init_with_prob is proposed in Focal Loss
>>> bias = bias_init_with_prob(0.01)
>>> bias
-4.59511985013459
```

11.3.2 Initializers and configs

在初始化方法的基础上，我们定义了相应的初始化类，并将它们注册到 `INITIALIZERS` 中，这样我们就可以使用 `config` 配置来初始化模型了。

我们提供以下初始化类：

- ConstantInit
- XavierInit
- NormalInit
- UniformInit
- KaimingInit
- Caffe2XavierInit
- PretrainedInit

接下来详细介绍 `initialize` 的使用方法

1. 通过关键字 `layer` 来初始化模型

   如果我们只定义了关键字 `layer`，那么只初始化 `layer` 中包含的层。

   注意：关键字 `layer` 支持的模块是带有 weights 和 bias 属性的 PyTorch 模块，所以不支持 `MultiheadAttention layer`

- 定义关键字 `layer` 列表并使用相同相同配置初始化模块

```
import torch.nn as nn
from mmcv.cnn import initialize

class FooNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.feat = nn.Conv1d(3, 1, 3)
        self.reg = nn.Conv2d(3, 3, 3)
        self.cls = nn.Linear(1, 2)

model = FooNet()
init_cfg = dict(type='Constant', layer=['Conv1d', 'Conv2d', 'Linear'], val=1)
# 使用相同的配置初始化整个模块
initialize(model, init_cfg)
# model.feat.weight
# Parameter containing:
# tensor([[[1., 1., 1.],
#          [1., 1., 1.],
#          [1., 1., 1.]]], requires_grad=True)
```

- 定义关键字 layer 用于初始化不同配置的层

```
import torch.nn as nn
from mmcv.cnn.utils import initialize

class FooNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.feat = nn.Conv1d(3, 1, 3)
        self.reg = nn.Conv2d(3, 3, 3)
        self.cls = nn.Linear(1,2)

model = FooNet()
init_cfg = [dict(type='Constant', layer='Conv1d', val=1),
            dict(type='Constant', layer='Conv2d', val=2),
            dict(type='Constant', layer='Linear', val=3)]
# nn.Conv1d 使用 dict(type='Constant', val=1) 初始化
# nn.Conv2d 使用 dict(type='Constant', val=2) 初始化
# nn.Linear 使用 dict(type='Constant', val=3) 初始化
initialize(model, init_cfg)
# model.reg.weight
# Parameter containing:
# tensor([[[[2., 2., 2.],
#           [2., 2., 2.],
#           [2., 2., 2.]],
#          ...,
#          [[2., 2., 2.],
#           [2., 2., 2.],
#           [2., 2., 2.]]]], requires_grad=True)
```

2. 定义关键字 override 初始化模型

- 当用属性名初始化某个特定部分时, 我们可以使用关键字 override, 关键字 override 对应的 Value 会替代 init_cfg 中相应的值

```
import torch.nn as nn
from mmcv.cnn import initialize

class FooNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.feat = nn.Conv1d(3, 1, 3)
        self.reg = nn.Conv2d(3, 3, 3)
        self.cls = nn.Sequential(nn.Conv1d(3, 1, 3), nn.Linear(1,2))
```

(下页继续)

(续上页)

```
# 如果我们想将模型的权重初始化为 1，将偏差初始化为 2
# 但希望 `reg` 中的权重为 3，偏差为 4，则我们可以使用关键字 override

model = FooNet()
init_cfg = dict(type='Constant', layer=['Conv1d','Conv2d'], val=1, bias=2,
                override=dict(type='Constant', name='reg', val=3, bias=4))
#  使用 dict(type='Constant', val=1, bias=2) 来初始化 self.feat and self.cls
# 使用 dict(type='Constant', val=3, bias=4) 来初始化 'reg' 模块。
initialize(model, init_cfg)
# model.reg.weight
# Parameter containing:
# tensor([[[[3., 3., 3.],
#           [3., 3., 3.],
#           [3., 3., 3.]],
#           ...,
#           [[3., 3., 3.],
#            [3., 3., 3.],
#            [3., 3., 3.]]]], requires_grad=True)
```

- 如果 init_cfg 中的关键字 `layer` 为 None，则只初始化在关键字 override 中的子模块，并且省略 override 中的 type 和其他参数

```
model = FooNet()
init_cfg = dict(type='Constant', val=1, bias=2, override=dict(name='reg'))
# self.feat 和 self.cls 使用 pyTorch 默认的初始化
# 将使用 dict(type='Constant', val=1, bias=2) 初始化名为 'reg' 的模块
initialize(model, init_cfg)
# model.reg.weight
# Parameter containing:
# tensor([[[[1., 1., 1.],
#           [1., 1., 1.],
#           [1., 1., 1.]],
#           ...,
#           [[1., 1., 1.],
#            [1., 1., 1.],
#            [1., 1., 1.]]]], requires_grad=True)
```

- 如果我们没有定义关键字 `layer` 或 `override`，将不会初始化任何东西
- 关键字 `override` 的无效用法

```
# 没有重写任何子模块
init_cfg = dict(type='Constant', layer=['Conv1d','Conv2d'],
                val=1, bias=2,
```

(下页继续)

(续上页)

```
                override=dict(type='Constant', val=3, bias=4))

# 没有指定 type，即便有其他参数，也是无效的。
init_cfg = dict(type='Constant', layer=['Conv1d','Conv2d'],
                val=1, bias=2,
                override=dict(name='reg', val=3, bias=4))
```

3. 用预训练模型初始化

```
import torch.nn as nn
import torchvision.models as models
from mmcv.cnn import initialize

# 使用预训练模型来初始化
model = models.resnet50()
# model.conv1.weight
# Parameter containing:
# tensor([[[[-6.7435e-03, -2.3531e-02, -9.0143e-03,  ..., -2.1245e-03,
#            -1.8077e-03,  3.0338e-03],
#           [-1.2603e-02, -2.7831e-02,  2.3187e-02,  ..., -1.5793e-02,
#             1.1655e-02,  4.5889e-03],
#           [-3.7916e-02,  1.2014e-02,  1.3815e-02,  ..., -4.2651e-03,
#             1.7314e-02, -9.9998e-03],
#           ...,

init_cfg = dict(type='Pretrained',
                checkpoint='torchvision://resnet50')
initialize(model, init_cfg)
# model.conv1.weight
# Parameter containing:
# tensor([[[[ 1.3335e-02,  1.4664e-02, -1.5351e-02,  ..., -4.0896e-02,
#            -4.3034e-02, -7.0755e-02],
#           [ 4.1205e-03,  5.8477e-03,  1.4948e-02,  ...,  2.2060e-03,
#            -2.0912e-02, -3.8517e-02],
#           [ 2.2331e-02,  2.3595e-02,  1.6120e-02,  ...,  1.0281e-01,
#             6.2641e-02,  5.1977e-02],
#           ...,

# 使用关键字'prefix' 用预训练模型的特定部分来初始化子模块权重
model = models.resnet50()
url = 'http://download.openmmlab.com/mmdetection/v2.0/retinanet/'\
      'retinanet_r50_fpn_1x_coco/'\
      'retinanet_r50_fpn_1x_coco_20200130-c2398f9e.pth'
init_cfg = dict(type='Pretrained',
```

(下页继续)

(续上页)

```
                checkpoint=url, prefix='backbone.')
initialize(model, init_cfg)
```

4. 初始化继承自 BaseModule、Sequential、ModuleList、ModuleDict 的模型

   BaseModule 继承自 torch.nn.Module，它们之间唯一的不同是 BaseModule 实现了 init_weight

   Sequential 继承自 BaseModule 和 torch.nn.Sequential

   ModuleList 继承自 BaseModule 和 torch.nn.ModuleList

   ModuleDict 继承自 BaseModule 和 torch.nn.ModuleDict

```
import torch.nn as nn
from mmcv.runner import BaseModule, Sequential, ModuleList, ModuleDict

class FooConv1d(BaseModule):

    def __init__(self, init_cfg=None):
        super().__init__(init_cfg)
        self.conv1d = nn.Conv1d(4, 1, 4)

    def forward(self, x):
        return self.conv1d(x)

class FooConv2d(BaseModule):

    def __init__(self, init_cfg=None):
        super().__init__(init_cfg)
        self.conv2d = nn.Conv2d(3, 1, 3)

    def forward(self, x):
        return self.conv2d(x)

# BaseModule
init_cfg = dict(type='Constant', layer='Conv1d', val=0., bias=1.)
model = FooConv1d(init_cfg)
model.init_weights()
# model.conv1d.weight
# Parameter containing:
# tensor([[[0., 0., 0., 0.],
#          [0., 0., 0., 0.],
#          [0., 0., 0., 0.],
#          [0., 0., 0., 0.]]], requires_grad=True)
```

(下页继续)

(续上页)

```
# Sequential
init_cfg1 = dict(type='Constant', layer='Conv1d', val=0., bias=1.)
init_cfg2 = dict(type='Constant', layer='Conv2d', val=2., bias=3.)
model1 = FooConv1d(init_cfg1)
model2 = FooConv2d(init_cfg2)
seq_model = Sequential(model1, model2)
seq_model.init_weights()
# seq_model[0].conv1d.weight
# Parameter containing:
# tensor([[[0., 0., 0., 0.],
#          [0., 0., 0., 0.],
#          [0., 0., 0., 0.],
#          [0., 0., 0., 0.]]], requires_grad=True)
# seq_model[1].conv2d.weight
# Parameter containing:
# tensor([[[[2., 2., 2.],
#           [2., 2., 2.],
#           [2., 2., 2.]],
#          ...,
#          [[2., 2., 2.],
#           [2., 2., 2.],
#           [2., 2., 2.]]]], requires_grad=True)

# inner init_cfg has higher priority
model1 = FooConv1d(init_cfg1)
model2 = FooConv2d(init_cfg2)
init_cfg = dict(type='Constant', layer=['Conv1d', 'Conv2d'], val=4., bias=5.)
seq_model = Sequential(model1, model2, init_cfg=init_cfg)
seq_model.init_weights()
# seq_model[0].conv1d.weight
# Parameter containing:
# tensor([[[0., 0., 0., 0.],
#          [0., 0., 0., 0.],
#          [0., 0., 0., 0.],
#          [0., 0., 0., 0.]]], requires_grad=True)
# seq_model[1].conv2d.weight
# Parameter containing:
# tensor([[[[2., 2., 2.],
#           [2., 2., 2.],
#           [2., 2., 2.]],
#          ...,
#          [[2., 2., 2.],
```

(下页继续)

(续上页)

```
#           [2., 2., 2.],
#           [2., 2., 2.]]], requires_grad=True)

# ModuleList
model1 = FooConv1d(init_cfg1)
model2 = FooConv2d(init_cfg2)
modellist = ModuleList([model1, model2])
modellist.init_weights()
# modellist[0].conv1d.weight
# Parameter containing:
# tensor([[[0., 0., 0., 0.],
#          [0., 0., 0., 0.],
#          [0., 0., 0., 0.],
#          [0., 0., 0., 0.]]], requires_grad=True)
# modellist[1].conv2d.weight
# Parameter containing:
# tensor([[[[2., 2., 2.],
#           [2., 2., 2.],
#           [2., 2., 2.]],
#          ...,
#          [[2., 2., 2.],
#           [2., 2., 2.],
#           [2., 2., 2.]]]], requires_grad=True)

# inner init_cfg has higher priority
model1 = FooConv1d(init_cfg1)
model2 = FooConv2d(init_cfg2)
init_cfg = dict(type='Constant', layer=['Conv1d', 'Conv2d'], val=4., bias=5.)
modellist = ModuleList([model1, model2], init_cfg=init_cfg)
modellist.init_weights()
# modellist[0].conv1d.weight
# Parameter containing:
# tensor([[[0., 0., 0., 0.],
#          [0., 0., 0., 0.],
#          [0., 0., 0., 0.],
#          [0., 0., 0., 0.]]], requires_grad=True)
# modellist[1].conv2d.weight
# Parameter containing:
# tensor([[[[2., 2., 2.],
#           [2., 2., 2.],
#           [2., 2., 2.]],
#          ...,
#          [[2., 2., 2.],
```

(下页继续)

(续上页)

```
#           [2., 2., 2.],
#           [2., 2., 2.]]], requires_grad=True)

# ModuleDict
model1 = FooConv1d(init_cfg1)
model2 = FooConv2d(init_cfg2)
modeldict = ModuleDict(dict(model1=model1, model2=model2))
modeldict.init_weights()
# modeldict['model1'].conv1d.weight
# Parameter containing:
# tensor([[[0., 0., 0., 0.],
#          [0., 0., 0., 0.],
#          [0., 0., 0., 0.],
#          [0., 0., 0., 0.]]], requires_grad=True)
# modeldict['model2'].conv2d.weight
# Parameter containing:
# tensor([[[[2., 2., 2.],
#           [2., 2., 2.],
#           [2., 2., 2.]],
#          ...,
#          [[2., 2., 2.],
#           [2., 2., 2.],
#           [2., 2., 2.]]]], requires_grad=True)

# inner init_cfg has higher priority
model1 = FooConv1d(init_cfg1)
model2 = FooConv2d(init_cfg2)
init_cfg = dict(type='Constant', layer=['Conv1d', 'Conv2d'], val=4., bias=5.)
modeldict = ModuleDict(dict(model1=model1, model2=model2), init_cfg=init_cfg)
modeldict.init_weights()
# modeldict['model1'].conv1d.weight
# Parameter containing:
# tensor([[[0., 0., 0., 0.],
#          [0., 0., 0., 0.],
#          [0., 0., 0., 0.],
#          [0., 0., 0., 0.]]], requires_grad=True)
# modeldict['model2'].conv2d.weight
# Parameter containing:
# tensor([[[[2., 2., 2.],
#           [2., 2., 2.],
#           [2., 2., 2.]],
#          ...,
#          [[2., 2., 2.],
```

(下页继续)

(续上页)

```
#           [2., 2., 2.],
#           [2., 2., 2.]]], requires_grad=True)
```

11.4 Model Zoo

除了 `torchvision` 的预训练模型，我们还提供以下 CNN 的预训练模型：

- VGG Caffe
- ResNet Caffe
- ResNeXt
- ResNet with Group Normalization
- ResNet with Group Normalization and Weight Standardization
- HRNetV2
- Res2Net
- RegNet

11.4.1 Model URLs in JSON

MMCV 中的 Model Zoo Link 由 JSON 文件管理。json 文件由模型名称及其 url 或 path 的键值对组成，一个 json 文件可能类似于：

```
{
    "model_a": "https://example.com/models/model_a_9e5bac.pth",
    "model_b": "pretrain/model_b_ab3ef2c.pth"
}
```

可以在此处找到托管在 OpenMMLab AWS 上的预训练模型的默认链接。

你可以通过将 `open-mmlab.json` 放在 `MMCV_HOME` 下来覆盖默认链接，如果在环境中找不到 `MMCV_HOME`，则默认使用 `~/.cache/mmcv`。当然你也可以使用命令 `export MMCV_HOME=/your/path` 来设置自己的路径。

外部的 json 文件将被合并为默认文件，如果相同的键出现在外部 `json` 和默认 `json` 中，则将使用外部 `json`。

11.4.2 Load Checkpoint

`mmcv.load_checkpoint()` 的参数 `filename` 支持以下类型：

- filepath: `checkpoint` 路径
- `http://xxx` and `https://xxx`: 下载 checkpoint 的链接，文件名中必需包含 `SHA256` 后缀
- `torchvision://xxx`: `torchvision.models` 中的模型链接，更多细节参考 torchvision
- `open-mmlab://xxx`: 默认和其他 json 文件中提供的模型链接或文件路径

CHAPTER 12

算子

MMCV 提供了检测、分割等任务中常用的算子

CHAPTER 13

辅助函数

13.1 进度条

如果你想跟踪函数批处理任务的进度，可以使用 `track_progress` 。它能以进度条的形式展示任务的完成情况以及剩余任务所需的时间（内部实现为 for 循环）。

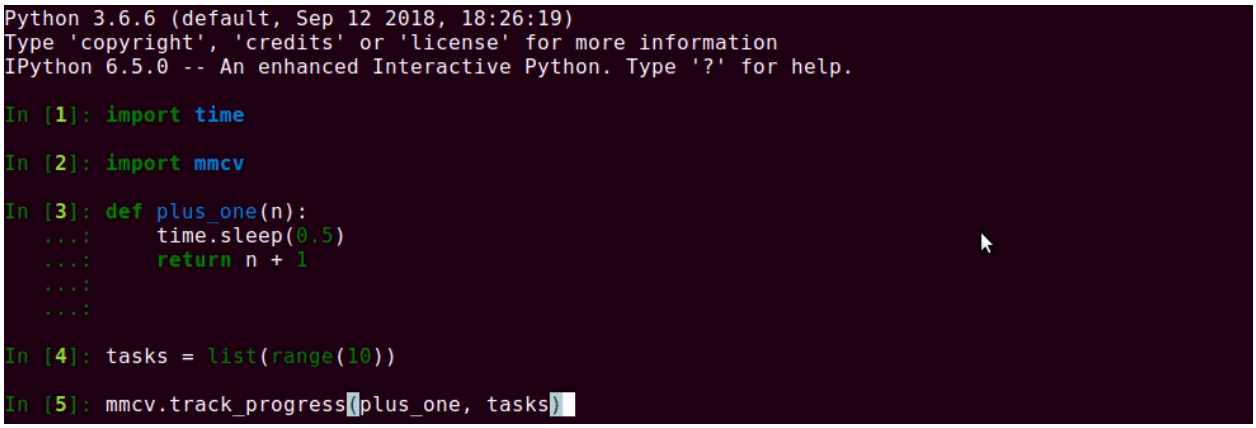
```
import mmcv

def func(item):
    # 执行相关操作
    pass

tasks = [item_1, item_2, ..., item_n]

mmcv.track_progress(func, tasks)
```

效果如下



如果你想可视化多进程任务的进度，你可以使用 `track_parallel_progress` 。

```
mmcv.track_parallel_progress(func, tasks, 8)  # 8 workers
```

如果你想要迭代或枚举数据列表并可视化进度，你可以使用 `track_iter_progress` 。

```
import mmcv

tasks = [item_1, item_2, ..., item_n]

for task in mmcv.track_iter_progress(tasks):
    # do something like print
    print(task)

for i, task in enumerate(mmcv.track_iter_progress(tasks)):
    # do something like print
    print(i)
    print(task)
```

13.2 计时器

mmcv 提供的 `Timer` 可以很方便地计算代码块的执行时间。

```
import time

with mmcv.Timer():
```

(下页继续)

(续上页)

```
    # simulate some code block
    time.sleep(1)
```

你也可以使用 since_start() 和 since_last_check() 。前者返回计时器启动后的运行时长，后者返回最近一次查看计时器后的运行时长。

```
timer = mmcv.Timer()
# code block 1 here
print(timer.since_start())
# code block 2 here
print(timer.since_last_check())
print(timer.since_start())
```

CHAPTER 14

MMCV 中 ONNX 模块简介 (实验性)

14.1 register_extra_symbolics

在将 PyTorch 模型导出成 ONNX 时，需要注册额外的符号函数

14.1.1 范例

```
import mmcv
from mmcv.onnx import register_extra_symbolics

opset_version = 11
register_extra_symbolics(opset_version)
```

14.1.2 常见问题

- 无

CHAPTER 15

MMCV 中的 ONNX Runtime 自定义算子

15.1 ONNX Runtime 介绍

ONNX Runtime 是一个跨平台的推理与训练加速器，适配许多常用的机器学习/深度神经网络框架。请访问github了解更多信息。

15.2 ONNX 介绍

ONNX 是 **Open Neural Network Exchange** 的缩写，是许多机器学习/深度神经网络框架使用的中间表示 *(IR)*。请访问github了解更多信息。

15.3 为什么要在 MMCV 中添加 ONNX 自定义算子？

- 为了验证 ONNX 模型在 ONNX Runtime 下的推理的正确性。
- 为了方便使用了 `mmcv.ops` 自定义算子的模型的部署工作。

15.4 MMCV 已支持的算子

15.5 如何编译 ONNX Runtime 自定义算子？

请注意我们仅在 ***onnxruntime>=1.8.1*** 的 *Linux x86-64 cpu* 平台上进行过测试

15.5.1 准备工作

- 克隆代码仓库

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

- 从 ONNX Runtime 下载 onnxruntime-linux: releases，解压缩，根据路径创建变量 ONNXRUNTIME_DIR 并把路径下的 lib 目录添加到 LD_LIBRARY_PATH，步骤如下：

```
wget https://github.com/microsoft/onnxruntime/releases/download/v1.8.1/onnxruntime-
↪linux-x64-1.8.1.tgz

tar -zxvf onnxruntime-linux-x64-1.8.1.tgz
cd onnxruntime-linux-x64-1.8.1
export ONNXRUNTIME_DIR=$(pwd)
export LD_LIBRARY_PATH=$ONNXRUNTIME_DIR/lib:$LD_LIBRARY_PATH
```

15.5.2 Linux 系统下编译

```
cd mmcv ## to MMCV root directory
MMCV_WITH_OPS=1 MMCV_WITH_ORT=1 python setup.py develop
```

15.6 如何在 python 下使用 ONNX Runtime 对导出的 ONNX 模型做编译

使用 pip 安装 ONNX Runtime

```
pip install onnxruntime==1.8.1
```

推理范例

```
import os

import numpy as np
import onnxruntime as ort
```

(下页继续)

(续上页)

```
from mmcv.ops import get_onnxruntime_op_path

ort_custom_op_path = get_onnxruntime_op_path()
assert os.path.exists(ort_custom_op_path)
session_options = ort.SessionOptions()
session_options.register_custom_ops_library(ort_custom_op_path)
## exported ONNX model with custom operators
onnx_file = 'sample.onnx'
input_data = np.random.randn(1, 3, 224, 224).astype(np.float32)
sess = ort.InferenceSession(onnx_file, session_options)
onnx_results = sess.run(None, {'input' : input_data})
```

15.7 如何为 MMCV 添加 ONNX Runtime 的自定义算子

15.7.1 开发前提醒

- 该算子的 ONNX Runtime 实现尚未在 MMCV 中支持已实现算子列表。
- 确保该自定义算子可以被 ONNX 导出。

15.7.2 添加方法

以 `soft_nms` 为例：

1. 在 ONNX Runtime 头文件目录 `mmcv/ops/csrc/onnxruntime/`下添加头文件 `soft_nms.h`
2. 在 ONNX Runtime 源码目录 `mmcv/ops/csrc/onnxruntime/cpu/`下添加算子实现 `soft_nms.cpp`
3. 在 onnxruntime_register.cpp 中注册实现的算子 `soft_nms`

```
#include "soft_nms.h"

SoftNmsOp c_SoftNmsOp;

if (auto status = ortApi->CustomOpDomain_Add(domain, &c_SoftNmsOp)) {
return status;
}
```

4. 在 `tests/test_ops/test_onnx.py` 添加单元测试，可以参考 here。

最后，欢迎为 MMCV 添加 ONNX Runtime 自定义算子:nerd_face:

15.8 已知问题

- "RuntimeError: tuple appears in op that does not forward tuples, unsupported kind: `prim::PythonOp`."

  1. 请注意 `cummax` 和 `cummin` 算子是在 torch >= 1.5.0 被添加的。但他们需要在 torch version >= 1.7.0 才能正确导出。否则会在导出时发生上面的错误。
  2. 解决方法：升级 PyTorch 到 1.7.0 以上版本

15.9 引用

- How to export Pytorch model with custom op to ONNX and run it in ONNX Runtime
- How to add a custom operator/kernel in ONNX Runtime

CHAPTER 16

ONNX Runtime 自定义算子

- *ONNX Runtime* 自定义算子
  - *SoftNMS*
    - 描述
    - 模型参数
    - 输入
    - 输出
    - 类型约束
  - *RoIAlign*
    - 描述
    - 模型参数
    - 输入
    - 输出
    - 类型约束
  - *NMS*
    - 描述
    - 模型参数
    - 输入

    * 输出
    * 类型约束
- *grid_sampler*
    * 描述
    * 模型参数
    * 输入
    * 输出
    * 类型约束
- *CornerPool*
    * 描述
    * 模型参数
    * 输入
    * 输出
    * 类型约束
- *cummax*
    * 描述
    * 模型参数
    * 输入
    * 输出
    * 类型约束
- *cummin*
    * 描述
    * 模型参数
    * 输入
    * 输出
    * 类型约束
- *MMCVModulatedDeformConv2d*
    * 描述
    * 模型参数
    * 输入

* 输出

* 类型约束

16.1 SoftNMS

16.1.1 描述

根据 `scores` 计算 `boxes` 的 soft NMS。请阅读Soft-NMS –Improving Object Detection With One Line of Code了解细节。

16.1.2 模型参数

16.1.3 输入

16.1.4 输出

16.1.5 类型约束

- T:tensor(float32)

16.2 RoIAlign

16.2.1 描述

在特征图上计算 RoIAlign，通常在双阶段目标检测模型的 bbox_head 中使用

16.2.2 模型参数

16.2.3 输入

16.2.4 输出

16.2.5 类型约束

- T:tensor(float32)

16.3 NMS

16.3.1 描述

根据 IoU 阈值对候选框进行非极大值抑制。

16.3.2 模型参数

16.3.3 输入

16.3.4 输出

16.3.5 类型约束

- T:tensor(float32)

16.4 grid_sampler

16.4.1 描述

根据 `grid` 的像素位置对 `input` 进行网格采样。

16.4.2 模型参数

16.4.3 输入

16.4.4 输出

16.4.5 类型约束

- T:tensor(float32, Linear)

16.5 CornerPool

16.5.1 描述

对 `input` 计算 CornerPool。请阅读CornerNet –Detecting Objects as Paired Keypoints了解更多细节。

16.5.2 模型参数

16.5.3 输入

16.5.4 输出

16.5.5 类型约束

- T:tensor(float32)

16.6 cummax

16.6.1 描述

返回一个元组 (`values`, `indices`), 其中 `values` 为 `input` 第 `dim` 维的累计最大值, `indices` 为第 `dim` 维最大值位置。请阅读torch.cummax了解更多细节。

16.6.2 模型参数

16.6.3 输入

16.6.4 输出

16.6.5 类型约束

- T:tensor(float32)

16.7 cummin

16.7.1 描述

返回一个元组(`values`, `indices`), 其中 `values` 为 `input` 第 `dim` 维的累计最小值, `indices` 为第 `dim` 维最小值位置。请阅读torch.cummin了解更多细节。

16.7.2 模型参数

16.7.3 输入

16.7.4 输出

16.7.5 类型约束

- T:tensor(float32)

16.8 MMCVModulatedDeformConv2d

16.8.1 描述

在输入特征上计算 Modulated Deformable Convolution，请阅读Deformable ConvNets v2: More Deformable, Better Results了解更多细节。

16.8.2 模型参数

16.8.3 输入

16.8.4 输出

16.8.5 类型约束

- T:tensor(float32, Linear)

CHAPTER 17

MMCV 中的 TensorRT 自定义算子 (实验性)

- *MMCV* 中的 *TensorRT* 自定义算子 *(*实验性*)*
  - 介绍
  - *MMCV* 中的 *TensorRT* 插件列表
  - 如何编译 *MMCV* 中的 *TensorRT* 插件
    - 准备
    - 在 *Linux* 上编译
  - 创建 *TensorRT* 推理引擎并在 *python* 下进行推理
  - 如何在 *MMCV* 中添加新的 *TensorRT* 自定义算子
    - 主要流程
    - 注意
  - 已知问题
  - 引用

17.1 介绍

NVIDIA TensorRT 是一个为深度学习模型高性能推理准备的软件开发工具 (SDK)。它包括深度学习推理优化器和运行时，可为深度学习推理应用提供低延迟和高吞吐量。请访问developer’s website了解更多信息。为了简化 TensorRT 部署带有 MMCV 自定义算子的模型的流程，MMCV 中添加了一系列 TensorRT 插件。

17.2 MMCV 中的 TensorRT 插件列表

注意

- 以上所有算子均在 TensorRT-7.2.1.6.Ubuntu-16.04.x86_64-gnu.cuda-10.2.cudnn8.0 环境下开发。

17.3 如何编译 MMCV 中的 TensorRT 插件

17.3.1 准备

- 克隆代码仓库

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

- 安装 TensorRT

从 NVIDIA Developer Zone 下载合适的 TensorRT 版本。

比如，对安装了 cuda-10.2 的 x86-64 的 Ubuntu 16.04，下载文件为 `TensorRT-7.2.1.6.Ubuntu-16.04.x86_64-gnu.cuda-10.2.cudnn8.0.tar.gz`.

然后使用下面方式安装并配置环境

```
cd ~/Downloads
tar -xvzf TensorRT-7.2.1.6.Ubuntu-16.04.x86_64-gnu.cuda-10.2.cudnn8.0.tar.gz
export TENSORRT_DIR=`pwd`/TensorRT-7.2.1.6
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$TENSORRT_DIR/lib
```

安装 python 依赖: tensorrt, graphsurgeon, onnx-graphsurgeon

```
pip install $TENSORRT_DIR/python/tensorrt-7.2.1.6-cp37-none-linux_x86_64.whl
pip install $TENSORRT_DIR/onnx_graphsurgeon/onnx_graphsurgeon-0.2.6-py2.py3-none-any.
↪whl
pip install $TENSORRT_DIR/graphsurgeon/graphsurgeon-0.4.5-py2.py3-none-any.whl
```

想了解更多通过 tar 包安装 TensorRT，请访问Nvidia’ website.

- 安装 cuDNN

参考Nvidia’ website安装 cuDNN 8。

17.3.2 在 Linux 上编译

```
cd mmcv ## to MMCV root directory
MMCV_WITH_OPS=1 MMCV_WITH_TRT=1 pip install -e .
```

17.4 创建 TensorRT 推理引擎并在 python 下进行推理

范例如下：

```
import torch
import onnx

from mmcv.tensorrt import (TRTWrapper, onnx2trt, save_trt_engine,
                                    is_tensorrt_plugin_loaded)

assert is_tensorrt_plugin_loaded(), 'Requires to complie TensorRT plugins in mmcv'

onnx_file = 'sample.onnx'
trt_file = 'sample.trt'
onnx_model = onnx.load(onnx_file)

## Model input
inputs = torch.rand(1, 3, 224, 224).cuda()
## Model input shape info
opt_shape_dict = {
    'input': [list(inputs.shape),
              list(inputs.shape),
              list(inputs.shape)]
}

## Create TensorRT engine
max_workspace_size = 1 << 30
trt_engine = onnx2trt(
    onnx_model,
    opt_shape_dict,
    max_workspace_size=max_workspace_size)

## Save TensorRT engine
save_trt_engine(trt_engine, trt_file)
```

(下页继续)

(续上页)

```
## Run inference with TensorRT
trt_model = TRTWrapper(trt_file, ['input'], ['output'])

with torch.no_grad():
    trt_outputs = trt_model({'input': inputs})
    output = trt_outputs['output']
```

17.5 如何在 MMCV 中添加新的 TensorRT 自定义算子

17.5.1 主要流程

下面是主要的步骤:

1. 添加 c++ 头文件
2. 添加 c++ 源文件
3. 添加 cuda kernel 文件
4. 在 `trt_plugin.cpp` 中注册插件
5. 在 `tests/test_ops/test_tensorrt.py` 中添加单元测试

以 RoIAlign 算子插件 `roi_align` 举例。

1. 在 TensorRT 包含目录 `mmcv/ops/csrc/tensorrt/`中添加头文件 `trt_roi_align.hpp`
2. 在 TensorRT 源码目录 `mmcv/ops/csrc/tensorrt/plugins/`中添加头文件 `trt_roi_align.cpp`
3. 在 TensorRT 源码目录 `mmcv/ops/csrc/tensorrt/plugins/`中添加 cuda kernel 文件 `trt_roi_align_kernel.cu`
4. 在trt_plugin.cpp中注册 `roi_align` 插件

```
#include "trt_plugin.hpp"

#include "trt_roi_align.hpp"

REGISTER_TENSORRT_PLUGIN(RoIAlignPluginDynamicCreator);

extern "C" {
bool initLibMMCVInferPlugins() { return true; }
}  // extern "C"
```

5. 在 `tests/test_ops/test_tensorrt.py` 中添加单元测试

17.5.2 注意

- 部分 MMCV 中的自定义算子存在对应的 cuda 实现，在进行 TensorRT 插件开发的时候可以参考。

17.6 已知问题

- 无

17.7 引用

- Developer guide of Nvidia TensorRT
- TensorRT Open Source Software
- onnx-tensorrt
- TensorRT python API
- TensorRT c++ plugin API

CHAPTER 18

TensorRT 自定义算子

- *TensorRT 自定义算子*
  - *MMCVRoIAlign*
    - 描述
    - 模型参数
    - 输入
    - 输出
    - 类型约束
  - *ScatterND*
    - 描述
    - 模型参数
    - 输入
    - 输出
    - 类型约束
  - *NonMaxSuppression*
    - 描述
    - 模型参数
    - 输入

    * 输出
    * 类型约束
- *MMCVDeformConv2d*
    * 描述
    * 模型参数
    * 输入
    * 输出
    * 类型约束
- *grid_sampler*
    * 描述
    * 模型参数
    * 输入
    * 输出
    * 类型约束
- *cummax*
    * 描述
    * 模型参数
    * 输入
    * 输出
    * 类型约束
- *cummin*
    * 描述
    * 模型参数
    * 输入
    * 输出
    * 类型约束
- *MMCVInstanceNormalization*
    * 描述
    * 模型参数
    * 输入

    * 输出
    * 类型约束
  - *MMCVModulatedDeformConv2d*
    * 描述
    * 模型参数
    * 输入
    * 输出
    * 类型约束

18.1 MMCVRoIAlign

18.1.1 描述

在特征图上计算 RoIAlign，在多数双阶段目标检测模型的 bbox_head 中使用

18.1.2 模型参数

18.1.3 输入

18.1.4 输出

- T:tensor(float32, Linear)

18.2 ScatterND

18.2.1 描述

ScatterND 接收三个输入，分别为秩为 r >= 1 的 `data`，秩为 q >= 1 的 `indices` 以及秩为 q + r - indices.shape[-1] -1 的 `update`。输出的计算方式为：首先创建一个 `data` 的拷贝，然后根据 `indces` 的值使用 `update` 对拷贝的 `data` 进行更新。注意 `indices` 中不应该存在相同的条目，也就是说对同一个位置进行一次以上的更新是不允许的。

输出的计算方式可以参考如下代码：

```
output = np.copy(data)
update_indices = indices.shape[:-1]
```

(下页继续)

(续上页)

```
for idx in np.ndindex(update_indices):
    output[indices[idx]] = updates[idx]
```

18.2.2 模型参数

无

18.2.3 输入

18.2.4 输出

18.2.5 类型约束

- T:tensor(float32, Linear), tensor(int32, Linear)

18.3 NonMaxSuppression

18.3.1 描述

根据 IoU 阈值对候选框进行非极大值抑制。

18.3.2 模型参数

18.3.3 输入

18.3.4 输出

18.3.5 类型约束

- T:tensor(float32, Linear)

18.4 MMCVDeformConv2d

18.4.1 描述

在输入特征上计算 Deformable Convolution，请阅读Deformable Convolutional Network了解更多细节。

18.4.2 模型参数

18.4.3 输入

18.4.4 输出

18.4.5 类型约束

- T:tensor(float32, Linear)

18.5 grid_sampler

18.5.1 描述

根据 `grid` 的像素位置对 `input` 进行网格采样。

18.5.2 模型参数

18.5.3 输入

18.5.4 输出

18.5.5 类型约束

- T:tensor(float32, Linear)

18.6 cummax

18.6.1 描述

返回一个元组 (values, indices), 其中 values 为 input 第 dim 维的累计最大值, indices 为第 dim 维最大值位置。请阅读torch.cummax了解更多细节。

18.6.2 模型参数

18.6.3 输入

18.6.4 输出

18.6.5 类型约束

- T:tensor(float32, Linear)

18.7 cummin

18.7.1 描述

返回一个元组 (values, indices), 其中 values 为 input 第 dim 维的累计最小值, indices 为第 dim 维最小值位置。请阅读torch.cummin了解更多细节。

18.7.2 模型参数

18.7.3 输入

18.7.4 输出

18.7.5 类型约束

- T:tensor(float32, Linear)

18.8 MMCVInstanceNormalization

18.8.1 描述

对特征计算 instance normalization，请阅读Instance Normalization: The Missing Ingredient for Fast Stylization了解更多详细信息。

18.8.2 模型参数

18.8.3 输入

18.8.4 输出

18.8.5 类型约束

- T:tensor(float32, Linear)

18.9 MMCVModulatedDeformConv2d

18.9.1 描述

在输入特征上计算 Modulated Deformable Convolution，请阅读Deformable ConvNets v2: More Deformable, Better Results了解更多细节。

18.9.2 模型参数

18.9.3 输入

18.9.4 输出

18.9.5 类型约束

- T:tensor(float32, Linear)

CHAPTER 19

English

CHAPTER 20

简体中文

CHAPTER 21

v1.3.18

部分自定义算子对于不同的设备有不同实现，为此添加的大量宏命令与类型检查使得代码变得难以维护。例如：

```
  if (input.device().is_cuda()) {
#ifdef MMCV_WITH_CUDA
    CHECK_CUDA_INPUT(input);
    CHECK_CUDA_INPUT(rois);
    CHECK_CUDA_INPUT(output);
    CHECK_CUDA_INPUT(argmax_y);
    CHECK_CUDA_INPUT(argmax_x);

    roi_align_forward_cuda(input, rois, output, argmax_y, argmax_x,
                           aligned_height, aligned_width, spatial_scale,
                           sampling_ratio, pool_mode, aligned);
#else
    AT_ERROR("RoIAlign is not compiled with GPU support");
#endif
  } else {
    CHECK_CPU_INPUT(input);
    CHECK_CPU_INPUT(rois);
    CHECK_CPU_INPUT(output);
    CHECK_CPU_INPUT(argmax_y);
    CHECK_CPU_INPUT(argmax_x);
    roi_align_forward_cpu(input, rois, output, argmax_y, argmax_x,
```

(下页继续)

(续上页)

```
                          aligned_height, aligned_width, spatial_scale,
                          sampling_ratio, pool_mode, aligned);
  }
```

为此我们设计了注册与分发的机制以更好的管理这些算子实现。

```
void ROIAlignForwardCUDAKernelLauncher(Tensor input, Tensor rois, Tensor output,
                                       Tensor argmax_y, Tensor argmax_x,
                                       int aligned_height, int aligned_width,
                                       float spatial_scale, int sampling_ratio,
                                       int pool_mode, bool aligned);

void roi_align_forward_cuda(Tensor input, Tensor rois, Tensor output,
                            Tensor argmax_y, Tensor argmax_x,
                            int aligned_height, int aligned_width,
                            float spatial_scale, int sampling_ratio,
                            int pool_mode, bool aligned) {
  ROIAlignForwardCUDAKernelLauncher(
      input, rois, output, argmax_y, argmax_x, aligned_height, aligned_width,
      spatial_scale, sampling_ratio, pool_mode, aligned);
}

// 注册算子的 cuda 实现
void roi_align_forward_impl(Tensor input, Tensor rois, Tensor output,
                            Tensor argmax_y, Tensor argmax_x,
                            int aligned_height, int aligned_width,
                            float spatial_scale, int sampling_ratio,
                            int pool_mode, bool aligned);
REGISTER_DEVICE_IMPL(roi_align_forward_impl, CUDA, roi_align_forward_cuda);

// roi_align.cpp
// 使用 dispatcher 根据参数中的 Tensor device 类型对实现进行分发
void roi_align_forward_impl(Tensor input, Tensor rois, Tensor output,
                            Tensor argmax_y, Tensor argmax_x,
                            int aligned_height, int aligned_width,
                            float spatial_scale, int sampling_ratio,
                            int pool_mode, bool aligned) {
  DISPATCH_DEVICE_IMPL(roi_align_forward_impl, input, rois, output, argmax_y,
                       argmax_x, aligned_height, aligned_width, spatial_scale,
                       sampling_ratio, pool_mode, aligned);
}
```

CHAPTER 22

v1.3.11

为了灵活地支持更多的后端和硬件，例如 NVIDIA GPUs 、AMD GPUs，我们重构了 mmcv/ops/csrc 目录。注意，这次重构不会影响 API 的使用。更多相关信息，请参考 PR1206。

原始的目录结构如下所示

```
.
├── common_cuda_helper.hpp
├── ops_cuda_kernel.cuh
├── pytorch_cpp_helper.hpp
├── pytorch_cuda_helper.hpp
├── parrots_cpp_helper.hpp
├── parrots_cuda_helper.hpp
├── parrots_cudawarpfunction.cuh
├── onnxruntime
│   ├── onnxruntime_register.h
│   ├── onnxruntime_session_options_config_keys.h
│   ├── ort_mmcv_utils.h
│   ├── ...
│   ├── onnx_ops.h
│   └── cpu
│       ├── onnxruntime_register.cpp
│       ├── ...
│       └── onnx_ops_impl.cpp
├── parrots
│   ├── ...
```

(下页继续)

(续上页)

```
|   ├── ops.cpp
|   ├── ops_cuda.cu
|   ├── ops_parrots.cpp
|   └── ops_pytorch.h
├── pytorch
|   ├── ...
|   ├── ops.cpp
|   ├── ops_cuda.cu
|   ├── pybind.cpp
└── tensorrt
    ├── trt_cuda_helper.cuh
    ├── trt_plugin_helper.hpp
    ├── trt_plugin.hpp
    ├── trt_serialize.hpp
    ├── ...
    ├── trt_ops.hpp
    └── plugins
        ├── trt_cuda_helper.cu
        ├── trt_plugin.cpp
        ├── ...
        ├── trt_ops.cpp
        └── trt_ops_kernel.cu
```

重构之后，它的结构如下所示

```
.
├── common
|   ├── box_iou_rotated_utils.hpp
|   ├── parrots_cpp_helper.hpp
|   ├── parrots_cuda_helper.hpp
|   ├── pytorch_cpp_helper.hpp
|   ├── pytorch_cuda_helper.hpp
|   └── cuda
|       ├── common_cuda_helper.hpp
|       ├── parrots_cudawarpfunction.cuh
|       ├── ...
|       └── ops_cuda_kernel.cuh
├── onnxruntime
|   ├── onnxruntime_register.h
|   ├── onnxruntime_session_options_config_keys.h
|   ├── ort_mmcv_utils.h
|   ├── ...
|   ├── onnx_ops.h
|   └── cpu
```

(下页继续)

(续上页)

