MMClassification Documentation

发布 *1.0.0rc6*

MMClassification Authors

2023 年 04 月 07 日

开始你的第一步

CHAPTER 1

依赖环境

在本节中，我们将演示如何准备 PyTorch 相关的依赖环境。

MMClassification 适用于 Linux、Windows 和 macOS。它需要 Python 3.7+、CUDA 9.2+ 和 PyTorch 1.6+。

备注： 如果你对配置 PyTorch 环境已经很熟悉，并且已经完成了配置，可以直接进入下一节。否则的话，请依照以下步骤完成配置。

第 1 步从官网下载并安装 Miniconda。

第 2 步创建一个 conda 虚拟环境并激活它。

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

第 3 步按照官方指南安装 PyTorch。例如：

在 GPU 平台：

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

警告： 以上命令会自动安装最新版的 PyTorch 与对应的 cudatoolkit，请检查它们是否与你的环境匹配。

在 CPU 平台：

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

CHAPTER 2

安装

我们推荐用户按照我们的最佳实践来安装 MMClassification。但除此之外，如果你想根据你的习惯完成安装流程，也可以参见*自定义安装*一节来获取更多信息。

2.1 最佳实践

根据具体需求，我们支持两种安装模式：

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

2.1.1 从源码安装

这种情况下，从源码按如下方式安装 mmcls：

```
git clone -b 1.x https://github.com/open-mmlab/mmclassification.git
cd mmclassification
pip install -U openmim && mim install -e .
```

备注： "-e" 表示以可编辑形式安装，这样可以在不重新安装的情况下，让本地修改直接生效

2.1.2 作为 Python 包安装

直接使用 mim 安装即可。

```
pip install -U openmim && mim install "mmcls>=1.0rc0"
```

备注： mim 是一个轻量级的命令行工具，可以根据 PyTorch 和 CUDA 版本为 OpenMMLab 算法库配置合适的环境。同时它也提供了一些对于深度学习实验很有帮助的功能。

2.2 验证安装

为了验证 MMClassification 的安装是否正确，我们提供了一些示例代码来执行模型推理。

第 1 步我们需要下载配置文件和模型权重文件

```
mim download mmcls --config resnet50_8xb32_in1k --dest .
```

第 2 步验证示例的推理流程

如果你是**从源码安装**的 mmcls，那么直接运行以下命令进行验证：

```
python demo/image_demo.py demo/demo.JPEG resnet50_8xb32_in1k.py resnet50_8xb32_in1k_
↪20210831-ea4938fc.pth --device cpu
```

你可以看到命令行中输出了结果字典，包括 pred_label，pred_score 和 pred_class 三个字段。

如果你是**作为 Python 包安装**，那么可以打开你的 Python 解释器，并粘贴如下代码：

```
from mmcls import get_model, inference_model

model = get_model('resnet18_8xb32_in1k', device='cpu')  # 或者 device='cuda:0'
inference_model(model, 'demo/demo.JPEG')
```

你会看到输出一个字典，包含预测的标签、得分及类别名。

备注： 以上示例中，resnet18_8xb32_in1k 是模型名称。你可以使用*mmcls.list_models* 接口来浏览所有的模型，或者在模型汇总页面进行查找。

2.3 自定义安装

2.3.1 CUDA 版本

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

- 对于 Ampere 架构的 NVIDIA GPU，例如 GeForce 30 series 以及 NVIDIA A100，CUDA 11 是必需的。
- 对于更早的 NVIDIA GPU，CUDA 11 是向前兼容的，但 CUDA 10.2 能够提供更好的兼容性，也更加轻量。

请确保你的 GPU 驱动版本满足最低的版本需求，参阅这张表。

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

2.3.2 在 CPU 环境中安装

MMClassification 可以仅在 CPU 环境中安装，在 CPU 模式下，你可以完成训练、测试和模型推理等所有操作。

2.3.3 在 Google Colab 中安装

参考 Colab 教程 安装即可。

2.3.4 通过 Docker 使用 MMClassification

MMClassification 提供 Dockerfile 用于构建镜像。请确保你的 Docker 版本 >=19.03。

```
# 构建默认的 PyTorch 1.8.1, CUDA 10.2 版本镜像
# 如果你希望使用其他版本，请修改 Dockerfile
docker build -t mmclassification docker/
```

用以下命令运行 Docker 镜像：

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

2.4 故障解决

如果你在安装过程中遇到了什么问题，请先查阅常见问题。如果没有找到解决方法，可以在 GitHub 上提出 issue。

CHAPTER 3

使用现有模型推理

MMClassification 在*Model Zoo* 中提供了用于分类的预训练模型。本说明将展示**如何使用现有模型对给定图像进行推理**。

至于如何在标准数据集上测试现有模型，请看这个指南

3.1 推理单张图片

MMClassification 为图像推理提供高级 Python API：

- *get_model*: 根据名称获取一个模型。
- *init_model*: 根据配置文件和权重文件初始化一个模型。
- *inference_model*：对给定图片进行推理。

下面是一个示例，如何使用一个 ImageNet-1k 预训练权重初始化模型并推理给定图像。

备注： 可以运行 `wget https://github.com/open-mmlab/mmclassification/raw/master/demo/demo.JPEG` 下载样例图片，或使用其他图片。

```
from mmcls import get_model, inference_model

img_path = 'demo.JPEG'   # 可以指定自己的图片路径
```

(下页继续)

(续上页)

```
# 构建模型
model = get_model('resnet50_8xb32_in1k', pretrained=True, device="cpu")  # `device`␣
↪可以为 'cuda:0'
# 执行推理
result = inference_model(model, img_path)
```

result 为一个包含了 pred_label, pred_score, pred_scores 和 pred_class 的字典，结果如下:

```
{"pred_label":65,"pred_score":0.6649366617202759,"pred_class":"sea snake", "pred_
↪scores": [..., 0.6649366617202759, ...]}
```

演示可以在 demo/image_demo.py 中找到。

CHAPTER 4

准备数据集

目前 MMClassification 所支持的数据集有：

- *CustomDataset*
- *ImageNet*
- *CIFAR*
- *MINIST*
- *OpenMMLab 2.0 标准数据集*
- *其他数据集*
- *数据集包装*

如果你使用的数据集不在以上所列公开数据集中，需要转换数据集格式来适配 **`CustomDataset`**。

4.1 CustomDataset

`CustomDataset` 是一个通用的数据集类，供您使用自己的数据集。目前 `CustomDataset` 支持以下两种方式组织你的数据集文件：

4.1.1 子文件夹方式

文件夹格式通过文件来区别图片的类别，如下，class_1 和 class_2 就代表了区分了不同的类别。

```
data_prefix/
├── class_1
|   ├── xxx.png
|   ├── xxy.png
|   └── ...
├── class_2
|   ├── 123.png
|   ├── 124.png
|   └── ...
```

假如你希望将之用于训练，那么配置文件中需要添加以下配置：

```
train_dataloader = dict(
    ...
    # 训练数据集配置
    dataset=dict(
        type='CustomDataset',
        data_prefix='path/to/data_prefix',
        pipeline=...
    )
)
```

4.1.2 标注文件方式

标注文件格式主要使用文本文件来保存类别信息，`data_prefix` 存放图片，`ann_file` 存放标注类别信息。

如下案例，dataset 目录如下：

```
data_root/
├── meta/
|   ├── ann_file
|   └── ...
├── data_prefix/
|   ├── folder_1
|   |   ├── xxx.png |   |   ├── xxy.png
|   |   └── ...
|   ├── 123.png
|   ├── nsdf3.png
|   └── ...
```

标注文件 `ann_file` 内为普通文本，分为两列，第一列为图片路径，第二列为**类别的序号**。如下：

```
folder_1/xxx.png 0
folder_1/xxy.png 1
123.png 1
nsdf3.png 2
...
```

备注： 类别序号的值应当在 [0, num_classes - 1] 范围的整数。

另外还需要数据集配置文件中指定 `classes` 字段，如：

```
train_dataloader = dict(
    ...
    # 训练数据集配置
    dataset=dict(
        type='CustomDataset',
        ann_file='path/to/ann_file_path',
        data_prefix='path/to/images',
        classes=['A', 'B', 'C', 'D', ...]
        pipeline=...,
    )
)
```

备注： 如果指定了‘ann_file’，则通过‘ann_file’得到标注信息；否则，按照子文件夹格式处理。

4.2 ImageNet

ImageNet 有多个版本，但最常用的一个是 ILSVRC 2012。可以通过以下步骤使用它。

1. 注册一个帐户并登录到下载页面。
2. 找到 ILSVRC2012 的下载链接，下载以下两个文件：
   - ILSVRC2012_img_train.tar (~138GB)
   - ILSVRC2012_img_val.tar (~6.3GB)
3. 解压已下载的图片。
4. 从此 链接 下载并解压标注文件。
5. 根据标注数据中的路径重新组织图像文件，应该是这样的:

```
   imagenet/
   ├── meta/
   │   ├── train.txt
   │   ├── test.txt
   │   └── val.txt
   ├── train/
   │   ├── n01440764
   │   │   ├── n01440764_10026.JPEG
   │   │   ├── n01440764_10027.JPEG
   │   │   ├── n01440764_10029.JPEG
   │   │   ├── n01440764_10040.JPEG
   │   │   ├── n01440764_10042.JPEG
   │   │   ├── n01440764_10043.JPEG
   │   │   └── n01440764_10048.JPEG
   │   ├── ...
   ├── val/
   │   ├── ILSVRC2012_val_00000001.JPEG
   │   ├── ILSVRC2012_val_00000002.JPEG
   │   ├── ILSVRC2012_val_00000003.JPEG
   │   ├── ILSVRC2012_val_00000004.JPEG
   │   ├── ...
```

然后，您可以使用具有以下配置的*ImageNet* 数据集：

```
train_dataloader = dict(
    ...
    # 训练数据集配置
    dataset=dict(
        type='ImageNet',
        data_root='imagenet_folder',
        ann_file='meta/train.txt',
        data_prefix='train/',
        pipeline=...,
    )
)
val_dataloader = dict(
    ...
    # 验证数据集配置
    dataset=dict(
        type='ImageNet',
        data_root='imagenet_folder',
        ann_file='meta/val.txt',
        data_prefix='val/',
        pipeline=...,
```

(下页继续)

(续上页)

```
    )
)
test_dataloader = val_dataloader
```

4.3 CIFAR

我们支持自动下载*CIFAR10*和*CIFAR100*数据集，您只需在 data_root 字段中指定下载文件夹即可。并且通过指定 test_mode=False / test_mode=True 来使用训练数据集或测试数据集。

```
train_dataloader = dict(
    ...
    # 训练数据集配置
    dataset=dict(
        type='CIFAR10',
        data_root='data/cifar10',
        test_mode=False,
        pipeline=...,
    )
)
val_dataloader = dict(
    ...
    # 验证数据集配置
    dataset=dict(
        type='CIFAR10',
        data_root='data/cifar10',
        test_mode=True,
        pipeline=...,
    )
)
test_dataloader = val_dataloader
```

4.4 MNIST

我们支持自动下载*MNIST*和*Fashion-MNIST*数据集，您只需指定 data_root 字段中的下载路径即可。并且通过指定 test_mode=False / test_mode=True 来使用训练数据集或测试数据集。

```
train_dataloader = dict(
    ...
    # 训练数据集配置
    dataset=dict(
```

(下页继续)

(续上页)

```
        type='MNIST',
        data_root='data/mnist',
        test_mode=False,
        pipeline=...,
    )
)
val_dataloader = dict(
    ...
    # 验证数据集配置
    dataset=dict(
        type='MNIST',
        data_root='data/mnist',
        test_mode=True,
        pipeline=...,
    )
)
test_dataloader = val_dataloader
```

4.5 OpenMMLab 2.0 标准数据集

为了统一不同任务的数据集接口，便于多任务的算法模型训练，OpenMMLab 制定了 **OpenMMLab 2.0 数据集格式规范**，数据集标注文件需符合该规范，数据集基类基于该规范去读取与解析数据标注文件。如果用户提供的数据标注文件不符合规定格式，用户可以选择将其转化为规定格式，并使用 OpenMMLab 的算法库基于该数据标注文件进行算法训练和测试。

OpenMMLab 2.0 数据集格式规范规定，标注文件必须为 `json` 或 `yaml`，`yml` 或 `pickle`，`pkl` 格式；标注文件中存储的字典必须包含 `metainfo` 和 `data_list` 两个字段。其中 `metainfo` 是一个字典，里面包含数据集的元信息；`data_list` 是一个列表，列表中每个元素是一个字典，该字典定义了一个原始数据（raw data），每个原始数据包含一个或若干个训练/测试样本。

假设您要使用训练数据集，那么配置文件如下所示：

```

{
    'metainfo':
        {
            'classes': ('cat', 'dog'), # 'cat' 的类别序号为 0，'dog' 为 1。
            ...
        },
    'data_list':
        [
            {
```

(下页继续)

(续上页)

```
                'img_path': "xxx/xxx_0.jpg",
                'img_label': 0,
                ...
            },
            {
                'img_path': "xxx/xxx_1.jpg",
                'img_label': 1,
                ...
            },
            ...
        ]
}
```

同时假设数据集存放路径如下：

```
data
├── annotations
|   ├── train.json
|   └── ...
├── train
|   ├── xxx/xxx_0.jpg
|   ├── xxx/xxx_1.jpg
|   ├── ...
```

通过以下字典构建：

```
dataset_cfg=dict(
    type='CustomDataset',
    ann_file='path/to/ann_file_path',
    data_prefix='path/to/images_folder',
    pipeline=transfrom_list)
```

4.6 其他数据集

MMCLassification 还是支持更多其他的数据集，可以通过查阅数据集文档获取它们的配置信息。

4.7 数据集包装

MMEngine 中支持以下数据包装器，您可以参考 MMEngine 教程 了解如何使用它。

- `ConcatDataset`
- `RepeatDataset`
- `ClassBalancedDataset`

除上述之外，MMClassification 还支持了*KFoldDataset*，需用通过使用 `tools/kfold-cross-valid.py` 来使用它。

CHAPTER 5

训练与测试

5.1 训练

5.1.1 单机单卡训练

你可以使用 `tools/train.py` 在电脑上用 CPU 或是 GPU 进行模型的训练。

以下是训练脚本的完整用法：

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

备注： 默认情况下，MMClassification 会自动调用你的 GPU 进行训练。如果你有 GPU 但仍想使用 CPU 进行训练，请设置环境变量 `CUDA_VISIBLE_DEVICES` 为空或者 -1 来对禁用 GPU。

```
CUDA_VISIBLE_DEVICES=-1 python tools/train.py ${CONFIG_FILE} [ARGS]
```

参数	描述
CONFIG_FILE	配置文件的路径。
--work-dir WORK_DIR	用来保存训练日志和权重文件的文件夹，默认是 ./work_dirs 目录下，与配置文件同名的文件夹。
--resume [RESUME]	恢复训练。如果指定了权重文件路径，则从指定的权重文件恢复；如果没有指定，则尝试从最新的权重文件进行恢复。
--amp	启用混合精度训练。
--no-validate	**不建议**在训练过程中不进行验证集上的精度验证。
--auto-scale-lr	自动根据实际的批次大小（batch size）和预设的批次大小对学习率进行缩放。
--cfg-options CFG_OPTIONS	重载配置文件中的一些设置。使用类似 xxx=yyy 的键值对形式指定，这些设置会被融合入从配置文件读取的配置。你可以使用 key="[a,b]" 或者 key=a,b 的格式来指定列表格式的值，且支持嵌套，例如 'key=”[(a,b),(c,d)]”，这里的引号是不可省略的。另外每个重载项内部不可出现空格。
--launcher {none, pytorch, slurm,mpi}	启动器，默认为 “none”。

5.1.2 单机多卡训练

我们提供了一个 shell 脚本，可以使用 torch.distributed.launch 启动多 GPU 任务。

```
bash ./tools/dist_train.sh ${CONFIG_FILE} ${GPU_NUM} [PY_ARGS]
```

参数	描述
CONFIG_FILE	配置文件的路径。
GPU_NUM	使用的 GPU 数量。
[PY_ARGS]	tools/train.py 支持的其他可选参数，参见上文。

你还可以使用环境变量来指定启动器的额外参数，比如用如下命令将启动器的通讯端口变更为 29666：

```
PORT=29666 bash ./tools/dist_train.sh ${CONFIG_FILE} ${GPU_NUM} [PY_ARGS]
```

如果你希望使用不同的 GPU 进行多项训练任务，可以在启动时指定不同的通讯端口和不同的可用设备。

```
CUDA_VISIBLE_DEVICES=0,1,2,3 PORT=29500 bash ./tools/dist_train.sh ${CONFIG_FILE1} 4␣
↪[PY_ARGS]
CUDA_VISIBLE_DEVICES=4,5,6,7 PORT=29501 bash ./tools/dist_train.sh ${CONFIG_FILE2} 4␣
↪[PY_ARGS]
```

5.1.3 多机训练

同一网络下的多机

如果你希望使用同一局域网下连接的多台电脑进行一个训练任务，可以使用如下命令：

在第一台机器上：

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

在第二台机器上：

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

和单机多卡相比，你需要指定一些额外的环境变量：

环境变量	描述
NNODES	机器总数。
NODE_RANK	本机的序号
PORT	通讯端口，它在所有机器上都应当是一致的。
MASTER_ADDR	主机的 IP 地址，它在所有机器上都应当是一致的。

通常来说，如果这几台机器之间不是高速网络连接，训练速度会非常慢。

Slurm 管理下的多机集群

如果你在 slurm 集群上，可以使用 `tools/slurm_train.sh` 脚本启动任务。

```
[ENV_VARS] ./tools/slurm_train.sh ${PARTITION} ${JOB_NAME} ${CONFIG_FILE} ${WORK_DIR}␣
↪[PY_ARGS]
```

这里是该脚本的一些参数：

参数	描述
PARTITION	使用的集群分区。
JOB_NAME	任务的名称，你可以随意起一个名字。
CONFIG_FILE	配置文件路径。
WORK_DIR	用以保存日志和权重文件的文件夹。
[PY_ARGS]	`tools/train.py` 支持的其他可选参数，参见上文。

这里是一些你可以用来配置 slurm 任务的环境变量：

环境变量	描述
GPUS	使用的 GPU 总数，默认为 8。
GPUS_PER_NODE	每个节点分配的 GPU 数，你可以根据节点情况指定。默认为 8。
CPUS_PER_TASK	每个任务分配的 CPU 数（通常一个 GPU 对应一个任务）。默认为 5。
SRUN_ARGS	srun 命令支持的其他参数。可用的选项参见官方文档。

5.2 测试

5.2.1 单机单卡测试

你可以使用 tools/test.py 在电脑上用 CPU 或是 GPU 进行模型的测试。

以下是测试脚本的完整用法：

```
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [ARGS]
```

备注： 默认情况下，MMClassification 会自动调用你的 GPU 进行测试。如果你有 GPU 但仍想使用 CPU 进行测试，请设置环境变量 CUDA_VISIBLE_DEVICES 为空或者 -1 来对禁用 GPU。

```
CUDA_VISIBLE_DEVICES=-1 python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [ARGS]
```

参数	描述
CONFIG_FILE	配置文件的路径。
CHECKPOINT_FILE	权重文件路径（支持 http 链接，你可以在这里寻找需要的权重文件）。
--work-dir WORK_DIR	用来保存测试指标结果的文件夹。
--out OUT	用来保存测试输出的文件。
--out-item OUT_ITEM	指定测试输出文件的内容，可以为“pred”或“metrics”，其中“pred”表示保存所有模型输出，这些数据可以用于离线测评；”metrics”表示输出测试指标。默认为“pred”。
--cfg-options CFG_OPTIONS	重载配置文件中的一些设置。使用类似 xxx=yyy 的键值对形式指定，这些设置会被融合入从配置文件读取的配置。你可以使用 key="[a,b]" 或者 key=a,b 的格式来指定列表格式的值，且支持嵌套，例如 'key=”[(a,b),(c,d)]”，这里的引号是不可省略的。另外每个重载项内部不可出现空格。
--show-dir SHOW_DIR	用于保存可视化预测结果图像的文件夹。
--show	在窗口中显示预测结果图像。
--interval INTERVAL	每隔多少样本进行一次预测结果可视化。
--wait-time WAIT_TIME	每个窗口的显示时间（单位为秒）。
--launcher {none, pytorch, slurm,mpi}	启动器，默认为“none”。

5.2.2 单机多卡测试

我们提供了一个 shell 脚本，可以使用 torch.distributed.launch 启动多 GPU 任务。

```
bash ./tools/dist_test.sh ${CONFIG_FILE} ${CHECKPOINT_FILE} ${GPU_NUM} [PY_ARGS]
```

参数	描述
CONFIG_FILE	配置文件的路径。
CHECKPOINT_FILE	权重文件路径（支持 http 链接，你可以在这里寻找需要的权重文件）。
GPU_NUM	使用的 GPU 数量。
[PY_ARGS]	tools/test.py 支持的其他可选参数，参见上文。

你还可以使用环境变量来指定启动器的额外参数，比如用如下命令将启动器的通讯端口变更为 29666：

```
PORT=29666 bash ./tools/dist_test.sh ${CONFIG_FILE} ${CHECKPOINT_FILE} ${GPU_NUM} [PY_
↪ARGS]
```

如果你希望使用不同的 GPU 进行多项测试任务，可以在启动时指定不同的通讯端口和不同的可用设备。

```
CUDA_VISIBLE_DEVICES=0,1,2,3 PORT=29500 bash ./tools/dist_test.sh ${CONFIG_FILE1} $
↪{CHECKPOINT_FILE} 4 [PY_ARGS]
CUDA_VISIBLE_DEVICES=4,5,6,7 PORT=29501 bash ./tools/dist_test.sh ${CONFIG_FILE2} $
↪{CHECKPOINT_FILE} 4 [PY_ARGS]
```

5.2.3 多机测试

同一网络下的多机

如果你希望使用同一局域网下连接的多台电脑进行一个测试任务，可以使用如下命令：

在第一台机器上：

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

在第二台机器上：

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

和单机多卡相比，你需要指定一些额外的环境变量：

环境变量	描述
NNODES	机器总数。
NODE_RANK	本机的序号
PORT	通讯端口，它在所有机器上都应当是一致的。
MASTER_ADDR	主机的 IP 地址，它在所有机器上都应当是一致的。

Slurm 管理下的多机集群

如果你在 slurm 集群上，可以使用 `tools/slurm_test.sh` 脚本启动任务。

```
[ENV_VARS] ./tools/slurm_test.sh ${PARTITION} ${JOB_NAME} ${CONFIG_FILE} ${CHECKPOINT_
↪FILE} [PY_ARGS]
```

这里是该脚本的一些参数：

参数	描述
PARTITION	使用的集群分区。
JOB_NAME	任务的名称，你可以随意起一个名字。
CONFIG_FILE	配置文件路径。
CHECKPOINT_FILE	权重文件路径（支持 http 链接，你可以在这里寻找需要的权重文件）。
[PY_ARGS]	tools/test.py 支持的其他可选参数，参见上文。

这里是一些你可以用来配置 slurm 任务的环境变量：

环境变量	描述
GPUS	使用的 GPU 总数，默认为 8。
GPUS_PER_NODE	每个节点分配的 GPU 数，你可以根据节点情况指定。默认为 8。
CPUS_PER_TASK	每个任务分配的 CPU 数（通常一个 GPU 对应一个任务）。默认为 5。
SRUN_ARGS	srun 命令支持的其他参数。可用的选项参见官方文档。

CHAPTER 6

学习配置文件

为了管理深度学习实验的各种设置，我们使用配置文件来记录所有这些配置。这种配置文件系统具有模块化和继承特性，更多细节可以在MMEngine 中的教程。

MMClassification 主要使用 python 文件作为配置文件，所有配置文件都放置在 `configs` 文件夹下，目录结构如下所示:

```
MMClassification/
    ├── configs/
    │   ├── _base_/                     # 原始配置文件夹
    │   │   ├── datasets/                   # 原始数据集
    │   │   ├── models/                     # 原始模型
    │   │   ├── schedules/                  # 原始优化策略
    │   │   └── default_runtime.py          # 原始运行设置
    │   ├── resnet/                     # ResNet 算法文件夹
    │   ├── swin_transformer/           # Swin 算法文件夹
    │   ├── vision_transformer/         # ViT 算法文件夹
    │   ├── ...
    └── ...
```

可以使用 `python tools/misc/print_config.py /PATH/TO/CONFIG` 命令来查看完整的配置信息，从而方便检查所对应的配置文件。

本文主要讲解 MMClassification 配置文件的命名和结构，以及如何基于已有的配置文件修改，并以 ResNet50 配置文件 逐行解释。

6.1 配置文件结构

在 configs/_base_ 文件夹下有 4 个基本组件类型，分别是：

- 模型 (model)
- 数据 (data)
- 训练策略 (schedule)
- 运行设置 (runtime)

你可以通过继承一些基本配置文件轻松构建自己的训练配置文件。我们称这些被继承的配置文件为 *原始配置文件*，如 _base_ 文件夹中的文件一般仅作为原始配置文件。

下面使用 ResNet50 配置文件 作为案例进行说明并注释每一行含义。

```
_base_ = [                                    # 此配置文件将继承所有 `_base_` 中的配置
    '../_base_/models/resnet50.py',           # 模型配置
    '../_base_/datasets/imagenet_bs32.py',    # 数据配置
    '../_base_/schedules/imagenet_bs256.py',  # 训练策略配置
    '../_base_/default_runtime.py'            # 默认运行设置
]
```

我们将在下面分别解释这四个原始配置文件。

6.1.1 模型配置

模型原始配置文件包含一个 model 字典数据结构，主要包括网络结构、损失函数等信息：

- type：分类器名称, 对于图像分类任务，通常为 ImageClassifier，更多细节请参考 *API* 文档。
- backbone：主干网设置，主干网络为主要的特征提取网络，比如 ResNet, Swin Transformer, Vision Transformer 等等。更多可用选项请参考 *API* 文档。
- neck：颈网络设置，颈网络主要是连接主干网和分类的中间网络，比如 GlobalAveragePooling 等，更多可用选项请参考 *API* 文档。
- head：头网络设置，头网络主要是最后关联任务的分类部件，更多可用选项请参考 *API* 文档。
  - loss：损失函数设置，支持 CrossEntropyLoss, LabelSmoothLoss 等，更多可用选项参考 *API* 文档。
- data_preprocessor：图像输入的预处理模块，输入在进入模型前的预处理操作，例如 ClsDataPreprocessor, 有关详细信息，请参阅 *API* 文档。
- train_cfg：训练模型时的额外设置。在 MMCLS 中，我们主要使用它来配置批量增强，例如 Mixup 和 CutMix。有关详细信息，请参阅文档。

备注： 配置文件中的‘type’不是构造时的参数，而是类名。

以下是 ResNet50 的模型配置’configs/*base*/models/resnet50.py’：

```
model = dict(
    type='ImageClassifier',      # 分类器类型， 目前只有 'ImageClassifier'
    backbone=dict(
        type='ResNet',           # 主干网络类型
        # 除了 `type` 之外的所有字段都来自 `ResNet` 类的 __init__ 方法
        # 可查阅 https://mmclassification.readthedocs.io/zh_CN/1.x/api/generated/
↪mmcls.models.ResNet.html
        depth=50,
        num_stages=4,            #␣
↪主干网络状态(stages)的数目，这些状态产生的特征图作为后续的 head 的输入。
        out_indices=(3, ),       # 输出的特征图输出索引。
        frozen_stages=-1,        # 冻结主干网的层数
        style='pytorch'),
    neck=dict(type='GlobalAveragePooling'),     # 颈网络类型
    head=dict(
        type='LinearClsHead',          # 分类颈网络类型
        # 除了 `type` 之外的所有字段都来自 `LinearClsHead` 类的 __init__ 方法
        # 可查阅 https://mmclassification.readthedocs.io/zh_CN/1.x/api/generated/
↪mmcls.models.LinearClsHead.html
        num_classes=1000,
        in_channels=2048,
        loss=dict(type='CrossEntropyLoss', loss_weight=1.0), # 损失函数配置信息
        topk=(1, 5),                   # 评估指标，Top-k 准确率， 这里为 top1 与 top5␣
↪准确率
    ))
```

6.1.2 数据

数据原始配置文件主要包括预处理设置、dataloader 以及评估器等设置：

- `data_preprocessor`: 模型输入预处理配置，与 `model.data_preprocessor` 相同，但优先级更低。
- `train_evaluator | val_evaluator | test_evaluator`: 构建评估器，参考 *API* 文档。
- `train_dataloader | val_dataloader | test_dataloader`: 构建 dataloader
  - `samples_per_gpu`: 每个 GPU 的 batch size
  - `workers_per_gpu`: 每个 GPU 的线程数
  - `sampler`: 采样器配置

- dataset: 数据集配置
  - type: 数据集类型，MMClassification 支持 ImageNet、Cifar 等数据集，参考*API 文档*
  - pipeline: 数据处理流水线，参考相关教程文档 如何设计数据处理流水线

以下是 ResNet50 的数据配置 ‘configs/*base*/datasets/imagenet_bs32.py’：

```
dataset_type = 'ImageNet'
# 预处理配置
data_preprocessor = dict(
    # 输入的图片数据通道以 'RGB' 顺序
    mean=[123.675, 116.28, 103.53],     # 输入图像归一化的 RGB 通道均值
    std=[58.395, 57.12, 57.375],        # 输入图像归一化的 RGB 通道标准差
    to_rgb=True,                        # 是否将通道翻转，从 BGR 转为 RGB 或者 RGB␣
↪转为 BGR
)

train_pipeline = [
    dict(type='LoadImageFromFile'),      # 读取图像
    dict(type='RandomResizedCrop', scale=224),      # 随机放缩裁剪
    dict(type='RandomFlip', prob=0.5, direction='horizontal'),    # 随机水平翻转
    dict(type='PackClsInputs'),          # 准备图像以及标签
]

test_pipeline = [
    dict(type='LoadImageFromFile'),      # 读取图像
    dict(type='ResizeEdge', scale=256, edge='short'),  # 短边对其256进行放缩
    dict(type='CenterCrop', crop_size=224),      # 中心裁剪
    dict(type='PackClsInputs'),                  # 准备图像以及标签
]

# 构造训练集 dataloader
train_dataloader = dict(
    batch_size=32,                      # 每张GPU的 batchsize
    num_workers=5,                      # 每个GPU的线程数
    dataset=dict(                       # 训练数据集
        type=dataset_type,
        data_root='data/imagenet',
        ann_file='meta/train.txt',
        data_prefix='train',
        pipeline=train_pipeline),
    sampler=dict(type='DefaultSampler', shuffle=True),    # 默认采样器
    persistent_workers=True,                              #␣
↪是否保持进程，可以缩短每个epoch的准备时间
)
```

(下页继续)

(续上页)

```
# 构造验证集 dataloader
val_dataloader = dict(
    batch_size=32,
    num_workers=5,
    dataset=dict(
        type=dataset_type,
        data_root='data/imagenet',
        ann_file='meta/val.txt',
        data_prefix='val',
        pipeline=test_pipeline),
    sampler=dict(type='DefaultSampler', shuffle=False),
    persistent_workers=True,
)
# 验证集评估设置，使用准确率为指标， 这里使用 topk1 以及 top5 准确率
val_evaluator = dict(type='Accuracy', topk=(1, 5))

test_dataloader = val_dataloader  # test dataloader配置，这里直接与 val_dataloader相同
test_evaluator = val_evaluator    # 测试集的评估配置，这里直接与 val_evaluator 相同
```

备注: ‘model.data_preprocessor’既可以在 `model=dict(data_preprocessor=dict())` 中定义，也可以使用此处的 `data_preprocessor` 定义，同时配置时，优先使用 `model.data_preprocessor` 的配置。

6.1.3 训练策略

训练策略原始配置文件主要包括预优化器设置和训练、验证及测试的循环控制器 (LOOP):

- `optim_wrapper`: 优化器装饰器配置信息，我们使用优化器装饰配置优化进程。
  - `optimizer`: 支持 `pytorch` 所有的优化器，参考相关 MMEngine 文档。
  - `paramwise_cfg`: 根据参数的类型或名称设置不同的优化参数，参考相关学习策略文档 文档。
  - `accumulative_counts`: 积累几个反向传播后再优化参数，你可以用它通过小批量来模拟大批量。
- `param_scheduler`: 学习率策略，你可以指定训练期间的学习率和动量曲线。有关详细信息，请参阅 MMEngine 中的 文档。
- `train_cfg | val_cfg | test_cfg`: 训练、验证以及测试的循环执行器配置，请参考相关的MMEngine 文档。

以下是 ResNet50 的训练策略配置’configs/*base*/schedules/imagenet_bs256.py’：

```
optim_wrapper = dict(
    # 使用 SGD 优化器来优化参数
    optimizer=dict(type='SGD', lr=0.1, momentum=0.9, weight_decay=0.0001))

# 学习率参数的调整策略
# 'MultiStepLR' 表示使用多步策略来调度学习率 (LR) 。
param_scheduler = dict(
    type='MultiStepLR', by_epoch=True, milestones=[30, 60, 90], gamma=0.1)

# 训练的配置，迭代 100 个epoch，每一个训练 epoch 后都做验证集评估
# 'by_epoch=True' 默认使用 `EpochBaseLoop`， 'by_epoch=False' 默认使用 `IterBaseLoop`
train_cfg = dict(by_epoch=True, max_epochs=100, val_interval=1)
# 使用默认的验证循环控制器
val_cfg = dict()
# 使用默认的测试循环控制器
test_cfg = dict()

# 通过默认策略自动缩放学习率，此策略适用于总批次大小 256
# 如果你使用不同的总批量大小，比如 512 并启用自动学习率缩放
# 我们将学习率扩大到 2 倍
auto_scale_lr = dict(base_batch_size=256)
```

6.1.4 运行设置

本部分主要包括保存权重策略、日志配置、训练参数、断点权重路径和工作目录等等。

以下是几乎所有算法都使用的运行配置’configs/*base*/default_runtime.py’：

```
# 默认所有注册器使用的域
default_scope = 'mmcls'

# 配置默认的hook
default_hooks = dict(
    # 记录每次迭代的时间。
    timer=dict(type='IterTimerHook'),

    # 每 100 次迭代打印一次日志。
    logger=dict(type='LoggerHook', interval=100),

    # 启用默认参数调度hook。
    param_scheduler=dict(type='ParamSchedulerHook'),

    # 每个epoch保存检查点。
    checkpoint=dict(type='CheckpointHook', interval=1),
```

(下页继续)

(续上页)

```
    # 在分布式环境中设置采样器种子。
    sampler_seed=dict(type='DistSamplerSeedHook'),

    # 验证结果可视化，默认不启用，设置 True 时启用。
    visualization=dict(type='VisualizationHook', enable=False),
)

# 配置环境
env_cfg = dict(
   # 是否开启cudnn benchmark
    cudnn_benchmark=False,

    # 设置多进程参数
    mp_cfg=dict(mp_start_method='fork', opencv_num_threads=0),

    # 设置分布式参数
    dist_cfg=dict(backend='nccl'),
)

# 设置可视化工具
vis_backends = [dict(type='LocalVisBackend')] # 使用磁盘(HDD)后端
visualizer = dict(
    type='ClsVisualizer', vis_backends=vis_backends, name='visualizer')

# 设置日志级别
log_level = 'INFO'

# 从哪个检查点加载
load_from = None

# 是否从加载的检查点恢复训练
resume = False
```

6.2 继承并修改配置文件

为了精简代码、更快的修改配置文件以及便于理解，我们建议继承现有方法。

对于在同一算法文件夹下的所有配置文件，MMClassification 推荐只存在 一个对应的 原始配置文件。所有其他的配置文件都应该继承 原始配置文件，这样就能保证配置文件的最大继承深度为 3。

例如，如果在 ResNet 的基础上做了一些修改，用户首先可以通过指定 `_base_ = './resnet50_8xb32_in1k.py'`（相对于你的配置文件的路径），来继承基础的 ResNet 结构、数据集

以及其他训练配置信息，然后修改配置文件中的必要参数以完成继承。如想在基础 resnet50 的基础上使用 CutMix 训练增强，将训练轮数由 100 改为 300 和修改学习率衰减轮数，同时修改数据集路径，可以建立新的配置文件 configs/resnet/resnet50_8xb32-300e_in1k.py，文件中写入以下内容:

```
# 在 'configs/resnet/' 创建此文件
_base_ = './resnet50_8xb32_in1k.py'

# 模型在之前的基础上使用 CutMix 训练增强
model = dict(
    train_cfg=dict(
        augments=dict(type='CutMix', alpha=1.0)
    )
)

# 优化策略在之前基础上训练更多个 epoch
train_cfg = dict(max_epochs=300, val_interval=10)  # 训练300个 epoch，每10个 epoch␣
↪评估一次
param_scheduler = dict(step=[150, 200, 250])  # 学习率调整也有所变动

# 使用自己的数据集目录
train_dataloader = dict(
    dataset=dict(data_root='mydata/imagenet/train'),
)
val_dataloader = dict(
    batch_size=64,                  # 验证时没有反向传播，可以使用更大的 batchsize
    dataset=dict(data_root='mydata/imagenet/val'),
)
test_dataloader = dict(
    batch_size=64,                  # 测试时没有反向传播，可以使用更大的 batchsize
    dataset=dict(data_root='mydata/imagenet/val'),
)
```

6.2.1 使用配置文件里的中间变量

用一些中间变量，中间变量让配置文件更加清晰，也更容易修改。

例如数据集里的 train_pipeline / test_pipeline 是作为数据流水线的中间变量。我们首先要定义它们，然后将它们传递到 train_dataloader / test_dataloader 中。如果想修改训练或测试时输入图片的大小，就需要修改 train_pipeline / test_pipeline 这些中间变量。

```
bgr_mean = [103.53, 116.28, 123.675]
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='RandomResizedCrop', scale=224, backend='pillow', interpolation='bicubic
↪'),
```

(下页继续)

(续上页)

```
    dict(type='RandomFlip', prob=0.5, direction='horizontal'),
    dict(
        type='RandAugment',
        policies='timm_increasing',
        num_policies=2,
        total_level=10,
        magnitude_level=6,
        magnitude_std=0.5,
        hparams=dict(pad_val=[round(x) for x in bgr_mean], interpolation='bicubic')),
    dict(type='PackClsInputs'),
]

test_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='ResizeEdge', scale=236, edge='short', backend='pillow', interpolation=
↪'bicubic'),
    dict(type='CenterCrop', crop_size=224),
    dict(type='PackClsInputs')
]

train_dataloader = dict(dataset=dict(pipeline=train_pipeline))
val_dataloader = dict(dataset=dict(pipeline=val_pipeline))
test_dataloader = dict(dataset=dict(pipeline=val_pipeline))
```

6.2.2 忽略基础配置文件里的部分内容

有时，您需要设置 `_delete_=True` 去忽略基础配置文件里的一些域内容。可以查看 MMEngine 文档 进一步了解该设计。

以下是一个简单应用案例。如果在上述 ResNet50 案例中使用余弦调度，使用继承并直接修改会报 `get unexcepected keyword 'step'` 错，因为基础配置文件 `param_scheduler` 域信息的 `'step'` 字段被保留下来了，需要加入 `_delete_=True` 去忽略基础配置文件里的 `param_scheduler` 相关域内容：

```
_base_ = '../../configs/resnet/resnet50_8xb32_in1k.py'

# 学习率调整策略
param_scheduler = dict(type='CosineAnnealingLR', by_epoch=True, _delete_=True)
```

6.2.3 引用基础配置文件里的变量

有时，您可以引用 _base_ 配置信息的一些域内容，这样可以避免重复定义。可以查看 MMEngine 文档 进一步了解该设计。

以下是一个简单应用案例，在训练数据预处理流水线中使用 auto augment 数据增强，参考配置文件 configs/resnest/resnest50_32xb64_in1k.py。在定义 train_pipeline 时，可以直接在 _base_ 中加入定义 auto augment 数据增强的文件命名，再通过 {{_base_.auto_increasing_policies}} 引用变量：

```
_base_ = [
    '../_base_/models/resnest50.py', '../_base_/datasets/imagenet_bs64.py',
    '../_base_/default_runtime.py', './_randaug_policies.py',
]

train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(
        type='RandAugment',
        policies={{_base_.policies}},     # 这里使用了 _base_ 里的 `policies` 参数。
        num_policies=2,
        magnitude_level=12),
    dict(type='EfficientNetRandomCrop', scale=224, backend='pillow'),
    dict(type='RandomFlip', prob=0.5, direction='horizontal'),
    dict(type='ColorJitter', brightness=0.4, contrast=0.4, saturation=0.4),
    dict(
        type='Lighting',
        eigval=EIGVAL,
        eigvec=EIGVEC,
        alphastd=0.1,
        to_rgb=False),
    dict(type='PackClsInputs'),
]

train_dataloader = dict(dataset=dict(pipeline=train_pipeline))
```

6.3 通过命令行参数修改配置信息

当用户使用脚本“tools/train.py”或者“tools/test.py”提交任务，以及使用一些工具脚本时，可以通过指定 --cfg-options 参数来直接修改所使用的配置文件内容。

- 更新配置文件内的字典

  可以按照原始配置文件中字典的键的顺序指定配置选项。例如，--cfg-options model.backbone.