```
|       ├── onnxruntime_register.cpp
|       ├── ...
|       └── onnx_ops_impl.cpp
├── parrots
|   ├── ...
|   ├── ops.cpp
|   ├── ops_parrots.cpp
|   └── ops_pytorch.h
├── pytorch
|   ├── info.cpp
|   ├── pybind.cpp
|   ├── ...
|   ├── ops.cpp
|   └── cuda
|       ├── ...
|       └── ops_cuda.cu
└── tensorrt
    ├── trt_cuda_helper.cuh
    ├── trt_plugin_helper.hpp
    ├── trt_plugin.hpp
    ├── trt_serialize.hpp
    ├── ...
    ├── trt_ops.hpp
    └── plugins
        ├── trt_cuda_helper.cu
        ├── trt_plugin.cpp
        ├── ...
        ├── trt_ops.cpp
        └── trt_ops_kernel.cu
```

CHAPTER 23

常见问题

在这里我们列出了用户经常遇到的问题以及对应的解决方法。如果您遇到了其他常见的问题，并且知道可以帮到大家的解决办法，欢迎随时丰富这个列表。

23.1 安装问题

- KeyError: “xxx: ‘yyy is not in the zzz registry’”

  只有模块所在的文件被导入时，注册机制才会被触发，所以您需要在某处导入该文件，更多详情请查看 KeyError: “MaskRCNN: ‘RefineRoIHead is not in the models registry’” 。

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

  1. 使用 `pip uninstall mmcv` 卸载您环境中的 mmcv
  2. 参考 installation instruction 或者 Build MMCV from source 安装 mmcv-full

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

  1. 检查 GPU 的 CUDA 计算能力
  2. 运行 `python mmdet/utils/collect_env.py` 来检查 PyTorch、torchvision 和 MMCV 是否是针对正确的 GPU 架构构建的，您可能需要去设置 `TORCH_CUDA_ARCH_LIST` 来重新安装 MMCV。兼容性问题可能会出现在使用旧版的 GPUs，如：colab 上的 Tesla K80 (3.7)
  3. 检查运行环境是否和 mmcv/mmdet 编译时的环境相同。例如，您可能使用 CUDA 10.0 编译 mmcv，但在 CUDA 9.0 的环境中运行它

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

1. 如果符号和 CUDA/C++ 相关（例如：libcudart.so 或者 GLIBCXX），请检查 CUDA/GCC 运行时的版本是否和编译 mmcv 的一致
2. 如果符号和 PyTorch 相关（例如：符号包含 caffe、aten 和 TH），请检查 PyTorch 运行时的版本是否和编译 mmcv 的一致
3. 运行 `python mmdet/utils/collect_env.py` 以检查 PyTorch、torchvision 和 MMCV 构建和运行的环境是否相同

- "RuntimeError: CUDA error: invalid configuration argument"

  这个错误可能是由于您的 GPU 性能不佳造成的。尝试降低 THREADS_PER_BLOCK 的值并重新编译 mmcv。

- "RuntimeError: nms is not compiled with GPU support"

  这个错误是由于您的 CUDA 环境没有正确安装。您可以尝试重新安装您的 CUDA 环境，然后删除 mmcv/build 文件夹并重新编译 mmcv。

- "Segmentation fault"

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

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

  3. 如果 `torch` 安装成功，那么检查 MMCV 是否安装成功。输入以下命令，如果没有报错说明 mmcv-full 安装成。

     ```
     python -c 'import mmcv; import mmcv.ops'
     ```

  4. 如果 MMCV 与 PyTorch 都安装成功了，则可以使用 `ipdb` 设置断点或者使用 `print` 函数，分析是哪一部分的代码导致了 `segmentation fault`

- "libtorch_cuda_cu.so: cannot open shared object file"

  `mmcv-full` 依赖 `libtorch_cuda_cu.so` 文件，但程序运行时没能找到该文件。我们可以检查该文件是否存在 `~/miniconda3/envs/{environment-name}/lib/python3.7/site-packages/torch/lib` 也可以尝试重装 PyTorch。

- "fatal error C1189: #error: –unsupported Microsoft Visual Studio version!"

  如果您在 Windows 上编译 mmcv-full 并且 CUDA 的版本是 9.2，您很可能会遇到这个问题 `"C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.2\include\crt/host_config.h(133): fatal error C1189: #error: -- unsupported Microsoft Visual Studio version! Only the versions 2012, 2013, 2015 and 2017 are supported!"`，您可以尝试使用低版本的 Microsoft Visual Studio，例如 vs2017。

- “error: member “torch::jit::detail::ModulePolicy::all_slots” may not be initialized”

  如果您在 Windows 上编译 mmcv-full 并且 PyTorch 的版本是 1.5.0，您很可能会遇到这个问题 `- torch/csrc/jit/api/module.h(474): error: member "torch::jit::detail::ModulePolicy::all_slots" may not be initialized`。 解决这个问题的方法是将 `torch/csrc/jit/api/module.h` 文件中所有 `static constexpr bool all_slots = false;` 替换为 `static bool all_slots = false;`。更多细节可以查看 member “torch::jit::detail::AttributePolicy::all_slots” may not be initialized。

- “error: a member with an in-class initializer must be const”

  如果您在 Windows 上编译 mmcv-full 并且 PyTorch 的版本是 1.6.0，您很可能会遇到这个问题 `"- torch/include\torch/csrc/jit/api/module.h(483): error: a member with an in-class initializer must be const"`. 解决这个问题的方法是将 `torch/include\torch/csrc/jit/api/module.h` 文件中的所有 `CONSTEXPR_EXCEPT_WIN_CUDA` 替换为 `const`。更多细节可以查看 Ninja: build stopped: subcommand failed。

- “error: member “torch::jit::ProfileOptionalOp::Kind” may not be initialized”

  如果您在 Windows 上编译 mmcv-full 并且 PyTorch 的版本是 1.7.0，您很可能会遇到这个问题 `torch/include\torch/csrc/jit/ir/ir.h(1347): error: member "torch::jit::ProfileOptionalOp::Kind" may not be initialized`. 解决这个问题的方法是修改 PyTorch 中的几个文件：

  - 删除 `torch/include\torch/csrc/jit/ir/ir.h` 文件中的 `static constexpr Symbol Kind = ::c10::prim::profile;` 和 `tatic constexpr Symbol Kind = ::c10::prim::profile_optional;`
  - 将 `torch\include\pybind11\cast.h` 文件中的 `explicit operator type&() { return *(this->value); }` 替换为 `explicit operator type&() { return *((type*)this->value); }`
  - 将 `torch/include\torch/csrc/jit/api/module.h` 文件中的所有 `CONSTEXPR_EXCEPT_WIN_CUDA` 替换为 `const`

  更多细节可以查看 Ensure default extra_compile_args。

- MMCV 和 MMDetection 的兼容性问题；”ConvWS is already registered in conv layer”

  请参考 installation instruction 为您的 MMDetection 版本安装正确版本的 MMCV。

23.2 使用问题

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

  1. 这个错误是因为有些参数没有参与 loss 的计算，可能是代码中存在多个分支，导致有些分支没有参与 loss 的计算。更多细节见 Expected to have finished reduction in the prior iteration before starting a new one。
  2. 你可以设置 DDP 中的 `find_unused_parameters` 为 `True`，或者手动查找哪些参数没有用到。

- “RuntimeError: Trying to backward through the graph a second time”

  不能同时设置 `GradientCumulativeOptimizerHook` 和 `OptimizerHook`，这会导致 `loss.backward()` 被调用两次，于是程序抛出 `RuntimeError`。我们只需设置其中的一个。更多细节见 Trying to backward through the graph a second time。

CHAPTER 24

贡献代码

欢迎任何类型的贡献，包括但不限于

- 修改拼写错误或代码错误
- 添加文档或将文档翻译成其他语言
- 添加新功能和新组件

24.1 工作流

l 详细工作流见拉取请求

1. 复刻并拉取最新的 OpenMMLab 算法库
2. 创建新的分支（不建议使用主分支提拉取请求）
3. 提交你的修改
4. 创建拉取请求

注解： 如果你计划添加新功能并且该功能包含比较大的改动，建议先开 issue 讨论

24.2 代码风格

24.2.1 Python

PEP8 作为 OpenMMLab 算法库首选的代码规范，我们使用以下工具检查和格式化代码

- flake8: Python 官方发布的代码规范检查工具，是多个检查工具的封装
- isort: 自动调整模块导入顺序的工具
- yapf: Google 发布的代码规范检查工具
- codespell: 检查单词拼写是否有误
- mdformat: 检查 markdown 文件的工具
- docformatter: 格式化 docstring 的工具

yapf 和 isort 的配置可以在 setup.cfg 找到

通过配置 pre-commit hook ，我们可以在提交代码时自动检查和格式化 `flake8`、`yapf`、`isort`、`trailing whitespaces`、`markdown files`，修复 `end-of-files`、`double-quoted-strings`、`python-encoding-pragma`、`mixed-line-ending`，调整 `requirments.txt` 的包顺序。pre-commit 钩子的配置可以在 .pre-commit-config 找到。

在克隆算法库后，你需要安装并初始化 pre-commit 钩子

```
pip install -U pre-commit
```

切换算法库根目录

```
pre-commit install
```

提交拉取请求前，请确保你的代码符合 yapf 的格式

24.2.2 C++ and CUDA

C++ 和 CUDA 的代码规范遵从 Google C++ Style Guide

CHAPTER 25

拉取请求

25.1 什么是拉取请求？

拉取请求 (Pull Request), GitHub 官方文档定义如下。

> 拉取请求是一种通知机制。你修改了他人的代码，将你的修改通知原来作者，希望他合并你的修改。

25.2 基本的工作流：

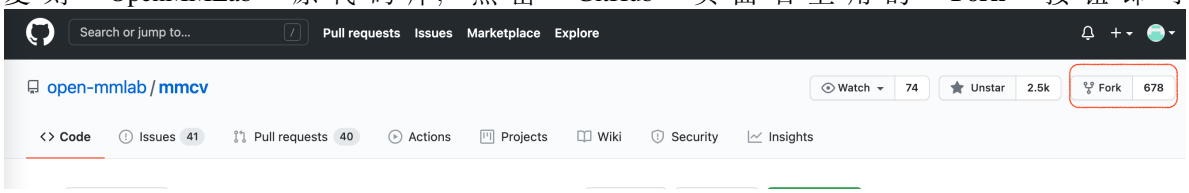
1. 获取最新的代码库
2. 从主分支创建最新的分支进行开发
3. 提交修改
4. 推送你的修改并创建一个 拉取请求
5. 讨论、审核代码
6. 将开发分支合并到主分支

25.3 具体步骤

25.3.1 1. 获取最新的代码库

- 当你第一次提 PR 时

  复刻 OpenMMLab 原代码库，点击 GitHub 页面右上角的 **Fork** 按钮即可



  克隆复刻的代码库到本地

```
git clone git@github.com:XXX/mmcv.git
```

  添加原代码库为上游代码库

```
git remote add upstream git@github.com:open-mmlab/mmcv
```

- 从第二个 PR 起

  检出本地代码库的主分支，然后从最新的原代码库的主分支拉取更新

```
git checkout master
git pull upstream master
```

25.3.2 2. 从主分支创建一个新的开发分支

```
git checkout -b branchname
```

小技巧: 为了保证提交历史清晰可读，我们强烈推荐您先检出主分支 (master)，再创建新的分支。

25.3.3 3. 提交你的修改

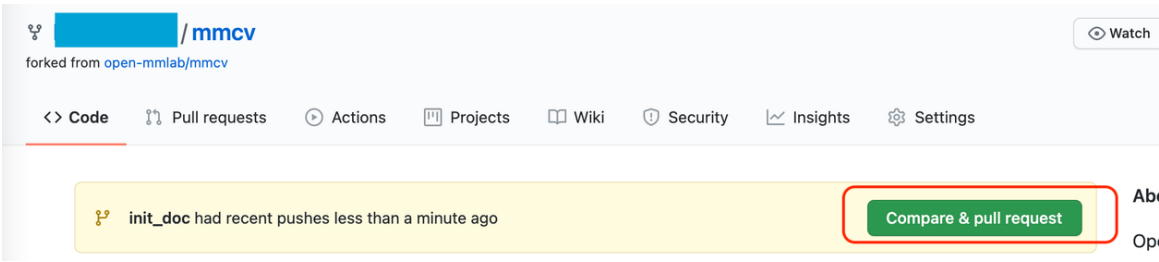
```
# coding
git add [files]
git commit -m 'messages'
```

25.3.4 4. 推送你的修改到复刻的代码库，并创建一个拉取请求

- 推送当前分支到远端复刻的代码库

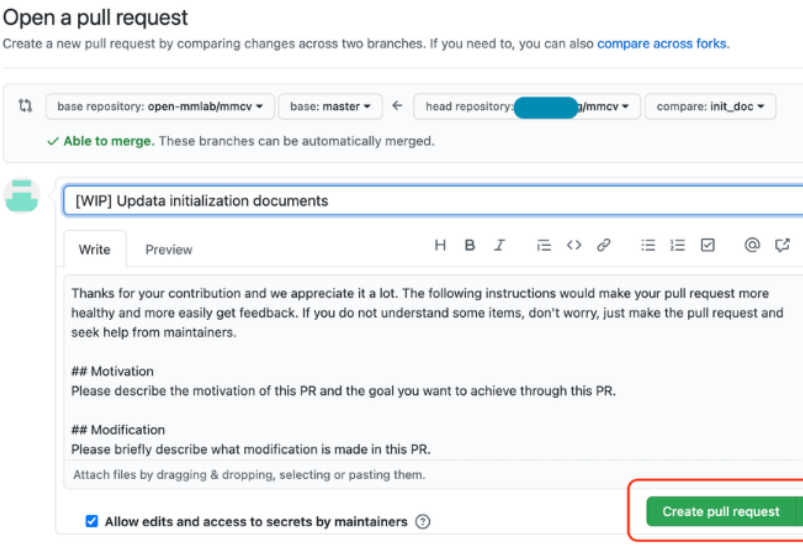
```
git push origin branchname
```

- 创建一个拉取请求



- 修改拉取请求信息模板，描述修改原因和修改内容。还可以在 PR 描述中，手动关联到相关的议题 (issue)，（更多细节，请参考官方文档）。

25.3.5 5. 讨论并评审你的代码



- 创建拉取请求时，可以关联给相关人员进行评审
- 根据评审人员的意见修改代码，并推送修改

25.3.6 6. 拉取请求合并之后删除该分支

```
git branch -d branchname # delete local branch
git push origin --delete branchname # delete remote branch
```

25.4 PR 规范

1. 使用 pre-commit hook，尽量减少代码风格相关问题
2. 一个 PR 对应一个短期分支
3. 粒度要细，一个 PR 只做一件事情，避免超大的 PR
   - Bad：实现 Faster R-CNN
   - Acceptable：给 Faster R-CNN 添加一个 box head
   - Good：给 box head 增加一个参数来支持自定义的 conv 层数
4. 每次 Commit 时需要提供清晰且有意义 commit 信息
5. 提供清晰且有意义的拉取请求描述
   - 标题写明白任务名称，一般格式:[Prefix] Short description of the pull request (Suffix)
   - prefix: 新增功能 [Feature], 修 bug [Fix], 文档相关 [Docs], 开发中 [WIP] (暂时不会被 review)
   - 描述里介绍拉取请求的主要修改内容，结果，以及对其他部分的影响, 参考拉取请求模板
   - 关联相关的议题 (issue) 和其他拉取请求

CHAPTER 26

fileio

class mmcv.fileio.**BaseStorageBackend**

Abstract class of storage backends.

All backends need to implement two apis: `get()` and `get_text()`. `get()` reads the file as a byte stream and `get_text()` reads the file as texts.

class mmcv.fileio.**FileClient**(*backend=None*, *prefix=None*, ***kwargs*)

A general file client to access files in different backends.

The client loads a file or text in a specified backend from its path and returns it as a binary or text file. There are two ways to choose a backend, the name of backend and the prefix of path. Although both of them can be used to choose a storage backend, `backend` has a higher priority that is if they are all set, the storage backend will be chosen by the backend argument. If they are all *None*, the disk backend will be chosen. Note that It can also register other backend accessor with a given name, prefixes, and backend class. In addition, We use the singleton pattern to avoid repeated object creation. If the arguments are the same, the same object will be returned.

参数

- **backend** (*str, optional*) –The storage backend type. Options are "disk", "ceph", "memcached", "lmdb", "http" and "petrel". Default: None.
- **prefix** (*str, optional*) –The prefix of the registered storage backend. Options are "s3", "http", "https". Default: None.

实际案例

```
>>> # only set backend
>>> file_client = FileClient(backend='petrel')
>>> # only set prefix
>>> file_client = FileClient(prefix='s3')
>>> # set both backend and prefix but use backend to choose client
>>> file_client = FileClient(backend='petrel', prefix='s3')
>>> # if the arguments are the same, the same object is returned
>>> file_client1 = FileClient(backend='petrel')
>>> file_client1 is file_client
True
```

client

The backend object.

Type *BaseStorageBackend*

exists (*filepath: Union[str, pathlib.Path]*) → bool

Check whether a file path exists.

参数 **filepath** (*str or Path*) –Path to be checked whether exists.

返回 Return `True` if `filepath` exists, `False` otherwise.

返回类型 bool

get (*filepath: Union[str, pathlib.Path]*) → Union[bytes, memoryview]

Read data from a given `filepath` with 'rb' mode.

注解: There are two types of return values for `get`, one is `bytes` and the other is `memoryview`. The advantage of using memoryview is that you can avoid copying, and if you want to convert it to `bytes`, you can use `.tobytes()`.

参数 **filepath** (*str or Path*) –Path to read data.

返回 Expected bytes object or a memory view of the bytes object.

返回类型 bytes | memoryview

get_local_path (*filepath: Union[str, pathlib.Path]*) → Generator[Union[str, pathlib.Path], None, None]

Download data from `filepath` and write the data to local path.

`get_local_path` is decorated by `contxtlib.contextmanager()`. It can be called with `with` statement, and when exists from the `with` statement, the temporary path will be released.

注解： If the `filepath` is a local path, just return itself.

警告： `get_local_path` is an experimental interface that may change in the future.

参数 **filepath** (*str or Path*) –Path to be read data.

实际案例

```
>>> file_client = FileClient(prefix='s3')
>>> with file_client.get_local_path('s3://bucket/abc.jpg') as path:
...     # do something here
```

生成器 *Iterable[str]* –Only yield one path.

get_text (*filepath: Union[str, pathlib.Path]*, *encoding='utf-8'*) → str

Read data from a given `filepath` with 'r' mode.

参数

- **filepath** (*str or Path*) –Path to read data.
- **encoding** (*str*) –The encoding format used to open the `filepath`. Default: 'utf-8'.

返回 Expected text reading from `filepath`.

返回类型 str

classmethod infer_client (*file_client_args: Optional[dict] = None*, *uri: Optional[Union[str, pathlib.Path]] = None*) → *mmcv.fileio.file_client.FileClient*

Infer a suitable file client based on the URI and arguments.

参数

- **file_client_args** (*dict, optional*) –Arguments to instantiate a FileClient. Default: None.
- **uri** (*str | Path, optional*) –Uri to be parsed that contains the file prefix. Default: None.

实际案例

```
>>> uri = 's3://path/of/your/file'
>>> file_client = FileClient.infer_client(uri=uri)
>>> file_client_args = {'backend': 'petrel'}
>>> file_client = FileClient.infer_client(file_client_args)
```

返回 Instantiated FileClient object.

返回类型 *FileClient*

isdir(*filepath: Union[str, pathlib.Path]*) → bool

Check whether a file path is a directory.

参数 **filepath** (*str or Path*) –Path to be checked whether it is a directory.

返回 Return `True` if `filepath` points to a directory, `False` otherwise.

返回类型 bool

isfile(*filepath: Union[str, pathlib.Path]*) → bool

Check whether a file path is a file.

参数 **filepath** (*str or Path*) –Path to be checked whether it is a file.

返回 Return `True` if `filepath` points to a file, `False` otherwise.

返回类型 bool

join_path(*filepath: Union[str, pathlib.Path]*, **filepaths: Union[str, pathlib.Path]*) → str

Concatenate all file paths.

Join one or more filepath components intelligently. The return value is the concatenation of filepath and any members of *filepaths.

参数 **filepath** (*str or Path*) –Path to be concatenated.

返回 The result of concatenation.

返回类型 str

list_dir_or_file(*dir_path: Union[str, pathlib.Path]*, *list_dir: bool = True*, *list_file: bool = True*, *suffix: Optional[Union[str, Tuple[str]]] = None*, *recursive: bool = False*) → Iterator[str]

Scan a directory to find the interested directories or files in arbitrary order.

注解: *list_dir_or_file()* returns the path relative to `dir_path`.

参数

- **dir_path** (*str | Path*) –Path of the directory.

- **list_dir** (*bool*) –List the directories. Default: True.
- **list_file** (*bool*) –List the path of files. Default: True.
- **suffix** (*str or tuple[str], optional*) –File suffix that we are interested in. Default: None.
- **recursive** (*bool*) –If set to True, recursively scan the directory. Default: False.

生成器 *Iterable[str]* –A relative path to `dir_path`.

static parse_uri_prefix(*uri: Union[str, pathlib.Path]*) → Optional[str]

Parse the prefix of a uri.

参数 **uri** (*str | Path*) –Uri to be parsed that contains the file prefix.

实际案例

```
>>> FileClient.parse_uri_prefix('s3://path/of/your/file')
's3'
```

返回 Return the prefix of uri if the uri contains '://' else `None`.

返回类型 str | None

put(*obj: bytes*, *filepath: Union[str, pathlib.Path]*) → None

Write data to a given `filepath` with 'wb' mode.

注解: `put` should create a directory if the directory of `filepath` does not exist.

参数

- **obj** (*bytes*) –Data to be written.
- **filepath** (*str or Path*) –Path to write data.

put_text(*obj: str*, *filepath: Union[str, pathlib.Path]*) → None

Write data to a given `filepath` with 'w' mode.

注解: `put_text` should create a directory if the directory of `filepath` does not exist.

参数

- **obj** (*str*) –Data to be written.
- **filepath** (*str or Path*) –Path to write data.

- **encoding** (*str, optional*) –The encoding format used to open the *filepath*. Default: 'utf-8'.

classmethod register_backend(*name*, *backend=None*, *force=False*, *prefixes=None*)

Register a backend to FileClient.

This method can be used as a normal class method or a decorator.

```
class NewBackend(BaseStorageBackend):

    def get(self, filepath):
        return filepath

    def get_text(self, filepath):
        return filepath

FileClient.register_backend('new', NewBackend)
```

or

```
@FileClient.register_backend('new')
class NewBackend(BaseStorageBackend):

    def get(self, filepath):
        return filepath

    def get_text(self, filepath):
        return filepath
```

参数

- **name** (*str*) –The name of the registered backend.
- **backend** (*class, optional*) –The backend class to be registered, which must be a subclass of *BaseStorageBackend*. When this method is used as a decorator, backend is None. Defaults to None.
- **force** (*bool, optional*) –Whether to override the backend if the name has already been registered. Defaults to False.
- **prefixes** (*str or list[str] or tuple[str], optional*) –The prefixes of the registered storage backend. Default: None. *New in version 1.3.15.*

remove(*filepath: Union[str, pathlib.Path]*) → None

Remove a file.

参数 **filepath** (*str, Path*) –Path to be removed.

mmcv.fileio.**dict_from_file**(*filename: Union[str, pathlib.Path], key_type: type = <class 'str'>, encoding: str = 'utf-8', file_client_args: Optional[Dict] = None*) → Dict

Load a text file and parse the content as a dict.

Each line of the text file will be two or more columns split by whitespaces or tabs. The first column will be parsed as dict keys, and the following columns will be parsed as dict values.

注解： In v1.3.16 and later, `dict_from_file` supports loading a text file which can be storaged in different backends and parsing the content as a dict.

参数

- **filename** (*str*) –Filename.
- **key_type** (*type*) –Type of the dict keys. str is user by default and type conversion will be performed if specified.
- **encoding** (*str*) –Encoding used to open the file. Default utf-8.
- **file_client_args** (*dict, optional*) –Arguments to instantiate a FileClient. See *mmcv.fileio.FileClient* for details. Default: None.

实际案例

```
>>> dict_from_file('/path/of/your/file')  # disk
{'key1': 'value1', 'key2': 'value2'}
>>> dict_from_file('s3://path/of/your/file')  # ceph or petrel
{'key1': 'value1', 'key2': 'value2'}
```

返回 The parsed contents.

返回类型 dict

mmcv.fileio.**dump**(*obj: Any*, *file: Optional[Union[str, pathlib.Path, TextIO, _io.StringIO, _io.BytesIO]] = None*, *file_format: Optional[str] = None*, *file_client_args: Optional[Dict] = None*, ***kwargs*)

Dump data to json/yaml/pickle strings or files.

This method provides a unified api for dumping data as strings or to files, and also supports custom arguments for each file format.

注解： In v1.3.16 and later, `dump` supports dumping data as strings or to files which is saved to different backends.

参数

- **obj** (*any*) –The python object to be dumped.
- **file** (str or Path or file-like object, optional) –If not specified, then the object is dumped to a str, otherwise to a file specified by the filename or file-like object.
- **file_format** (*str, optional*) –Same as *load()*.
- **file_client_args** (*dict, optional*) –Arguments to instantiate a FileClient. See *mmcv.fileio.FileClient* for details. Default: None.

实际案例

```
>>> dump('hello world', '/path/of/your/file')  # disk
>>> dump('hello world', 's3://path/of/your/file')  # ceph or petrel
```

返回 True for success, False otherwise.

返回类型 bool

mmcv.fileio.**list_from_file**(*filename: Union[str, pathlib.Path]*, *prefix: str = ''*, *offset: int = 0*, *max_num: int = 0*, *encoding: str = 'utf-8'*, *file_client_args: Optional[Dict] = None*) → List

Load a text file and parse the content as a list of strings.

注解: In v1.3.16 and later, `list_from_file` supports loading a text file which can be storaged in different backends and parsing the content as a list for strings.

参数

- **filename** (*str*) –Filename.
- **prefix** (*str*) –The prefix to be inserted to the beginning of each item.
- **offset** (*int*) –The offset of lines.
- **max_num** (*int*) –The maximum number of lines to be read, zeros and negatives mean no limitation.
- **encoding** (*str*) –Encoding used to open the file. Default utf-8.
- **file_client_args** (*dict, optional*) –Arguments to instantiate a FileClient. See *mmcv.fileio.FileClient* for details. Default: None.

实际案例

```
>>> list_from_file('/path/of/your/file')  # disk
['hello', 'world']
>>> list_from_file('s3://path/of/your/file')  # ceph or petrel
['hello', 'world']
```

返回 A list of strings.

返回类型 list[str]

mmcv.fileio.**load**(*file: Union[str, pathlib.Path, TextIO, _io.StringIO, _io.BytesIO]*, *file_format: Optional[str] = None*, *file_client_args: Optional[Dict] = None*, ***kwargs*)

Load data from json/yaml/pickle files.

This method provides a unified api for loading data from serialized files.

注解: In v1.3.16 and later, `load` supports loading data from serialized files those can be storaged in different backends.

参数

- **file** (str or `Path` or file-like object) –Filename or a file-like object.
- **file_format** (*str, optional*) –If not specified, the file format will be inferred from the file extension, otherwise use the specified one. Currently supported formats include "json", "yaml/yml" and "pickle/pkl".
- **file_client_args** (*dict, optional*) –Arguments to instantiate a FileClient. See *mmcv.fileio.FileClient* for details. Default: None.

实际案例

```
>>> load('/path/of/your/file')  # file is storaged in disk
>>> load('https://path/of/your/file')  # file is storaged in Internet
>>> load('s3://path/of/your/file')  # file is storaged in petrel
```

返回 The content from the file.

CHAPTER 27

image

mmcv.image.**adjust_brightness**(*img*, *factor=1.0*, *backend=None*)

Adjust image brightness.

This function controls the brightness of an image. An enhancement factor of 0.0 gives a black image. A factor of 1.0 gives the original image. This function blends the source image and the degenerated black image:

$$output = img * factor + degenerated * (1 - factor)$$

参数

- **img** (*ndarray*) –Image to be brightened.
- **factor** (*float*) –A value controls the enhancement. Factor 1.0 returns the original image, lower factors mean less color (brightness, contrast, etc), and higher values more. Default 1.
- **backend** (*str | None*) –The image processing backend type. Options are *cv2*, *pillow*, *None*. If backend is None, the global `imread_backend` specified by `mmcv.use_backend()` will be used. Defaults to None.

返回 The brightened image.

返回类型 ndarray

mmcv.image.**adjust_color**(*img*, *alpha=1*, *beta=None*, *gamma=0*, *backend=None*)

It blends the source image and its gray image:

$$output = img * alpha + gray\_img * beta + gamma$$

参数

- **img** (*ndarray*) –The input source image.
- **alpha** (*int | float*) –Weight for the source image. Default 1.
- **beta** (*int | float*) –Weight for the converted gray image. If None, it's assigned the value (1 - *alpha*).
- **gamma** (*int | float*) –Scalar added to each sum. Same as `cv2.addWeighted()`. Default 0.
- **backend** (*str | None*) –The image processing backend type. Options are *cv2*, *pillow*, *None*. If backend is None, the global `imread_backend` specified by `mmcv.use_backend()` will be used. Defaults to None.

返回 Colored image which has the same size and dtype as input.

返回类型 ndarray

`mmcv.image.`**adjust_contrast**(*img*, *factor=1.0*, *backend=None*)

Adjust image contrast.

This function controls the contrast of an image. An enhancement factor of 0.0 gives a solid grey image. A factor of 1.0 gives the original image. It blends the source image and the degenerated mean image:

$$output = img * factor + degenerated * (1 - factor)$$

参数

- **img** (*ndarray*) –Image to be contrasted. BGR order.
- **factor** (*float*) –Same as `mmcv.adjust_brightness()`.
- **backend** (*str | None*) –The image processing backend type. Options are *cv2*, *pillow*, *None*. If backend is None, the global `imread_backend` specified by `mmcv.use_backend()` will be used. Defaults to None.

返回 The contrasted image.

返回类型 ndarray

`mmcv.image.`**adjust_hue**(*img: numpy.ndarray*, *hue_factor: float*, *backend: Optional[str] = None*) → numpy.ndarray

Adjust hue of an image.

The image hue is adjusted by converting the image to HSV and cyclically shifting the intensities in the hue channel (H). The image is then converted back to original image mode.

hue_factor is the amount of shift in H channel and must be in the interval *[-0.5, 0.5]*.

Modified from https://github.com/pytorch/vision/blob/main/torchvision/ transforms/functional.py

参数

- **img** (*ndarray*) –Image to be adjusted.

- **hue_factor** (*float*) –How much to shift the hue channel. Should be in [-0.5, 0.5]. 0.5 and -0.5 give complete reversal of hue channel in HSV space in positive and negative direction respectively. 0 means no shift. Therefore, both -0.5 and 0.5 will give an image with complementary colors while 0 gives the original image.
- **backend** (*str | None*) –The image processing backend type. Options are *cv2*, *pillow*, *None*. If backend is None, the global `imread_backend` specified by `mmcv.use_backend()` will be used. Defaults to None.

返回 Hue adjusted image.

返回类型 ndarray

`mmcv.image.`**adjust_lighting**(*img*, *eigval*, *eigvec*, *alphastd=0.1*, *to_rgb=True*)

AlexNet-style PCA jitter.

This data augmentation is proposed in ImageNet Classification with Deep Convolutional Neural Networks.

参数

- **img** (*ndarray*) –Image to be adjusted lighting. BGR order.
- **eigval** (*ndarray*) –the eigenvalue of the convariance matrix of pixel values, respectively.
- **eigvec** (*ndarray*) –the eigenvector of the convariance matrix of pixel values, respectively.
- **alphastd** (*float*) –The standard deviation for distribution of alpha. Defaults to 0.1
- **to_rgb** (*bool*) –Whether to convert img to rgb.

返回 The adjusted image.

返回类型 ndarray

`mmcv.image.`**adjust_sharpness**(*img*, *factor=1.0*, *kernel=None*)

Adjust image sharpness.

This function controls the sharpness of an image. An enhancement factor of 0.0 gives a blurred image. A factor of 1.0 gives the original image. And a factor of 2.0 gives a sharpened image. It blends the source image and the degenerated mean image:

$$output = img * factor + degenerated * (1 - factor)$$

参数

- **img** (*ndarray*) –Image to be sharpened. BGR order.
- **factor** (*float*) –Same as `mmcv.adjust_brightness()`.
- **kernel** (*np.ndarray, optional*) –Filter kernel to be applied on the img to obtain the degenerated img. Defaults to None.

注解: No value sanity check is enforced on the kernel set by users. So with an inappropriate kernel, the `adjust_sharpness` may fail to perform the function its name indicates but end up performing whatever transform determined by the kernel.

返回 The sharpened image.

返回类型 ndarray

mmcv.image.**auto_contrast**(*img*, *cutoff=0*)

Auto adjust image contrast.

This function maximize (normalize) image contrast by first removing cutoff percent of the lightest and darkest pixels from the histogram and remapping the image so that the darkest pixel becomes black (0), and the lightest becomes white (255).

参数

- **img** (*ndarray*) –Image to be contrasted. BGR order.
- **cutoff** (*int | float | tuple*) –The cutoff percent of the lightest and darkest pixels to be removed. If given as tuple, it shall be (low, high). Otherwise, the single value will be used for both. Defaults to 0.

返回 The contrasted image.

返回类型 ndarray

mmcv.image.**bgr2gray**(*img: numpy.ndarray*, *keepdim: bool = False*) → numpy.ndarray

Convert a BGR image to grayscale image.

参数

- **img** (*ndarray*) –The input image.
- **keepdim** (*bool*) –If False (by default), then return the grayscale image with 2 dims, otherwise 3 dims.

返回 The converted grayscale image.

返回类型 ndarray

mmcv.image.**bgr2hls**(*img: numpy.ndarray*) → numpy.ndarray

Convert a BGR image to HLS image.

参数 **img** (*ndarray or str*) –The input image.

返回 The converted HLS image.

返回类型 ndarray

mmcv.image.**bgr2hsv**(*img: numpy.ndarray*) → numpy.ndarray

Convert a BGR image to HSV image.

参数 **img** (*ndarray or str*) –The input image.

返回 The converted HSV image.

返回类型 ndarray

mmcv.image.**bgr2rgb**(*img: numpy.ndarray*) → numpy.ndarray

Convert a BGR image to RGB image.

参数 **img** (*ndarray or str*) –The input image.

返回 The converted RGB image.

返回类型 ndarray

mmcv.image.**bgr2ycbcr**(*img: numpy.ndarray*, *y_only: bool = False*) → numpy.ndarray

Convert a BGR image to YCbCr image.

The bgr version of rgb2ycbcr. It implements the ITU-R BT.601 conversion for standard-definition television. See more details in https://en.wikipedia.org/wiki/YCbCr#ITU-R_BT.601_conversion.

It differs from a similar function in cv2.cvtColor: *BGR <-> YCrCb*. In OpenCV, it implements a JPEG conversion. See more details in https://en.wikipedia.org/wiki/YCbCr#JPEG_conversion.

参数

- **img** (*ndarray*) –The input image. It accepts: 1. np.uint8 type with range [0, 255]; 2. np.float32 type with range [0, 1].
- **y_only** (*bool*) –Whether to only return Y channel. Default: False.

返回 The converted YCbCr image. The output image has the same type and range as input image.

返回类型 ndarray

mmcv.image.**clahe**(*img*, *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

- **img** (*ndarray*) –Image to be processed.
- **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).

返回 The processed image.

返回类型 ndarray

mmcv.image.**cutout**(*img: numpy.ndarray*, *shape: Union[int, Tuple[int, int]]*, *pad_val: Union[int, float, tuple] = 0*) → numpy.ndarray

Randomly cut out a rectangle from the original img.

参数

- **img** (*ndarray*) –Image to be cutout.
- **shape** (*int | tuple[int]*) –Expected cutout shape (h, w). If given as a int, the value will be used for both h and w.
- **pad_val** (*int | float | tuple[int | float]*) –Values to be filled in the cut area. Defaults to 0.

返回 The cutout image.

返回类型 ndarray

mmcv.image.**gray2bgr**(*img: numpy.ndarray*) → numpy.ndarray

Convert a grayscale image to BGR image.

参数 **img** (*ndarray*) –The input image.

返回 The converted BGR image.

返回类型 ndarray

mmcv.image.**gray2rgb**(*img: numpy.ndarray*) → numpy.ndarray

Convert a grayscale image to RGB image.

参数 **img** (*ndarray*) –The input image.

返回 The converted RGB image.

返回类型 ndarray

mmcv.image.**hls2bgr**(*img: numpy.ndarray*) → numpy.ndarray

Convert a HLS image to BGR image.

参数 **img** (*ndarray or str*) –The input image.

返回 The converted BGR image.

返回类型 ndarray

mmcv.image.**hsv2bgr**(*img: numpy.ndarray*) → numpy.ndarray

Convert a HSV image to BGR image.

参数 **img** (*ndarray or str*) –The input image.

返回 The converted BGR image.

返回类型 ndarray

mmcv.image.**imconvert**(*img: numpy.ndarray*, *src: str*, *dst: str*) → numpy.ndarray

Convert an image from the src colorspace to dst colorspace.

参数

- **img** (*ndarray*) –The input image.
- **src** (*str*) –The source colorspace, e.g., 'rgb', 'hsv'.
- **dst** (*str*) –The destination colorspace, e.g., 'rgb', 'hsv'.

返回 The converted image.

返回类型 ndarray

mmcv.image.**imcrop**(*img: numpy.ndarray*, *bboxes: numpy.ndarray*, *scale: float = 1.0*, *pad_fill: Optional[Union[float, list]] = None*) → Union[numpy.ndarray, List[numpy.ndarray]]

Crop image patches.

3 steps: scale the bboxes -> clip bboxes -> crop and pad.

参数

- **img** (*ndarray*) –Image to be cropped.
- **bboxes** (*ndarray*) –Shape (k, 4) or (4,), location of cropped bboxes.
- **scale** (*float, optional*) –Scale ratio of bboxes, the default value 1.0 means no padding.
- **pad_fill** (*Number | list[Number]*) –Value to be filled for padding. Default: None, which means no padding.

返回 The cropped image patches.

返回类型 list[ndarray] | ndarray

mmcv.image.**imequalize**(*img*)

Equalize the image histogram.

This function applies a non-linear mapping to the input image, in order to create a uniform distribution of grayscale values in the output image.

参数 **img** (*ndarray*) –Image to be equalized.

返回 The equalized image.

返回类型 ndarray

mmcv.image.**imflip**(*img: numpy.ndarray*, *direction: str = 'horizontal'*) → numpy.ndarray

Flip an image horizontally or vertically.

参数

- **img** (*ndarray*) –Image to be flipped.
- **direction** (*str*) –The flip direction, either "horizontal" or "vertical" or "diagonal".

返回 The flipped image.

返回类型 ndarray

mmcv.image.**imflip_**(*img: numpy.ndarray*, *direction: str = 'horizontal'*) → numpy.ndarray

Inplace flip an image horizontally or vertically.

参数

- **img** (*ndarray*) –Image to be flipped.
- **direction** (*str*) –The flip direction, either "horizontal" or "vertical" or "diagonal".

返回 The flipped image (inplace).

返回类型 ndarray

mmcv.image.**imfrombytes**(*content: bytes*, *flag: str = 'color'*, *channel_order: str = 'bgr'*, *backend: Optional[str] = None*) → numpy.ndarray

Read an image from bytes.

参数

- **content** (*bytes*) –Image bytes got from files or other streams.
- **flag** (*str*) –Same as *imread()*.
- **channel_order** (*str*) –The channel order of the output, candidates are 'bgr' and 'rgb'. Default to 'bgr'.
- **backend** (*str | None*) –The image decoding backend type. Options are *cv2*, *pillow*, *turbojpeg*, *tifffile*, *None*. If backend is None, the global imread_backend specified by `mmcv.use_backend()` will be used. Default: None.

返回 Loaded image array.

返回类型 ndarray

实际案例

```
>>> img_path = '/path/to/img.jpg'
>>> with open(img_path, 'rb') as f:
>>>     img_buff = f.read()
>>> img = mmcv.imfrombytes(img_buff)
>>> img = mmcv.imfrombytes(img_buff, flag='color', channel_order='rgb')
>>> img = mmcv.imfrombytes(img_buff, backend='pillow')
>>> img = mmcv.imfrombytes(img_buff, backend='cv2')
```

mmcv.image.**iminvert**(*img*)

Invert (negate) an image.

参数 **img** (*ndarray*) –Image to be inverted.

返回 The inverted image.

返回类型 ndarray

mmcv.image.**imnormalize**(*img*, *mean*, *std*, *to_rgb=True*)

Normalize an image with mean and std.

参数

- **img** (*ndarray*) –Image to be normalized.
- **mean** (*ndarray*) –The mean to be used for normalize.
- **std** (*ndarray*) –The std to be used for normalize.
- **to_rgb** (*bool*) –Whether to convert to rgb.

返回 The normalized image.

返回类型 ndarray

mmcv.image.**imnormalize_**(*img*, *mean*, *std*, *to_rgb=True*)

Inplace normalize an image with mean and std.

参数

- **img** (*ndarray*) –Image to be normalized.
- **mean** (*ndarray*) –The mean to be used for normalize.
- **std** (*ndarray*) –The std to be used for normalize.
- **to_rgb** (*bool*) –Whether to convert to rgb.

返回 The normalized image.

返回类型 ndarray

mmcv.image.**impad**(*img: numpy.ndarray*, ***, *shape: Optional[Tuple[int, int]] = None*, *padding: Optional[Union[int, tuple]] = None*, *pad_val: Union[float, List] = 0*, *padding_mode: str = 'constant'*) → numpy.ndarray

Pad the given image to a certain shape or pad on all sides with specified padding mode and padding value.

参数

- **img** (*ndarray*) –Image to be padded.
- **shape** (*tuple[int]*) –Expected padding shape (h, w). Default: None.
- **padding** (*int or tuple[int]*) –Padding on each border. If a single int is provided this is used to pad all borders. If tuple of length 2 is provided this is the padding on left/right and top/bottom respectively. If a tuple of length 4 is provided this is the padding for the left, top, right and bottom borders respectively. Default: None. Note that *shape* and *padding* can not be both set.
- **pad_val** (*Number | Sequence[Number]*) –Values to be filled in padding areas when padding_mode is 'constant'. Default: 0.
- **padding_mode** (*str*) –Type of padding. Should be: constant, edge, reflect or symmetric. Default: constant. - constant: pads with a constant value, this value is specified

  with pad_val.

  - edge: pads with the last value at the edge of the image.
  - reflect: pads with reflection of image without repeating the last value on the edge. For example, padding [1, 2, 3, 4] with 2 elements on both sides in reflect mode will result in [3, 2, 1, 2, 3, 4, 3, 2].
  - symmetric: pads with reflection of image repeating the last value on the edge. For example, padding [1, 2, 3, 4] with 2 elements on both sides in symmetric mode will result in [2, 1, 1, 2, 3, 4, 4, 3]

返回 The padded image.

返回类型 ndarray

mmcv.image.**impad_to_multiple**(*img: numpy.ndarray*, *divisor: int*, *pad_val: Union[float, List] = 0*) → numpy.ndarray

Pad an image to ensure each edge to be multiple to some number.

参数

- **img** (*ndarray*) –Image to be padded.
- **divisor** (*int*) –Padded image edges will be multiple to divisor.
- **pad_val** (*Number | Sequence[Number]*) –Same as *impad()*.

返回 The padded image.

返回类型 ndarray

mmcv.image.**imread**(*img_or_path: Union[numpy.ndarray, str, pathlib.Path]*, *flag: str = 'color'*, *channel_order: str = 'bgr'*, *backend: Optional[str] = None*, *file_client_args: Optional[dict] = None*) → numpy.ndarray

Read an image.

注解： In v1.4.1 and later, add *file_client_args* parameters.

参数

- **img_or_path** (*ndarray or str or Path*) –Either a numpy array or str or pathlib.Path. If it is a numpy array (loaded image), then it will be returned as is.
- **flag** (*str*) –Flags specifying the color type of a loaded image, candidates are *color*, *grayscale*, *unchanged*, *color_ignore_orientation* and *grayscale_ignore_orientation*. By default, *cv2* and *pillow* backend would rotate the image according to its EXIF info unless called with *unchanged* or **_ignore_orientation* flags. *turbojpeg* and *tifffile* backend always ignore image's EXIF info regardless of the flag. The *turbojpeg* backend only supports *color* and *grayscale*.
- **channel_order** (*str*) –Order of channel, candidates are *bgr* and *rgb*.
- **backend** (*str | None*) –The image decoding backend type. Options are *cv2*, *pillow*, *turbojpeg*, *tifffile*, *None*. If backend is None, the global imread_backend specified by `mmcv.use_backend()` will be used. Default: None.
- **file_client_args** (*dict | None*) –Arguments to instantiate a FileClient. See *mmcv.fileio.FileClient* for details. Default: None.

返回 Loaded image array.

返回类型 ndarray

实际案例

```
>>> import mmcv
>>> img_path = '/path/to/img.jpg'
>>> img = mmcv.imread(img_path)
>>> img = mmcv.imread(img_path, flag='color', channel_order='rgb',
...     backend='cv2')
>>> img = mmcv.imread(img_path, flag='color', channel_order='bgr',
...     backend='pillow')
>>> s3_img_path = 's3://bucket/img.jpg'
>>> # infer the file backend by the prefix s3
>>> img = mmcv.imread(s3_img_path)
```

(下页继续)

(续上页)

```
>>> # manually set the file backend petrel
>>> img = mmcv.imread(s3_img_path, file_client_args={
...     'backend': 'petrel'})
>>> http_img_path = 'http://path/to/img.jpg'
>>> img = mmcv.imread(http_img_path)
>>> img = mmcv.imread(http_img_path, file_client_args={
...     'backend': 'http'})
```

mmcv.image.**imrescale**(*img: numpy.ndarray*, *scale: Union[float, Tuple[int, int]]*, *return_scale: bool = False*, *interpolation: str = 'bilinear'*, *backend: Optional[str] = None*) → Union[numpy.ndarray, Tuple[numpy.ndarray, float]]

Resize image while keeping the aspect ratio.

参数

- **img** (*ndarray*) –The input image.
- **scale** (*float | tuple[int]*) –The scaling factor or maximum size. If it is a float number, then the image will be rescaled by this factor, else if it is a tuple of 2 integers, then the image will be rescaled as large as possible within the scale.
- **return_scale** (*bool*) –Whether to return the scaling factor besides the rescaled image.
- **interpolation** (*str*) –Same as `resize()`.
- **backend** (*str | None*) –Same as `resize()`.

返回 The rescaled image.

返回类型 ndarray

mmcv.image.**imresize**(*img: numpy.ndarray*, *size: Tuple[int, int]*, *return_scale: bool = False*, *interpolation: str = 'bilinear'*, *out: Optional[numpy.ndarray] = None*, *backend: Optional[str] = None*) → Union[Tuple[numpy.ndarray, float, float], numpy.ndarray]

Resize image to a given size.

参数

- **img** (*ndarray*) –The input image.
- **size** (*tuple[int]*) –Target size (w, h).
- **return_scale** (*bool*) –Whether to return *w_scale* and *h_scale*.
- **interpolation** (*str*) –Interpolation method, accepted values are "nearest", "bilinear", "bicubic", "area", "lanczos" for 'cv2' backend, "nearest", "bilinear" for 'pillow' backend.
- **out** (*ndarray*) –The output destination.

- **backend** (*str | None*) –The image resize backend type. Options are *cv2*, *pillow*, *None*. If backend is None, the global imread_backend specified by `mmcv.use_backend()` will be used. Default: None.

返回 (*resized_img*, *w_scale*, *h_scale*) or *resized_img*.

返回类型 tuple | ndarray

mmcv.image.**imresize_like**(*img: numpy.ndarray*, *dst_img: numpy.ndarray*, *return_scale: bool = False*, *interpolation: str = 'bilinear'*, *backend: Optional[str] = None*) → Union[Tuple[numpy.ndarray, float, float], numpy.ndarray]

Resize image to the same size of a given image.

参数

- **img** (*ndarray*) –The input image.
- **dst_img** (*ndarray*) –The target image.
- **return_scale** (*bool*) –Whether to return *w_scale* and *h_scale*.
- **interpolation** (*str*) –Same as `resize()`.
- **backend** (*str | None*) –Same as `resize()`.

返回 (*resized_img*, *w_scale*, *h_scale*) or *resized_img*.

返回类型 tuple or ndarray

mmcv.image.**imresize_to_multiple**(*img: numpy.ndarray*, *divisor: Union[int, Tuple[int, int]]*, *size: Optional[Union[int, Tuple[int, int]]] = None*, *scale_factor: Optional[Union[float, Tuple[float, float]]] = None*, *keep_ratio: bool = False*, *return_scale: bool = False*, *interpolation: str = 'bilinear'*, *out: Optional[numpy.ndarray] = None*, *backend: Optional[str] = None*) → Union[Tuple[numpy.ndarray, float, float], numpy.ndarray]

Resize image according to a given size or scale factor and then rounds up the the resized or rescaled image size to the nearest value that can be divided by the divisor.

参数

- **img** (*ndarray*) –The input image.
- **divisor** (*int | tuple*) –Resized image size will be a multiple of divisor. If divisor is a tuple, divisor should be (w_divisor, h_divisor).
- **size** (*None | int | tuple[int]*) –Target size (w, h). Default: None.
- **scale_factor** (*None | float | tuple[float]*) –Multiplier for spatial size. Should match input size if it is a tuple and the 2D style is (w_scale_factor, h_scale_factor). Default: None.
- **keep_ratio** (*bool*) –Whether to keep the aspect ratio when resizing the image. Default: False.

- **return_scale** (*bool*) –Whether to return *w_scale* and *h_scale*.
- **interpolation** (*str*) –Interpolation method, accepted values are "nearest", "bilinear", "bicubic", "area", "lanczos" for 'cv2' backend, "nearest", "bilinear" for 'pillow' backend.
- **out** (*ndarray*) –The output destination.
- **backend** (*str | None*) –The image resize backend type. Options are *cv2*, *pillow*, *None*. If backend is None, the global imread_backend specified by `mmcv.use_backend()` will be used. Default: None.

返回 (*resized_img*, *w_scale*, *h_scale*) or *resized_img*.

返回类型 tuple | ndarray

mmcv.image.**imrotate**(*img: numpy.ndarray*, *angle: float*, *center: Optional[Tuple[float, float]] = None*, *scale: float = 1.0*, *border_value: int = 0*, *interpolation: str = 'bilinear'*, *auto_bound: bool = False*, *border_mode: str = 'constant'*) → numpy.ndarray

Rotate an image.

参数

- **img** (*np.ndarray*) –Image to be rotated.
- **angle** (*float*) –Rotation angle in degrees, positive values mean clockwise rotation.
- **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.
- **scale** (*float*) –Isotropic scale factor.
- **border_value** (*int*) –Border value used in case of a constant border. Defaults to 0.
- **interpolation** (*str*) –Same as `resize()`.
- **auto_bound** (*bool*) –Whether to adjust the image size to cover the whole rotated image.
- **border_mode** (*str*) –Pixel extrapolation method. Defaults to 'constant'.

返回 The rotated image.

返回类型 np.ndarray

mmcv.image.**imshear**(*img: numpy.ndarray*, *magnitude: Union[int, float]*, *direction: str = 'horizontal'*, *border_value: Union[int, Tuple[int, int]] = 0*, *interpolation: str = 'bilinear'*) → numpy.ndarray

Shear an image.

参数

- **img** (*ndarray*) –Image to be sheared with format (h, w) or (h, w, c).
- **magnitude** (*int | float*) –The magnitude used for shear.

- **direction** (*str*) –The flip direction, either "horizontal" or "vertical".
- **border_value** (*int | tuple[int]*) –Value used in case of a constant border.
- **interpolation** (*str*) –Same as `resize()`.

返回 The sheared image.

返回类型 ndarray

mmcv.image.**imtranslate**(*img: numpy.ndarray*, *offset: Union[int, float]*, *direction: str = 'horizontal'*, *border_value: Union[int, tuple] = 0*, *interpolation: str = 'bilinear'*) → numpy.ndarray

Translate an image.

参数

- **img** (*ndarray*) –Image to be translated with format (h, w) or (h, w, c).
- **offset** (*int | float*) –The offset used for translate.
- **direction** (*str*) –The translate direction, either "horizontal" or "vertical".
- **border_value** (*int | tuple[int]*) –Value used in case of a constant border.
- **interpolation** (*str*) –Same as `resize()`.

返回 The translated image.

返回类型 ndarray

mmcv.image.**imwrite**(*img: numpy.ndarray*, *file_path: str*, *params: Optional[list] = None*, *auto_mkdir: Optional[bool] = None*, *file_client_args: Optional[dict] = None*) → bool

Write image to file.

注解: In v1.4.1 and later, add *file_client_args* parameters.

警告: The parameter *auto_mkdir* will be deprecated in the future and every file clients will make directory automatically.

参数

- **img** (*ndarray*) –Image array to be written.
- **file_path** (*str*) –Image file path.
- **params** (*None or list*) –Same as opencv *imwrite()* interface.
- **auto_mkdir** (*bool*) –If the parent folder of *file_path* does not exist, whether to create it automatically. It will be deprecated.

- **file_client_args** (*dict | None*) –Arguments to instantiate a FileClient. See *mmcv.fileio.FileClient* for details. Default: None.

返回 Successful or not.

返回类型 bool

实际案例

```
>>> # write to hard disk client
>>> ret = mmcv.imwrite(img, '/path/to/img.jpg')
>>> # infer the file backend by the prefix s3
>>> ret = mmcv.imwrite(img, 's3://bucket/img.jpg')
>>> # manually set the file backend petrel
>>> ret = mmcv.imwrite(img, 's3://bucket/img.jpg', file_client_args={
...     'backend': 'petrel'})
```

mmcv.image.**lut_transform**(*img*, *lut_table*)

Transform array by look-up table.

The function lut_transform fills the output array with values from the look-up table. Indices of the entries are taken from the input array.

参数

- **img** (*ndarray*) –Image to be transformed.
- **lut_table** (*ndarray*) –look-up table of 256 elements; in case of multi-channel input array, the table should either have a single channel (in this case the same table is used for all channels) or the same number of channels as in the input array.

返回 The transformed image.

返回类型 ndarray

mmcv.image.**posterize**(*img*, *bits*)

Posterize an image (reduce the number of bits for each color channel)

参数

- **img** (*ndarray*) –Image to be posterized.
- **bits** (*int*) –Number of bits (1 to 8) to use for posterizing.

返回 The posterized image.

返回类型 ndarray

mmcv.image.**rescale_size**(*old_size: tuple*, *scale: Union[float, int, tuple]*, *return_scale: bool = False*) → tuple

Calculate the new size to be rescaled to.

参数

- **old_size** (*tuple[int]*) –The old size (w, h) of image.
- **scale** (*float | tuple[int]*) –The scaling factor or maximum size. If it is a float number, then the image will be rescaled by this factor, else if it is a tuple of 2 integers, then the image will be rescaled as large as possible within the scale.
- **return_scale** (*bool*) –Whether to return the scaling factor besides the rescaled image size.

返回 The new rescaled image size.

返回类型 tuple[int]

mmcv.image.**rgb2bgr**(*img: numpy.ndarray*) → numpy.ndarray

Convert a RGB image to BGR image.

参数 **img** (*ndarray or str*) –The input image.

返回 The converted BGR image.

返回类型 ndarray

mmcv.image.**rgb2gray**(*img: numpy.ndarray*, *keepdim: bool = False*) → numpy.ndarray

Convert a RGB image to grayscale image.

参数

- **img** (*ndarray*) –The input image.
- **keepdim** (*bool*) –If False (by default), then return the grayscale image with 2 dims, otherwise 3 dims.

返回 The converted grayscale image.

返回类型 ndarray

mmcv.image.**rgb2ycbcr**(*img: numpy.ndarray*, *y_only: bool = False*) → numpy.ndarray

Convert a RGB image to YCbCr image.

This function produces the same results as Matlab's *rgb2ycbcr* function. It implements the ITU-R BT.601 conversion for standard-definition television. See more details in https://en.wikipedia.org/wiki/YCbCr#ITU-R_BT.601_conversion.

It differs from a similar function in cv2.cvtColor: *RGB <-> YCrCb*. In OpenCV, it implements a JPEG conversion. See more details in https://en.wikipedia.org/wiki/YCbCr#JPEG_conversion.

参数

- **img** (*ndarray*) –The input image. It accepts: 1. np.uint8 type with range [0, 255]; 2. np.float32 type with range [0, 1].
- **y_only** (*bool*) –Whether to only return Y channel. Default: False.

返回 The converted YCbCr image. The output image has the same type and range as input image.

返回类型 ndarray

mmcv.image.**solarize**(*img*, *thr=128*)

Solarize an image (invert all pixel values above a threshold)

参数

- **img** (*ndarray*) –Image to be solarized.
- **thr** (*int*) –Threshold for solarizing (0 - 255).

返回 The solarized image.

返回类型 ndarray

mmcv.image.**tensor2imgs**(*tensor*, *mean: Optional[tuple] = None*, *std: Optional[tuple] = None*, *to_rgb: bool = True*) → list

Convert tensor to 3-channel images or 1-channel gray images.

参数

- **tensor** (*torch.Tensor*) –Tensor that contains multiple images, shape (N, C, H, W). *C* can be either 3 or 1.
- **mean** (*tuple[float], optional*) –Mean of images. If None, (0, 0, 0) will be used for tensor with 3-channel, while (0,) for tensor with 1-channel. Defaults to None.
- **std** (*tuple[float], optional*) –Standard deviation of images. If None, (1, 1, 1) will be used for tensor with 3-channel, while (1,) for tensor with 1-channel. Defaults to None.
- **to_rgb** (*bool, optional*) –Whether the tensor was converted to RGB format in the first place. If so, convert it back to BGR. For the tensor with 1 channel, it must be False. Defaults to True.

返回 A list that contains multiple images.

返回类型 list[np.ndarray]

mmcv.image.**use_backend**(*backend: str*) → None

Select a backend for image decoding.

参数

- **backend** (*str*) –The image decoding backend type. Options are *cv2*,
- **pillow** –//github.com/lilohuang/PyTurboJPEG)
- **(see https** (*turbojpeg*) –//github.com/lilohuang/PyTurboJPEG)
- **tifffile. turbojpeg is faster but it only supports .jpeg** (*and*) –
- **format.** (*file*) –

mmcv.image.**ycbcr2bgr**(*img: numpy.ndarray*) → numpy.ndarray

Convert a YCbCr image to BGR image.

The bgr version of ycbcr2rgb. It implements the ITU-R BT.601 conversion for standard-definition television. See more details in https://en.wikipedia.org/wiki/YCbCr#ITU-R_BT.601_conversion.

It differs from a similar function in cv2.cvtColor: *YCrCb <-> BGR*. In OpenCV, it implements a JPEG conversion. See more details in https://en.wikipedia.org/wiki/YCbCr#JPEG_conversion.

参数 **img** (*ndarray*) –The input image. It accepts: 1. np.uint8 type with range [0, 255]; 2. np.float32 type with range [0, 1].

返回 The converted BGR image. The output image has the same type and range as input image.

返回类型 ndarray

mmcv.image.**ycbcr2rgb**(*img: numpy.ndarray*) → numpy.ndarray

Convert a YCbCr image to RGB image.

This function produces the same results as Matlab' s ycbcr2rgb function. It implements the ITU-R BT.601 conversion for standard-definition television. See more details in https://en.wikipedia.org/wiki/YCbCr#ITU-R_BT.601_conversion.

It differs from a similar function in cv2.cvtColor: *YCrCb <-> RGB*. In OpenCV, it implements a JPEG conversion. See more details in https://en.wikipedia.org/wiki/YCbCr#JPEG_conversion.

参数 **img** (*ndarray*) –The input image. It accepts: 1. np.uint8 type with range [0, 255]; 2. np.float32 type with range [0, 1].

返回 The converted RGB image. The output image has the same type and range as input image.

返回类型 ndarray

CHAPTER 28

video

class mmcv.video.**VideoReader**(*filename*, *cache_capacity=10*)

Video class with similar usage to a list object.

This video wrapper class provides convenient apis to access frames. There exists an issue of OpenCV's VideoCapture class that jumping to a certain frame may be inaccurate. It is fixed in this class by checking the position after jumping each time. Cache is used when decoding videos. So if the same frame is visited for the second time, there is no need to decode again if it is stored in the cache.

实际案例

```
>>> import mmcv
>>> v = mmcv.VideoReader('sample.mp4')
>>> len(v)  # get the total frame number with `len()`
120
>>> for img in v:  # v is iterable
>>>     mmcv.imshow(img)
>>> v[5]  # get the 6th frame
```

current_frame()

Get the current frame (frame that is just visited).

返回 If the video is fresh, return None, otherwise return the frame.

返回类型 ndarray or None

cvt2frames(*frame_dir*, *file_start=0*, *filename_tmpl='{:06d}.jpg'*, *start=0*, *max_num=0*, *show_progress=True*)

Convert a video to frame images.

参数

- **frame_dir** (*str*) –Output directory to store all the frame images.
- **file_start** (*int*) –Filenames will start from the specified number.
- **filename_tmpl** (*str*) –Filename template with the index as the placeholder.
- **start** (*int*) –The starting frame index.
- **max_num** (*int*) –Maximum number of frames to be written.
- **show_progress** (*bool*) –Whether to show a progress bar.

property fourcc

"Four character code" of the video.

Type str

property fps

FPS of the video.

Type float

property frame_cnt

Total frames of the video.

Type int

get_frame(*frame_id*)

Get frame by index.

参数 **frame_id** (*int*) –Index of the expected frame, 0-based.

返回 Return the frame if successful, otherwise None.

返回类型 ndarray or None

property height

Height of video frames.

Type int

property opened

Indicate whether the video is opened.

Type bool

property position

Current cursor position, indicating frame decoded.

Type int

read()

Read the next frame.

If the next frame have been decoded before and in the cache, then return it directly, otherwise decode, cache and return it.

返回 Return the frame if successful, otherwise None.

返回类型 ndarray or None

property resolution

Video resolution (width, height).

Type tuple

property vcap

The raw VideoCapture object.

Type cv2.VideoCapture

property width

Width of video frames.

Type int

mmcv.video.**concat_video**(*video_list: List*, *out_file: str*, *vcodec: Optional[str] = None*, *acodec: Optional[str] = None*, *log_level: str = 'info'*, *print_cmd: bool = False*) → None

Concatenate multiple videos into a single one.

参数

- **video_list** (*list*) –A list of video filenames
- **out_file** (*str*) –Output video filename
- **vcodec** (*None or str*) –Output video codec, None for unchanged
- **acodec** (*None or str*) –Output audio codec, None for unchanged
- **log_level** (*str*) –Logging level of ffmpeg.
- **print_cmd** (*bool*) –Whether to print the final ffmpeg command.

mmcv.video.**convert_video**(*in_file: str*, *out_file: str*, *print_cmd: bool = False*, *pre_options: str = ''*, ***kwargs*) → None

Convert a video with ffmpeg.

This provides a general api to ffmpeg, the executed command is:

```
`ffmpeg -y <pre_options> -i <in_file> <options> <out_file>`
```

Options(kwargs) are mapped to ffmpeg commands with the following rules:

- key=val: "-key val"

- key=True: "-key"
- key=False: ""

参数

- **in_file** (*str*) –Input video filename.
- **out_file** (*str*) –Output video filename.
- **pre_options** (*str*) –Options appears before "-i <in_file>".
- **print_cmd** (*bool*) –Whether to print the final ffmpeg command.

mmcv.video.**cut_video**(*in_file: str*, *out_file: str*, *start: Optional[float] = None*, *end: Optional[float] = None*, *vcodec: Optional[str] = None*, *acodec: Optional[str] = None*, *log_level: str = 'info'*, *print_cmd: bool = False*) → None

Cut a clip from a video.

参数

- **in_file** (*str*) –Input video filename.
- **out_file** (*str*) –Output video filename.
- **start** (*None or float*) –Start time (in seconds).
- **end** (*None or float*) –End time (in seconds).
- **vcodec** (*None or str*) –Output video codec, None for unchanged.
- **acodec** (*None or str*) –Output audio codec, None for unchanged.
- **log_level** (*str*) –Logging level of ffmpeg.
- **print_cmd** (*bool*) –Whether to print the final ffmpeg command.

mmcv.video.**dequantize_flow**(*dx: numpy.ndarray*, *dy: numpy.ndarray*, *max_val: float = 0.02*, *denorm: bool = True*) → numpy.ndarray

Recover from quantized flow.

参数

- **dx** (*ndarray*) –Quantized dx.
- **dy** (*ndarray*) –Quantized dy.
- **max_val** (*float*) –Maximum value used when quantizing.
- **denorm** (*bool*) –Whether to multiply flow values with width/height.

返回 Dequantized flow.

返回类型 ndarray

mmcv.video.**flow_from_bytes**(*content: bytes*) → numpy.ndarray

Read dense optical flow from bytes.

注解： This load optical flow function works for FlyingChairs, FlyingThings3D, Sintel, FlyingChairsOcc datasets, but cannot load the data from ChairsSDHom.

参数 **content** (*bytes*) –Optical flow bytes got from files or other streams.

返回 Loaded optical flow with the shape (H, W, 2).

返回类型 ndarray

mmcv.video.**flow_warp**(*img: numpy.ndarray*, *flow: numpy.ndarray*, *filling_value: int = 0*, *interpolate_mode: str = 'nearest'*) → numpy.ndarray

Use flow to warp img.

参数

- **img** (*ndarray*) –Image to be warped.
- **flow** (*ndarray*) –Optical Flow.
- **filling_value** (*int*) –The missing pixels will be set with filling_value.
- **interpolate_mode** (*str*) –bilinear -> Bilinear Interpolation; nearest -> Nearest Neighbor.

返回 Warped image with the same shape of img

返回类型 ndarray

mmcv.video.**flowread**(*flow_or_path: Union[numpy.ndarray, str]*, *quantize: bool = False*, *concat_axis: int = 0*, **args*, ***kwargs*) → numpy.ndarray

Read an optical flow map.

参数

- **flow_or_path** (*ndarray or str*) –A flow map or filepath.
- **quantize** (*bool*) –whether to read quantized pair, if set to True, remaining args will be passed to *dequantize_flow()*.
- **concat_axis** (*int*) –The axis that dx and dy are concatenated, can be either 0 or 1. Ignored if quantize is False.

返回 Optical flow represented as a (h, w, 2) numpy array

返回类型 ndarray

mmcv.video.**flowwrite**(*flow: numpy.ndarray*, *filename: str*, *quantize: bool = False*, *concat_axis: int = 0*, **args*, ***kwargs*) → None

Write optical flow to file.

If the flow is not quantized, it will be saved as a .flo file losslessly, otherwise a jpeg image which is lossy but of much smaller size. (dx and dy will be concatenated horizontally into a single image if quantize is True.)

参数

- **flow** (*ndarray*) –(h, w, 2) array of optical flow.
- **filename** (*str*) –Output filepath.
- **quantize** (*bool*) –Whether to quantize the flow and save it to 2 jpeg images. If set to True, remaining args will be passed to *quantize_flow()*.
- **concat_axis** (*int*) –The axis that dx and dy are concatenated, can be either 0 or 1. Ignored if quantize is False.

mmcv.video.**frames2video**(*frame_dir: str*, *video_file: str*, *fps: float = 30*, *fourcc: str = 'XVID'*, *filename_tmpl: str = '{:06d}.jpg'*, *start: int = 0*, *end: int = 0*, *show_progress: bool = True*) → None

Read the frame images from a directory and join them as a video.

参数

- **frame_dir** (*str*) –The directory containing video frames.
- **video_file** (*str*) –Output filename.
- **fps** (*float*) –FPS of the output video.
- **fourcc** (*str*) –Fourcc of the output video, this should be compatible with the output file type.
- **filename_tmpl** (*str*) –Filename template with the index as the variable.
- **start** (*int*) –Starting frame index.
- **end** (*int*) –Ending frame index.
- **show_progress** (*bool*) –Whether to show a progress bar.

mmcv.video.**quantize_flow**(*flow: numpy.ndarray*, *max_val: float = 0.02*, *norm: bool = True*) → tuple

Quantize flow to [0, 255].

After this step, the size of flow will be much smaller, and can be dumped as jpeg images.

参数

- **flow** (*ndarray*) –(h, w, 2) array of optical flow.
- **max_val** (*float*) –Maximum value of flow, values beyond [-max_val, max_val] will be truncated.
- **norm** (*bool*) –Whether to divide flow values by image width/height.

返回 Quantized dx and dy.

返回类型 tuple[ndarray]

mmcv.video.**resize_video**(*in_file: str*, *out_file: str*, *size: Optional[tuple] = None*, *ratio: Optional[Union[tuple, float]] = None*, *keep_ar: bool = False*, *log_level: str = 'info'*, *print_cmd: bool = False*) → None

Resize a video.

参数

- **in_file** (*str*) –Input video filename.
- **out_file** (*str*) –Output video filename.
- **size** (*tuple*) –Expected size (w, h), eg, (320, 240) or (320, -1).
- **ratio** (*tuple or float*) –Expected resize ratio, (2, 0.5) means (w*2, h*0.5).
- **keep_ar** (*bool*) –Whether to keep original aspect ratio.
- **log_level** (*str*) –Logging level of ffmpeg.
- **print_cmd** (*bool*) –Whether to print the final ffmpeg command.

mmcv.video.**sparse_flow_from_bytes**(*content: bytes*) → Tuple[numpy.ndarray, numpy.ndarray]

Read the optical flow in KITTI datasets from bytes.

This function is modified from RAFT load the KITTI datasets.

参数 **content** (*bytes*) –Optical flow bytes got from files or other streams.

返回 Loaded optical flow with the shape (H, W, 2) and flow valid mask with the shape (H, W).

返回类型 Tuple(ndarray, ndarray)

CHAPTER 29

arraymisc

mmcv.arraymisc.**dequantize**(*arr: numpy.ndarray, min_val: Union[int, float], max_val: Union[int, float], levels: int, dtype=<class 'numpy.float64'>*) → tuple

Dequantize an array.

参数

- **arr** (*ndarray*) –Input array.
- **min_val** (*int or float*) –Minimum value to be clipped.
- **max_val** (*int or float*) –Maximum value to be clipped.
- **levels** (*int*) –Quantization levels.
- **dtype** (*np.type*) –The type of the dequantized array.

返回 Dequantized array.

返回类型 tuple

mmcv.arraymisc.**quantize**(*arr: numpy.ndarray, min_val: Union[int, float], max_val: Union[int, float], levels: int, dtype=<class 'numpy.int64'>*) → tuple

Quantize an array of (-inf, inf) to [0, levels-1].

参数

- **arr** (*ndarray*) –Input array.
- **min_val** (*int or float*) –Minimum value to be clipped.
- **max_val** (*int or float*) –Maximum value to be clipped.

- **levels** (*int*) –Quantization levels.
- **dtype** (*np.type*) –The type of the quantized array.

返回 Quantized array.

返回类型 tuple

CHAPTER 30

visualization

class mmcv.visualization.**Color**(*value*)

An enum that defines common colors.

Contains red, green, blue, cyan, yellow, magenta, white and black.

mmcv.visualization.**color_val**(*color: Union[*mmcv.visualization.color.Color*, str, tuple, int, numpy.ndarray]*) → tuple

Convert various input to color tuples.

参数 **color** (*Color*/str/tuple/int/ndarray) –Color inputs

返回 A tuple of 3 integers indicating BGR channels.

返回类型 tuple[int]

mmcv.visualization.**flow2rgb**(*flow: numpy.ndarray*, *color_wheel: Optional[numpy.ndarray] = None*, *unknown_thr: float = 1000000.0*) → numpy.ndarray

Convert flow map to RGB image.

参数

- **flow** (*ndarray*) –Array of optical flow.
- **color_wheel** (*ndarray or None*) –Color wheel used to map flow field to RGB colorspace. Default color wheel will be used if not specified.
- **unknown_thr** (*float*) –Values above this threshold will be marked as unknown and thus ignored.

返回 RGB image that can be visualized.

返回类型 ndarray

mmcv.visualization.**flowshow**(*flow: Union[numpy.ndarray, str]*, *win_name: str = ''*, *wait_time: int = 0*) → None

Show optical flow.

参数

- **flow** (*ndarray or str*) –The optical flow to be displayed.
- **win_name** (*str*) –The window name.
- **wait_time** (*int*) –Value of waitKey param.

mmcv.visualization.**imshow**(*img: Union[str, numpy.ndarray]*, *win_name: str = ''*, *wait_time: int = 0*)

Show an image.

参数

- **img** (*str or ndarray*) –The image to be displayed.
- **win_name** (*str*) –The window name.
- **wait_time** (*int*) –Value of waitKey param.

mmcv.visualization.**imshow_bboxes**(*img: Union[str, numpy.ndarray]*, *bboxes: Union[list, numpy.ndarray]*, *colors: Union[*mmcv.visualization.color.Color*, str, tuple, int, numpy.ndarray] = 'green'*, *top_k: int = - 1*, *thickness: int = 1*, *show: bool = True*, *win_name: str = ''*, *wait_time: int = 0*, *out_file: Optional[str] = None*)

Draw bboxes on an image.

参数

- **img** (*str or ndarray*) –The image to be displayed.
- **bboxes** (*list or ndarray*) –A list of ndarray of shape (k, 4).
- **colors** (*Color or str or tuple or int or ndarray*) –A list of colors.
- **top_k** (*int*) –Plot the first k bboxes only if set positive.
- **thickness** (*int*) –Thickness of lines.
- **show** (*bool*) –Whether to show the image.
- **win_name** (*str*) –The window name.
- **wait_time** (*int*) –Value of waitKey param.
- **out_file** (*str, optional*) –The filename to write the image.

返回 The image with bboxes drawn on it.

返回类型 ndarray

mmcv.visualization.**imshow_det_bboxes**(*img: Union[str, numpy.ndarray]*, *bboxes: numpy.ndarray*, *labels: numpy.ndarray*, *class_names: Optional[List[str]] = None*, *score_thr: float = 0*, *bbox_color: Union[*mmcv.visualization.color.Color*, str, tuple, int, numpy.ndarray] = 'green'*, *text_color: Union[*mmcv.visualization.color.Color*, str, tuple, int, numpy.ndarray] = 'green'*, *thickness: int = 1*, *font_scale: float = 0.5*, *show: bool = True*, *win_name: str = ''*, *wait_time: int = 0*, *out_file: Optional[str] = None*)

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

参数

- **img** (*str or ndarray*) –The image to be displayed.
- **bboxes** (*ndarray*) –Bounding boxes (with scores), shaped (n, 4) or (n, 5).
- **labels** (*ndarray*) –Labels of bboxes.
- **class_names** (*list[str]*) –Names of each classes.
- **score_thr** (*float*) –Minimum score of bboxes to be shown.
- **bbox_color** (*Color or str or tuple or int or ndarray*) –Color of bbox lines.
- **text_color** (*Color or str or tuple or int or ndarray*) –Color of texts.
- **thickness** (*int*) –Thickness of lines.
- **font_scale** (*float*) –Font scales of texts.
- **show** (*bool*) –Whether to show the image.
- **win_name** (*str*) –The window name.
- **wait_time** (*int*) –Value of waitKey param.
- **out_file** (*str or None*) –The filename to write the image.

返回 The image with bboxes drawn on it.

返回类型 ndarray

mmcv.visualization.**make_color_wheel**(*bins: Optional[Union[list, tuple]] = None*) → numpy.ndarray

Build a color wheel.

参数 **bins** (*list or tuple, optional*) –Specify the number of bins for each color range, corresponding to six ranges: red -> yellow, yellow -> green, green -> cyan, cyan -> blue, blue -> magenta, magenta -> red. [15, 6, 4, 11, 13, 6] is used for default (see Middlebury).

返回 Color wheel of shape (total_bins, 3).

返回类型 ndarray

CHAPTER 31

utils

class mmcv.utils.**BuildExtension**(**args*, ***kwargs*)

A custom `setuptools` build extension .

This `setuptools.build_ext` subclass takes care of passing the minimum required compiler flags (e.g. `-std=c++14`) as well as mixed C++/CUDA compilation (and support for CUDA files in general).

When using *BuildExtension*, it is allowed to supply a dictionary for `extra_compile_args` (rather than the usual list) that maps from languages (`cxx` or `nvcc`) to a list of additional compiler flags to supply to the compiler. This makes it possible to supply different flags to the C++ and CUDA compiler during mixed compilation.

`use_ninja` (bool): If `use_ninja` is `True` (default), then we attempt to build using the Ninja backend. Ninja greatly speeds up compilation compared to the standard `setuptools.build_ext`. Fallbacks to the standard distutils backend if Ninja is not available.

注解: By default, the Ninja backend uses #CPUS + 2 workers to build the extension. This may use up too many resources on some systems. One can control the number of workers by setting the *MAX_JOBS* environment variable to a non-negative number.

---

finalize_options() → None

Set final values for all the options that this command supports. This is always called as late as possible, ie. after any option assignments from the command-line or from other commands have been done. Thus, this is the place to code option dependencies: if 'foo' depends on 'bar', then it is safe to set 'foo' from 'bar' as long as 'foo' still has the same value it was assigned in 'initialize_options()'.

This method must be implemented by all command classes.

get_ext_filename(*ext_name*)

Convert the name of an extension (eg. "foo.bar") into the name of the file from which it will be loaded (eg. "foo/bar.so", or "foobar.pyd").

classmethod with_options(***options*)

Returns a subclass with alternative constructor that extends any original keyword arguments to the original constructor with the given options.

mmcv.utils.**CUDAExtension**(*name*, *sources*, **args*, ***kwargs*)

Creates a `setuptools.Extension` for CUDA/C++.

Convenience method that creates a `setuptools.Extension` with the bare minimum (but often sufficient) arguments to build a CUDA/C++ extension. This includes the CUDA include path, library path and runtime library.

All arguments are forwarded to the `setuptools.Extension` constructor.

示例

```
>>> from setuptools import setup
>>> from torch.utils.cpp_extension import BuildExtension, CUDAExtension
>>> setup(
        name='cuda_extension',
        ext_modules=[
            CUDAExtension(
                    name='cuda_extension',
                    sources=['extension.cpp', 'extension_kernel.cu'],
                    extra_compile_args={'cxx': ['-g'],
                                        'nvcc': ['-O2']})
        ],
        cmdclass={
            'build_ext': BuildExtension
        })
```

Compute capabilities:

By default the extension will be compiled to run on all archs of the cards visible during the building process of the extension, plus PTX. If down the road a new card is installed the extension may need to be recompiled. If a visible card has a compute capability (CC) that's newer than the newest version for which your nvcc can build fully-compiled binaries, Pytorch will make nvcc fall back to building kernels with the newest version of PTX your nvcc does support (see below for details on PTX).

You can override the default behavior using *TORCH_CUDA_ARCH_LIST* to explicitly specify which CCs you want the extension to support:

TORCH_CUDA_ARCH_LIST="6.1 8.6" python build_my_extension.py TORCH_CUDA_ARCH_LIST="5.2 6.0 6.1 7.0 7.5 8.0 8.6+PTX" python build_my_extension.py

The +PTX option causes extension kernel binaries to include PTX instructions for the specified CC. PTX is an intermediate representation that allows kernels to runtime-compile for any CC >= the specified CC (for example, 8.6+PTX generates PTX that can runtime-compile for any GPU with CC >= 8.6). This improves your binary's forward compatibility. However, relying on older PTX to provide forward compat by runtime-compiling for newer CCs can modestly reduce performance on those newer CCs. If you know exact CC(s) of the GPUs you want to target, you're always better off specifying them individually. For example, if you want your extension to run on 8.0 and 8.6, "8.0+PTX" would work functionally because it includes PTX that can runtime-compile for 8.6, but "8.0 8.6" would be better.

Note that while it's possible to include all supported archs, the more archs get included the slower the building process will be, as it will build a separate kernel image for each arch.

Note that CUDA-11.5 nvcc will hit internal compiler error while parsing torch/extension.h on Windows. To workaround the issue, move python binding logic to pure C++ file.

Example use:

```
>>> #include <ATen/ATen.h>
>>> at::Tensor SigmoidAlphaBlendForwardCuda(....)
```

Instead of:

```
>>> #include <torch/extension.h>
>>> torch::Tensor SigmoidAlphaBlendForwardCuda(...)
```

Currently open issue for nvcc bug: https://github.com/pytorch/pytorch/issues/69460 Complete workaround code example: https://github.com/facebookresearch/pytorch3d/commit/cb170ac024a949f1f9614ffe6af1c38d972f7d48

class mmcv.utils.**Config**(*cfg_dict=None*, *cfg_text=None*, *filename=None*)

A facility for config and config files.

It supports common file formats as configs: python/json/yaml. The interface is the same as a dict object and also allows access config values as attributes.

示例

```
>>> cfg = Config(dict(a=1, b=dict(b1=[0, 1])))
>>> cfg.a
1
>>> cfg.b
{'b1': [0, 1]}
>>> cfg.b.b1
[0, 1]
>>> cfg = Config.fromfile('tests/data/config/a.py')
```

(下页继续)

(续上页)

```
>>> cfg.filename
"/home/kchen/projects/mmcv/tests/data/config/a.py"
>>> cfg.item4
'test'
>>> cfg
"Config [path: /home/kchen/projects/mmcv/tests/data/config/a.py]: "
"{'item1': [1, 2], 'item2': {'a': 0}, 'item3': True, 'item4': 'test'}"
```

static auto_argparser(*description=None*)

Generate argparser from config file automatically (experimental)

dump(*file=None*)

Dumps config into a file or returns a string representation of the config.

If a file argument is given, saves the config to that file using the format defined by the file argument extension.

Otherwise, returns a string representing the config. The formatting of this returned string is defined by the extension of *self.filename*. If *self.filename* is not defined, returns a string representation of a

dict (lowercased and using ‘for strings).

实际案例

```
>>> cfg_dict = dict(item1=[1, 2], item2=dict(a=0),
...     item3=True, item4='test')
>>> cfg = Config(cfg_dict=cfg_dict)
>>> dump_file = "a.py"
>>> cfg.dump(dump_file)
```

参数 **file**(*str, optional*) –Path of the output file where the config will be dumped. Defaults to None.

static fromstring(*cfg_str*, *file_format*)

Generate config from config str.

参数

- **cfg_str**(*str*) –Config str.
- **file_format** (*str*) –Config file format corresponding to the config str. Only py/yml/yaml/json type are supported now!

返回 Config obj.

返回类型 *Config*

merge_from_dict(*options*, *allow_list_keys=True*)

Merge list into cfg_dict.

Merge the dict parsed by MultipleKVAction into this cfg.

实际案例

```
>>> options = {'model.backbone.depth': 50,
...            'model.backbone.with_cp':True}
>>> cfg = Config(dict(model=dict(backbone=dict(type='ResNet'))))
>>> cfg.merge_from_dict(options)
>>> cfg_dict = super(Config, self).__getattribute__('_cfg_dict')
>>> assert cfg_dict == dict(
...     model=dict(backbone=dict(depth=50, with_cp=True)))
```

```
>>> # Merge list element
>>> cfg = Config(dict(pipeline=[
...     dict(type='LoadImage'), dict(type='LoadAnnotations')]))
>>> options = dict(pipeline={'0': dict(type='SelfLoadImage')})
>>> cfg.merge_from_dict(options, allow_list_keys=True)
>>> cfg_dict = super(Config, self).__getattribute__('_cfg_dict')
>>> assert cfg_dict == dict(pipeline=[
...     dict(type='SelfLoadImage'), dict(type='LoadAnnotations')])
```

参数

- **options** (*dict*) –dict of configs to merge from.
- **allow_list_keys** (*bool*) –If True, int string keys (e.g. ‘0’, ‘1’) are allowed in `options` and will replace the element of the corresponding index in the config if the config is a list. Default: True.

class mmcv.utils.**ConfigDict**(**args*, ***kwargs*)

mmcv.utils.**CppExtension**(*name*, *sources*, **args*, ***kwargs*)

Creates a `setuptools.Extension` for C++.

Convenience method that creates a `setuptools.Extension` with the bare minimum (but often sufficient) arguments to build a C++ extension.

All arguments are forwarded to the `setuptools.Extension` constructor.

示例

```
>>> from setuptools import setup
>>> from torch.utils.cpp_extension import BuildExtension, CppExtension
>>> setup(
        name='extension',
        ext_modules=[
            CppExtension(
                name='extension',
                sources=['extension.cpp'],
                extra_compile_args=['-g']),
        ],
        cmdclass={
            'build_ext': BuildExtension
        })
```

class mmcv.utils.**DataLoader**(*dataset: torch.utils.data.dataset.Dataset[torch.utils.data.dataloader.T_co]*, *batch_size: Optional[int] = 1*, *shuffle: Optional[bool] = None*, *sampler: Optional[Union[torch.utils.data.sampler.Sampler, Iterable]] = None*, *batch_sampler: Optional[Union[torch.utils.data.sampler.Sampler[Sequence], Iterable[Sequence]]] = None*, *num_workers: int = 0*, *collate_fn: Optional[Callable[[List[torch.utils.data.dataloader.T]], Any]] = None*, *pin_memory: bool = False*, *drop_last: bool = False*, *timeout: float = 0*, *worker_init_fn: Optional[Callable[[int], None]] = None*, *multiprocessing_context=None*, *generator=None*, ***, *prefetch_factor: int = 2*, *persistent_workers: bool = False*, *pin_memory_device: str = ''*)

Data loader. Combines a dataset and a sampler, and provides an iterable over the given dataset.

The `DataLoader` supports both map-style and iterable-style datasets with single- or multi-process loading, customizing loading order and optional automatic batching (collation) and memory pinning.

See `torch.utils.data` documentation page for more details.

参数

- **dataset** (*Dataset*) –dataset from which to load the data.
- **batch_size** (*int, optional*) –how many samples per batch to load (default: `1`).
- **shuffle** (*bool, optional*) –set to `True` to have the data reshuffled at every epoch (default: `False`).
- **sampler** (*Sampler or Iterable, optional*) –defines the strategy to draw samples from the dataset. Can be any `Iterable` with `__len__` implemented. If specified, `shuffle` must not be specified.
- **batch_sampler** (*Sampler or Iterable, optional*) –like `sampler`, but returns a batch of indices at a time. Mutually exclusive with `batch_size`, `shuffle`,

sampler, and drop_last.

- **num_workers** (*int, optional*) –how many subprocesses to use for data loading. 0 means that the data will be loaded in the main process. (default: 0)
- **collate_fn** (*callable, optional*) –merges a list of samples to form a mini-batch of Tensor(s). Used when using batched loading from a map-style dataset.
- **pin_memory** (*bool, optional*) –If True, the data loader will copy Tensors into device/CUDA pinned memory before returning them. If your data elements are a custom type, or your collate_fn returns a batch that is a custom type, see the example below.
- **drop_last** (*bool, optional*) –set to True to drop the last incomplete batch, if the dataset size is not divisible by the batch size. If False and the size of dataset is not divisible by the batch size, then the last batch will be smaller. (default: False)
- **timeout** (*numeric, optional*) –if positive, the timeout value for collecting a batch from workers. Should always be non-negative. (default: 0)
- **worker_init_fn** (*callable, optional*) –If not None, this will be called on each worker subprocess with the worker id (an int in [0, num_workers - 1]) as input, after seeding and before data loading. (default: None)
- **generator** (*torch.Generator, optional*) –If not None, this RNG will be used by RandomSampler to generate random indexes and multiprocessing to generate *base_seed* for workers. (default: None)
- **prefetch_factor** (*int, optional, keyword-only arg*) –Number of batches loaded in advance by each worker. 2 means there will be a total of 2 * num_workers batches prefetched across all workers. (default: 2)
- **persistent_workers** (*bool, optional*) –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. (default: False)
- **pin_memory_device** (*str, optional*) –the data loader will copy Tensors into device pinned memory before returning them if pin_memory is set to true.

警告: If the spawn start method is used, *worker_init_fn* cannot be an unpicklable object, e.g., a lambda function. See multiprocessing-best-practices on more details related to multiprocessing in PyTorch.

警告: len(dataloader) heuristic is based on the length of the sampler used. When dataset is an IterableDataset, it instead returns an estimate based on len(dataset) / batch_size, with proper rounding depending on drop_last, regardless of multi-process loading configurations. This represents the best guess PyTorch can make because PyTorch trusts user dataset code in correctly handling

multi-process loading to avoid duplicate data.

However, if sharding results in multiple workers having incomplete last batches, this estimate can still be inaccurate, because (1) an otherwise complete batch can be broken into multiple ones and (2) more than one batch worth of samples can be dropped when `drop_last` is set. Unfortunately, PyTorch can not detect such cases in general.

See **'Dataset Types'_** for more details on these two types of datasets and how `IterableDataset` interacts with **'Multi-process data loading'_**.

警告: See reproducibility, and dataloader-workers-random-seed, and data-loading-randomness notes for random seed related questions.

class mmcv.utils.**DictAction**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)

argparse action to split an argument into KEY=VALUE form on the first = and append to a dictionary. List options can be passed as comma separated values, i.e 'KEY=V1,V2,V3', or with explicit brackets, i.e. 'KEY=[V1,V2,V3]'. It also support nested brackets to build list/tuple values. e.g. 'KEY=[(V1,V2),(V3,V4)]'

mmcv.utils.**PoolDataLoader**

alias of *torch.utils.data.dataloader.DataLoader*

class mmcv.utils.**ProgressBar**(*task_num=0*, *bar_width=50*, *start=True*, *file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

A progress bar which can print the progress.

class mmcv.utils.**Registry**(*name*, *build_func=None*, *parent=None*, *scope=None*)

A registry to map strings to classes or functions.

Registered object could be built from registry. Meanwhile, registered functions could be called from registry.

示例