`norm_eval=False` 将主干网络中的所有 BN 模块更改为 `train` 模式。

- 更新配置文件内列表的键

  一些配置字典在配置文件中会形成一个列表。例如，训练流水线 `data.train.pipeline` 通常是一个列表。例如，`[dict(type='LoadImageFromFile'), dict(type='TopDownRandomFlip', flip_prob=0.5), ...]` 。如果要将流水线中的 `'flip_prob=0.5'` 更改为 `'flip_prob=0.0'`，您可以这样指定 `--cfg-options data.train.pipeline.1.flip_prob=0.0` 。

- 更新列表/元组的值。

  当配置文件中需要更新的是一个列表或者元组，例如，配置文件通常会设置 `val_evaluator = dict(type='Accuracy', topk=(1, 5))`，用户如果想更改 `topk`，需要指定 `--cfg-options val_evaluator.topk="(1,3)"`。注意这里的引号”对于列表以及元组数据类型的修改是必要的，并且 **不允许**引号内所指定的值的书写存在空格。

CHAPTER 7

如何微调模型

在很多场景下，我们需要快速地将模型应用到新的数据集上，但从头训练模型通常很难快速收敛，这种不确定性会浪费额外的时间。通常，已有的、在大数据集上训练好的模型会比随机初始化提供更为有效的先验信息，粗略来讲，在此基础上的学习我们称之为模型微调。已经证明，在 ImageNet 数据集上预先训练的分类模型对于其他数据集和其他下游任务有很好的效果。

因此，该教程提供了如何将*Model Zoo* 中提供的预训练模型用于其他数据集，已获得更好的效果。

在新数据集上微调模型分为两步：

- 按照数据集准备 添加对新数据集的支持。
- 按照本教程中讨论的内容修改配置文件

假设我们现在有一个在 ImageNet-2012 数据集上训练好的 ResNet-50 模型，并且希望在 CIFAR-10 数据集上进行模型微调，我们需要修改配置文件中的五个部分。

7.1 继承基础配置

首先，创建一个新的配置文件 `configs/tutorial/resnet50_finetune_cifar.py` 来保存我们的配置，当然，这个文件名可以自由设定。

为了重用不同基础配置之间的通用部分，我们支持从多个现有配置中继承配置，其中包括：

- 模型配置：要微调 ResNet-50 模型，可以继承 `_base_/models/resnet50.py` 来搭建模型的基本结构。
- 数据集配置：使用 CIFAR10 数据集，可以继承 `_base_/datasets/cifar10_bs16.py`。

- 训练策略配置：可以继承 batchsize 为 128 的 CIFAR10 数据集基本训练配置文件 `_base_/schedules/cifar10_bs128.py`。
- 运行配置：为了保留运行相关设置，比如默认训练钩子、环境配置等，需要继承 `_base_/default_runtime.py`。

要继承以上这些配置文件，只需要把下面一段代码放在我们的配置文件开头。

```
_base_ = [
    '../_base_/models/resnet50.py',
    '../_base_/datasets/cifar10_bs16.py',
    '../_base_/schedules/cifar10_bs128.py',
    '../_base_/default_runtime.py',
]
```

除此之外，你也可以不使用继承，直接编写完整的配置文件，例如 `configs/lenet/lenet5_mnist.py`。

7.2 修改模型

在进行模型微调时，我们通常希望在主干网络（backbone）加载预训练模型，再用我们的数据集训练一个新的分类头（head）。

为了在主干网络加载预训练模型，我们需要修改主干网络的初始化设置，使用 `Pretrained` 类型的初始化函数。另外，在初始化设置中，我们使用 `prefix='backbone'` 来告诉初始化函数需要加载的子模块的前缀，`backbone` 即指加载模型中的主干网络。方便起见，我们这里使用一个在线的权重文件链接，它会在训练前自动下载对应的文件，你也可以提前下载这个模型，然后使用本地路径。

接下来，新的配置文件需要按照新数据集的类别数目来修改分类头的配置。只需要修改分类头中的 `num_classes` 设置即可。

```
model = dict(
    backbone=dict(
        init_cfg=dict(
            type='Pretrained',
            checkpoint='https://download.openmmlab.com/mmclassification/v0/resnet/
↪resnet50_8xb32_in1k_20210831-ea4938fc.pth',
            prefix='backbone',
        )),
    head=dict(num_classes=10),
)
```

小技巧： 这里我们只需要设定我们想要修改的部分配置，其他配置将会自动从我们的父配置文件中获取。

另外，当新的小数据集和原本预训练的大数据中的数据分布较为类似的话，我们在进行微调时会希望冻结主干网络前面几层的参数，只训练后面层以及分类头的参数，这么做有助于在后续训练中，保持网络从预训练权重中获得的提取低阶特征的能力。在 MMClassification 中，这一功能可以通过简单的一个 `frozen_stages` 参数来实现。比如我们需要冻结前两层网络的参数，只需要在上面的配置中添加一行：

```
model = dict(
    backbone=dict(
        frozen_stages=2,
        init_cfg=dict(
            type='Pretrained',
            checkpoint='https://download.openmmlab.com/mmclassification/v0/resnet/
↪resnet50_8xb32_in1k_20210831-ea4938fc.pth',
            prefix='backbone',
        )),
    head=dict(num_classes=10),
)
```

备注： 目前还不是所有的网络都支持 `frozen_stages` 参数，在使用之前，请先检查 文档 以确认你所使用的主干网络是否支持。

7.3 修改数据集

当针对一个新的数据集进行微调时，我们通常都需要修改一些数据集相关的配置。比如这里，我们就需要把 CIFAR-10 数据集中的图像大小从 32 缩放到 224 来配合 ImageNet 上预训练模型的输入。这一需要可以通过修改数据集的预处理流水线（pipeline）并覆盖数据加载器（dataloader）来实现。

```
# 数据流水线设置
train_pipeline = [
    dict(type='RandomCrop', crop_size=32, padding=4),
    dict(type='RandomFlip', prob=0.5, direction='horizontal'),
    dict(type='Resize', scale=224),
    dict(type='PackClsInputs'),
]
test_pipeline = [
    dict(type='Resize', scale=224),
    dict(type='PackClsInputs'),
]
# 数据加载器设置
train_dataloader = dict(dataset=dict(pipeline=train_pipeline))
val_dataloader = dict(dataset=dict(pipeline=test_pipeline))
test_dataloader = val_dataloader
```

7.4 修改训练策略设置

用于微调任务的超参数与默认配置不同，通常只需要较小的学习率以及较快的衰减策略。

```
# 用于批大小为 128 的优化器学习率
optim_wrapper = dict(
    optimizer=dict(type='SGD', lr=0.01, momentum=0.9, weight_decay=0.0001))
# 学习率衰减策略
param_scheduler = dict(
    type='MultiStepLR', by_epoch=True, milestones=[15], gamma=0.1)
```

小技巧： 更多可修改的细节可以参考如何编写配置文件.

7.5 开始训练

现在，我们完成了用于微调的配置文件，完整的文件如下：

```
_base_ = [
    '../_base_/models/resnet50.py',
    '../_base_/datasets/cifar10_bs16.py',
    '../_base_/schedules/cifar10_bs128.py',
    '../_base_/default_runtime.py',
]

# 模型设置
model = dict(
    backbone=dict(
        frozen_stages=2,
        init_cfg=dict(
            type='Pretrained',
            checkpoint='https://download.openmmlab.com/mmclassification/v0/resnet/
↪resnet50_8xb32_in1k_20210831-ea4938fc.pth',
            prefix='backbone',
        )),
    head=dict(num_classes=10),
)

# 数据集设置
# 数据流水线设置
train_pipeline = [
    dict(type='RandomCrop', crop_size=32, padding=4),
```

(下页继续)

(续上页)

```
    dict(type='RandomFlip', prob=0.5, direction='horizontal'),
    dict(type='Resize', scale=224),
    dict(type='PackClsInputs'),
]
test_pipeline = [
    dict(type='Resize', scale=224),
    dict(type='PackClsInputs'),
]
# 数据加载器设置
train_dataloader = dict(dataset=dict(pipeline=train_pipeline))
val_dataloader = dict(dataset=dict(pipeline=test_pipeline))
test_dataloader = val_dataloader

# 训练策略设置
# 用于批大小为 128 的优化器学习率
optim_wrapper = dict(
    optimizer=dict(type='SGD', lr=0.01, momentum=0.9, weight_decay=0.0001))
# 学习率衰减策略
param_scheduler = dict(
    type='MultiStepLR', by_epoch=True, milestones=[15], gamma=0.1)
```

接下来，我们使用一台 8 张 GPU 的电脑来训练我们的模型，指令如下：

```
bash tools/dist_train.sh configs/tutorial/resnet50_finetune_cifar.py 8
```

当然，我们也可以使用单张 GPU 来进行训练，使用如下命令：

```
python tools/train.py configs/tutorial/resnet50_finetune_cifar.py
```

但是如果我们使用单张 GPU 进行训练的话，需要在数据集设置部分作如下修改：

```
train_dataloader = dict(
    batch_size=128,
    dataset=dict(pipeline=train_pipeline),
)
val_dataloader = dict(
    batch_size=128,
    dataset=dict(pipeline=test_pipeline),
)
test_dataloader = val_dataloader
```

这是因为我们的训练策略是针对批次大小（batch size）为 128 设置的。在父配置文件中，设置了单张 `batch_size=16`，如果使用 8 张 GPU，总的批次大小就是 128。而如果使用单张 GPU，就必须手动修改 `batch_size=128` 来匹配训练策略。

CHAPTER 8

数据集可视化

8.1 数据集可视化工具简介

```
python tools/visualizations/browse_dataset.py \
    ${CONFIG_FILE} \
    [-o, --output-dir ${OUTPUT_DIR}] \
    [-p, --phase ${DATASET_PHASE}] \
    [-n, --show-number ${NUMBER_IMAGES_DISPLAY}] \
    [-i, --show-interval ${SHOW_INTERRVAL}] \
    [-m, --mode ${DISPLAY_MODE}] \
    [-r, --rescale-factor ${RESCALE_FACTOR}] \
    [-c, --channel-order ${CHANNEL_ORDER}] \
    [--cfg-options ${CFG_OPTIONS}]
```

所有参数的说明:

- `config`: 模型配置文件的路径。
- `-o, --output-dir`: 保存图片文件夹，如果没有指定，默认为 `''`，表示不保存。
- **`-p, --phase`**: 可视化数据集的阶段，只能为 `['train', 'val', 'test']` 之一，默认为 `'train'`。
- **`-n, --show-number`**: 可视化样本数量。如果没有指定，默认展示数据集的所有图片。
- `-i, --show-interval`: 浏览时，每张图片的停留间隔，单位为秒。

- **-m, --mode**: 可视化的模式，只能为 ['original', 'transformed', 'concat', 'pipeline'] 之一。默认为'transformed'.
- **-r, --rescale-factor**: 对可视化图片的放缩倍数，在图片过大或过小时设置。
- -c, --channel-order: 图片的通道顺序，为 ['BGR', 'RGB'] 之一，默认为 'BGR'。
- --cfg-options : 对配置文件的修改，参考学习配置文件。

备注:

1. -m, --mode 用于设置可视化的模式，默认设置为‘transformed’。
   - 如果 --mode 设置为‘original’，则获取原始图片；
   - 如果 --mode 设置为‘transformed’，则获取预处理后的图片；
   - 如果 --mode 设置为‘concat’，获取原始图片和预处理后图片拼接的图片；
   - 如果 --mode 设置为‘pipeline’，则获得数据流水线所有中间过程图片。
2. -r, --rescale-factor 在数据集中图片的分辨率过大或者过小时设置。比如在可视化 CIFAR 数据集时，由于图片的分辨率非常小，可将 -r, --rescale-factor 设置为 10。

8.2 如何可视化原始图像

使用 **‘original’**模式：

```
python ./tools/visualizations/browse_dataset.py ./configs/resnet/resnet101_8xb16_
↪cifar10.py --phase val --output-dir tmp --mode original --show-number 100 --rescale-
↪factor 10 --channel-order RGB
```

- --phase val: 可视化验证集, 可简化为 -p val;
- --output-dir tmp: 可视化结果保存在“tmp”文件夹, 可简化为 -o tmp;
- --mode original: 可视化原图, 可简化为 -m original;
- --show-number 100: 可视化 100 张图，可简化为 -n 100;
- --rescale-factor: 图像放大 10 倍，可简化为 -r 10;
- --channel-order RGB: 可视化图像的通道顺序为“RGB”, 可简化为 -c RGB。

8.3 如何可视化处理后图像

使用 “**transformed**” 模式：

```
python ./tools/visualizations/browse_dataset.py ./configs/resnet/resnet50_8xb32_in1k.
↪py -n 100 -r 2
```

8.4 如何同时可视化原始图像与处理后图像

使用 “**concat**” 模式：

```
python ./tools/visualizations/browse_dataset.py configs/swin_transformer/swin-small_
↪16xb64_in1k.py -n 10 -m concat
```

使用 “**pipeline**” 模式：

```
python ./tools/visualizations/browse_dataset.py configs/swin_transformer/swin-small_
↪16xb64_in1k.py -m pipeline
```

CHAPTER 9

优化器参数策略可视化

该工具旨在帮助用户检查优化器的超参数调度器（无需训练），支持学习率（learning rate）和动量（momentum）。

9.1 工具简介

```
python tools/visualizations/vis_scheduler.py \
    ${CONFIG_FILE} \
    [-p, --parameter ${PARAMETER_NAME}] \
    [-d, --dataset-size ${DATASET_SIZE}] \
    [-n, --ngpus ${NUM_GPUs}] \
    [-s, --save-path ${SAVE_PATH}] \
    [--title ${TITLE}] \
    [--style ${STYLE}] \
    [--window-size ${WINDOW_SIZE}] \
    [--cfg-options]
```

所有参数的说明：

- `config`：模型配置文件的路径。
- **`-p, parameter`**：可视化参数名，只能为 `["lr", "momentum"]` 之一，默认为 `"lr"`.
- **`-d, --dataset-size`**：数据集的大小。如果指定，`build_dataset` 将被跳过并使用这个大小作为数据集大小，默认使用 `build_dataset` 所得数据集的大小。
- **`-n, --ngpus`**：使用 GPU 的数量，默认为 1。

- **-s, --save-path**: 保存的可视化图片的路径，默认不保存。
- --title: 可视化图片的标题，默认为配置文件名。
- --style: 可视化图片的风格，默认为 whitegrid。
- --window-size: 可视化窗口大小，如果没有指定，默认为 12*7。如果需要指定，按照格式 'W*H'。
- --cfg-options: 对配置文件的修改，参考学习配置文件。

备注： 部分数据集在解析标注阶段比较耗时，可直接将 -d, dataset-size 指定数据集的大小，以节约时间。

9.2 如何在开始训练前可视化学习率曲线

你可以使用如下命令来绘制配置文件 configs/resnet/resnet50_b16x8_cifar100.py 将会使用的变化率曲线：

```
python tools/visualizations/vis_scheduler.py configs/resnet/resnet50_b16x8_cifar100.py
```

当数据集为 ImageNet 时，通过直接指定数据集大小来节约时间，并保存图片：

```
python tools/visualizations/vis_scheduler.py configs/repvgg/repvgg-B3g4_4xb64-autoaug-
↪lbs-mixup-coslr-200e_in1k.py --dataset-size 1281167 --ngpus 4 --save-path ./repvgg-
↪B3g4_4xb64-lr.jpg
```

CHAPTER 10

类别激活图（CAM）可视化

10.1 类别激活图可视化工具介绍

MMClassification 提供 `tools\visualizations\vis_cam.py` 工具来可视化类别激活图。请使用 `pip install "grad-cam>=1.3.6"` 安装依赖的 pytorch-grad-cam。

目前支持的方法有：

Method	What it does
GradCAM	使用平均梯度对 2D 激活进行加权
Grad-CAM++	类似 GradCAM，但使用了二阶梯度
XGradCAM	类似 GradCAM，但通过归一化的激活对梯度进行了加权
EigenCAM	使用 2D 激活的第一主成分（无法区分类别，但效果似乎不错）
EigenGrad-CAM	类似 EigenCAM，但支持类别区分，使用了激活 * 梯度的第一主成分，看起来和 GradCAM 差不多，但是更干净
LayerCAM	使用正梯度对激活进行空间加权，对于浅层有更好的效果

命令行：

```
python tools/visualizations/vis_cam.py \
    ${IMG} \
    ${CONFIG_FILE} \
    ${CHECKPOINT} \
```

(下页继续)

(续上页)

```
    [--target-layers ${TARGET-LAYERS}] \
    [--preview-model] \
    [--method ${METHOD}] \
    [--target-category ${TARGET-CATEGORY}] \
    [--save-path ${SAVE_PATH}] \
    [--vit-like] \
    [--num-extra-tokens ${NUM-EXTRA-TOKENS}]
    [--aug_smooth] \
    [--eigen_smooth] \
    [--device ${DEVICE}] \
    [--cfg-options ${CFG-OPTIONS}]
```

所有参数的说明:

- `img`：目标图片路径。
- `config`：模型配置文件的路径。
- `checkpoint`：权重路径。
- `--target-layers`：所查看的网络层名称，可输入一个或者多个网络层, 如果不设置，将使用最后一个 `block` 中的 `norm` 层。
- `--preview-model`：是否查看模型所有网络层。
- `--method`：类别激活图图可视化的方法，目前支持 `GradCAM`, `GradCAM++`, `XGradCAM`, `EigenCAM`, `EigenGradCAM`, `LayerCAM`，不区分大小写。如果不设置，默认为 `GradCAM`。
- `--target-category`：查看的目标类别，如果不设置，使用模型检测出来的类别做为目标类别。
- `--save-path`：保存的可视化图片的路径，默认不保存。
- `--eigen-smooth`：是否使用主成分降低噪音，默认不开启。
- `--vit-like`: 是否为 `ViT` 类似的 Transformer-based 网络
- `--num-extra-tokens`: `ViT` 类网络的额外的 tokens 通道数，默认使用主干网络的 `num_extra_tokens`。
- `--aug-smooth`：是否使用测试时增强
- `--device`：使用的计算设备，如果不设置，默认为’cpu’。
- `--cfg-options`：对配置文件的修改，参考学习配置文件。

备注: 在指定 `--target-layers` 时，如果不知道模型有哪些网络层，可使用命令行添加 `--preview-model` 查看所有网络层名称;

10.2 如何可视化 CNN 网络的类别激活图（如 ResNet-50）

`--target-layers` 在 `Resnet-50` 中的一些示例如下:

- `'backbone.layer4'`，表示第四个 `ResLayer` 层的输出。
- `'backbone.layer4.2'` 表示第四个 `ResLayer` 层中第三个 `BottleNeck` 块的输出。
- `'backbone.layer4.2.conv1'` 表示上述 `BottleNeck` 块中 `conv1` 层的输出。

备注： 对于 `ModuleList` 或者 `Sequential` 类型的网络层，可以直接使用索引的方式指定子模块。比如 `backbone.layer4[-1]` 和 `backbone.layer4.2` 是相同的，因为 `layer4` 是一个拥有三个子模块的 `Sequential`。

---

1. 使用不同方法可视化 `ResNet50`，默认 `target-category` 为模型检测的结果，使用默认推导的 `target-layers`。

```
python tools/visualizations/vis_cam.py \
    demo/bird.JPEG \
    configs/resnet/resnet50_8xb32_in1k.py \
    https://download.openmmlab.com/mmclassification/v0/resnet/resnet50_batch256_
↪imagenet_20200708-cfb998bf.pth \
    --method GradCAM
    # GradCAM++, XGradCAM, EigenCAM, EigenGradCAM, LayerCAM
```

Image	GradCAM	GradCAM++	EigenGradCAM	LayerCAM

2. 同一张图不同类别的激活图效果图，在 `ImageNet` 数据集中，类别 238 为‘Greater Swiss Mountain dog’，类别 281 为‘tabby, tabby cat’。

```
python tools/visualizations/vis_cam.py \
    demo/cat-dog.png configs/resnet/resnet50_8xb32_in1k.py \
    https://download.openmmlab.com/mmclassification/v0/resnet/resnet50_batch256_
↪imagenet_20200708-cfb998bf.pth \
    --target-layers 'backbone.layer4.2' \
    --method GradCAM \
    --target-category 238
    # --target-category 281
```

Category	Image	GradCAM	XGradCAM	LayerCAM
Dog				
Cat				

3. 使用 `--eigen-smooth` 以及 `--aug-smooth` 获取更好的可视化效果。

```
python tools/visualizations/vis_cam.py \
    demo/dog.jpg  \
    configs/mobilenet_v3/mobilenet-v3-large_8xb128_in1k.py \
    https://download.openmmlab.com/mmclassification/v0/mobilenet_v3/convert/
↪mobilenet_v3_large-3ea3c186.pth \
    --target-layers 'backbone.layer16' \
    --method LayerCAM \
    --eigen-smooth --aug-smooth
```

Image	LayerCAM	eigen-smooth	aug-smooth	eigen&aug

10.3 如何可视化 Transformer 类型网络的类别激活图

`--target-layers` 在 Transformer-based 网络中的一些示例如下:

- Swin-Transformer 中: `'backbone.norm3'`
- ViT 中: `'backbone.layers[-1].ln1'`

对于 Transformer-based 的网络，比如 ViT、T2T-ViT 和 Swin-Transformer，特征是被展平的。为了绘制 CAM 图，我们需要指定 `--vit-like` 选项，从而让被展平的特征恢复方形的特征图。

除了特征被展平之外，一些类 ViT 的网络还会添加额外的 tokens。比如 ViT 和 T2T-ViT 中添加了分类 token，DeiT 中还添加了蒸馏 token。在这些网络中，分类计算在最后一个注意力模块之后就已经完成了，分类得分也只和这些额外的 tokens 有关，与特征图无关，也就是说，分类得分对这些特征图的导数为 0。因此，我们不能使用最后一个注意力模块的输出作为 CAM 绘制的目标层。

另外，为了去除这些额外的 toekns 以获得特征图，我们需要知道这些额外 tokens 的数量。MMClassification 中几乎所有 Transformer-based 的网络都拥有 `num_extra_tokens` 属性。而如果你希望将此工具应用于新的，或者第三方的网络，而且该网络没有指定 `num_extra_tokens` 属性，那么可以使用 `--num-extra-tokens` 参数手动指定其数量。

1. 对 `Swin Transformer` 使用默认 `target-layers` 进行 CAM 可视化:

```
python tools/visualizations/vis_cam.py \
    demo/bird.JPEG  \
    configs/swin_transformer/swin-tiny_16xb64_in1k.py \
    https://download.openmmlab.com/mmclassification/v0/swin-transformer/swin_tiny_
↪224_b16x64_300e_imagenet_20210616_090925-66df6be6.pth \
    --vit-like
```

2. 对 Vision Transformer(ViT) 进行 CAM 可视化：

```
python tools/visualizations/vis_cam.py \
    demo/bird.JPEG  \
    configs/vision_transformer/vit-base-p16_ft-64xb64_in1k-384.py \
    https://download.openmmlab.com/mmclassification/v0/vit/finetune/vit-base-p16_
↪in21k-pre-3rdparty_ft-64xb64_in1k-384_20210928-98e8652b.pth \
    --vit-like \
    --target-layers 'backbone.layers[-1].ln1'
```

3. 对 T2T-ViT 进行 CAM 可视化：

```
python tools/visualizations/vis_cam.py \
    demo/bird.JPEG  \
    configs/t2t_vit/t2t-vit-t-14_8xb64_in1k.py \
    https://download.openmmlab.com/mmclassification/v0/t2t-vit/t2t-vit-t-14_
↪3rdparty_8xb64_in1k_20210928-b7c09b62.pth \
    --vit-like \
    --target-layers 'backbone.encoder[-1].ln1'
```

Image	ResNet50	ViT	Swin	T2T-ViT

CHAPTER 11

打印完整配置文件

print_config.py 脚本脚本会解析所有输入变量，并打印完整配置信息。

11.1 说明：

tools/misc/print_config.py 脚本会逐字打印整个配置文件，并展示所有导入的文件。

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

所有参数的说明：

- config：模型配置文件的路径。
- --cfg-options:: 额外的配置选项，会被合入配置文件，参考学习配置文件。

11.2 示例：

打印 configs/t2t_vit/t2t-vit-t-14_8xb64_in1k.py 文件的完整配置

```
# 打印完成的配置文件
python tools/misc/print_config.py configs/t2t_vit/t2t-vit-t-14_8xb64_in1k.py

# 将完整的配置文件保存为一个独立的配置文件
python tools/misc/print_config.py configs/t2t_vit/t2t-vit-t-14_8xb64_in1k.py > final_
↪config.py
```

(下页继续)

(续上页)

CHAPTER 12

数据集验证

在 MMClassification 中，`tools/misc/verify_dataset.py` 脚本会检查数据集的所有图片，查看是否有**已经损坏**的图片。

12.1 工具介绍

```
python tools/print_config.py \
    ${CONFIG} \
    [--out-path ${OUT-PATH}] \
    [--phase ${PHASE}] \
    [--num-process ${NUM-PROCESS}]
    [--cfg-options ${CFG_OPTIONS}]
```

所有参数说明:

- `config`：配置文件的路径。
- `--out-path`：输出结果路径，默认为‘brokenfiles.log’。
- `--phase`：检查哪个阶段的数据集，可用值为“train”、”test”或者“val”，默认为“train”。
- `--num-process`：指定的进程数，默认为 1。
- `--cfg-options`: 额外的配置选项，会被合入配置文件，参考教程 1：如何编写配置文件。

12.2 示例：

```
python tools/misc/verify_dataset.py configs/t2t_vit/t2t-vit-t-14_8xb64_in1k.py --out-
↪path broken_imgs.log --phase val --num-process 8
```

CHAPTER 13

日志分析工具

13.1 日志分析

13.1.1 日志分析工具介绍

`tools/analysis_tools/analyze_logs.py` 脚本绘制指定键值的变化曲线。

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

所有参数的说明：

- `json_logs`：模型配置文件的路径（可同时传入多个，使用空格分开）。
- `--keys`：分析日志的关键字段，数量为 `len(${JSON_LOGS}) * len(${KEYS})` 默认为 'loss'。
- `--title`：分析日志的图片名称，默认使用配置文件名，默认为空。
- `--legend`：图例的名称，其数目必须与相等 `len(${JSON_LOGS}) * len(${KEYS})`。默认使用 `"${JSON_LOG}-${KEYS}"`.

- `--backend` : matplotlib 的绘图后端，默认由 matplotlib 自动选择。
- `--style` : 绘图配色风格，默认为 `whitegrid`。
- `--out` : 保存分析图片的路径，如不指定则不保存。
- `--window-size`: 可视化窗口大小，如果没有指定，默认为 `'12*7'`。如果需要指定，需按照格式 `'W*H'`。

备注： The `--style` option depends on `seaborn` package, please install it before setting it.

13.1.2 如何或绘制损失/精度曲线

我们将给出一些示例，来展示如何使用 `tools/analysis_tools/analyze_logs.py` 脚本绘制精度曲线的损失曲线

绘制某日志文件对应的损失曲线图

```
python tools/analysis_tools/analyze_logs.py plot_curve your_log_json --keys loss --
↪legend loss
```

绘制某日志文件对应的 top-1 和 top-5 准确率曲线图，并将曲线图导出为 results.jpg 文件。

```
python tools/analysis_tools/analyze_logs.py plot_curve your_log_json --keys accuracy/
↪top1 accuracy/top5  --legend top1 top5 --out results.jpg
```

在同一图像内绘制两份日志文件对应的 top-1 准确率曲线图。

```
python tools/analysis_tools/analyze_logs.py plot_curve log1.json log2.json --keys␣
↪accuracy_top-1 --legend exp1 exp2
```

13.1.3 如何统计训练时间

`tools/analysis_tools/analyze_logs.py` 也可以根据日志文件统计训练耗时。

```
python tools/analysis_tools/analyze_logs.py cal_train_time \
    ${JSON_LOGS}
    [--include-outliers]
```

所有参数的说明：

- `json_logs`：模型配置文件的路径（可同时传入多个，使用空格分开）。
- `--include-outliers`：如果指定，将不会排除每个轮次中第一个时间记录（有时第一轮迭代会耗时较长）。

示例：

```
python tools/analysis_tools/analyze_logs.py cal_train_time work_dirs/your_exp/
↪20230206_181002/vis_data/scalars.json
```

预计输出结果如下所示：

```
-----Analyze train time of work_dirs/your_exp/20230206_181002/vis_data/scalars.json---
↪--
slowest epoch 68, average time is 0.3818
fastest epoch 1, average time is 0.3694
time std over epochs is 0.0020
average iter time: 0.3777 s/iter
```

13.2 结果分析

利用 `tools/test.py` 的`--out`，我们可以将所有的样本的推理结果保存到输出文件中。利用这一文件，我们可以进行进一步的分析。

13.2.1 如何进行离线度量评估

我们提供了 `tools/analysis_tools/eval_metric.py` 脚本，使用户能够根据预测文件评估模型。

```
python tools/analysis_tools/eval_metric.py \
      ${RESULT} \
      [--metric ${METRIC_OPTIONS} ...]  \
```

所有参数说明：

- `result`：`tools/test.py` 输出的结果文件。
- `--metric`：用于评估结果的指标，请至少指定一个指标，并且你可以通过指定多个 `--metric` 来同时计算多个指标。

请参考评估文档选择可用的评估指标和对应的选项。

备注： 在 `tools/test.py` 中，我们支持使用 `--out-item` 选项来选择保存何种结果至输出文件。请确保没有额外指定 `--out-item`，或指定了 `--out-item=pred`。

示例:

```
# 获取结果文件
python tools/test.py configs/resnet/resnet18_8xb16_cifar10.py \
    https://download.openmmlab.com/mmclassification/v0/resnet/resnet18_b16x8_cifar10_
↪20210528-bd6371c8.pth \
    --out results.pkl

# 计算 top-1 和 top-5 准确率
python tools/analysis_tools/eval_metric.py results.pkl --metric type=Accuracy topk=1,5

# 计算准确率、精确度、召回率、F1-score
python tools/analysis_tools/eval_metric.py results.pkl --metric type=Accuracy \
    --metric type=SingleLabelMetric items=precision,recall,f1-score
```

13.2.2 如何将预测结果可视化

我们可以使用脚本 `tools/analysis_tools/analyze_results.py` 来保存预测成功或失败时得分最高的图像。

```
python tools/analysis_tools/analyze_results.py \
      ${CONFIG} \
      ${RESULT} \
      [--out-dir ${OUT_DIR}] \
      [--topk ${TOPK}] \
      [--rescale-factor ${RESCALE_FACTOR}] \
      [--cfg-options ${CFG_OPTIONS}]
```

所有参数说明：：

- `config`：配置文件的路径。
- `result`：`tools/test.py` 的输出结果文件。
- `--out_dir`：保存结果分析的文件夹路径。
- `--topk`：分别保存多少张预测成功/失败的图像。如果不指定，默认为 `20`。
- `--rescale-factor`：图像的缩放系数，如果样本图像过大或过小时可以使用（过小的图像可能导致结果标签非常模糊）。
- `--cfg-options`：额外的配置选项，会被合入配置文件，参考学习配置文件。

备注： 在 `tools/test.py` 中，我们支持使用 `--out-item` 选项来选择保存何种结果至输出文件。请确保没有额外指定 `--out-item`，或指定了 `--out-item=pred`。

示例:

```
# 获取预测结果文件
python tools/test.py configs/resnet/resnet18_8xb16_cifar10.py \
    https://download.openmmlab.com/mmclassification/v0/resnet/resnet18_b16x8_cifar10_
↪20210528-bd6371c8.pth \
    --out results.pkl

# 保存预测成功/失败的图像中，得分最高的前 10 张，并在可视化时将输出图像放大 10 倍。
python tools/analysis_tools/analyze_results.py \
       configs/resnet/resnet18_8xb16_cifar10.py \
       results.pkl \
       --out-dir output \
       --topk 10 \
       --rescale-factor 10
```

CHAPTER 14

模型复杂度分析

14.1 计算 FLOPs 和参数数量（实验性的）

我们根据 fvcore 提供了一个脚本用于计算给定模型的 FLOPs 和参数量。

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

所有参数说明：

- `config`：配置文件的路径。
- `--shape`：输入尺寸，支持单值或者双值，如：`--shape 256`、`--shape 224 256`。默认为 `224 224`。

你将获得如下结果：

```
==============================
Input shape: (3, 224, 224)
Flops: 17.582G
Params: 91.234M
Activation: 23.895M
==============================
```

同时，你会得到每层的详细复杂度信息，如下所示：

```
| module                                      | #parameters or shape   | #flops    |
↪#activations   |
|:--------------------------------------------|:-----------------------|:----------|:---
↪------------|
| model                                       | 91.234M                | 17.582G   | 23.
↪895M        |
|  backbone                                   |  85.799M               |  17.582G  | ␣
↪23.895M       |
|   backbone.cls_token                        |   (1, 1, 768)          |           |   ␣
↪             |
|   backbone.pos_embed                        |   (1, 197, 768)        |           |   ␣
↪             |
|   backbone.patch_embed.projection           |   0.591M               |   0.116G  |  ␣
↪0.151M       |
|    backbone.patch_embed.projection.weight |    (768, 3, 16, 16)    |           |   ␣
↪             |
|    backbone.patch_embed.projection.bias   |    (768,)              |           |   ␣
↪             |
|   backbone.layers                           |   85.054M              |   17.466G |  ␣
↪23.744M      |
|    backbone.layers.0                        |    7.088M              |    1.455G |   ␣
↪1.979M       |
|    backbone.layers.1                        |    7.088M              |    1.455G |   ␣
↪1.979M       |
|    backbone.layers.2                        |    7.088M              |    1.455G |   ␣
↪1.979M       |
|    backbone.layers.3                        |    7.088M              |    1.455G |   ␣
↪1.979M       |
|    backbone.layers.4                        |    7.088M              |    1.455G |   ␣
↪1.979M       |
|    backbone.layers.5                        |    7.088M              |    1.455G |   ␣
↪1.979M       |
|    backbone.layers.6                        |    7.088M              |    1.455G |   ␣
↪1.979M       |
|    backbone.layers.7                        |    7.088M              |    1.455G |   ␣
↪1.979M       |
|    backbone.layers.8                        |    7.088M              |    1.455G |   ␣
↪1.979M       |
|    backbone.layers.9                        |    7.088M              |    1.455G |   ␣
↪1.979M       |
|    backbone.layers.10                       |    7.088M              |    1.455G |   ␣
↪1.979M       |
|    backbone.layers.11                       |    7.088M              |    1.455G |   ␣
↪1.979M       |
```

(下页继续)

(续上页)

```
|   backbone.ln1                          |   1.536K              |   0.756M  |  ␣
↪0            |
|    backbone.ln1.weight                  |    (768,)             |           |  ␣
↪             |
|    backbone.ln1.bias                    |    (768,)             |           |  ␣
↪             |
|  head.layers                            |  5.435M               |           |  ␣
↪             |
|   head.layers.pre_logits                |   2.362M              |           |  ␣
↪             |
|    head.layers.pre_logits.weight        |    (3072, 768)        |           |  ␣
↪             |
|    head.layers.pre_logits.bias          |    (3072,)            |           |  ␣
↪             |
|   head.layers.head                      |   3.073M              |           |  ␣
↪             |
|    head.layers.head.weight              |    (1000, 3072)       |           |  ␣
↪             |
|    head.layers.head.bias                |    (1000,)            |           |  ␣
↪             |
```

警告: 警告

此工具仍处于试验阶段，我们不保证该数字正确无误。您最好将结果用于简单比较，但在技术报告或论文中采用该结果之前，请仔细检查。

- FLOPs 与输入的尺寸有关，而参数量与输入尺寸无关。默认输入尺寸为 (1, 3, 224, 224)
- 一些运算不会被计入 FLOPs 的统计中，例如某些自定义运算。详细信息请参考 `fvcore.nn.flop_count._DEFAULT_SUPPORTED_OPS`。

CHAPTER 15

TorchServe 部署

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

15.1 1. 转换 MMClassification 模型至 TorchServe

```
python tools/torchserve/mmcls2torchserve.py ${CONFIG_FILE} ${CHECKPOINT_FILE} \
--output-folder ${MODEL_STORE} \
--model-name ${MODEL_NAME}
```

备注： ${MODEL_STORE} 需要是一个文件夹的绝对路径。

示例：

```
python tools/torchserve/mmcls2torchserve.py \
  configs/resnet/resnet18_8xb32_in1k.py \
  checkpoints/resnet18_8xb32_in1k_20210831-fbbb1da6.pth \
  --output-folder ./checkpoints \
  --model-name resnet18_in1k
```

15.2 2. 构建 mmcls-serve docker 镜像

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

15.3 3. 运行 mmcls-serve 镜像

请参考官方文档 基于 docker 运行 TorchServe.

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

示例：

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

备注： realpath ./checkpoints 是“./checkpoints”的绝对路径，你可以将其替换为你保存 TorchServe 模型的目录的绝对路径。

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

15.4 4. 测试部署

```
curl http://127.0.0.1:8080/predictions/${MODEL_NAME} -T demo/demo.JPEG
```

您应该获得类似于以下内容的响应：

```
{
  "pred_label": 58,
  "pred_score": 0.38102269172668457,
  "pred_class": "water snake"
}
```

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

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

示例：

```
python tools/torchserve/test_torchserver.py \
  demo/demo.JPEG \
  configs/resnet/resnet18_8xb32_in1k.py \
  checkpoints/resnet18_8xb32_in1k_20210831-fbbb1da6.pth \
  resnet18_in1k
```

CHAPTER 16

添加新数据集

用户可以编写一个继承自 BasesDataset 的新数据集类，并重载 load_data_list(self) 方法，类似 CIFAR10 和 ImageNet。

通常，此方法返回一个包含所有样本的列表，其中的每个样本都是一个字典。字典中包含了必要的数据信息，例如 img 和 gt_label。

假设我们将要实现一个 Filelist 数据集，该数据集将使用文件列表进行训练和测试。注释列表的格式如下：

```
000001.jpg 0
000002.jpg 1
...
```

16.1 1. 创建数据集类

我们可以在 mmcls/datasets/filelist.py 中创建一个新的数据集类以加载数据。

```
from mmcls.registry import DATASETS
from .base_dataset import BaseDataset


@DATASETS.register_module()
class Filelist(BaseDataset):
```

(下页继续)

(续上页)

```
    def load_data_list(self):
        assert isinstance(self.ann_file, str)

        data_list = []
        with open(self.ann_file) as f:
            samples = [x.strip().split(' ') for x in f.readlines()]
            for filename, gt_label in samples:
                img_path = add_prefix(filename, self.img_prefix)
                info = {'img_path': img_path, 'gt_label': int(gt_label)}
                data_list.append(info)
        return data_list
```

16.2 2. 添加进 MMCls 库

将新的数据集类加入到 `mmcls/datasets/__init__.py` 中:

```
from .base_dataset import BaseDataset
...
from .filelist import Filelist

__all__ = [
    'BaseDataset', ... ,'Filelist'
]
```

16.2.1 3. 修改相关配置文件

然后在配置文件中，为了使用 `Filelist`，用户可以按以下方式修改配置

```
train_dataloader = dict(
    ...
    dataset=dict(
        type='Filelist',
        ann_file='image_list.txt',
        pipeline=train_pipeline,
    )
)
```

所有继承 `BaseDataset` 的数据集类都具有**懒加载**以及**节省内存**的特性，可以参考相关文档 mmengine.basedataset。

备注： 如果数据样本时获取的字典中，只包含了‘img_path’不包含‘img’，则在 pipeline 中必须包含‘LoadImgFromFile’。

CHAPTER 17

自定义数据处理流程（待更新）

请参见英文文档，如果你有兴趣参与中文文档的翻译，欢迎在 讨论区进行报名。

CHAPTER 18

自定义模型

在我们的设计中，我们定义一个完整的模型为 ImageClassifer。根据功能的不同，一个 ImageClassifer 基本由以下 4 种类型的模型组件组成。

- 主干网络：通常是一个特征提取网络，涵盖了模型直接绝大多数的差异，例如 ResNet、MobileNet。
- 颈部：用于连接主干网络和头部的组件，例如 GlobalAveragePooling。
- 头部：用于执行特定任务的组件，例如分类和回归。
- 损失函数：在头部用于计算损失函数的组件，例如 CrossEntropyLoss、LabelSmoothLoss。

18.1 添加新的主干网络

这里，我们以 ResNet_CIFAR 为例，展示了如何开发一个新的主干网络组件。

ResNet_CIFAR 针对 CIFAR 32x32 的图像输入，远小于大多数模型使用的 ImageNet 默认的 224x224 输入配置，所以我们将骨干网络中 kernel_size=7,stride=2 的设置替换为 kernel_size=3, stride=1，并移除了 stem 层之后的 MaxPooling，以避免传递过小的特征图到残差块中。

最简单的方式就是继承自 ResNet 并只修改 stem 层。

1. 创建一个新文件 mmcls/models/backbones/resnet_cifar.py。

```
import torch.nn as nn

from mmcls.registry import MODELS
```

(下页继续)

(续上页)

```python
from .resnet import ResNet


@MODELS.register_module()
class ResNet_CIFAR(ResNet):

    """ResNet backbone for CIFAR.

    （对这个主干网络的简短描述）

    Args:
        depth(int): Network depth, from {18, 34, 50, 101, 152}.
        ...
         （参数文档）
    """

    def __init__(self, depth, deep_stem=False, **kwargs):
        # 调用基类 ResNet 的初始化函数
        super(ResNet_CIFAR, self).__init__(depth, deep_stem=deep_stem **kwargs)
        # 其他特殊的初始化流程
        assert not self.deep_stem, 'ResNet_CIFAR do not support deep_stem'

    def _make_stem_layer(self, in_channels, base_channels):
        # 重载基类的方法，以实现对网络结构的修改
        self.conv1 = build_conv_layer(
            self.conv_cfg,
            in_channels,
            base_channels,
            kernel_size=3,
            stride=1,
            padding=1,
            bias=False)
        self.norm1_name, norm1 = build_norm_layer(
            self.norm_cfg, base_channels, postfix=1)
        self.add_module(self.norm1_name, norm1)
        self.relu = nn.ReLU(inplace=True)

    def forward(self, x):
        # 如果需要的话，可以自定义forward方法
        x = self.conv1(x)
        x = self.norm1(x)
        x = self.relu(x)
        outs = []
```

(下页继续)

(续上页)

```
        for i, layer_name in enumerate(self.res_layers):
            res_layer = getattr(self, layer_name)
            x = res_layer(x)
            if i in self.out_indices:
                outs.append(x)
        # 输出值需要是一个包含不同层多尺度输出的元组
        # 如果不需要多尺度特征，可以直接在最终输出上包一层元组
        return tuple(outs)

    def init_weights(self):
        # 如果需要的话，可以自定义权重初始化的方法
        super().init_weights()

        # 如果有预训练模型，则不需要进行权重初始化
        if self.init_cfg is not None and self.init_cfg['type'] == 'Pretrained':
            return

        # 通常来说，我们建议用`init_cfg`去列举不同层权重初始化方法
        # 包括卷积层，线性层，归一化层等等
        # 如果有特殊需要，可以在这里进行额外的初始化操作
        ...
```

备注： 在 OpenMMLab 2.0 的设计中，将原有的 BACKBONES、NECKS、HEADS、LOSSES 等注册名统一为 MODELS.

2. 在 mmcls/models/backbones/__init__.py 中导入新模块

```
...
from .resnet_cifar import ResNet_CIFAR

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

3. 在配置文件中使用新的主干网络

```
model = dict(
    ...
    backbone=dict(
        type='ResNet_CIFAR',
        depth=18,
        other_arg=xxx),
```

(下页继续)

(续上页)

```
    ...
```

18.2 添加新的颈部组件

这里我们以 GlobalAveragePooling 为例。这是一个非常简单的颈部组件，没有任何参数。

要添加新的颈部组件，我们主要需要实现 forward 函数，该函数对主干网络的输出进行一些操作并将结果传递到头部。

1. 创建一个新文件 mmcls/models/necks/gap.py

```python
import torch.nn as nn

from mmcls.registry import MODELS

@MODELS.register_module()
class GlobalAveragePooling(nn.Module):

    def __init__(self):
        self.gap = nn.AdaptiveAvgPool2d((1, 1))

    def forward(self, inputs):
        # 简单起见，我们默认输入是一个张量
        outs = self.gap(inputs)
        outs = outs.view(inputs.size(0), -1)
        return outs
```

2. 在 mmcls/models/necks/__init__.py 中导入新模块

```
...
from .gap import GlobalAveragePooling

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

3. 修改配置文件以使用新的颈部组件

```
model = dict(
    neck=dict(type='GlobalAveragePooling'),
)
```

18.3 添加新的头部组件

在此，我们以一个简化的 VisionTransformerClsHead 为例，说明如何开发新的头部组件。

要添加一个新的头部组件，基本上我们需要实现 pre_logits 函数用于进入最后的分类头之前需要的处理，以及 forward 函数。

1. 创建一个文件 mmcls/models/heads/vit_head.py.

```python
import torch.nn as nn

from mmcls.registry import MODELS
from .cls_head import ClsHead


@MODELS.register_module()
class LinearClsHead(ClsHead):

    def __init__(self, num_classes, in_channels, hidden_dim, **kwargs):
        super().__init__(**kwargs)
        self.in_channels = in_channels
        self.num_classes = num_classes
        self.hidden_dim = hidden_dim

        self.fc1 = nn.Linear(in_channels, hidden_dim)
        self.act = nn.Tanh()
        self.fc2 = nn.Linear(hidden_dim, num_classes)

    def pre_logits(self, feats):
        # 骨干网络的输出通常包含多尺度信息的元组
        # 对于分类任务来说，我们只需要关注最后的输出
        feat = feats[-1]

        # VisionTransformer的最终输出是一个包含patch tokens和cls tokens的元组
        # 这里我们只需要cls tokens
        _, cls_token = feat

        # 完成除了最后的线性分类头以外的操作
        return self.act(self.fc1(cls_token))

    def forward(self, feats):
        pre_logits = self.pre_logits(feats)

        # 完成最后的分类头
        cls_score = self.fc(pre_logits)
```

(下页继续)

(续上页)

```
        return cls_score
```

2. 在 mmcls/models/heads/__init__.py 中导入这个模块

```
...
from .vit_head import VisionTransformerClsHead

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

3. 修改配置文件以使用新的头部组件。

```
model = dict(
    head=dict(
        type='VisionTransformerClsHead',
        ...,
    ))
```

18.4 添加新的损失函数

要添加新的损失函数，我们主要需要在损失函数模块中 forward 函数。这里需要注意的是，损失模块也应该注册到 MODELS 中。另外，利用装饰器 weighted_loss 可以方便的实现对每个元素的损失进行加权平均。

假设我们要模拟从另一个分类模型生成的概率分布，需要添加 L1loss 来实现该目的。

1. 创建一个新文件 mmcls/models/losses/l1_loss.py

```
import torch
import torch.nn as nn

from mmcls.registry import MODELS
from .utils import weighted_loss

@weighted_loss
def l1_loss(pred, target):
    assert pred.size() == target.size() and target.numel() > 0
    loss = torch.abs(pred - target)
    return loss

@MODELS.register_module()
class L1Loss(nn.Module):
```

(下页继续)

(续上页)

```
    def __init__(self, reduction='mean', loss_weight=1.0):
        super(L1Loss, self).__init__()
        self.reduction = reduction
        self.loss_weight = loss_weight

    def forward(self,
                pred,
                target,
                weight=None,
                avg_factor=None,
                reduction_override=None):
        assert reduction_override in (None, 'none', 'mean', 'sum')
        reduction = (
            reduction_override if reduction_override else self.reduction)
        loss = self.loss_weight * l1_loss(
            pred, target, weight, reduction=reduction, avg_factor=avg_factor)
        return loss
```

2. 在文件 `mmcls/models/losses/__init__.py` 中导入这个模块

```
...
from .l1_loss import L1Loss

__all__ = [
    ..., 'L1Loss'
]
```

3. 修改配置文件中的 `loss` 字段以使用新的损失函数

```
model = dict(
    head=dict(
        loss=dict(type='L1Loss', loss_weight=1.0),
    ))
```

最后我们可以在配置文件中结合所有新增的模型组件来使用新的模型。由于 `ResNet_CIFAR` 不是一个基于 ViT 的骨干网络，这里我们不用 `VisionTransformerClsHead` 的配置。

```
model = dict(
    type='ImageClassifier',
    backbone=dict(
        type='ResNet_CIFAR',
        depth=18,
        num_stages=4,
```

(下页继续)

(续上页)

```
        out_indices=(3, ),
        style='pytorch'),
    neck=dict(type='GlobalAveragePooling'),
    head=dict(
        type='LinearClsHead',
        num_classes=10,
        in_channels=512,
        loss=dict(type='L1Loss', loss_weight=1.0),
        topk=(1, 5),
    ))
```

小技巧: 为了方便，相同的模型组件可以直接从已有的 config 文件里继承，更多细节可以参考学习配置文件。

CHAPTER 19

自定义训练优化策略

在我们的算法库中，已经提供了通用数据集（如 ImageNet，CIFAR）的默认训练策略配置。如果想要在这些数据集上继续提升模型性能，或者在不同数据集和方法上进行新的尝试，我们通常需要修改这些默认的策略。

在本教程中，我们将介绍如何在运行自定义训练时，通过修改配置文件进行构造优化器、参数化精细配置、梯度裁剪、梯度累计以及定制动量调整策略等。同时也会通过模板简单介绍如何自定义开发优化器和构造器。

19.1 配置训练优化策略

我们通过 `optim_wrapper` 来配置主要的优化策略，包括优化器的选择，混合精度训练的选择，参数化精细配置，梯度裁剪以及梯度累计。接下来将分别介绍这些内容。

19.1.1 构造 PyTorch 内置优化器

MMClassification 支持 PyTorch 实现的所有优化器，仅需在配置文件中，指定优化器封装需要的 `optimizer` 字段。

如果要使用 `SGD`，则修改如下。这里要注意所有优化相关的配置都需要封装在 `optim_wrapper` 配置里。

```
optim_wrapper = dict(
    type='OptimWrapper',
    optimizer=dict(type='SGD', lr=0.0003, weight_decay=0.0001)
)
```

备注： 配置文件中的‘type’不是构造时的参数，而是 PyTorch 内置优化器的类名。更多优化器选择可以参考PyTorch 支持的优化器列表。

要修改模型的学习率，只需要在优化器的配置中修改 `lr` 即可。要配置其他参数，可直接根据 PyTorch API 文档 进行。

例如，如果想使用 `Adam` 并设置参数为 `torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsgrad=False)`。则需要进行如下修改：

```
optim_wrapper = dict(
    type='OptimWrapper',
    optimizer = dict(
        type='Adam',
        lr=0.001,
        betas=(0.9, 0.999),
        eps=1e-08,
        weight_decay=0,
        amsgrad=False),
)
```

备注： 考虑到对于单精度训练来说，优化器封装的默认类型就是 `OptimWrapper`，我们在这里可以直接省略，因此配置文件可以进一步简化为：

```
optim_wrapper = dict(
    optimizer=dict(
        type='Adam',
        lr=0.001,
        betas=(0.9, 0.999),
        eps=1e-08,
        weight_decay=0,
        amsgrad=False))
```

19.1.2 混合精度训练

如果我们想要使用混合精度训练（Automactic Mixed Precision），我们只需简单地将 `optim_wrapper` 的类型改为 `AmpOptimWrapper`。

```
optim_wrapper = dict(type='AmpOptimWrapper', optimizer=...)
```

另外，为了方便，我们同时在启动训练脚本 `tools/train.py` 中提供了 `--amp` 参数作为开启混合精度训练的开关，更多细节可以参考训练与测试教程。

19.1.3 参数化精细配置

在一些模型中，不同的优化策略需要适应特定的参数，例如不在 BatchNorm 层使用权重衰减，或者在不同层使用不同的学习率等等。我们需要用到 optim_wrapper 中的 paramwise_cfg 参数来进行精细化配置。

- **为不同类型的参数设置超参乘子**

  例如，我们可以在 paramwise_cfg 配置中设置 norm_decay_mult=0. 来改变归一化层权重和偏移的衰减为 0。

```
optim_wrapper = dict(
    optimizer=dict(type='SGD', lr=0.8, weight_decay=1e-4),
    paramwise_cfg=dict(norm_decay_mult=0.))
```

  支持更多类型的参数配置，参考以下列表：

  - lr_mult：所有参数的学习率系数
  - decay_mult：所有参数的衰减系数
  - bias_lr_mult：偏置的学习率系数（不包括正则化层的偏置以及可变形卷积的 offset），默认值为 1
  - bias_decay_mult：偏置的权值衰减系数（不包括正则化层的偏置以及可变形卷积的 offset），默认值为 1
  - norm_decay_mult：正则化层权重和偏置的权值衰减系数，默认值为 1
  - dwconv_decay_mult：Depth-wise 卷积的权值衰减系数，默认值为 1
  - bypass_duplicate：是否跳过重复的参数，默认为 False
  - dcn_offset_lr_mult：可变形卷积（Deformable Convolution）的学习率系数，默认值为 1

- **为特定参数设置超参乘子**

  MMClassification 通过 paramwise_cfg 的 custom_keys 参数来配置特定参数的超参乘子。

  例如，我们可以通过以下配置来设置所有 backbone.layer0 层的学习率和权重衰减为 0，backbone 的其余层和优化器保持一致，另外 head 层的学习率为 0.001.

```
optim_wrapper = dict(
    optimizer=dict(type='SGD', lr=0.01, weight_decay=0.0001),
    paramwise_cfg=dict(
        custom_keys={
            'backbone.layer0': dict(lr_mult=0, decay_mult=0),
            'backbone': dict(lr_mult=1),
            'head': dict(lr_mult=0.1)
        }))
```

19.1.4 梯度裁剪

在训练过程中，损失函数可能接近于一些异常陡峭的区域，从而导致梯度爆炸。而梯度裁剪可以帮助稳定训练过程，更多介绍可以参见该页面。

目前我们支持在 `optim_wrapper` 字段中添加 `clip_grad` 参数来进行梯度裁剪，更详细的参数可参考 PyTorch 文档。

用例如下：

```
optim_wrapper = dict(
    optimizer=dict(type='SGD', lr=0.01, weight_decay=0.0001),
    # norm_type: 使用的范数类型，此处使用范数2。
    clip_grad=dict(max_norm=35, norm_type=2))
```

19.1.5 梯度累计

计算资源缺乏缺乏时，每个训练批次的大小（batch size）只能设置为较小的值，这可能会影响模型的性能。

可以使用梯度累计来规避这一问题。我们支持在 `optim_wrapper` 字段中添加 `accumulative_counts` 参数来进行梯度累计。

用例如下：

```
train_dataloader = dict(batch_size=64)
optim_wrapper = dict(
    optimizer=dict(type='SGD', lr=0.01, weight_decay=0.0001),
    accumulative_counts=4)
```

表示训练时，每 4 个 iter 执行一次反向传播。由于此时单张 GPU 上的批次大小为 64，也就等价于单张 GPU 上一次迭代的批次大小为 256，也即：

```
train_dataloader = dict(batch_size=256)
optim_wrapper = dict(
    optimizer=dict(type='SGD', lr=0.01, weight_decay=0.0001))
```

19.2 配置参数优化策略

在训练过程中，优化参数例如学习率、动量，通常不会是固定不变，而是随着训练进程的变化而调整。PyTorch 支持一些学习率调整的调度器，但是不足以完成复杂的策略。在 MMClassification 中，我们提供 `param_scheduler` 来更好地控制不同优化参数的策略。

19.2.1 配置学习率调整策略

深度学习研究中，广泛应用学习率衰减来提高网络的性能。我们支持大多数 PyTorch 学习率调度器，其中包括 ExponentialLR, LinearLR, StepLR, MultiStepLR 等等。

- **单个学习率策略**

  多数情况下，我们使用单一学习率策略，这里 param_scheduler 会是一个字典。比如在默认的 ResNet 网络训练中，我们使用阶梯式的学习率衰减策略 MultiStepLR，配置文件为：

```
param_scheduler = dict(
    type='MultiStepLR',
    by_epoch=True,
    milestones=[100, 150],
    gamma=0.1)
```

  或者我们想使用 CosineAnnealingLR 来进行学习率衰减：

```
param_scheduler = dict(
    type='CosineAnnealingLR',
    by_epoch=True,
    T_max=num_epochs)
```

- **多个学习率策略**

  然而在一些其他情况下，为了提高模型的精度，通常会使用多种学习率策略。例如，在训练的早期阶段，网络容易不稳定，而学习率的预热就是为了减少这种不稳定性。

  整个学习过程中，学习率将会通过预热从一个很小的值逐步提高到预定值，再会通过其他的策略进一步调整。

  在 MMClassification 中，我们同样使用 param_scheduler ，将多种学习策略写成列表就可以完成上述预热策略的组合。

  例如：

  1. 在前 50 次迭代中逐**迭代次数**地**线性**预热

```
param_scheduler = [
    # 逐迭代次数，线性预热
    dict(type='LinearLR',
         start_factor=0.001,
         by_epoch=False,  # 逐迭代次数
         end=50),  # 只预热50次迭代次数
    # 主要的学习率策略
    dict(type='MultiStepLR',
         by_epoch=True,
         milestones=[8, 11],
```

(下页继续)

(续上页)

```
        gamma=0.1)
]
```

2. 在前 10 轮迭代中逐**迭代次数**地**线性**预热

```
param_scheduler = [
    # 在前10轮迭代中，逐迭代次数，线性预热
    dict(type='LinearLR',
        start_factor=0.001,
        by_epoch=True,
        end=10,
        convert_to_iter_based=True,  # 逐迭代次数更新学习率.
    ),
    # 在 10 轮次后，通过余弦退火衰减
    dict(type='CosineAnnealingLR', by_epoch=True, begin=10)
]
```

注意这里增加了 `begin` 和 `end` 参数，这两个参数指定了调度器的**生效区间**。生效区间通常只在多个调度器组合时才需要去设置，使用单个调度器时可以忽略。当指定了 `begin` 和 `end` 参数时，表示该调度器只在 [begin, end) 区间内生效，其单位是由 `by_epoch` 参数决定。在组合不同调度器时，各调度器的 `by_epoch` 参数不必相同。如果没有指定的情况下，`begin` 为 0，`end` 为最大迭代轮次或者最大迭代次数。

如果相邻两个调度器的生效区间没有紧邻，而是有一段区间没有被覆盖，那么这段区间的学习率维持不变。而如果两个调度器的生效区间发生了重叠，则对多组调度器叠加使用，学习率的调整会按照调度器配置文件中的顺序触发（行为与 PyTorch 中 `ChainedScheduler` 一致）。

小技巧： 为了避免学习率曲线与预期不符，配置完成后，可以使用 MMClassification 提供的学习率可视化工具 画出对应学习率调整曲线。

19.2.2 配置动量调整策略

MMClassification 支持动量调度器根据学习率修改优化器的动量，从而使损失函数收敛更快。用法和学习率调度器一致。

我们支持的动量策略和详细的使用细节可以参考这里。我们只将调度器中的 `LR` 替换为了 `Momentum`，动量策略可以直接追加 `param_scheduler` 列表中。

这里是一个用例：

```
param_scheduler = [
    # 学习率策略
```

(下页继续)

(续上页)

```
    dict(type='LinearLR', ...),
    # 动量策略
    dict(type='LinearMomentum',
         start_factor=0.001,
         by_epoch=False,
         begin=0,
         end=1000)
]
```

19.3 新增优化器或者优化器构造器

备注： 本部分将修改 MMClassification 源码或者向 MMClassification 框架添加代码，初学者可跳过。

19.3.1 新增优化器

在学术研究和工业实践中，可能需要使用 MMClassification 未实现的优化方法，可以通过以下方法添加。

1. 定义一个新的优化器

一个自定义的优化器可根据如下规则进行定制：

假设我们想添加一个名为 `MyOptimzer` 的优化器，其拥有参数 `a`，`b` 和 `c`。可以创建一个名为 `mmcls/engine/optimizer` 的文件夹，并在目录下的一个文件，如 `mmcls/engine/optimizer/my_optimizer.py` 中实现该自定义优化器：

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


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

    def __init__(self, a, b, c):
        ...

    def step(self, closure=None):
        ...
```

2. 注册优化器

要注册上面定义的上述模块，首先需要将此模块导入到主命名空间中。有两种方法可以实现它。

- 修改 mmcls/engine/optimizers/__init__.py，将其导入至 mmcls.engine 包。

```
# 在 mmcls/engine/optimizers/__init__.py 中
...
from .my_optimizer import MyOptimizer # MyOptimizer 是我们自定义的优化器的名字

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

在运行过程中，我们会自动导入 mmcls.engine 包并同时注册 MyOptimizer。

- 在配置中使用 custom_imports 手动导入

```
custom_imports = dict(
    imports=['mmcls.engine.optimizers.my_optimizer'],
    allow_failed_imports=False,
)
```

mmcls.engine.optimizers.my_optimizer 模块将会在程序开始阶段被导入，MyOptimizer 类会随之自动被注册。注意，这里只需要导入包含 MyOptmizer 类的包。如果填写 mmcls.engine.optimizers.my_optimizer.MyOptimizer 则 **不会**被直接导入。

3. 在配置文件中指定优化器

之后，用户便可在配置文件的 optim_wrapper.optimizer 域中使用 MyOptimizer：

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

19.3.2 新增优化器构造器

某些模型可能具有一些特定于参数的设置以进行优化，例如为所有 BatchNorm 层设置不同的权重衰减。

尽管我们已经可以使用*optim_wrapper.paramwise_cfg 字段*来配置特定参数的优化设置，但可能仍然无法覆盖你的需求。

当然你可以在此基础上进行修改。我们默认使用 DefaultOptimWrapperConstructor 来构造优化器。在构造过程中，通过 paramwise_cfg 来精细化配置不同设置。这个默认构造器可以作为新优化器构造器实现的模板。

我们可以新增一个优化器构造器来覆盖这些行为。

```python
# 在 mmcls/engine/optimizers/my_optim_constructor.py 中
from mmengine.optim import DefaultOptimWrapperConstructor
from mmcls.registry import OPTIM_WRAPPER_CONSTRUCTORS


@OPTIM_WRAPPER_CONSTRUCTORS.register_module()
class MyOptimWrapperConstructor:

    def __init__(self, optim_wrapper_cfg, paramwise_cfg=None):
        ...

    def __call__(self, model):
        ...
```

接下来类似新增优化器教程 来导入并使用新的优化器构造器。

1. 修改 mmcls/engine/optimizers/__init__.py，将其导入至 mmcls.engine 包。

   ```python
   # 在 mmcls/engine/optimizers/__init__.py 中
   ...
   from .my_optim_constructor import MyOptimWrapperConstructor

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

2. 在配置文件的 optim_wrapper.constructor 字段中使用 MyOptimWrapperConstructor 。

   ```python
   optim_wrapper = dict(
       constructor=dict(type='MyOptimWrapperConstructor'),
       optimizer=...,
       paramwise_cfg=...,
   )
   ```

CHAPTER 20

自定义运行参数

运行参数配置包括许多有用的功能，如权重文件保存、日志配置等等，在本教程中，我们将介绍如何配置这些功能。

20.1 保存权重文件

权重文件保存功能是一个在训练阶段默认注册的钩子，你可以通过配置文件中的 `default_hooks.checkpoint` 字段配置它。

备注： 钩子机制在 OpenMMLab 开源算法库中应用非常广泛。通过钩子，你可以在不修改运行器的主要执行逻辑的情况下插入许多功能。

可以通过相关文章进一步理解钩子。

默认配置：

```
default_hooks = dict(
    ...
    checkpoint = dict(type='CheckpointHook', interval=1)
    ...
)
```

下面是一些权重文件钩子 (CheckpointHook)的常用可配置参数。

- **interval** (int): 文件保存周期。如果使用-1，它将永远不会保存权重。
- **by_epoch** (bool): 选择 **interval** 是基于 epoch 还是基于 iteration，默认为 `True`.
- **out_dir** (str): 保存权重文件的根目录。如果不指定，检查点将被保存在工作目录中。如果指定，检查点将被保存在 **out_dir** 的子文件夹中。
- **max_keep_ckpts** (int): 要保留的权重文件数量。在某些情况下，为了节省磁盘空间，我们希望只保留最近的几个权重文件。默认为-1，也就是无限制。
- **save_best** (str, List[str]): 如果指定，它将保存具有最佳评估结果的权重。通常情况下，你可以直接使用 `save_best="auto"` 来自动选择评估指标。而如果你想要更高级的配置，请参考权重文件钩子(CheckpointHook)。

20.2 权重加载 / 断点训练

在配置文件中，你可以加载指定模型权重或者断点继续训练，如下所示:

```
# 从指定权重文件加载
load_from = "Your checkpoint path"

# 是否从加载的断点继续训练
resume = False
```

`load_from` 字段可以是本地路径，也可以是 HTTP 路径。你可以从检查点恢复训练，方法是指定 `resume=True`。

小技巧: 你也可以通过指定 `load_from=None` 和 `resume=True` 启用从最新的断点自动恢复。Runner 执行器将自动从工作目录中找到最新的权重文件。

如果你用我们的 `tools/train.py` 脚本来训练模型，你只需使用 `--resume` 参数来恢复训练，就不用手动修改配置文件了。如下所示:

```
# 自动从最新的断点恢复
python tools/train.py configs/resnet/resnet50_8xb32_in1k.py --resume

# 从指定的断点恢复
python tools/train.py configs/resnet/resnet50_8xb32_in1k.py --resume checkpoints/
↪resnet.pth
```

20.3 随机性 (Randomness) 配置

为了让实验尽可能是可复现的，我们在 randomness 字段中提供了一些控制随机性的选项。

默认情况下，我们不会在配置文件中指定随机数种子，在每次实验中，程序会生成一个不同的随机数种子。

默认配置:

```
randomness = dict(seed=None, deterministic=False)
```

为了使实验更具可复现性，你可以指定一个种子并设置 deterministic=True。deterministic 选项的使用效果可以在这里找到。

20.4 日志配置

日志的配置与多个字段有关。

在 log_level 字段中，你可以指定全局日志级别。参见 Logging Levels 以获得日志级别列表。

```
log_level = 'INFO'
```

在 default_hooks.logger 字段中，你可以指定训练和测试期间的日志间隔。而所有可用的参数可以在日志钩子文档中找到。

```
default_hooks = dict(
    ...
    # 每100次迭代就打印一次日志
    logger=dict(type='LoggerHook', interval=100),
    ...
)
```

在 log_processor 字段中，你可以指定日志信息的平滑方法。通常，我们使用一个长度为 10 的窗口来平滑日志中的值，并输出所有信息的平均值。如果你想特别指定某些信息的平滑方法，请参阅日志处理器文档

```
# 默认设置，它将通过一个10长度的窗口平滑训练日志中的值
log_processor = dict(window_size=10)
```

在 visualizer 字段中，你可以指定多个后端来保存日志信息，如 TensorBoard 和 WandB。更多的细节可以在可视化工具找到。

20.5 自定义钩子

上述许多功能是由钩子实现的，你也可以通过修改 `custom_hooks` 字段来插入其他的自定义钩子。下面是 MMEngine 和 MMClassification 中的一些钩子，你可以直接使用，例如：

- *EMAHook*
- SyncBuffersHook
- EmptyCacheHook
- *ClassNumCheckHook*
- ……

例如，EMA（Exponential Moving Average）在模型训练中被广泛使用，你可以以下方式启用它：

```
custom_hooks = [
    dict(type='EMAHook', momentum=4e-5, priority='ABOVE_NORMAL'),
]
```

20.6 验证可视化

验证可视化钩子是一个验证过程中默认注册的钩子。你可以在 `default_hooks.visualization` 字段中来配置它。

默认情况下，我们禁用这个钩子，你可以通过指定 `enable=True` 来启用它。而更多的参数可以在可视化钩子文档中找到。

```
default_hooks = dict(
    ...
    visualization=dict(type='VisualizationHook', enable=False),
    ...
)
```

这个钩子将在验证数据集中选择一部分图像，在每次验证过程中记录并可视化它们的预测结果。你可以用它来观察训练期间模型在实际图像上的性能变化。

此外，如果你的验证数据集中的图像很小（<100，如 Cifra 数据集），你可以指定 rescale_factor 来缩放它们，如 `rescale_factor=2.`，将可视化的图像放大两倍。

20.7 Visualizer

Visualizer 用于记录训练和测试过程中的各种信息，包括日志、图像和标量。默认情况下，记录的信息将被保存在工作目录下的 `vis_data` 文件夹中。

默认配置:

```
visualizer = dict(
    type='ClsVisualizer',
    vis_backends=[
        dict(type='LocalVisBackend'),
    ]
)
```

通常，最有用的功能是将日志和标量如 `loss` 保存到不同的后端。例如，要把它们保存到 TensorBoard，只需像下面这样设置：

```
visualizer = dict(
    type='ClsVisualizer',
    vis_backends=[
        dict(type='LocalVisBackend'),
        dict(type='TensorboardVisBackend'),
    ]
)
```

或者像下面这样把它们保存到 WandB：

```
visualizer = dict(
    type='ClsVisualizer',
    vis_backends=[
        dict(type='LocalVisBackend'),
        dict(type='WandbVisBackend'),
    ]
)
```

20.8 环境配置

在 `env_cfg` 字段中，你可以配置一些底层的参数，如 cuDNN、多进程和分布式通信。

在修改这些参数之前，请确保你理解这些参数的含义。

```
env_cfg = dict(
    # 是否启用cudnn基准测试
```

(下页继续)

(续上页)

```
    cudnn_benchmark=False,

    # 设置多进程参数
    mp_cfg=dict(mp_start_method='fork', opencv_num_threads=0),

    # 设置分布式参数
    dist_cfg=dict(backend='nccl'),
)
```

20.9 FAQ

1. **load_from 和 init_cfg 之间的关系是什么？**

   - load_from: 如果 resume=False，只导入模型权重，主要用于加载训练过的模型；如果 resume=True ，加载所有的模型权重、优化器状态和其他训练信息，主要用于恢复中断的训练。
   - init_cfg: 你也可以指定 init=dict(type="Pretrained", checkpoint=xxx) 来加载权重，表示在模型权重初始化时加载权重，通常在训练的开始阶段执行。主要用于微调预训练模型，你可以在骨干网络的配置中配置它，还可以使用 prefix 字段来只加载对应的权重，例如:

     ```
     model = dict(
        backbone=dict(
            type='ResNet',
            depth=50,
            init_cfg=dict(type='Pretrained', checkpoints=xxx, prefix='backbone'),
        )
        ...
     )
     ```

     参见微调模型 以了解更多关于模型微调的细节。

2. **default_hooks 和 custom_hooks 之间有什么区别？**

   几乎没有区别。通常，default_hooks 字段用于指定几乎所有实验都会使用的钩子，而 custom_hooks 字段指部分实验特有的钩子。

   另一个区别是 default_hooks 是一个字典，而 custom_hooks 是一个列表，请不要混淆。

3. **在训练期间，我没有收到训练日志，这是什么原因？**

   如果你的训练数据集很小，而批处理量却很大，我们默认的日志间隔可能太大，无法记录你的训练日志。

   你可以缩减日志间隔，再试一次，比如:

```
default_hooks = dict(
    ...
    logger=dict(type='LoggerHook', interval=10),
    ...
)
```

CHAPTER 21

自定义评估指标（待更新）

请参见英文文档，如果你有兴趣参与中文文档的翻译，欢迎在 讨论区进行报名。

CHAPTER 22

数据流（待更新）

CHAPTER 23

MMCLS 中的约定

23.1 配置文件命名规则

MMClassification 按照以下风格进行配置文件命名，代码库的贡献者需要遵循相同的命名规则。文件名总体分为四部分：算法信息，模块信息，训练信息和数据信息。逻辑上属于不同部分的单词之间用下划线 `'_'` 连接，同一部分有多个单词用短横线 `'-'` 连接。

```
{algorithm info}_{module info}_{training info}_{data info}.py
```

- `algorithm info`：算法信息，算法名称或者网络架构，如 resnet 等；
- `module info`：模块信息，因任务而异，用以表示一些特殊的 neck、head 和 pretrain 信息；
- `training info`：一些训练信息，训练策略设置，包括 batch size，schedule 以及数据增强等；
- `data info`：数据信息，数据集名称、模态、输入尺寸等，如 imagenet, cifar 等；

23.1.1 算法信息

指论文中的算法名称缩写，以及相应的分支架构信息。例如：

- `resnet50`
- `mobilenet-v3-large`
- `vit-small-patch32`：`patch32` 表示 `ViT` 切分的分块大小

- `seresnext101-32x4d`：SeResNet101 基本网络结构，32x4d 表示在 Bottleneck 中 groups 和 width_per_group 分别为 32 和 4

23.1.2 模块信息

指一些特殊的 neck 、head 或者 pretrain 的信息，在分类中常见为预训练信息，比如：

- `in21k-pre`：在 ImageNet21k 上预训练
- `in21k-pre-3rd-party`：在 ImageNet21k 上预训练，其权重来自其他仓库

23.1.3 训练信息

训练策略的一些设置，包括训练类型、batch size、lr schedule、数据增强以及特殊的损失函数等等，比如: Batch size 信息：

- 格式为 `{gpu x batch_per_gpu}`, 如 8xb32

训练类型 (主要见于 transformer 网络，如 ViT 算法，这类算法通常分为预训练和微调两种模式):

- `ft`：Finetune config，用于微调的配置文件
- `pt`：Pretrain config，用于预训练的配置文件

训练策略信息，训练策略以复现配置文件为基础，此基础不必标注训练策略。但如果在此基础上进行改进，则需注明训练策略，按照应用点位顺序排列，如：`{pipeline aug}-{train aug}-{loss trick}-{scheduler}-{epochs}`

- `coslr-200e`：使用 cosine scheduler, 训练 200 个 epoch
- `autoaug-mixup-lbs-coslr-50e`：使用了 autoaug、mixup、label smooth、cosine scheduler, 训练了 50 个轮次

23.1.4 数据信息

- `in1k`：ImageNet1k 数据集，默认使用 224x224 大小的图片
- `in21k`：ImageNet21k 数据集，有些地方也称为 ImageNet22k 数据集，默认使用 224x224 大小的图片
- `in1k-384px`：表示训练的输出图片大小为 384x384
- `cifar100`

23.1.5 配置文件命名案例

```
repvgg-D2se_deploy_4xb64-autoaug-lbs-mixup-coslr-200e_in1k.py
```

- repvgg-D2se: 算法信息
  - repvgg: 主要算法名称。
  - D2se: 模型的结构。
- deploy: 模块信息，该模型为推理状态。
- 4xb64-autoaug-lbs-mixup-coslr-200e: 训练信息
  - 4xb64: 使用 4 块 GPU 并且每块 GPU 的批大小为 64。
  - autoaug: 使用 AutoAugment 数据增强方法。
  - lbs: 使用 label smoothing 损失函数。
  - mixup: 使用 mixup 训练增强方法。
  - coslr: 使用 cosine scheduler 优化策略。
  - 200e: 训练 200 轮次。
- in1k: 数据信息。配置文件用于 ImageNet1k 数据集上使用 224x224 大小图片训练。

23.1.6 权重命名规则

权重的命名主要包括配置文件名，日期和哈希值。

```
{config_name}_{date}-{hash}.pth
```

CHAPTER 24

模型库统计

在本页面中，我们列举了我们支持的所有算法。你可以点击链接跳转至对应的模型详情页面。

另外，我们还列出了我们提供的所有模型权重文件。你可以使用排序和搜索功能找到需要的模型权重，并使用链接跳转至模型详情页面。

24.1 所有已支持的算法

- 论文数量：54
  - Algorithm: 54
- 模型权重文件数量：416
  - [Algorithm] *MobileNetV2: Inverted Residuals and Linear Bottlenecks* (1 ckpts)
  - [Algorithm] *Searching for MobileNetV3* (6 ckpts)
  - [Algorithm] *Deep Residual Learning for Image Recognition* (25 ckpts)
  - [Algorithm] *Res2Net: A New Multi-scale Backbone Architecture* (3 ckpts)
  - [Algorithm] *Aggregated Residual Transformations for Deep Neural Networks* (4 ckpts)
  - [Algorithm] *Squeeze-and-Excitation Networks* (2 ckpts)
  - [Algorithm] *ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices* (1 ckpts)
  - [Algorithm] *ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design* (1 ckpts)

- [Algorithm] *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows* (14 ckpts)
- [Algorithm] *Very Deep Convolutional Networks for Large-Scale Image* (8 ckpts)
- [Algorithm] *RepVGG: Making VGG-style ConvNets Great Again* (12 ckpts)
- [Algorithm] *Transformer in Transformer* (1 ckpts)
- [Algorithm] *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale* (8 ckpts)
- [Algorithm] *Tokens-to-Token ViT: Training Vision Transformers from Scratch on ImageNet* (3 ckpts)
- [Algorithm] *TinyViT: Fast Pretraining Distillation for Small Vision Transformers* (8 ckpts)
- [Algorithm] *MLP-Mixer: An all-MLP Architecture for Vision* (2 ckpts)
- [Algorithm] *Conformer: Local Features Coupling Global Representations for Visual Recognition* (4 ckpts)
- [Algorithm] *Designing Network Design Spaces* (16 ckpts)
- [Algorithm] *Training data-efficient image transformers & distillation through attention* (9 ckpts)
- [Algorithm] *Twins: Revisiting the Design of Spatial Attention in Vision Transformers* (6 ckpts)
- [Algorithm] *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks* (32 ckpts)
- [Algorithm] *A ConvNet for the 2020s* (23 ckpts)
- [Algorithm] *Deep High-Resolution Representation Learning for Visual Recognition* (9 ckpts)
- [Algorithm] *RepMLP: Re-parameterizing Convolutions into Fully-connected Layers for Image Recognition* (2 ckpts)
- [Algorithm] *Wide Residual Networks* (3 ckpts)
- [Algorithm] *Visual Attention Network* (4 ckpts)
- [Algorithm] *CSPNet: A New Backbone that can Enhance Learning Capability of CNN* (3 ckpts)
- [Algorithm] *Patches Are All You Need?* (3 ckpts)
- [Algorithm] *Densely Connected Convolutional Networks* (4 ckpts)
- [Algorithm] *MetaFormer is Actually What You Need for Vision* (5 ckpts)
- [Algorithm] *Rethinking the Inception Architecture for Computer Vision* (1 ckpts)
- [Algorithm] *MViTv2: Improved Multiscale Vision Transformers for Classification and Detection* (4 ckpts)
- [Algorithm] *EdgeNeXt: Efficiently Amalgamated CNN-Transformer Architecture for Mobile Vision Applications* (6 ckpts)
- [Algorithm] *An Improved One millisecond Mobile Backbone* (5 ckpts)
- [Algorithm] *EfficientFormer: Vision Transformers at MobileNet Speed* (3 ckpts)
- [Algorithm] *Swin Transformer V2: Scaling Up Capacity and Resolution* (12 ckpts)
- [Algorithm] *DeiT III: Revenge of the ViT* (16 ckpts)

- [Algorithm] *HorNet: Efficient High-Order Spatial Interactions with Recursive Gated Convolutions* (9 ckpts)
- [Algorithm] *MobileViT Light-weight, General-purpose, and Mobile-friendly Vision Transformer* (3 ckpts)
- [Algorithm] *DaViT: Dual Attention Vision Transformers* (3 ckpts)
- [Algorithm] *Scaling Up Your Kernels to 31x31: Revisiting Large Kernel Design in CNNs* (6 ckpts)
- [Algorithm] *Residual Attention: A Simple but Effective Method for Multi-Label Recognition* (1 ckpts)
- [Algorithm] *BEiT: BERT Pre-Training of Image Transformers* (1 ckpts)
- [Algorithm] *BEiT v2: Masked Image Modeling with Vector-Quantized Visual Tokenizers* (1 ckpts)
- [Algorithm] *EVA: Exploring the Limits of Masked Visual Representation Learning at Scale* (10 ckpts)
- [Algorithm] *Reversible Vision Transformers* (2 ckpts)
- [Algorithm] *Learning Transferable Visual Models From Natural Language Supervision* (14 ckpts)
- [Algorithm] *MixMIM: Mixed and Masked Image Modeling for Efficient Visual Representation Learning* (1 ckpts)
- [Algorithm] *EfficientNetV2: Smaller Models and Faster Training* (15 ckpts)
- [Algorithm] *Co-designing and Scaling ConvNets with Masked Autoencoders* (26 ckpts)
- [Algorithm] *LeViT: a Vision Transformer in ConvNet' s Clothing for Faster Inference* (5 ckpts)
- [Algorithm] *Vision GNN: An Image is Worth Graph of Nodes* (7 ckpts)
- [Algorithm] *ArcFace: Additive Angular Margin Loss for Deep Face Recognition* (1 ckpts)
- [Algorithm] *XCiT: Cross-Covariance Image Transformers* (42 ckpts)

24.2 所有模型权重文件

24.2.1 ImageNet-1k

模型	参数量 (M)	Flops (G)	Top-1 (%)	Top-5 (%)	Readme
mobilenet-v2_8xb32_in1k	3.50	0.32	71.86	90.42	链接
mobilenet-v3-small-050_3rdparty_in1k	1.59	0.02	57.91	80.19	链接
mobilenet-v3-small-075_3rdparty_in1k	2.04	0.04	65.23	85.44	链接
mobilenet-v3-small_8xb128_in1k	2.54	0.06	66.68	86.74	链接
mobilenet-v3-small_3rdparty_in1k	2.54	0.06	67.66	87.41	链接
mobilenet-v3-large_8xb128_in1k	5.48	0.23	73.49	91.31	链接
mobilenet-v3-large_3rdparty_in1k	5.48	0.23	74.04	91.34	链接
resnet18_8xb32_in1k	11.69	1.82	69.90	89.43	链接

下页继续

表 1 – 续上页

模型	参数量 (M)	Flops (G)	Top-1 (%)	Top-5 (%)	Readme
resnet34_8xb32_in1k	2.18	3.68	73.62	91.59	链接
resnet50_8xb32_in1k	25.56	4.12	76.55	93.06	链接
resnet101_8xb32_in1k	44.55	7.85	77.97	94.06	链接
resnet152_8xb32_in1k	60.19	11.58	78.48	94.13	链接
resnetv1d50_8xb32_in1k	25.58	4.36	77.54	93.57	链接
resnetv1d101_8xb32_in1k	44.57	8.09	78.93	94.48	链接
resnetv1d152_8xb32_in1k	60.21	11.82	79.41	94.70	链接
resnet50_8xb32-fp16_in1k	25.56	4.12	76.30	93.07	链接
resnet50_8xb256-rsb-a1-600e_in1k	25.56	4.12	80.12	94.78	链接
resnet50_8xb256-rsb-a2-300e_in1k	25.56	4.12	79.55	94.37	链接
resnet50_8xb256-rsb-a3-100e_in1k	25.56	4.12	78.30	93.80	链接
resnetv1c50_8xb32_in1k	25.58	4.36	77.01	93.58	链接
resnetv1c101_8xb32_in1k	44.57	8.09	78.30	94.27	链接
resnetv1c152_8xb32_in1k	60.21	11.82	78.76	94.41	链接
res2net50-w14-s8_3rdparty_8xb32_in1k	25.06	4.22	78.14	93.85	链接
res2net50-w26-s8_3rdparty_8xb32_in1k	48.40	8.39	79.20	94.36	链接
res2net101-w26-s4_3rdparty_8xb32_in1k	45.21	8.12	79.19	94.44	链接
resnext50-32x4d_8xb32_in1k	25.03	4.27	77.90	93.66	链接
resnext101-32x4d_8xb32_in1k	44.18	8.03	78.61	94.17	链接
resnext101-32x8d_8xb32_in1k	88.79	16.50	79.27	94.58	链接
resnext152-32x4d_8xb32_in1k	59.95	11.80	78.88	94.33	链接
seresnet50_8xb32_in1k	28.09	4.13	77.74	93.84	链接
seresnet101_8xb32_in1k	49.33	7.86	78.26	94.07	链接
shufflenet-v1-1x_16xb64_in1k	1.87	0.15	68.13	87.81	链接
shufflenet-v2-1x_16xb64_in1k	2.28	0.15	69.55	88.92	链接
swin-tiny_16xb64_in1k	28.29	4.36	81.18	95.61	链接
swin-small_16xb64_in1k	49.61	8.52	83.02	96.29	链接
swin-base_16xb64_in1k	87.77	15.14	83.36	96.44	链接
swin-tiny_3rdparty_in1k	28.29	4.36	81.18	95.52	链接
swin-small_3rdparty_in1k	49.61	8.52	83.21	96.25	链接
swin-base_3rdparty_in1k	87.77	15.14	83.42	96.44	链接
swin-base_3rdparty_in1k-384	87.90	44.49	84.49	96.95	链接
swin-base_in21k-pre-3rdparty_in1k	87.77	15.14	85.16	97.50	链接
swin-base_in21k-pre-3rdparty_in1k-384	87.90	44.49	86.44	98.05	链接
swin-large_in21k-pre-3rdparty_in1k	196.53	34.04	86.24	97.88	链接
swin-large_in21k-pre-3rdparty_in1k-384	196.74	100.04	87.25	98.25	链接
vgg11_8xb32_in1k	132.86	7.63	68.75	88.87	链接