```
>>> MODELS = Registry('models')
>>> @MODELS.register_module()
>>> class ResNet:
>>>     pass
>>> resnet = MODELS.build(dict(type='ResNet'))
>>> @MODELS.register_module()
>>> def resnet50():
>>>     pass
>>> resnet = MODELS.build(dict(type='resnet50'))
```

Please refer to https://mmcv.readthedocs.io/en/latest/understand_mmcv/registry.html for advanced usage.

参数

- **name** (*str*) –Registry name.
- **build_func** (*func, optional*) –Build function to construct instance from Registry, func:*build_from_cfg* is used if neither `parent` or `build_func` is specified. If `parent` is specified and `build_func` is not given, `build_func` will be inherited from `parent`. Default: None.
- **parent** (`Registry`, *optional*) –Parent registry. The class registered in children registry could be built from parent. Default: None.
- **scope** (*str, optional*) –The scope of registry. It is the key to search for children registry. If not specified, scope will be the name of the package where class is defined, e.g. mmdet, mmcls, mmseg. Default: None.

get (*key*)

Get the registry record.

参数 **key** (*str*) –The class name in string format.

返回 The corresponding class.

返回类型 class

static infer_scope ()

Infer the scope of registry.

The name of the package where registry is defined will be returned.

示例

```
>>> # in mmdet/models/backbone/resnet.py
>>> MODELS = Registry('models')
>>> @MODELS.register_module()
>>> class ResNet:
>>>     pass
The scope of ``ResNet`` will be ``mmdet``.
```

返回 The inferred scope name.

返回类型 str

register_module (*name=None*, *force=False*, *module=None*)

Register a module.

A record will be added to *self._module_dict*, whose key is the class name or the specified name, and value is the class itself. It can be used as a decorator or a normal function.

示例

```
>>> backbones = Registry('backbone')
>>> @backbones.register_module()
>>> class ResNet:
>>>     pass
```

```
>>> backbones = Registry('backbone')
>>> @backbones.register_module(name='mnet')
>>> class MobileNet:
>>>     pass
```

```
>>> backbones = Registry('backbone')
>>> class ResNet:
>>>     pass
>>> backbones.register_module(ResNet)
```

参数

- **name** (*str | None*) –The module name to be registered. If not specified, the class name will be used.
- **force** (*bool, optional*) –Whether to override an existing class with the same name. Default: False.
- **module** (*type*) –Module class or function to be registered.

static split_scope_key(*key*)

Split scope and key.

The first scope will be split from key.

实际案例

```
>>> Registry.split_scope_key('mmdet.ResNet')
'mmdet', 'ResNet'
>>> Registry.split_scope_key('ResNet')
None, 'ResNet'
```

返回 The former element is the first scope of the key, which can be `None`. The latter is the remaining key.

返回类型 tuple[str | None, str]

class mmcv.utils.**SyncBatchNorm**(*num_features: int*, *eps: float = 1e-05*, *momentum: float = 0.1*, *affine: bool = True*, *track_running_stats: bool = True*, *process_group: Optional[Any] = None*, *device=None*, *dtype=None*)

class mmcv.utils.**Timer**(*start=True*, *print_tmpl=None*)

A flexible Timer class.

实际案例

```
>>> import time
>>> import mmcv
>>> with mmcv.Timer():
>>>     # simulate a code block that will run for 1s
>>>     time.sleep(1)
1.000
>>> with mmcv.Timer(print_tmpl='it takes {:.1f} seconds'):
>>>     # simulate a code block that will run for 1s
>>>     time.sleep(1)
it takes 1.0 seconds
>>> timer = mmcv.Timer()
>>> time.sleep(0.5)
>>> print(timer.since_start())
0.500
>>> time.sleep(0.5)
>>> print(timer.since_last_check())
0.500
>>> print(timer.since_start())
1.000
```

property is_running

indicate whether the timer is running

Type bool

since_last_check()

Time since the last checking.

Either *since_start()* or *since_last_check()* is a checking operation.

返回 Time in seconds.

返回类型 float

since_start()

Total time since the timer is started.

返回 Time in seconds.

返回类型 float

start()

Start the timer.

exception mmcv.utils.**TimerError**(*message*)

mmcv.utils.**assert_attrs_equal**(*obj: Any*, *expected_attrs: Dict[str, Any]*) → bool

Check if attribute of class object is correct.

参数

- **obj** (*object*) –Class object to be checked.
- **expected_attrs** (*Dict[str, Any]*) –Dict of the expected attrs.

返回 Whether the attribute of class object is correct.

返回类型 bool

mmcv.utils.**assert_dict_contains_subset**(*dict_obj: Dict[Any, Any]*, *expected_subset: Dict[Any, Any]*) → bool

Check if the dict_obj contains the expected_subset.

参数

- **dict_obj** (*Dict[Any, Any]*) –Dict object to be checked.
- **expected_subset** (*Dict[Any, Any]*) –Subset expected to be contained in dict_obj.

返回 Whether the dict_obj contains the expected_subset.

返回类型 bool

mmcv.utils.**assert_dict_has_keys**(*obj: Dict[str, Any]*, *expected_keys: List[str]*) → bool

Check if the obj has all the expected_keys.

参数

- **obj** (*Dict[str, Any]*) –Object to be checked.
- **expected_keys** (*List[str]*) –Keys expected to contained in the keys of the obj.

返回 Whether the obj has the expected keys.

返回类型 bool

mmcv.utils.**assert_is_norm_layer**(*module*) → bool

Check if the module is a norm layer.

参数 **module** (*nn.Module*) –The module to be checked.

返回 Whether the module is a norm layer.

返回类型 bool

mmcv.utils.**assert_keys_equal**(*result_keys: List[str]*, *target_keys: List[str]*) → bool

Check if target_keys is equal to result_keys.

参数

- **result_keys** (*List[str]*) –Result keys to be checked.
- **target_keys** (*List[str]*) –Target keys to be checked.

返回 Whether target_keys is equal to result_keys.

返回类型 bool

mmcv.utils.**assert_params_all_zeros**(*module*) → bool

Check if the parameters of the module is all zeros.

参数 **module** (*nn.Module*) –The module to be checked.

返回 Whether the parameters of the module is all zeros.

返回类型 bool

mmcv.utils.**build_from_cfg**(*cfg: Dict*, *registry:* mmcv.utils.registry.Registry, *default_args: Optional[Dict] = None*) → Any

Build a module from config dict when it is a class configuration, or call a function from config dict when it is a function configuration.

示例

```
>>> MODELS = Registry('models')
>>> @MODELS.register_module()
>>> class ResNet:
>>>     pass
>>> resnet = build_from_cfg(dict(type='Resnet'), MODELS)
>>> # Returns an instantiated object
>>> @MODELS.register_module()
>>> def resnet50():
>>>     pass
>>> resnet = build_from_cfg(dict(type='resnet50'), MODELS)
>>> # Return a result of the calling function
```

参数

- **cfg** (*dict*) –Config dict. It should at least contain the key “type”.
- **registry** (*Registry*) –The registry to search the type from.
- **default_args** (*dict, optional*) –Default initialization arguments.

返回 The constructed object.

返回类型 object

mmcv.utils.**check_prerequisites**(*prerequisites*, *checker*, *msg_tmpl='Prerequisites "{}" are required in method "{}" but not found, please install them first.'*)

A decorator factory to check if prerequisites are satisfied.

参数

- **prerequisites** (*str of list[str]*) –Prerequisites to be checked.
- **checker** (*callable*) –The checker method that returns True if a prerequisite is meet, False otherwise.
- **msg_tmpl** (*str*) –The message template with two variables.

返回 A specific decorator.

返回类型 decorator

mmcv.utils.**check_python_script**(*cmd*)

Run the python cmd script with *__main__*. The difference between *os.system* is that, this function exectues code in the current process, so that it can be tracked by coverage tools. Currently it supports two forms:

- ./tests/data/scripts/hello.py zz
- python tests/data/scripts/hello.py zz

mmcv.utils.**check_time**(*timer_id*)

Add check points in a single line.

This method is suitable for running a task on a list of items. A timer will be registered when the method is called for the first time.

实际案例

```
>>> import time
>>> import mmcv
>>> for i in range(1, 6):
>>>     # simulate a code block
>>>     time.sleep(i)
>>>     mmcv.check_time('task1')
2.000
3.000
4.000
5.000
```

参数 **str** –Timer identifier.

mmcv.utils.**collect_env**()

Collect the information of the running environments.

返回

The environment information. The following fields are contained.

- sys.platform: The variable of `sys.platform`.
- Python: Python version.
- CUDA available: Bool, indicating if CUDA is available.
- GPU devices: Device type of each GPU.
- CUDA_HOME (optional): The env var `CUDA_HOME`.
- NVCC (optional): NVCC version.
- GCC: GCC version, "n/a" if GCC is not installed.
- MSVC: Microsoft Virtual C++ Compiler version, Windows only.
- PyTorch: PyTorch version.
- PyTorch compiling details: The output of `torch.__config__.show()`.
- TorchVision (optional): TorchVision version.
- OpenCV: OpenCV version.
- MMCV: MMCV version.
- MMCV Compiler: The GCC version for compiling MMCV ops.
- MMCV CUDA Compiler: The CUDA version for compiling MMCV ops.

返回类型 dict

mmcv.utils.**concat_list**(*in_list*)

Concatenate a list of list into a single list.

参数 **in_list** (*list*) –The list of list to be merged.

返回 The concatenated flat list.

返回类型 list

mmcv.utils.**deprecated_api_warning**(*name_dict*, *cls_name=None*)

A decorator to check if some arguments are deprecate and try to replace deprecate src_arg_name to dst_arg_name.

参数 **name_dict** (*dict*) –key (str): Deprecate argument names. val (str): Expected argument names.

返回 New function.

返回类型 func

mmcv.utils.**digit_version**(*version_str: str*, *length: int = 4*)

Convert a version string into a tuple of integers.

This method is usually used for comparing two versions. For pre-release versions: alpha < beta < rc.

参数

- **version_str** (*str*) –The version string.
- **length** (*int*) –The maximum number of version levels. Default: 4.

返回 The version info in digits (integers).

返回类型 tuple[int]

mmcv.utils.**get_git_hash**(*fallback='unknown'*, *digits=None*)

Get the git hash of the current repo.

参数

- **fallback** (*str, optional*) –The fallback string when git hash is unavailable. Defaults to 'unknown'.
- **digits** (*int, optional*) –kept digits of the hash. Defaults to None, meaning all digits are kept.

返回 Git commit hash.

返回类型 str

mmcv.utils.**get_logger**(*name*, *log_file=None*, *log_level=20*, *file_mode='w'*)

Initialize and get a logger by name.

If the logger has not been initialized, this method will initialize the logger by adding one or two handlers, otherwise the initialized logger will be directly returned. During initialization, a StreamHandler will always be added. If *log_file* is specified and the process rank is 0, a FileHandler will also be added.

参数

- **name** (*str*) –Logger name.
- **log_file** (*str | None*) –The log filename. If specified, a FileHandler will be added to the logger.
- **log_level** (*int*) –The logger level. Note that only the process of rank 0 is affected, and other processes will set the level to "Error" thus be silent most of the time.
- **file_mode** (*str*) –The file mode used in opening log file. Defaults to 'w'.

返回 The expected logger.

返回类型 logging.Logger

mmcv.utils.**has_method**(*obj: object*, *method: str*) → bool

Check whether the object has a method.

参数

- **method** (*str*) –The method name to check.
- **obj** (*object*) –The object to check.

返回 True if the object has the method else False.

返回类型 bool

mmcv.utils.**import_modules_from_strings**(*imports*, *allow_failed_imports=False*)

Import modules from the given list of strings.

参数

- **imports** (*list | str | None*) –The given module names to be imported.
- **allow_failed_imports** (*bool*) –If True, the failed imports will return None. Otherwise, an ImportError is raise. Default: False.

返回 The imported modules.

返回类型 list[module] | module | None

实际案例

```
>>> osp, sys = import_modules_from_strings(
...     ['os.path', 'sys'])
>>> import os.path as osp_
>>> import sys as sys_
>>> assert osp == osp_
>>> assert sys == sys_
```

mmcv.utils.**is_list_of**(*seq*, *expected_type*)

Check whether it is a list of some type.

A partial method of *is_seq_of()*.

mmcv.utils.**is_method_overridden**(*method*, *base_class*, *derived_class*)

Check if a method of base class is overridden in derived class.

参数

- **method** (*str*) –the method name to check.
- **base_class** (*type*) –the class of the base class.
- **derived_class** (*type | Any*) –the class or instance of the derived class.

mmcv.utils.**is_seq_of**(*seq*, *expected_type*, *seq_type=None*)

Check whether it is a sequence of some type.

参数

- **seq** (*Sequence*) –The sequence to be checked.
- **expected_type** (*type*) –Expected type of sequence items.
- **seq_type** (*type, optional*) –Expected sequence type.

返回 Whether the sequence is valid.

返回类型 bool

mmcv.utils.**is_str**(*x*)

Whether the input is an string instance.

Note: This method is deprecated since python 2 is no longer supported.

mmcv.utils.**is_tuple_of**(*seq*, *expected_type*)

Check whether it is a tuple of some type.

A partial method of *is_seq_of()*.

mmcv.utils.**iter_cast**(*inputs*, *dst_type*, *return_type=None*)

Cast elements of an iterable object into some type.

参数

- **inputs** (*Iterable*) –The input object.
- **dst_type** (*type*) –Destination type.
- **return_type** (*type, optional*) –If specified, the output object will be converted to this type, otherwise an iterator.

返回 The converted object.

返回类型 iterator or specified type

mmcv.utils.**list_cast**(*inputs*, *dst_type*)

Cast elements of an iterable object into a list of some type.

A partial method of *iter_cast()*.

mmcv.utils.**load_url**(*url: str*, *model_dir: Optional[str] = None*, *map_location: Optional[Union[Callable[[str], str], Dict[str, str]]] = None*, *progress: bool = True*, *check_hash: bool = False*, *file_name: Optional[str] = None*) → Dict[str, Any]

Loads the Torch serialized object at the given URL.

If downloaded file is a zip file, it will be automatically decompressed.

If the object is already present in *model_dir*, it's deserialized and returned. The default value of `model_dir` is `<hub_dir>/checkpoints` where `hub_dir` is the directory returned by `get_dir()`.

参数

- **url** (*string*) –URL of the object to download
- **model_dir** (*string, optional*) –directory in which to save the object

- **map_location** (*optional*) –a function or a dict specifying how to remap storage locations (see torch.load)
- **progress** (*bool, optional*) –whether or not to display a progress bar to stderr. Default: True
- **check_hash** (*bool, optional*) –If True, the filename part of the URL should follow the naming convention `filename-<sha256>.ext` where `<sha256>` is the first eight or more digits of the SHA256 hash of the contents of the file. The hash is used to ensure unique names and to verify the contents of the file. Default: False
- **file_name** (*string, optional*) –name for the downloaded file. Filename from `url` will be used if not set.

示例

```
>>> state_dict = torch.hub.load_state_dict_from_url('https://s3.amazonaws.com/
↪pytorch/models/resnet18-5c106cde.pth')
```

mmcv.utils.**print_log**(*msg*, *logger=None*, *level=20*)

Print a log message.

参数

- **msg** (*str*) –The message to be logged.
- **logger** (*logging.Logger | str | None*) –The logger to be used. Some special loggers are:
  - ” silent” : no message will be printed.
  - other str: the logger obtained with *get_root_logger(logger)*.
  - None: The *print()* method will be used to print log messages.
- **level** (*int*) –Logging level. Only available when *logger* is a Logger object or “root” .

mmcv.utils.**requires_executable**(*prerequisites*)

A decorator to check if some executable files are installed.

示例

```
>>> @requires_executable('ffmpeg')
>>> func(arg1, args):
>>>     print(1)
1
```

mmcv.utils.**requires_package**(*prerequisites*)

A decorator to check if some python packages are installed.

示例

```
>>> @requires_package('numpy')
>>> func(arg1, args):
>>>     return numpy.zeros(1)
array([0.])
>>> @requires_package(['numpy', 'non_package'])
>>> func(arg1, args):
>>>     return numpy.zeros(1)
ImportError
```

mmcv.utils.**scandir**(*dir_path*, *suffix=None*, *recursive=False*, *case_sensitive=True*)

Scan a directory to find the interested files.

参数

- **dir_path** (str | Path) –Path of the directory.
- **suffix** (*str | tuple(str), optional*) –File suffix that we are interested in. Default: None.
- **recursive** (*bool, optional*) –If set to True, recursively scan the directory. Default: False.
- **case_sensitive** (*bool, optional*) –If set to False, ignore the case of suffix. Default: True.

返回 A generator for all the interested files with relative paths.

mmcv.utils.**slice_list**(*in_list*, *lens*)

Slice a list into several sub lists by a list of given length.

参数

- **in_list** (*list*) –The list to be sliced.
- **lens** (*int or list*) –The expected length of each out list.

返回 A list of sliced list.

返回类型 list

mmcv.utils.**torch_meshgrid**(**tensors*)

A wrapper of torch.meshgrid to compat different PyTorch versions.

Since PyTorch 1.10.0a0, torch.meshgrid supports the arguments `indexing`. So we implement a wrapper here to avoid warning when using high-version PyTorch and avoid compatibility issues when using previous versions of PyTorch.

参数 **tensors** (*List[Tensor]*) –List of scalars or 1 dimensional tensors.

返回 Sequence of meshgrid tensors.

返回类型 Sequence[Tensor]

mmcv.utils.**track_iter_progress**(*tasks*, *bar_width=50*, *file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

Track the progress of tasks iteration or enumeration with a progress bar.

Tasks are yielded with a simple for-loop.

参数

- **tasks** (*list or tuple[Iterable, int]*) –A list of tasks or (tasks, total num).
- **bar_width** (*int*) –Width of progress bar.

生成器 *list* –The task results.

mmcv.utils.**track_parallel_progress**(*func*, *tasks*, *nproc*, *initializer=None*, *initargs=None*, *bar_width=50*, *chunksize=1*, *skip_first=False*, *keep_order=True*, *file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

Track the progress of parallel task execution with a progress bar.

The built-in `multiprocessing` module is used for process pools and tasks are done with `Pool.map()` or `Pool.imap_unordered()`.

参数

- **func** (*callable*) –The function to be applied to each task.
- **tasks** (*list or tuple[Iterable, int]*) –A list of tasks or (tasks, total num).
- **nproc** (*int*) –Process (worker) number.
- **initializer** (*None or callable*) –Refer to `multiprocessing.Pool` for details.
- **initargs** (*None or tuple*) –Refer to `multiprocessing.Pool` for details.
- **chunksize** (*int*) –Refer to `multiprocessing.Pool` for details.
- **bar_width** (*int*) –Width of progress bar.
- **skip_first** (*bool*) –Whether to skip the first sample for each worker when estimating fps, since the initialization step may takes longer.
- **keep_order** (*bool*) –If True, `Pool.imap()` is used, otherwise `Pool.imap_unordered()` is used.

返回 The task results.

返回类型 list

mmcv.utils.**track_progress**(*func*, *tasks*, *bar_width=50*, *file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, ***kwargs*)

Track the progress of tasks execution with a progress bar.

Tasks are done with a simple for-loop.

参数

- **func** (*callable*) –The function to be applied to each task.
- **tasks** (*list or tuple[Iterable, int]*) –A list of tasks or (tasks, total num).
- **bar_width** (*int*) –Width of progress bar.

返回 The task results.

返回类型 list

mmcv.utils.**tuple_cast**(*inputs*, *dst_type*)

Cast elements of an iterable object into a tuple of some type.

A partial method of *iter_cast()*.

mmcv.utils.**worker_init_fn**(*worker_id: int*, *num_workers: int*, *rank: int*, *seed: int*)

Function to initialize each worker.

The seed of each worker equals to `num_worker * rank + worker_id + user_seed`.

参数

- **worker_id** (*int*) –Id for each worker.
- **num_workers** (*int*) –Number of workers.
- **rank** (*int*) –Rank in distributed training.
- **seed** (*int*) –Random seed.

CHAPTER 32

cnn

class mmcv.cnn.**AlexNet**(*num_classes: int = - 1*)

AlexNet backbone.

参数 **num_classes** (*int*) –number of classes for classification.

forward(*x: torch.Tensor*) → torch.Tensor

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 mmcv.cnn.**Caffe2XavierInit**(***kwargs*)

class mmcv.cnn.**ConstantInit**(*val: Union[int, float]*, ***kwargs*)

Initialize module parameters with constant values.

参数

- **val** (*int | float*) –the value to fill the weights in the module with
- **bias** (*int | float*) –the value to fill the bias. Defaults to 0.
- **bias_prob** (*float, optional*) –the probability for bias initialization. Defaults to None.

- **layer** (*str | list[str], optional*) –the layer will be initialized. Defaults to None.

class mmcv.cnn.**ContextBlock**(*in_channels: int*, *ratio: float*, *pooling_type: str = 'att'*, *fusion_types: tuple = ('channel_add')*)

ContextBlock module in GCNet.

See ‘GCNet: Non-local Networks Meet Squeeze-Excitation Networks and Beyond’ (https://arxiv.org/abs/1904.11492) for details.

参数

- **in_channels** (*int*) –Channels of the input feature map.
- **ratio** (*float*) –Ratio of channels of transform bottleneck
- **pooling_type** (*str*) –Pooling method for context modeling. Options are ‘att’ and ‘avg’, stand for attention pooling and average pooling respectively. Default: ‘att’.
- **fusion_types** (*Sequence[str]*) –Fusion method for feature fusion, Options are ‘channels_add’, ‘channel_mul’, stand for channelwise addition and multiplication respectively. Default: (‘channel_add’,)

forward(*x: torch.Tensor*) → torch.Tensor

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 mmcv.cnn.**Conv2d**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, int]]*, *stride: Union[int, Tuple[int, int]] = 1*, *padding: Union[str, int, Tuple[int, int]] = 0*, *dilation: Union[int, Tuple[int, int]] = 1*, *groups: int = 1*, *bias: bool = True*, *padding_mode: str = 'zeros'*, *device=None*, *dtype=None*)

forward(*x: torch.Tensor*) → torch.Tensor

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 mmcv.cnn.**Conv3d**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, int, int]]*, *stride: Union[int, Tuple[int, int, int]] = 1*, *padding: Union[str, int, Tuple[int, int, int]] = 0*, *dilation: Union[int, Tuple[int, int, int]] = 1*, *groups: int = 1*, *bias: bool = True*, *padding_mode: str = 'zeros'*, *device=None*, *dtype=None*)

forward(*x: torch.Tensor*) → torch.Tensor

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 mmcv.cnn.**ConvAWS2d**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, int]]*, *stride: Union[int, Tuple[int, int]] = 1*, *padding: Union[int, Tuple[int, int]] = 0*, *dilation: Union[int, Tuple[int, int]] = 1*, *groups: int = 1*, *bias: bool = True*)

AWS (Adaptive Weight Standardization)

This is a variant of Weight Standardization (https://arxiv.org/pdf/1903.10520.pdf) It is used in DetectoRS to avoid NaN (https://arxiv.org/pdf/2006.02334.pdf)

参数

- **in_channels** (*int*) –Number of channels in the input image
- **out_channels** (*int*) –Number of channels produced by the convolution
- **kernel_size** (*int or tuple*) –Size of the conv kernel
- **stride** (*int or tuple, optional*) –Stride of the convolution. Default: 1
- **padding** (*int or tuple, optional*) –Zero-padding added to both sides of the input. Default: 0
- **dilation** (*int or tuple, optional*) –Spacing between kernel elements. Default: 1
- **groups** (*int, optional*) –Number of blocked connections from input channels to output channels. Default: 1
- **bias** (*bool, optional*) –If set True, adds a learnable bias to the output. Default: True

forward(*x: torch.Tensor*) → torch.Tensor

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 mmcv.cnn.**ConvModule**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, int]]*, *stride: Union[int, Tuple[int, int]] = 1*, *padding: Union[int, Tuple[int, int]] = 0*, *dilation: Union[int, Tuple[int, int]] = 1*, *groups: int = 1*, *bias: Union[bool, str] = 'auto'*, *conv_cfg: Optional[Dict] = None*, *norm_cfg: Optional[Dict] = None*, *act_cfg: Optional[Dict] = {'type': 'ReLU'}*, *inplace: bool = True*, *with_spectral_norm: bool = False*, *padding_mode: str = 'zeros'*, *order: tuple = ('conv', 'norm', 'act')*)

A conv block that bundles conv/norm/activation layers.

This block simplifies the usage of convolution layers, which are commonly used with a norm layer (e.g., BatchNorm) and activation layer (e.g., ReLU). It is based upon three build methods: *build_conv_layer()*, *build_norm_layer()* and *build_activation_layer()*.

Besides, we add some additional features in this module. 1. Automatically set *bias* of the conv layer. 2. Spectral norm is supported. 3. More padding modes are supported. Before PyTorch 1.5, nn.Conv2d only supports zero and circular padding, and we add "reflect" padding mode.

参数

- **in_channels** (*int*) –Number of channels in the input feature map. Same as that in `nn._ConvNd`.
- **out_channels** (*int*) –Number of channels produced by the convolution. Same as that in `nn._ConvNd`.
- **kernel_size** (*int | tuple[int]*) –Size of the convolving kernel. Same as that in `nn._ConvNd`.
- **stride** (*int | tuple[int]*) –Stride of the convolution. Same as that in `nn._ConvNd`.
- **padding** (*int | tuple[int]*) –Zero-padding added to both sides of the input. Same as that in `nn._ConvNd`.
- **dilation** (*int | tuple[int]*) –Spacing between kernel elements. Same as that in `nn._ConvNd`.
- **groups** (*int*) –Number of blocked connections from input channels to output channels. Same as that in `nn._ConvNd`.
- **bias** (*bool | str*) –If specified as *auto*, it will be decided by the norm_cfg. Bias will be set as True if *norm_cfg* is None, otherwise False. Default: "auto".

- **conv_cfg** (*dict*) –Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: None.
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type=’ ReLU’).
- **inplace** (*bool*) –Whether to use inplace mode for activation. Default: True.
- **with_spectral_norm** (*bool*) –Whether use spectral norm in conv module. Default: False.
- **padding_mode** (*str*) –If the *padding_mode* has not been supported by current *Conv2d* in PyTorch, we will use our own padding layer instead. Currently, we support [‘zeros’ , ‘circular’] with official implementation and [‘reflect’] with our own implementation. Default: ‘zeros’ .
- **order** (*tuple[str]*) –The order of conv/norm/activation layers. It is a sequence of “conv” , “norm” and “act” . Common examples are (“conv” , “norm” , “act”) and (“act” , “conv” , “norm”). Default: (‘conv’ , ‘norm’ , ‘act’).

forward(*x: torch.Tensor*, *activate: bool = True*, *norm: bool = True*) → torch.Tensor

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 mmcv.cnn.**ConvTranspose2d**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, int]]*, *stride: Union[int, Tuple[int, int]] = 1*, *padding: Union[int, Tuple[int, int]] = 0*, *output_padding: Union[int, Tuple[int, int]] = 0*, *groups: int = 1*, *bias: bool = True*, *dilation: Union[int, Tuple[int, int]] = 1*, *padding_mode: str = 'zeros'*, *device=None*, *dtype=None*)

forward(*x: torch.Tensor*) → torch.Tensor

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 mmcv.cnn.**ConvTranspose3d**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, int, int]]*, *stride: Union[int, Tuple[int, int, int]] = 1*, *padding: Union[int, Tuple[int, int, int]] = 0*, *output_padding: Union[int, Tuple[int, int, int]] = 0*, *groups: int = 1*, *bias: bool = True*, *dilation: Union[int, Tuple[int, int, int]] = 1*, *padding_mode: str = 'zeros'*, *device=None*, *dtype=None*)

forward(*x: torch.Tensor*) → torch.Tensor

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 mmcv.cnn.**ConvWS2d**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, int]]*, *stride: Union[int, Tuple[int, int]] = 1*, *padding: Union[int, Tuple[int, int]] = 0*, *dilation: Union[int, Tuple[int, int]] = 1*, *groups: int = 1*, *bias: bool = True*, *eps: float = 1e-05*)

forward(*x: torch.Tensor*) → torch.Tensor

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 mmcv.cnn.**DepthwiseSeparableConvModule**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, int]]*, *stride: Union[int, Tuple[int, int]] = 1*, *padding: Union[int, Tuple[int, int]] = 0*, *dilation: Union[int, Tuple[int, int]] = 1*, *norm_cfg: Optional[Dict] = None*, *act_cfg: Dict = {'type': 'ReLU'}*, *dw_norm_cfg: Union[Dict, str] = 'default'*, *dw_act_cfg: Union[Dict, str] = 'default'*, *pw_norm_cfg: Union[Dict, str] = 'default'*, *pw_act_cfg: Union[Dict, str] = 'default'*, ***kwargs*)

Depthwise separable convolution module.

See https://arxiv.org/pdf/1704.04861.pdf for details.

This module can replace a ConvModule with the conv block replaced by two conv block: depthwise conv block and pointwise conv block. The depthwise conv block contains depthwise-conv/norm/activation layers. The pointwise conv block contains pointwise-conv/norm/activation layers. It should be noted that there will be norm/activation layer in the depthwise conv block if *norm_cfg* and *act_cfg* are specified.

参数

- **in_channels** (*int*) –Number of channels in the input feature map. Same as that in `nn._ConvNd`.
- **out_channels** (*int*) –Number of channels produced by the convolution. Same as that in `nn._ConvNd`.
- **kernel_size** (*int | tuple[int]*) –Size of the convolving kernel. Same as that in `nn._ConvNd`.
- **stride** (*int | tuple[int]*) –Stride of the convolution. Same as that in `nn._ConvNd`. Default: 1.
- **padding** (*int | tuple[int]*) –Zero-padding added to both sides of the input. Same as that in `nn._ConvNd`. Default: 0.
- **dilation** (*int | tuple[int]*) –Spacing between kernel elements. Same as that in `nn._ConvNd`. Default: 1.
- **norm_cfg** (*dict*) –Default norm config for both depthwise ConvModule and pointwise ConvModule. Default: None.
- **act_cfg** (*dict*) –Default activation config for both depthwise ConvModule and pointwise ConvModule. Default: dict(type='ReLU').
- **dw_norm_cfg** (*dict*) –Norm config of depthwise ConvModule. If it is 'default', it will be the same as *norm_cfg*. Default: 'default'.
- **dw_act_cfg** (*dict*) –Activation config of depthwise ConvModule. If it is 'default', it will be the same as *act_cfg*. Default: 'default'.
- **pw_norm_cfg** (*dict*) –Norm config of pointwise ConvModule. If it is 'default', it will be the same as *norm_cfg*. Default: 'default'.
- **pw_act_cfg** (*dict*) –Activation config of pointwise ConvModule. If it is 'default', it will be the same as *act_cfg*. Default: 'default'.
- **kwargs** (*optional*) –Other shared arguments for depthwise and pointwise ConvModule. See ConvModule for ref.

forward(*x: torch.Tensor*) → torch.Tensor

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 mmcv.cnn.**GeneralizedAttention**(*in_channels: int*, *spatial_range: int = - 1*, *num_heads: int = 9*, *position_embedding_dim: int = - 1*, *position_magnitude: int = 1*, *kv_stride: int = 2*, *q_stride: int = 1*, *attention_type: str = '1111'*)

GeneralizedAttention module.

See 'An Empirical Study of Spatial Attention Mechanisms in Deep Networks' (https://arxiv.org/abs/1711.07971) for details.

参数

- **in_channels** (*int*) –Channels of the input feature map.
- **spatial_range** (*int*) –The spatial range. -1 indicates no spatial range constraint. Default: -1.
- **num_heads** (*int*) –The head number of empirical_attention module. Default: 9.
- **position_embedding_dim** (*int*) –The position embedding dimension. Default: -1.
- **position_magnitude** (*int*) –A multiplier acting on coord difference. Default: 1.
- **kv_stride** (*int*) –The feature stride acting on key/value feature map. Default: 2.
- **q_stride** (*int*) –The feature stride acting on query feature map. Default: 1.
- **attention_type** (*str*) –A binary indicator string for indicating which items in generalized empirical_attention module are used. Default: '1111'.
  - '1000' indicates 'query and key content' (appr - appr) item,
  - '0100' indicates 'query content and relative position' (appr - position) item,
  - '0010' indicates 'key content only' (bias - appr) item,
  - '0001' indicates 'relative position only' (bias - position) item.

forward(*x_input: torch.Tensor*) → torch.Tensor

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 mmcv.cnn.**HSigmoid**(*bias: float = 3.0*, *divisor: float = 6.0*, *min_value: float = 0.0*, *max_value: float = 1.0*)

Hard Sigmoid Module. Apply the hard sigmoid function: Hsigmoid(x) = min(max((x + bias) / divisor, min_value), max_value) Default: Hsigmoid(x) = min(max((x + 3) / 6, 0), 1)

注解: In MMCV v1.4.4, we modified the default value of args to align with PyTorch official.

参数

- **bias** (*float*) –Bias of the input feature map. Default: 3.0.
- **divisor** (*float*) –Divisor of the input feature map. Default: 6.0.
- **min_value** (*float*) –Lower bound value. Default: 0.0.
- **max_value** (*float*) –Upper bound value. Default: 1.0.

返回 The output tensor.

返回类型 Tensor

forward(*x: torch.Tensor*) → torch.Tensor

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 mmcv.cnn.**HSwish**(*inplace: bool = False*)

Hard Swish Module.

This module applies the hard swish function:

$$Hswish(x) = x * ReLU6(x + 3)/6$$

参数 **inplace** (*bool*) –can optionally do the operation in-place. Default: False.

返回 The output tensor.

返回类型 Tensor

forward(*x: torch.Tensor*) → torch.Tensor

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 mmcv.cnn.**KaimingInit**(*a: float = 0*, *mode: str = 'fan_out'*, *nonlinearity: str = 'relu'*, *distribution: str = 'normal'*, ***kwargs*)

Initialize module parameters with the values according to the method described in 'Delving deep into rectifiers: Surpassing human-level.

performance on ImageNet classification - He, K. et al. (2015). <https://www.cv-foundation.org/openaccess/content_iccv_2015/ papers/He_Delving_Deep_into_ICCV_2015_paper.pdf>'_

参数

- **a** (*int | float*) –the negative slope of the rectifier used after this layer (only used with `'leaky_relu'`). Defaults to 0.
- **mode** (*str*) –either `'fan_in'` or `'fan_out'`. Choosing `'fan_in'` preserves the magnitude of the variance of the weights in the forward pass. Choosing `'fan_out'` preserves the magnitudes in the backwards pass. Defaults to `'fan_out'`.
- **nonlinearity** (*str*) –the non-linear function (*nn.functional* name), recommended to use only with `'relu'` or `'leaky_relu'` . Defaults to 'relu'.
- **bias** (*int | float*) –the value to fill the bias. Defaults to 0.
- **bias_prob** (*float, optional*) –the probability for bias initialization. Defaults to None.
- **distribution** (*str*) –distribution either be `'normal'` or `'uniform'`. Defaults to `'normal'`.
- **layer** (*str | list[str], optional*) –the layer will be initialized. Defaults to None.

class mmcv.cnn.**Linear**(*in_features: int*, *out_features: int*, *bias: bool = True*, *device=None*, *dtype=None*)

forward(*x: torch.Tensor*) → torch.Tensor

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 mmcv.cnn.**MaxPool2d**(*kernel_size: Union[int, Tuple[int, ...]]*, *stride: Optional[Union[int, Tuple[int, ...]]] = None*, *padding: Union[int, Tuple[int, ...]] = 0*, *dilation: Union[int, Tuple[int, ...]] = 1*, *return_indices: bool = False*, *ceil_mode: bool = False*)

forward(*x: torch.Tensor*) → torch.Tensor

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 mmcv.cnn.**MaxPool3d**(*kernel_size: Union[int, Tuple[int, ...]]*, *stride: Optional[Union[int, Tuple[int, ...]]] = None*, *padding: Union[int, Tuple[int, ...]] = 0*, *dilation: Union[int, Tuple[int, ...]] = 1*, *return_indices: bool = False*, *ceil_mode: bool = False*)

forward(*x: torch.Tensor*) → torch.Tensor

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 mmcv.cnn.**NonLocal1d**(*in_channels: int*, *sub_sample: bool = False*, *conv_cfg: Dict = {'type': 'Conv1d'}*, ***kwargs*)

1D Non-local module.

参数

- **in_channels** (*int*) –Same as *NonLocalND*.
- **sub_sample** (*bool*) –Whether to apply max pooling after pairwise function (Note that the *sub_sample* is applied on spatial only). Default: False.
- **conv_cfg** (*None | dict*) –Same as *NonLocalND*. Default: dict(type='Conv1d').

class mmcv.cnn.**NonLocal2d**(*in_channels: int*, *sub_sample: bool = False*, *conv_cfg: Dict = {'type': 'Conv2d'}*, ***kwargs*)

2D Non-local module.

参数

- **in_channels** (*int*) –Same as *NonLocalND*.
- **sub_sample** (*bool*) –Whether to apply max pooling after pairwise function (Note that the *sub_sample* is applied on spatial only). Default: False.
- **conv_cfg** (*None | dict*) –Same as *NonLocalND*. Default: dict(type='Conv2d').

class mmcv.cnn.**NonLocal3d**(*in_channels: int*, *sub_sample: bool = False*, *conv_cfg: Dict = {'type': 'Conv3d'}*, ***kwargs*)

3D Non-local module.

参数

- **in_channels** (*int*) –Same as *NonLocalND*.
- **sub_sample** (*bool*) –Whether to apply max pooling after pairwise function (Note that the *sub_sample* is applied on spatial only). Default: False.
- **conv_cfg** (*None | dict*) –Same as *NonLocalND*. Default: dict(type='Conv3d').

class mmcv.cnn.**NormalInit**(*mean: float = 0*, *std: float = 1*, ***kwargs*)

Initialize module parameters with the values drawn from the normal distribution $\mathcal{N}(\text{mean}, \text{std}^2)$.

参数

- **mean** (*int | float*) –the mean of the normal distribution. Defaults to 0.
- **std** (*int | float*) –the standard deviation of the normal distribution. Defaults to 1.
- **bias** (*int | float*) –the value to fill the bias. Defaults to 0.
- **bias_prob** (*float, optional*) –the probability for bias initialization. Defaults to None.
- **layer** (*str | list[str], optional*) –the layer will be initialized. Defaults to None.

class mmcv.cnn.**PretrainedInit**(*checkpoint: str*, *prefix: Optional[str] = None*, *map_location: Optional[str] = None*)

Initialize module by loading a pretrained model.

参数

- **checkpoint** (*str*) –the checkpoint file of the pretrained model should be load.
- **prefix** (*str, optional*) –the prefix of a sub-module in the pretrained model. it is for loading a part of the pretrained model to initialize. For example, if we would like to only load the backbone of a detector model, we can set `prefix='backbone.'`. Defaults to None.
- **map_location** (*str*) –map tensors into proper locations.

class mmcv.cnn.**ResNet**(*depth: int*, *num_stages: int = 4*, *strides: Sequence[int] = (1, 2, 2, 2)*, *dilations: Sequence[int] = (1, 1, 1, 1)*, *out_indices: Sequence[int] = (0, 1, 2, 3)*, *style: str = 'pytorch'*, *frozen_stages: int = - 1*, *bn_eval: bool = True*, *bn_frozen: bool = False*, *with_cp: bool = False*)

ResNet backbone.

参数

- **depth** (*int*) –Depth of resnet, from {18, 34, 50, 101, 152}.
- **num_stages** (*int*) –Resnet stages, normally 4.
- **strides** (*Sequence[int]*) –Strides of the first block of each stage.
- **dilations** (*Sequence[int]*) –Dilation of each stage.
- **out_indices** (*Sequence[int]*) –Output from which stages.
- **style** (*str*) –*pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer.
- **frozen_stages** (*int*) –Stages to be frozen (all param fixed). -1 means not freezing any parameters.
- **bn_eval** (*bool*) –Whether to set BN layers as eval mode, namely, freeze running stats (mean and var).
- **bn_frozen** (*bool*) –Whether to freeze weight and bias of BN layers.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed.

forward(*x: torch.Tensor*) → Union[torch.Tensor, Tuple[torch.Tensor]]

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: bool = True*) → None

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 mmcv.cnn.**Scale**(*scale: float = 1.0*)

A learnable scale parameter.

This layer scales the input by a learnable factor. It multiplies a learnable scale parameter of shape (1,) with input of any shape.

参数 **scale** (*float*) –Initial value of scale factor. Default: 1.0

forward(*x: torch.Tensor*) → torch.Tensor

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 mmcv.cnn.**Swish**

Swish Module.

This module applies the swish function:

$$Swish(x) = x * Sigmoid(x)$$

返回 The output tensor.

返回类型 Tensor

forward(*x: torch.Tensor*) → torch.Tensor

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 mmcv.cnn.**TruncNormalInit**(*mean: float = 0*, *std: float = 1*, *a: float = - 2*, *b: float = 2*, ***kwargs*)

Initialize module parameters with the values drawn from the normal distribution $\mathcal{N}(\text{mean}, \text{std}^2)$ with values outside $[a, b]$.

参数

- **mean** (*float*) –the mean of the normal distribution. Defaults to 0.
- **std** (*float*) –the standard deviation of the normal distribution. Defaults to 1.

- **a** (*float*) –The minimum cutoff value.
- **b** (*float*) –The maximum cutoff value.
- **bias** (*float*) –the value to fill the bias. Defaults to 0.
- **bias_prob** (*float, optional*) –the probability for bias initialization. Defaults to None.
- **layer** (*str | list[str], optional*) –the layer will be initialized. Defaults to None.

class mmcv.cnn.**UniformInit**(*a: float = 0.0*, *b: float = 1.0*, ***kwargs*)

Initialize module parameters with values drawn from the uniform distribution $\mathcal{U}(a, b)$.

参数

- **a** (*int | float*) –the lower bound of the uniform distribution. Defaults to 0.
- **b** (*int | float*) –the upper bound of the uniform distribution. Defaults to 1.
- **bias** (*int | float*) –the value to fill the bias. Defaults to 0.
- **bias_prob** (*float, optional*) –the probability for bias initialization. Defaults to None.
- **layer** (*str | list[str], optional*) –the layer will be initialized. Defaults to None.

class mmcv.cnn.**VGG**(*depth: int*, *with_bn: bool = False*, *num_classes: int = - 1*, *num_stages: int = 5*, *dilations: Sequence[int] = (1, 1, 1, 1, 1)*, *out_indices: Sequence[int] = (0, 1, 2, 3, 4)*, *frozen_stages: int = - 1*, *bn_eval: bool = True*, *bn_frozen: bool = False*, *ceil_mode: bool = False*, *with_last_pool: bool = True*)

VGG backbone.

参数

- **depth** (*int*) –Depth of vgg, from {11, 13, 16, 19}.
- **with_bn** (*bool*) –Use BatchNorm or not.
- **num_classes** (*int*) –number of classes for classification.
- **num_stages** (*int*) –VGG stages, normally 5.
- **dilations** (*Sequence[int]*) –Dilation of each stage.
- **out_indices** (*Sequence[int]*) –Output from which stages.
- **frozen_stages** (*int*) –Stages to be frozen (all param fixed). -1 means not freezing any parameters.
- **bn_eval** (*bool*) –Whether to set BN layers as eval mode, namely, freeze running stats (mean and var).

- **bn_frozen** (*bool*) –Whether to freeze weight and bias of BN layers.

forward (*x: torch.Tensor*) → Union[torch.Tensor, Tuple[torch.Tensor, ...]]

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: bool = True*) → None

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 mmcv.cnn.**XavierInit** (*gain: float = 1*, *distribution: str = 'normal'*, ***kwargs*)

Initialize module parameters with values according to the method described in ‘Understanding the difficulty of training deep feedforward.

neural networks - Glorot, X. & Bengio, Y. (2010). <http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf>‘_

参数

- **gain** (*int | float*) –an optional scaling factor. Defaults to 1.
- **bias** (*int | float*) –the value to fill the bias. Defaults to 0.
- **bias_prob** (*float, optional*) –the probability for bias initialization. Defaults to None.
- **distribution** (*str*) –distribution either be `'normal'` or `'uniform'`. Defaults to `'normal'`.
- **layer** (*str | list[str], optional*) –the layer will be initialized. Defaults to None.

mmcv.cnn.**bias_init_with_prob** (*prior_prob: float*) → float

initialize conv/fc bias value according to a given probability value.

mmcv.cnn.**build_activation_layer** (*cfg: Dict*) → torch.nn.modules.module.Module

Build activation layer.

参数 **cfg** (*dict*) –The activation layer config, which should contain:

- type (str): Layer type.
- layer args: Args needed to instantiate an activation layer.

返回 Created activation layer.

返回类型 nn.Module

mmcv.cnn.**build_conv_layer**(*cfg: Optional[Dict]*, **args*, ***kwargs*) → torch.nn.modules.module.Module

Build convolution layer.

参数

- **cfg** (*None or dict*) –The conv layer config, which should contain: - type (str): Layer type. - layer args: Args needed to instantiate an conv layer.
- **args** (*argument list*) –Arguments passed to the *__init__* method of the corresponding conv layer.
- **kwargs** (*keyword arguments*) –Keyword arguments passed to the *__init__* method of the corresponding conv layer.

返回 Created conv layer.

返回类型 nn.Module

mmcv.cnn.**build_model_from_cfg**(*cfg*, *registry*, *default_args=None*)

Build a PyTorch model from config dict(s). Different from `build_from_cfg`, if cfg is a list, a `nn.Sequential` will be built.

参数

- **cfg** (*dict, list[dict]*) –The config of modules, is is either a config dict or a list of config dicts. If cfg is a list, a the built modules will be wrapped with `nn.Sequential`.
- **registry** (`Registry`) –A registry the module belongs to.
- **default_args** (*dict, optional*) –Default arguments to build the module. Defaults to None.

返回 A built nn module.

返回类型 nn.Module

mmcv.cnn.**build_norm_layer**(*cfg: Dict*, *num_features: int*, *postfix: Union[int, str] = ''*) → Tuple[str, torch.nn.modules.module.Module]

Build normalization layer.

参数

- **cfg** (*dict*) –The norm layer config, which should contain:
  - type (str): Layer type.

- layer args: Args needed to instantiate a norm layer.
- requires_grad (bool, optional): Whether stop gradient updates.

- **num_features** (*int*) –Number of input channels.
- **postfix** (*int | str*) –The postfix to be appended into norm abbreviation to create named layer.

返回 The first element is the layer name consisting of abbreviation and postfix, e.g., bn1, gn. The second element is the created norm layer.

返回类型 tuple[str, nn.Module]

mmcv.cnn.**build_padding_layer**(*cfg: Dict*, **args*, ***kwargs*) → torch.nn.modules.module.Module

Build padding layer.

参数 **cfg** (*dict*) –The padding layer config, which should contain: - type (str): Layer type. - layer args: Args needed to instantiate a padding layer.

返回 Created padding layer.

返回类型 nn.Module

mmcv.cnn.**build_plugin_layer**(*cfg: Dict*, *postfix: Union[int, str] = ''*, ***kwargs*) → Tuple[str, torch.nn.modules.module.Module]

Build plugin layer.

参数

- **cfg** (*dict*) –cfg should contain:
  - type (str): identify plugin layer type.
  - layer args: args needed to instantiate a plugin layer.
- **postfix** (*int, str*) –appended into norm abbreviation to create named layer. Default: ''.

返回 The first one is the concatenation of abbreviation and postfix. The second is the created plugin layer.

返回类型 tuple[str, nn.Module]

mmcv.cnn.**build_upsample_layer**(*cfg: Dict*, **args*, ***kwargs*) → torch.nn.modules.module.Module

Build upsample layer.

参数

- **cfg** (*dict*) –The upsample layer config, which should contain:
  - type (str): Layer type.
  - scale_factor (int): Upsample ratio, which is not applicable to deconv.
  - layer args: Args needed to instantiate a upsample layer.

- **args** (*argument list*) –Arguments passed to the __init__ method of the corresponding conv layer.
- **kwargs** (*keyword arguments*) –Keyword arguments passed to the __init__ method of the corresponding conv layer.

返回 Created upsample layer.

返回类型 nn.Module

mmcv.cnn.**fuse_conv_bn**(*module: torch.nn.modules.module.Module*) → torch.nn.modules.module.Module

Recursively fuse conv and bn in a module.

During inference, the functionary of batch norm layers is turned off but only the mean and var alone channels are used, which exposes the chance to fuse it with the preceding conv layers to save computations and simplify network structures.

参数 **module** (*nn.Module*) –Module to be fused.

返回 Fused module.

返回类型 nn.Module

mmcv.cnn.**get_model_complexity_info**(*model: torch.nn.modules.module.Module*, *input_shape: tuple*, *print_per_layer_stat: bool = True*, *as_strings: bool = True*, *input_constructor: Optional[Callable] = None*, *flush: bool = False*, *ost: TextIO = <_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*) → tuple

Get complexity information of a model.

This method can calculate FLOPs and parameter counts of a model with corresponding input shape. It can also print complexity information for each layer in a model.

Supported layers are listed as below:

- Convolutions: nn.Conv1d, nn.Conv2d, nn.Conv3d.
- Activations: nn.ReLU, nn.PReLU, nn.ELU, nn.LeakyReLU, nn.ReLU6.
- Poolings: nn.MaxPool1d, nn.MaxPool2d, nn.MaxPool3d, nn.AvgPool1d, nn.AvgPool2d, nn.AvgPool3d, nn.AdaptiveMaxPool1d, nn.AdaptiveMaxPool2d, nn.AdaptiveMaxPool3d, nn.AdaptiveAvgPool1d, nn.AdaptiveAvgPool2d, nn.AdaptiveAvgPool3d.
- BatchNorms: nn.BatchNorm1d, nn.BatchNorm2d, nn.BatchNorm3d, nn.GroupNorm, nn.InstanceNorm1d, InstanceNorm2d, InstanceNorm3d, nn.LayerNorm.
- Linear: nn.Linear.
- Deconvolution: nn.ConvTranspose2d.
- Upsample: nn.Upsample.

参数

- **model** (*nn.Module*) –The model for complexity calculation.
- **input_shape** (*tuple*) –Input shape used for calculation.
- **print_per_layer_stat** (*bool*) –Whether to print complexity information for each layer in a model. Default: True.
- **as_strings** (*bool*) –Output FLOPs and params counts in a string form. Default: True.
- **input_constructor** (*None | callable*) –If specified, it takes a callable method that generates input. otherwise, it will generate a random tensor with input shape to calculate FLOPs. Default: None.
- **flush** (*bool*) –same as that in `print()`. Default: False.
- **ost** (*stream*) –same as `file` param in `print()`. Default: sys.stdout.

返回 If `as_strings` is set to True, it will return FLOPs and parameter counts in a string format. otherwise, it will return those in a float number format.

返回类型 tuple[float | str]

mmcv.cnn.**initialize**(*module: torch.nn.modules.module.Module*, *init_cfg: Union[Dict, List[dict]]*) → None

Initialize a module.

参数

- **module** (`torch.nn.Module`) –the module will be initialized.
- **init_cfg** (*dict | list[dict]*) –initialization configuration dict to define initializer. OpenMMLab has implemented 6 initializers including `Constant`, `Xavier`, `Normal`, `Uniform`, `Kaiming`, and `Pretrained`.

示例