下页继续

表 1 – 续上页

模型	参数量 (M)	Flops (G)	Top-1 (%)	Top-5 (%)	Readme
vgg13_8xb32_in1k	133.05	11.34	70.02	89.46	链接
vgg16_8xb32_in1k	138.36	15.50	71.62	90.49	链接
vgg19_8xb32_in1k	143.67	19.67	72.41	90.80	链接
vgg11bn_8xb32_in1k	132.87	7.64	70.67	90.16	链接
vgg13bn_8xb32_in1k	133.05	11.36	72.12	90.66	链接
vgg16bn_8xb32_in1k	138.37	15.53	73.74	91.66	链接
vgg19bn_8xb32_in1k	143.68	19.70	74.68	92.27	链接
repvgg-A0_8xb32_in1k	8.31	1.36	72.37	90.56	链接
repvgg-A1_8xb32_in1k	12.79	2.36	74.23	91.80	链接
repvgg-A2_8xb32_in1k	25.50	5.12	76.49	93.09	链接
repvgg-B0_8xb32_in1k	3.42	15.82	75.27	92.21	链接
repvgg-B1_8xb32_in1k	51.83	11.81	78.19	94.04	链接
repvgg-B1g2_8xb32_in1k	41.36	8.81	77.87	93.99	链接
repvgg-B1g4_8xb32_in1k	36.13	7.30	77.81	93.77	链接
repvgg-B2_8xb32_in1k	80.32	18.37	78.58	94.23	链接
repvgg-B2g4_8xb32_in1k	55.78	11.33	79.44	94.72	链接
repvgg-B3_8xb32_in1k	110.96	26.21	80.58	95.33	链接
repvgg-B3g4_8xb32_in1k	75.63	16.06	80.26	95.15	链接
repvgg-D2se_3rdparty_in1k	120.39	32.84	81.81	95.94	链接
tnt-small-p16_3rdparty_in1k	23.76	3.36	81.52	95.73	链接
vit-base-p16_in21k-pre-3rdparty_ft-64xb64_in1k-384	86.86	33.03	85.43	97.77	链接
vit-base-p32_in21k-pre-3rdparty_ft-64xb64_in1k-384	88.30	8.56	84.01	97.08	链接
vit-large-p16_in21k-pre-3rdparty_ft-64xb64_in1k-384	304.72	116.68	85.63	97.63	链接
vit-base-p16_pt-32xb128-mae_in1k	86.86	33.03	82.37	96.15	链接
t2t-vit-t-14_8xb64_in1k	21.47	4.34	81.83	95.84	链接
t2t-vit-t-19_8xb64_in1k	39.08	7.80	82.63	96.18	链接
t2t-vit-t-24_8xb64_in1k	64.00	12.69	82.71	96.09	链接
tinyvit-5m_3rdparty_in1k	5.39	1.29	79.02	94.74	链接
tinyvit-5m_in21k-distill-pre_3rdparty_in1k	5.39	1.29	80.71	95.57	链接
tinyvit-11m_3rdparty_in1k	11.00	2.05	81.44	95.79	链接
tinyvit-11m_in21k-distill-pre_3rdparty_in1k	11.00	2.05	83.19	96.53	链接
tinyvit-21m_3rdparty_in1k	21.20	4.30	83.08	96.58	链接
tinyvit-21m_in21k-distill-pre_3rdparty_in1k	21.20	4.30	84.85	97.27	链接
tinyvit-21m_in21k-distill-pre_3rdparty_in1k-384px	21.23	13.85	86.21	97.77	链接
tinyvit-21m_in21k-distill-pre_3rdparty_in1k-512px	21.27	27.15	86.44	97.89	链接
mlp-mixer-base-p16_3rdparty_64xb64_in1k	59.88	12.61	76.68	92.25	链接
mlp-mixer-large-p16_3rdparty_64xb64_in1k	208.20	44.57	72.34	88.02	链接

下页继续

表 1 – 续上页

模型	参数量 (M)	Flops (G)	Top-1 (%)	Top-5 (%)	Readme
conformer-tiny-p16_3rdparty_8xb128_in1k	23.52	4.90	81.31	95.60	链接
conformer-small-p16_3rdparty_8xb128_in1k	37.67	10.31	83.32	96.46	链接
conformer-small-p32_8xb128_in1k	38.85	7.09	81.96	96.02	链接
conformer-base-p16_3rdparty_8xb128_in1k	83.29	22.89	83.82	96.59	链接
regnetx-400mf_8xb128_in1k	5.16	0.41	72.56	90.78	链接
regnetx-800mf_8xb128_in1k	7.26	0.81	74.76	92.32	链接
regnetx-1.6gf_8xb128_in1k	9.19	1.63	76.84	93.31	链接
regnetx-3.2gf_8xb64_in1k	3.21	1.53	78.09	94.08	链接
regnetx-4.0gf_8xb64_in1k	22.12	4.00	78.60	94.17	链接
regnetx-6.4gf_8xb64_in1k	26.21	6.51	79.38	94.65	链接
regnetx-8.0gf_8xb64_in1k	39.57	8.03	79.12	94.51	链接
regnetx-12gf_8xb64_in1k	46.11	12.15	79.67	95.03	链接
deit-tiny_pt-4xb256_in1k	5.72	1.08	74.50	92.24	链接
deit-tiny-distilled_3rdparty_pt-4xb256_in1k	5.72	1.08	74.51	91.90	链接
deit-small_pt-4xb256_in1k	22.05	4.24	80.69	95.06	链接
deit-small-distilled_3rdparty_pt-4xb256_in1k	22.05	4.24	81.17	95.40	链接
deit-base_pt-16xb64_in1k	86.57	16.86	81.76	95.81	链接
deit-base_3rdparty_pt-16xb64_in1k	86.57	16.86	81.79	95.59	链接
deit-base-distilled_3rdparty_pt-16xb64_in1k	86.57	16.86	83.33	96.49	链接
deit-base_3rdparty_ft-16xb32_in1k-384px	86.86	49.37	83.04	96.31	链接
deit-base-distilled_3rdparty_ft-16xb32_in1k-384px	86.86	49.37	85.55	97.35	链接
twins-pcpvt-small_3rdparty_8xb128_in1k	24.11	3.67	81.14	95.69	链接
twins-pcpvt-base_3rdparty_8xb128_in1k	43.83	6.45	82.66	96.26	链接
twins-pcpvt-large_3rdparty_16xb64_in1k	60.99	9.51	83.09	96.59	链接
twins-svt-small_3rdparty_8xb128_in1k	24.06	2.82	81.77	95.57	链接
twins-svt-base_8xb128_3rdparty_in1k	56.07	8.35	83.13	96.29	链接
twins-svt-large_3rdparty_16xb64_in1k	99.27	14.82	83.60	96.50	链接
efficientnet-b0_3rdparty_8xb32_in1k	5.29	0.42	76.74	93.17	链接
efficientnet-b0_3rdparty_8xb32-aa_in1k	5.29	0.42	77.26	93.41	链接
efficientnet-b0_3rdparty_8xb32-aa-advprop_in1k	5.29	0.42	77.53	93.61	链接
efficientnet-b0_3rdparty-ra-noisystudent_in1k	5.29	0.42	77.63	94.00	链接
efficientnet-b1_3rdparty_8xb32_in1k	7.79	0.74	78.68	94.28	链接
efficientnet-b1_3rdparty_8xb32-aa_in1k	7.79	0.74	79.20	94.42	链接
efficientnet-b1_3rdparty_8xb32-aa-advprop_in1k	7.79	0.74	79.52	94.43	链接
efficientnet-b1_3rdparty-ra-noisystudent_in1k	7.79	0.74	81.44	95.83	链接
efficientnet-b2_3rdparty_8xb32_in1k	9.11	1.07	79.64	94.80	链接
efficientnet-b2_3rdparty_8xb32-aa_in1k	9.11	1.07	80.21	94.96	链接

下页继续

表 1 – 续上页

模型	参数量 (M)	Flops (G)	Top-1 (%)	Top-5 (%)	Readme
efficientnet-b2_3rdparty_8xb32-aa-advprop_in1k	9.11	1.07	80.45	95.07	链接
efficientnet-b2_3rdparty-ra-noisystudent_in1k	9.11	1.07	82.47	96.23	链接
efficientnet-b3_3rdparty_8xb32_in1k	12.23	1.95	81.01	95.34	链接
efficientnet-b3_3rdparty_8xb32-aa_in1k	12.23	1.95	81.58	95.67	链接
efficientnet-b3_3rdparty_8xb32-aa-advprop_in1k	12.23	1.95	81.81	95.69	链接
efficientnet-b3_3rdparty-ra-noisystudent_in1k	12.23	1.95	84.02	96.89	链接
efficientnet-b4_3rdparty_8xb32_in1k	19.34	4.66	82.57	96.09	链接
efficientnet-b4_3rdparty_8xb32-aa_in1k	19.34	4.66	82.95	96.26	链接
efficientnet-b4_3rdparty_8xb32-aa-advprop_in1k	19.34	4.66	83.25	96.44	链接
efficientnet-b4_3rdparty-ra-noisystudent_in1k	19.34	4.66	85.25	97.52	链接
efficientnet-b5_3rdparty_8xb32_in1k	30.39	10.80	83.18	96.47	链接
efficientnet-b5_3rdparty_8xb32-aa_in1k	30.39	10.80	83.82	96.76	链接
efficientnet-b5_3rdparty_8xb32-aa-advprop_in1k	30.39	10.80	84.21	96.98	链接
efficientnet-b5_3rdparty-ra-noisystudent_in1k	30.39	10.80	86.08	97.75	链接
efficientnet-b6_3rdparty_8xb32-aa_in1k	43.04	19.97	84.05	96.82	链接
efficientnet-b6_3rdparty_8xb32-aa-advprop_in1k	43.04	19.97	84.74	97.14	链接
efficientnet-b6_3rdparty-ra-noisystudent_in1k	43.04	19.97	86.47	97.87	链接
efficientnet-b7_3rdparty_8xb32-aa_in1k	66.35	39.32	84.38	96.88	链接
efficientnet-b7_3rdparty_8xb32-aa-advprop_in1k	66.35	39.32	85.14	97.23	链接
efficientnet-b7_3rdparty-ra-noisystudent_in1k	66.35	39.32	86.83	98.08	链接
efficientnet-b8_3rdparty_8xb32-aa-advprop_in1k	87.41	65.00	85.38	97.28	链接
efficientnet-l2_3rdparty-ra-noisystudent_in1k-800px	480.31	174.20	88.33	98.65	链接
efficientnet-l2_3rdparty-ra-noisystudent_in1k-475px	480.31	484.98	88.18	98.55	链接
convnext-tiny_32xb128_in1k	28.59	4.46	82.14	96.06	链接
convnext-tiny_32xb128-noema_in1k	28.59	4.46	81.95	95.89	链接
convnext-tiny_in21k-pre_3rdparty_in1k	28.59	4.46	82.90	96.62	链接
convnext-tiny_in21k-pre_3rdparty_in1k-384px	28.59	13.14	84.11	97.14	链接
convnext-small_32xb128_in1k	50.22	8.69	83.16	96.56	链接
convnext-small_32xb128-noema_in1k	50.22	8.69	83.21	96.48	链接
convnext-small_in21k-pre_3rdparty_in1k	50.22	8.69	84.59	97.41	链接
convnext-small_in21k-pre_3rdparty_in1k-384px	50.22	25.58	85.75	97.88	链接
convnext-base_32xb128_in1k	88.59	15.36	83.66	96.74	链接
convnext-base_32xb128-noema_in1k	88.59	15.36	83.64	96.61	链接
convnext-base_3rdparty_in1k	88.59	15.36	83.85	96.74	链接
convnext-base_3rdparty-noema_in1k	88.59	15.36	83.71	96.60	链接
convnext-base_3rdparty_in1k-384px	88.59	45.21	85.10	97.34	链接
convnext-base_in21k-pre_3rdparty_in1k	88.59	15.36	85.81	97.86	链接

下页继续

表 1 – 续上页

模型	参数量 (M)	Flops (G)	Top-1 (%)	Top-5 (%)	Readme
convnext-base_in21k-pre-3rdparty_in1k-384px	88.59	45.21	86.82	98.25	链接
convnext-large_3rdparty_in1k	197.77	34.37	84.30	96.89	链接
convnext-large_3rdparty_in1k-384px	197.77	101.10	85.50	97.59	链接
convnext-large_in21k-pre_3rdparty_in1k	197.77	34.37	86.61	98.04	链接
convnext-large_in21k-pre-3rdparty_in1k-384px	197.77	101.10	87.46	98.37	链接
convnext-xlarge_in21k-pre_3rdparty_in1k	350.20	60.93	86.97	98.20	链接
convnext-xlarge_in21k-pre-3rdparty_in1k-384px	350.20	179.20	87.76	98.55	链接
hrnet-w18_3rdparty_8xb32_in1k	21.30	4.33	76.75	93.44	链接
hrnet-w30_3rdparty_8xb32_in1k	37.71	8.17	78.19	94.22	链接
hrnet-w32_3rdparty_8xb32_in1k	41.23	8.99	78.44	94.19	链接
hrnet-w40_3rdparty_8xb32_in1k	57.55	12.77	78.94	94.47	链接
hrnet-w44_3rdparty_8xb32_in1k	67.06	14.96	78.88	94.37	链接
hrnet-w48_3rdparty_8xb32_in1k	77.47	17.36	79.32	94.52	链接
hrnet-w64_3rdparty_8xb32_in1k	128.06	29.00	79.46	94.65	链接
hrnet-w18_3rdparty_8xb32-ssld_in1k	21.30	4.33	81.06	95.70	链接
hrnet-w48_3rdparty_8xb32-ssld_in1k	77.47	17.36	83.63	96.79	链接
repmlp-base_3rdparty_8xb64_in1k	68.24	6.71	80.41	95.14	链接
repmlp-base_3rdparty_8xb64_in1k-256px.py	96.45	9.69	81.11	95.50	链接
wide-resnet50_3rdparty_8xb32_in1k	68.88	11.44	78.48	94.08	链接
wide-resnet101_3rdparty_8xb32_in1k	126.89	22.81	78.84	94.28	链接
wide-resnet50_3rdparty-timm_8xb32_in1k	68.88	11.44	81.45	95.53	链接
van-tiny_8xb128_in1k	4.11	0.88	75.41	93.02	链接
van-small_8xb128_in1k	13.86	2.52	81.01	95.63	链接
van-base_8xb128_in1k	26.58	5.03	82.80	96.21	链接
van-large_8xb128_in1k	44.77	8.99	83.86	96.73	链接
cspdarknet50_3rdparty_8xb32_in1k	27.64	5.04	80.05	95.07	链接
cspresnet50_3rdparty_8xb32_in1k	21.62	3.48	79.55	94.68	链接
cspresnext50_3rdparty_8xb32_in1k	20.57	3.11	79.96	94.96	链接
convmixer-768-32_10xb64_in1k	21.11	19.62	80.16	95.08	链接
convmixer-1024-20_10xb64_in1k	24.38	5.55	76.94	93.36	链接
convmixer-1536-20_10xb64_in1k	51.63	48.71	81.37	95.61	链接
densenet121_4xb256_in1k	7.98	2.88	74.96	92.21	链接
densenet169_4xb256_in1k	14.15	3.42	76.08	93.11	链接
densenet201_4xb256_in1k	20.01	4.37	77.32	93.64	链接
densenet161_4xb256_in1k	28.68	7.82	77.61	93.83	链接
poolformer-s12_3rdparty_32xb128_in1k	11.92	1.87	77.24	93.51	链接
poolformer-s24_3rdparty_32xb128_in1k	21.39	3.51	80.33	95.05	链接

下页继续

表 1 – 续上页

模型	参数量 (M)	Flops (G)	Top-1 (%)	Top-5 (%)	Readme
poolformer-s36_3rdparty_32xb128_in1k	30.86	5.15	81.43	95.45	链接
poolformer-m36_3rdparty_32xb128_in1k	56.17	8.96	82.14	95.71	链接
poolformer-m48_3rdparty_32xb128_in1k	73.47	11.80	82.51	95.95	链接
inception-v3_3rdparty_8xb32_in1k	23.83	5.75	77.57	93.58	链接
mvitv2-tiny_3rdparty_in1k	24.17	4.70	82.33	96.15	链接
mvitv2-small_3rdparty_in1k	34.87	7.00	83.63	96.51	链接
mvitv2-base_3rdparty_in1k	51.47	10.16	84.34	96.86	链接
mvitv2-large_3rdparty_in1k	217.99	43.87	85.25	97.14	链接
edgenext-xxsmall_3rdparty_in1k	1.33	0.26	71.20	89.91	链接
edgenext-xsmall_3rdparty_in1k	2.34	0.53	74.86	92.31	链接
edgenext-small_3rdparty_in1k	5.59	1.25	79.41	94.53	链接
edgenext-small-usi_3rdparty_in1k	5.59	1.25	81.06	95.34	链接
edgenext-base_3rdparty_in1k	18.51	3.81	82.48	96.20	链接
edgenext-base_3rdparty-usi_in1k	18.51	3.81	83.67	96.70	链接
mobileone-s0_8xb32_in1k	2.08	0.27	71.34	89.87	链接
mobileone-s1_8xb32_in1k	4.76	0.82	75.72	92.54	链接
mobileone-s2_8xb32_in1k	7.81	1.30	77.37	93.34	链接
mobileone-s3_8xb32_in1k	10.08	1.89	78.06	93.83	链接
mobileone-s4_8xb32_in1k	14.84	2.98	79.69	94.46	链接
efficientformer-l1_3rdparty_8xb128_in1k	12.28	1.30	80.46	94.99	链接
efficientformer-l3_3rdparty_8xb128_in1k	31.41	3.74	82.45	96.18	链接
efficientformer-l7_3rdparty_8xb128_in1k	82.23	10.16	83.40	96.60	链接
swinv2-tiny-w8_3rdparty_in1k-256px	28.35	4.35	81.76	95.87	链接
swinv2-tiny-w16_3rdparty_in1k-256px	28.35	4.40	82.81	96.23	链接
swinv2-small-w8_3rdparty_in1k-256px	49.73	8.45	83.74	96.60	链接
swinv2-small-w16_3rdparty_in1k-256px	49.73	8.57	84.13	96.83	链接
swinv2-base-w8_3rdparty_in1k-256px	87.92	14.99	84.20	96.86	链接
swinv2-base-w16_3rdparty_in1k-256px	87.92	15.14	84.60	97.05	链接
swinv2-base-w16_in21k-pre_3rdparty_in1k-256px	87.92	15.14	86.17	97.88	链接
swinv2-base-w24_in21k-pre_3rdparty_in1k-384px	87.92	34.07	87.14	98.23	链接
swinv2-large-w16_in21k-pre_3rdparty_in1k-256px	196.75	33.86	86.93	98.06	链接
swinv2-large-w24_in21k-pre_3rdparty_in1k-384px	196.75	76.20	87.59	98.27	链接
deit3-small-p16_3rdparty_in1k	22.06	4.61	81.35	95.31	链接
deit3-small-p16_3rdparty_in1k-384px	22.21	15.52	83.43	96.68	链接
deit3-small-p16_in21k-pre_3rdparty_in1k	22.06	4.61	83.06	96.77	链接
deit3-small-p16_in21k-pre_3rdparty_in1k-384px	22.21	15.52	84.84	97.48	链接
deit3-medium-p16_3rdparty_in1k	38.85	8.00	82.99	96.22	链接

下页继续

表 1 – 续上页

模型	参数量 (M)	Flops (G)	Top-1 (%)	Top-5 (%)	Readme
deit3-medium-p16_in21k-pre_3rdparty_in1k	38.85	8.00	84.56	97.19	链接
deit3-base-p16_3rdparty_in1k	86.59	17.58	83.80	96.55	链接
deit3-base-p16_3rdparty_in1k-384px	86.88	55.54	85.08	97.25	链接
deit3-base-p16_in21k-pre_3rdparty_in1k	86.59	17.58	85.70	97.75	链接
deit3-base-p16_in21k-pre_3rdparty_in1k-384px	86.88	55.54	86.73	98.11	链接
deit3-large-p16_3rdparty_in1k	304.37	61.60	84.87	97.01	链接
deit3-large-p16_3rdparty_in1k-384px	304.76	191.21	85.82	97.60	链接
deit3-large-p16_in21k-pre_3rdparty_in1k	304.37	61.60	86.97	98.24	链接
deit3-large-p16_in21k-pre_3rdparty_in1k-384px	304.76	191.21	87.73	98.51	链接
deit3-huge-p14_3rdparty_in1k	632.13	167.40	85.21	97.36	链接
deit3-huge-p14_in21k-pre_3rdparty_in1k	632.13	167.40	87.19	98.26	链接
hornet-tiny_3rdparty_in1k	22.41	3.98	82.84	96.24	链接
hornet-tiny-gf_3rdparty_in1k	22.99	3.90	82.98	96.38	链接
hornet-small_3rdparty_in1k	49.53	8.83	83.79	96.75	链接
hornet-small-gf_3rdparty_in1k	50.40	8.71	83.98	96.77	链接
hornet-base_3rdparty_in1k	87.26	15.58	84.24	96.94	链接
hornet-base-gf_3rdparty_in1k	88.42	15.42	84.32	96.95	链接
mobilevit-small_3rdparty_in1k	5.58	2.03	78.25	94.09	链接
mobilevit-xsmall_3rdparty_in1k	2.32	1.05	74.75	92.32	链接
mobilevit-xxsmall_3rdparty_in1k	1.27	0.42	69.02	88.91	链接
davit-tiny_3rdparty_in1k	28.36	4.54	82.24	96.13	链接
davit-small_3rdparty_in1k	49.75	8.80	83.61	96.75	链接
davit-base_3rdparty_in1k	87.95	15.51	84.09	96.82	链接
replknet-31B_3rdparty_in1k	79.86	15.64	83.48	96.57	链接
replknet-31B_3rdparty_in1k-384px	79.86	45.95	84.84	97.34	链接
replknet-31B_in21k-pre_3rdparty_in1k	79.86	15.64	85.20	97.56	链接
replknet-31B_in21k-pre_3rdparty_in1k-384px	79.86	45.95	85.99	97.75	链接
replknet-31L_in21k-pre_3rdparty_in1k-384px	172.67	97.24	86.63	98.00	链接
replknet-XL_meg73m-pre_3rdparty_in1k-320px	335.44	129.57	87.57	98.39	链接
beit-base_3rdparty_in1k	86.53	17.58	85.28	97.59	链接
beitv2-base_3rdparty_in1k	86.53	17.58	86.47	97.99	链接
eva-g-p14_30m-in21k-pre_3rdparty_in1k-336px	1013.01	620.64	89.61	98.93	链接
eva-g-p14_30m-in21k-pre_3rdparty_in1k-560px	1014.45	1906.76	89.71	98.96	链接
eva-l-p14_mim-pre_3rdparty_in1k-336px	304.53	191.10	88.66	98.75	链接
eva-l-p14_mim-in21k-pre_3rdparty_in1k-336px	304.53	191.10	89.17	98.86	链接
eva-l-p14_mim-pre_3rdparty_in1k-196px	304.14	61.57	87.94	98.50	链接
eva-l-p14_mim-in21k-pre_3rdparty_in1k-196px	304.14	61.57	88.58	98.65	链接

下页继续

表 1 – 续上页

模型	参数量 (M)	Flops (G)	Top-1 (%)	Top-5 (%)	Readme
revvit-small_3rdparty_in1k	22.44	4.58	79.87	94.90	链接
revvit-base_3rdparty_in1k	87.34	17.49	81.81	95.56	链接
clip-vit-base-p32_laion2b-in12k-pre_3rdparty_in1k	88.22	4.36	83.06	96.49	链接
clip-vit-base-p32_laion2b-pre_3rdparty_in1k	88.22	4.36	82.46	96.12	链接
clip-vit-base-p32_openai-pre_3rdparty_in1k	88.22	4.36	81.77	95.89	链接
clip-vit-base-p32_laion2b-in12k-pre_3rdparty_in1k-384px	88.22	12.66	85.39	97.67	链接
clip-vit-base-p32_openai-in12k-pre_3rdparty_in1k-384px	88.22	12.66	85.13	97.42	链接
clip-vit-base-p16_laion2b-in12k-pre_3rdparty_in1k	86.57	16.86	86.02	97.76	链接
clip-vit-base-p16_laion2b-pre_3rdparty_in1k	86.57	16.86	85.49	97.59	链接
clip-vit-base-p16_openai-in12k-pre_3rdparty_in1k	86.57	16.86	85.99	97.72	链接
clip-vit-base-p16_openai-pre_3rdparty_in1k	86.57	16.86	85.30	97.50	链接
clip-vit-base-p32_laion2b-in12k-pre_3rdparty_in1k-448px	88.22	17.20	85.76	97.63	链接
clip-vit-base-p16_laion2b-in12k-pre_3rdparty_in1k-384px	86.57	49.37	87.17	98.02	链接
clip-vit-base-p16_laion2b-pre_3rdparty_in1k-384px	86.57	49.37	86.52	97.97	链接
clip-vit-base-p16_openai-in12k-pre_3rdparty_in1k-384px	86.57	49.37	86.87	98.05	链接
clip-vit-base-p16_openai-pre_3rdparty_in1k-384px	86.57	49.37	86.25	97.90	链接
mixmim-base_3rdparty_in1k	88.34	16.35	84.60	97.00	链接
efficientnetv2-b0_3rdparty_in1k	7.14	0.92	78.52	94.44	链接
efficientnetv2-b1_3rdparty_in1k	8.14	1.44	79.80	94.89	链接
efficientnetv2-b2_3rdparty_in1k	10.10	1.99	80.63	95.30	链接
efficientnetv2-b3_3rdparty_in1k	14.36	3.50	82.03	95.88	链接
efficientnetv2-s_3rdparty_in1k	21.46	9.72	83.82	96.67	链接
efficientnetv2-m_3rdparty_in1k	54.14	26.88	85.01	97.26	链接
efficientnetv2-l_3rdparty_in1k	118.52	60.14	85.43	97.31	链接
efficientnetv2-s_in21k-pre_3rdparty_in1k	21.46	9.72	84.29	97.26	链接
efficientnetv2-m_in21k-pre_3rdparty_in1k	54.14	26.88	85.47	97.76	链接
efficientnetv2-l_in21k-pre_3rdparty_in1k	118.52	60.14	86.31	97.99	链接
efficientnetv2-xl_in21k-pre_3rdparty_in1k	208.12	98.34	86.39	97.83	链接
convnext-v2-atto_fcmae-pre_3rdparty_in1k	3.71	0.55	76.64	93.04	链接
convnext-v2-femto_fcmae-pre_3rdparty_in1k	5.23	0.78	78.48	93.98	链接
convnext-v2-pico_fcmae-pre_3rdparty_in1k	9.07	1.37	80.31	95.08	链接
convnext-v2-nano_fcmae-pre_3rdparty_in1k	15.62	2.45	81.86	95.75	链接
convnext-v2-nano_fcmae-in21k-pre_3rdparty_in1k	15.62	2.45	82.04	96.16	链接
convnext-v2-tiny_fcmae-pre_3rdparty_in1k	28.64	4.47	82.94	96.29	链接
convnext-v2-tiny_fcmae-in21k-pre_3rdparty_in1k	28.64	4.47	83.89	96.96	链接
convnext-v2-nano_fcmae-in21k-pre_3rdparty_in1k-384px	15.62	7.21	83.36	96.75	链接
convnext-v2-tiny_fcmae-in21k-pre_3rdparty_in1k-384px	28.64	13.14	85.09	97.63	链接

下页继续

表 1 – 续上页

模型	参数量 (M)	Flops (G)	Top-1 (%)	Top-5 (%)	Readme
convnext-v2-base_fcmae-pre_3rdparty_in1k	88.72	15.38	84.87	97.08	链接
convnext-v2-base_fcmae-in21k-pre_3rdparty_in1k	88.72	15.38	86.74	98.02	链接
convnext-v2-large_fcmae-pre_3rdparty_in1k	197.96	34.40	85.76	97.59	链接
convnext-v2-large_fcmae-in21k-pre_3rdparty_in1k	197.96	34.40	87.26	98.24	链接
convnext-v2-base_fcmae-in21k-pre_3rdparty_in1k-384px	88.72	45.21	87.63	98.42	链接
convnext-v2-large_fcmae-in21k-pre_3rdparty_in1k-384px	197.96	101.10	88.18	98.52	链接
convnext-v2-huge_fcmae-pre_3rdparty_in1k	660.29	115.00	86.25	97.75	链接
convnext-v2-huge_fcmae-in21k-pre_3rdparty_in1k-384px	660.29	337.96	88.68	98.73	链接
convnext-v2-huge_fcmae-in21k-pre_3rdparty_in1k-512px	660.29	600.81	88.86	98.74	链接
levit-128s_3rdparty_in1k	7.39	0.31	76.51	92.90	链接
levit-128_3rdparty_in1k	8.83	0.41	78.58	93.95	链接
levit-192_3rdparty_in1k	10.56	0.67	79.86	94.75	链接
levit-256_3rdparty_in1k	18.38	1.14	81.59	95.46	链接
levit-384_3rdparty_in1k	38.36	2.37	82.59	95.95	链接
vig-tiny_3rdparty_in1k	7.18	1.31	74.40	92.34	链接
vig-small_3rdparty_in1k	22.75	4.54	80.61	95.28	链接
vig-base_3rdparty_in1k	20.68	17.68	82.62	96.04	链接
pvig-tiny_3rdparty_in1k	9.46	1.71	78.38	94.38	链接
pvig-small_3rdparty_in1k	29.02	4.57	82.00	95.97	链接
pvig-medium_3rdparty_in1k	51.68	8.89	83.12	96.35	链接
pvig-base_3rdparty_in1k	95.21	16.86	83.59	96.52	链接
xcit-nano-12-p16_3rdparty_in1k	3.05	0.56	70.35	89.98	链接
xcit-nano-12-p16_3rdparty-dist_in1k	3.05	0.56	72.36	91.02	链接
xcit-tiny-12-p16_3rdparty_in1k	6.72	1.24	77.21	93.62	链接
xcit-tiny-12-p16_3rdparty-dist_in1k	6.72	1.24	78.70	94.12	链接
xcit-nano-12-p16_3rdparty-dist_in1k-384px	3.05	1.64	74.93	92.42	链接
xcit-nano-12-p8_3rdparty_in1k	3.05	2.16	73.80	92.08	链接
xcit-nano-12-p8_3rdparty-dist_in1k	3.05	2.16	76.17	93.08	链接
xcit-tiny-24-p16_3rdparty_in1k	12.12	2.34	79.47	94.85	链接
xcit-tiny-24-p16_3rdparty-dist_in1k	12.12	2.34	80.51	95.17	链接
xcit-tiny-12-p16_3rdparty-dist_in1k-384px	6.72	3.64	80.58	95.38	链接
xcit-tiny-12-p8_3rdparty_in1k	6.71	4.81	79.75	94.88	链接
xcit-tiny-12-p8_3rdparty-dist_in1k	6.71	4.81	81.26	95.46	链接
xcit-small-12-p16_3rdparty_in1k	26.25	4.81	81.87	95.77	链接
xcit-small-12-p16_3rdparty-dist_in1k	26.25	4.81	83.12	96.41	链接
xcit-nano-12-p8_3rdparty-dist_in1k-384px	3.05	6.34	77.69	94.09	链接
xcit-tiny-24-p16_3rdparty-dist_in1k-384px	12.12	6.87	82.43	96.20	链接

下页继续

表 1 – 续上页

模型	参数量 (M)	Flops (G)	Top-1 (%)	Top-5 (%)	Readme
xcit-small-24-p16_3rdparty_in1k	47.67	9.10	82.38	95.93	链接
xcit-small-24-p16_3rdparty-dist_in1k	47.67	9.10	83.70	96.61	链接
xcit-tiny-24-p8_3rdparty_in1k	12.11	9.21	81.70	95.90	链接
xcit-tiny-24-p8_3rdparty-dist_in1k	12.11	9.21	82.62	96.16	链接
xcit-tiny-12-p8_3rdparty-dist_in1k-384px	6.71	14.13	82.46	96.22	链接
xcit-small-12-p16_3rdparty-dist_in1k-384px	26.25	14.14	84.74	97.19	链接
xcit-medium-24-p16_3rdparty_in1k	84.40	16.13	82.56	95.82	链接
xcit-medium-24-p16_3rdparty-dist_in1k	84.40	16.13	84.15	96.82	链接
xcit-small-12-p8_3rdparty_in1k	26.21	18.69	83.21	96.41	链接
xcit-small-12-p8_3rdparty-dist_in1k	26.21	18.69	83.97	96.81	链接
xcit-small-24-p16_3rdparty-dist_in1k-384px	47.67	26.72	85.10	97.32	链接
xcit-tiny-24-p8_3rdparty-dist_in1k-384px	12.11	27.05	83.77	96.72	链接
xcit-small-24-p8_3rdparty_in1k	47.63	35.81	83.62	96.51	链接
xcit-small-24-p8_3rdparty-dist_in1k	47.63	35.81	84.68	97.07	链接
xcit-large-24-p16_3rdparty_in1k	189.10	35.86	82.97	95.86	链接
xcit-large-24-p16_3rdparty-dist_in1k	189.10	35.86	84.61	97.07	链接
xcit-medium-24-p16_3rdparty-dist_in1k-384px	84.40	47.39	85.47	97.49	链接
xcit-small-12-p8_3rdparty-dist_in1k-384px	26.21	54.92	85.12	97.31	链接
xcit-medium-24-p8_3rdparty_in1k	84.32	63.52	83.61	96.23	链接
xcit-medium-24-p8_3rdparty-dist_in1k	84.32	63.52	85.00	97.16	链接
xcit-small-24-p8_3rdparty-dist_in1k-384px	47.63	105.24	85.57	97.60	链接
xcit-large-24-p16_3rdparty-dist_in1k-384px	189.10	105.35	85.78	97.60	链接
xcit-large-24-p8_3rdparty_in1k	188.93	141.23	84.23	96.58	链接
xcit-large-24-p8_3rdparty-dist_in1k	188.93	141.23	85.14	97.32	链接
xcit-medium-24-p8_3rdparty-dist_in1k-384px	84.32	186.67	85.87	97.61	链接
xcit-large-24-p8_3rdparty-dist_in1k-384px	188.93	415.00	86.13	97.75	链接

24.2.2 CIFAR-10

模型	参数量 (M)	Flops (G)	Top-1 (%)	Top-5 (%)	Readme
resnet18_8xb16_cifar10	11.17	0.56	94.82		链接
resnet34_8xb16_cifar10	21.28	1.16	95.34		链接
resnet50_8xb16_cifar10	23.52	1.31	95.55		链接
resnet101_8xb16_cifar10	42.51	2.52	95.58		链接
resnet152_8xb16_cifar10	58.16	3.74	95.76		链接

24.2.3 CIFAR-100

模型	参数量 (M)	Flops (G)	Top-1 (%)	Top-5 (%)	Readme
resnet50_8xb16_cifar100	23.71	1.31	79.90	95.19	链接

24.2.4 CUB-200-2011

模型	参数量 (M)	Flops (G)	Top-1 (%)	Top-5 (%)	Readme
resnet50_8xb8_cub	23.92	16.48	88.45		链接
swin-large_8xb8_cub_384px	195.51	100.04	91.87		链接

24.2.5 PASCAL VOC 2007

模型	参数量 (M)	Flops (G)	Top-1 (%)	Top-5 (%)	Readme
resnet101-csra_1xb16_voc07-448px	23.55	4.12			链接

24.2.6 InShop

模型	参数量 (M)	Flops (G)	Top-1 (%)	Top-5 (%)	Readme
resnet50-arcface_inshop	31.69	16.57			链接

CHAPTER 25

ArcFace

ArcFace: Additive Angular Margin Loss for Deep Face Recognition

25.1 摘要

Recently, a popular line of research in face recognition is adopting margins in the well-established softmax loss function to maximize class separability. In this paper, we first introduce an Additive Angular Margin Loss (ArcFace), which not only has a clear geometric interpretation but also significantly enhances the discriminative power. Since ArcFace is susceptible to the massive label noise, we further propose sub-center ArcFace, in which each class contains K sub-centers and training samples only need to be close to any of the K positive sub-centers. Sub-center ArcFace encourages one dominant sub-class that contains the majority of clean faces and non-dominant sub-classes that include hard or noisy faces. Based on this self-propelled isolation, we boost the performance through automatically purifying raw web faces under massive real-world noise. Besides discriminative feature embedding, we also explore the inverse problem, mapping feature vectors to face images. Without training any additional generator or discriminator, the pre-trained ArcFace model can generate identity-preserved face images for both subjects inside and outside the training data only by using the network gradient and Batch Normalization (BN) priors. Extensive experiments demonstrate that ArcFace can enhance the discriminative feature embedding as well as strengthen the generative face synthesis.

25.2 结果和模型

25.2.1 InShop

Model	Pretrain	Params(M)	Flops(G)	Recall@1	Config	Download
Resnet50-ArcFace	ImageNet-21k-mill	31.69	16.48	90.18	config	model | log

25.3 引用

```
@inproceedings{deng2018arcface,
title={ArcFace: Additive Angular Margin Loss for Deep Face Recognition},
author={Deng, Jiankang and Guo, Jia and Niannan, Xue and Zafeiriou, Stefanos},
booktitle={CVPR},
year={2019}
}
```

CHAPTER 26

BEiT

BEiT: BERT Pre-Training of Image Transformers

26.1 摘要

We introduce a self-supervised vision representation model BEiT, which stands for Bidirectional Encoder representation from Image Transformers. Following BERT developed in the natural language processing area, we propose a masked image modeling task to pretrain vision Transformers. Specifically, each image has two views in our pre-training, i.e, image patches (such as 16x16 pixels), and visual tokens (i.e., discrete tokens). We first "tokenize" the original image into visual tokens. Then we randomly mask some image patches and fed them into the backbone Transformer. The pre-training objective is to recover the original visual tokens based on the corrupted image patches. After pre-training BEiT, we directly fine-tune the model parameters on downstream tasks by appending task layers upon the pretrained encoder. Experimental results on image classification and semantic segmentation show that our model achieves competitive results with previous pre-training methods. For example, base-size BEiT achieves 83.2% top-1 accuracy on ImageNet-1K, significantly outperforming from-scratch DeiT training (81.8%) with the same setup. Moreover, large-size BEiT obtains 86.3% only using ImageNet-1K, even outperforming ViT-L with supervised pre-training on ImageNet-22K (85.2%).

26.2 结果和模型

26.2.1 ImageNet-1k

模型	预训练	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
BEiT-base*	ImageNet-21k	86.53	17.58	85.28	97.59	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference.*

For BEiT self-supervised learning algorithm, welcome to MMSelfSup page to get more information.

26.3 引用

```
@article{beit,
    title={{BEiT}: {BERT} Pre-Training of Image Transformers},
    author={Hangbo Bao and Li Dong and Furu Wei},
    year={2021},
    eprint={2106.08254},
    archivePrefix={arXiv},
    primaryClass={cs.CV}
}
```

CHAPTER 27

BEiT V2

BEiT v2: Masked Image Modeling with Vector-Quantized Visual Tokenizers

27.1 摘要

Masked image modeling (MIM) has demonstrated impressive results in self-supervised representation learning by recovering corrupted image patches. However, most existing studies operate on low-level image pixels, which hinders the exploitation of high-level semantics for representation models. In this work, we propose to use a semantic-rich visual tokenizer as the reconstruction target for masked prediction, providing a systematic way to promote MIM from pixel-level to semantic-level. Specifically, we propose vector-quantized knowledge distillation to train the tokenizer, which discretizes a continuous semantic space to compact codes. We then pretrain vision Transformers by predicting the original visual tokens for the masked image patches. Furthermore, we introduce a patch aggregation strategy which associates discrete image patches to enhance global semantic representation. Experiments on image classification and semantic segmentation show that BEiT v2 outperforms all compared MIM methods. On ImageNet-1K (224 size), the base-size BEiT v2 achieves 85.5% top-1 accuracy for fine-tuning and 80.1% top-1 accuracy for linear probing. The large-size BEiT v2 obtains 87.3% top-1 accuracy for ImageNet-1K (224 size) fine-tuning, and 56.7% mIoU on ADE20K for semantic segmentation.

27.2 结果和模型

27.2.1 ImageNet-1k

模型	预训练	参数量 (M)	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
BEiTv2-base*	ImageNet-1k & ImageNet-21k	86.53	17.58	86.47	97.99	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference.*

For BEiTv2 self-supervised learning algorithm, welcome to MMSelfSup page to get more information.

27.3 引用

```
@article{beitv2,
    title={{BEiT v2}: Masked Image Modeling with Vector-Quantized Visual Tokenizers},
    author={Zhiliang Peng and Li Dong and Hangbo Bao and Qixiang Ye and Furu Wei},
    year={2022},
    eprint={2208.06366},
    archivePrefix={arXiv},
    primaryClass={cs.CV}
}
```

CHAPTER 28

CLIP

Learning Transferable Visual Models From Natural Language Supervision

28.1 摘要

State-of-the-art computer vision systems are trained to predict a fixed set of predetermined object categories. This restricted form of supervision limits their generality and usability since additional labeled data is needed to specify any other visual concept. Learning directly from raw text about images is a promising alternative which leverages a much broader source of supervision. We demonstrate that the simple pre-training task of predicting which caption goes with which image is an efficient and scalable way to learn SOTA image representations from scratch on a dataset of 400 million (image, text) pairs collected from the internet. After pre-training, natural language is used to reference learned visual concepts (or describe new ones) enabling zero-shot transfer of the model to downstream tasks. We study the performance of this approach by benchmarking on over 30 different existing computer vision datasets, spanning tasks such as OCR, action recognition in videos, geo-localization, and many types of fine-grained object classification. The model transfers non-trivially to most tasks and is often competitive with a fully supervised baseline without the need for any dataset specific training. For instance, we match the accuracy of the original ResNet-50 on ImageNet zero-shot without needing to use any of the 1.28 million training examples it was trained on. We release our code and pre-trained model weights at this https URL.

28.2 结果和模型

28.2.1 ImageNet-1k

模型	预训练	参数量 (M)	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
clip-vit-base-p32_laion2b-in12k-pre_3rdparty_in1k*	LAION-2B & ImageNet-12k	88.22	4.36	83.06	96.49	config	model
clip-vit-base-p32_laion2b-pre_3rdparty_in1k*	LAION-2B	88.22	4.36	82.46	96.12	config	model
clip-vit-base-p32_openai-pre_3rdparty_in1k*	OpenAI	88.22	4.36	81.77	95.89	config	model
clip-vit-base-p32_laion2b-in12k-pre_3rdparty_in1k-384px*	LAION-2B & ImageNet-12k	88.22	12.66	85.39	97.67	config	model
clip-vit-base-p32_openai-in12k-pre_3rdparty_in1k-384px*	OpenAI & ImageNet-12k	88.22	12.66	85.13	97.42	config	model
clip-vit-base-p16_laion2b-in12k-pre_3rdparty_in1k*	LAION-2B & ImageNet-12k	86.57	16.86	86.02	97.76	config	model
clip-vit-base-p16_laion2b-pre_3rdparty_in1k*	LAION-2B	86.57	16.86	85.49	97.59	config	model
clip-vit-base-p16_openai-in12k-pre_3rdparty_in1k*	OpenAI & ImageNet-12k	86.57	16.86	85.99	97.72	config	model
clip-vit-base-p16_openai-pre_3rdparty_in1k*	OpenAI	86.57	16.86	85.30	97.50	config	model
clip-vit-base-p32_laion2b-in12k-pre_3rdparty_in1k-448px*	LAION-2B & ImageNet-12k	88.22	17.20	85.76	97.63	config	model
clip-vit-base-p16_laion2b-in12k-pre_3rdparty_in1k-384px*	LAION-2B & ImageNet-12k	86.57	49.37	87.17	98.02	config	model
clip-vit-base-p16_laion2b-pre_3rdparty_in1k-384px*	LAION-2B	86.57	49.37	86.52	97.97	config	model
clip-vit-base-p16_openai-in12k-pre_3rdparty_in1k-384px*	OpenAI & ImageNet-12k	86.57	49.37	86.87	98.05	config	model
clip-vit-base-p16_openai-pre_3rdparty_in1k-384px*	OpenAI	86.57	49.37	86.25	97.90	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

28.3 引用

```
@InProceedings{pmlr-v139-radford21a,
title = {Learning Transferable Visual Models From Natural Language Supervision},
author = {Radford, Alec and Kim, Jong Wook and Hallacy, Chris and Ramesh, Aditya and␣
↪Goh, Gabriel and Agarwal, Sandhini and Sastry, Girish and Askell, Amanda and␣
↪Mishkin, Pamela and Clark, Jack and Krueger, Gretchen and Sutskever, Ilya},
booktitle = {Proceedings of the 38th International Conference on Machine Learning},
year = {2021},
series = {Proceedings of Machine Learning Research},
publisher = {PMLR},
}
```

CHAPTER 29

Conformer

Conformer: Local Features Coupling Global Representations for Visual Recognition

29.1 摘要

Within Convolutional Neural Network (CNN), the convolution operations are good at extracting local features but experience difficulty to capture global representations. Within visual transformer, the cascaded self-attention modules can capture long-distance feature dependencies but unfortunately deteriorate local feature details. In this paper, we propose a hybrid network structure, termed Conformer, to take advantage of convolutional operations and self-attention mechanisms for enhanced representation learning. Conformer roots in the Feature Coupling Unit (FCU), which fuses local features and global representations under different resolutions in an interactive fashion. Conformer adopts a concurrent structure so that local features and global representations are retained to the maximum extent. Experiments show that Conformer, under the comparable parameter complexity, outperforms the visual transformer (DeiT-B) by 2.3% on ImageNet. On MSCOCO, it outperforms ResNet-101 by 3.7% and 3.6% mAPs for object detection and instance segmentation, respectively, demonstrating the great potential to be a general backbone network.

29.2 结果和模型

29.2.1 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
Conformer-tiny-p16*	23.52	4.90	81.31	95.60	config	model
Conformer-small-p32*	38.85	7.09	81.96	96.02	config	model
Conformer-small-p16*	37.67	10.31	83.32	96.46	config	model
Conformer-base-p16*	83.29	22.89	83.82	96.59	config	model

*Models with * are converted from the official repo. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

29.3 引用

```
@article{peng2021conformer,
      title={Conformer: Local Features Coupling Global Representations for Visual␣
↪Recognition},
      author={Zhiliang Peng and Wei Huang and Shanzhi Gu and Lingxi Xie and Yaowei␣
↪Wang and Jianbin Jiao and Qixiang Ye},
      journal={arXiv preprint arXiv:2105.03889},
      year={2021},
}
```

CHAPTER 30

ConvMixer

Patches Are All You Need?

30.1 摘要

Although convolutional networks have been the dominant architecture for vision tasks for many years, recent experiments have shown that Transformer-based models, most notably the Vision Transformer (ViT), may exceed their performance in some settings. However, due to the quadratic runtime of the self-attention layers in Transformers, ViTs require the use of patch embeddings, which group together small regions of the image into single input features, in order to be applied to larger image sizes. This raises a question: Is the performance of ViTs due to the inherently-more-powerful Transformer architecture, or is it at least partly due to using patches as the input representation? In this paper, we present some evidence for the latter: specifically, we propose the ConvMixer, an extremely simple model that is similar in spirit to the ViT and the even-more-basic MLP-Mixer in that it operates directly on patches as input, separates the mixing of spatial and channel dimensions, and maintains equal size and resolution throughout the network. In contrast, however, the ConvMixer uses only standard convolutions to achieve the mixing steps. Despite its simplicity, we show that the ConvMixer outperforms the ViT, MLP-Mixer, and some of their variants for similar parameter counts and data set sizes, in addition to outperforming classical vision models such as the ResNet.

30.2 结果和模型

30.2.1 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
ConvMixer-768/32*	21.11	19.62	80.16	95.08	config	model
ConvMixer-1024/20*	24.38	5.55	76.94	93.36	config	model
ConvMixer-1536/20*	51.63	48.71	81.37	95.61	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

30.3 引用

```
@misc{trockman2022patches,
      title={Patches Are All You Need?},
      author={Asher Trockman and J. Zico Kolter},
      year={2022},
      eprint={2201.09792},
      archivePrefix={arXiv},
      primaryClass={cs.CV}
}
```

CHAPTER 31

ConvNeXt

A ConvNet for the 2020s

31.1 简介

ConvNeXt is initially described in A ConvNet for the 2020s, which is a pure convolutional model (ConvNet), inspired by the design of Vision Transformers. The ConvNeXt has the pyramid structure and achieve competitive performance on various vision tasks, with simplicity and efficiency.

31.2 摘要

31.3 使用方式

推理图片

```
>>> import torch
>>> from mmcls.apis import get_model, inference_model
>>>
>>> model = get_model('convnext-tiny_32xb128_in1k', pretrained=True)
>>> predict = inference_model(model, 'demo/demo.JPEG')
>>> print(predict['pred_class'])
sea snake
```

(下页继续)

(续上页)

```
>>> print(predict['pred_score'])
0.8915778398513794
```

调用模型

```
>>> import torch
>>> from mmcls.apis import get_model
>>>
>>> model = get_model('convnext-tiny_32xb128_in1k', pretrained=True)
>>> inputs = torch.rand(1, 3, 224, 224))
>>> # To get classification scores.
>>> out = model(inputs)
>>> print(out.shape)
torch.Size([1, 1000])
>>> # To extract features.
>>> outs = model.extract_feat(inputs)
>>> print(outs[0].shape)
torch.Size([1, 768])
```

训练/测试

将 ImageNet 数据集放置在 `data/imagenet` 目录下，或者根据 docs 准备其他数据集。

训练：

```
python tools/train.py configs/convnext/convnext-tiny_32xb128_in1k.py

Test:

```shell
python tools/test.py configs/convnext/convnext-tiny_32xb128_in1k.py https://download.
↪openmmlab.com/mmclassification/v0/convnext/convnext-tiny_3rdparty_32xb128-noema_
↪in1k_20220222-2908964a.pth
```

For more configurable parameters, please refer to the API.

31.4 结果和模型

31.4.1 Pre-trained Models

The pre-trained models on ImageNet-1k or ImageNet-21k are used to fine-tune on the downstream tasks.
```

模型	Training Data	参数量 (M)	Flops(G)	Top-1 (%)	Top-5 (%)	下载
ConvNeXt-T (convnext-tiny_32xb128-noema_in1k)	ImageNet-1k	28.59	4.46	81.95	95.89	model
ConvNeXt-S (convnext-small_32xb128-noema_in1k)	ImageNet-1k	50.22	8.69	83.21	96.48	model
ConvNeXt-B (convnext-base_32xb128-noema_in1k)	ImageNet-1k	88.59	15.36	83.64	96.61	model
ConvNeXt-B* (convnext-base_3rdparty-noema_in1k)	ImageNet-1k	88.59	15.36	83.71	96.60	model
ConvNeXt-B* (convnext-base_3rdparty_in21k)	ImageNet-21k	88.59	15.36	N/A	N/A	model
ConvNeXt-L* (convnext-large_3rdparty_in21k)	ImageNet-21k	197.77	34.37	N/A	N/A	model
ConvNeXt-XL* (convnext-xlarge_3rdparty_in21k)	ImageNet-21k	350.20	60.93	N/A	N/A	model

*Models with * are converted from the official repo.*

31.4.2 ImageNet-1k

模型	预训练	分辨率	参数量 (M)	Flops (G)	Top-1 (%)	Top-5 (%)	配置文件	下载
ConvNeXt-T (convnext-tiny_32xb128_in1k)	从头训练	224x224	28.59	4.46	82.14	96.06	config	model | log
ConvNeXt-T* (convnext-tiny_in21k-pre_3rdparty_in1k)	ImageNet-21k	224x224	28.59	4.46	82.90	96.62	config	model
ConvNeXt-T* (convnext-tiny_in21k-pre_3rdparty_in1k-384px)	ImageNet-21k	384x384	28.59	13.13	84.11	97.14	config	model
ConvNeXt-S (convnext-small_32xb128_in1k)	从头训练	224x224	50.22	8.69	83.16	96.56	config	model | log
ConvNeXt-S* (convnext-small_in21k-pre_3rdparty_in1k)	ImageNet-21k	224x224	50.22	8.69	84.59	97.41	config	model
ConvNeXt-S* (convnext-small_in21k-pre_3rdparty_in1k-384px)	ImageNet-21k	384x384	50.22	25.58	85.75	97.88	config	model
ConvNeXt-B (convnext-base_32xb128_in1k)	从头训练	224x224	88.59	15.36	83.66	96.74	config	model | log
ConvNeXt-B* (convnext-base_3rdparty_in1k)	从头训练	224x224	88.59	15.36	83.85	96.74	config	model
ConvNeXt-B (convnext-base_3rdparty_in1k-384px)	从头训练	384x384	88.59	45.21	85.10	97.34	config	model
ConvNeXt-B* (convnext-base_in21k-pre_3rdparty_in1k)	ImageNet-21k	224x224	88.59	15.36	85.81	97.86	config	model
ConvNeXt-B* (convnext-base_in21k-pre-3rdparty_in1k-384px)	ImageNet-21k	384x384	88.59	45.21	86.82	98.25	config	model
ConvNeXt-L* (convnext-large_3rdparty_in1k)	从头训练	224x224	197.77	34.37	84.30	96.89	config	model
ConvNeXt-L* (convnext-large_3rdparty_in1k-384px)	从头训练	384x384	197.77	101.10	85.50	97.59	config	model
ConvNeXt-L* (convnext-large_in21k-pre_3rdparty_in1k)	ImageNet-21k	224x224	197.77	34.37	86.61	98.04	config	model
ConvNeXt-L (convnext-large_in21k-pre-3rdparty_in1k-384px)*	ImageNet-21k	384x384	197.77	101.10	87.46	98.37	config	model
ConvNeXt-XL* (convnext-xlarge_in21k-pre_3rdparty_in1k)	ImageNet-21k	224x224	350.20	60.93	86.97	98.20	config	model
ConvNeXt-XL* (convnext-xlarge_in21k-pre-3rdparty_in1k-384px)	ImageNet-21k	384x384	350.20	179.20	87.76	98.55	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

31.5 引用

```
@Article{liu2022convnet,
  author  = {Zhuang Liu and Hanzi Mao and Chao-Yuan Wu and Christoph Feichtenhofer
↪and Trevor Darrell and Saining Xie},
  title   = {A ConvNet for the 2020s},
  journal = {arXiv preprint arXiv:2201.03545},
  year    = {2022},
}
```

CHAPTER 32

ConvNeXt V2

Co-designing and Scaling ConvNets with Masked Autoencoders

32.1 摘要

Driven by improved architectures and better representation learning frameworks, the field of visual recognition has enjoyed rapid modernization and performance boost in the early 2020s. For example, modern ConvNets, represented by ConvNeXt, have demonstrated strong performance in various scenarios. While these models were originally designed for supervised learning with ImageNet labels, they can also potentially benefit from self-supervised learning techniques such as masked autoencoders (MAE). However, we found that simply combining these two approaches leads to subpar performance. In this paper, we propose a fully convolutional masked autoencoder framework and a new Global Response Normalization (GRN) layer that can be added to the ConvNeXt architecture to enhance inter-channel feature competition. This co-design of self-supervised learning techniques and architectural improvement results in a new model family called ConvNeXt V2, which significantly improves the performance of pure ConvNets on various recognition benchmarks, including ImageNet classification, COCO detection, and ADE20K segmentation. We also provide pre-trained ConvNeXt V2 models of various sizes, ranging from an efficient 3.7M-parameter Atto model with 76.7% top-1 accuracy on ImageNet, to a 650M Huge model that achieves a state-of-the-art 88.9% accuracy using only public training data.

32.2 结果和模型

32.2.1 Pre-trained Models

The pre-trained models are only used to fine-tune, and therefore cannot be trained and don’t have evaluation results.

模型	参数量（M）	Flops(G)	配置文件	下载
convnext-v2-atto_3rdparty-fcmae_in1k*	3.71	0.55	config	model
convnext-v2-femto_3rdparty-fcmae_in1k*	5.23	0.78	config	model
convnext-v2-pico_3rdparty-fcmae_in1k*	9.07	1.37	config	model
convnext-v2-nano_3rdparty-fcmae_in1k*	15.62	2.45	config	model
convnext-v2-tiny_3rdparty-fcmae_in1k*	28.64	4.47	config	model
convnext-v2-base_3rdparty-fcmae_in1k*	88.72	15.38	config	model
convnext-v2-large_3rdparty-fcmae_in1k*	197.96	34.40	config	model
convnext-v2-huge_3rdparty-fcmae_in1k*	660.29	115.00	config	model

*Models with * are converted from the official repo.*

32.2.2 ImageNet-1k

模型	预训练	参数量(M)	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
convnext-v2-atto_fcmae-pre_3rdparty_in1k*	FCMAE	3.71	0.55	76.64	93.04	config	model
convnext-v2-femto_fcmae-pre_3rdparty_in1k*	FCMAE	5.23	0.78	78.48	93.98	config	model
convnext-v2-pico_fcmae-pre_3rdparty_in1k*	FCMAE	9.07	1.37	80.31	95.08	config	model
convnext-v2-nano_fcmae-pre_3rdparty_in1k*	FCMAE	15.62	2.45	81.86	95.75	config	model
convnext-v2-nano_fcmae-in21k-pre_3rdparty_in1k*	FCMAE + ImageNet 21k	15.62	2.45	82.04	96.16	config	model
convnext-v2-tiny_fcmae-pre_3rdparty_in1k*	FCMAE	28.64	4.47	82.94	96.29	config	model
convnext-v2-tiny_fcmae-in21k-pre_3rdparty_in1k*	FCMAE + ImageNet 21k	28.64	4.47	83.89	96.96	config	model
convnext-v2-nano_fcmae-in21k-pre_3rdparty_in1k-384px*	FCMAE + ImageNet 21k	15.62	7.21	83.36	96.75	config	model
convnext-v2-tiny_fcmae-in21k-pre_3rdparty_in1k-384px*	FCMAE + ImageNet 21k	28.64	13.14	85.09	97.63	config	model
convnext-v2-base_fcmae-pre_3rdparty_in1k*	FCMAE	88.72	15.38	84.87	97.08	config	model
convnext-v2-base_fcmae-in21k-pre_3rdparty_in1k*	FCMAE + ImageNet 21k	88.72	15.38	86.74	98.02	config	model
convnext-v2-large_fcmae-pre_3rdparty_in1k*	FCMAE	197.96	34.40	85.76	97.59	config	model
convnext-v2-large_fcmae-in21k-pre_3rdparty_in1k*	FCMAE + ImageNet 21k	197.96	34.40	87.26	98.24	config	model
convnext-v2-base_fcmae-in21k-pre_3rdparty_in1k-384px*	FCMAE + ImageNet 21k	88.72	45.21	87.63	98.42	config	model
convnext-v2-large_fcmae-in21k-pre_3rdparty_in1k-384px*	FCMAE + ImageNet 21k	197.96	101.10	88.18	98.52	config	model
convnext-v2-huge_fcmae-pre_3rdparty_in1k*	FCMAE	660.29	115.00	86.25	97.75	config	model
convnext-v2-huge_fcmae-in21k-pre_3rdparty_in1k-384px*	FCMAE + ImageNet 21k	660.29	337.96	88.68	98.73	config	model
convnext-v2-huge_fcmae-in21k-pre_3rdparty_in1k-512px*	FCMAE + ImageNet 21k	660.29	600.81	88.86	98.74	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

32.3 引用

```
@article{Woo2023ConvNeXtV2,
  title={ConvNeXt V2: Co-designing and Scaling ConvNets with Masked Autoencoders},
  author={Sanghyun Woo, Shoubhik Debnath, Ronghang Hu, Xinlei Chen, Zhuang Liu, In So
↪Kweon and Saining Xie},
  year={2023},
  journal={arXiv preprint arXiv:2301.00808},
}
```

CHAPTER 33

CSPNet

CSPNet: A New Backbone that can Enhance Learning Capability of CNN

33.1 摘要

Neural networks have enabled state-of-the-art approaches to achieve incredible results on computer vision tasks such as object detection. However, such success greatly relies on costly computation resources, which hinders people with cheap devices from appreciating the advanced technology. In this paper, we propose Cross Stage Partial Network (CSPNet) to mitigate the problem that previous works require heavy inference computations from the network architecture perspective. We attribute the problem to the duplicate gradient information within network optimization. The proposed networks respect the variability of the gradients by integrating feature maps from the beginning and the end of a network stage, which, in our experiments, reduces computations by 20% with equivalent or even superior accuracy on the ImageNet dataset, and significantly outperforms state-of-the-art approaches in terms of AP50 on the MS COCO object detection dataset. The CSPNet is easy to implement and general enough to cope with architectures based on ResNet, ResNeXt, and DenseNet. Source code is at this https URL.

33.2 结果和模型

33.2.1 ImageNet-1k

模型	预训练	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
CSPDarkNet50*	从头训练	27.64	5.04	80.05	95.07	config	model
CSPResNet50*	从头训练	21.62	3.48	79.55	94.68	config	model
CSPResNeXt50*	从头训练	20.57	3.11	79.96	94.96	config	model

*Models with * are converted from the timm repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

33.3 引用

```
@inproceedings{wang2020cspnet,
  title={CSPNet: A new backbone that can enhance learning capability of CNN},
  author={Wang, Chien-Yao and Liao, Hong-Yuan Mark and Wu, Yueh-Hua and Chen, Ping-
↪Yang and Hsieh, Jun-Wei and Yeh, I-Hau},
  booktitle={Proceedings of the IEEE/CVF conference on computer vision and pattern␣
↪recognition workshops},
  pages={390--391},
  year={2020}
}
```

CHAPTER 34

CSRA

Residual Attention: A Simple but Effective Method for Multi-Label Recognition

34.1 摘要

Multi-label image recognition is a challenging computer vision task of practical use. Progresses in this area, however, are often characterized by complicated methods, heavy computations, and lack of intuitive explanations. To effectively capture different spatial regions occupied by objects from different categories, we propose an embarrassingly simple module, named class-specific residual attention (CSRA). CSRA generates class-specific features for every category by proposing a simple spatial attention score, and then combines it with the class-agnostic average pooling feature. CSRA achieves state-of-the-art results on multilabel recognition, and at the same time is much simpler than them. Furthermore, with only 4 lines of code, CSRA also leads to consistent improvement across many diverse pretrained models and datasets without any extra training. CSRA is both easy to implement and light in computations, which also enjoys intuitive explanations and visualizations.

34.2 结果和模型

34.2.1 VOC2007

模型	预训练	参数量(M)	Flops(G)	mAP	OF1 (%)	CF1 (%)	配置文件	下载
Resnet101-CSRA	ImageNet-1k	23.55	4.12	94.98	90.80	89.16	config	model | log

34.3 引用

```
@misc{https://doi.org/10.48550/arxiv.2108.02456,
  doi = {10.48550/ARXIV.2108.02456},
  url = {https://arxiv.org/abs/2108.02456},
  author = {Zhu, Ke and Wu, Jianxin},
  keywords = {Computer Vision and Pattern Recognition (cs.CV), FOS: Computer and␣
↪information sciences, FOS: Computer and information sciences},
  title = {Residual Attention: A Simple but Effective Method for Multi-Label␣
↪Recognition},
  publisher = {arXiv},
  year = {2021},
  copyright = {arXiv.org perpetual, non-exclusive license}
}
```

CHAPTER 35

DaViT

DaViT: Dual Attention Vision Transformers

35.1 摘要

In this work, we introduce Dual Attention Vision Transformers (DaViT), a simple yet effective vision transformer architecture that is able to capture global context while maintaining computational efficiency. We propose approaching the problem from an orthogonal angle: exploiting self-attention mechanisms with both "spatial tokens" and "channel tokens". With spatial tokens, the spatial dimension defines the token scope, and the channel dimension defines the token feature dimension. With channel tokens, we have the inverse: the channel dimension defines the token scope, and the spatial dimension defines the token feature dimension. We further group tokens along the sequence direction for both spatial and channel tokens to maintain the linear complexity of the entire model. We show that these two self-attentions complement each other: (i) since each channel token contains an abstract representation of the entire image, the channel attention naturally captures global interactions and representations by taking all spatial positions into account when computing attention scores between channels; (ii) the spatial attention refines the local representations by performing fine-grained interactions across spatial locations, which in turn helps the global information modeling in channel attention. Extensive experiments show our DaViT achieves state-of-the-art performance on four different tasks with efficient computations. Without extra data, DaViT-Tiny, DaViT-Small, and DaViT-Base achieve 82.8%, 84.2%, and 84.6% top-1 accuracy on ImageNet-1K with 28.3M, 49.7M, and 87.9M parameters, respectively. When we further scale up DaViT with 1.5B weakly supervised image and text pairs, DaViT-Gaint reaches 90.4% top-1 accuracy on ImageNet-1K.

35.2 结果和模型

35.2.1 ImageNet-1k

模型	预训练	分辨率	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
DaViT-T*	从头训练	224x224	28.36	4.54	82.24	96.13	config	model
DaViT-S*	从头训练	224x224	49.74	8.79	83.61	96.75	config	model
DaViT-B*	从头训练	224x224	87.95	15.5	84.09	96.82	config	model

*Models with * are converted from the official repo. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

Note: Inference accuracy is a bit lower than paper result because of inference code for classification doesn't exist.

35.3 引用

```
@inproceedings{ding2022davit,
    title={DaViT: Dual Attention Vision Transformer},
    author={Ding, Mingyu and Xiao, Bin and Codella, Noel and Luo, Ping and Wang,␣
↪Jingdong and Yuan, Lu},
    booktitle={ECCV},
    year={2022},
}
```

CHAPTER 36

DeiT

Training data-efficient image transformers & distillation through attention

36.1 摘要

Recently, neural networks purely based on attention were shown to address image understanding tasks such as image classification. However, these visual transformers are pre-trained with hundreds of millions of images using an expensive infrastructure, thereby limiting their adoption. In this work, we produce a competitive convolution-free transformer by training on Imagenet only. We train them on a single computer in less than 3 days. Our reference vision transformer (86M parameters) achieves top-1 accuracy of 83.1% (single-crop evaluation) on ImageNet with no external data. More importantly, we introduce a teacher-student strategy specific to transformers. It relies on a distillation token ensuring that the student learns from the teacher through attention. We show the interest of this token-based distillation, especially when using a convnet as a teacher. This leads us to report results competitive with convnets for both Imagenet (where we obtain up to 85.2% accuracy) and when transferring to other tasks. We share our code and models.

36.2 结果和模型

36.2.1 ImageNet-1k

The teacher of the distilled version DeiT is RegNetY-16GF.

模型	预训练	参数量(M)	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
DeiT-tiny	从头训练	5.72	1.08	74.50	92.24	config	model | log
DeiT-tiny distilled*	从头训练	5.72	1.08	74.51	91.90	config	model
DeiT-small	从头训练	22.05	4.24	80.69	95.06	config	model | log
DeiT-small distilled*	从头训练	22.05	4.24	81.17	95.40	config	model
DeiT-base	从头训练	86.57	16.86	81.76	95.81	config	model | log
DeiT-base*	从头训练	86.57	16.86	81.79	95.59	config	model
DeiT-base distilled*	从头训练	86.57	16.86	83.33	96.49	config	model
DeiT-base 384px*	ImageNet-1k	86.86	49.37	83.04	96.31	config	model
DeiT-base distilled 384px*	ImageNet-1k	86.86	49.37	85.55	97.35	config	model

*Models with * are converted from the official repo. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

警告: MMClassification doesn't support training the distilled version DeiT. And we provide distilled version checkpoints for inference only.

36.3 引用

```
@InProceedings{pmlr-v139-touvron21a,
  title =     {Training data-efficient image transformers &amp; distillation through attention},
  author =    {Touvron, Hugo and Cord, Matthieu and Douze, Matthijs and Massa, Francisco and Sablayrolles, Alexandre and Jegou, Herve},
  booktitle = {International Conference on Machine Learning},
  pages =     {10347--10357},
  year =      {2021},
  volume =    {139},
  month =     {July}
}
```

CHAPTER 37

DeiT III: Revenge of the ViT

DeiT III: Revenge of the ViT

37.1 摘要

A Vision Transformer (ViT) is a simple neural architecture amenable to serve several computer vision tasks. It has limited built-in architectural priors, in contrast to more recent architectures that incorporate priors either about the input data or of specific tasks. Recent works show that ViTs benefit from self-supervised pre-training, in particular BerT-like pre-training like BeiT. In this paper, we revisit the supervised training of ViTs. Our procedure builds upon and simplifies a recipe introduced for training ResNet-50. It includes a new simple data-augmentation procedure with only 3 augmentations, closer to the practice in self-supervised learning. Our evaluations on Image classification (ImageNet-1k with and without pre-training on ImageNet-21k), transfer learning and semantic segmentation show that our procedure outperforms by a large margin previous fully supervised training recipes for ViT. It also reveals that the performance of our ViT trained with supervision is comparable to that of more recent architectures. Our results could serve as better baselines for recent self-supervised approaches demonstrated on ViT.

37.2 结果和模型

37.2.1 ImageNet-1k

模型	预训练	分辨率	参数量 (M)	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
DeiT3-S*	从头训练	224x224	22.06	4.61	81.35	95.31	config	model
DeiT3-S*	从头训练	384x384	22.21	15.52	83.43	96.68	config	model
DeiT3-S*	ImageNet-21k	224x224	22.06	4.61	83.06	96.77	config	model
DeiT3-S*	ImageNet-21k	384x384	22.21	15.52	84.84	97.48	config	model
DeiT3-M*	从头训练	224x224	38.85	8.00	82.99	96.22	config	model
DeiT3-M*	ImageNet-21k	224x224	38.85	8.00	84.56	97.19	config	model
DeiT3-B*	从头训练	224x224	86.59	17.58	83.80	96.55	config	model
DeiT3-B*	从头训练	384x384	86.88	55.54	85.08	97.25	config	model
DeiT3-B*	ImageNet-21k	224x224	86.59	17.58	85.70	97.75	config	model
DeiT3-B*	ImageNet-21k	384x384	86.88	55.54	86.73	98.11	config	model
DeiT3-L*	从头训练	224x224	304.37	61.60	84.87	97.01	config	model
DeiT3-L*	从头训练	384x384	304.76	191.21	85.82	97.60	config	model
DeiT3-L*	ImageNet-21k	224x224	304.37	61.60	86.97	98.24	config	model
DeiT3-L*	ImageNet-21k	384x384	304.76	191.21	87.73	98.51	config	model
DeiT3-H*	从头训练	224x224	632.13	167.40	85.21	97.36	config	model
DeiT3-H*	ImageNet-21k	224x224	632.13	167.40	87.19	98.26	config	model

*Models with * are converted from the official repo. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

37.3 引用

```
@article{Touvron2022DeiTIR,
  title={DeiT III: Revenge of the ViT},
  author={Hugo Touvron and Matthieu Cord and Herve Jegou},
  journal={arXiv preprint arXiv:2204.07118},
  year={2022},
}
```

CHAPTER 38

DenseNet

Densely Connected Convolutional Networks

38.1 摘要

Recent work has shown that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output. In this paper, we embrace this observation and introduce the Dense Convolutional Network (DenseNet), which connects each layer to every layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections - one between each layer and its subsequent layer - our network has L(L+1)/2 direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters. We evaluate our proposed architecture on four highly competitive object recognition benchmark tasks (CIFAR-10, CIFAR-100, SVHN, and ImageNet). DenseNets obtain significant improvements over the state-of-the-art on most of them, whilst requiring less computation to achieve high performance.

38.2 结果和模型

38.2.1 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
DenseNet121*	7.98	2.88	74.96	92.21	config	model
DenseNet169*	14.15	3.42	76.08	93.11	config	model
DenseNet201*	20.01	4.37	77.32	93.64	config	model
DenseNet161*	28.68	7.82	77.61	93.83	config	model

*Models with * are converted from pytorch, guided by original repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

38.3 引用

```
@misc{https://doi.org/10.48550/arxiv.1608.06993,
      doi = {10.48550/ARXIV.1608.06993},
      url = {https://arxiv.org/abs/1608.06993},
      author = {Huang, Gao and Liu, Zhuang and van der Maaten, Laurens and Weinberger,
↪ Kilian Q.},
      keywords = {Computer Vision and Pattern Recognition (cs.CV), Machine Learning␣
↪(cs.LG), FOS: Computer and information sciences, FOS: Computer and information␣
↪sciences},
      title = {Densely Connected Convolutional Networks},
      publisher = {arXiv},
      year = {2016},
      copyright = {arXiv.org perpetual, non-exclusive license}
}
```

CHAPTER 39

EdgeNeXt

EdgeNeXt: Efficiently Amalgamated CNN-Transformer Architecture for Mobile Vision Applications

39.1 摘要

In the pursuit of achieving ever-increasing accuracy, large and complex neural networks are usually developed. Such models demand high computational resources and therefore cannot be deployed on edge devices. It is of great interest to build resource-efficient general purpose networks due to their usefulness in several application areas. In this work, we strive to effectively combine the strengths of both CNN and Transformer models and propose a new efficient hybrid architecture EdgeNeXt. Specifically in EdgeNeXt, we introduce split depth-wise transpose attention (SDTA) encoder that splits input tensors into multiple channel groups and utilizes depth-wise convolution along with self-attention across channel dimensions to implicitly increase the receptive field and encode multi-scale features. Our extensive experiments on classification, detection and segmentation tasks, reveal the merits of the proposed approach, outperforming state-of-the-art methods with comparatively lower compute requirements. Our EdgeNeXt model with 1.3M parameters achieves 71.2% top-1 accuracy on ImageNet-1K, outperforming MobileViT with an absolute gain of 2.2% with 28% reduction in FLOPs. Further, our EdgeNeXt model with 5.6M parameters achieves 79.4% top-1 accuracy on ImageNet-1K.

39.2 结果和模型

39.2.1 ImageNet-1k

模型	预训练	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
EdgeNeXt-Base-usi*	从头训练	18.51	3.84	83.67	96.7	config	model
EdgeNeXt-Base*	从头训练	18.51	3.84	82.48	96.2	config	model
EdgeNeXt-Small-usi*	从头训练	5.59	1.26	81.06	95.34	config	model
EdgeNeXt-Small*	从头训练	5.59	1.26	79.41	94.53	config	model
EdgeNeXt-X-Small*	从头训练	2.34	0.538	74.86	92.31	config	model
EdgeNeXt-XX-Small*	从头训练	1.33	0.261	71.2	89.91	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

39.3 引用

```
@article{Maaz2022EdgeNeXt,
    title={EdgeNeXt: Efficiently Amalgamated CNN-Transformer Architecture for Mobile Vision Applications},
    author={Muhammad Maaz and Abdelrahman Shaker and Hisham Cholakkal and Salman Khan and Syed Waqas Zamir and Rao Muhammad Anwer and Fahad Shahbaz Khan},
    journal={2206.10589},
    year={2022}
}
```

CHAPTER 40

EfficientFormer

EfficientFormer: Vision Transformers at MobileNet Speed

40.1 摘要

Vision Transformers (ViT) have shown rapid progress in computer vision tasks, achieving promising results on various benchmarks. However, due to the massive number of parameters and model design, e.g., attention mechanism, ViT-based models are generally times slower than lightweight convolutional networks. Therefore, the deployment of ViT for real-time applications is particularly challenging, especially on resource-constrained hardware such as mobile devices. Recent efforts try to reduce the computation complexity of ViT through network architecture search or hybrid design with MobileNet block, yet the inference speed is still unsatisfactory. This leads to an important question: can transformers run as fast as MobileNet while obtaining high performance? To answer this, we first revisit the network architecture and operators used in ViT-based models and identify inefficient designs. Then we introduce a dimension-consistent pure transformer (without MobileNet blocks) as a design paradigm. Finally, we perform latency-driven slimming to get a series of final models dubbed EfficientFormer. Extensive experiments show the superiority of EfficientFormer in performance and speed on mobile devices. Our fastest model, EfficientFormer-L1, achieves 79.2% top-1 accuracy on ImageNet-1K with only 1.6 ms inference latency on iPhone 12 (compiled with CoreML), which runs as fast as MobileNetV2×1.4 (1.6 ms, 74.7% top-1), and our largest model, EfficientFormer-L7, obtains 83.3% accuracy with only 7.0 ms latency. Our work proves that properly designed transformers can reach extremely low latency on mobile devices while maintaining high performance.

40.2 结果和模型

40.2.1 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
EfficientFormer-l1*	12.19	1.30	80.46	94.99	config	model
EfficientFormer-l3*	31.41	3.93	82.45	96.18	config	model
EfficientFormer-l7*	82.23	10.16	83.40	96.60	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

40.3 引用

```
@misc{https://doi.org/10.48550/arxiv.2206.01191,
  doi = {10.48550/ARXIV.2206.01191},

  url = {https://arxiv.org/abs/2206.01191},

  author = {Li, Yanyu and Yuan, Geng and Wen, Yang and Hu, Eric and Evangelidis,␣
↪Georgios and Tulyakov, Sergey and Wang, Yanzhi and Ren, Jian},

  keywords = {Computer Vision and Pattern Recognition (cs.CV), FOS: Computer and␣
↪information sciences, FOS: Computer and information sciences},

  title = {EfficientFormer: Vision Transformers at MobileNet Speed},

  publisher = {arXiv},

  year = {2022},

  copyright = {Creative Commons Attribution 4.0 International}
}
```

CHAPTER 41

EfficientNet

Rethinking Model Scaling for Convolutional Neural Networks

41.1 简介

EfficientNets are a family of image classification models, which achieve state-of-the-art accuracy, yet being an order-of-magnitude smaller and faster than previous models.

EfficientNets are based on AutoML and Compound Scaling. In particular, we first use AutoML MNAS Mobile framework to develop a mobile-size baseline network, named as EfficientNet-B0; Then, we use the compound scaling method to scale up this baseline to obtain EfficientNet-B1 to B7.

41.2 摘要

41.3 结果和模型

41.3.1 ImageNet-1k

In the result table, AA means trained with AutoAugment pre-processing, more details can be found in the paper; AdvProp is a method to train with adversarial examples, more details can be found in the paper; RA means trained with RandAugment pre-processing, more details can be found in the paper; NoisyStudent means trained with extra JFT-300M

unlabeled data, more details can be found in the paper; L2-475 means the same L2 architecture with input image size 475.

Note: In MMClassification, we support training with AutoAugment, don't support AdvProp by now.

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
EfficientNet-B0*	5.29	0.42	76.74	93.17	config	model
EfficientNet-B0 (AA)*	5.29	0.42	77.26	93.41	config	model
EfficientNet-B0 (AA + AdvProp)*	5.29	0.42	77.53	93.61	config	model
EfficientNet-B0 (RA + NoisyStudent)*	5.29	0.42	77.63	94.00	config	model
EfficientNet-B1*	7.79	0.74	78.68	94.28	config	model
EfficientNet-B1 (AA)*	7.79	0.74	79.20	94.42	config	model
EfficientNet-B1 (AA + AdvProp)*	7.79	0.74	79.52	94.43	config	model
EfficientNet-B1 (RA + NoisyStudent)*	7.79	0.74	81.44	95.83	config	model
EfficientNet-B2*	9.11	1.07	79.64	94.80	config	model
EfficientNet-B2 (AA)*	9.11	1.07	80.21	94.96	config	model
EfficientNet-B2 (AA + AdvProp)*	9.11	1.07	80.45	95.07	config	model
EfficientNet-B2 (RA + NoisyStudent)*	9.11	1.07	82.47	96.23	config	model
EfficientNet-B3*	12.23	1.07	81.01	95.34	config	model
EfficientNet-B3 (AA)*	12.23	1.07	81.58	95.67	config	model
EfficientNet-B3 (AA + AdvProp)*	12.23	1.07	81.81	95.69	config	model
EfficientNet-B3 (RA + NoisyStudent)*	12.23	1.07	84.02	96.89	config	model
EfficientNet-B4*	19.34	1.95	82.57	96.09	config	model
EfficientNet-B4 (AA)*	19.34	1.95	82.95	96.26	config	model
EfficientNet-B4 (AA + AdvProp)*	19.34	1.95	83.25	96.44	config	model
EfficientNet-B4 (RA + NoisyStudent)*	19.34	1.95	85.25	97.52	config	model
EfficientNet-B5*	30.39	10.1	83.18	96.47	config	model
EfficientNet-B5 (AA)*	30.39	10.1	83.82	96.76	config	model
EfficientNet-B5 (AA + AdvProp)*	30.39	10.1	84.21	96.98	config	model
EfficientNet-B5 (RA + NoisyStudent)*	30.39	10.1	86.08	97.75	config	model
EfficientNet-B6 (AA)*	43.04	20.0	84.05	96.82	config	model
EfficientNet-B6 (AA + AdvProp)*	43.04	20.0	84.74	97.14	config	model
EfficientNet-B6 (RA + NoisyStudent)*	43.04	20.0	86.47	97.87	config	model
EfficientNet-B7 (AA)*	66.35	39.3	84.38	96.88	config	model
EfficientNet-B7 (AA + AdvProp)*	66.35	39.3	85.14	97.23	config	model
EfficientNet-B7 (RA + NoisyStudent)*	66.35	65.0	86.83	98.08	config	model
EfficientNet-B8 (AA + AdvProp)*	87.41	65.0	85.38	97.28	config	model
EfficientNet-L2-475 (RA + NoisyStudent)*	480.30	174.20	88.18	98.55	config	model
EfficientNet-L2 (RA + NoisyStudent)*	480.30	484.98	88.33	98.65	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure*

these config files' training accuracy and welcome you to contribute your reproduction results.