```
>>> module = nn.Linear(2, 3, bias=True)
>>> init_cfg = dict(type='Constant', layer='Linear', val =1 , bias =2)
>>> initialize(module, init_cfg)
```

```
>>> module = nn.Sequential(nn.Conv1d(3, 1, 3), nn.Linear(1,2))
>>> # define key ``'layer'`` for initializing layer with different
>>> # configuration
>>> init_cfg = [dict(type='Constant', layer='Conv1d', val=1),
        dict(type='Constant', layer='Linear', val=2)]
>>> initialize(module, init_cfg)
```

```
>>> # define key``'override'`` to initialize some specific part in
>>> # module
>>> class FooNet(nn.Module):
>>>     def __init__(self):
>>>         super().__init__()
>>>         self.feat = nn.Conv2d(3, 16, 3)
>>>         self.reg = nn.Conv2d(16, 10, 3)
>>>         self.cls = nn.Conv2d(16, 5, 3)
>>> model = FooNet()
>>> init_cfg = dict(type='Constant', val=1, bias=2, layer='Conv2d',
>>>     override=dict(type='Constant', name='reg', val=3, bias=4))
>>> initialize(model, init_cfg)
```

```
>>> model = ResNet(depth=50)
>>> # Initialize weights with the pretrained model.
>>> init_cfg = dict(type='Pretrained',
        checkpoint='torchvision://resnet50')
>>> initialize(model, init_cfg)
```

```
>>> # Initialize weights of a sub-module with the specific part of
>>> # a pretrained model by using "prefix".
>>> url = 'http://download.openmmlab.com/mmdetection/v2.0/retinanet/'\
>>>     'retinanet_r50_fpn_1x_coco/'\
>>>     'retinanet_r50_fpn_1x_coco_20200130-c2398f9e.pth'
>>> init_cfg = dict(type='Pretrained',
        checkpoint=url, prefix='backbone.')
```

mmcv.cnn.**is_norm**(*layer: torch.nn.modules.module.Module*, *exclude: Optional[Union[type, tuple]] = None*) → bool

Check if a layer is a normalization layer.

参数

- **layer** (*nn.Module*) –The layer to be checked.
- **exclude** (*type | tuple[type]*) –Types to be excluded.

返回 Whether the layer is a norm layer.

返回类型 bool

CHAPTER 33

runner

class mmcv.runner.**BaseModule**(*init_cfg: Optional[dict] = None*)

Base module for all modules in openmmlab.

`BaseModule` is a wrapper of `torch.nn.Module` with additional functionality of parameter initialization. Compared with `torch.nn.Module`, `BaseModule` mainly adds three attributes.

- `init_cfg`: the config to control the initialization.
- `init_weights`: The function of parameter initialization and recording initialization information.
- `_params_init_info`: Used to track the parameter initialization information. This attribute only exists during executing the `init_weights`.

参数 **init_cfg**(*dict, optional*) –Initialization config dict.

init_weights() → None

Initialize the weights.

class mmcv.runner.**BaseRunner**(*model: torch.nn.modules.module.Module*, *batch_processor: Optional[Callable] = None*, *optimizer: Optional[Union[Dict, torch.optim.optimizer.Optimizer]] = None*, *work_dir: Optional[str] = None*, *logger: Optional[logging.Logger] = None*, *meta: Optional[Dict] = None*, *max_iters: Optional[int] = None*, *max_epochs: Optional[int] = None*)

The base class of Runner, a training helper for PyTorch.

All subclasses should implement the following APIs:

- `run()`

- `train()`
- `val()`
- `save_checkpoint()`

参数

- **model** (`torch.nn.Module`) –The model to be run.
- **batch_processor** (*callable*) –A callable method that process a data batch. The interface of this method should be *batch_processor(model, data, train_mode) -> dict*
- **optimizer** (dict or `torch.optim.Optimizer`) –It can be either an optimizer (in most cases) or a dict of optimizers (in models that requires more than one optimizer, e.g., GAN).
- **work_dir** (*str, optional*) –The working directory to save checkpoints and logs. Defaults to None.
- **logger** (`logging.Logger`) –Logger used during training. Defaults to None. (The default value is just for backward compatibility)
- **meta** (*dict | None*) –A dict records some import information such as environment info and seed, which will be logged in logger hook. Defaults to None.
- **max_epochs** (*int, optional*) –Total training epochs.
- **max_iters** (*int, optional*) –Total training iterations.

call_hook(*fn_name: str*) → None

Call all hooks.

参数 **fn_name** (*str*) –The function name in each hook to be called, such as "before_train_epoch".

current_lr() → Union[List[float], Dict[str, List[float]]]

Get current learning rates.

返回 Current learning rates of all param groups. If the runner has a dict of optimizers, this method will return a dict.

返回类型 list[float] | dict[str, list[float]]

current_momentum() → Union[List[float], Dict[str, List[float]]]

Get current momentums.

返回 Current momentums of all param groups. If the runner has a dict of optimizers, this method will return a dict.

返回类型 list[float] | dict[str, list[float]]

property epoch: int

Current epoch.

Type int

property hooks: List[mmcv.runner.hooks.hook.Hook]

A list of registered hooks.

Type list[Hook]

property inner_iter: int

Iteration in an epoch.

Type int

property iter: int

Current iteration.

Type int

property max_epochs

Maximum training epochs.

Type int

property max_iters

Maximum training iterations.

Type int

property model_name: str

Name of the model, usually the module class name.

Type str

property rank: int

Rank of current process. (distributed training)

Type int

register_hook(*hook: mmcv.runner.hooks.hook.Hook*, *priority: Union[int, str, mmcv.runner.priority.Priority] = 'NORMAL'*) → None

Register a hook into the hook list.

The hook will be inserted into a priority queue, with the specified priority (See *Priority* for details of priorities). For hooks with the same priority, they will be triggered in the same order as they are registered.

参数

- **hook** (Hook) –The hook to be registered.
- **priority** (int or str or *Priority*) –Hook priority. Lower value means higher priority.

register_hook_from_cfg(*hook_cfg: Dict*) → None

Register a hook from its cfg.

参数 **hook_cfg** (*dict*) –Hook config. It should have at least keys 'type' and 'priority' indicating its type and priority.

注解: The specific hook class to register should not use 'type' and 'priority' arguments during initialization.

---

register_training_hooks(*lr_config: Optional[Union[Dict, mmcv.runner.hooks.hook.Hook]]*, *optimizer_config: Optional[Union[Dict, mmcv.runner.hooks.hook.Hook]] = None*, *checkpoint_config: Optional[Union[Dict, mmcv.runner.hooks.hook.Hook]] = None*, *log_config: Optional[Dict] = None*, *momentum_config: Optional[Union[Dict, mmcv.runner.hooks.hook.Hook]] = None*, *timer_config: Union[Dict, mmcv.runner.hooks.hook.Hook] = {'type': 'IterTimerHook'}*, *custom_hooks_config: Optional[Union[List, Dict, mmcv.runner.hooks.hook.Hook]] = None*) → None

Register default and custom hooks for training.

Default and custom hooks include:

Hooks	Priority
LrUpdaterHook	VERY_HIGH (10)
MomentumUpdaterHook	HIGH (30)
OptimizerStepperHook	ABOVE_NORMAL (40)
CheckpointSaverHook	NORMAL (50)
IterTimerHook	LOW (70)
LoggerHook(s)	VERY_LOW (90)
CustomHook(s)	defaults to NORMAL (50)

If custom hooks have same priority with default hooks, custom hooks will be triggered after default hooks.

property world_size: int

Number of processes participating in the job. (distributed training)

Type int

class mmcv.runner.**CheckpointHook**(*interval: int = - 1*, *by_epoch: bool = True*, *save_optimizer: bool = True*, *out_dir: Optional[str] = None*, *max_keep_ckpts: int = - 1*, *save_last: bool = True*, *sync_buffer: bool = False*, *file_client_args: Optional[dict] = None*, ***kwargs*)

Save checkpoints periodically.

参数

- **interval** (*int*) –The saving period. If `by_epoch=True`, interval indicates epochs, otherwise it indicates iterations. Default: -1, which means "never".

- **by_epoch** (*bool*) –Saving checkpoints by epoch or by iteration. Default: True.
- **save_optimizer** (*bool*) –Whether to save optimizer state_dict in the checkpoint. It is usually used for resuming experiments. Default: True.
- **out_dir** (*str, optional*) –The root directory to save checkpoints. If not specified, `runner.work_dir` will be used by default. If specified, the `out_dir` will be the concatenation of `out_dir` and the last level directory of `runner.work_dir`. *Changed in version 1.3.16.*
- **max_keep_ckpts** (*int, optional*) –The maximum checkpoints to keep. In some cases we want only the latest few checkpoints and would like to delete old ones to save the disk space. Default: -1, which means unlimited.
- **save_last** (*bool, optional*) –Whether to force the last checkpoint to be saved regardless of interval. Default: True.
- **sync_buffer** (*bool, optional*) –Whether to synchronize buffers in different gpus. Default: False.
- **file_client_args** (*dict, optional*) –Arguments to instantiate a FileClient. See *mmcv.fileio.FileClient* for details. Default: None. *New in version 1.3.16.*

警告: Before v1.3.16, the `out_dir` argument indicates the path where the checkpoint is stored. However, since v1.3.16, `out_dir` indicates the root directory and the final path to save checkpoint is the concatenation of `out_dir` and the last level directory of `runner.work_dir`. Suppose the value of `out_dir` is "/path/of/A"and the value of `runner.work_dir` is "/path/of/B", then the final path will be "/path/of/A/B" .

class mmcv.runner.**CheckpointLoader**

A general checkpoint loader to manage all schemes.

classmethod load_checkpoint(*filename: str*, *map_location: Optional[Union[str, Callable]] = None*, *logger: Optional[logging.Logger] = None*) → Union[dict, collections.OrderedDict]

load checkpoint through URL scheme path.

参数

- **filename** (*str*) –checkpoint file name with given prefix
- **map_location** (*str, optional*) –Same as `torch.load()`. Default: None
- **logger** (`logging.Logger`, optional) –The logger for message. Default: None

返回 The loaded checkpoint.

返回类型 dict or OrderedDict

classmethod register_scheme(*prefixes: Union[str, List[str], Tuple[str, ...]]*, *loader: Optional[Callable] = None*, *force: bool = False*) → Callable

Register a loader to CheckpointLoader.

This method can be used as a normal class method or a decorator.

参数

- **prefixes** (*str or Sequence[str]*) –
- **prefix of the registered loader.** (*The*) –
- **loader** (*function, optional*) –The loader function to be registered. When this method is used as a decorator, loader is None. Defaults to None.
- **force** (*bool, optional*) –Whether to override the loader if the prefix has already been registered. Defaults to False.

class mmcv.runner.**ClearMLLoggerHook**(*init_kwargs: Optional[Dict] = None*, *interval: int = 10*, *ignore_last: bool = True*, *reset_flag: bool = False*, *by_epoch: bool = True*)

Class to log metrics with clearml.

It requires clearml to be installed.

参数

- **init_kwargs** (*dict*) –A dict contains the *clearml.Task.init* initialization keys. See taskinit for more details.
- **interval** (*int*) –Logging interval (every k iterations). Default 10.
- **ignore_last** (*bool*) –Ignore the log of last iterations in each epoch if less than *interval*. Default: True.
- **reset_flag** (*bool*) –Whether to clear the output buffer after logging. Default: False.
- **by_epoch** (*bool*) –Whether EpochBasedRunner is used. Default: True.

class mmcv.runner.**CosineAnnealingLrUpdaterHook**(*min_lr: Optional[float] = None*, *min_lr_ratio: Optional[float] = None*, ***kwargs*)

CosineAnnealing LR scheduler.

参数

- **min_lr** (*float, optional*) –The minimum lr. Default: None.
- **min_lr_ratio** (*float, optional*) –The ratio of minimum lr to the base lr. Either *min_lr* or *min_lr_ratio* should be specified. Default: None.

class mmcv.runner.**CosineAnnealingMomentumUpdaterHook**(*min_momentum: Optional[float] = None*, *min_momentum_ratio: Optional[float] = None*, ***kwargs*)

Cosine annealing LR Momentum decays the Momentum of each parameter group linearly.

参数

- **min_momentum** (*float, optional*) –The minimum momentum. Default: None.
- **min_momentum_ratio** (*float, optional*) –The ratio of minimum momentum to the base momentum. Either *min_momentum* or *min_momentum_ratio* should be specified. Default: None.

class mmcv.runner.**CosineRestartLrUpdaterHook**(*periods: List[int]*, *restart_weights: List[float] = [1]*, *min_lr: Optional[float] = None*, *min_lr_ratio: Optional[float] = None*, ***kwargs*)

Cosine annealing with restarts learning rate scheme.

参数

- **periods** (*list[int]*) –Periods for each cosine anneling cycle.
- **restart_weights** (*list[float]*) –Restart weights at each restart iteration. Defaults to [1].
- **min_lr** (*float, optional*) –The minimum lr. Default: None.
- **min_lr_ratio** (*float, optional*) –The ratio of minimum lr to the base lr. Either *min_lr* or *min_lr_ratio* should be specified. Default: None.

class mmcv.runner.**CyclicLrUpdaterHook**(*by_epoch: bool = False*, *target_ratio: Union[float, tuple] = (10, 0.0001)*, *cyclic_times: int = 1*, *step_ratio_up: float = 0.4*, *anneal_strategy: str = 'cos'*, *gamma: float = 1*, ***kwargs*)

Cyclic LR Scheduler.

Implement the cyclical learning rate policy (CLR) described in https://arxiv.org/pdf/1506.01186.pdf

Different from the original paper, we use cosine annealing rather than triangular policy inside a cycle. This improves the performance in the 3D detection area.

参数

- **by_epoch** (*bool, optional*) –Whether to update LR by epoch.
- **target_ratio** (*tuple[float], optional*) –Relative ratio of the highest LR and the lowest LR to the initial LR.
- **cyclic_times** (*int, optional*) –Number of cycles during training
- **step_ratio_up** (*float, optional*) –The ratio of the increasing process of LR in the total cycle.

- **anneal_strategy** (*str, optional*) –{ ‘cos’, ‘linear’ } Specifies the annealing strategy: ‘cos’ for cosine annealing, ‘linear’ for linear annealing. Default: ‘cos’.
- **gamma** (*float, optional*) –Cycle decay ratio. Default: 1. It takes values in the range (0, 1]. The difference between the maximum learning rate and the minimum learning rate decreases periodically when it is less than 1. *New in version 1.4.4.*

class mmcv.runner.**CyclicMomentumUpdaterHook**(*by_epoch: bool = False*, *target_ratio: Tuple[float, float] = (0.8947368421052632, 1.0)*, *cyclic_times: int = 1*, *step_ratio_up: float = 0.4*, *anneal_strategy: str = 'cos'*, *gamma: float = 1.0*, ***kwargs*)

Cyclic momentum Scheduler.

Implement the cyclical momentum scheduler policy described in https://arxiv.org/pdf/1708.07120.pdf

This momentum scheduler usually used together with the CyclicLRUpdater to improve the performance in the 3D detection area.

参数

- **target_ratio** (*tuple[float]*) –Relative ratio of the lowest momentum and the highest momentum to the initial momentum.
- **cyclic_times** (*int*) –Number of cycles during training
- **step_ratio_up** (*float*) –The ratio of the increasing process of momentum in the total cycle.
- **by_epoch** (*bool*) –Whether to update momentum by epoch.
- **anneal_strategy** (*str, optional*) –{ ‘cos’, ‘linear’ } Specifies the annealing strategy: ‘cos’ for cosine annealing, ‘linear’ for linear annealing. Default: ‘cos’.
- **gamma** (*float, optional*) –Cycle decay ratio. Default: 1. It takes values in the range (0, 1]. The difference between the maximum learning rate and the minimum learning rate decreases periodically when it is less than 1. *New in version 1.4.4.*

class mmcv.runner.**DefaultOptimizerConstructor**(*optimizer_cfg: Dict*, *paramwise_cfg: Optional[Dict] = None*)

Default constructor for optimizers.

By default each parameter share the same optimizer settings, and we provide an argument `paramwise_cfg` to specify parameter-wise settings. It is a dict and may contain the following fields:

- `custom_keys` (dict): Specified parameters-wise settings by keys. If one of the keys in `custom_keys` is a substring of the name of one parameter, then the setting of the parameter will be specified by `custom_keys[key]` and other setting like `bias_lr_mult` etc. will be ignored. It should be noted that the aforementioned `key` is the longest key that is a substring of the name of the parameter. If there are multiple matched keys with the same length, then the key with lower alphabet order will be chosen.

`custom_keys[key]` should be a dict and may contain fields `lr_mult` and `decay_mult`. See Example 2 below.

- `bias_lr_mult` (float): It will be multiplied to the learning rate for all bias parameters (except for those in normalization layers and offset layers of DCN).
- `bias_decay_mult` (float): It will be multiplied to the weight decay for all bias parameters (except for those in normalization layers, depthwise conv layers, offset layers of DCN).
- `norm_decay_mult` (float): It will be multiplied to the weight decay for all weight and bias parameters of normalization layers.
- `dwconv_decay_mult` (float): It will be multiplied to the weight decay for all weight and bias parameters of depthwise conv layers.
- `dcn_offset_lr_mult` (float): It will be multiplied to the learning rate for parameters of offset layer in the deformable convs of a model.
- `bypass_duplicate` (bool): If true, the duplicate parameters would not be added into optimizer. Default: False.

注解: 1. If the option `dcn_offset_lr_mult` is used, the constructor will override the effect of `bias_lr_mult` in the bias of offset layer. So be careful when using both `bias_lr_mult` and `dcn_offset_lr_mult`. If you wish to apply both of them to the offset layer in deformable convs, set `dcn_offset_lr_mult` to the original `dcn_offset_lr_mult` * `bias_lr_mult`.

2. If the option `dcn_offset_lr_mult` is used, the constructor will apply it to all the DCN layers in the model. So be careful when the model contains multiple DCN layers in places other than backbone.

参数

- **model** (`nn.Module`) –The model with parameters to be optimized.
- **optimizer_cfg** (*dict*) –The config dict of the optimizer. Positional fields are
  - *type*: class name of the optimizer.

  Optional fields are
  - any arguments of the corresponding optimizer type, e.g., lr, weight_decay, momentum, etc.
- **paramwise_cfg** (*dict, optional*) –Parameter-wise options.

Example 1:

```
>>> model = torch.nn.modules.Conv1d(1, 1, 1)
>>> optimizer_cfg = dict(type='SGD', lr=0.01, momentum=0.9,
>>>                      weight_decay=0.0001)
>>> paramwise_cfg = dict(norm_decay_mult=0.)
```

(下页继续)

(续上页)

```
>>> optim_builder = DefaultOptimizerConstructor(
>>>     optimizer_cfg, paramwise_cfg)
>>> optimizer = optim_builder(model)
```

Example 2:

```
>>> # assume model have attribute model.backbone and model.cls_head
>>> optimizer_cfg = dict(type='SGD', lr=0.01, weight_decay=0.95)
>>> paramwise_cfg = dict(custom_keys={
        'backbone': dict(lr_mult=0.1, decay_mult=0.9)})
>>> optim_builder = DefaultOptimizerConstructor(
>>>     optimizer_cfg, paramwise_cfg)
>>> optimizer = optim_builder(model)
>>> # Then the `lr` and `weight_decay` for model.backbone is
>>> # (0.01 * 0.1, 0.95 * 0.9). `lr` and `weight_decay` for
>>> # model.cls_head is (0.01, 0.95).
```

add_params(*params: List[Dict]*, *module: torch.nn.modules.module.Module*, *prefix: str = ''*, *is_dcn_module: Optional[Union[int, float]] = None*) → None

Add all parameters of module to the params list.

The parameters of the given module will be added to the list of param groups, with specific rules defined by paramwise_cfg.

参数

- **params** (*list[dict]*) –A list of param groups, it will be modified in place.
- **module** (*nn.Module*) –The module to be added.
- **prefix** (*str*) –The prefix of the module
- **is_dcn_module** (*int | float | None*) –If the current module is a submodule of DCN, *is_dcn_module* will be passed to control conv_offset layer's learning rate. Defaults to None.

class mmcv.runner.**DefaultRunnerConstructor**(*runner_cfg: dict*, *default_args: Optional[dict] = None*)

Default constructor for runners.

Custom existing *Runner* like *EpocBasedRunner* though *RunnerConstructor*. For example, We can inject some new properties and functions for *Runner*.

示例

```
>>> from mmcv.runner import RUNNER_BUILDERS, build_runner
>>> # Define a new RunnerReconstructor
>>> @RUNNER_BUILDERS.register_module()
>>> class MyRunnerConstructor:
...     def __init__(self, runner_cfg, default_args=None):
...         if not isinstance(runner_cfg, dict):
...             raise TypeError('runner_cfg should be a dict',
...                             f'but got {type(runner_cfg)}')
...         self.runner_cfg = runner_cfg
...         self.default_args = default_args
...
...     def __call__(self):
...         runner = RUNNERS.build(self.runner_cfg,
...                                default_args=self.default_args)
...         # Add new properties for existing runner
...         runner.my_name = 'my_runner'
...         runner.my_function = lambda self: print(self.my_name)
...         ...
>>> # build your runner
>>> runner_cfg = dict(type='EpochBasedRunner', max_epochs=40,
...                   constructor='MyRunnerConstructor')
>>> runner = build_runner(runner_cfg)
```

class mmcv.runner.**DistEvalHook**(*dataloader: torch.utils.data.dataloader.DataLoader*, *start: Optional[int] = None*, *interval: int = 1*, *by_epoch: bool = True*, *save_best: Optional[str] = None*, *rule: Optional[str] = None*, *test_fn: Optional[Callable] = None*, *greater_keys: Optional[List[str]] = None*, *less_keys: Optional[List[str]] = None*, *broadcast_bn_buffer: bool = True*, *tmpdir: Optional[str] = None*, *gpu_collect: bool = False*, *out_dir: Optional[str] = None*, *file_client_args: Optional[dict] = None*, ***eval_kwargs*)

Distributed evaluation hook.

This hook will regularly perform evaluation in a given interval when performing in distributed environment.

参数

- **dataloader** (DataLoader) –A PyTorch dataloader, whose dataset has implemented `evaluate` function.
- **start** (*int | None, optional*) –Evaluation starting epoch. It enables evaluation before the training starts if `start` <= the resuming epoch. If None, whether to evaluate is merely decided by `interval`. Default: None.

- **interval** (*int*) –Evaluation interval. Default: 1.
- **by_epoch** (*bool*) –Determine perform evaluation by epoch or by iteration. If set to True, it will perform by epoch. Otherwise, by iteration. default: True.
- **save_best** (*str, optional*) –If a metric is specified, it would measure the best checkpoint during evaluation. The information about best checkpoint would be saved in `runner.meta['hook_msgs']` to keep best score value and best checkpoint path, which will be also loaded when resume checkpoint. Options are the evaluation metrics on the test dataset. e.g., `bbox_mAP`, `segm_mAP` for bbox detection and instance segmentation. `AR@100` for proposal recall. If `save_best` is `auto`, the first key of the returned `OrderedDict` result will be used. Default: None.
- **rule** (*str | None, optional*) –Comparison rule for best score. If set to None, it will infer a reasonable rule. Keys such as 'acc', 'top' .etc will be inferred by 'greater' rule. Keys contain 'loss' will be inferred by 'less' rule. Options are 'greater', 'less', None. Default: None.
- **test_fn** (*callable, optional*) –test a model with samples from a dataloader in a multi-gpu manner, and return the test results. If `None`, the default test function `mmcv.engine.multi_gpu_test` will be used. (default: `None`)
- **tmpdir** (*str | None*) –Temporary directory to save the results of all processes. Default: None.
- **gpu_collect** (*bool*) –Whether to use gpu or cpu to collect results. Default: False.
- **broadcast_bn_buffer** (*bool*) –Whether to broadcast the buffer(running_mean and running_var) of rank 0 to other rank before evaluation. Default: True.
- **out_dir** (*str, optional*) –The root directory to save checkpoints. If not specified, *runner.work_dir* will be used by default. If specified, the *out_dir* will be the concatenation of *out_dir* and the last level directory of *runner.work_dir*.
- **file_client_args** (*dict*) –Arguments to instantiate a FileClient. See *mmcv.fileio.FileClient* for details. Default: None.
- ****eval_kwargs** –Evaluation arguments fed into the evaluate function of the dataset.

class mmcv.runner.**DistSamplerSeedHook**

Data-loading sampler for distributed training.

When distributed training, it is only useful in conjunction with *EpochBasedRunner*, while *IterBasedRunner* achieves the same purpose with `IterLoader`.

class mmcv.runner.**DvcliveLoggerHook**(*model_file: Optional[str] = None*, *interval: int = 10*, *ignore_last: bool = True*, *reset_flag: bool = False*, *by_epoch: bool = True*, *dvclive=None*, ***kwargs*)

Class to log metrics with dvclive.

It requires dvclive to be installed.

参数

- **model_file** (*str*) –Default None. If not None, after each epoch the model will be saved to {model_file}.
- **interval** (*int*) –Logging interval (every k iterations). Default 10.
- **ignore_last** (*bool*) –Ignore the log of last iterations in each epoch if less than *interval*. Default: True.
- **reset_flag** (*bool*) –Whether to clear the output buffer after logging. Default: False.
- **by_epoch** (*bool*) –Whether EpochBasedRunner is used. Default: True.
- **dvclive** (*Live, optional*) –An instance of the Live logger to use instead of initializing a new one internally. Defaults to None.
- **kwargs** –Arguments for instantiating Live (ignored if *dvclive* is provided).

class mmcv.runner.**EMAHook**(*momentum: float = 0.0002*, *interval: int = 1*, *warm_up: int = 100*, *resume_from: Optional[str] = None*)

Exponential Moving Average Hook.

Use Exponential Moving Average on all parameters of model in training process. All parameters have a ema backup, which update by the formula as below. EMAHook takes priority over EvalHook and CheckpointSaverHook.

$$Xema\_t + 1 = (1 - \text{momentum}) \times Xema\_t + \text{momentum} \times X_t$$

参数

- **momentum** (*float*) –The momentum used for updating ema parameter. Defaults to 0.0002.
- **interval** (*int*) –Update ema parameter every interval iteration. Defaults to 1.
- **warm_up** (*int*) –During first warm_up steps, we may use smaller momentum to update ema parameters more slowly. Defaults to 100.
- **resume_from** (*str, optional*) –The checkpoint path. Defaults to None.

after_train_epoch(*runner*)

We load parameter values from ema backup to model before the EvalHook.

after_train_iter(*runner*)

Update ema parameter every self.interval iterations.

before_run(*runner*)

To resume model with it' s ema parameters more friendly.

Register ema parameter as named_buffer to model

before_train_epoch(*runner*)

We recover model's parameter from ema backup after last epoch's EvalHook.

class mmcv.runner.**EpochBasedRunner**(*model: torch.nn.modules.module.Module*, *batch_processor: Optional[Callable] = None*, *optimizer: Optional[Union[Dict, torch.optim.optimizer.Optimizer]] = None*, *work_dir: Optional[str] = None*, *logger: Optional[logging.Logger] = None*, *meta: Optional[Dict] = None*, *max_iters: Optional[int] = None*, *max_epochs: Optional[int] = None*)

Epoch-based Runner.

This runner train models epoch by epoch.

run(*data_loaders: List[*torch.utils.data.dataloader.DataLoader*]*, *workflow: List[Tuple[str, int]]*, *max_epochs: Optional[int] = None*, ***kwargs*) → None

Start running.

参数

- **data_loaders** (list[DataLoader]) –Dataloaders for training and validation.
- **workflow** (*list[tuple]*) –A list of (phase, epochs) to specify the running order and epochs. E.g, [('train', 2), ('val', 1)] means running 2 epochs for training and 1 epoch for validation, iteratively.

save_checkpoint(*out_dir: str*, *filename_tmpl: str = 'epoch_{}.pth'*, *save_optimizer: bool = True*, *meta: Optional[Dict] = None*, *create_symlink: bool = True*) → None

Save the checkpoint.

参数

- **out_dir** (*str*) –The directory that checkpoints are saved.
- **filename_tmpl** (*str, optional*) –The checkpoint filename template, which contains a placeholder for the epoch number. Defaults to 'epoch_{}.pth'.
- **save_optimizer** (*bool, optional*) –Whether to save the optimizer to the checkpoint. Defaults to True.
- **meta** (*dict, optional*) –The meta information to be saved in the checkpoint. Defaults to None.
- **create_symlink** (*bool, optional*) –Whether to create a symlink "latest.pth" to point to the latest checkpoint. Defaults to True.

class mmcv.runner.**EvalHook**(*dataloader: torch.utils.data.dataloader.DataLoader*, *start: Optional[int] = None*, *interval: int = 1*, *by_epoch: bool = True*, *save_best: Optional[str] = None*, *rule: Optional[str] = None*, *test_fn: Optional[Callable] = None*, *greater_keys: Optional[List[str]] = None*, *less_keys: Optional[List[str]] = None*, *out_dir: Optional[str] = None*, *file_client_args: Optional[dict] = None*, ***eval_kwargs*)

Non-Distributed evaluation hook.

This hook will regularly perform evaluation in a given interval when performing in non-distributed environment.

参数

- **dataloader** (DataLoader) –A PyTorch dataloader, whose dataset has implemented `evaluate` function.
- **start** (*int | None, optional*) –Evaluation starting epoch. It enables evaluation before the training starts if `start` <= the resuming epoch. If None, whether to evaluate is merely decided by `interval`. Default: None.
- **interval** (*int*) –Evaluation interval. Default: 1.
- **by_epoch** (*bool*) –Determine perform evaluation by epoch or by iteration. If set to True, it will perform by epoch. Otherwise, by iteration. Default: True.
- **save_best** (*str, optional*) –If a metric is specified, it would measure the best checkpoint during evaluation. The information about best checkpoint would be saved in `runner.meta['hook_msgs']` to keep best score value and best checkpoint path, which will be also loaded when resume checkpoint. Options are the evaluation metrics on the test dataset. e.g., `bbox_mAP`, `segm_mAP` for bbox detection and instance segmentation. `AR@100` for proposal recall. If `save_best` is `auto`, the first key of the returned `OrderedDict` result will be used. Default: None.
- **rule** (*str | None, optional*) –Comparison rule for best score. If set to None, it will infer a reasonable rule. Keys such as 'acc', 'top' .etc will be inferred by 'greater' rule. Keys contain 'loss' will be inferred by 'less' rule. Options are 'greater', 'less', None. Default: None.
- **test_fn** (*callable, optional*) –test a model with samples from a dataloader, and return the test results. If `None`, the default test function `mmcv.engine.single_gpu_test` will be used. (default: `None`)
- **greater_keys** (*List[str] | None, optional*) –Metric keys that will be inferred by 'greater' comparison rule. If `None`, _default_greater_keys will be used. (default: `None`)
- **less_keys** (*List[str] | None, optional*) –Metric keys that will be inferred by 'less' comparison rule. If `None`, _default_less_keys will be used. (default: `None`)
- **out_dir** (*str, optional*) –The root directory to save checkpoints. If not specified,

runner.work_dir will be used by default. If specified, the *out_dir* will be the concatenation of *out_dir* and the last level directory of *runner.work_dir*. *New in version 1.3.16.*

- **file_client_args** (*dict*) –Arguments to instantiate a FileClient. See *mmcv.fileio.FileClient* for details. Default: None. *New in version 1.3.16.*
- ****eval_kwargs** –Evaluation arguments fed into the evaluate function of the dataset.

注解: If new arguments are added for EvalHook, tools/test.py, tools/eval_metric.py may be affected.

---

after_train_epoch(*runner*)

Called after every training epoch to evaluate the results.

after_train_iter(*runner*)

Called after every training iter to evaluate the results.

before_train_epoch(*runner*)

Evaluate the model only at the start of training by epoch.

before_train_iter(*runner*)

Evaluate the model only at the start of training by iteration.

evaluate(*runner*, *results*)

Evaluate the results.

参数

- **runner** (mmcv.Runner) –The underlined training runner.
- **results** (*list*) –Output results.

class mmcv.runner.**ExpLrUpdaterHook**(*gamma: float*, ***kwargs*)

class mmcv.runner.**FixedLrUpdaterHook**(***kwargs*)

class mmcv.runner.**FlatCosineAnnealingLrUpdaterHook**(*start_percent: float = 0.75*, *min_lr: Optional[float] = None*, *min_lr_ratio: Optional[float] = None*, ***kwargs*)

Flat + Cosine lr schedule.

Modified from https://github.com/fastai/fastai/blob/master/fastai/callback/schedule.py#L128 # noqa: E501

参数

- **start_percent** (*float*) –When to start annealing the learning rate after the percentage of the total training steps. The value should be in range [0, 1). Default: 0.75
- **min_lr** (*float, optional*) –The minimum lr. Default: None.
- **min_lr_ratio** (*float, optional*) –The ratio of minimum lr to the base lr. Either *min_lr* or *min_lr_ratio* should be specified. Default: None.

class mmcv.runner.**Fp16OptimizerHook**(*grad_clip: Optional[dict] = None*, *coalesce: bool = True*, *bucket_size_mb: int = - 1*, *loss_scale: Union[float, str, dict] = 512.0*, *distributed: bool = True*)

FP16 optimizer hook (using PyTorch's implementation).

If you are using PyTorch >= 1.6, torch.cuda.amp is used as the backend, to take care of the optimization procedure.

参数 **loss_scale** (*float | str | dict*) –Scale factor configuration. If loss_scale is a float, static loss scaling will be used with the specified scale. If loss_scale is a string, it must be 'dynamic', then dynamic loss scaling will be used. It can also be a dict containing arguments of GradScalar. Defaults to 512. For Pytorch >= 1.6, mmcv uses official implementation of GradScaler. If you use a dict version of loss_scale to create GradScaler, please refer to: https://pytorch.org/docs/stable/amp.html#torch.cuda.amp.GradScaler for the parameters.

实际案例

```
>>> loss_scale = dict(
...     init_scale=65536.0,
...     growth_factor=2.0,
...     backoff_factor=0.5,
...     growth_interval=2000
... )
>>> optimizer_hook = Fp16OptimizerHook(loss_scale=loss_scale)
```

after_train_iter(*runner*) → None

Backward optimization steps for Mixed Precision Training. For dynamic loss scaling, please refer to https://pytorch.org/docs/stable/amp.html#torch.cuda.amp.GradScaler.

1. Scale the loss by a scale factor.
2. Backward the loss to obtain the gradients.
3. Unscale the optimizer's gradient tensors.
4. Call optimizer.step() and update scale factor.
5. Save loss_scaler state_dict for resume purpose.

before_run(*runner*) → None

Preparing steps before Mixed Precision Training.

copy_grads_to_fp32(*fp16_net: torch.nn.modules.module.Module*, *fp32_weights: torch.Tensor*) → None

Copy gradients from fp16 model to fp32 weight copy.

copy_params_to_fp16(*fp16_net: torch.nn.modules.module.Module*, *fp32_weights: torch.Tensor*) → None

Copy updated params from fp32 weight copy to fp16 model.

class mmcv.runner.**GradientCumulativeFp16OptimizerHook**(**args*, ***kwargs*)

Fp16 optimizer Hook (using PyTorch's implementation) implements multi-iters gradient cumulating.

If you are using PyTorch >= 1.6, torch.cuda.amp is used as the backend, to take care of the optimization procedure.

after_train_iter(*runner*) → None

Backward optimization steps for Mixed Precision Training. For dynamic loss scaling, please refer to https://pytorch.org/docs/stable/amp.html#torch.cuda.amp.GradScaler.

1. Scale the loss by a scale factor.
2. Backward the loss to obtain the gradients.
3. Unscale the optimizer's gradient tensors.
4. Call optimizer.step() and update scale factor.
5. Save loss_scaler state_dict for resume purpose.

class mmcv.runner.**GradientCumulativeOptimizerHook**(*cumulative_iters: int = 1*, ***kwargs*)

Optimizer Hook implements multi-iters gradient cumulating.

参数 **cumulative_iters** (*int, optional*) –Num of gradient cumulative iters. The optimizer will step every *cumulative_iters* iters. Defaults to 1.

实际案例