41.4 使用方式

推理图片

```
>>> import torch
>>> from mmcls.apis import init_model, inference_model
>>>
>>> model = init_model('configs/efficientnet/efficientnet-b0_8xb32_in1k.py', "https://
↪download.openmmlab.com/mmclassification/v0/efficientnet/efficientnet-b0_3rdparty_
↪8xb32_in1k_20220119-a7e2a0b1.pth")
>>> predict = inference_model(model, 'demo/demo.JPEG')
>>> print(predict['pred_class'])
sea snake
>>> print(predict['pred_score'])
0.6968820691108704
```

调用模型

```
>>> import torch
>>> from mmcls.apis import init_model
>>>
>>> model = init_model('configs/efficientnet/efficientnet-b0_8xb32_in1k.py', "https://
↪download.openmmlab.com/mmclassification/v0/efficientnet/efficientnet-b0_3rdparty_
↪8xb32_in1k_20220119-a7e2a0b1.pth")
>>> inputs = torch.rand(1, 3, 224, 224).to(model.data_preprocessor.device)
>>> # To get classification scores.
>>> out = model(inputs)
>>> print(out.shape)
torch.Size([1, 1000])
>>> # To extract features.
>>> outs = model.extract_feat(inputs)
>>> print(outs[0].shape)
torch.Size([1, 1280])
```

训练/测试

将 ImageNet 数据集放置在 `data/imagenet` 目录下，或者根据 docs 准备其他数据集。

训练：

```
python tools/train.py configs/efficientnet/efficientnet-b0_8xb32_in1k.py
```

测试：

```
python tools/test.py configs/efficientnet/efficientnet-b0_8xb32_in1k.py https://
↪download.openmmlab.com/mmclassification/v0/efficientnet/efficientnet-b0_3rdparty_
↪8xb32_in1k_20220119-a7e2a0b1.pth
```

For more configurable parameters, please refer to the API.

41.5 引用

```
@inproceedings{tan2019efficientnet,
  title={Efficientnet: Rethinking model scaling for convolutional neural networks},
  author={Tan, Mingxing and Le, Quoc},
  booktitle={International Conference on Machine Learning},
  pages={6105--6114},
  year={2019},
  organization={PMLR}
}
```

CHAPTER 42

EfficientNetV2

EfficientNetV2: Smaller Models and Faster Training

42.1 摘要

This paper introduces EfficientNetV2, a new family of convolutional networks that have faster training speed and better parameter efficiency than previous models. To develop this family of models, we use a combination of training-aware neural architecture search and scaling, to jointly optimize training speed and parameter efficiency. The models were searched from the search space enriched with new ops such as Fused-MBConv. Our experiments show that EfficientNetV2 models train much faster than state-of-the-art models while being up to 6.8x smaller. Our training can be further sped up by progressively increasing the image size during training, but it often causes a drop in accuracy. To compensate for this accuracy drop, we propose to adaptively adjust regularization (e.g., dropout and data augmentation) as well, such that we can achieve both fast training and good accuracy. With progressive learning, our EfficientNetV2 significantly outperforms previous models on ImageNet and CIFAR/Cars/Flowers datasets. By pretraining on the same ImageNet21k, our EfficientNetV2 achieves 87.3% top-1 accuracy on ImageNet ILSVRC2012, outperforming the recent ViT by 2.0% accuracy while training 5x-11x faster using the same computing resources. Code will be available at https://github.com/google/automl/tree/master/efficientnetv2.

42.2 使用方式

推理图片

```
>>> import torch
>>> from mmcls.apis import init_model, inference_model
>>>
>>> model = init_model('configs/efficientnet_v2/efficientnetv2-b0_8xb32_in1k.py',
↪"https://download.openmmlab.com/mmclassification/v0/efficientnetv2/efficientnetv2-
↪b0_8xb32_in1k_20221219-9689f21f.pth")
>>> predict = inference_model(model, 'demo/demo.JPEG')
>>> print(predict['pred_class'])
sea snake
>>> print(predict['pred_score'])
0.3147328197956085
```

调用模型

```
>>> import torch
>>> from mmcls import get_model
>>>
>>> model = get_model("efficientnetv2-b0_3rdparty_in1k", pretrained=True)
>>> model.eval()
>>> inputs = torch.rand(1, 3, 224, 224).to(model.data_preprocessor.device)
>>> # To get classification scores.
>>> out = model(inputs)
>>> print(out.shape)
torch.Size([1, 1000])
>>> # To extract features.
>>> outs = model.extract_feat(inputs)
>>> print(outs[0].shape)
torch.Size([1, 1280])
```

训练/测试

将 ImageNet 数据集放置在 `data/imagenet` 目录下，或者根据 docs 准备其他数据集。

训练：

```
python tools/train.py configs/efficientnet_v2/efficientnetv2-b0_8xb32_in1k.py
```

测试：

```
python tools/test.py configs/efficientnet_v2/efficientnetv2-b0_8xb32_in1k.py https://
↪download.openmmlab.com/mmclassification/v0/efficientnetv2/efficientnetv2-b0_8xb32_
↪in1k_20221219-9689f21f.pth
```

For more configurable parameters, please refer to the API.

42.3 结果和模型

42.3.1 ImageNet-1k

模型	预训练	参数量 (M)	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
EfficientNetV2-b0* (efficientnetv2-b0_3rdparty_in1k)	从头训练	7.14	0.92	78.52	94.44	config	model
EfficientNetV2-b1* (efficientnetv2-b1_3rdparty_in1k)	从头训练	8.14	1.44	79.80	94.89	config	model
EfficientNetV2-b2* (efficientnetv2-b2_3rdparty_in1k)	从头训练	10.10	1.99	80.63	95.30	config	model
EfficientNetV2-b3* (efficientnetv2-b3_3rdparty_in1k)	从头训练	14.36	3.50	82.03	95.88	config	model
EfficientNetV2-s* (efficientnetv2-s_3rdparty_in1k)	从头训练	21.46	9.72	83.82	96.67	config	model
EfficientNetV2-m* (efficientnetv2-m_3rdparty_in1k)	从头训练	54.14	26.88	85.01	97.26	config	model
EfficientNetV2-l* (efficientnetv2-l_3rdparty_in1k)	从头训练	118.52	60.14	85.43	97.31	config	model
EfficientNetV2-s* (efficientnetv2-s_in21k-pre_3rdparty_in1k)	ImageNet 21k	21.46	9.72	84.29	97.26	config	model
EfficientNetV2-m* (efficientnetv2-m_in21k-pre_3rdparty_in1k)	ImageNet 21k	54.14	26.88	85.47	97.76	config	model
EfficientNetV2-l* (efficientnetv2-l_in21k-pre_3rdparty_in1k)	ImageNet 21k	118.52	60.14	86.31	97.99	config	model
EfficientNetV2-xl* (efficientnetv2-xl_in21k-pre_3rdparty_in1k)	ImageNet 21k	208.12	98.34	86.39	97.83	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

42.3.2 Pre-trained Models In ImageNet-21K

The pre-trained models are only used to fine-tune, and therefore cannot be trained and don't have evaluation results.

模型	参数量 (M)	Flops(G)	配置文件	下载
EfficientNetV2-s* (efficientnetv2-s_3rdparty_in21k)	21.46	9.72	config	model
EfficientNetV2-m* (efficientnetv2-m_3rdparty_in21k)	54.14	26.88	config	model
EfficientNetV2-l* (efficientnetv2-l_3rdparty_in21k)	118.52	60.14	config	model
EfficientNetV2-xl* (efficientnetv2-xl_3rdparty_in21k)	208.12	98.34	config	model

*Models with * are converted from the official repo.*

42.4 引用

```
@inproceedings{tan2021efficientnetv2,
  title={Efficientnetv2: Smaller models and faster training},
  author={Tan, Mingxing and Le, Quoc},
  booktitle={International Conference on Machine Learning},
  pages={10096--10106},
  year={2021},
  organization={PMLR}
}
```

CHAPTER 43

EVA

EVA: Exploring the Limits of Masked Visual Representation Learning at Scale

43.1 摘要

We launch EVA, a vision-centric foundation model to explore the limits of visual representation at scale using only publicly accessible data. EVA is a vanilla ViT pre-trained to reconstruct the masked out image-text aligned vision features conditioned on visible image patches. Via this pretext task, we can efficiently scale up EVA to one billion parameters, and sets new records on a broad range of representative vision downstream tasks, such as image recognition, video action recognition, object detection, instance segmentation and semantic segmentation without heavy supervised training. Moreover, we observe quantitative changes in scaling EVA result in qualitative changes in transfer learning performance that are not present in other models. For instance, EVA takes a great leap in the challenging large vocabulary instance segmentation task: our model achieves almost the same state-of-the-art performance on LVISv1.0 dataset with over a thousand categories and COCO dataset with only eighty categories. Beyond a pure vision encoder, EVA can also serve as a vision-centric, multi-modal pivot to connect images and text. We find initializing the vision tower of a giant CLIP from EVA can greatly stabilize the training and outperform the training from scratch counterpart with much fewer samples and less compute, providing a new direction for scaling up and accelerating the costly training of multi-modal foundation models.

43.2 结果和模型

43.2.1 merged-30M

The pre-trained models on merged-30M are used to fine-tune, and therefore don’t have evaluation results.

模型	patch size	分辨率	下载
EVA-G (`eva-g-p14_3rdparty_30m`)*	14	224x224	model
EVA-G (`eva-g-p16_3rdparty_30m`)*	14 to 16	224x224	model

*Models with * are converted from the official repo.*

43.2.2 ImageNet-21k

The pre-trained models on ImageNet-21k are used to fine-tune, and therefore don’t have evaluation results.

模型	预训练	分辨率	下载
EVA-G (`eva-g-p14_30m-pre_3rdparty_in21k`)*	merged-30M	224x224	model
EVA-L (`eva-l-p14_3rdparty-mim_in21k`)*	From scratch with MIM	224x224	model
EVA-L (`eva-l-p14_mim-pre_3rdparty_in21k`)*	MIM	224x224	model

*Models with * are converted from the official repo.*

43.2.3 ImageNet-1k

模型	预训练	分辨率	参数量 (M)	Flops (G)	Top-1 (%)	Top-5 (%)	配置文件	下载
EVA-G (eva-g-p14_30m-in21k-pre_3rdparty_in1k-336px)*	merged-30M & ImageNet-21k	336x336	1013.01	620.64	89.61	98.93	config	model
EVA-G (eva-g-p14_30m-in21k-pre_3rdparty_in1k-560px)*	merged-30M & ImageNet-21k	560x560	1014.45	1906.76	89.71	98.96	config	model
EVA-L (eva-l-p14_mim-pre_3rdparty_in1k-336px)*	MIM	336x336	304.53	191.10	88.66	98.75	config	model
EVA-L (eva-l-p14_mim-in21k-pre_3rdparty_in1k-336px)*	MIM & ImageNet-21k	336x336	304.53	191.10	89.17	98.86	config	model
EVA-L (eva-l-p14_mim-pre_3rdparty_in1k-196px)*	MIM	196x196	304.14	61.57	87.94	98.50	config	model
EVA-L (eva-l-p14_mim-in21k-pre_3rdparty_in1k-196px)*	MIM & ImageNet-21k	196x196	304.14	61.57	88.58	98.65	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference.*

43.3 引用

```
@article{EVA,
  title={EVA: Exploring the Limits of Masked Visual Representation Learning at Scale},
  author={Fang, Yuxin and Wang, Wen and Xie, Binhui and Sun, Quan and Wu, Ledell and
↪Wang, Xinggang and Huang, Tiejun and Wang, Xinlong and Cao, Yue},
  journal={arXiv preprint arXiv:2211.07636},
  year={2022}
}
```

CHAPTER 44

HorNet

HorNet: Efficient High-Order Spatial Interactions with Recursive Gated Convolutions

44.1 摘要

Recent progress in vision Transformers exhibits great success in various tasks driven by the new spatial modeling mechanism based on dot-product self-attention. In this paper, we show that the key ingredients behind the vision Transformers, namely input-adaptive, long-range and high-order spatial interactions, can also be efficiently implemented with a convolution-based framework. We present the Recursive Gated Convolution (g nConv) that performs high-order spatial interactions with gated convolutions and recursive designs. The new operation is highly flexible and customizable, which is compatible with various variants of convolution and extends the two-order interactions in self-attention to arbitrary orders without introducing significant extra computation. g nConv can serve as a plug-and-play module to improve various vision Transformers and convolution-based models. Based on the operation, we construct a new family of generic vision backbones named HorNet. Extensive experiments on ImageNet classification, COCO object detection and ADE20K semantic segmentation show HorNet outperform Swin Transformers and ConvNeXt by a significant margin with similar overall architecture and training configurations. HorNet also shows favorable scalability to more training data and a larger model size. Apart from the effectiveness in visual encoders, we also show g nConv can be applied to task-specific decoders and consistently improve dense prediction performance with less computation. Our results demonstrate that g nConv can be a new basic module for visual modeling that effectively combines the merits of both vision Transformers and CNNs. Code is available at https://github.com/raoyongming/HorNet.

44.2 结果和模型

44.2.1 ImageNet-1k

模型	预训练	分辨率	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
HorNet-T*	从头训练	224x224	22.41	3.98	82.84	96.24	config	model
HorNet-T-GF*	从头训练	224x224	22.99	3.9	82.98	96.38	config	model
HorNet-S*	从头训练	224x224	49.53	8.83	83.79	96.75	config	model
HorNet-S-GF*	从头训练	224x224	50.4	8.71	83.98	96.77	config	model
HorNet-B*	从头训练	224x224	87.26	15.59	84.24	96.94	config	model
HorNet-B-GF*	从头训练	224x224	88.42	15.42	84.32	96.95	config	model

*Models with * are converted from the official repo. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.

44.2.2 Pre-trained Models

The pre-trained models on ImageNet-21k are used to fine-tune on the downstream tasks.

模型	预训练	分辨率	参数量（M）	Flops(G)	下载
HorNet-L*	ImageNet-21k	224x224	194.54	34.83	model
HorNet-L-GF*	ImageNet-21k	224x224	196.29	34.58	model
HorNet-L-GF384*	ImageNet-21k	384x384	201.23	101.63	model

*Models with * are converted from the official repo.

44.3 引用

```
@article{rao2022hornet,
  title={HorNet: Efficient High-Order Spatial Interactions with Recursive Gated Convolutions},
  author={Rao, Yongming and Zhao, Wenliang and Tang, Yansong and Zhou, Jie and Lim, Ser-Lam and Lu, Jiwen},
  journal={arXiv preprint arXiv:2207.14284},
  year={2022}
}
```

CHAPTER 45

HRNet

Deep High-Resolution Representation Learning for Visual Recognition

45.1 摘要

High-resolution representations are essential for position-sensitive vision problems, such as human pose estimation, semantic segmentation, and object detection. Existing state-of-the-art frameworks first encode the input image as a low-resolution representation through a subnetwork that is formed by connecting high-to-low resolution convolutions *in series* (e.g., ResNet, VGGNet), and then recover the high-resolution representation from the encoded low-resolution representation. Instead, our proposed network, named as High-Resolution Network (HRNet), maintains high-resolution representations through the whole process. There are two key characteristics: (i) Connect the high-to-low resolution convolution streams *in parallel*; (ii) Repeatedly exchange the information across resolutions. The benefit is that the resulting representation is semantically richer and spatially more precise. We show the superiority of the proposed HRNet in a wide range of applications, including human pose estimation, semantic segmentation, and object detection, suggesting that the HRNet is a stronger backbone for computer vision problems.

45.2 结果和模型

45.3 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
HRNet-W18*	21.30	4.33	76.75	93.44	config	model
HRNet-W30*	37.71	8.17	78.19	94.22	config	model
HRNet-W32*	41.23	8.99	78.44	94.19	config	model
HRNet-W40*	57.55	12.77	78.94	94.47	config	model
HRNet-W44*	67.06	14.96	78.88	94.37	config	model
HRNet-W48*	77.47	17.36	79.32	94.52	config	model
HRNet-W64*	128.06	29.00	79.46	94.65	config	model
HRNet-W18 (ssld)*	21.30	4.33	81.06	95.70	config	model
HRNet-W48 (ssld)*	77.47	17.36	83.63	96.79	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

45.4 引用

```
@article{WangSCJDZLMTWLX19,
  title={Deep High-Resolution Representation Learning for Visual Recognition},
  author={Jingdong Wang and Ke Sun and Tianheng Cheng and
          Borui Jiang and Chaorui Deng and Yang Zhao and Dong Liu and Yadong Mu and
          Mingkui Tan and Xinggang Wang and Wenyu Liu and Bin Xiao},
  journal   = {TPAMI}
  year={2019}
}
```

CHAPTER 46

Inception V3

Rethinking the Inception Architecture for Computer Vision

46.1 摘要

Convolutional networks are at the core of most state-of-the-art computer vision solutions for a wide variety of tasks. Since 2014 very deep convolutional networks started to become mainstream, yielding substantial gains in various benchmarks. Although increased model size and computational cost tend to translate to immediate quality gains for most tasks (as long as enough labeled data is provided for training), computational efficiency and low parameter count are still enabling factors for various use cases such as mobile vision and big-data scenarios. Here we explore ways to scale up networks in ways that aim at utilizing the added computation as efficiently as possible by suitably factorized convolutions and aggressive regularization. We benchmark our methods on the ILSVRC 2012 classification challenge validation set demonstrate substantial gains over the state of the art: 21.2% top-1 and 5.6% top-5 error for single frame evaluation using a network with a computational cost of 5 billion multiply-adds per inference and with using less than 25 million parameters. With an ensemble of 4 models and multi-crop evaluation, we report 3.5% top-5 error on the validation set (3.6% error on the test set) and 17.3% top-1 error on the validation set.

46.2 结果和模型

46.2.1 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
Inception V3*	23.83	5.75	77.57	93.58	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

46.3 引用

```
@inproceedings{szegedy2016rethinking,
  title={Rethinking the inception architecture for computer vision},
  author={Szegedy, Christian and Vanhoucke, Vincent and Ioffe, Sergey and Shlens, Jon
↪and Wojna, Zbigniew},
  booktitle={Proceedings of the IEEE conference on computer vision and pattern
↪recognition},
  pages={2818--2826},
  year={2016}
}
```

CHAPTER 47

LeViT

LeViT: a Vision Transformer in ConvNet’s Clothing for Faster Inference

47.1 摘要

We design a family of image classification architectures that optimize the trade-off between accuracy and efficiency in a high-speed regime. Our work exploits recent findings in attention-based architectures, which are competitive on highly parallel processing hardware. We revisit principles from the extensive literature on convolutional neural networks to apply them to transformers, in particular activation maps with decreasing resolutions. We also introduce the attention bias, a new way to integrate positional information in vision transformers. As a result, we propose LeVIT: a hybrid neural network for fast inference image classification. We consider different measures of efficiency on different hardware platforms, so as to best reflect a wide range of application scenarios. Our extensive experiments empirically validate our technical choices and show they are suitable to most architectures. Overall, LeViT significantly outperforms existing convnets and vision transformers with respect to the speed/accuracy tradeoff. For example, at 80% ImageNet top-1 accuracy, LeViT is 5 times faster than EfficientNet on CPU.

47.2 结果和模型

47.2.1 ImageNet-1k

模型	预训练	参数量(M)	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
levit-128s_3rdparty_in1k*	从头训练	7.39	0.31	76.51	92.90	config | deploy	model
levit-128_3rdparty_in1k*	从头训练	8.83	0.41	78.58	93.95	config | deploy	model
levit-192_3rdparty_in1k*	从头训练	10.56	0.67	79.86	94.75	config | deploy	model
levit-256_3rdparty_in1k*	从头训练	18.38	1.14	81.59	95.46	config | deploy	model
levit-384_3rdparty_in1k*	从头训练	38.36	2.37	82.59	95.95	config | deploy	model

*Models with * are converted from the official repo. The config files of these models are only for inference. All these models are trained by distillation on RegNet, and MMClassification doesn't support distillation by now. See MMRazor for model distillation.*

47.3 引用

```
@InProceedings{Graham_2021_ICCV,
    author    = {Graham, Benjamin and El-Nouby, Alaaeldin and Touvron, Hugo and Stock,
↪ Pierre and Joulin, Armand and Jegou, Herve and Douze, Matthijs},
    title     = {LeViT: A Vision Transformer in ConvNet's Clothing for Faster␣
↪Inference},
    booktitle = {Proceedings of the IEEE/CVF International Conference on Computer␣
↪Vision (ICCV)},
    month     = {October},
    year      = {2021},
    pages     = {12259-12269}
}
```

CHAPTER 48

MixMIM

MixMIM: Mixed and Masked Image Modeling for Efficient Visual Representation Learning

48.1 摘要

In this study, we propose Mixed and Masked Image Modeling (MixMIM), a simple but efficient MIM method that is applicable to various hierarchical Vision Transformers. Existing MIM methods replace a random subset of input tokens with a special [MASK] symbol and aim at reconstructing original image tokens from the corrupted image. However, we find that using the [MASK] symbol greatly slows down the training and causes training-finetuning inconsistency, due to the large masking ratio (e.g., 40% in BEiT). In contrast, we replace the masked tokens of one image with visible tokens of another image, i.e., creating a mixed image. We then conduct dual reconstruction to reconstruct the original two images from the mixed input, which significantly improves efficiency. While MixMIM can be applied to various architectures, this paper explores a simpler but stronger hierarchical Transformer, and scales with MixMIM-B, -L, and -H. Empirical results demonstrate that MixMIM can learn high-quality visual representations efficiently. Notably, MixMIM-B with 88M parameters achieves 85.1% top-1 accuracy on ImageNet-1K by pretraining for 600 epochs, setting a new record for neural networks with comparable model sizes (e.g., ViT-B) among MIM methods. Besides, its transferring performances on the other 6 datasets show MixMIM has better FLOPs / performance tradeoff than previous MIM methods

48.2 使用方式

48.2.1 Inference

推理图片

```
>>> import torch
>>> import mmcls
>>> model = mmcls.get_model('mixmim-base_3rdparty_in1k', pretrained=True)
>>> predict = mmcls.inference_model(model, 'demo/demo.JPEG')
>>> print(predict['pred_class'])
sea snake
>>> print(predict['pred_score'])
0.865431010723114
```

调用模型

```
>>> import torch
>>> import mmcls
>>>
>>> model = mmcls.get_model('mixmim-base_3rdparty_in1k', pretrained=True)
>>> inputs = torch.rand(1, 3, 224, 224)
>>> # To get classification scores.
>>> out = model(inputs)
>>> print(out.shape)
torch.Size([1, 1000])
>>> # To extract features.
>>> outs = model.extract_feat(inputs)
>>> print(outs[0].shape)
torch.Size([1, 1024])
```

48.3 Models

模型	参数量 (M)	Pretrain Epochs	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
mixmim-base_3rdparty_in1k*	88	300	16.3	84.6	97.0	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference.*

For MixMIM self-supervised learning algorithm, welcome to MMSelfSup page to get more information.

48.4 引用

```
@article{MixMIM2022,
  author  = {Jihao Liu, Xin Huang, Yu Liu, Hongsheng Li},
  journal = {arXiv:2205.13137},
  title   = {MixMIM: Mixed and Masked Image Modeling for Efficient Visual␣
↪Representation Learning},
  year    = {2022},
}
```

CHAPTER 49

Mlp-Mixer

MLP-Mixer: An all-MLP Architecture for Vision

49.1 摘要

Convolutional Neural Networks (CNNs) are the go-to model for computer vision. Recently, attention-based networks, such as the Vision Transformer, have also become popular. In this paper we show that while convolutions and attention are both sufficient for good performance, neither of them are necessary. We present MLP-Mixer, an architecture based exclusively on multi-layer perceptrons (MLPs). MLP-Mixer contains two types of layers: one with MLPs applied independently to image patches (i.e. "mixing" the per-location features), and one with MLPs applied across patches (i.e. "mixing" spatial information). When trained on large datasets, or with modern regularization schemes, MLP-Mixer attains competitive scores on image classification benchmarks, with pre-training and inference cost comparable to state-of-the-art models. We hope that these results spark further research beyond the realms of well established CNNs and Transformers.

49.2 结果和模型

49.2.1 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
Mixer-B/16*	59.88	12.61	76.68	92.25	config	model
Mixer-L/16*	208.2	44.57	72.34	88.02	config	model

*Models with * are converted from timm. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

49.3 引用

```
@misc{tolstikhin2021mlpmixer,
      title={MLP-Mixer: An all-MLP Architecture for Vision},
      author={Ilya Tolstikhin and Neil Houlsby and Alexander Kolesnikov and Lucas␣
↪Beyer and Xiaohua Zhai and Thomas Unterthiner and Jessica Yung and Andreas Steiner␣
↪and Daniel Keysers and Jakob Uszkoreit and Mario Lucic and Alexey Dosovitskiy},
      year={2021},
      eprint={2105.01601},
      archivePrefix={arXiv},
      primaryClass={cs.CV}
}
```

CHAPTER 50

MobileNet V2

MobileNetV2: Inverted Residuals and Linear Bottlenecks

50.1 简介

MobileNet V2 is initially described in the paper, which improves the state of the art performance of mobile models on multiple tasks. MobileNetV2 is an improvement on V1. Its new ideas include Linear Bottleneck and Inverted Residuals, and is based on an inverted residual structure where the input and output of the residual block are thin bottleneck layers. The intermediate expansion layer uses lightweight depthwise convolutions to filter features as a source of non-linearity. The author of MobileNet V2 measure its performance on Imagenet classification, COCO object detection, and VOC image segmentation.

50.2 摘要

The MobileNetV2 architecture is based on an inverted residual structure where the input and output of the residual block are thin bottleneck layers opposite to traditional residual models which use expanded representations in the input an MobileNetV2 uses lightweight depthwise convolutions to filter features in the intermediate expansion layer. Additionally, we find that it is important to remove non-linearities in the narrow layers in order to maintain representational power. We demonstrate that this improves performance and provide an intuition that led to this design. Finally, our approach allows decoupling of the input/output domains from the expressiveness of the transformation, which provides a convenient framework for further analysis. We measure our performance on Imagenet classification, COCO object detection, VOC

image segmentation. We evaluate the trade-offs between accuracy, and number of operations measured by multiply-adds (MAdd), as well as the number of parameters.

50.3 使用方式

推理图片

```
>>> import torch
>>> from mmcls.apis import init_model, inference_model
>>>
>>> model = init_model('configs/mobilenet_v2/mobilenet-v2_8xb32_in1k.py', 'https://
↪download.openmmlab.com/mmclassification/v0/mobilenet_v2/mobilenet_v2_batch256_
↪imagenet_20200708-3b2dc3af.pth')
>>> predict = inference_model(model, 'demo/demo.JPEG')
>>> print(predict['pred_class'])
leatherback turtle, leatherback, leathery turtle, Dermochelys coriacea
>>> print(predict['pred_score'])
0.6252720952033997
```

调用模型

```
>>> import torch
>>> from mmcls.apis import init_model
>>>
>>> model = init_model('configs/mobilenet_v2/mobilenet-v2_8xb32_in1k.py', 'https://
↪download.openmmlab.com/mmclassification/v0/mobilenet_v2/mobilenet_v2_batch256_
↪imagenet_20200708-3b2dc3af.pth')
>>> inputs = torch.rand(1, 3, 224, 224).to(model.data_preprocessor.device)
>>> # To get classification scores.
>>> out = model(inputs)
>>> print(out.shape)
torch.Size([1, 1000])
>>> # To extract features.
>>> outs = model.extract_feat(inputs)
>>> print(outs[0].shape)
torch.Size([1, 1280])
```

训练/测试

将 ImageNet 数据集放置在 `data/imagenet` 目录下，或者根据 docs 准备其他数据集。

训练：

```
python tools/train.py configs/mobilenet_v2/mobilenet-v2_8xb32_in1k.py
```

测试：

```
python tools/test.py configs/mobilenet_v2/mobilenet-v2_8xb32_in1k.py https://download.
↪openmmlab.com/mmclassification/v0/mobilenet_v2/mobilenet_v2_batch256_imagenet_
↪20200708-3b2dc3af.pth
```

For more configurable parameters, please refer to the API.

50.4 结果和模型

50.4.1 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
MobileNet V2	3.5	0.319	71.86	90.42	config	model | log

50.5 引用

```
@INPROCEEDINGS{8578572,
  author={M. {Sandler} and A. {Howard} and M. {Zhu} and A. {Zhmoginov} and L. {Chen}},
  booktitle={2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition},
  title={MobileNetV2: Inverted Residuals and Linear Bottlenecks},
  year={2018},
  volume={},
  number={},
  pages={4510-4520},
  doi={10.1109/CVPR.2018.00474}}
}
```

CHAPTER 51

MobileNet V3

Searching for MobileNetV3

51.1 简介

MobileNet V3 is initially described in the paper. MobileNetV3 parameters are obtained by NAS (network architecture search) search, and some practical results of V1 and V2 are inherited, and the attention mechanism of SE channel is attracted, which can be considered as a masterpiece. The author create two new MobileNet models for release: MobileNetV3-Large and MobileNetV3-Small which are targeted for high and low resource use cases. These models are then adapted and applied to the tasks of object detection and semantic segmentation. The author of MobileNet V3 measure its performance on Imagenet classification, COCO object detection, and Cityscapes segmentation.

51.2 摘要

51.3 使用方式

推理图片

```
>>> import torch
>>> from mmcls.apis import init_model, inference_model
>>>
```

(下页继续)

(续上页)

```
>>> model = init_model('configs/mobilenet_v3/mobilenet-v3-small-050_8xb128_in1k.py',
↪'https://download.openmmlab.com/mmclassification/v0/mobilenet_v3/mobilenet-v3-small-
↪050_3rdparty_in1k_20221114-e0b86be1.pth')
>>> predict = inference_model(model, 'demo/demo.JPEG')
>>> print(predict['pred_class'])
print(predict['pred_class'])
>>> print(predict['pred_score'])
print(predict['pred_score'])
```

调用模型

```
>>> import torch
>>> from mmcls.apis import init_model
>>>
>>> model = init_model('configs/mobilenet_v3/mobilenet-v3-small-050_8xb128_in1k.py',
↪'https://download.openmmlab.com/mmclassification/v0/mobilenet_v3/mobilenet-v3-small-
↪050_3rdparty_in1k_20221114-e0b86be1.pth')
>>> inputs = torch.rand(1, 3, 224, 224).to(model.data_preprocessor.device)
>>> # To get classification scores.
>>> out = model(inputs)
>>> print(out.shape)
torch.Size([1, 1000])
>>> # To extract features.
>>> outs = model.extract_feat(inputs)
>>> print(outs[0].shape)
torch.Size([1, 288])
```

训练/测试

将 ImageNet 数据集放置在 `data/imagenet` 目录下，或者根据 docs 准备其他数据集。

训练：

```
python tools/train.py configs/mobilenet_v3/mobilenet-v3-small_8xb128_in1k.py
```

测试：

```
python tools/test.py configs/mobilenet_v3/mobilenet-v3-small_8xb128_in1k.py https://
↪download.openmmlab.com/mmclassification/v0/mobilenet_v3/mobilenet-v3-small_8xb128_
↪in1k_20221114-bd1bfcde.pth
```

For more configurable parameters, please refer to the API.

51.4 结果和模型

51.4.1 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
MobileNetV3-Small-050**	1.59	0.02	57.91	80.19	config	model
MobileNetV3-Small-075**	2.04	0.04	65.23	85.44	config	model
MobileNetV3-Small	2.54	0.06	66.68	86.74	config	model | log
MobileNetV3-Small*	2.54	0.06	67.66	87.41	config	model
MobileNetV3-Large	5.48	0.23	73.49	91.31	config	model | log
MobileNetV3-Large*	5.48	0.23	74.04	91.34	config	model

We cannot reproduce the performances provided by TorchVision with the training script, and the accuracy results we got are 65.5±0.5% and 73.39% for small and large model respectively. Here we provide checkpoints trained by MMClassification that outperform the aforementioned results, and the original checkpoints provided by TorchVision.

*Models with * are converted from torchvision. Models with ** are converted from timm. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

51.5 引用

```
@inproceedings{Howard_2019_ICCV,
    author = {Howard, Andrew and Sandler, Mark and Chu, Grace and Chen, Liang-Chieh␣
↪and Chen, Bo and Tan, Mingxing and Wang, Weijun and Zhu, Yukun and Pang, Ruoming␣
↪and Vasudevan, Vijay and Le, Quoc V. and Adam, Hartwig},
    title = {Searching for MobileNetV3},
    booktitle = {Proceedings of the IEEE/CVF International Conference on Computer␣
↪Vision (ICCV)},
    month = {October},
    year = {2019}
}
```

CHAPTER 52

MobileOne

An Improved One millisecond Mobile Backbone

52.1 简介

Mobileone is proposed by apple and based on reparameterization. On the apple chips, the accuracy of the model is close to 0.76 on the ImageNet dataset when the latency is less than 1ms. Its main improvements based on RepVGG are fllowing:

- Reparameterization using Depthwise convolution and Pointwise convolution instead of normal convolution.
- Removal of the residual structure which is not friendly to access memory.

52.2 摘要

52.3 How to use

The checkpoints provided are all `training-time` models. Use the reparameterize tool or `switch_to_deploy` interface to switch them to more efficient `inference-time` architecture, which not only has fewer parameters but also less calculations.

推理图片

Use `classifier.backbone.switch_to_deploy()` interface to switch the MobileOne to a inference mode.

```
>>> import torch
>>> from mmcls.apis import init_model, inference_model
>>>
>>> model = init_model('configs/mobileone/mobileone-s0_8xb32_in1k.py', 'https://
↪download.openmmlab.com/mmclassification/v0/mobileone/mobileone-s0_8xb32_in1k_
↪20221110-0bc94952.pth')
>>> predict = inference_model(model, 'demo/demo.JPEG')
>>> print(predict['pred_class'])
sea snake
>>> print(predict['pred_score'])
0.4539405107498169
>>>
>>> # switch to deploy mode
>>> model.backbone.switch_to_deploy()
>>> predict_deploy = inference_model(model, 'demo/demo.JPEG')
>>> print(predict_deploy['pred_class'])
sea snake
>>> print(predict_deploy['pred_score'])
0.4539395272731781
```

调用模型

```
>>> import torch
>>> from mmcls.apis import init_model
>>>
>>> model = init_model('configs/mobileone/mobileone-s0_8xb32_in1k.py', 'https://
↪download.openmmlab.com/mmclassification/v0/mobileone/mobileone-s0_8xb32_in1k_
↪20221110-0bc94952.pth')
>>> inputs = torch.rand(1, 3, 224, 224).to(model.data_preprocessor.device)
>>> # To get classification scores.
>>> out = model(inputs)
>>> print(out.shape)
torch.Size([1, 1000])
>>> # To extract features.
>>> outs = model.extract_feat(inputs)
>>> print(outs[0].shape)
torch.Size([1, 768])
>>>
>>> # switch to deploy mode
>>> model.backbone.switch_to_deploy()
>>> out_deploy = model(inputs)
>>> print(out.shape)
torch.Size([1, 1000])
>>> assert torch.allclose(out, out_deploy) # pass without error
```

训练/测试

将 ImageNet 数据集放置在 `data/imagenet` 目录下，或者根据 docs 准备其他数据集。

训练：

```
python tools/train.py configs/mobileone/mobileone-s0_8xb32_in1k.py
```

Download Checkpoint:

```
wget https://download.openmmlab.com/mmclassification/v0/mobileone/mobileone-s0_8xb32_
↪in1k_20221110-0bc94952.pth
```

Test use unfused model:

```
python tools/test.py configs/mobileone/mobileone-s0_8xb32_in1k.py mobileone-s0_8xb32_
↪in1k_20221110-0bc94952.pth
```

Reparameterize checkpoint:

```
python ./tools/convert_models/reparameterize_model.py ./configs/mobileone/mobileone-
↪s0_8xb32_in1k.py mobileone-s0_8xb32_in1k_20221110-0bc94952.pth mobileone_s0_deploy.
↪pth
```

Test use fused model:

```
python tools/test.py configs/mobileone/deploy/mobileone-s0_deploy_8xb32_in1k.py␣
↪mobileone_s0_deploy.pth
```

52.3.1 Reparameterize Tool

Use provided tool to reparameterize the given model and save the checkpoint:

```
python tools/convert_models/reparameterize_model.py ${CFG_PATH} ${SRC_CKPT_PATH} $
↪{TARGET_CKPT_PATH}
```

`${CFG_PATH}` is the config file path, `${SRC_CKPT_PATH}` is the source chenpoint file path, `${TARGET_CKPT_PATH}` is the target deploy weight file path.

For example:

```
wget https://download.openmmlab.com/mmclassification/v0/mobileone/mobileone-s0_8xb32_
↪in1k_20221110-0bc94952.pth
python ./tools/convert_models/reparameterize_model.py ./configs/mobileone/mobileone-
↪s0_8xb32_in1k.py mobileone-s0_8xb32_in1k_20221110-0bc94952.pth mobileone_s0_deploy.
↪pth
```

To use reparameterized weights, the config file must switch to **the deploy config files**.

```
python tools/test.py ${Deploy_CFG} ${Deploy_Checkpoint}
```

For example of using the reparameterized weights above:

```
python ./tools/test.py ./configs/mobileone/deploy/mobileone-s0_deploy_8xb32_in1k.py ␣
↪mobileone_s0_deploy.pth
```

For more configurable parameters, please refer to the API.

52.4 结果和模型

52.4.1 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
MobileOne-s0	5.29（train） | 2.08 (deploy)	1.09 (train) | 0.28 (deploy)	71.34	89.87	config (train) | config (deploy)	model | log
MobileOne-s1	4.83 (train) | 4.76 (deploy)	0.86 (train) | 0.84 (deploy)	75.72	92.54	config (train) | config (deploy)	model | log
MobileOne-s2	7.88 (train) | 7.88 (deploy)	1.34 (train) | 1.31 (deploy)	77.37	93.34	config (train) |config (deploy)	model | log
MobileOne-s3	10.17 (train) | 10.08 (deploy)	1.95 (train) | 1.91 (deploy)	78.06	93.83	config (train) |config (deploy)	model | log
MobileOne-s4	14.95 (train) | 14.84 (deploy)	3.05 (train) | 3.00 (deploy)	79.69	94.46	config (train) |config (deploy)	model | log

52.5 引用

```
@article{mobileone2022,
  title={An Improved One millisecond Mobile Backbone},
  author={Vasu, Pavan Kumar Anasosalu and Gabriel, James and Zhu, Jeff and Tuzel,␣
↪Oncel and Ranjan, Anurag},
  journal={arXiv preprint arXiv:2206.04040},
  year={2022}
}
```

CHAPTER 53

MobileVit

MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer

53.1 简介

MobileViT aims at introducing a light-weight network, which takes the advantages of both ViTs and CNNs, uses the `InvertedResidual` blocks in MobileNetV2 and `MobileViTBlock` which refers to ViT transformer blocks to build a standard 5-stage model structure.

The MobileViTBlock reckons transformers as convolutions to perform a global representation, meanwhile conbined with original convolution layers for local representation to build a block with global receptive field. This is different from ViT, which adds an extra class token and position embeddings for learning relative relationship. Without any position embeddings, MobileViT can benfit from multi-scale inputs during training.

Also, this paper puts forward a strategy for multi-scale training to dynamically adjust batch size based on the image size to both improve training efficiency and final performance.

It is also proven effective in downstream tasks such as object detection and segmentation.

53.2 摘要

Light-weight convolutional neural networks (CNNs) are the de-facto for mobile vision tasks. Their spatial inductive biases allow them to learn representations with fewer parameters across different vision tasks. However, these networks are spatially local. To learn global representations, self-attention-based vision trans-formers (ViTs) have been adopted. Unlike CNNs, ViTs are heavy-weight. In this paper, we ask the following question: is it possible to combine the strengths of CNNs and ViTs to build a light-weight and low latency network for mobile vision tasks? Towards this end, we introduce MobileViT, a light-weight and general-purpose vision transformer for mobile devices. MobileViT presents a different perspective for the global processing of information with transformers, i.e., transformers as convolutions. Our results show that MobileViT significantly outperforms CNN- and ViT-based networks across different tasks and datasets. On the ImageNet-1k dataset, MobileViT achieves top-1 accuracy of 78.4% with about 6 million parameters, which is 3.2% and 6.2% more accurate than MobileNetv3 (CNN-based) and DeIT (ViT-based) for a similar number of parameters. On the MS-COCO object detection task, MobileViT is 5.7% more accurate than MobileNetv3 for a similar number of parameters.

53.3 使用方式

推理图片

```
>>> import torch
>>> from mmcls.apis import init_model, inference_model
>>>
>>> model = init_model('configs/mobilevit/mobilevit-small_8xb128_in1k.py', 'https://
↪download.openmmlab.com/mmclassification/v0/mobilevit/mobilevit-small_3rdparty_in1k_
↪20221018-cb4f741c.pth')
>>> predict = inference_model(model, 'demo/demo.JPEG')
>>> print(predict['pred_class'])
sea snake
>>> print(predict['pred_score'])
0.9839211702346802
```

调用模型

```
>>> import torch
>>> from mmcls.apis import init_model
>>>
>>> model = init_model('configs/mobilevit/mobilevit-small_8xb128_in1k.py', 'https://
↪download.openmmlab.com/mmclassification/v0/mobilevit/mobilevit-small_3rdparty_in1k_
↪20221018-cb4f741c.pth')
>>> inputs = torch.rand(1, 3, 224, 224).to(model.data_preprocessor.device)
>>> # To get classification scores.
>>> out = model(inputs)
```

(下页继续)

(续上页)

```
>>> print(out.shape)
torch.Size([1, 1000])
>>> # To extract features.
>>> outs = model.extract_feat(inputs)
>>> print(outs[0].shape)
torch.Size([1, 640])
```

训练/测试

将 ImageNet 数据集放置在 `data/imagenet` 目录下，或者根据 docs 准备其他数据集。

训练：

```
python tools/train.py configs/mobilevit/mobilevit-small_8xb128_in1k.py
```

测试：

```
python tools/test.py configs/mobilevit/mobilevit-small_8xb128_in1k.py https://
↪download.openmmlab.com/mmclassification/v0/mobilevit/mobilevit-small_3rdparty_in1k_
↪20221018-cb4f741c.pth
```

For more configurable parameters, please refer to the API.

53.4 结果和模型

53.4.1 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
MobileViT-XXSmall*	1.27	0.42	69.02	88.91	config	model
MobileViT-XSmall*	2.32	1.05	74.75	92.32	config	model
MobileViT-Small*	5.58	2.03	78.25	94.09	config	model

*Models with * are converted from ml-cvnets. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

53.5 引用

```
@article{mehta2021mobilevit,
  title={MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision
↪Transformer},
  author={Mehta, Sachin and Rastegari, Mohammad},
  journal={arXiv preprint arXiv:2110.02178},
  year={2021}
}
```

CHAPTER 54

MViT V2

MViTv2: Improved Multiscale Vision Transformers for Classification and Detection

54.1 摘要

In this paper, we study Multiscale Vision Transformers (MViTv2) as a unified architecture for image and video classification, as well as object detection. We present an improved version of MViT that incorporates decomposed relative positional embeddings and residual pooling connections. We instantiate this architecture in five sizes and evaluate it for ImageNet classification, COCO detection and Kinetics video recognition where it outperforms prior work. We further compare MViTv2s' pooling attention to window attention mechanisms where it outperforms the latter in accuracy/compute. Without bells-and-whistles, MViTv2 has state-of-the-art performance in 3 domains: 88.8% accuracy on ImageNet classification, 58.7 boxAP on COCO object detection as well as 86.1% on Kinetics-400 video classification.

54.2 结果和模型

54.2.1 ImageNet-1k

模型	预训练	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
MViTv2-tiny*	从头训练	24.17	4.70	82.33	96.15	config	model
MViTv2-small*	从头训练	34.87	7.00	83.63	96.51	config	model
MViTv2-base*	从头训练	51.47	10.20	84.34	96.86	config	model
MViTv2-large*	从头训练	217.99	42.10	85.25	97.14	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

54.3 引用

```
@inproceedings{li2021improved,
  title={MViTv2: Improved multiscale vision transformers for classification and
↪detection},
  author={Li, Yanghao and Wu, Chao-Yuan and Fan, Haoqi and Mangalam, Karttikeya and
↪Xiong, Bo and Malik, Jitendra and Feichtenhofer, Christoph},
  booktitle={CVPR},
  year={2022}
}
```

CHAPTER 55

PoolFormer

MetaFormer is Actually What You Need for Vision

55.1 摘要

Transformers have shown great potential in computer vision tasks. A common belief is their attention-based token mixer module contributes most to their competence. However, recent works show the attention-based module in transformers can be replaced by spatial MLPs and the resulted models still perform quite well. Based on this observation, we hypothesize that the general architecture of the transformers, instead of the specific token mixer module, is more essential to the model' s performance. To verify this, we deliberately replace the attention module in transformers with an embarrassingly simple spatial pooling operator to conduct only basic token mixing. Surprisingly, we observe that the derived model, termed as PoolFormer, achieves competitive performance on multiple computer vision tasks. For example, on ImageNet-1K, PoolFormer achieves 82.1% top-1 accuracy, surpassing well-tuned vision transformer/MLP-like baselines DeiT-B/ResMLP-B24 by 0.3%/1.1% accuracy with 35%/52% fewer parameters and 49%/61% fewer MACs. The effectiveness of PoolFormer verifies our hypothesis and urges us to initiate the concept of "MetaFormer", a general architecture abstracted from transformers without specifying the token mixer. Based on the extensive experiments, we argue that MetaFormer is the key player in achieving superior results for recent transformer and MLP-like models on vision tasks. This work calls for more future research dedicated to improving MetaFormer instead of focusing on the token mixer modules. Additionally, our proposed PoolFormer could serve as a starting baseline for future MetaFormer architecture design.

55.2 结果和模型

55.2.1 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
PoolFormer-S12*	11.92	1.87	77.24	93.51	config	model
PoolFormer-S24*	21.39	3.51	80.33	95.05	config	model
PoolFormer-S36*	30.86	5.15	81.43	95.45	config	model
PoolFormer-M36*	56.17	8.96	82.14	95.71	config	model
PoolFormer-M48*	73.47	11.80	82.51	95.95	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

55.3 引用

```
@article{yu2021metaformer,
  title={MetaFormer is Actually What You Need for Vision},
  author={Yu, Weihao and Luo, Mi and Zhou, Pan and Si, Chenyang and Zhou, Yichen and
↪Wang, Xinchao and Feng, Jiashi and Yan, Shuicheng},
  journal={arXiv preprint arXiv:2111.11418},
  year={2021}
}
```

CHAPTER 56

RegNet

Designing Network Design Spaces

56.1 摘要

In this work, we present a new network design paradigm. Our goal is to help advance the understanding of network design and discover design principles that generalize across settings. Instead of focusing on designing individual network instances, we design network design spaces that parametrize populations of networks. The overall process is analogous to classic manual design of networks, but elevated to the design space level. Using our methodology we explore the structure aspect of network design and arrive at a low-dimensional design space consisting of simple, regular networks that we call RegNet. The core insight of the RegNet parametrization is surprisingly simple: widths and depths of good networks can be explained by a quantized linear function. We analyze the RegNet design space and arrive at interesting findings that do not match the current practice of network design. The RegNet design space provides simple and fast networks that work well across a wide range of flop regimes. Under comparable training settings and flops, the RegNet models outperform the popular EfficientNet models while being up to 5x faster on GPUs.

56.2 结果和模型

56.2.1 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
RegNetX-400MF	5.16	0.41	72.56	90.78	config	model | log
RegNetX-800MF	7.26	0.81	74.76	92.32	config	model | log
RegNetX-1.6GF	9.19	1.63	76.84	93.31	config	model | log
RegNetX-3.2GF	15.3	3.21	78.09	94.08	config	model | log
RegNetX-4.0GF	22.12	4.0	78.60	94.17	config	model | log
RegNetX-6.4GF	26.21	6.51	79.38	94.65	config	model | log
RegNetX-8.0GF	39.57	8.03	79.12	94.51	config	model | log
RegNetX-12GF	46.11	12.15	79.67	95.03	config	model | log
RegNetX-400MF*	5.16	0.41	72.55	90.91	config	model
RegNetX-800MF*	7.26	0.81	75.21	92.37	config	model
RegNetX-1.6GF*	9.19	1.63	77.04	93.51	config	model
RegNetX-3.2GF*	15.3	3.21	78.26	94.20	config	model
RegNetX-4.0GF*	22.12	4.0	78.72	94.22	config	model
RegNetX-6.4GF*	26.21	6.51	79.22	94.61	config	model
RegNetX-8.0GF*	39.57	8.03	79.31	94.57	config	model
RegNetX-12GF*	46.11	12.15	79.91	94.78	config	model

*Models with * are converted from pycls. The config files of these models are only for validation.*

56.3 引用

```
@article{radosavovic2020designing,
    title={Designing Network Design Spaces},
    author={Ilija Radosavovic and Raj Prateek Kosaraju and Ross Girshick and Kaiming
↪He and Piotr Dollár},
    year={2020},
    eprint={2003.13678},
    archivePrefix={arXiv},
    primaryClass={cs.CV}
}
```

CHAPTER 57

RepLKNet

Scaling Up Your Kernels to 31x31: Revisiting Large Kernel Design in CNNs

57.1 摘要

We revisit large kernel design in modern convolutional neural networks (CNNs). Inspired by recent advances in vision transformers (ViTs), in this paper, we demonstrate that using a few large convolutional kernels instead of a stack of small kernels could be a more powerful paradigm. We suggested five guidelines, e.g., applying re-parameterized large depth-wise convolutions, to design efficient highperformance large-kernel CNNs. Following the guidelines, we propose RepLKNet, a pure CNN architecture whose kernel size is as large as 31×31, in contrast to commonly used 3×3. RepLKNet greatly closes the performance gap between CNNs and ViTs, e.g., achieving comparable or superior results than Swin Transformer on ImageNet and a few typical downstream tasks, with lower latency. RepLKNet also shows nice scalability to big data and large models, obtaining 87.8% top-1 accuracy on ImageNet and 56.0% mIoU on ADE20K, which is very competitive among the state-of-the-arts with similar model sizes. Our study further reveals that, in contrast to small-kernel CNNs, large kernel CNNs have much larger effective receptive fields and higher shape bias rather than texture bias.

57.2 结果和模型

57.2.1 ImageNet-1k

模型	Resolution	Pre-trained Dataset	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
RepLKNet-31B*	224x224	From Scratch	79.9（train） | 79.5 (deploy)	15.6 (train) | 15.4 (deploy)	83.48	96.57	config (train) | config (deploy)	model
RepLKNet-31B*	384x384	From Scratch	79.9（train） | 79.5 (deploy)	46.0 (train) | 45.3 (deploy)	84.84	97.34	config (train) | config (deploy)	model
RepLKNet-31B*	224x224	ImageNet-21K	79.9（train） | 79.5 (deploy)	15.6 (train) | 15.4 (deploy)	85.20	97.56	config (train) | config (deploy)	model
RepLKNet-31B*	384x384	ImageNet-21K	79.9（train） | 79.5 (deploy)	46.0 (train) | 45.3 (deploy)	85.99	97.75	config (train) | config (deploy)	model
RepLKNet-31L*	384x384	ImageNet-21K	172.7（train） | 172.0 (deploy)	97.2 (train) | 97.0 (deploy)	86.63	98.00	config (train) | config (deploy)	model
RepLKNet-XL*	320x320	MegData-73M	335.4（train） | 335.0 (deploy)	129.6 (train) | 129.0 (deploy)	87.57	98.39	config (train) | config (deploy)	model

*Models with * are converted from the official repo. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

57.3 How to use

The checkpoints provided are all `training-time` models. Use the reparameterize tool to switch them to more efficient `inference-time` architecture, which not only has fewer parameters but also less calculations.

57.3.1 Use tool

Use provided tool to reparameterize the given model and save the checkpoint:

```
python tools/convert_models/reparameterize_model.py ${CFG_PATH} ${SRC_CKPT_PATH} $
↪{TARGET_CKPT_PATH}
```

`${CFG_PATH}` is the config file, `${SRC_CKPT_PATH}` is the source chenpoint file, `${TARGET_CKPT_PATH}` is the target deploy weight file path.

To use reparameterized weights, the config file must switch to the deploy config files.

```
python tools/test.py ${Deploy_CFG} ${Deploy_Checkpoint} --metrics accuracy
```

57.3.2 In the code

Use `backbone.switch_to_deploy()` or `classificer.backbone.switch_to_deploy()` to switch to the deploy mode. For example:

```
from mmcls.models import build_backbone

backbone_cfg=dict(type='RepLKNet',arch='31B'),
backbone = build_backbone(backbone_cfg)
backbone.switch_to_deploy()
```

or

```
from mmcls.models import build_classifier

cfg = dict(
    type='ImageClassifier',
    backbone=dict(
        type='RepLKNet',
        arch='31B'),
    neck=dict(type='GlobalAveragePooling'),
    head=dict(
        type='LinearClsHead',
        num_classes=1000,
        in_channels=1024,
        loss=dict(type='CrossEntropyLoss', loss_weight=1.0),
        topk=(1, 5),
    ))

classifier = build_classifier(cfg)
classifier.backbone.switch_to_deploy()
```

57.4 引用

```
@inproceedings{ding2022scaling,
  title={Scaling up your kernels to 31x31: Revisiting large kernel design in cnns},
  author={Ding, Xiaohan and Zhang, Xiangyu and Han, Jungong and Ding, Guiguang},
  booktitle={Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern␣
↪Recognition},
```

(下页继续)

(续上页)

```
  pages={11963--11975},
  year={2022}
}
```

CHAPTER 58

RepMLP

RepMLP: Re-parameterizing Convolutions into Fully-connected Layers forImage Recognition

58.1 摘要

We propose RepMLP, a multi-layer-perceptron-style neural network building block for image recognition, which is composed of a series of fully-connected (FC) layers. Compared to convolutional layers, FC layers are more efficient, better at modeling the long-range dependencies and positional patterns, but worse at capturing the local structures, hence usually less favored for image recognition. We propose a structural re-parameterization technique that adds local prior into an FC to make it powerful for image recognition. Specifically, we construct convolutional layers inside a RepMLP during training and merge them into the FC for inference. On CIFAR, a simple pure-MLP model shows performance very close to CNN. By inserting RepMLP in traditional CNN, we improve ResNets by 1.8% accuracy on ImageNet, 2.9% for face recognition, and 2.3% mIoU on Cityscapes with lower FLOPs. Our intriguing findings highlight that combining the global representational capacity and positional perception of FC with the local prior of convolution can improve the performance of neural network with faster speed on both the tasks with translation invariance (e.g., semantic segmentation) and those with aligned images and positional patterns (e.g., face recognition).

58.2 结果和模型

58.2.1 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
RepMLP-B224*	68.24	6.71	80.41	95.12	train_cfg | deploy_cfg	model
RepMLP-B256*	96.45	9.69	81.11	95.5	train_cfg | deploy_cfg	model

*Models with * are converted from the official repo.. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

58.3 How to use

The checkpoints provided are all `training-time` models. Use the reparameterize tool to switch them to more efficient `inference-time` architecture, which not only has fewer parameters but also less calculations.

58.3.1 Use tool

Use provided tool to reparameterize the given model and save the checkpoint:

```
python tools/convert_models/reparameterize_model.py ${CFG_PATH} ${SRC_CKPT_PATH} $
↪{TARGET_CKPT_PATH}
```

`${CFG_PATH}` is the config file, `${SRC_CKPT_PATH}` is the source chenpoint file, `${TARGET_CKPT_PATH}` is the target deploy weight file path.

To use reparameterized weights, the config file must switch to the deploy config files.

```
python tools/test.py ${Deploy_CFG} ${Deploy_Checkpoint} --metrics accuracy
```

58.3.2 In the code

Use `backbone.switch_to_deploy()` or `classificer.backbone.switch_to_deploy()` to switch to the deploy mode. For example:

```
from mmcls.models import build_backbone

backbone_cfg=dict(type='RepMLPNet', arch='B', img_size=224, reparam_conv_kernels=(1,␣
↪3), deploy=False)
```

（下页继续）

(续上页)

```
backbone = build_backbone(backbone_cfg)
backbone.switch_to_deploy()
```

or

```
from mmcls.models import build_classifier

cfg = dict(
    type='ImageClassifier',
    backbone=dict(
        type='RepMLPNet',
        arch='B',
        img_size=224,
        reparam_conv_kernels=(1, 3),
        deploy=False),
    neck=dict(type='GlobalAveragePooling'),
    head=dict(
        type='LinearClsHead',
        num_classes=1000,
        in_channels=768,
        loss=dict(type='CrossEntropyLoss', loss_weight=1.0),
        topk=(1, 5),
    ))

classifier = build_classifier(cfg)
classifier.backbone.switch_to_deploy()
```

58.4 引用

```
@article{ding2021repmlp,
  title={Repmlp: Re-parameterizing convolutions into fully-connected layers for image
↪recognition},
  author={Ding, Xiaohan and Xia, Chunlong and Zhang, Xiangyu and Chu, Xiaojie and Han,
↪ Jungong and Ding, Guiguang},
  journal={arXiv preprint arXiv:2105.01883},
  year={2021}
}
```

CHAPTER 59

RepVGG

RepVGG: Making VGG-style ConvNets Great Again

59.1 简介

RepVGG is a VGG-style convolutional architecture. It has the following advantages:

1. The model has a VGG-like plain (a.k.a. feed-forward) topology 1 without any branches. I.e., every layer takes the output of its only preceding layer as input and feeds the output into its only following layer.
2. The model’ s body uses only 3 × 3 conv and ReLU.
3. The concrete architecture (including the specific depth and layer widths) is instantiated with no automatic search, manual refinement, compound scaling, nor other heavy designs.

59.2 摘要

59.3 How to use

The checkpoints provided are all `training-time` models. Use the reparameterize tool or `switch_to_deploy` interface to switch them to more efficient `inference-time` architecture, which not only has fewer parameters but also less calculations.

推理图片

Use `classifier.backbone.switch_to_deploy()` interface to switch the RepVGG models into inference mode.

```
>>> import torch
>>> from mmcls.apis import init_model, inference_model
>>>
>>> model = init_model('configs/repvgg/repvgg-A0_8xb32_in1k.py', 'https://download.
↪openmmlab.com/mmclassification/v0/repvgg/repvgg-A0_8xb32_in1k_20221213-60ae8e23.pth
↪')
>>> results = inference_model(model, 'demo/demo.JPEG')
>>> print( (results['pred_class'], results['pred_score']) )
('sea snake' 0.8338906168937683)
>>>
>>> # switch to deploy mode
>>> model.backbone.switch_to_deploy()
>>> results = inference_model(model, 'demo/demo.JPEG')
>>> print( (results['pred_class'], results['pred_score']) )
('sea snake', 0.7883061170578003)
```

调用模型

```
>>> import torch
>>> from mmcls.apis import get_model
>>>
>>> model = get_model("repvgg-a0_8xb32_in1k", pretrained=True)
>>> model.eval()
>>> inputs = torch.rand(1, 3, 224, 224).to(model.data_preprocessor.device)
>>> # To get classification scores.
>>> out = model(inputs)
>>> print(out.shape)
torch.Size([1, 1000])
>>> # To extract features.
>>> outs = model.extract_feat(inputs)
>>> print(outs[0].shape)
torch.Size([1, 1280])
>>>
>>> # switch to deploy mode
>>> model.backbone.switch_to_deploy()
>>> out_deploy = model(inputs)
>>> print(out.shape)
torch.Size([1, 1000])
>>> assert torch.allclose(out, out_deploy, rtol=1e-4, atol=1e-5) # pass without error
```

训练/测试

将 ImageNet 数据集放置在 `data/imagenet` 目录下，或者根据 docs 准备其他数据集。

训练：

```
python tools/train.py configs/repvgg/repvgg-a0_8xb32_in1k.py
```

Download Checkpoint:

```
wget https://download.openmmlab.com/mmclassification/v0/repvgg/repvgg-A0_8xb32_in1k_
↪20221213-60ae8e23.pth
```

Test use unfused model:

```
python tools/test.py configs/repvgg/repvgg-a0_8xb32_in1k.py repvgg-A0_8xb32_in1k_
↪20221213-60ae8e23.pth
```

Reparameterize checkpoint:

```
python ./tools/convert_models/reparameterize_model.py configs/repvgg/repvgg-a0_8xb32_
↪in1k.py repvgg-A0_8xb32_in1k_20221213-60ae8e23.pth repvgg_A0_deploy.pth
```

Test use fused model:

```
python tools/test.py configs/repvgg/repvgg-A0_8xb32_in1k.py repvgg_A0_deploy.pth --
↪cfg-options model.backbone.deploy=True
```

or

```
python tools/test.py configs/repvgg/repvgg-A0_deploy_in1k.py repvgg_A0_deploy.pth
```

For more configurable parameters, please refer to the API.

Use provided tool to reparameterize the given model and save the checkpoint:

```
python tools/convert_models/reparameterize_model.py ${CFG_PATH} ${SRC_CKPT_PATH} $
↪{TARGET_CKPT_PATH}
```

`${CFG_PATH}` is the config file path, `${SRC_CKPT_PATH}` is the source chenpoint file path, `${TARGET_CKPT_PATH}` is the target deploy weight file path.

For example:

```
# download the weight
wget https://download.openmmlab.com/mmclassification/v0/repvgg/repvgg-A0_8xb32_in1k_
↪20221213-60ae8e23.pth
# reparameterize unfused weight to fused weight
python ./tools/convert_models/reparameterize_model.py configs/repvgg/repvgg-a0_8xb32_
↪in1k.py repvgg-A0_8xb32_in1k_20221213-60ae8e23.pth repvgg-A0_deploy.pth
```

To use reparameterized weights, the config file must switch to **the deploy config files** as the deploy_A0 example or add `--cfg-options model.backbone.deploy=True` in command.

For example of using the reparameterized weights above:

```
python ./tools/test.py ./configs/repvgg/repvgg-A0_deploy_in1k.py  repvgg-A0_deploy.pth
```

You can get other deploy configs by modifying the A0_deploy example:

```
# in repvgg-A0_deploy_in1k.py
_base_ = '../repvgg-A0_8xb32_in1k.py'  # basic A0 config

model = dict(backbone=dict(deploy=True))  # switch model into deploy mode
```

or add `--cfg-options model.backbone.deploy=True` in command as following:

```
python tools/test.py configs/repvgg/repvgg-A0_8xb32_in1k.py repvgg_A0_deploy.pth --
↪cfg-options model.backbone.deploy=True
```

59.4 结果和模型

59.4.1 ImageNet-1k

模型	预训练	Params(M) (train|deploy)	Flops(G) (train|deploy)	Top-1 (%)	Top-5 (%)	配置文件	下载
repvgg-A0_8xb32_in1k	从头训练	9.11 | 8.31	1.53 | 1.36	72.37	90.56	config	model | log
repvgg-A1_8xb32_in1k	从头训练	14.09 | 12.79	2.65 | 2.37	74.47	91.85	config	model | log
repvgg-A2_8xb32_in1k	从头训练	28.21 | 25.5	5.72 | 5.12	76.49	93.09	config	model | log
repvgg-B0_8xb32_in1k	从头训练	15.82 | 14.34	3.43 | 3.06	75.27	92.21	config	model | log
repvgg-B1_8xb32_in1k	从头训练	57.42 | 51.83	13.20 | 11.81	78.19	94.04	config	model | log
repvgg-B1g2_8xb32_in1k	从头训练	45.78 | 41.36	9.86 | 8.80	77.87	93.99	config	model | log
repvgg-B1g4_8xb32_in1k	从头训练	39.97 | 36.13	8.19 | 7.30	77.81	93.77	config	model | log
repvgg-B2_8xb32_in1k	从头训练	89.02 | 80.32	20.5 | 18.4	78.58	94.23	config	model | log
repvgg-B2g4_8xb32_in1k	从头训练	61.76 | 55.78	12.7 | 11.3	79.44	94.72	config	model | log
repvgg-B3_8xb32_in1k	从头训练	123.09 | 110.96	29.2 | 26.2	80.58	95.33	config	model | log
repvgg-B3g4_8xb32_in1k	从头训练	83.83 | 75.63	18.0 | 16.1	80.26	95.15	config	model | log
repvgg-D2se_3rdparty_in1k*	从头训练	133.33 | 120.39	36.6 | 32.8	81.81	95.94	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

59.5 引用

```
@inproceedings{ding2021repvgg,
  title={Repvgg: Making vgg-style convnets great again},
  author={Ding, Xiaohan and Zhang, Xiangyu and Ma, Ningning and Han, Jungong and Ding,
↪ Guiguang and Sun, Jian},
  booktitle={Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern␣
↪Recognition},
  pages={13733--13742},
  year={2021}
}
```

CHAPTER 60

Res2Net

Res2Net: A New Multi-scale Backbone Architecture

60.1 摘要

Representing features at multiple scales is of great importance for numerous vision tasks. Recent advances in backbone convolutional neural networks (CNNs) continually demonstrate stronger multi-scale representation ability, leading to consistent performance gains on a wide range of applications. However, most existing methods represent the multi-scale features in a layer-wise manner. In this paper, we propose a novel building block for CNNs, namely Res2Net, by constructing hierarchical residual-like connections within one single residual block. The Res2Net represents multi-scale features at a granular level and increases the range of receptive fields for each network layer. The proposed Res2Net block can be plugged into the state-of-the-art backbone CNN models, e.g., ResNet, ResNeXt, and DLA. We evaluate the Res2Net block on all these models and demonstrate consistent performance gains over baseline models on widely-used datasets, e.g., CIFAR-100 and ImageNet. Further ablation studies and experimental results on representative computer vision tasks, i.e., object detection, class activation mapping, and salient object detection, further verify the superiority of the Res2Net over the state-of-the-art baseline methods.

60.2 结果和模型

60.2.1 ImageNet-1k

模型	分辨率	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
Res2Net-50-14w-8s*	224x224	25.06	4.22	78.14	93.85	config	model
Res2Net-50-26w-8s*	224x224	48.40	8.39	79.20	94.36	config	model
Res2Net-101-26w-4s*	224x224	45.21	8.12	79.19	94.44	config	model

*Models with * are converted from the official repo. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

60.3 引用

```
@article{gao2019res2net,
  title={Res2Net: A New Multi-scale Backbone Architecture},
  author={Gao, Shang-Hua and Cheng, Ming-Ming and Zhao, Kai and Zhang, Xin-Yu and
↪Yang, Ming-Hsuan and Torr, Philip},
  journal={IEEE TPAMI},
  year={2021},
  doi={10.1109/TPAMI.2019.2938758},
}
```

CHAPTER 61

ResNet

Deep Residual Learning for Image Recognition

61.1 简介

Residual Networks, or **ResNets**, learn residual functions with reference to the layer inputs, instead of learning unreferenced functions. In the mainstream previous works, like VGG, the neural networks are a stack of layers and every layer attempts to fit a desired underlying mapping. In ResNets, a few stacked layers are grouped as a block, and the layers in a block attempts to learn a residual mapping.

Formally, denoting the desired underlying mapping of a block as $\mathcal{H}(x)$, split the underlying mapping into the sum of the identity and the residual mapping as $\mathcal{H}(x) = x + \mathcal{F}(x)$, and let the stacked non-linear layers fit the residual mapping $\mathcal{F}(x)$.

Many works proved this method makes deep neural networks easier to optimize, and can gain accuracy from considerably increased depth. Recently, the residual structure is widely used in various models.

61.2 摘要

The depth of representations is of central importance for many visual recognition tasks. Solely due to our extremely deep representations, we obtain a 28% relative improvement on the COCO object detection dataset. Deep residual nets are foundations of our submissions to ILSVRC & COCO 2015 competitions, where we also won the 1st places on the tasks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation.

61.3 使用方式

推理图片

```
>>> import torch
>>> from mmcls.apis import init_model, inference_model
>>>
>>> model = init_model('configs/resnet/resnet50_8xb32_in1k.py', 'https://download.
↪openmmlab.com/mmclassification/v0/resnet/resnet50_8xb32_in1k_20210831-ea4938fc.pth')
>>> predict = inference_model(model, 'demo/demo.JPEG')
>>> print(predict['pred_class'])
sea snake
>>> print(predict['pred_score'])
0.6649363040924072
```

调用模型

```
>>> import torch
>>> from mmcls.apis import init_model
>>>
>>> model = init_model('configs/resnet/resnet50_8xb32_in1k.py', 'https://download.
↪openmmlab.com/mmclassification/v0/resnet/resnet50_8xb32_in1k_20210831-ea4938fc.pth')
>>> inputs = torch.rand(1, 3, 224, 224).to(model.data_preprocessor.device)
>>> # To get classification scores.
>>> out = model(inputs)
>>> print(out.shape)
torch.Size([1, 1000])
>>> # To extract features.
>>> outs = model.extract_feat(inputs)
>>> print(outs[0].shape)
torch.Size([1, 2048])
```

训练/测试

将 ImageNet 数据集放置在 `data/imagenet` 目录下，或者根据 docs 准备其他数据集。

训练：

```
python tools/train.py configs/resnet/resnet50_8xb32_in1k.py
```

测试：

```
python tools/test.py configs/resnet/resnet50_8xb32_in1k.py https://download.openmmlab.
↪com/mmclassification/v0/resnet/resnet50_b16x8_cifar10_20210528-f54bfad9.pth
```

For more configurable parameters, please refer to the API.

61.4 结果和模型

The pre-trained models on ImageNet-21k are used to fine-tune, and therefore don’t have evaluation results.

模型	分辨率	参数量（M）	Flops(G)	下载
ResNet-50-mill	224x224	86.74	15.14	model

The “mill” means using the mutil-label pretrain weight from ImageNet-21K Pretraining for the Masses.

61.4.1 Cifar10

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
ResNet-18	11.17	0.56	94.82	99.87	config	model | log
ResNet-34	21.28	1.16	95.34	99.87	config	model | log
ResNet-50	23.52	1.31	95.55	99.91	config	model | log
ResNet-101	42.51	2.52	95.58	99.87	config	model | log
ResNet-152	58.16	3.74	95.76	99.89	config	model | log

61.4.2 Cifar100

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
ResNet-50	23.71	1.31	79.90	95.19	config	model | log

61.4.3 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
ResNet-18	11.69	1.82	69.90	89.43	config	model | log
ResNet-34	21.8	3.68	73.62	91.59	config	model | log
ResNet-50	25.56	4.12	76.55	93.06	config	model | log
ResNet-101	44.55	7.85	77.97	94.06	config	model | log
ResNet-152	60.19	11.58	78.48	94.13	config	model | log
ResNetV1C-50	25.58	4.36	77.01	93.58	config	model | log
ResNetV1C-101	44.57	8.09	78.30	94.27	config	model | log
ResNetV1C-152	60.21	11.82	78.76	94.41	config	model | log
ResNetV1D-50	25.58	4.36	77.54	93.57	config	model | log
ResNetV1D-101	44.57	8.09	78.93	94.48	config	model | log
ResNetV1D-152	60.21	11.82	79.41	94.70	config	model | log
ResNet-50 (fp16)	25.56	4.12	76.30	93.07	config	model | log
Wide-ResNet-50*	68.88	11.44	78.48	94.08	config	model
Wide-ResNet-101*	126.89	22.81	78.84	94.28	config	model
ResNet-50 (rsb-a1)	25.56	4.12	80.12	94.78	config	model | log
ResNet-50 (rsb-a2)	25.56	4.12	79.55	94.37	config	model | log
ResNet-50 (rsb-a3)	25.56	4.12	78.30	93.80	config	model | log

The "rsb" means using the training settings from ResNet strikes back: An improved training procedure in timm.

*Models with * are converted from the official repo. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

61.4.4 CUB-200-2011

模型	预训练	分辨率	参数量（M）	Flops(G)	Top-1 (%)	配置文件	下载
ResNet-50	ImageNet-21k-mill	448x448	23.92	16.48	88.45	config	model | log

61.5 引用

```
@inproceedings{he2016deep,
  title={Deep residual learning for image recognition},
  author={He, Kaiming and Zhang, Xiangyu and Ren, Shaoqing and Sun, Jian},
  booktitle={Proceedings of the IEEE conference on computer vision and pattern
↪recognition},
  pages={770--778},
  year={2016}
}
```

CHAPTER 62

ResNeXt

Aggregated Residual Transformations for Deep Neural Networks

62.1 摘要

We present a simple, highly modularized network architecture for image classification. Our network is constructed by repeating a building block that aggregates a set of transformations with the same topology. Our simple design results in a homogeneous, multi-branch architecture that has only a few hyper-parameters to set. This strategy exposes a new dimension, which we call "cardinality" (the size of the set of transformations), as an essential factor in addition to the dimensions of depth and width. On the ImageNet-1K dataset, we empirically show that even under the restricted condition of maintaining complexity, increasing cardinality is able to improve classification accuracy. Moreover, increasing cardinality is more effective than going deeper or wider when we increase the capacity. Our models, named ResNeXt, are the foundations of our entry to the ILSVRC 2016 classification task in which we secured 2nd place. We further investigate ResNeXt on an ImageNet-5K set and the COCO detection set, also showing better results than its ResNet counterpart. The code and models are publicly available online.

62.2 结果和模型

62.2.1 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
ResNeXt-32x4d-50	25.03	4.27	77.90	93.66	config	model | log
ResNeXt-32x4d-101	44.18	8.03	78.61	94.17	config	model | log
ResNeXt-32x8d-101	88.79	16.5	79.27	94.58	config	model | log
ResNeXt-32x4d-152	59.95	11.8	78.88	94.33	config	model | log

62.3 引用

```
@inproceedings{xie2017aggregated,
  title={Aggregated residual transformations for deep neural networks},
  author={Xie, Saining and Girshick, Ross and Doll{\'a}r, Piotr and Tu, Zhuowen and
↪He, Kaiming},
  booktitle={Proceedings of the IEEE conference on computer vision and pattern
↪recognition},
  pages={1492--1500},
  year={2017}
}
```

CHAPTER 63

Reversible Vision Transformers

Reversible Vision Tranformers

63.1 简介

RevViT is initially described in Reversible Vision Tranformers, which introduce the reversible idea into vision transformer, to reduce the GPU memory footprint required for training.

63.2 摘要

63.3 结果和模型

模型	预训练	参数量 (M)	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
revvit-small_3rdparty_in1k*	从头训练	22.43	4.58	79.87	94.90	config	model
revvit-base_3rdparty_in1k*	从头训练	87.33	17.49	81.81	95.56	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

63.4 引用

```
@inproceedings{mangalam2022reversible,
  title={Reversible Vision Transformers},
  author={Mangalam, Karttikeya and Fan, Haoqi and Li, Yanghao and Wu, Chao-Yuan and
↪Xiong, Bo and Feichtenhofer, Christoph and Malik, Jitendra},
  booktitle={Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern
↪Recognition},
  pages={10830--10840},
  year={2022}
}
```

CHAPTER 64

SE-ResNet

Squeeze-and-Excitation Networks

64.1 摘要

The central building block of convolutional neural networks (CNNs) is the convolution operator, which enables networks to construct informative features by fusing both spatial and channel-wise information within local receptive fields at each layer. A broad range of prior research has investigated the spatial component of this relationship, seeking to strengthen the representational power of a CNN by enhancing the quality of spatial encodings throughout its feature hierarchy. In this work, we focus instead on the channel relationship and propose a novel architectural unit, which we term the "Squeeze-and-Excitation" (SE) block, that adaptively recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels. We show that these blocks can be stacked together to form SENet architectures that generalise extremely effectively across different datasets. We further demonstrate that SE blocks bring significant improvements in performance for existing state-of-the-art CNNs at slight additional computational cost. Squeeze-and-Excitation Networks formed the foundation of our ILSVRC 2017 classification submission which won first place and reduced the top-5 error to 2.251%, surpassing the winning entry of 2016 by a relative improvement of ~25%.

64.2 结果和模型

64.2.1 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
SE-ResNet-50	28.09	4.13	77.74	93.84	config	model | log
SE-ResNet-101	49.33	7.86	78.26	94.07	config	model | log

64.3 引用

```
@inproceedings{hu2018squeeze,
  title={Squeeze-and-excitation networks},
  author={Hu, Jie and Shen, Li and Sun, Gang},
  booktitle={Proceedings of the IEEE conference on computer vision and pattern␣
↪recognition},
  pages={7132--7141},
  year={2018}
}
```

CHAPTER 65

ShuffleNet V1

ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices

65.1 摘要

We introduce an extremely computation-efficient CNN architecture named ShuffleNet, which is designed specially for mobile devices with very limited computing power (e.g., 10-150 MFLOPs). The new architecture utilizes two new operations, pointwise group convolution and channel shuffle, to greatly reduce computation cost while maintaining accuracy. Experiments on ImageNet classification and MS COCO object detection demonstrate the superior performance of ShuffleNet over other structures, e.g. lower top-1 error (absolute 7.8%) than recent MobileNet on ImageNet classification task, under the computation budget of 40 MFLOPs. On an ARM-based mobile device, ShuffleNet achieves ~13x actual speedup over AlexNet while maintaining comparable accuracy.

65.2 结果和模型

65.2.1 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
ShuffleNetV1 1.0x (group=3)	1.87	0.146	68.13	87.81	config	model | log

65.3 引用

```
@inproceedings{zhang2018shufflenet,
  title={Shufflenet: An extremely efficient convolutional neural network for mobile␣
↪devices},
  author={Zhang, Xiangyu and Zhou, Xinyu and Lin, Mengxiao and Sun, Jian},
  booktitle={Proceedings of the IEEE conference on computer vision and pattern␣
↪recognition},
  pages={6848--6856},
  year={2018}
}
```

CHAPTER 66

ShuffleNet V2

Shufflenet v2: Practical guidelines for efficient cnn architecture design

66.1 摘要

Currently, the neural network architecture design is mostly guided by the *indirect* metric of computation complexity, i.e., FLOPs. However, the *direct* metric, e.g., speed, also depends on the other factors such as memory access cost and platform characterics. Thus, this work proposes to evaluate the direct metric on the target platform, beyond only considering FLOPs. Based on a series of controlled experiments, this work derives several practical *guidelines* for efficient network design. Accordingly, a new architecture is presented, called *ShuffleNet V2*. Comprehensive ablation experiments verify that our model is the state-of-the-art in terms of speed and accuracy tradeoff.

66.2 结果和模型

66.2.1 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
ShuffleNetV2 1.0x	2.28	0.149	69.55	88.92	config	model | log

66.3 引用

```
@inproceedings{ma2018shufflenet,
  title={Shufflenet v2: Practical guidelines for efficient cnn architecture design},
  author={Ma, Ningning and Zhang, Xiangyu and Zheng, Hai-Tao and Sun, Jian},
  booktitle={Proceedings of the European conference on computer vision (ECCV)},
  pages={116--131},
  year={2018}
}
```

CHAPTER 67

Swin Transformer

Swin Transformer: Hierarchical Vision Transformer using Shifted Windows

67.1 简介

Swin Transformer (the name **Swin** stands for Shifted window) is initially described in the paper, which capably serves as a general-purpose backbone for computer vision. It is basically a hierarchical Transformer whose representation is computed with shifted windows. The shifted windowing scheme brings greater efficiency by limiting self-attention computation to non-overlapping local windows while also allowing for cross-window connection.

Swin Transformer achieves strong performance on COCO object detection (58.7 box AP and 51.1 mask AP on test-dev) and ADE20K semantic segmentation (53.5 mIoU on val), surpassing previous models by a large margin.

67.2 摘要

67.3 使用方式

推理图片

```
>>> import torch
>>> from mmcls.apis import init_model, inference_model
>>>
```

(下页继续)

(续上页)

```
>>> model = init_model('configs/swin_transformer/swin-tiny_16xb64_in1k.py', 'https://
↪download.openmmlab.com/mmclassification/v0/swin-transformer/convert/swin_tiny_
↪patch4_window7_224-160bb0a5.pth')
>>> predict = inference_model(model, 'demo/demo.JPEG')
>>> print(predict['pred_class'])
sea snake
>>> print(predict['pred_score'])
0.9092751741409302
```

调用模型

```
>>> import torch
>>> from mmcls.apis import init_model
>>>
>>> model = init_model('configs/swin_transformer/swin-tiny_16xb64_in1k.py', 'https://
↪download.openmmlab.com/mmclassification/v0/swin-transformer/convert/swin_tiny_
↪patch4_window7_224-160bb0a5.pth')
>>> inputs = torch.rand(1, 3, 224, 224).to(model.data_preprocessor.device)
>>> # To get classification scores.
>>> out = model(inputs)
>>> print(out.shape)
torch.Size([1, 1000])
>>> # To extract features.
>>> outs = model.extract_feat(inputs)
>>> print(outs[0].shape)
torch.Size([1, 768])
```

训练/测试

将 ImageNet 数据集放置在 `data/imagenet` 目录下，或者根据 docs 准备其他数据集。

训练：

```
python tools/train.py configs/swin_transformer/swin-base_16xb64_in1k.py
```

测试：

```
python tools/test.py configs/swin_transformer/swin-base_16xb64_in1k.py https://
↪download.openmmlab.com/mmclassification/v0/swin-transformer/convert/swin_tiny_
↪patch4_window7_224-160bb0a5.pth
```

For more configurable parameters, please refer to the API.

67.4 结果和模型

67.4.1 ImageNet-21k

The pre-trained models on ImageNet-21k are used to fine-tune, and therefore don’t have evaluation results.