```
>>> # Use cumulative_iters to simulate a large batch size
>>> # It is helpful when the hardware cannot handle a large batch size.
>>> loader = DataLoader(data, batch_size=64)
>>> optim_hook = GradientCumulativeOptimizerHook(cumulative_iters=4)
>>> # almost equals to
>>> loader = DataLoader(data, batch_size=256)
>>> optim_hook = OptimizerHook()
```

class mmcv.runner.**InvLrUpdaterHook**(*gamma: float*, *power: float = 1.0*, ***kwargs*)

class mmcv.runner.**IterBasedRunner**(*model: torch.nn.modules.module.Module*, *batch_processor: Optional[Callable] = None*, *optimizer: Optional[Union[Dict, torch.optim.optimizer.Optimizer]] = None*, *work_dir: Optional[str] = None*, *logger: Optional[logging.Logger] = None*, *meta: Optional[Dict] = None*, *max_iters: Optional[int] = None*, *max_epochs: Optional[int] = None*)

Iteration-based Runner.

This runner train models iteration by iteration.

register_training_hooks(*lr_config*, *optimizer_config=None*, *checkpoint_config=None*, *log_config=None*, *momentum_config=None*, *custom_hooks_config=None*)

Register default hooks for iter-based training.

Checkpoint hook, optimizer stepper hook and logger hooks will be set to *by_epoch=False* by default.

Default hooks include:

Hooks	Priority
LrUpdaterHook	VERY_HIGH (10)
MomentumUpdaterHook	HIGH (30)
OptimizerStepperHook	ABOVE_NORMAL (40)
CheckpointSaverHook	NORMAL (50)
IterTimerHook	LOW (70)
LoggerHook(s)	VERY_LOW (90)
CustomHook(s)	defaults to NORMAL (50)

If custom hooks have same priority with default hooks, custom hooks will be triggered after default hooks.

resume(*checkpoint: str*, *resume_optimizer: bool = True*, *map_location: Union[str, Callable] = 'default'*) → None

Resume model from checkpoint.

参数

- **checkpoint** (*str*) –Checkpoint to resume from.
- **resume_optimizer** (*bool, optional*) –Whether resume the optimizer(s) if the checkpoint file includes optimizer(s). Default to True.
- **map_location** (*str, optional*) –Same as `torch.load()`. Default to 'default'.

run(*data_loaders: List[*torch.utils.data.dataloader.DataLoader*]*, *workflow: List[Tuple[str, int]]*, *max_iters: Optional[int] = None*, ***kwargs*) → None

Start running.

参数

- **data_loaders** (list[`DataLoader`]) –Dataloaders for training and validation.
- **workflow** (*list[tuple]*) –A list of (phase, iters) to specify the running order and iterations. E.g, [('train', 10000), ('val', 1000)] means running 10000 iterations for training and 1000 iterations for validation, iteratively.

save_checkpoint(*out_dir: str*, *filename_tmpl: str = 'iter_{}.pth'*, *meta: Optional[Dict] = None*, *save_optimizer: bool = True*, *create_symlink: bool = True*) → None

Save checkpoint to file.

参数

- **out_dir** (*str*) –Directory to save checkpoint files.

- **filename_tmpl** (*str, optional*) –Checkpoint file template. Defaults to 'iter_{}.pth'.
- **meta** (*dict, optional*) –Metadata to be saved in checkpoint. Defaults to None.
- **save_optimizer** (*bool, optional*) –Whether save optimizer. Defaults to True.
- **create_symlink** (*bool, optional*) –Whether create symlink to the latest checkpoint file. Defaults to True.

class mmcv.runner.**LinearAnnealingLrUpdaterHook**(*min_lr: Optional[float] = None*, *min_lr_ratio: Optional[float] = None*, ***kwargs*)

Linear annealing LR Scheduler decays the learning rate of each parameter group linearly.

参数

- **min_lr** (*float, optional*) –The minimum lr. Default: None.
- **min_lr_ratio** (*float, optional*) –The ratio of minimum lr to the base lr. Either *min_lr* or *min_lr_ratio* should be specified. Default: None.

class mmcv.runner.**LinearAnnealingMomentumUpdaterHook**(*min_momentum: Optional[float] = None*, *min_momentum_ratio: Optional[float] = None*, ***kwargs*)

Linear annealing LR Momentum decays the Momentum of each parameter group linearly.

参数

- **min_momentum** (*float, optional*) –The minimum momentum. Default: None.
- **min_momentum_ratio** (*float, optional*) –The ratio of minimum momentum to the base momentum. Either *min_momentum* or *min_momentum_ratio* should be specified. Default: None.

class mmcv.runner.**LoggerHook**(*interval: int = 10*, *ignore_last: bool = True*, *reset_flag: bool = False*, *by_epoch: bool = True*)

Base class for logger hooks.

参数

- **interval** (*int*) –Logging interval (every k iterations). Default 10.
- **ignore_last** (*bool*) –Ignore the log of last iterations in each epoch if less than *interval*. Default True.
- **reset_flag** (*bool*) –Whether to clear the output buffer after logging. Default False.
- **by_epoch** (*bool*) –Whether EpochBasedRunner is used. Default True.

get_iter(*runner*, *inner_iter: bool = False*) → int

Get the current training iteration step.

static is_scalar(*val*, *include_np: bool = True*, *include_torch: bool = True*) → bool

Tell the input variable is a scalar or not.

参数

- **val** –Input variable.
- **include_np** (*bool*) –Whether include 0-d np.ndarray as a scalar.
- **include_torch** (*bool*) –Whether include 0-d torch.Tensor as a scalar.

返回 True or False.

返回类型 bool

class mmcv.runner.**LossScaler**(*init_scale: float = 4294967296*, *mode: str = 'dynamic'*, *scale_factor: float = 2.0*, *scale_window: int = 1000*)

Class that manages loss scaling in mixed precision training which supports both dynamic or static mode.

The implementation refers to https://github.com/NVIDIA/apex/blob/master/apex/fp16_utils/loss_scaler.py. Indirectly, by supplying `mode='dynamic'` for dynamic loss scaling. It's important to understand how *LossScaler* operates. Loss scaling is designed to combat the problem of underflowing gradients encountered at long times when training fp16 networks. Dynamic loss scaling begins by attempting a very high loss scale. Ironically, this may result in OVERflowing gradients. If overflowing gradients are encountered, `FP16_Optimizer` then skips the update step for this particular iteration/minibatch, and *LossScaler* adjusts the loss scale to a lower value. If a certain number of iterations occur without overflowing gradients detected,:class:*LossScaler* increases the loss scale once more. In this way *LossScaler* attempts to "ride the edge" of always using the highest loss scale possible without incurring overflow.

参数

- **init_scale** (*float*) –Initial loss scale value, default: 2**32.
- **scale_factor** (*float*) –Factor used when adjusting the loss scale. Default: 2.
- **mode** (*str*) –Loss scaling mode. 'dynamic' or 'static'
- **scale_window** (*int*) –Number of consecutive iterations without an overflow to wait before increasing the loss scale. Default: 1000.

has_overflow(*params: List[torch.nn.parameter.Parameter]*) → bool

Check if params contain overflow.

load_state_dict(*state_dict: dict*) → None

Loads the loss_scaler state dict.

参数 **state_dict** (*dict*) –scaler state.

state_dict() → dict

Returns the state of the scaler as a `dict`.

update_scale(*overflow: bool*) → None

update the current loss scale value when overflow happens.

class mmcv.runner.**LrUpdaterHook**(*by_epoch: bool = True*, *warmup: Optional[str] = None*, *warmup_iters: int = 0*, *warmup_ratio: float = 0.1*, *warmup_by_epoch: bool = False*)

LR Scheduler in MMCV.

参数

- **by_epoch** (*bool*) –LR changes epoch by epoch
- **warmup** (*string*) –Type of warmup used. It can be None(use no warmup), 'constant', 'linear' or 'exp'
- **warmup_iters** (*int*) –The number of iterations or epochs that warmup lasts
- **warmup_ratio** (*float*) –LR used at the beginning of warmup equals to warmup_ratio * initial_lr
- **warmup_by_epoch** (*bool*) –When warmup_by_epoch == True, warmup_iters means the number of epochs that warmup lasts, otherwise means the number of iteration that warmup lasts

class mmcv.runner.**MlflowLoggerHook**(*exp_name: Optional[str] = None*, *tags: Optional[Dict] = None*, *params: Optional[Dict] = None*, *log_model: bool = True*, *interval: int = 10*, *ignore_last: bool = True*, *reset_flag: bool = False*, *by_epoch: bool = True*)

Class to log metrics and (optionally) a trained model to MLflow.

It requires MLflow to be installed.

参数

- **exp_name** (*str, optional*) –Name of the experiment to be used. Default None. If not None, set the active experiment. If experiment does not exist, an experiment with provided name will be created.
- **tags** (*Dict[str], optional*) –Tags for the current run. Default None. If not None, set tags for the current run.
- **params** (*Dict[str], optional*) –Params for the current run. Default None. If not None, set params for the current run.
- **log_model** (*bool, optional*) –Whether to log an MLflow artifact. Default True. If True, log runner.model as an MLflow artifact for the current run.
- **interval** (*int*) –Logging interval (every k iterations). Default: 10.
- **ignore_last** (*bool*) –Ignore the log of last iterations in each epoch if less than *interval*. Default: True.
- **reset_flag** (*bool*) –Whether to clear the output buffer after logging. Default: False.
- **by_epoch** (*bool*) –Whether EpochBasedRunner is used. Default: True.

class mmcv.runner.**ModuleDict**(*modules: Optional[dict] = None*, *init_cfg: Optional[dict] = None*)

ModuleDict in openmmlab.

参数

- **modules** (*dict, optional*) –a mapping (dictionary) of (string: module) or an iterable of key-value pairs of type (string, module).
- **init_cfg** (*dict, optional*) –Initialization config dict.

class mmcv.runner.**ModuleList**(*modules: Optional[Iterable] = None*, *init_cfg: Optional[dict] = None*)

ModuleList in openmmlab.

参数

- **modules** (*iterable, optional*) –an iterable of modules to add.
- **init_cfg** (*dict, optional*) –Initialization config dict.

class mmcv.runner.**NeptuneLoggerHook**(*init_kwargs: Optional[Dict] = None*, *interval: int = 10*, *ignore_last: bool = True*, *reset_flag: bool = True*, *with_step: bool = True*, *by_epoch: bool = True*)

Class to log metrics to NeptuneAI.

It requires Neptune to be installed.

参数

- **init_kwargs** (*dict*) –a dict contains the initialization keys as below:
  - project (str): Name of a project in a form of namespace/project_name. If None, the value of NEPTUNE_PROJECT environment variable will be taken.
  - api_token (str): User's API token. If None, the value of NEPTUNE_API_TOKEN environment variable will be taken. Note: It is strongly recommended to use NEPTUNE_API_TOKEN environment variable rather than placing your API token in plain text in your source code.
  - name (str, optional, default is 'Untitled'): Editable name of the run. Name is displayed in the run's Details and in Runs table as a column.

  Check https://docs.neptune.ai/api-reference/neptune#init for more init arguments.
- **interval** (*int*) –Logging interval (every k iterations). Default: 10.
- **ignore_last** (*bool*) –Ignore the log of last iterations in each epoch if less than `interval`. Default: True.
- **reset_flag** (*bool*) –Whether to clear the output buffer after logging. Default: True.
- **with_step** (*bool*) –If True, the step will be logged from `self.get_iters`. Otherwise, step will not be logged. Default: True.
- **by_epoch** (*bool*) –Whether EpochBasedRunner is used. Default: True.

class mmcv.runner.**OneCycleLrUpdaterHook**(*max_lr: Union[float, List]*, *total_steps: Optional[int] = None*, *pct_start: float = 0.3*, *anneal_strategy: str = 'cos'*, *div_factor: float = 25*, *final_div_factor: float = 10000.0*, *three_phase: bool = False*, ***kwargs*)

One Cycle LR Scheduler.

The 1cycle learning rate policy changes the learning rate after every batch. The one cycle learning rate policy is described in https://arxiv.org/pdf/1708.07120.pdf

参数

- **max_lr** (*float or list*) –Upper learning rate boundaries in the cycle for each parameter group.
- **total_steps** (*int, optional*) –The total number of steps in the cycle. Note that if a value is not provided here, it will be the max_iter of runner. Default: None.
- **pct_start** (*float*) –The percentage of the cycle (in number of steps) spent increasing the learning rate. Default: 0.3
- **anneal_strategy** (*str*) –{'cos', 'linear'} Specifies the annealing strategy: 'cos' for cosine annealing, 'linear' for linear annealing. Default: 'cos'
- **div_factor** (*float*) –Determines the initial learning rate via initial_lr = max_lr/div_factor Default: 25
- **final_div_factor** (*float*) –Determines the minimum learning rate via min_lr = initial_lr/final_div_factor Default: 1e4
- **three_phase** (*bool*) –If three_phase is True, use a third phase of the schedule to annihilate the learning rate according to final_div_factor instead of modifying the second phase (the first two phases will be symmetrical about the step indicated by pct_start). Default: False

class mmcv.runner.**OneCycleMomentumUpdaterHook**(*base_momentum: Union[float, list, dict] = 0.85*, *max_momentum: Union[float, list, dict] = 0.95*, *pct_start: float = 0.3*, *anneal_strategy: str = 'cos'*, *three_phase: bool = False*, ***kwargs*)

OneCycle momentum Scheduler.

This momentum scheduler usually used together with the OneCycleLrUpdater to improve the performance.

参数

- **base_momentum** (*float or list*) –Lower momentum boundaries in the cycle for each parameter group. Note that momentum is cycled inversely to learning rate; at the peak of a cycle, momentum is 'base_momentum' and learning rate is 'max_lr'. Default: 0.85
- **max_momentum** (*float or list*) –Upper momentum boundaries in the cycle for each parameter group. Functionally, it defines the cycle amplitude (max_momentum -

base_momentum). Note that momentum is cycled inversely to learning rate; at the start of a cycle, momentum is 'max_momentum' and learning rate is 'base_lr' Default: 0.95

- **pct_start** (*float*) –The percentage of the cycle (in number of steps) spent increasing the learning rate. Default: 0.3
- **anneal_strategy** (*str*) –{'cos', 'linear'} Specifies the annealing strategy: 'cos' for cosine annealing, 'linear' for linear annealing. Default: 'cos'
- **three_phase** (*bool*) –If three_phase is True, use a third phase of the schedule to annihilate the learning rate according to final_div_factor instead of modifying the second phase (the first two phases will be symmetrical about the step indicated by pct_start). Default: False

class mmcv.runner.**OptimizerHook**(*grad_clip: Optional[dict] = None*, *detect_anomalous_params: bool = False*)

A hook contains custom operations for the optimizer.

参数

- **grad_clip** (*dict, optional*) –A config dict to control the clip_grad. Default: None.
- **detect_anomalous_params** (*bool*) –This option is only used for debugging which will slow down the training speed. Detect anomalous parameters that are not included in the computational graph with *loss* as the root. There are two cases
  - Parameters were not used during forward pass.
  - Parameters were not used to produce loss.

  Default: False.

class mmcv.runner.**PaviLoggerHook**(*init_kwargs: Optional[Dict] = None*, *add_graph: Optional[bool] = None*, *img_key: Optional[str] = None*, *add_last_ckpt: bool = False*, *interval: int = 10*, *ignore_last: bool = True*, *reset_flag: bool = False*, *by_epoch: bool = True*, *add_graph_kwargs: Optional[Dict] = None*, *add_ckpt_kwargs: Optional[Dict] = None*)

Class to visual model, log metrics (for internal use).

参数

- **init_kwargs** (*dict*) –A dict contains the initialization keys as below:
  - name (str, optional): Custom training name. Defaults to None, which means current work_dir.
  - project (str, optional): Project name. Defaults to "default".
  - model (str, optional): Training model name. Defaults to current model.
  - session_text (str, optional): Session string in YAML format. Defaults to current config.
  - training_id (int, optional): Training ID in PAVI, if you want to use an existing training. Defaults to None.

  - compare_id (int, optional): Compare ID in PAVI, if you want to add the task to an existing compare. Defaults to None.
  - overwrite_last_training (bool, optional): Whether to upload data to the training with the same name in the same project, rather than creating a new one. Defaults to False.

- **add_graph** (*bool, optional*) –**Deprecated**. Whether to visual model. Default: False.
- **img_key** (*str, optional*) –**Deprecated**. Image key. Defaults to None.
- **add_last_ckpt** (*bool*) –Whether to save checkpoint after run. Default: False.
- **interval** (*int*) –Logging interval (every k iterations). Default: True.
- **ignore_last** (*bool*) –Ignore the log of last iterations in each epoch if less than *interval*. Default: True.
- **reset_flag** (*bool*) –Whether to clear the output buffer after logging. Default: False.
- **by_epoch** (*bool*) –Whether EpochBasedRunner is used. Default: True.
- **add_graph_kwargs** (*dict, optional*) –A dict contains the params for adding graph, the keys are as below: - active (bool): Whether to use `add_graph`. Default: False. - start (int): The epoch or iteration to start. Default: 0. - interval (int): Interval of `add_graph`. Default: 1. - img_key (str): Get image data from Dataset. Default: 'img'. - opset_version (int): `opset_version` of exporting onnx.

  Default: 11.

  - **dummy_forward_kwargs (dict, optional): Set default parameters to** model forward function except image. For example, you can set {'return_loss': False} for mmcls. Default: None.

- **add_ckpt_kwargs** (*dict, optional*) –A dict contains the params for adding checkpoint, the keys are as below: - active (bool): Whether to upload checkpoint. Default: False. - start (int): The epoch or iteration to start. Default: 0. - interval (int): Interval of upload checkpoint. Default: 1.

get_step (*runner*) → int

Get the total training step/epoch.

class mmcv.runner.**PolyLrUpdaterHook** (*power: float = 1.0*, *min_lr: float = 0.0*, ***kwargs*)

class mmcv.runner.**Priority** (*value*)

Hook priority levels.

Level	Value
HIGHEST	0
VERY_HIGH	10
HIGH	30
ABOVE_NORMAL	40
NORMAL	50
BELOW_NORMAL	60
LOW	70
VERY_LOW	90
LOWEST	100

class mmcv.runner.**Runner**(**args*, ***kwargs*)

Deprecated name of EpochBasedRunner.

class mmcv.runner.**SegmindLoggerHook**(*interval: int = 10*, *ignore_last: bool = True*, *reset_flag: bool = False*, *by_epoch=True*)

Class to log metrics to Segmind.

It requires Segmind to be installed.

参数

- **interval** (*int*) –Logging interval (every k iterations). Default: 10.
- **ignore_last** (*bool*) –Ignore the log of last iterations in each epoch if less than *interval*. Default True.
- **reset_flag** (*bool*) –Whether to clear the output buffer after logging. Default False.
- **by_epoch** (*bool*) –Whether EpochBasedRunner is used. Default True.

class mmcv.runner.**Sequential**(**args*, *init_cfg: Optional[dict] = None*)

Sequential module in openmmlab.

参数 **init_cfg** (*dict, optional*) –Initialization config dict.

class mmcv.runner.**StepLrUpdaterHook**(*step: Union[int, List[int]]*, *gamma: float = 0.1*, *min_lr: Optional[float] = None*, ***kwargs*)

Step LR scheduler with min_lr clipping.

参数

- **step** (*int | list[int]*) –Step to decay the LR. If an int value is given, regard it as the decay interval. If a list is given, decay LR at these steps.
- **gamma** (*float*) –Decay LR ratio. Defaults to 0.1.
- **min_lr** (*float, optional*) –Minimum LR value to keep. If LR after decay is lower than *min_lr*, it will be clipped to this value. If None is given, we don’t perform lr clipping.

Default: None.

class mmcv.runner.**StepMomentumUpdaterHook**(*step: Union[int, List[int]]*, *gamma: float = 0.5*, *min_momentum: Optional[float] = None*, ***kwargs*)

Step momentum scheduler with min value clipping.

参数

- **step** (*int | list[int]*) –Step to decay the momentum. If an int value is given, regard it as the decay interval. If a list is given, decay momentum at these steps.
- **gamma** (*float, optional*) –Decay momentum ratio. Default: 0.5.
- **min_momentum** (*float, optional*) –Minimum momentum value to keep. If momentum after decay is lower than this value, it will be clipped accordingly. If None is given, we don't perform lr clipping. Default: None.

class mmcv.runner.**SyncBuffersHook**(*distributed: bool = True*)

Synchronize model buffers such as running_mean and running_var in BN at the end of each epoch.

参数 **distributed** (*bool*) –Whether distributed training is used. It is effective only for distributed training. Defaults to True.

after_epoch(*runner*)

All-reduce model buffers at the end of each epoch.

class mmcv.runner.**TensorboardLoggerHook**(*log_dir: Optional[str] = None*, *interval: int = 10*, *ignore_last: bool = True*, *reset_flag: bool = False*, *by_epoch: bool = True*)

Class to log metrics to Tensorboard.

参数

- **log_dir** (*string*) –Save directory location. Default: None. If default values are used, directory location is `runner.work_dir`/tf_logs.
- **interval** (*int*) –Logging interval (every k iterations). Default: True.
- **ignore_last** (*bool*) –Ignore the log of last iterations in each epoch if less than *interval*. Default: True.
- **reset_flag** (*bool*) –Whether to clear the output buffer after logging. Default: False.
- **by_epoch** (*bool*) –Whether EpochBasedRunner is used. Default: True.

class mmcv.runner.**TextLoggerHook**(*by_epoch: bool = True*, *interval: int = 10*, *ignore_last: bool = True*, *reset_flag: bool = False*, *interval_exp_name: int = 1000*, *out_dir: Optional[str] = None*, *out_suffix: Union[str, tuple] = ('.log.json', '.log', '.py')*, *keep_local: bool = True*, *file_client_args: Optional[Dict] = None*)

Logger hook in text.

In this logger hook, the information will be printed on terminal and saved in json file.

参数

- **by_epoch** (*bool, optional*) –Whether EpochBasedRunner is used. Default: True.
- **interval** (*int, optional*) –Logging interval (every k iterations). Default: 10.
- **ignore_last** (*bool, optional*) –Ignore the log of last iterations in each epoch if less than `interval`. Default: True.
- **reset_flag** (*bool, optional*) –Whether to clear the output buffer after logging. Default: False.
- **interval_exp_name** (*int, optional*) –Logging interval for experiment name. This feature is to help users conveniently get the experiment information from screen or log file. Default: 1000.
- **out_dir** (*str, optional*) –Logs are saved in `runner.work_dir` default. If `out_dir` is specified, logs will be copied to a new directory which is the concatenation of `out_dir` and the last level directory of `runner.work_dir`. Default: None. *New in version 1.3.16.*
- **out_suffix** (*str or tuple[str], optional*) –Those filenames ending with `out_suffix` will be copied to `out_dir`. Default: ('.log.json', '.log', '.py'). *New in version 1.3.16.*
- **keep_local** (*bool, optional*) –Whether to keep local log when `out_dir` is specified. If False, the local log will be removed. Default: True. *New in version 1.3.16.*
- **file_client_args** (*dict, optional*) –Arguments to instantiate a FileClient. See *mmcv.fileio.FileClient* for details. Default: None. *New in version 1.3.16.*

class mmcv.runner.**WandbLoggerHook**(*init_kwargs: Optional[Dict] = None*, *interval: int = 10*, *ignore_last: bool = True*, *reset_flag: bool = False*, *commit: bool = True*, *by_epoch: bool = True*, *with_step: bool = True*, *log_artifact: bool = True*, *out_suffix: Union[str, tuple] = ('.log.json', '.log', '.py')*, *define_metric_cfg: Optional[Dict] = None*)

Class to log metrics with wandb.

It requires wandb to be installed.

参数

- **init_kwargs** (*dict*) –A dict contains the initialization keys. Check https://docs.wandb.ai/ref/python/init for more init arguments.
- **interval** (*int*) –Logging interval (every k iterations). Default 10.
- **ignore_last** (*bool*) –Ignore the log of last iterations in each epoch if less than *interval*. Default: True.

- **reset_flag** (*bool*) –Whether to clear the output buffer after logging. Default: False.
- **commit** (*bool*) –Save the metrics dict to the wandb server and increment the step. If false `wandb.log` just updates the current metrics dict with the row argument and metrics won't be saved until `wandb.log` is called with `commit=True`. Default: True.
- **by_epoch** (*bool*) –Whether EpochBasedRunner is used. Default: True.
- **with_step** (*bool*) –If True, the step will be logged from `self.get_iters`. Otherwise, step will not be logged. Default: True.
- **log_artifact** (*bool*) –If True, artifacts in {work_dir} will be uploaded to wandb after training ends. Default: True *New in version 1.4.3.*
- **out_suffix** (*str or tuple[str], optional*) –Those filenames ending with `out_suffix` will be uploaded to wandb. Default: ('.log.json', '.log', '.py'). *New in version 1.4.3.*
- **define_metric_cfg** (*dict, optional*) –A dict of metrics and summaries for wandb.define_metric. The key is metric and the value is summary. The summary should be in ["min", "max", "mean","best", "last",

  "none"].

  For example, if setting `define_metric_cfg={'coco/bbox_mAP': 'max'}`, the maximum value of `coco/bbox_mAP` will be logged on wandb UI. See wandb docs for details. Defaults to None. *New in version 1.6.3.*

mmcv.runner.**allreduce_grads**(*params: List[torch.nn.parameter.Parameter]*, *coalesce: bool = True*, *bucket_size_mb: int = - 1*) → None

Allreduce gradients.

参数

- **params** (*list[torch.nn.Parameter]*) –List of parameters of a model.
- **coalesce** (*bool, optional*) –Whether allreduce parameters as a whole. Defaults to True.
- **bucket_size_mb** (*int, optional*) –Size of bucket, the unit is MB. Defaults to -1.

mmcv.runner.**allreduce_params**(*params: List[torch.nn.parameter.Parameter]*, *coalesce: bool = True*, *bucket_size_mb: int = - 1*) → None

Allreduce parameters.

参数

- **params** (*list[torch.nn.Parameter]*) –List of parameters or buffers of a model.
- **coalesce** (*bool, optional*) –Whether allreduce parameters as a whole. Defaults to True.
- **bucket_size_mb** (*int, optional*) –Size of bucket, the unit is MB. Defaults to -1.

mmcv.runner.**auto_fp16**(*apply_to: Optional[Iterable] = None*, *out_fp32: bool = False*, *supported_types: tuple = (<class 'torch.nn.modules.module.Module'>,)*) → Callable

Decorator to enable fp16 training automatically.

This decorator is useful when you write custom modules and want to support mixed precision training. If inputs arguments are fp32 tensors, they will be converted to fp16 automatically. Arguments other than fp32 tensors are ignored. If you are using PyTorch >= 1.6, torch.cuda.amp is used as the backend, otherwise, original mmcv implementation will be adopted.

参数

- **apply_to** (*Iterable, optional*) –The argument names to be converted. *None* indicates all arguments.
- **out_fp32** (*bool*) –Whether to convert the output back to fp32.
- **supported_types** (*tuple*) –Classes can be decorated by auto_fp16. *New in version 1.5.0.*

示例

```
>>> import torch.nn as nn
>>> class MyModule1(nn.Module):
>>>
>>>     # Convert x and y to fp16
>>>     @auto_fp16()
>>>     def forward(self, x, y):
>>>         pass
```

```
>>> import torch.nn as nn
>>> class MyModule2(nn.Module):
>>>
>>>     # convert pred to fp16
>>>     @auto_fp16(apply_to=('pred', ))
>>>     def do_something(self, pred, others):
>>>         pass
```

mmcv.runner.**force_fp32**(*apply_to: Optional[Iterable] = None*, *out_fp16: bool = False*) → Callable

Decorator to convert input arguments to fp32 in force.

This decorator is useful when you write custom modules and want to support mixed precision training. If there are some inputs that must be processed in fp32 mode, then this decorator can handle it. If inputs arguments are fp16 tensors, they will be converted to fp32 automatically. Arguments other than fp16 tensors are ignored. If you are using PyTorch >= 1.6, torch.cuda.amp is used as the backend, otherwise, original mmcv implementation will be adopted.

参数

- **apply_to** (*Iterable, optional*) –The argument names to be converted. *None* indicates all arguments.
- **out_fp16** (*bool*) –Whether to convert the output back to fp16.

示例

```
>>> import torch.nn as nn
>>> class MyModule1(nn.Module):
>>>
>>>     # Convert x and y to fp32
>>>     @force_fp32()
>>>     def loss(self, x, y):
>>>         pass
```

```
>>> import torch.nn as nn
>>> class MyModule2(nn.Module):
>>>
>>>     # convert pred to fp32
>>>     @force_fp32(apply_to=('pred', ))
>>>     def post_process(self, pred, others):
>>>         pass
```

mmcv.runner.**get_host_info**() → str

Get hostname and username.

Return empty string if exception raised, e.g. `getpass.getuser()` will lead to error in docker container

mmcv.runner.**get_priority**(*priority: Union[int, str,* mmcv.runner.priority.Priority*]*) → int

Get priority value.

参数 **priority** (int or str or *Priority*) –Priority.

返回 The priority value.

返回类型 int

mmcv.runner.**load_checkpoint**(*model: torch.nn.modules.module.Module*, *filename: str*, *map_location: Optional[Union[str, Callable]] = None*, *strict: bool = False*, *logger: Optional[logging.Logger] = None*, *revise_keys: list = [('^module\\.', '')]*) → Union[dict, collections.OrderedDict]

Load checkpoint from a file or URI.

参数

- **model** (*Module*) –Module to load checkpoint.

- **filename** (*str*) –Accept local filepath, URL, `torchvision://xxx`, `open-mmlab://xxx`. Please refer to `docs/model_zoo.md` for details.
- **map_location** (*str*) –Same as `torch.load()`.
- **strict** (*bool*) –Whether to allow different params for the model and checkpoint.
- **logger** (`logging.Logger` or None) –The logger for error message.
- **revise_keys** (*list*) –A list of customized keywords to modify the state_dict in checkpoint. Each item is a (pattern, replacement) pair of the regular expression operations. Default: strip the prefix 'module.' by [(r'^module.', '')].

返回 The loaded checkpoint.

返回类型 dict or OrderedDict

`mmcv.runner.`**load_state_dict**(*module: torch.nn.modules.module.Module*, *state_dict: Union[dict, collections.OrderedDict]*, *strict: bool = False*, *logger: Optional[logging.Logger] = None*) → None

Load state_dict to a module.

This method is modified from `torch.nn.Module.load_state_dict()`. Default value for `strict` is set to `False` and the message for param mismatch will be shown even if strict is False.

参数

- **module** (*Module*) –Module that receives the state_dict.
- **state_dict** (*dict or OrderedDict*) –Weights.
- **strict** (*bool*) –whether to strictly enforce that the keys in `state_dict` match the keys returned by this module's `state_dict()` function. Default: `False`.
- **logger** (`logging.Logger`, optional) –Logger to log the error message. If not specified, print function will be used.

`mmcv.runner.`**obj_from_dict**(*info: dict*, *parent: Optional[module] = None*, *default_args: Optional[dict] = None*)

Initialize an object from dict.

The dict must contain the key "type", which indicates the object type, it can be either a string or type, such as "list" or `list`. Remaining fields are treated as the arguments for constructing the object.

参数

- **info** (*dict*) –Object types and arguments.
- **parent** (`module`) –Module which may containing expected object classes.
- **default_args** (*dict, optional*) –Default arguments for initializing the object.

返回 Object built from the dict.

返回类型 any type

mmcv.runner.**save_checkpoint**(*model: torch.nn.modules.module.Module*, *filename: str*, *optimizer: Optional[torch.optim.optimizer.Optimizer] = None*, *meta: Optional[dict] = None*, *file_client_args: Optional[dict] = None*) → None

Save checkpoint to file.

The checkpoint will have 3 fields: `meta`, `state_dict` and `optimizer`. By default `meta` will contain version and time info.

参数

- **model** (*Module*) –Module whose params are to be saved.
- **filename** (*str*) –Checkpoint filename.
- **optimizer** (`Optimizer`, optional) –Optimizer to be saved.
- **meta** (*dict, optional*) –Metadata to be saved in checkpoint.
- **file_client_args** (*dict, optional*) –Arguments to instantiate a FileClient. See *mmcv.fileio.FileClient* for details. Default: None. *New in version 1.3.16.*

mmcv.runner.**set_random_seed**(*seed: int*, *deterministic: bool = False*, *use_rank_shift: bool = False*) → None

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.
- **rank_shift** (*bool*) –Whether to add rank number to the random seed to have different random seed in different threads. Default: False.

mmcv.runner.**weights_to_cpu**(*state_dict: collections.OrderedDict*) → collections.OrderedDict

Copy a model state_dict to cpu.

参数 **state_dict** (*OrderedDict*) –Model weights on GPU.

返回 Model weights on GPU.

返回类型 OrderedDict

mmcv.runner.**wrap_fp16_model**(*model: torch.nn.modules.module.Module*) → None

Wrap the FP32 model to FP16.

If you are using PyTorch >= 1.6, torch.cuda.amp is used as the backend, otherwise, original mmcv implementation will be adopted.

For PyTorch >= 1.6, this function will 1. Set fp16 flag inside the model to True.

Otherwise: 1. Convert FP32 model to FP16. 2. Remain some necessary layers to be FP32, e.g., normalization layers. 3. Set *fp16_enabled* flag inside the model to True.

参数 **model** (*nn.Module*) –Model in FP32.

CHAPTER 34

engine

mmcv.engine.**collect_results_cpu**(*result_part: list*, *size: int*, *tmpdir: Optional[str] = None*) → Optional[list]

Collect results under cpu mode.

On cpu mode, this function will save the results on different gpus to `tmpdir` and collect them by the rank 0 worker.

参数

- **result_part** (*list*) –Result list containing result parts to be collected.
- **size** (*int*) –Size of the results, commonly equal to length of the results.
- **tmpdir** (*str | None*) –temporal directory for collected results to store. If set to None, it will create a random temporal directory for it.

返回 The collected results.

返回类型 list

mmcv.engine.**collect_results_gpu**(*result_part: list*, *size: int*) → Optional[list]

Collect results under gpu mode.

On gpu mode, this function will encode results to gpu tensors and use gpu communication for results collection.

参数

- **result_part** (*list*) –Result list containing result parts to be collected.
- **size** (*int*) –Size of the results, commonly equal to length of the results.

返回 The collected results.

返回类型 list

mmcv.engine.**multi_gpu_test**(*model: torch.nn.modules.module.Module*, *data_loader:* torch.utils.data.dataloader.DataLoader, *tmpdir: Optional[str] = None*, *gpu_collect: bool = False*) → Optional[list]

Test model with multiple gpus.

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** (*nn.Dataloader*) –Pytorch data loader.
- **tmpdir** (*str*) –Path of directory to save the temporary results from different gpus under cpu mode.
- **gpu_collect** (*bool*) –Option to use either gpu or cpu to collect results.

返回 The prediction results.

返回类型 list

mmcv.engine.**single_gpu_test**(*model: torch.nn.modules.module.Module*, *data_loader:* torch.utils.data.dataloader.DataLoader) → list

Test model with a single gpu.

This method tests model with a single gpu and displays test progress bar.

参数

- **model** (*nn.Module*) –Model to be tested.
- **data_loader** (*nn.Dataloader*) –Pytorch data loader.

返回 The prediction results.

返回类型 list

CHAPTER 35

ops

class mmcv.ops.**BorderAlign**(*pool_size: int*)

Border align pooling layer.

Applies border_align over the input feature based on predicted bboxes. The details were described in the paper BorderDet: Border Feature for Dense Object Detection.

For each border line (e.g. top, left, bottom or right) of each box, border_align does the following:

1. uniformly samples `pool_size` +1 positions on this line, involving the start and end points.
2. the corresponding features on these points are computed by bilinear interpolation.
3. max pooling over all the `pool_size` +1 positions are used for computing pooled feature.

参数 **pool_size** (*int*) –number of positions sampled over the boxes' borders (e.g. top, bottom, left, right).

forward(*input: torch.Tensor*, *boxes: torch.Tensor*) → torch.Tensor

参数

- **input** –Features with shape [N,4C,H,W]. Channels ranged in [0,C), [C,2C), [2C,3C), [3C,4C) represent the top, left, bottom, right features respectively.
- **boxes** –Boxes with shape [N,H*W,4]. Coordinate format (x1,y1,x2,y2).

返回 Pooled features with shape [N,C,H*W,4]. The order is (top,left,bottom,right) for the last dimension.

返回类型 torch.Tensor

class mmcv.ops.**CARAFE**(*kernel_size: int*, *group_size: int*, *scale_factor: int*)

CARAFE: Content-Aware ReAssembly of FEatures

Please refer to CARAFE: Content-Aware ReAssembly of FEatures for more details.

参数

- **kernel_size** (*int*) –reassemble kernel size
- **group_size** (*int*) –reassemble group size
- **scale_factor** (*int*) –upsample ratio

返回 upsampled feature map

forward(*features: torch.Tensor*, *masks: torch.Tensor*) → torch.Tensor

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 mmcv.ops.**CARAFENaive**(*kernel_size: int*, *group_size: int*, *scale_factor: int*)

forward(*features: torch.Tensor*, *masks: torch.Tensor*) → torch.Tensor

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 mmcv.ops.**CARAFEPack**(*channels: int*, *scale_factor: int*, *up_kernel: int = 5*, *up_group: int = 1*, *encoder_kernel: int = 3*, *encoder_dilation: int = 1*, *compressed_channels: int = 64*)

A unified package of CARAFE upsampler that contains: 1) channel compressor 2) content encoder 3) CARAFE op.

Official implementation of ICCV 2019 paper CARAFE: Content-Aware ReAssembly of FEatures.

参数

- **channels** (*int*) –input feature channels

- **scale_factor** (*int*) –upsample ratio
- **up_kernel** (*int*) –kernel size of CARAFE op
- **up_group** (*int*) –group size of CARAFE op
- **encoder_kernel** (*int*) –kernel size of content encoder
- **encoder_dilation** (*int*) –dilation of content encoder
- **compressed_channels** (*int*) –output channels of channels compressor

返回 upsampled feature map

forward(*x: torch.Tensor*) → torch.Tensor

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.

---

mmcv.ops.**Conv2d**

alias of `mmcv.ops.deprecated_wrappers.Conv2d_deprecated`

mmcv.ops.**ConvTranspose2d**

alias of `mmcv.ops.deprecated_wrappers.ConvTranspose2d_deprecated`

class mmcv.ops.**CornerPool**(*mode: str*)

Corner Pooling.

Corner Pooling is a new type of pooling layer that helps a convolutional network better localize corners of bounding boxes.

Please refer to CornerNet: Detecting Objects as Paired Keypoints for more details.

Code is modified from https://github.com/princeton-vl/CornerNet-Lite.

参数 **mode** (*str*) –Pooling orientation for the pooling layer

- 'bottom': Bottom Pooling
- 'left': Left Pooling
- 'right': Right Pooling
- 'top': Top Pooling

返回 Feature map after pooling.

forward(*x: torch.Tensor*) → torch.Tensor

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 mmcv.ops.**Correlation**(*kernel_size: int = 1*, *max_displacement: int = 1*, *stride: int = 1*, *padding: int = 0*, *dilation: int = 1*, *dilation_patch: int = 1*)

Correlation operator.

This correlation operator works for optical flow correlation computation.

There are two batched tensors with shape (N, C, H, W), and the correlation output' s shape is $(N, max\_displacement \times 2 + 1, max\_displacement * 2 + 1, H_{out}, W_{out})$

where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times padding - dilation \times (kernel\_size - 1) - 1}{stride} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times padding - dilation \times (kernel\_size - 1) - 1}{stride} + 1 \right\rfloor$$

the correlation item (N_i, dy, dx) is formed by taking the sliding window convolution between input1 and shifted input2,

$$Corr(N_i, dx, dy) = \sum_{c=0}^{C-1} input1(N_i, c) \star \mathcal{S}(input2(N_i, c), dy, dx)$$

where $\star$ is the valid 2d sliding window convolution operator, and $\mathcal{S}$ means shifting the input features (auto-complete zero marginal), and dx, dy are shifting distance, $dx, dy \in [-max\_displacement \times dilation\_patch, max\_displacement \times dilation\_patch]$.

参数

- **kernel_size** (*int*) –The size of sliding window i.e. local neighborhood representing the center points and involved in correlation computation. Defaults to 1.
- **max_displacement** (*int*) –The radius for computing correlation volume, but the actual working space can be dilated by dilation_patch. Defaults to 1.
- **stride** (*int*) –The stride of the sliding blocks in the input spatial dimensions. Defaults to 1.
- **padding** (*int*) –Zero padding added to all four sides of the input1. Defaults to 0.
- **dilation** (*int*) –The spacing of local neighborhood that will involved in correlation. Defaults to 1.
- **dilation_patch** (*int*) –The spacing between position need to compute correlation. Defaults to 1.

forward(*input1: torch.Tensor*, *input2: torch.Tensor*) → torch.Tensor

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 mmcv.ops.**CrissCrossAttention**(*in_channels: int*)

Criss-Cross Attention Module.

注解： Before v1.3.13, we use a CUDA op. Since v1.3.13, we switch to a pure PyTorch and equivalent implementation. For more details, please refer to https://github.com/open-mmlab/mmcv/pull/1201.

Speed comparison for one forward pass

- Input size: [2,512,97,97]
- Device: 1 NVIDIA GeForce RTX 2080 Ti

	PyTorch version	CUDA version	Relative speed
with torch.no_grad()	0.00554402 s	0.0299619 s	5.4x
no with torch.no_grad()	0.00562803 s	0.0301349 s	5.4x

参数 **in_channels** (*int*) –Channels of the input feature map.

forward(*x: torch.Tensor*) → torch.Tensor

forward function of Criss-Cross Attention.

参数 **x** (*torch.Tensor*) –Input feature with the shape of (batch_size, in_channels, height, width).

返回 Output of the layer, with the shape of (batch_size, in_channels, height, width)

返回类型 torch.Tensor

class mmcv.ops.**DeformConv2d**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, ...]]*, *stride: Union[int, Tuple[int, ...]] = 1*, *padding: Union[int, Tuple[int, ...]] = 0*, *dilation: Union[int, Tuple[int, ...]] = 1*, *groups: int = 1*, *deform_groups: int = 1*, *bias: bool = False*, *im2col_step: int = 32*)

Deformable 2D convolution.

Applies a deformable 2D convolution over an input signal composed of several input planes. DeformConv2d was described in the paper Deformable Convolutional Networks

---

注解： The argument `im2col_step` was added in version 1.3.17, which means number of samples processed by the `im2col_cuda_kernel` per call. It enables users to define `batch_size` and `im2col_step` more flexibly and solved issue mmcv#1440.

参数

- **in_channels** (*int*) –Number of channels in the input image.
- **out_channels** (*int*) –Number of channels produced by the convolution.
- **kernel_size** (*int, tuple*) –Size of the convolving kernel.
- **stride** (*int, tuple*) –Stride of the convolution. Default: 1.
- **padding** (*int or tuple*) –Zero-padding added to both sides of the input. Default: 0.
- **dilation** (*int or tuple*) –Spacing between kernel elements. Default: 1.
- **groups** (*int*) –Number of blocked connections from input. channels to output channels. Default: 1.
- **deform_groups** (*int*) –Number of deformable group partitions.
- **bias** (*bool*) –If True, adds a learnable bias to the output. Default: False.
- **im2col_step** (*int*) –Number of samples processed by im2col_cuda_kernel per call. It will work when `batch_size` > `im2col_step`, but `batch_size` must be divisible by `im2col_step`. Default: 32. *New in version 1.3.17.*

forward(*x: torch.Tensor*, *offset: torch.Tensor*) → torch.Tensor

Deformable Convolutional forward function.

参数

- **x** (*Tensor*) –Input feature, shape (B, C_in, H_in, W_in)
- **offset** (*Tensor*) –Offset for deformable convolution, shape (B, deform_groups*kernel_size[0]*kernel_size[1]*2, H_out, W_out), H_out, W_out are equal to the output's.

  An offset is like *[y0, x0, y1, x1, y2, x2, ⋯, y8, x8]*. The spatial arrangement is like:

```
(x0, y0) (x1, y1) (x2, y2)
(x3, y3) (x4, y4) (x5, y5)
(x6, y6) (x7, y7) (x8, y8)
```

返回 Output of the layer.

返回类型 Tensor

class mmcv.ops.**DeformConv2dPack**(**args*, ***kwargs*)

A Deformable Conv Encapsulation that acts as normal Conv layers.

The offset tensor is like *[y0, x0, y1, x1, y2, x2, ⋯, y8, x8]*. The spatial arrangement is like:

```
(x0, y0) (x1, y1) (x2, y2)
(x3, y3) (x4, y4) (x5, y5)
(x6, y6) (x7, y7) (x8, y8)
```

参数

- **in_channels** (*int*) –Same as nn.Conv2d.
- **out_channels** (*int*) –Same as nn.Conv2d.
- **kernel_size** (*int or tuple[int]*) –Same as nn.Conv2d.
- **stride** (*int or tuple[int]*) –Same as nn.Conv2d.
- **padding** (*int or tuple[int]*) –Same as nn.Conv2d.
- **dilation** (*int or tuple[int]*) –Same as nn.Conv2d.
- **groups** (*int*) –Same as nn.Conv2d.
- **bias** (*bool or str*) –If specified as *auto*, it will be decided by the norm_cfg. Bias will be set as True if norm_cfg is None, otherwise False.

forward(*x: torch.Tensor*) → torch.Tensor

Deformable Convolutional forward function.

参数

- **x** (*Tensor*) –Input feature, shape (B, C_in, H_in, W_in)
- **offset** (*Tensor*) –Offset for deformable convolution, shape (B, deform_groups*kernel_size[0]*kernel_size[1]*2, H_out, W_out), H_out, W_out are equal to the output's.

  An offset is like *[y0, x0, y1, x1, y2, x2, ⋯, y8, x8]*. The spatial arrangement is like:

```
(x0, y0) (x1, y1) (x2, y2)
(x3, y3) (x4, y4) (x5, y5)
(x6, y6) (x7, y7) (x8, y8)
```

返回 Output of the layer.

返回类型 Tensor

class mmcv.ops.**DeformRoIPool**(*output_size: Tuple[int, ...]*, *spatial_scale: float = 1.0*, *sampling_ratio: int = 0*, *gamma: float = 0.1*)

forward(*input: torch.Tensor*, *rois: torch.Tensor*, *offset: Optional[torch.Tensor] = None*) → torch.Tensor

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 mmcv.ops.**DeformRoIPoolPack**(*output_size: Tuple[int, ...]*, *output_channels: int*, *deform_fc_channels: int = 1024*, *spatial_scale: float = 1.0*, *sampling_ratio: int = 0*, *gamma: float = 0.1*)

forward(*input: torch.Tensor*, *rois: torch.Tensor*) → torch.Tensor

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 mmcv.ops.**DynamicScatter**(*voxel_size: List*, *point_cloud_range: List*, *average_points: bool*)

Scatters points into voxels, used in the voxel encoder with dynamic voxelization.

注解: The CPU and GPU implementation get the same output, but have numerical difference after summation and division (e.g., 5e-7).

参数

- **voxel_size** (*list*) –list [x, y, z] size of three dimension.
- **point_cloud_range** (*list*) –The coordinate range of points, [x_min, y_min, z_min, x_max, y_max, z_max].
- **average_points** (*bool*) –whether to use avg pooling to scatter points into voxel.

forward(*points: torch.Tensor*, *coors: torch.Tensor*) → Tuple[torch.Tensor, torch.Tensor]

Scatters points/features into voxels.

参数

- **points** (*torch.Tensor*) –Points to be reduced into voxels.

- **coors** (*torch.Tensor*) –Corresponding voxel coordinates (specifically multi-dim voxel index) of each points.

返回 A tuple contains two elements. The first one is the voxel features with shape [M, C] which are respectively reduced from input features that share the same voxel coordinates. The second is voxel coordinates with shape [M, ndim].

返回类型 tuple[torch.Tensor]

forward_single(*points: torch.Tensor*, *coors: torch.Tensor*) → Tuple[torch.Tensor, torch.Tensor]

Scatters points into voxels.

参数

- **points** (*torch.Tensor*) –Points to be reduced into voxels.
- **coors** (*torch.Tensor*) –Corresponding voxel coordinates (specifically multi-dim voxel index) of each points.

返回 A tuple contains two elements. The first one is the voxel features with shape [M, C] which are respectively reduced from input features that share the same voxel coordinates. The second is voxel coordinates with shape [M, ndim].

返回类型 tuple[torch.Tensor]

class mmcv.ops.**FusedBiasLeakyReLU**(*num_channels: int*, *negative_slope: float = 0.2*, *scale: float = 1.4142135623730951*)

Fused bias leaky ReLU.

This function is introduced in the StyleGAN2: Analyzing and Improving the Image Quality of StyleGAN

The bias term comes from the convolution operation. In addition, to keep the variance of the feature map or gradients unchanged, they also adopt a scale similarly with Kaiming initialization. However, since the $1 + alpha^2$ is too small, we can just ignore it. Therefore, the final scale is just $\sqrt{2}$. Of course, you may change it with your own scale.

TODO: Implement the CPU version.

参数

- **num_channels** (*int*) –The channel number of the feature map.
- **negative_slope** (*float, optional*) –Same as nn.LeakyRelu. Defaults to 0.2.
- **scale** (*float, optional*) –A scalar to adjust the variance of the feature map. Defaults to 2**0.5.

forward(*input: torch.Tensor*) → torch.Tensor

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 mmcv.ops.**GroupAll**(*use_xyz: bool = True*)

Group xyz with feature.

参数 **use_xyz** (*bool*) –Whether to use xyz.

forward(*xyz: torch.Tensor*, *new_xyz: torch.Tensor*, *features: Optional[torch.Tensor] = None*) → torch.Tensor

参数

- **xyz** (*Tensor*) –(B, N, 3) xyz coordinates of the features.
- **new_xyz** (*Tensor*) –new xyz coordinates of the features.
- **features** (*Tensor*) –(B, C, N) features to group.

返回 (B, C + 3, 1, N) Grouped feature.

返回类型 Tensor

mmcv.ops.**Linear**

alias of mmcv.ops.deprecated_wrappers.Linear_deprecated

class mmcv.ops.**MaskedConv2d**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, ...]]*, *stride: int = 1*, *padding: int = 0*, *dilation: int = 1*, *groups: int = 1*, *bias: bool = True*)

A MaskedConv2d which inherits the official Conv2d.

The masked forward doesn’t implement the backward function and only supports the stride parameter to be 1 currently.

forward(*input: torch.Tensor*, *mask: Optional[torch.Tensor] = None*) → torch.Tensor

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.

---

mmcv.ops.**MaxPool2d**

alias of mmcv.ops.deprecated_wrappers.MaxPool2d_deprecated

class mmcv.ops.**ModulatedDeformConv2d**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int]]*, *stride: int = 1*, *padding: int = 0*, *dilation: int = 1*, *groups: int = 1*, *deform_groups: int = 1*, *bias: Union[bool, str] = True*)

forward(*x: torch.Tensor*, *offset: torch.Tensor*, *mask: torch.Tensor*) → torch.Tensor

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 mmcv.ops.**ModulatedDeformConv2dPack**(**args*, ***kwargs*)

A ModulatedDeformable Conv Encapsulation that acts as normal Conv layers.

参数

- **in_channels** (*int*) –Same as nn.Conv2d.
- **out_channels** (*int*) –Same as nn.Conv2d.
- **kernel_size** (*int or tuple[int]*) –Same as nn.Conv2d.
- **stride** (*int*) –Same as nn.Conv2d, while tuple is not supported.
- **padding** (*int*) –Same as nn.Conv2d, while tuple is not supported.
- **dilation** (*int*) –Same as nn.Conv2d, while tuple is not supported.
- **groups** (*int*) –Same as nn.Conv2d.
- **bias** (*bool or str*) –If specified as *auto*, it will be decided by the norm_cfg. Bias will be set as True if norm_cfg is None, otherwise False.

forward(*x: torch.Tensor*) → torch.Tensor

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 mmcv.ops.**ModulatedDeformRoIPoolPack**(*output_size: Tuple[int, ...]*, *output_channels: int*, *deform_fc_channels: int = 1024*, *spatial_scale: float = 1.0*, *sampling_ratio: int = 0*, *gamma: float = 0.1*)

forward(*input: torch.Tensor*, *rois: torch.Tensor*) → torch.Tensor

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 mmcv.ops.**MultiScaleDeformableAttention**(*embed_dims: int = 256*, *num_heads: int = 8*, *num_levels: int = 4*, *num_points: int = 4*, *im2col_step: int = 64*, *dropout: float = 0.1*, *batch_first: bool = False*, *norm_cfg: Optional[dict] = None*, *init_cfg: Optional[*mmcv.utils.config.ConfigDict*] = None*)

An attention module used in Deformable-Detr.

Deformable DETR: Deformable Transformers for End-to-End Object Detection..

参数

- **embed_dims** (*int*) –The embedding dimension of Attention. Default: 256.
- **num_heads** (*int*) –Parallel attention heads. Default: 64.
- **num_levels** (*int*) –The number of feature map used in Attention. Default: 4.
- **num_points** (*int*) –The number of sampling points for each query in each head. Default: 4.
- **im2col_step** (*int*) –The step used in image_to_column. Default: 64.
- **dropout** (*float*) –A Dropout layer on *inp_identity*. Default: 0.1.
- **batch_first** (*bool*) –Key, Query and Value are shape of (batch, n, embed_dim) or (n, batch, embed_dim). Default to False.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: None.
- **(obj** (*init_cfg*) –*mmcv.ConfigDict*): The Config for initialization. Default: None.

forward(*query: torch.Tensor*, *key: Optional[torch.Tensor] = None*, *value: Optional[torch.Tensor] = None*, *identity: Optional[torch.Tensor] = None*, *query_pos: Optional[torch.Tensor] = None*, *key_padding_mask: Optional[torch.Tensor] = None*, *reference_points: Optional[torch.Tensor] = None*, *spatial_shapes: Optional[torch.Tensor] = None*, *level_start_index: Optional[torch.Tensor] = None*, ***kwargs*) → torch.Tensor

Forward Function of MultiScaleDeformAttention.

参数

- **query** (*torch.Tensor*) –Query of Transformer with shape (num_query, bs, embed_dims).
- **key** (*torch.Tensor*) –The key tensor with shape *(num_key, bs, embed_dims)*.
- **value** (*torch.Tensor*) –The value tensor with shape *(num_key, bs, embed_dims)*.
- **identity** (*torch.Tensor*) –The tensor used for addition, with the same shape as *query*. Default None. If None, *query* will be used.
- **query_pos** (*torch.Tensor*) –The positional encoding for *query*. Default: None.
- **key_padding_mask** (*torch.Tensor*) –ByteTensor for *query*, with shape [bs, num_key].
- **reference_points** (*torch.Tensor*) –The normalized reference points with shape (bs, num_query, num_levels, 2), all elements is range in [0, 1], top-left (0,0), bottom-right (1, 1), including padding area. or (N, Length_{query}, num_levels, 4), add additional two dimensions is (w, h) to form reference boxes.
- **spatial_shapes** (*torch.Tensor*) –Spatial shape of features in different levels. With shape (num_levels, 2), last dimension represents (h, w).
- **level_start_index** (*torch.Tensor*) –The start index of each level. A tensor has shape `(num_levels, )` and can be represented as [0, h_0*w_0, h_0*w_0+h_1*w_1, ⋯].

返回 forwarded results with shape [num_query, bs, embed_dims].

返回类型 torch.Tensor

init_weights() → None

Default initialization for Parameters of Module.

class mmcv.ops.**PSAMask**(*psa_type: str*, *mask_size: Optional[tuple] = None*)

forward(*input: torch.Tensor*) → torch.Tensor

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 mmcv.ops.**PointsSampler**(*num_point: List[int]*, *fps_mod_list: List[str] = ['D-FPS']*, *fps_sample_range_list: List[int] = [- 1]*)

Points sampling.

参数

- **num_point** (*list[int]*) –Number of sample points.
- **fps_mod_list** (*list[str], optional*) –Type of FPS method, valid mod ['F-FPS' , 'D-FPS' , 'FS'], Default: ['D-FPS']. F-FPS: using feature distances for FPS. D-FPS: using Euclidean distances of points for FPS. FS: using F-FPS and D-FPS simultaneously.
- **fps_sample_range_list** (*list[int], optional*) –Range of points to apply FPS. Default: [-1].

forward(*points_xyz: torch.Tensor*, *features: torch.Tensor*) → torch.Tensor

参数

- **points_xyz** (*torch.Tensor*) –(B, N, 3) xyz coordinates of the points.
- **features** (*torch.Tensor*) –(B, C, N) features of the points.

返回 (B, npoint, sample_num) Indices of sampled points.

返回类型 torch.Tensor

class mmcv.ops.**PrRoIPool**(*output_size: Union[int, tuple]*, *spatial_scale: float = 1.0*)

The operation of precision RoI pooling. The implementation of PrRoIPool is modified from https://github.com/vacancy/PreciseRoIPooling/

Precise RoI Pooling (PrRoIPool) is an integration-based (bilinear interpolation) average pooling method for RoI Pooling. It avoids any quantization and has a continuous gradient on bounding box coordinates. It is:

1. different from the original RoI Pooling proposed in Fast R-CNN. PrRoI Pooling uses average pooling instead of max pooling for each bin and has a continuous gradient on bounding box coordinates. That is, one can take the derivatives of some loss function w.r.t the coordinates of each RoI and optimize the RoI coordinates. 2. different from the RoI Align proposed in Mask R-CNN. PrRoI Pooling uses a full integration-based average pooling instead of sampling a constant number of points. This makes the gradient w.r.t. the coordinates continuous.

参数

- **output_size** (*Union[int, tuple]*) –h, w.

- **spatial_scale** (*float, optional*) –scale the input boxes by this number. Defaults to 1.0.

forward(*features: torch.Tensor*, *rois: torch.Tensor*) → torch.Tensor

Forward function.

参数

- **features** (*torch.Tensor*) –The feature map.
- **rois** (*torch.Tensor*) –The RoI bboxes in [tl_x, tl_y, br_x, br_y] format.

返回 The pooled results.

返回类型 torch.Tensor

class mmcv.ops.**QueryAndGroup**(*max_radius: float*, *sample_num: int*, *min_radius: float = 0.0*, *use_xyz: bool = True*, *return_grouped_xyz: bool = False*, *normalize_xyz: bool = False*, *uniform_sample: bool = False*, *return_unique_cnt: bool = False*, *return_grouped_idx: bool = False*)

Groups points with a ball query of radius.

参数

- **max_radius** (*float*) –The maximum radius of the balls. If None is given, we will use kNN sampling instead of ball query.
- **sample_num** (*int*) –Maximum number of features to gather in the ball.
- **min_radius** (*float, optional*) –The minimum radius of the balls. Default: 0.
- **use_xyz** (*bool, optional*) –Whether to use xyz. Default: True.
- **return_grouped_xyz** (*bool, optional*) –Whether to return grouped xyz. Default: False.
- **normalize_xyz** (*bool, optional*) –Whether to normalize xyz. Default: False.
- **uniform_sample** (*bool, optional*) –Whether to sample uniformly. Default: False
- **return_unique_cnt** (*bool, optional*) –Whether to return the count of unique samples. Default: False.
- **return_grouped_idx** (*bool, optional*) –Whether to return grouped idx. Default: False.

forward(*points_xyz: torch.Tensor*, *center_xyz: torch.Tensor*, *features: Optional[torch.Tensor] = None*) → Union[torch.Tensor, Tuple]

参数

- **points_xyz** (*torch.Tensor*) –(B, N, 3) xyz coordinates of the points.

- **center_xyz** (*torch.Tensor*) –(B, npoint, 3) coordinates of the centriods.
- **features** (*torch.Tensor*) –(B, C, N) The features of grouped points.

返回 (B, 3 + C, npoint, sample_num) Grouped concatenated coordinates and features of points.

返回类型 Tuple | torch.Tensor

class mmcv.ops.**RiRoIAlignRotated**(*out_size: tuple*, *spatial_scale: float*, *num_samples: int = 0*, *num_orientations: int = 8*, *clockwise: bool = False*)

Rotation-invariant RoI align pooling layer for rotated proposals.

It accepts a feature map of shape (N, C, H, W) and rois with shape (n, 6) with each roi decoded as (batch_index, center_x, center_y, w, h, angle). The angle is in radian.

The details are described in the paper ReDet: A Rotation-equivariant Detector for Aerial Object Detection.

参数

- **out_size** (*tuple*) –fixed dimensional RoI output with shape (h, w).
- **spatial_scale** (*float*) –scale the input boxes by this number
- **num_samples** (*int*) –number of inputs samples to take for each output sample. 0 to take samples densely for current models.
- **num_orientations** (*int*) –number of oriented channels.
- **clockwise** (*bool*) –If True, the angle in each proposal follows a clockwise fashion in image space, otherwise, the angle is counterclockwise. Default: False.

forward(*features: torch.Tensor*, *rois: torch.Tensor*) → torch.Tensor

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 mmcv.ops.**RoIAlign**(*output_size: tuple*, *spatial_scale: float = 1.0*, *sampling_ratio: int = 0*, *pool_mode: str = 'avg'*, *aligned: bool = True*, *use_torchvision: bool = False*)

RoI align pooling layer.

参数

- **output_size** (*tuple*) –h, w
- **spatial_scale** (*float*) –scale the input boxes by this number
- **sampling_ratio** (*int*) –number of inputs samples to take for each output sample. 0 to take samples densely for current models.

- **pool_mode** (*str, 'avg' or 'max'*) –pooling mode in each bin.
- **aligned** (*bool*) –if False, use the legacy implementation in MMDetection. If True, align the results more perfectly.
- **use_torchvision** (*bool*) –whether to use roi_align from torchvision.

注解: The implementation of RoIAlign when aligned=True is modified from https://github.com/facebookresearch/detectron2/

The meaning of aligned=True:

Given a continuous coordinate c, its two neighboring pixel indices (in our pixel model) are computed by floor(c - 0.5) and ceil(c - 0.5). For example, c=1.3 has pixel neighbors with discrete indices [0] and [1] (which are sampled from the underlying signal at continuous coordinates 0.5 and 1.5). But the original roi_align (aligned=False) does not subtract the 0.5 when computing neighboring pixel indices and therefore it uses pixels with a slightly incorrect alignment (relative to our pixel model) when performing bilinear interpolation.

With *aligned=True*, we first appropriately scale the ROI and then shift it by -0.5 prior to calling roi_align. This produces the correct neighbors;

The difference does not make a difference to the model's performance if ROIAlign is used together with conv layers.

---

forward(*input: torch.Tensor*, *rois: torch.Tensor*) → torch.Tensor

参数

- **input** –NCHW images
- **rois** –Bx5 boxes. First column is the index into N. The other 4 columns are xyxy.

class mmcv.ops.**RoIAlignRotated**(*output_size: Union[int, tuple]*, *spatial_scale: float*, *sampling_ratio: int = 0*, *aligned: bool = True*, *clockwise: bool = False*)

RoI align pooling layer for rotated proposals.

It accepts a feature map of shape (N, C, H, W) and rois with shape (n, 6) with each roi decoded as (batch_index, center_x, center_y, w, h, angle). The angle is in radian.

参数

- **output_size** (*tuple*) –h, w
- **spatial_scale** (*float*) –scale the input boxes by this number
- **sampling_ratio** (*int*) –number of inputs samples to take for each output sample. 0 to take samples densely for current models.
- **aligned** (*bool*) –if False, use the legacy implementation in MMDetection. If True, align the results more perfectly. Default: True.

- **clockwise** (*bool*) –If True, the angle in each proposal follows a clockwise fashion in image space, otherwise, the angle is counterclockwise. Default: False.

注解: The implementation of RoIAlign when aligned=True is modified from https://github.com/facebookresearch/detectron2/

The meaning of aligned=True:

Given a continuous coordinate c, its two neighboring pixel indices (in our pixel model) are computed by floor(c - 0.5) and ceil(c - 0.5). For example, c=1.3 has pixel neighbors with discrete indices [0] and [1] (which are sampled from the underlying signal at continuous coordinates 0.5 and 1.5). But the original roi_align (aligned=False) does not subtract the 0.5 when computing neighboring pixel indices and therefore it uses pixels with a slightly incorrect alignment (relative to our pixel model) when performing bilinear interpolation.

With *aligned=True*, we first appropriately scale the ROI and then shift it by -0.5 prior to calling roi_align. This produces the correct neighbors;

The difference does not make a difference to the model's performance if ROIAlign is used together with conv layers.

---

forward (*input: torch.Tensor*, *rois: torch.Tensor*) → torch.Tensor

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 mmcv.ops.**RoIAwarePool3d** (*out_size: Union[int, tuple]*, *max_pts_per_voxel: int = 128*, *mode: str = 'max'*)

Encode the geometry-specific features of each 3D proposal.

Please refer to PartA2 for more details.

参数

- **out_size** (*int or tuple*) –The size of output features. n or [n1, n2, n3].
- **max_pts_per_voxel** (*int, optional*) –The maximum number of points per voxel. Default: 128.
- **mode** (*str, optional*) –Pooling method of RoIAware, 'max' or 'avg'. Default: 'max'.

forward (*rois: torch.Tensor*, *pts: torch.Tensor*, *pts_feature: torch.Tensor*) → torch.Tensor

参数

- **rois** (*torch.Tensor*) –[N, 7], in LiDAR coordinate, (x, y, z) is the bottom center of rois.
- **pts** (*torch.Tensor*) –[npoints, 3], coordinates of input points.
- **pts_feature** (*torch.Tensor*) –[npoints, C], features of input points.

返回 Pooled features whose shape is [N, out_x, out_y, out_z, C].

返回类型 torch.Tensor

class mmcv.ops.**RoIPointPool3d**(*num_sampled_points: int = 512*)

Encode the geometry-specific features of each 3D proposal.

Please refer to Paper of PartA2 for more details.

参数 **num_sampled_points** (*int, optional*) –Number of samples in each roi. Default: 512.

forward(*points: torch.Tensor*, *point_features: torch.Tensor*, *boxes3d: torch.Tensor*) → Tuple[torch.Tensor]

参数

- **points** (*torch.Tensor*) –Input points whose shape is (B, N, C).
- **point_features** (*torch.Tensor*) –Features of input points whose shape is (B, N, C).
- **boxes3d** (*B, M, 7), Input bounding boxes whose shape is (B, M, 7*) –

返回 A tuple contains two elements. The first one is the pooled features whose shape is (B, M, 512, 3 + C). The second is an empty flag whose shape is (B, M).

返回类型 tuple[torch.Tensor]

class mmcv.ops.**RoIPool**(*output_size: Union[int, tuple]*, *spatial_scale: float = 1.0*)

forward(*input: torch.Tensor*, *rois: torch.Tensor*) → torch.Tensor

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 mmcv.ops.**SAConv2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *use_deform=False*)

SAC (Switchable Atrous Convolution)

This is an implementation of DetectoRS: Detecting Objects with Recursive Feature Pyramid and Switchable Atrous Convolution.

参数

- **in_channels** (*int*) –Number of channels in the input image
- **out_channels** (*int*) –Number of channels produced by the convolution
- **kernel_size** (*int or tuple*) –Size of the convolving kernel
- **stride** (*int or tuple, optional*) –Stride of the convolution. Default: 1
- **padding** (*int or tuple, optional*) –Zero-padding added to both sides of the input. Default: 0
- **padding_mode** (*string, optional*) –`'zeros'`, `'reflect'`, `'replicate'` or `'circular'`. Default: `'zeros'`
- **dilation** (*int or tuple, optional*) –Spacing between kernel elements. Default: 1
- **groups** (*int, optional*) –Number of blocked connections from input channels to output channels. Default: 1
- **bias** (*bool, optional*) –If `True`, adds a learnable bias to the output. Default: `True`
- **use_deform** –If `True`, replace convolution with deformable convolution. 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.

---

class mmcv.ops.**SigmoidFocalLoss**(*gamma: float*, *alpha: float*, *weight: Optional[torch.Tensor] = None*, *reduction: str = 'mean'*)

forward(*input: torch.Tensor*, *target: Union[torch.LongTensor, torch.cuda.LongTensor]*) → torch.Tensor

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 mmcv.ops.**SimpleRoIAlign**(*output_size: Tuple[int]*, *spatial_scale: float*, *aligned: bool = True*)

forward(*features: torch.Tensor*, *rois: torch.Tensor*) → torch.Tensor

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 mmcv.ops.**SoftmaxFocalLoss**(*gamma: float*, *alpha: float*, *weight: Optional[torch.Tensor] = None*, *reduction: str = 'mean'*)

forward(*input: torch.Tensor*, *target: Union[torch.LongTensor, torch.cuda.LongTensor]*) → torch.Tensor

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 mmcv.ops.**SparseConv2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *indice_key=None*)

class mmcv.ops.**SparseConv3d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *indice_key=None*)

class mmcv.ops.**SparseConvTranspose2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *indice_key=None*)

class mmcv.ops.**SparseConvTranspose3d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *indice_key=None*)

class mmcv.ops.**SparseInverseConv2d**(*in_channels*, *out_channels*, *kernel_size*, *indice_key=None*, *bias=True*)

class mmcv.ops.**SparseInverseConv3d**(*in_channels*, *out_channels*, *kernel_size*, *indice_key=None*, *bias=True*)

class mmcv.ops.**SparseMaxPool2d**(*kernel_size*, *stride=1*, *padding=0*, *dilation=1*)

class mmcv.ops.**SparseMaxPool3d**(*kernel_size*, *stride=1*, *padding=0*, *dilation=1*)

class mmcv.ops.**SparseModule**

place holder, All module subclass from this will take sptensor in SparseSequential.

class mmcv.ops.**SparseSequential**(**args*, ***kwargs*)

A sequential container. Modules will be added to it in the order they are passed in the constructor. Alternatively, an ordered dict of modules can also be passed in.

To make it easier to understand, given is a small example:

```
.. rubric:: 示例
```

```
>>> # using Sequential:
>>> from mmcv.ops import SparseSequential
>>> model = SparseSequential(
            SparseConv2d(1,20,5),
            nn.ReLU(),
            SparseConv2d(20,64,5),
            nn.ReLU()
            )
```

```
>>> # using Sequential with OrderedDict
>>> model = SparseSequential(OrderedDict([
              ('conv1', SparseConv2d(1,20,5)),
              ('relu1', nn.ReLU()),
              ('conv2', SparseConv2d(20,64,5)),
              ('relu2', nn.ReLU())
            ]))
```

```
>>> # using Sequential with kwargs(python 3.6+)
>>> model = SparseSequential(
              conv1=SparseConv2d(1,20,5),
              relu1=nn.ReLU(),
              conv2=SparseConv2d(20,64,5),
              relu2=nn.ReLU()
            )
```

forward(*input: torch.Tensor*) → torch.Tensor

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 mmcv.ops.**SubMConv2d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *indice_key=None*)

class mmcv.ops.**SubMConv3d**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *indice_key=None*)

class mmcv.ops.**SyncBatchNorm**(*num_features: int*, *eps: float = 1e-05*, *momentum: float = 0.1*, *affine: bool = True*, *track_running_stats: bool = True*, *group: Optional[int] = None*, *stats_mode: str = 'default'*)

Synchronized Batch Normalization.

参数

- **num_features** (*int*) –number of features/chennels in input tensor
- **eps** (*float, optional*) –a value added to the denominator for numerical stability. Defaults to 1e-5.
- **momentum** (*float, optional*) –the value used for the running_mean and running_var computation. Defaults to 0.1.
- **affine** (*bool, optional*) –whether to use learnable affine parameters. Defaults to True.
- **track_running_stats** (*bool, optional*) –whether to track the running mean and variance during training. When set to False, this module does not track such statistics, and initializes statistics buffers `running_mean` and `running_var` as `None`. When these buffers are `None`, this module always uses batch statistics in both training and eval modes. Defaults to True.
- **group** (*int, optional*) –synchronization of stats happen within each process group individually. By default it is synchronization across the whole world. Defaults to None.
- **stats_mode** (*str, optional*) –The statistical mode. Available options includes `'default'` and `'N'`. Defaults to 'default'. When `stats_mode=='default'`, it computes the overall statistics using those from each worker with equal weight, i.e., the statistics are synchronized and simply divied by `group`. This mode will produce inaccurate statistics when empty tensors occur. When `stats_mode=='N'`, it compute the overall statistics using the total number of batches in each worker ignoring the number of group, i.e., the statistics are synchronized and then divied by the total batch `N`. This mode is beneficial when empty tensors occur during training, as it average the total mean by the real number of batch.

forward(*input: torch.Tensor*) → torch.Tensor

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 mmcv.ops.**TINShift**

Temporal Interlace Shift.

Temporal Interlace shift is a differentiable temporal-wise frame shifting which is proposed in "Temporal Interlacing Network"

Please refer to Temporal Interlacing Network for more details.

Code is modified from https://github.com/mit-han-lab/temporal-shift-module

forward(*input*, *shift*)

Perform temporal interlace shift.

参数

- **input** (*torch.Tensor*) –Feature map with shape [N, num_segments, C, H * W].
- **shift** (*torch.Tensor*) –Shift tensor with shape [N, num_segments].

返回 Feature map after temporal interlace shift.

class mmcv.ops.**Voxelization**(*voxel_size: List*, *point_cloud_range: List*, *max_num_points: int*, *max_voxels: Union[tuple, int] = 20000*, *deterministic: bool = True*)

Convert kitti points(N, >=3) to voxels.

Please refer to Point-Voxel CNN for Efficient 3D Deep Learning for more details.

参数

- **voxel_size** (*tuple or float*) –The size of voxel with the shape of [3].
- **point_cloud_range** (*tuple or float*) –The coordinate range of voxel with the shape of [6].
- **max_num_points** (*int*) –maximum points contained in a voxel. if max_points=-1, it means using dynamic_voxelize.
- **max_voxels** (*int, optional*) –maximum voxels this function create. for second, 20000 is a good choice. Users should shuffle points before call this function because max_voxels may drop points. Default: 20000.

forward(*input: torch.Tensor*) → torch.Tensor

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.

---

mmcv.ops.**batched_nms**(*boxes: torch.Tensor*, *scores: torch.Tensor*, *idxs: torch.Tensor*, *nms_cfg: Optional[Dict]*, *class_agnostic: bool = False*) → Tuple[torch.Tensor, torch.Tensor]

Performs non-maximum suppression in a batched fashion.

Modified from torchvision/ops/boxes.py#L39. In order to perform NMS independently per class, we add an offset to all the boxes. The offset is dependent only on the class idx, and is large enough so that boxes from different classes do not overlap.

注解： In v1.4.1 and later, `batched_nms` supports skipping the NMS and returns sorted raw results when *nms_cfg* is None.

参数

- **boxes** (*torch.Tensor*) –boxes in shape (N, 4) or (N, 5).
- **scores** (*torch.Tensor*) –scores in shape (N,).
- **idxs** (*torch.Tensor*) –each index value correspond to a bbox cluster, and NMS will not be applied between elements of different idxs, shape (N,).
- **nms_cfg** (*dict | optional*) –Supports skipping the nms when *nms_cfg* is None, otherwise it should specify nms type and other parameters like *iou_thr*. Possible keys includes the following.
  - iou_threshold (float): IoU threshold used for NMS.
  - split_thr (float): threshold number of boxes. In some cases the number of boxes is large (e.g., 200k). To avoid OOM during training, the users could set *split_thr* to a small value. If the number of boxes is greater than the threshold, it will perform NMS on each group of boxes separately and sequentially. Defaults to 10000.
- **class_agnostic** (*bool*) –if true, nms is class agnostic, i.e. IoU thresholding happens over all boxes, regardless of the predicted class. Defaults to False.

返回

kept dets and indice.

- boxes (Tensor): Bboxes with score after nms, has shape (num_bboxes, 5). last dimension 5 arrange as (x1, y1, x2, y2, score)
- keep (Tensor): The indices of remaining boxes in input boxes.

返回类型 tuple

mmcv.ops.**bbox_overlaps**(*bboxes1: torch.Tensor*, *bboxes2: torch.Tensor*, *mode: str = 'iou'*, *aligned: bool = False*, *offset: int = 0*) → torch.Tensor

Calculate overlap between two set of bboxes.

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

参数

- **bboxes1** (*torch.Tensor*) –shape (m, 4) in <x1, y1, x2, y2> format or empty.
- **bboxes2** (*torch.Tensor*) –shape (n, 4) in <x1, y1, x2, y2> format or empty. If aligned is `True`, then m and n must be equal.
- **mode** (*str*) – "iou" (intersection over union) or iof (intersection over foreground).

返回 Return the ious betweens boxes. If `aligned` is `False`, the shape of ious is (m, n) else (m, 1).

返回类型 torch.Tensor

示例

```
>>> bboxes1 = torch.FloatTensor([
>>>     [0, 0, 10, 10],
>>>     [10, 10, 20, 20],
>>>     [32, 32, 38, 42],
>>> ])
>>> bboxes2 = torch.FloatTensor([
>>>     [0, 0, 10, 20],
>>>     [0, 10, 10, 19],
>>>     [10, 10, 20, 20],
>>> ])
>>> bbox_overlaps(bboxes1, bboxes2)
tensor([[0.5000, 0.0000, 0.0000],
        [0.0000, 0.0000, 1.0000],
        [0.0000, 0.0000, 0.0000]])
```

示例

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

mmcv.ops.**box_iou_rotated**(*bboxes1: torch.Tensor*, *bboxes2: torch.Tensor*, *mode: str = 'iou'*, *aligned: bool = False*, *clockwise: bool = True*) → torch.Tensor

Return intersection-over-union (Jaccard index) of boxes.

Both sets of boxes are expected to be in (x_center, y_center, width, height, angle) format.

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

注解： The operator assumes:

1) The positive direction along x axis is left -> right.

2) The positive direction along y axis is top -> down.

3) The w border is in parallel with x axis when angle = 0.

However, there are 2 opposite definitions of the positive angular direction, clockwise (CW) and counter-clockwise (CCW). MMCV supports both definitions and uses CW by default.

Please set `clockwise=False` if you are using the CCW definition.

The coordinate system when `clockwise` is `True` (default)

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

In such coordination system the rotation matrix is

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

The coordinates of the corner point A can be calculated as:

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

The coordinate system when `clockwise` is `False`

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

In such coordination system the rotation matrix is

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

The coordinates of the corner point A can be calculated as:

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

参数

- **boxes1** (*torch.Tensor*) –rotated bboxes 1. It has shape (N, 5), indicating (x, y, w, h, theta) for each row. Note that theta is in radian.
- **boxes2** (*torch.Tensor*) –rotated bboxes 2. It has shape (M, 5), indicating (x, y, w, h, theta) for each row. Note that theta is in radian.
- **mode** (*str*) – "iou" (intersection over union) or iof (intersection over foreground).
- **clockwise** (*bool*) –flag indicating whether the positive angular orientation is clockwise. default True. *New in version 1.4.3.*

返回 Return the ious betweens boxes. If `aligned` is `False`, the shape of ious is (N, M) else (N,).

返回类型 torch.Tensor

mmcv.ops.**boxes_iou3d**(*boxes_a: torch.Tensor*, *boxes_b: torch.Tensor*) → torch.Tensor

Calculate boxes 3D IoU.

参数

- **boxes_a** (*torch.Tensor*) –Input boxes a with shape (M, 7).
- **boxes_b** (*torch.Tensor*) –Input boxes b with shape (N, 7).

返回 3D IoU result with shape (M, N).

返回类型 torch.Tensor

mmcv.ops.**boxes_iou_bev**(*boxes_a: torch.Tensor*, *boxes_b: torch.Tensor*) → torch.Tensor

Calculate boxes IoU in the Bird’ s Eye View.

参数

- **boxes_a** (*torch.Tensor*) –Input boxes a with shape (M, 5) ([x1, y1, x2, y2, ry]).
- **boxes_b** (*torch.Tensor*) –Input boxes b with shape (N, 5) ([x1, y1, x2, y2, ry]).

返回 IoU result with shape (M, N).

返回类型 torch.Tensor

mmcv.ops.**boxes_overlap_bev**(*boxes_a: torch.Tensor*, *boxes_b: torch.Tensor*) → torch.Tensor

Calculate boxes BEV overlap.

参数

- **boxes_a** (*torch.Tensor*) –Input boxes a with shape (M, 7).
- **boxes_b** (*torch.Tensor*) –Input boxes b with shape (N, 7).

返回 BEV overlap result with shape (M, N).

返回类型 torch.Tensor

mmcv.ops.**contour_expand**(*kernel_mask: Union[numpy.array, torch.Tensor]*, *internal_kernel_label: Union[numpy.array, torch.Tensor]*, *min_kernel_area: int*, *kernel_num: int*) → list

Expand kernel contours so that foreground pixels are assigned into instances.

参数

- **kernel_mask** (*np.array or torch.Tensor*) –The instance kernel mask with size hxw.
- **internal_kernel_label** (*np.array or torch.Tensor*) –The instance internal kernel label with size hxw.
- **min_kernel_area** (*int*) –The minimum kernel area.
- **kernel_num** (*int*) –The instance kernel number.

返回 The instance index map with size hxw.

返回类型 list

mmcv.ops.**convex_giou**(*pointsets: torch.Tensor*, *polygons: torch.Tensor*) → Tuple[torch.Tensor, torch.Tensor]

Return generalized intersection-over-union (Jaccard index) between point sets and polygons.

参数

- **pointsets** (*torch.Tensor*) –It has shape (N, 18), indicating (x1, y1, x2, y2, ⋯, x9, y9) for each row.
- **polygons** (*torch.Tensor*) –It has shape (N, 8), indicating (x1, y1, x2, y2, x3, y3, x4, y4) for each row.

返回 The first element is the gious between point sets and polygons with the shape (N,). The second element is the gradient of point sets with the shape (N, 18).

返回类型 tuple[torch.Tensor, torch.Tensor]

mmcv.ops.**convex_iou**(*pointsets: torch.Tensor*, *polygons: torch.Tensor*) → torch.Tensor

Return intersection-over-union (Jaccard index) between point sets and polygons.

参数

- **pointsets** (*torch.Tensor*) –It has shape (N, 18), indicating (x1, y1, x2, y2, ⋯, x9, y9) for each row.
- **polygons** (*torch.Tensor*) –It has shape (K, 8), indicating (x1, y1, x2, y2, x3, y3, x4, y4) for each row.

返回 Return the ious between point sets and polygons with the shape (N, K).

返回类型 torch.Tensor

mmcv.ops.**diff_iou_rotated_2d**(*box1: torch.Tensor*, *box2: torch.Tensor*) → torch.Tensor

Calculate differentiable iou of rotated 2d boxes.

参数

- **box1** (*Tensor*) –(B, N, 5) First box.
- **box2** (*Tensor*) –(B, N, 5) Second box.

返回 (B, N) IoU.

返回类型 Tensor

mmcv.ops.**diff_iou_rotated_3d**(*box3d1: torch.Tensor*, *box3d2: torch.Tensor*) → torch.Tensor

Calculate differentiable iou of rotated 3d boxes.

参数

- **box3d1** (*Tensor*) –(B, N, 3+3+1) First box (x,y,z,w,h,l,alpha).
- **box3d2** (*Tensor*) –(B, N, 3+3+1) Second box (x,y,z,w,h,l,alpha).

返回 (B, N) IoU.

返回类型 Tensor

mmcv.ops.**fused_bias_leakyrelu**(*input: torch.Tensor*, *bias: torch.nn.parameter.Parameter*, *negative_slope: float = 0.2*, *scale: float = 1.4142135623730951*) → torch.Tensor

Fused bias leaky ReLU function.

This function is introduced in the StyleGAN2: Analyzing and Improving the Image Quality of StyleGAN

The bias term comes from the convolution operation. In addition, to keep the variance of the feature map or gradients unchanged, they also adopt a scale similarly with Kaiming initialization. However, since the $1 + alpha^2$ is too small, we can just ignore it. Therefore, the final scale is just $\sqrt{2}$. Of course, you may change it with your own scale.

参数

- **input** (*torch.Tensor*) –Input feature map.
- **bias** (*nn.Parameter*) –The bias from convolution operation.
- **negative_slope** (*float, optional*) –Same as nn.LeakyRelu. Defaults to 0.2.
- **scale** (*float, optional*) –A scalar to adjust the variance of the feature map. Defaults to 2**0.5.

返回 Feature map after non-linear activation.

返回类型 torch.Tensor

mmcv.ops.**min_area_polygons**(*pointsets: torch.Tensor*) → torch.Tensor

Find the smallest polygons that surrounds all points in the point sets.

参数 **pointsets** (*Tensor*) –point sets with shape (N, 18).

返回 Return the smallest polygons with shape (N, 8).

返回类型 torch.Tensor

mmcv.ops.**nms**(*boxes: Union[torch.Tensor, numpy.ndarray]*, *scores: Union[torch.Tensor, numpy.ndarray]*, *iou_threshold: float*, *offset: int = 0*, *score_threshold: float = 0*, *max_num: int = - 1*) → Tuple[Union[torch.Tensor, numpy.ndarray], Union[torch.Tensor, numpy.ndarray]]

Dispatch to either CPU or GPU NMS implementations.

The input can be either torch tensor or numpy array. GPU NMS will be used if the input is gpu tensor, otherwise CPU NMS will be used. The returned type will always be the same as inputs.

参数

- **boxes** (*torch.Tensor or np.ndarray*) –boxes in shape (N, 4).
- **scores** (*torch.Tensor or np.ndarray*) –scores in shape (N,).
- **iou_threshold** (*float*) –IoU threshold for NMS.
- **offset** (*int, 0 or 1*) –boxes' width or height is (x2 - x1 + offset).

- **score_threshold** (*float*) –score threshold for NMS.
- **max_num** (*int*) –maximum number of boxes after NMS.

返回 kept dets (boxes and scores) and indice, which always have the same data type as the input.

返回类型 tuple

示例

```
>>> boxes = np.array([[49.1, 32.4, 51.0, 35.9],
>>>                   [49.3, 32.9, 51.0, 35.3],
>>>                   [49.2, 31.8, 51.0, 35.4],
>>>                   [35.1, 11.5, 39.1, 15.7],
>>>                   [35.6, 11.8, 39.3, 14.2],
>>>                   [35.3, 11.5, 39.9, 14.5],
>>>                   [35.2, 11.7, 39.7, 15.7]], dtype=np.float32)
>>> scores = np.array([0.9, 0.9, 0.5, 0.5, 0.5, 0.4, 0.3],                dtype=np.
↪float32)
>>> iou_threshold = 0.6
>>> dets, inds = nms(boxes, scores, iou_threshold)
>>> assert len(inds) == len(dets) == 3
```

mmcv.ops.**nms3d**(*boxes: torch.Tensor*, *scores: torch.Tensor*, *iou_threshold: float*) → torch.Tensor

3D NMS function GPU implementation (for BEV boxes).

参数

- **boxes** (*torch.Tensor*) –Input boxes with the shape of (N, 7) ([x, y, z, dx, dy, dz, heading]).
- **scores** (*torch.Tensor*) –Scores of boxes with the shape of (N).
- **iou_threshold** (*float*) –Overlap threshold of NMS.

返回 Indexes after NMS.

返回类型 torch.Tensor

mmcv.ops.**nms3d_normal**(*boxes: torch.Tensor*, *scores: torch.Tensor*, *iou_threshold: float*) → torch.Tensor

Normal 3D NMS function GPU implementation. The overlap of two boxes for IoU calculation is defined as the exact overlapping area of the two boxes WITH their yaw angle set to 0.

参数

- **boxes** (*torch.Tensor*) –Input boxes with shape (N, 7). ([x, y, z, dx, dy, dz, heading]).
- **scores** (*torch.Tensor*) –Scores of predicted boxes with shape (N).
- **iou_threshold** (*float*) –Overlap threshold of NMS.

返回 Remaining indices with scores in descending order.

返回类型 torch.Tensor

mmcv.ops.**nms_bev**(*boxes: torch.Tensor*, *scores: torch.Tensor*, *thresh: float*, *pre_max_size: Optional[int] = None*, *post_max_size: Optional[int] = None*) → torch.Tensor

NMS function GPU implementation (for BEV boxes).

The overlap of two boxes for IoU calculation is defined as the exact overlapping area of the two boxes. In this function, one can also set `pre_max_size` and `post_max_size`. :param boxes: Input boxes with the shape of (N, 5)

([x1, y1, x2, y2, ry]).

参数

- **scores** (*torch.Tensor*) –Scores of boxes with the shape of (N,).
- **thresh** (*float*) –Overlap threshold of NMS.
- **pre_max_size** (*int, optional*) –Max size of boxes before NMS. Default: None.
- **post_max_size** (*int, optional*) –Max size of boxes after NMS. Default: None.

返回 Indexes after NMS.

返回类型 torch.Tensor

mmcv.ops.**nms_match**(*dets: Union[torch.Tensor, numpy.ndarray]*, *iou_threshold: float*) → List[Union[torch.Tensor, numpy.ndarray]]

Matched dets into different groups by NMS.

NMS match is Similar to NMS but when a bbox is suppressed, nms match will record the indice of suppressed bbox and form a group with the indice of kept bbox. In each group, indice is sorted as score order.

参数

- **dets** (*torch.Tensor | np.ndarray*) –Det boxes with scores, shape (N, 5).
- **iou_threshold** (*float*) –IoU thresh for NMS.

返回 The outer list corresponds different matched group, the inner Tensor corresponds the indices for a group in score order.

返回类型 list[torch.Tensor | np.ndarray]

mmcv.ops.**nms_normal_bev**(*boxes: torch.Tensor*, *scores: torch.Tensor*, *thresh: float*) → torch.Tensor

Normal NMS function GPU implementation (for BEV boxes).

The overlap of two boxes for IoU calculation is defined as the exact overlapping area of the two boxes WITH their yaw angle set to 0. :param boxes: Input boxes with shape (N, 5)

([x1, y1, x2, y2, ry]).

参数

- **scores** (*torch.Tensor*) –Scores of predicted boxes with shape (N,).
- **thresh** (*float*) –Overlap threshold of NMS.

返回 Remaining indices with scores in descending order.

返回类型 torch.Tensor

mmcv.ops.**nms_rotated**(*dets: torch.Tensor*, *scores: torch.Tensor*, *iou_threshold: float*, *labels: Optional[torch.Tensor] = None*, *clockwise: bool = True*) → Tuple[torch.Tensor, torch.Tensor]

Performs non-maximum suppression (NMS) on the rotated boxes according to their intersection-over-union (IoU).

Rotated NMS iteratively removes lower scoring rotated boxes which have an IoU greater than iou_threshold with another (higher scoring) rotated box.

参数

- **dets** (*torch.Tensor*) –Rotated boxes in shape (N, 5). They are expected to be in (x_ctr, y_ctr, width, height, angle_radian) format.
- **scores** (*torch.Tensor*) –scores in shape (N,).
- **iou_threshold** (*float*) –IoU thresh for NMS.
- **labels** (*torch.Tensor, optional*) –boxes' label in shape (N,).
- **clockwise** (*bool*) –flag indicating whether the positive angular orientation is clockwise. default True. *New in version 1.4.3.*

返回 kept dets(boxes and scores) and indice, which is always the same data type as the input.

返回类型 tuple

mmcv.ops.**pixel_group**(*score: Union[numpy.ndarray, torch.Tensor]*, *mask: Union[numpy.ndarray, torch.Tensor]*, *embedding: Union[numpy.ndarray, torch.Tensor]*, *kernel_label: Union[numpy.ndarray, torch.Tensor]*, *kernel_contour: Union[numpy.ndarray, torch.Tensor]*, *kernel_region_num: int*, *distance_threshold: float*) → List[List[float]]

Group pixels into text instances, which is widely used text detection methods.

参数

- **score** (*np.array or torch.Tensor*) –The foreground score with size hxw.
- **mask** (*np.array or Tensor*) –The foreground mask with size hxw.
- **embedding** (*np.array or torch.Tensor*) –The embedding with size hxwxc to distinguish instances.
- **kernel_label** (*np.array or torch.Tensor*) –The instance kernel index with size hxw.

- **kernel_contour** (*np.array or torch.Tensor*) –The kernel contour with size hxw.
- **kernel_region_num** (*int*) –The instance kernel region number.
- **distance_threshold** (*float*) –The embedding distance threshold between kernel and pixel in one instance.

返回 The instance coordinates and attributes list. Each element consists of averaged confidence, pixel number, and coordinates (x_i, y_i for all pixels) in order.

返回类型 list[list[float]]

mmcv.ops.**point_sample**(*input: torch.Tensor*, *points: torch.Tensor*, *align_corners: bool = False*, ***kwargs*) → torch.Tensor

A wrapper around `grid_sample()` to support 3D point_coords tensors Unlike `torch.nn.functional.grid_sample()` it assumes point_coords to lie inside `[0, 1] x [0, 1]` square.

参数

- **input** (*torch.Tensor*) –Feature map, shape (N, C, H, W).
- **points** (*torch.Tensor*) –Image based absolute point coordinates (normalized), range [0, 1] x [0, 1], shape (N, P, 2) or (N, Hgrid, Wgrid, 2).
- **align_corners** (*bool, optional*) –Whether align_corners. Default: False

返回 Features of *point* on *input*, shape (N, C, P) or (N, C, Hgrid, Wgrid).

返回类型 torch.Tensor

mmcv.ops.**points_in_boxes_all**(*points: torch.Tensor*, *boxes: torch.Tensor*) → torch.Tensor

Find all boxes in which each point is (CUDA).

参数

- **points** (*torch.Tensor*) –[B, M, 3], [x, y, z] in LiDAR/DEPTH coordinate
- **boxes** (*torch.Tensor*) –[B, T, 7], num_valid_boxes <= T, [x, y, z, x_size, y_size, z_size, rz], (x, y, z) is the bottom center.

返回 Return the box indices of points with the shape of (B, M, T). Default background = 0.

返回类型 torch.Tensor

mmcv.ops.**points_in_boxes_cpu**(*points: torch.Tensor*, *boxes: torch.Tensor*) → torch.Tensor

Find all boxes in which each point is (CPU). The CPU version of *points_in_boxes_all()*.

参数

- **points** (*torch.Tensor*) –[B, M, 3], [x, y, z] in LiDAR/DEPTH coordinate
- **boxes** (*torch.Tensor*) –[B, T, 7], num_valid_boxes <= T, [x, y, z, x_size, y_size, z_size, rz], (x, y, z) is the bottom center.

返回 Return the box indices of points with the shape of (B, M, T). Default background = 0.

返回类型 torch.Tensor

mmcv.ops.**points_in_boxes_part**(*points: torch.Tensor*, *boxes: torch.Tensor*) → torch.Tensor

Find the box in which each point is (CUDA).

参数

- **points** (*torch.Tensor*) –[B, M, 3], [x, y, z] in LiDAR/DEPTH coordinate.
- **boxes** (*torch.Tensor*) –[B, T, 7], num_valid_boxes <= T, [x, y, z, x_size, y_size, z_size, rz] in LiDAR/DEPTH coordinate, (x, y, z) is the bottom center.

返回 Return the box indices of points with the shape of (B, M). Default background = -1.

返回类型 torch.Tensor

mmcv.ops.**points_in_polygons**(*points: torch.Tensor*, *polygons: torch.Tensor*) → torch.Tensor

Judging whether points are inside polygons, which is used in the ATSS assignment for the rotated boxes.

It should be noted that when the point is just at the polygon boundary, the judgment will be inaccurate, but the effect on assignment is limited.

参数

- **points** (*torch.Tensor*) –It has shape (B, 2), indicating (x, y). M means the number of predicted points.
- **polygons** (*torch.Tensor*) –It has shape (M, 8), indicating (x1, y1, x2, y2, x3, y3, x4, y4). M means the number of ground truth polygons.

返回 Return the result with the shape of (B, M), 1 indicates that the point is inside the polygon, 0 indicates that the point is outside the polygon.

返回类型 torch.Tensor

mmcv.ops.**rel_roi_point_to_rel_img_point**(*rois: torch.Tensor*, *rel_roi_points: torch.Tensor*, *img: Union[tuple, torch.Tensor]*, *spatial_scale: float = 1.0*) → torch.Tensor

Convert roi based relative point coordinates to image based absolute point coordinates.

参数

- **rois** (*torch.Tensor*) –RoIs or BBoxes, shape (N, 4) or (N, 5)
- **rel_roi_points** (*torch.Tensor*) –Point coordinates inside RoI, relative to RoI, location, range (0, 1), shape (N, P, 2)
- **img** (*tuple or torch.Tensor*) –(height, width) of image or feature map.
- **spatial_scale** (*float, optional*) –Scale points by this factor. Default: 1.

返回 Image based relative point coordinates for sampling, shape (N, P, 2).

返回类型 torch.Tensor

mmcv.ops.**scatter_nd**(*indices: torch.Tensor*, *updates: torch.Tensor*, *shape: torch.Tensor*) → torch.Tensor

pytorch edition of tensorflow scatter_nd.

this function don't contain except handle code. so use this carefully when indice repeats, don't support repeat add which is supported in tensorflow.

mmcv.ops.**soft_nms**(*boxes: Union[torch.Tensor, numpy.ndarray]*, *scores: Union[torch.Tensor, numpy.ndarray]*, *iou_threshold: float = 0.3*, *sigma: float = 0.5*, *min_score: float = 0.001*, *method: str = 'linear'*, *offset: int = 0*) → Tuple[Union[torch.Tensor, numpy.ndarray], Union[torch.Tensor, numpy.ndarray]]

Dispatch to only CPU Soft NMS implementations.

The input can be either a torch tensor or numpy array. The returned type will always be the same as inputs.

参数

- **boxes** (*torch.Tensor or np.ndarray*) –boxes in shape (N, 4).
- **scores** (*torch.Tensor or np.ndarray*) –scores in shape (N,).
- **iou_threshold** (*float*) –IoU threshold for NMS.
- **sigma** (*float*) –hyperparameter for gaussian method
- **min_score** (*float*) –score filter threshold
- **method** (*str*) –either 'linear' or 'gaussian'
- **offset** (*int, 0 or 1*) –boxes' width or height is (x2 - x1 + offset).

返回 kept dets (boxes and scores) and indice, which always have the same data type as the input.

返回类型 tuple

示例

```
>>> boxes = np.array([[4., 3., 5., 3.],
>>>                   [4., 3., 5., 4.],
>>>                   [3., 1., 3., 1.],
>>>                   [3., 1., 3., 1.],
>>>                   [3., 1., 3., 1.],
>>>                   [3., 1., 3., 1.]], dtype=np.float32)
>>> scores = np.array([0.9, 0.9, 0.5, 0.5, 0.4, 0.0], dtype=np.float32)
>>> iou_threshold = 0.6
>>> dets, inds = soft_nms(boxes, scores, iou_threshold, sigma=0.5)
>>> assert len(inds) == len(dets) == 5
```

mmcv.ops.**upfirdn2d**(*input: torch.Tensor*, *kernel: torch.Tensor*, *up: Union[int, tuple] = 1*, *down: Union[int, tuple] = 1*, *pad: tuple = (0, 0)*) → torch.Tensor

UpFRIDn for 2d features.

UpFIRDn is short for upsample, apply FIR filter and downsample. More details can be found in: https://www.mathworks.com/help/signal/ref/upfirdn.html

参数

- **input** (*torch.Tensor*) –Tensor with shape of (n, c, h, w).
- **kernel** (*torch.Tensor*) –Filter kernel.
- **up** (*int | tuple[int], optional*) –Upsampling factor. If given a number, we will use this factor for the both height and width side. Defaults to 1.
- **down** (*int | tuple[int], optional*) –Downsampling factor. If given a number, we will use this factor for the both height and width side. Defaults to 1.
- **pad** (*tuple[int], optional*) –Padding for tensors, (x_pad, y_pad) or (x_pad_0, x_pad_1, y_pad_0, y_pad_1). Defaults to (0, 0).

返回 Tensor after UpFIRDn.

返回类型 torch.Tensor

CHAPTER 36

Indices and tables

- genindex
- search

Python 模块索引

m

mmcv.arraymisc, 155
mmcv.cnn, 183
mmcv.engine, 241
mmcv.fileio, 117
mmcv.image, 127
mmcv.ops, 243
mmcv.runner, 205
mmcv.utils, 161
mmcv.video, 147
mmcv.visualization, 157

索引

A

add_params() (*mmcv.runner.DefaultOptimizerConstructor* 方法), 214
adjust_brightness() (在 *mmcv.image* 模块中), 127
adjust_color() (在 *mmcv.image* 模块中), 127
adjust_contrast() (在 *mmcv.image* 模块中), 128
adjust_hue() (在 *mmcv.image* 模块中), 128
adjust_lighting() (在 *mmcv.image* 模块中), 129
adjust_sharpness() (在 *mmcv.image* 模块中), 129
after_epoch() (*mmcv.runner.SyncBuffersHook* 方法), 232
after_train_epoch() (*mmcv.runner.EMAHook* 方法), 217
after_train_epoch() (*mmcv.runner.EvalHook* 方法), 220
after_train_iter() (*mmcv.runner.EMAHook* 方法), 217
after_train_iter() (*mmcv.runner.EvalHook* 方法), 220
after_train_iter() (*mmcv.runner.Fp16OptimizerHook* 方法), 221
after_train_iter() (*mmcv.runner.GradientCumulativeFp16OptimizerHook* 方法), 222
AlexNet (*mmcv.cnn* 中的类), 183
allreduce_grads() (在 *mmcv.runner* 模块中), 234
allreduce_params() (在 *mmcv.runner* 模块中), 234
assert_attrs_equal() (在 *mmcv.utils* 模块中), 172
assert_dict_contains_subset() (在 *mmcv.utils* 模块中), 172
assert_dict_has_keys() (在 *mmcv.utils* 模块中), 172
assert_is_norm_layer() (在 *mmcv.utils* 模块中), 172
assert_keys_equal() (在 *mmcv.utils* 模块中), 172
assert_params_all_zeros() (在 *mmcv.utils* 模块中), 173
auto_argparser() (*mmcv.utils.Config* 静态方法), 164
auto_contrast() (在 *mmcv.image* 模块中), 130
auto_fp16() (在 *mmcv.runner* 模块中), 234

B

BaseModule (*mmcv.runner* 中的类), 205
BaseRunner (*mmcv.runner* 中的类), 205
BaseStorageBackend (*mmcv.fileio* 中的类), 117
batched_nms() (在 *mmcv.ops* 模块中), 267
bbox_overlaps() (在 *mmcv.ops* 模块中), 268
before_run() (*mmcv.runner.EMAHook* 方法), 217
before_run() (*mmcv.runner.Fp16OptimizerHook* 方法), 221
before_train_epoch() (*mmcv.runner.EMAHook* 方法), 218
before_train_epoch() (*mmcv.runner.EvalHook* 方法), 220
before_train_iter() (*mmcv.runner.EvalHook* 方

法), 220
bgr2gray() (在 mmcv.image 模块中), 130
bgr2hls() (在 mmcv.image 模块中), 130
bgr2hsv() (在 mmcv.image 模块中), 131
bgr2rgb() (在 mmcv.image 模块中), 131
bgr2ycbcr() (在 mmcv.image 模块中), 131
bias_init_with_prob() (在 mmcv.cnn 模块中), 198
BorderAlign (mmcv.ops 中的类), 243
box_iou_rotated() (在 mmcv.ops 模块中), 269
boxes_iou3d() (在 mmcv.ops 模块中), 270
boxes_iou_bev() (在 mmcv.ops 模块中), 271
boxes_overlap_bev() (在 mmcv.ops 模块中), 271
build_activation_layer() (在 mmcv.cnn 模块中), 198
build_conv_layer() (在 mmcv.cnn 模块中), 199
build_from_cfg() (在 mmcv.utils 模块中), 173
build_model_from_cfg() (在 mmcv.cnn 模块中), 199
build_norm_layer() (在 mmcv.cnn 模块中), 199
build_padding_layer() (在 mmcv.cnn 模块中), 200
build_plugin_layer() (在 mmcv.cnn 模块中), 200
build_upsample_layer() (在 mmcv.cnn 模块中), 200
BuildExtension (mmcv.utils 中的类), 161

C

Caffe2XavierInit (mmcv.cnn 中的类), 183
call_hook() (mmcv.runner.BaseRunner 方法), 206
CARAFE (mmcv.ops 中的类), 244
CARAFENaive (mmcv.ops 中的类), 244
CARAFEPack (mmcv.ops 中的类), 244
check_prerequisites() (在 mmcv.utils 模块中), 174
check_python_script() (在 mmcv.utils 模块中), 174
check_time() (在 mmcv.utils 模块中), 174
CheckpointHook (mmcv.runner 中的类), 208
CheckpointLoader (mmcv.runner 中的类), 209
clahe() (在 mmcv.image 模块中), 131
ClearMLLoggerHook (mmcv.runner 中的类), 210
client (mmcv.fileio.FileClient 属性), 118
collect_env() (在 mmcv.utils 模块中), 174
collect_results_cpu() (在 mmcv.engine 模块中), 241
collect_results_gpu() (在 mmcv.engine 模块中), 241
Color (mmcv.visualization 中的类), 157
color_val() (在 mmcv.visualization 模块中), 157
concat_list() (在 mmcv.utils 模块中), 175
concat_video() (在 mmcv.video 模块中), 149
Config (mmcv.utils 中的类), 163
ConfigDict (mmcv.utils 中的类), 165
ConstantInit (mmcv.cnn 中的类), 183
ContextBlock (mmcv.cnn 中的类), 184
contour_expand() (在 mmcv.ops 模块中), 271
Conv2d (mmcv.cnn 中的类), 184
Conv2d() (在 mmcv.ops 模块中), 245
Conv3d (mmcv.cnn 中的类), 184
ConvAWS2d (mmcv.cnn 中的类), 185
convert_video() (在 mmcv.video 模块中), 149
convex_giou() (在 mmcv.ops 模块中), 272
convex_iou() (在 mmcv.ops 模块中), 272
ConvModule (mmcv.cnn 中的类), 186
ConvTranspose2d (mmcv.cnn 中的类), 187
ConvTranspose2d() (在 mmcv.ops 模块中), 245
ConvTranspose3d (mmcv.cnn 中的类), 187
ConvWS2d (mmcv.cnn 中的类), 188
copy_grads_to_fp32() (mmcv.runner.Fp16OptimizerHook 方法), 221
copy_params_to_fp16() (mmcv.runner.Fp16OptimizerHook 方法), 221
CornerPool (mmcv.ops 中的类), 245
Correlation (mmcv.ops 中的类), 246
CosineAnnealingLrUpdaterHook (mmcv.runner 中的类), 210
CosineAnnealingMomentumUpdaterHook (mmcv.runner 中的类), 210
CosineRestartLrUpdaterHook (mmcv.runner 中的类), 211

CppExtension() (在 mmcv.utils 模块中), 165
CrissCrossAttention (mmcv.ops 中的类), 247
CUDAExtension() (在 mmcv.utils 模块中), 162
current_frame() (mmcv.video.VideoReader 方法), 147
current_lr() (mmcv.runner.BaseRunner 方法), 206
current_momentum() (mmcv.runner.BaseRunner 方法), 206
cut_video() (在 mmcv.video 模块中), 150
cutout() (在 mmcv.image 模块中), 132
cvt2frames() (mmcv.video.VideoReader 方法), 147
CyclicLrUpdaterHook (mmcv.runner 中的类), 211
CyclicMomentumUpdaterHook (mmcv.runner 中的类), 212

D

DataLoader (mmcv.utils 中的类), 166
DefaultOptimizerConstructor (mmcv.runner 中的类), 212
DefaultRunnerConstructor (mmcv.runner 中的类), 214
DeformConv2d (mmcv.ops 中的类), 247
DeformConv2dPack (mmcv.ops 中的类), 248
DeformRoIPool (mmcv.ops 中的类), 249
DeformRoIPoolPack (mmcv.ops 中的类), 250
deprecated_api_warning() (在 mmcv.utils 模块中), 175
DepthwiseSeparableConvModule (mmcv.cnn 中的类), 188
dequantize() (在 mmcv.arraymisc 模块中), 155
dequantize_flow() (在 mmcv.video 模块中), 150
dict_from_file() (在 mmcv.fileio 模块中), 122
DictAction (mmcv.utils 中的类), 168
diff_iou_rotated_2d() (在 mmcv.ops 模块中), 272
diff_iou_rotated_3d() (在 mmcv.ops 模块中), 272
digit_version() (在 mmcv.utils 模块中), 175
DistEvalHook (mmcv.runner 中的类), 215
DistSamplerSeedHook (mmcv.runner 中的类), 216
dump() (mmcv.utils.Config 方法), 164
dump() (在 mmcv.fileio 模块中), 123
DvcliveLoggerHook (mmcv.runner 中的类), 216
DynamicScatter (mmcv.ops 中的类), 250

E

EMAHook (mmcv.runner 中的类), 217
epoch (mmcv.runner.BaseRunner property), 206
EpochBasedRunner (mmcv.runner 中的类), 218
EvalHook (mmcv.runner 中的类), 218
evaluate() (mmcv.runner.EvalHook 方法), 220
exists() (mmcv.fileio.FileClient 方法), 118
ExpLrUpdaterHook (mmcv.runner 中的类), 220

F

FileClient (mmcv.fileio 中的类), 117
finalize_options() (mmcv.utils.BuildExtension 方法), 161
FixedLrUpdaterHook (mmcv.runner 中的类), 220
FlatCosineAnnealingLrUpdaterHook (mmcv.runner 中的类), 220
flow2rgb() (在 mmcv.visualization 模块中), 157
flow_from_bytes() (在 mmcv.video 模块中), 150
flow_warp() (在 mmcv.video 模块中), 151
flowread() (在 mmcv.video 模块中), 151
flowshow() (在 mmcv.visualization 模块中), 158
flowwrite() (在 mmcv.video 模块中), 151
force_fp32() (在 mmcv.runner 模块中), 235
forward() (mmcv.cnn.AlexNet 方法), 183
forward() (mmcv.cnn.ContextBlock 方法), 184
forward() (mmcv.cnn.Conv2d 方法), 184
forward() (mmcv.cnn.Conv3d 方法), 185
forward() (mmcv.cnn.ConvAWS2d 方法), 185
forward() (mmcv.cnn.ConvModule 方法), 187
forward() (mmcv.cnn.ConvTranspose2d 方法), 187
forward() (mmcv.cnn.ConvTranspose3d 方法), 188
forward() (mmcv.cnn.ConvWS2d 方法), 188
forward() (mmcv.cnn.DepthwiseSeparableConvModule 方法), 189
forward() (mmcv.cnn.GeneralizedAttention 方法), 190
forward() (mmcv.cnn.HSigmoid 方法), 191
forward() (mmcv.cnn.HSwish 方法), 191
forward() (mmcv.cnn.Linear 方法), 192
forward() (mmcv.cnn.MaxPool2d 方法), 193

forward() (*mmcv.cnn.MaxPool3d 方法*), 193
forward() (*mmcv.cnn.ResNet 方法*), 195
forward() (*mmcv.cnn.Scale 方法*), 196
forward() (*mmcv.cnn.Swish 方法*), 196
forward() (*mmcv.cnn.VGG 方法*), 198
forward() (*mmcv.ops.BorderAlign 方法*), 243
forward() (*mmcv.ops.CARAFE 方法*), 244
forward() (*mmcv.ops.CARAFENaive 方法*), 244
forward() (*mmcv.ops.CARAFEPack 方法*), 245
forward() (*mmcv.ops.CornerPool 方法*), 245
forward() (*mmcv.ops.Correlation 方法*), 246
forward() (*mmcv.ops.CrissCrossAttention 方法*), 247
forward() (*mmcv.ops.DeformConv2d 方法*), 248
forward() (*mmcv.ops.DeformConv2dPack 方法*), 249
forward() (*mmcv.ops.DeformRoIPool 方法*), 249
forward() (*mmcv.ops.DeformRoIPoolPack 方法*), 250
forward() (*mmcv.ops.DynamicScatter 方法*), 250
forward() (*mmcv.ops.FusedBiasLeakyReLU 方法*), 251
forward() (*mmcv.ops.GroupAll 方法*), 252
forward() (*mmcv.ops.MaskedConv2d 方法*), 252
forward() (*mmcv.ops.ModulatedDeformConv2d 方法*), 253
forward() (*mmcv.ops.ModulatedDeformConv2dPack 方法*), 253
forward() (*mmcv.ops.ModulatedDeformRoIPoolPack 方法*), 254
forward() (*mmcv.ops.MultiScaleDeformableAttention 方法*), 254
forward() (*mmcv.ops.PointsSampler 方法*), 256
forward() (*mmcv.ops.PrRoIPool 方法*), 257
forward() (*mmcv.ops.PSAMask 方法*), 255
forward() (*mmcv.ops.QueryAndGroup 方法*), 257
forward() (*mmcv.ops.RiRoIAlignRotated 方法*), 258
forward() (*mmcv.ops.RoIAlign 方法*), 259
forward() (*mmcv.ops.RoIAlignRotated 方法*), 260
forward() (*mmcv.ops.RoIAwarePool3d 方法*), 260
forward() (*mmcv.ops.RoIPointPool3d 方法*), 261
forward() (*mmcv.ops.RoIPool 方法*), 261
forward() (*mmcv.ops.SAConv2d 方法*), 262
forward() (*mmcv.ops.SigmoidFocalLoss 方法*), 262
forward() (*mmcv.ops.SimpleRoIAlign 方法*), 263
forward() (*mmcv.ops.SoftmaxFocalLoss 方法*), 263
forward() (*mmcv.ops.SparseSequential 方法*), 264
forward() (*mmcv.ops.SyncBatchNorm 方法*), 265
forward() (*mmcv.ops.TINShift 方法*), 266
forward() (*mmcv.ops.Voxelization 方法*), 266
forward_single() (*mmcv.ops.DynamicScatter 方法*), 251
fourcc (*mmcv.video.VideoReader property*), 148
Fp16OptimizerHook (*mmcv.runner 中的类*), 220
fps (*mmcv.video.VideoReader property*), 148
frame_cnt (*mmcv.video.VideoReader property*), 148
frames2video() (*在 mmcv.video 模块中*), 152
fromstring() (*mmcv.utils.Config 静态方法*), 164
fuse_conv_bn() (*在 mmcv.cnn 模块中*), 201
fused_bias_leakyrelu() (*在 mmcv.ops 模块中*), 273
FusedBiasLeakyReLU (*mmcv.ops 中的类*), 251

G

GeneralizedAttention (*mmcv.cnn 中的类*), 190
get() (*mmcv.fileio.FileClient 方法*), 118
get() (*mmcv.utils.Registry 方法*), 169
get_ext_filename() (*mmcv.utils.BuildExtension 方法*), 161
get_frame() (*mmcv.video.VideoReader 方法*), 148
get_git_hash() (*在 mmcv.utils 模块中*), 176
get_host_info() (*在 mmcv.runner 模块中*), 236
get_iter() (*mmcv.runner.LoggerHook 方法*), 224
get_local_path() (*mmcv.fileio.FileClient 方法*), 118
get_logger() (*在 mmcv.utils 模块中*), 176
get_model_complexity_info() (*在 mmcv.cnn 模块中*), 201
get_priority() (*在 mmcv.runner 模块中*), 236
get_step() (*mmcv.runner.PaviLoggerHook 方法*), 230
get_text() (*mmcv.fileio.FileClient 方法*), 119
GradientCumulativeFp16OptimizerHook (*mmcv.runner 中的类*), 221
GradientCumulativeOptimizerHook (*mmcv.runner 中的类*), 222
gray2bgr() (*在 mmcv.image 模块中*), 132
gray2rgb() (*在 mmcv.image 模块中*), 132
GroupAll (*mmcv.ops 中的类*), 252

H

has_method() (在 mmcv.utils 模块中), 176
has_overflow() (mmcv.runner.LossScaler 方法), 225
height (mmcv.video.VideoReader property), 148
hls2bgr() (在 mmcv.image 模块中), 132
hooks (mmcv.runner.BaseRunner property), 207
HSigmoid (mmcv.cnn 中的类), 190
hsv2bgr() (在 mmcv.image 模块中), 132
HSwish (mmcv.cnn 中的类), 191

I

imconvert() (在 mmcv.image 模块中), 133
imcrop() (在 mmcv.image 模块中), 133
imequalize() (在 mmcv.image 模块中), 133
imflip() (在 mmcv.image 模块中), 134
imflip_() (在 mmcv.image 模块中), 134
imfrombytes() (在 mmcv.image 模块中), 134
iminvert() (在 mmcv.image 模块中), 135
imnormalize() (在 mmcv.image 模块中), 135
imnormalize_() (在 mmcv.image 模块中), 135
impad() (在 mmcv.image 模块中), 135
impad_to_multiple() (在 mmcv.image 模块中), 136
import_modules_from_strings() (在 mmcv.utils 模块中), 177
imread() (在 mmcv.image 模块中), 137
imrescale() (在 mmcv.image 模块中), 138
imresize() (在 mmcv.image 模块中), 138
imresize_like() (在 mmcv.image 模块中), 139
imresize_to_multiple() (在 mmcv.image 模块中), 139
imrotate() (在 mmcv.image 模块中), 140
imshear() (在 mmcv.image 模块中), 140
imshow() (在 mmcv.visualization 模块中), 158
imshow_bboxes() (在 mmcv.visualization 模块中), 158
imshow_det_bboxes() (在 mmcv.visualization 模块中), 158
imtranslate() (在 mmcv.image 模块中), 141
imwrite() (在 mmcv.image 模块中), 141
infer_client() (mmcv.fileio.FileClient 类方法), 119
infer_scope() (mmcv.utils.Registry 静态方法), 169
init_weights() (mmcv.ops.MultiScaleDeformableAttention 方法), 255
init_weights() (mmcv.runner.BaseModule 方法), 205
initialize() (在 mmcv.cnn 模块中), 202
inner_iter (mmcv.runner.BaseRunner property), 207
InvLrUpdaterHook (mmcv.runner 中的类), 222
is_list_of() (在 mmcv.utils 模块中), 177
is_method_overridden() (在 mmcv.utils 模块中), 177
is_norm() (在 mmcv.cnn 模块中), 203
is_running (mmcv.utils.Timer property), 171
is_scalar() (mmcv.runner.LoggerHook 静态方法), 224
is_seq_of() (在 mmcv.utils 模块中), 177
is_str() (在 mmcv.utils 模块中), 178
is_tuple_of() (在 mmcv.utils 模块中), 178
isdir() (mmcv.fileio.FileClient 方法), 120
isfile() (mmcv.fileio.FileClient 方法), 120
iter (mmcv.runner.BaseRunner property), 207
iter_cast() (在 mmcv.utils 模块中), 178
IterBasedRunner (mmcv.runner 中的类), 222

J

join_path() (mmcv.fileio.FileClient 方法), 120

K

KaimingInit (mmcv.cnn 中的类), 192

L

Linear (mmcv.cnn 中的类), 192
Linear() (在 mmcv.ops 模块中), 252
LinearAnnealingLrUpdaterHook (mmcv.runner 中的类), 224
LinearAnnealingMomentumUpdaterHook (mmcv.runner 中的类), 224
list_cast() (在 mmcv.utils 模块中), 178
list_dir_or_file() (mmcv.fileio.FileClient 方法), 120
list_from_file() (在 mmcv.fileio 模块中), 124
load() (在 mmcv.fileio 模块中), 125

load_checkpoint() (*mmcv.runner.CheckpointLoader* 类方法), 209
load_checkpoint() (在 *mmcv.runner* 模块中), 236
load_state_dict() (*mmcv.runner.LossScaler* 方法), 225
load_state_dict() (在 *mmcv.runner* 模块中), 237
load_url() (在 *mmcv.utils* 模块中), 178
LoggerHook (*mmcv.runner* 中的类), 224
LossScaler (*mmcv.runner* 中的类), 225
LrUpdaterHook (*mmcv.runner* 中的类), 226
lut_transform() (在 *mmcv.image* 模块中), 142

M

make_color_wheel() (在 *mmcv.visualization* 模块中), 159
MaskedConv2d (*mmcv.ops* 中的类), 252
max_epochs (*mmcv.runner.BaseRunner property*), 207
max_iters (*mmcv.runner.BaseRunner property*), 207
MaxPool2d (*mmcv.cnn* 中的类), 192
MaxPool2d() (在 *mmcv.ops* 模块中), 252
MaxPool3d (*mmcv.cnn* 中的类), 193
merge_from_dict() (*mmcv.utils.Config* 方法), 164
min_area_polygons() (在 *mmcv.ops* 模块中), 273
MlflowLoggerHook (*mmcv.runner* 中的类), 226
mmcv.arraymisc
    模块, 155
mmcv.cnn
    模块, 183
mmcv.engine
    模块, 241
mmcv.fileio
    模块, 117
mmcv.image
    模块, 127
mmcv.ops
    模块, 243
mmcv.runner
    模块, 205
mmcv.utils
    模块, 161
mmcv.video
    模块, 147
mmcv.visualization
    模块, 157
model_name (*mmcv.runner.BaseRunner property*), 207
ModulatedDeformConv2d (*mmcv.ops* 中的类), 252
ModulatedDeformConv2dPack (*mmcv.ops* 中的类), 253
ModulatedDeformRoIPoolPack (*mmcv.ops* 中的类), 253
ModuleDict (*mmcv.runner* 中的类), 226
ModuleList (*mmcv.runner* 中的类), 227
multi_gpu_test() (在 *mmcv.engine* 模块中), 242
MultiScaleDeformableAttention (*mmcv.ops* 中的类), 254

N

NeptuneLoggerHook (*mmcv.runner* 中的类), 227
nms() (在 *mmcv.ops* 模块中), 273
nms3d() (在 *mmcv.ops* 模块中), 274
nms3d_normal() (在 *mmcv.ops* 模块中), 274
nms_bev() (在 *mmcv.ops* 模块中), 275
nms_match() (在 *mmcv.ops* 模块中), 275
nms_normal_bev() (在 *mmcv.ops* 模块中), 275
nms_rotated() (在 *mmcv.ops* 模块中), 276
NonLocal1d (*mmcv.cnn* 中的类), 193
NonLocal2d (*mmcv.cnn* 中的类), 193
NonLocal3d (*mmcv.cnn* 中的类), 194
NormalInit (*mmcv.cnn* 中的类), 194

O

obj_from_dict() (在 *mmcv.runner* 模块中), 237
OneCycleLrUpdaterHook (*mmcv.runner* 中的类), 227
OneCycleMomentumUpdaterHook (*mmcv.runner* 中的类), 228
opened (*mmcv.video.VideoReader property*), 148
OptimizerHook (*mmcv.runner* 中的类), 229

P

parse_uri_prefix() (*mmcv.fileio.FileClient* 静态方法), 121
PaviLoggerHook (*mmcv.runner* 中的类), 229

pixel_group() (在 mmcv.ops 模块中), 276
point_sample() (在 mmcv.ops 模块中), 277
points_in_boxes_all() (在 mmcv.ops 模块中), 277
points_in_boxes_cpu() (在 mmcv.ops 模块中), 277
points_in_boxes_part() (在 mmcv.ops 模块中), 278
points_in_polygons() (在 mmcv.ops 模块中), 278
PointsSampler (mmcv.ops 中的类), 256
PolyLrUpdaterHook (mmcv.runner 中的类), 230
PoolDataLoader() (在 mmcv.utils 模块中), 168
position (mmcv.video.VideoReader property), 148
posterize() (在 mmcv.image 模块中), 142
PretrainedInit (mmcv.cnn 中的类), 194
print_log() (在 mmcv.utils 模块中), 179
Priority (mmcv.runner 中的类), 230
ProgressBar (mmcv.utils 中的类), 168
PrRoIPool (mmcv.ops 中的类), 256
PSAMask (mmcv.ops 中的类), 255
put() (mmcv.fileio.FileClient 方法), 121
put_text() (mmcv.fileio.FileClient 方法), 121

Q

quantize() (在 mmcv.arraymisc 模块中), 155
quantize_flow() (在 mmcv.video 模块中), 152
QueryAndGroup (mmcv.ops 中的类), 257

R

rank (mmcv.runner.BaseRunner property), 207
read() (mmcv.video.VideoReader 方法), 148
register_backend() (mmcv.fileio.FileClient 类方法), 122
register_hook() (mmcv.runner.BaseRunner 方法), 207
register_hook_from_cfg() (mmcv.runner.BaseRunner 方法), 207
register_module() (mmcv.utils.Registry 方法), 169
register_scheme() (mmcv.runner.CheckpointLoader 类方法), 209
register_training_hooks() (mmcv.runner.BaseRunner 方法), 208
register_training_hooks() (mmcv.runner.IterBasedRunner 方法), 222
Registry (mmcv.utils 中的类), 168
rel_roi_point_to_rel_img_point() (在 mmcv.ops 模块中), 278
remove() (mmcv.fileio.FileClient 方法), 122
requires_executable() (在 mmcv.utils 模块中), 179
requires_package() (在 mmcv.utils 模块中), 179
rescale_size() (在 mmcv.image 模块中), 142
resize_video() (在 mmcv.video 模块中), 153
ResNet (mmcv.cnn 中的类), 194
resolution (mmcv.video.VideoReader property), 149
resume() (mmcv.runner.IterBasedRunner 方法), 223
rgb2bgr() (在 mmcv.image 模块中), 143
rgb2gray() (在 mmcv.image 模块中), 143
rgb2ycbcr() (在 mmcv.image 模块中), 143
RiRoIAlignRotated (mmcv.ops 中的类), 258
RoIAlign (mmcv.ops 中的类), 258
RoIAlignRotated (mmcv.ops 中的类), 259
RoIAwarePool3d (mmcv.ops 中的类), 260
RoIPointPool3d (mmcv.ops 中的类), 261
RoIPool (mmcv.ops 中的类), 261
run() (mmcv.runner.EpochBasedRunner 方法), 218
run() (mmcv.runner.IterBasedRunner 方法), 223
Runner (mmcv.runner 中的类), 231

S

SAConv2d (mmcv.ops 中的类), 261
save_checkpoint() (mmcv.runner.EpochBasedRunner 方法), 218
save_checkpoint() (mmcv.runner.IterBasedRunner 方法), 223
save_checkpoint() (在 mmcv.runner 模块中), 238
Scale (mmcv.cnn 中的类), 196
scandir() (在 mmcv.utils 模块中), 180
scatter_nd() (在 mmcv.ops 模块中), 279
SegmindLoggerHook (mmcv.runner 中的类), 231
Sequential (mmcv.runner 中的类), 231

set_random_seed() (在 mmcv.runner 模块中), 238
SigmoidFocalLoss (mmcv.ops 中的类), 262
SimpleRoIAlign (mmcv.ops 中的类), 263
since_last_check() (mmcv.utils.Timer 方法), 171
since_start() (mmcv.utils.Timer 方法), 171
single_gpu_test() (在 mmcv.engine 模块中), 242
slice_list() (在 mmcv.utils 模块中), 180
soft_nms() (在 mmcv.ops 模块中), 279
SoftmaxFocalLoss (mmcv.ops 中的类), 263
solarize() (在 mmcv.image 模块中), 144
sparse_flow_from_bytes() (在 mmcv.video 模块中), 153
SparseConv2d (mmcv.ops 中的类), 263
SparseConv3d (mmcv.ops 中的类), 263
SparseConvTranspose2d (mmcv.ops 中的类), 263
SparseConvTranspose3d (mmcv.ops 中的类), 263
SparseInverseConv2d (mmcv.ops 中的类), 263
SparseInverseConv3d (mmcv.ops 中的类), 263
SparseMaxPool2d (mmcv.ops 中的类), 264
SparseMaxPool3d (mmcv.ops 中的类), 264
SparseModule (mmcv.ops 中的类), 264
SparseSequential (mmcv.ops 中的类), 264
split_scope_key() (mmcv.utils.Registry 静态方法), 170
start() (mmcv.utils.Timer 方法), 172
state_dict() (mmcv.runner.LossScaler 方法), 225
StepLrUpdaterHook (mmcv.runner 中的类), 231
StepMomentumUpdaterHook (mmcv.runner 中的类), 232
SubMConv2d (mmcv.ops 中的类), 265
SubMConv3d (mmcv.ops 中的类), 265
Swish (mmcv.cnn 中的类), 196
SyncBatchNorm (mmcv.ops 中的类), 265
SyncBatchNorm (mmcv.utils 中的类), 170
SyncBuffersHook (mmcv.runner 中的类), 232

T

tensor2imgs() (在 mmcv.image 模块中), 144
TensorboardLoggerHook (mmcv.runner 中的类), 232
TextLoggerHook (mmcv.runner 中的类), 232
Timer (mmcv.utils 中的类), 171
TimerError, 172
TINShift (mmcv.ops 中的类), 266
torch_meshgrid() (在 mmcv.utils 模块中), 180
track_iter_progress() (在 mmcv.utils 模块中), 181
track_parallel_progress() (在 mmcv.utils 模块中), 181
track_progress() (在 mmcv.utils 模块中), 181
train() (mmcv.cnn.ResNet 方法), 195
train() (mmcv.cnn.VGG 方法), 198
TruncNormalInit (mmcv.cnn 中的类), 196
tuple_cast() (在 mmcv.utils 模块中), 182

U

UniformInit (mmcv.cnn 中的类), 197
update_scale() (mmcv.runner.LossScaler 方法), 225
upfirdn2d() (在 mmcv.ops 模块中), 279
use_backend() (在 mmcv.image 模块中), 144

V

vcap (mmcv.video.VideoReader property), 149
VGG (mmcv.cnn 中的类), 197
VideoReader (mmcv.video 中的类), 147
Voxelization (mmcv.ops 中的类), 266

W

WandbLoggerHook (mmcv.runner 中的类), 233
weights_to_cpu() (在 mmcv.runner 模块中), 238
width (mmcv.video.VideoReader property), 149
with_options() (mmcv.utils.BuildExtension 类方法), 162
worker_init_fn() (在 mmcv.utils 模块中), 182
world_size (mmcv.runner.BaseRunner property), 208
wrap_fp16_model() (在 mmcv.runner 模块中), 238

X

XavierInit (mmcv.cnn 中的类), 198

Y

ycbcr2bgr() (在 mmcv.image 模块中), 144
ycbcr2rgb() (在 mmcv.image 模块中), 145

模块

mmcv.arraymisc, 155
mmcv.cnn, 183
mmcv.engine, 241
mmcv.fileio, 117
mmcv.image, 127
mmcv.ops, 243
mmcv.runner, 205
mmcv.utils, 161
mmcv.video, 147
mmcv.visualization, 157