模型	分辨率	参数量（M）	Flops(G)	下载
Swin-B	224x224	86.74	15.14	model
Swin-B	384x384	86.88	44.49	model
Swin-L	224x224	195.00	34.04	model
Swin-L	384x384	195.20	100.04	model

67.4.2 ImageNet-1k

模型	预训练	分辨率	参数量(M)	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
Swin-T	从头训练	224x224	28.29	4.36	81.18	95.61	config	model | log
Swin-S	从头训练	224x224	49.61	8.52	83.02	96.29	config	model | log
Swin-B	从头训练	224x224	87.77	15.14	83.36	96.44	config	model | log
Swin-S*	从头训练	224x224	49.61	8.52	83.21	96.25	config	model
Swin-B*	从头训练	224x224	87.77	15.14	83.42	96.44	config	model
Swin-B*	从头训练	384x384	87.90	44.49	84.49	96.95	config	model
Swin-B*	ImageNet-21k	224x224	87.77	15.14	85.16	97.50	config	model
Swin-B*	ImageNet-21k	384x384	87.90	44.49	86.44	98.05	config	model
Swin-L*	ImageNet-21k	224x224	196.53	34.04	86.24	97.88	config	model
Swin-L*	ImageNet-21k	384x384	196.74	100.04	87.25	98.25	config	model

*Models with * are converted from the official repo. The config files of these models are only for validation. We don’t ensure these config files’ training accuracy and welcome you to contribute your reproduction results.*

67.4.3 CUB-200-2011

模型	预训练	分辨率	参数量（M）	Flops(G)	Top-1 (%)	配置文件	下载
Swin-L	ImageNet-21k	384x384	195.51	100.04	91.87	config	model | log

67.5 引用

```
@article{liu2021Swin,
  title={Swin Transformer: Hierarchical Vision Transformer using Shifted Windows},
  author={Liu, Ze and Lin, Yutong and Cao, Yue and Hu, Han and Wei, Yixuan and Zhang,␣
↪Zheng and Lin, Stephen and Guo, Baining},
  journal={arXiv preprint arXiv:2103.14030},
  year={2021}
}
```

CHAPTER 68

Swin Transformer V2

Swin Transformer V2: Scaling Up Capacity and Resolution

68.1 简介

Swin Transformer V2 is a work on the scale up visual model based on Swin Transformer. In the visual field, We can not increase the performance by just simply scaling up the visual model like NLP models. The possible reasons mentioned in the article are:

- Training instability when increasing the vision model
- Migrating the model trained at low resolution to a larger scale resolution task
- Too mush GPU memory

To solve it, The following method improvements are proposed in the paper:

- post normalization: layer normalization after self-attention layer and MLP block
- scaled cosine attention approach: use cosine similarity to calculate the relationship between token pairs
- log-spaced continuous position bias: redefine relative position encoding

68.2 摘要

Large-scale NLP models have been shown to significantly improve the performance on language tasks with no signs of saturation. They also demonstrate amazing few-shot capabilities like that of human beings. This paper aims to explore large-scale models in computer vision. We tackle three major issues in training and application of large vision models, including training instability, resolution gaps between pre-training and fine-tuning, and hunger on labelled data. Three main techniques are proposed: 1) a residual-post-norm method combined with cosine attention to improve training stability; 2) A log-spaced continuous position bias method to effectively transfer models pre-trained using low-resolution images to downstream tasks with high-resolution inputs; 3) A self-supervised pre-training method, SimMIM, to reduce the needs of vast labeled images. Through these techniques, this paper successfully trained a 3 billion-parameter Swin Transformer V2 model, which is the largest dense vision model to date, and makes it capable of training with images of up to 1,536×1,536 resolution. It set new performance records on 4 representative vision tasks, including ImageNet-V2 image classification, COCO object detection, ADE20K semantic segmentation, and Kinetics-400 video action classification. Also note our training is much more efficient than that in Google's billion-level visual models, which consumes 40 times less labelled data and 40 times less training time.

68.3 使用方式

推理图片

```
>>> import torch
>>> from mmcls.apis import init_model, inference_model
>>>
>>> model = init_model('configs/swin_transformer_v2/swinv2-tiny-w16_16xb64_in1k-256px.
↪py', 'https://download.openmmlab.com/mmclassification/v0/swin-v2/swinv2-tiny-w16_
↪3rdparty_in1k-256px_20220803-9651cdd7.pth')
>>> predict = inference_model(model, 'demo/demo.JPEG')
>>> print(predict['pred_class'])
sea snake
>>> print(predict['pred_score'])
0.9504518508911133
```

调用模型

```
>>> import torch
>>> from mmcls.apis import init_model
>>>
>>> model = init_model('configs/swin_transformer_v2/swinv2-tiny-w16_16xb64_in1k-256px.
↪py', 'https://download.openmmlab.com/mmclassification/v0/swin-v2/swinv2-tiny-w16_
↪3rdparty_in1k-256px_20220803-9651cdd7.pth')
>>> inputs = torch.rand(1, 3, 224, 224).to(model.data_preprocessor.device)
>>> # To get classification scores.
```

(下页继续)

(续上页)

```
>>> out = model(inputs)
>>> print(out.shape)
torch.Size([1, 1000])
>>> # To extract features.
>>> outs = model.extract_feat(inputs)
>>> print(outs[0].shape)
torch.Size([1, 2048])
```

训练/测试

将 ImageNet 数据集放置在 `data/imagenet` 目录下，或者根据 docs 准备其他数据集。

训练：

```
python tools/train.py configs/swin_transformer_v2/swinv2-tiny-w16_16xb64_in1k-256px.py
```

测试：

```
python tools/test.py configs/swin_transformer_v2/swinv2-tiny-w16_16xb64_in1k-256px.py␣
↪https://download.openmmlab.com/mmclassification/v0/swin-v2/swinv2-tiny-w16_3rdparty_
↪in1k-256px_20220803-9651cdd7.pth
```

For more configurable parameters, please refer to the API.

68.4 结果和模型

68.4.1 ImageNet-21k

The pre-trained models on ImageNet-21k are used to fine-tune, and therefore don’t have evaluation results.

模型	分辨率	参数量（M）	Flops(G)	下载
Swin-B*	192x192	87.92	8.51	model
Swin-L*	192x192	196.74	19.04	model

68.4.2 ImageNet-1k

模型	预训练	分辨率	window	参数量(M)	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
Swin-T*	从头训练	256x256	8x8	28.35	4.35	81.76	95.87	config	model
Swin-T*	从头训练	256x256	16x16	28.35	4.4	82.81	96.23	config	model
Swin-S*	从头训练	256x256	8x8	49.73	8.45	83.74	96.6	config	model
Swin-S*	从头训练	256x256	16x16	49.73	8.57	84.13	96.83	config	model
Swin-B*	从头训练	256x256	8x8	87.92	14.99	84.2	96.86	config	model
Swin-B*	从头训练	256x256	16x16	87.92	15.14	84.6	97.05	config	model
Swin-B*	ImageNet-21k	256x256	16x16	87.92	15.14	86.17	97.88	config	model
Swin-B*	ImageNet-21k	384x384	24x24	87.92	34.07	87.14	98.23	config	model
Swin-L*	ImageNet-21k	256X256	16x16	196.75	33.86	86.93	98.06	config	model
Swin-L*	ImageNet-21k	384x384	24x24	196.75	76.2	87.59	98.27	config	model

*Models with * are converted from the official repo. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

ImageNet-21k pretrained models with input resolution of 256x256 and 384x384 both fine-tuned from the same pre-training model using a smaller input resolution of 192x192.

68.5 引用

```
@article{https://doi.org/10.48550/arxiv.2111.09883,
  doi = {10.48550/ARXIV.2111.09883},
  url = {https://arxiv.org/abs/2111.09883},
  author = {Liu, Ze and Hu, Han and Lin, Yutong and Yao, Zhuliang and Xie, Zhenda and
↪Wei, Yixuan and Ning, Jia and Cao, Yue and Zhang, Zheng and Dong, Li and Wei, Furu
↪and Guo, Baining},
  keywords = {Computer Vision and Pattern Recognition (cs.CV), FOS: Computer and
↪information sciences, FOS: Computer and information sciences},
```

(下页继续)

(续上页)

```
  title = {Swin Transformer V2: Scaling Up Capacity and Resolution},
  publisher = {arXiv},
  year = {2021},
  copyright = {Creative Commons Attribution 4.0 International}
}
```

CHAPTER 69

Tokens-to-Token ViT

Tokens-to-Token ViT: Training Vision Transformers from Scratch on ImageNet

69.1 摘要

Transformers, which are popular for language modeling, have been explored for solving vision tasks recently, e.g., the Vision Transformer (ViT) for image classification. The ViT model splits each image into a sequence of tokens with fixed length and then applies multiple Transformer layers to model their global relation for classification. However, ViT achieves inferior performance to CNNs when trained from scratch on a midsize dataset like ImageNet. We find it is because: 1) the simple tokenization of input images fails to model the important local structure such as edges and lines among neighboring pixels, leading to low training sample efficiency; 2) the redundant attention backbone design of ViT leads to limited feature richness for fixed computation budgets and limited training samples. To overcome such limitations, we propose a new Tokens-To-Token Vision Transformer (T2T-ViT), which incorporates 1) a layer-wise Tokens-to-Token (T2T) transformation to progressively structurize the image to tokens by recursively aggregating neighboring Tokens into one Token (Tokens-to-Token), such that local structure represented by surrounding tokens can be modeled and tokens length can be reduced; 2) an efficient backbone with a deep-narrow structure for vision transformer motivated by CNN architecture design after empirical study. Notably, T2T-ViT reduces the parameter count and MACs of vanilla ViT by half, while achieving more than 3.0% improvement when trained from scratch on ImageNet. It also outperforms ResNets and achieves comparable performance with MobileNets by directly training on ImageNet. For example, T2T-ViT with comparable size to ResNet50 (21.5M parameters) can achieve 83.3% top1 accuracy in image resolution 384×384 on ImageNet.

69.2 结果和模型

69.2.1 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
T2T-ViT_t-14	21.47	4.34	81.83	95.84	config	model | log
T2T-ViT_t-19	39.08	7.80	82.63	96.18	config	model | log
T2T-ViT_t-24	64.00	12.69	82.71	96.09	config	model | log

In consistent with the official repo, we adopt the best checkpoints during training.

69.3 引用

```
@article{yuan2021tokens,
  title={Tokens-to-token vit: Training vision transformers from scratch on imagenet},
  author={Yuan, Li and Chen, Yunpeng and Wang, Tao and Yu, Weihao and Shi, Yujun and
↪Tay, Francis EH and Feng, Jiashi and Yan, Shuicheng},
  journal={arXiv preprint arXiv:2101.11986},
  year={2021}
}
```

CHAPTER 70

TinyViT

TinyViT: Fast Pretraining Distillation for Small Vision Transformers

70.1 摘要

Vision transformer (ViT) recently has drawn great attention in computer vision due to its remarkable model capability. However, most prevailing ViT models suffer from huge number of parameters, restricting their applicability on devices with limited resources. To alleviate this issue, we propose TinyViT, a new family of tiny and efficient small vision transformers pretrained on large-scale datasets with our proposed fast distillation framework. The central idea is to transfer knowledge from large pretrained models to small ones, while enabling small models to get the dividends of massive pretraining data. More specifically, we apply distillation during pretraining for knowledge transfer. The logits of large teacher models are sparsified and stored in disk in advance to save the memory cost and computation overheads. The tiny student transformers are automatically scaled down from a large pretrained model with computation and parameter constraints. Comprehensive experiments demonstrate the efficacy of TinyViT. It achieves a top-1 accuracy of 84.8% on ImageNet-1k with only 21M parameters, being comparable to SwinB pretrained on ImageNet-21k while using 4.2 times fewer parameters. Moreover, increasing image resolutions, TinyViT can reach 86.5% accuracy, being slightly better than Swin-L while using only 11% parameters. Last but not the least, we demonstrate a good transfer ability of TinyViT on various downstream tasks.

70.2 结果和模型

70.2.1 ImageNet-1k

模型	预训练	参数量 (M)	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
tinyvit-5m_3rdparty_in1k*	从头训练	5.39	1.29	79.02	94.74	config	model
tinyvit-5m_in21k-distill-pre_3rdparty_in1k*	ImageNet-21k (distill)	5.39	1.29	80.71	95.57	config	model
tinyvit-11m_3rdparty_in1k*	从头训练	11.00	2.05	81.44	95.79	config	model
tinyvit-11m_in21k-distill-pre_3rdparty_in1k*	ImageNet-21k (distill)	11.00	2.05	83.19	96.53	config	model
tinyvit-21m_3rdparty_in1k*	从头训练	21.20	4.30	83.08	96.58	config	model
tinyvit-21m_in21k-distill-pre_3rdparty_in1k*	ImageNet-21k (distill)	21.20	4.30	84.85	97.27	config	model
tinyvit-21m_in21k-distill-pre_3rdparty_in1k-384px*	ImageNet-21k (distill)	21.23	13.85	86.21	97.77	config	model
tinyvit-21m_in21k-distill-pre_3rdparty_in1k-512px*	ImageNet-21k (distill)	21.27	27.15	86.44	97.89	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

CHAPTER 71

TNT

Transformer in Transformer

71.1 摘要

Transformer is a new kind of neural architecture which encodes the input data as powerful features via the attention mechanism. Basically, the visual transformers first divide the input images into several local patches and then calculate both representations and their relationship. Since natural images are of high complexity with abundant detail and color information, the granularity of the patch dividing is not fine enough for excavating features of objects in different scales and locations. In this paper, we point out that the attention inside these local patches are also essential for building visual transformers with high performance and we explore a new architecture, namely, Transformer iN Transformer (TNT). Specifically, we regard the local patches (e.g., 16×16) as "visual sentences" and present to further divide them into smaller patches (e.g., 4×4) as "visual words". The attention of each word will be calculated with other words in the given visual sentence with negligible computational costs. Features of both words and sentences will be aggregated to enhance the representation ability. Experiments on several benchmarks demonstrate the effectiveness of the proposed TNT architecture, e.g., we achieve an 81.5% top-1 accuracy on the ImageNet, which is about 1.7% higher than that of the state-of-the-art visual transformer with similar computational cost.

71.2 结果和模型

71.2.1 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
TNT-small*	23.76	3.36	81.52	95.73	config	model

*Models with * are converted from timm. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

71.3 引用

```
@misc{han2021transformer,
      title={Transformer in Transformer},
      author={Kai Han and An Xiao and Enhua Wu and Jianyuan Guo and Chunjing Xu and
↪Yunhe Wang},
      year={2021},
      eprint={2103.00112},
      archivePrefix={arXiv},
      primaryClass={cs.CV}
}
```

CHAPTER 72

Twins

Twins: Revisiting the Design of Spatial Attention in Vision Transformers

72.1 摘要

Very recently, a variety of vision transformer architectures for dense prediction tasks have been proposed and they show that the design of spatial attention is critical to their success in these tasks. In this work, we revisit the design of the spatial attention and demonstrate that a carefully-devised yet simple spatial attention mechanism performs favourably against the state-of-the-art schemes. As a result, we propose two vision transformer architectures, namely, Twins-PCPVT and Twins-SVT. Our proposed architectures are highly-efficient and easy to implement, only involving matrix multiplications that are highly optimized in modern deep learning frameworks. More importantly, the proposed architectures achieve excellent performance on a wide range of visual tasks, including image level classification as well as dense detection and segmentation. The simplicity and strong performance suggest that our proposed architectures may serve as stronger backbones for many vision tasks. Our code is released at this https URL.

72.2 结果和模型

72.2.1 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
PCPVT-small*	24.11	3.67	81.14	95.69	config	model
PCPVT-base*	43.83	6.45	82.66	96.26	config	model
PCPVT-large*	60.99	9.51	83.09	96.59	config	model
SVT-small*	24.06	2.82	81.77	95.57	config	model
SVT-base*	56.07	8.35	83.13	96.29	config	model
SVT-large*	99.27	14.82	83.60	96.50	config	model

*Models with * are converted from the official repo. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results. The validation accuracy is a little different from the official paper because of the PyTorch version. This result is get in PyTorch=1.9 while the official result is get in PyTorch=1.7*

72.3 引用

```
@article{chu2021twins,
  title={Twins: Revisiting spatial attention design in vision transformers},
  author={Chu, Xiangxiang and Tian, Zhi and Wang, Yuqing and Zhang, Bo and Ren,␣
↪Haibing and Wei, Xiaolin and Xia, Huaxia and Shen, Chunhua},
  journal={arXiv preprint arXiv:2104.13840},
  year={2021}altgvt
}
```

CHAPTER 73

Visual Attention Network

Visual Attention Network

73.1 摘要

While originally designed for natural language processing (NLP) tasks, the self-attention mechanism has recently taken various computer vision areas by storm. However, the 2D nature of images brings three challenges for applying self-attention in computer vision. (1) Treating images as 1D sequences neglects their 2D structures. (2) The quadratic complexity is too expensive for high-resolution images. (3) It only captures spatial adaptability but ignores channel adaptability. In this paper, we propose a novel large kernel attention (LKA) module to enable self-adaptive and long-range correlations in self-attention while avoiding the above issues. We further introduce a novel neural network based on LKA, namely Visual Attention Network (VAN). While extremely simple and efficient, VAN outperforms the state-of-the-art vision transformers and convolutional neural networks with a large margin in extensive experiments, including image classification, object detection, semantic segmentation, instance segmentation, etc.

73.2 结果和模型

73.2.1 ImageNet-1k

模型	预训练	分辨率	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
VAN-T*	从头训练	224x224	4.11	0.88	75.41	93.02	config	model
VAN-S*	从头训练	224x224	13.86	2.52	81.01	95.63	config	model
VAN-B*	从头训练	224x224	26.58	5.03	82.80	96.21	config	model
VAN-L*	从头训练	224x224	44.77	8.99	83.86	96.73	config	model

*Models with * are converted from the official repo. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.

73.3 引用

```
@article{guo2022visual,
  title={Visual Attention Network},
  author={Guo, Meng-Hao and Lu, Cheng-Ze and Liu, Zheng-Ning and Cheng, Ming-Ming and
↪Hu, Shi-Min},
  journal={arXiv preprint arXiv:2202.09741},
  year={2022}
}
```

CHAPTER 74

VGG

Very Deep Convolutional Networks for Large-Scale Image Recognition

74.1 摘要

In this work we investigate the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. Our main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small (3x3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16-19 weight layers. These findings were the basis of our ImageNet Challenge 2014 submission, where our team secured the first and the second places in the localisation and classification tracks respectively. We also show that our representations generalise well to other datasets, where they achieve state-of-the-art results. We have made our two best-performing ConvNet models publicly available to facilitate further research on the use of deep visual representations in computer vision.

74.2 结果和模型

74.2.1 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
VGG-11	132.86	7.63	68.75	88.87	config	model | log
VGG-13	133.05	11.34	70.02	89.46	config	model | log
VGG-16	138.36	15.5	71.62	90.49	config	model | log
VGG-19	143.67	19.67	72.41	90.80	config	model | log
VGG-11-BN	132.87	7.64	70.67	90.16	config	model | log
VGG-13-BN	133.05	11.36	72.12	90.66	config	model | log
VGG-16-BN	138.37	15.53	73.74	91.66	config	model | log
VGG-19-BN	143.68	19.7	74.68	92.27	config	model | log

74.3 引用

```
@article{simonyan2014very,
  title={Very deep convolutional networks for large-scale image recognition},
  author={Simonyan, Karen and Zisserman, Andrew},
  journal={arXiv preprint arXiv:1409.1556},
  year={2014}
}
```

CHAPTER 75

VIG

Vision GNN: An Image is Worth Graph of Nodes

75.1 摘要

Network architecture plays a key role in the deep learning-based computer vision system. The widely-used convolutional neural network and transformer treat the image as a grid or sequence structure, which is not flexible to capture irregular and complex objects. In this paper, we propose to represent the image as a graph structure and introduce a new Vision GNN (ViG) architecture to extract graph-level feature for visual tasks. We first split the image to a number of patches which are viewed as nodes, and construct a graph by connecting the nearest neighbors. Based on the graph representation of images, we build our ViG model to transform and exchange information among all the nodes. ViG consists of two basic modules: Grapher module with graph convolution for aggregating and updating graph information, and FFN module with two linear layers for node feature transformation. Both isotropic and pyramid architectures of ViG are built with different model sizes. Extensive experiments on image recognition and object detection tasks demonstrate the superiority of our ViG architecture. We hope this pioneering study of GNN on general visual tasks will provide useful inspiration and experience for future research.

75.2 结果和模型

75.2.1 ImageNet-1k

模型	预训练	参数量(M)	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
vig-tiny_3rdparty_in1k*	从头训练	7.18	1.31	74.40	92.34	config	model
vig-small_3rdparty_in1k*	从头训练	22.75	4.54	80.61	95.28	config	model
vig-base_3rdparty_in1k*	从头训练	20.68	17.68	82.64	96.04	config	model
pvig-tiny_3rdparty_in1k*	从头训练	9.46	1.71	78.38	94.38	config	model
pvig-small_3rdparty_in1k*	从头训练	29.02	4.57	82.00	95.97	config	model
pvig-medium_3rdparty_in1k*	从头训练	51.68	8.89	83.12	96.35	config	model
pvig-base_3rdparty_in1k*	从头训练	95.21	16.86	83.59	96.52	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

75.3 引用

```
@inproceedings{han2022vig,
  title={Vision GNN: An Image is Worth Graph of Nodes},
  author={Kai Han and Yunhe Wang and Jianyuan Guo and Yehui Tang and Enhua Wu},
  booktitle={NeurIPS},
  year={2022}
}
```

CHAPTER 76

Vision Transformer

An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

76.1 简介

Vision Transformer, known as **ViT**, succeeded in using a full transformer to outperform previous works that based on convolutional networks in vision field. ViT splits image into patches to feed the multi-head attentions, concatenates a learnable class token for final prediction and adds a learnable position embeddings for relative positional message between patches. Based on these three techniques with attentions, ViT provides a brand-new pattern to build a basic structure in vision field.

The strategy works even better when coupled with large datasets pre-trainings. Because of its simplicity and effectiveness, some after works in classification field are originated from ViT. And even in recent multi-modality field, ViT-based method still plays a role in it.

76.2 摘要

While the Transformer architecture has become the de-facto standard for natural language processing tasks, its applications to computer vision remain limited. In vision, attention is either applied in conjunction with convolutional networks, or used to replace certain components of convolutional networks while keeping their overall structure in place. We show that this reliance on CNNs is not necessary and a pure transformer applied directly to sequences of image patches can perform very well on image classification tasks. When pre-trained on large amounts of data and transferred to multiple mid-sized or small image recognition benchmarks (ImageNet, CIFAR-100, VTAB, etc.), Vision Transformer (ViT)

attains excellent results compared to state-of-the-art convolutional networks while requiring substantially fewer computational resources to train.

76.3 使用方式

推理图片

```
>>> import torch
>>> from mmcls.apis import init_model, inference_model
>>>
>>> model = init_model('configs/vision_transformer/vit-base-p16_pt-32xb128-mae_in1k-
↪224.py', 'https://download.openmmlab.com/mmclassification/v0/vit/vit-base-p16_pt-
↪32xb128-mae_in1k_20220623-4c544545.pth')
>>> predict = inference_model(model, 'demo/demo.JPEG')
>>> print(predict['pred_class'])
sea snake
>>> print(predict['pred_score'])
0.9184340238571167
```

调用模型

```
>>> import torch
>>> from mmcls.apis import init_model
>>>
>>> model = init_model('configs/vision_transformer/vit-base-p16_pt-32xb128-mae_in1k-
↪224.py', 'https://download.openmmlab.com/mmclassification/v0/vit/vit-base-p16_pt-
↪32xb128-mae_in1k_20220623-4c544545.pth')
>>> inputs = torch.rand(1, 3, 224, 224).to(model.data_preprocessor.device)
>>> # To get classification scores.
>>> out = model(inputs)
>>> print(out.shape)
torch.Size([1, 1000])
>>> # To extract features.
>>> outs = model.extract_feat(inputs)
>>> # The patch token features
>>> print(outs[0][0].shape)
torch.Size([1, 768, 14, 14])
>>> # The cls token features
>>> print(outs[0][1].shape)
torch.Size([1, 768])
```

训练/测试

将 ImageNet 数据集放置在 `data/imagenet` 目录下，或者根据 docs 准备其他数据集。

训练：

```
python tools/train.py configs/vision_transformer/vit-base-p16_pt-32xb128-mae_in1k-224.
↪py
```

测试：

```
python tools/test.py configs/vision_transformer/vit-base-p16_pt-32xb128-mae_in1k-224.
↪py https://download.openmmlab.com/mmclassification/v0/vit/vit-base-p16_pt-32xb128-
↪mae_in1k_20220623-4c544545.pth
```

For more configurable parameters, please refer to the API.

76.4 结果和模型

The training step of Vision Transformers is divided into two steps. The first step is training the model on a large dataset, like ImageNet-21k, and get the pre-trained model. And the second step is training the model on the target dataset, like ImageNet-1k, and get the fine-tuned model. Here, we provide both pre-trained models and fine-tuned models.

76.4.1 ImageNet-21k

The pre-trained models on ImageNet-21k are used to fine-tune, and therefore don’t have evaluation results.

模型	分辨率	参数量（M）	Flops(G)	下载
ViT-B16*	224x224	86.86	33.03	model
ViT-B32*	224x224	88.30	8.56	model
ViT-L16*	224x224	304.72	116.68	model

*Models with * are converted from the official repo.*

76.4.2 ImageNet-1k

模型	预训练	分辨率	参数量(M)	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
ViT-B16	从头训练	224x224	86.86	33.03	82.37	96.15	config	model | log
ViT-B16*	ImageNet-21k	384x384	86.86	33.03	85.43	97.77	config	model
ViT-B16 (IPU)	ImageNet-21k	224x224	86.86	33.03	81.22	95.56	config	model | log
ViT-B32*	ImageNet-21k	384x384	88.30	8.56	84.01	97.08	config	model
ViT-L16*	ImageNet-21k	384x384	304.72	116.68	85.63	97.63	config	model

*Models with * are converted from the official repo. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

76.5 引用

```
@inproceedings{
  dosovitskiy2021an,
  title={An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale},
  author={Alexey Dosovitskiy and Lucas Beyer and Alexander Kolesnikov and Dirk␣
↪Weissenborn and Xiaohua Zhai and Thomas Unterthiner and Mostafa Dehghani and␣
↪Matthias Minderer and Georg Heigold and Sylvain Gelly and Jakob Uszkoreit and Neil␣
↪Houlsby},
  booktitle={International Conference on Learning Representations},
  year={2021},
  url={https://openreview.net/forum?id=YicbFdNTTy}
}
```

CHAPTER 77

Wide-ResNet

Wide Residual Networks

77.1 摘要

Deep residual networks were shown to be able to scale up to thousands of layers and still have improving performance. However, each fraction of a percent of improved accuracy costs nearly doubling the number of layers, and so training very deep residual networks has a problem of diminishing feature reuse, which makes these networks very slow to train. To tackle these problems, in this paper we conduct a detailed experimental study on the architecture of ResNet blocks, based on which we propose a novel architecture where we decrease depth and increase width of residual networks. We call the resulting network structures wide residual networks (WRNs) and show that these are far superior over their commonly used thin and very deep counterparts. For example, we demonstrate that even a simple 16-layer-deep wide residual network outperforms in accuracy and efficiency all previous deep residual networks, including thousand-layer-deep networks, achieving new state-of-the-art results on CIFAR, SVHN, COCO, and significant improvements on ImageNet.

77.2 结果和模型

77.2.1 ImageNet-1k

模型	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件	下载
WRN-50*	68.88	11.44	78.48	94.08	config	model
WRN-101*	126.89	22.81	78.84	94.28	config	model
WRN-50 (timm)*	68.88	11.44	81.45	95.53	config	model

*Models with * are converted from the TorchVision and TIMM. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

77.3 引用

```
@INPROCEEDINGS{Zagoruyko2016WRN,
    author = {Sergey Zagoruyko and Nikos Komodakis},
    title = {Wide Residual Networks},
    booktitle = {BMVC},
    year = {2016}}
```

CHAPTER 78

XCiT

XCiT: Cross-Covariance Image Transformers

78.1 摘要

Following their success in natural language processing, transformers have recently shown much promise for computer vision. The self-attention operation underlying transformers yields global interactions between all tokens ,i.e. words or image patches, and enables flexible modelling of image data beyond the local interactions of convolutions. This flexibility, however, comes with a quadratic complexity in time and memory, hindering application to long sequences and high-resolution images. We propose a "transposed" version of self-attention that operates across feature channels rather than tokens, where the interactions are based on the cross-covariance matrix between keys and queries. The resulting cross-covariance attention (XCA) has linear complexity in the number of tokens, and allows efficient processing of high-resolution images. Our cross-covariance image transformer (XCiT) is built upon XCA. It combines the accuracy of conventional transformers with the scalability of convolutional architectures. We validate the effectiveness and generality of XCiT by reporting excellent results on multiple vision benchmarks, including image classification and self-supervised feature learning on ImageNet-1k, object detection and instance segmentation on COCO, and semantic segmentation on ADE20k.

78.2 结果和模型

78.2.1 ImageNet-1k

模型	预训练	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件
xcit-nano-12-p16_3rdparty_in1k*	从头训练	3.05	0.56	70.35	89.98	config
xcit-nano-12-p16_3rdparty-dist_in1k*	Distillation	3.05	0.56	72.36	91.02	config
xcit-nano-12-p16_3rdparty-dist_in1k-384px*	Distillation	3.05	1.64	74.93	92.42	config
xcit-nano-12-p8_3rdparty_in1k*	从头训练	3.05	2.16	73.80	92.08	config
xcit-nano-12-p8_3rdparty-dist_in1k*	Distillation	3.05	2.16	76.17	93.08	config
xcit-nano-12-p8_3rdparty-dist_in1k-384px*	Distillation	3.05	6.34	77.69	94.09	config
xcit-tiny-12-p16_3rdparty_in1k*	从头训练	6.72	1.24	77.21	93.62	config
xcit-tiny-12-p16_3rdparty-dist_in1k*	Distillation	6.72	1.24	78.70	94.12	config
xcit-tiny-24-p16_3rdparty_in1k*	从头训练	12.12	2.34	79.47	94.85	config
xcit-tiny-24-p16_3rdparty-dist_in1k*	Distillation	12.12	2.34	80.51	95.17	config
xcit-tiny-12-p16_3rdparty-dist_in1k-384px*	Distillation	6.72	3.64	80.58	95.38	config
xcit-tiny-12-p8_3rdparty_in1k*	从头训练	6.71	4.81	79.75	94.88	config
xcit-tiny-12-p8_3rdparty-dist_in1k*	Distillation	6.71	4.81	81.26	95.46	config
xcit-tiny-24-p8_3rdparty_in1k*	从头训练	12.11	9.21	81.70	95.90	config
xcit-tiny-24-p8_3rdparty-dist_in1k*	Distillation	12.11	9.21	82.62	96.16	config
xcit-tiny-12-p8_3rdparty-dist_in1k-384px*	Distillation	6.71	14.13	82.46	96.22	config
xcit-tiny-24-p16_3rdparty-dist_in1k-384px*	Distillation	12.12	6.87	82.43	96.20	config
xcit-tiny-24-p8_3rdparty-dist_in1k-384px*	Distillation	12.11	27.05	83.77	96.72	config
xcit-small-12-p16_3rdparty_in1k*	从头训练	26.25	4.81	81.87	95.77	config
xcit-small-12-p16_3rdparty-dist_in1k*	Distillation	26.25	4.81	83.12	96.41	config
xcit-small-24-p16_3rdparty_in1k*	从头训练	47.67	9.10	82.38	95.93	config
xcit-small-24-p16_3rdparty-dist_in1k*	Distillation	47.67	9.10	83.70	96.61	config
xcit-small-12-p16_3rdparty-dist_in1k-384px*	Distillation	26.25	14.14	84.74	97.19	config
xcit-small-12-p8_3rdparty_in1k*	从头训练	26.21	18.69	83.21	96.41	config
xcit-small-12-p8_3rdparty-dist_in1k*	Distillation	26.21	18.69	83.97	96.81	config
xcit-small-24-p16_3rdparty-dist_in1k-384px*	Distillation	47.67	26.72	85.10	97.32	config
xcit-small-24-p8_3rdparty_in1k*	从头训练	47.63	35.81	83.62	96.51	config
xcit-small-24-p8_3rdparty-dist_in1k*	Distillation	47.63	35.81	84.68	97.07	config
xcit-small-12-p8_3rdparty-dist_in1k-384px*	Distillation	26.21	54.92	85.12	97.31	config
xcit-small-24-p8_3rdparty-dist_in1k-384px*	Distillation	47.63	105.24	85.57	97.60	config
xcit-medium-24-p16_3rdparty_in1k*	从头训练	84.40	16.13	82.56	95.82	config
xcit-medium-24-p16_3rdparty-dist_in1k*	Distillation	84.40	16.13	84.15	96.82	config
xcit-medium-24-p16_3rdparty-dist_in1k-384px*	Distillation	84.40	47.39	85.47	97.49	config

下

表 1 – 续上页

模型	预训练	参数量（M）	Flops(G)	Top-1 (%)	Top-5 (%)	配置文件
xcit-medium-24-p8_3rdparty_in1k*	从头训练	84.32	63.52	83.61	96.23	config
xcit-medium-24-p8_3rdparty-dist_in1k*	Distillation	84.32	63.52	85.00	97.16	config
xcit-medium-24-p8_3rdparty-dist_in1k-384px*	Distillation	84.32	186.67	85.87	97.61	config
xcit-large-24-p16_3rdparty_in1k*	从头训练	189.10	35.86	82.97	95.86	config
xcit-large-24-p16_3rdparty-dist_in1k*	Distillation	189.10	35.86	84.61	97.07	config
xcit-large-24-p16_3rdparty-dist_in1k-384px*	Distillation	189.10	105.35	85.78	97.60	config
xcit-large-24-p8_3rdparty_in1k*	从头训练	188.93	141.23	84.23	96.58	config
xcit-large-24-p8_3rdparty-dist_in1k*	Distillation	188.93	141.23	85.14	97.32	config
xcit-large-24-p8_3rdparty-dist_in1k-384px*	Distillation	188.93	415.00	86.13	97.75	config

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

78.3 引用

```
@article{el2021xcit,
  title={XCiT: Cross-Covariance Image Transformers},
  author={El-Nouby, Alaaeldin and Touvron, Hugo and Caron, Mathilde and Bojanowski,␣
↪Piotr and Douze, Matthijs and Joulin, Armand and Laptev, Ivan and Neverova, Natalia␣
↪and Synnaeve, Gabriel and Verbeek, Jakob and others},
  journal={arXiv preprint arXiv:2106.09681},
  year={2021}
}
```

CHAPTER 79

从 MMClassification 0.x 迁移

我们在 MMClassification 1.x 版本中引入了一些修改，可能会产生兼容性问题。请按照本教程从 MMClassification 0.x 迁移您的项目。

79.1 新的依赖

MMClassification 1.x 依赖一些新的包。你可以准备一个干净的新环境，并按照安装教程重新安装；或者手动安装以下软件包。

1. MMEngine：MMEngine 是 OpenMMLab 2.0 架构的核心库，我们将许多与计算机视觉无关的组件从 MMCV 拆分到了 MMEngine。
2. MMCV：OpenMMLab 计算机视觉基础库，这不是一个新的依赖，但你需要将其升级到 `2.0.0rc1` 版本以上。
3. rich：一个命令行美化库，用以在命令行中呈现更美观的输出。

79.2 配置文件

在 MMClassification 1.x 中，我们重构了配置文件的结构，绝大部分原来的配置文件无法直接使用。

在本节中，我们将介绍配置文件的所有变化。我们假设您已经对配置文件有所了解。

79.2.1 模型设置

model.backbone、model.neck 和 model.head 字段没有变化。

model.train_cfg 字段的变化：

- BatchMixup 被重命名为 *Mixup*
- BatchCutMix 被重命名为 *CutMix*
- BatchResizeMix 被重命名为 *ResizeMix*
- 以上增强中的 prob 参数均被移除，现在在 train_cfg 中使用一个统一的 probs 字段指定每个增强的概率。如果没有指定 probs 字段，现在将均匀地随机选择一种增强。

```
model = dict(
    ...
    train_cfg=dict(augments=[
        dict(type='BatchMixup', alpha=0.8, num_classes=1000, prob=0.5),
        dict(type='BatchCutMix', alpha=1.0, num_classes=1000, prob=0.5)
    ]
)
```

```
model = dict(
    ...
    train_cfg=dict(augments=[
        dict(type='Mixup', alpha=0.8),
        dict(type='CutMix', alpha=1.0),
    ]
)
```

79.2.2 数据设置

data 字段的变化：

- 原先的 data 字段被拆分为 train_dataloader，val_dataloader 和 test_dataloader 字段。这允许我们进行更加细粒度的配置。比如在训练和测试中指定不同的采样器、批次大小等。
- samples_per_gpu 字段被重命名为 batch_size
- workers_per_gpu 字段被重命名为 num_workers

```
data = dict(
    samples_per_gpu=32,
    workers_per_gpu=2,
    train=dict(...),
    val=dict(...),
```

(下页继续)

(续上页)

```
    test=dict(...),
)
```

```
train_dataloader = dict(
    batch_size=32,
    num_workers=2,
    dataset=dict(...),
    sampler=dict(type='DefaultSampler', shuffle=True)   # 必要的
)

val_dataloader = dict(
    batch_size=32,
    num_workers=2,
    dataset=dict(...),
    sampler=dict(type='DefaultSampler', shuffle=False)   # 必要的
)

test_dataloader = val_dataloader
```

pipeline 字段的变化：

- 原先的 **ToTensor**、**ImageToTensor** 和 **Collect** 被合并为 *PackClsInputs*
- 我们建议去除数据集流水线中的 **Normalize** 变换，转而使用 data_preprocessor 字段进行归一化预处理。
- **RandomFlip** 中的 flip_prob 参数被重命名为 flip
- ***RandomCrop*** 中的 size 参数被重命名为 crop_size
- ***RandomResizedCrop*** 中的 size 参数被重命名为 scale
- **Resize** 中的 size 参数被重命名为 scale。并且不再支持形如 (256, -1) 的尺寸，请使用 *ResizeEdge*
- ***AutoAugment*** 和 ***RandAugment*** 中的 policies 参数现在支持使用字符串来指定某些预设的策略集，AutoAugment 支持 “imagenet”，RandAugment 支持 “timm_increasing”
- **RandomResizedCrop** 和 **CenterCrop** 不再支持 efficientnet_style 参数，请使用 *EfficientNetRandomCrop* 和*EfficientNetCenterCrop*

备注： 我们将一些数据变换工作移至数据预处理器进行，如归一化，请参阅文档了解更多详细信息。

```
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
```

(下页继续)

(续上页)

```
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='RandomResizedCrop', size=224),
    dict(type='RandomFlip', flip_prob=0.5, direction='horizontal'),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='ImageToTensor', keys=['img']),
    dict(type='ToTensor', keys=['gt_label']),
    dict(type='Collect', keys=['img', 'gt_label'])
]
```

```
data_preprocessor = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)

train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='RandomResizedCrop', scale=224),
    dict(type='RandomFlip', prob=0.5, direction='horizontal'),
    dict(type='PackClsInputs'),
]
```

evaluation 字段的变化：

- 原先的 **evaluation** 字段被拆分为 `val_evaluator` 和 `test_evaluator`，并且不再支持 `interval` 和 `save_best` 参数。`interval` 参数被移动至 `train_cfg.val_interval` 字段，详见训练策略配置。而 `save_best` 参数被移动至 `default_hooks.checkpoint.save_best` 字段，详见运行设置。
- ‘accuracy’ 指标被重命名为 *Accuracy*
- ‘precision’，’ recall’，’ f1-score’ 和 ‘support’ 指标被组合为 *SingleLabelMetric*，并使用 `items` 参数指定具体计算哪些指标。
- ‘mAP’ 指标被重命名为 *AveragePrecision*
- ‘CP’，’ CR’，’ CF1’，’ OP’，’ OR’ 和 ‘OF1’ 指标被组合为 *MultiLabelMetric*，并使用 `items` 和 `average` 参数指定具体计算哪些指标。

```
evaluation = dict(
    interval=1,
    metric='accuracy',
    metric_options=dict(topk=(1, 5))
)
```

```
val_evaluator = dict(type='Accuracy', topk=(1, 5))
test_evaluator = val_evaluator
```

```
evaluation = dict(
    interval=1,
    metric=['mAP', 'CP', 'OP', 'CR', 'OR', 'CF1', 'OF1'],
    metric_options=dict(thr=0.5),
)
```

```
val_evaluator = [
    dict(type='AveragePrecision'),
    dict(type='MultiLabelMetric',
         items=['precision', 'recall', 'f1-score'],
         average='both',
         thr=0.5),
]
test_evaluator = val_evaluator
```

79.2.3 训练策略设置

optimizer 和 **optimizer_config** 字段的变化：

- 现在我们使用 optim_wrapper 字段指定与优化过程有关的所有配置。而 optimizer 字段是 optim_wrapper 的一个子字段。
- paramwise_cfg 字段不再是 optimizer 的子字段，而是 optim_wrapper 的子字段。
- optimizer_config 字段被移除，其配置项被移入 optim_wrapper 字段。
- grad_clip 被重命名为 clip_grad

```
optimizer = dict(
    type='AdamW',
    lr=0.0015,
    weight_decay=0.3,
    paramwise_cfg = dict(
         norm_decay_mult=0.0,
         bias_decay_mult=0.0,
    ))

optimizer_config = dict(grad_clip=dict(max_norm=1.0))
```

```
optim_wrapper = dict(
    optimizer=dict(type='AdamW', lr=0.0015, weight_decay=0.3),
```

(下页继续)

(续上页)

```
    paramwise_cfg = dict(
        norm_decay_mult=0.0,
        bias_decay_mult=0.0,
    ),
    clip_grad=dict(max_norm=1.0),
)
```

lr_config 字段的变化：

- lr_config 字段被移除，我们使用新的 param_scheduler 配置取代。
- warmup 相关的字段都被移除，因为学习率预热可以通过多个学习率规划器的组合来实现，因此不再单独实现。

新的优化器参数规划器组合机制非常灵活，你可以使用它来设计多种学习率、动量曲线，详见MMEngine 中的教程。

```
lr_config = dict(
    policy='CosineAnnealing',
    min_lr=0,
    warmup='linear',
    warmup_iters=5,
    warmup_ratio=0.01,
    warmup_by_epoch=True)
```

```
param_scheduler = [
    # 学习率预热
    dict(
        type='LinearLR',
        start_factor=0.01,
        by_epoch=True,
        end=5,
        # 每轮迭代都更新学习率，而不是每个 epoch
        convert_to_iter_based=True),
    # 主学习率规划器
    dict(type='CosineAnnealingLR', by_epoch=True, begin=5),
]
```

runner 字段的变化：

原 runner 字段被拆分为 train_cfg，val_cfg 和 test_cfg 三个字段，分别配置训练、验证和测试循环。

```
runner = dict(type='EpochBasedRunner', max_epochs=100)
```

```
# `val_interval` 字段来自原配置中 `evaluation.interval` 字段
train_cfg = dict(by_epoch=True, max_epochs=100, val_interval=1)
val_cfg = dict()   # 空字典表示使用默认验证配置
test_cfg = dict()  # 空字典表示使用默认测试配置
```

在 OpenMMLab 2.0 中，我们引入了“循环控制器”来控制训练、验证和测试行为，而原先 Runner 功能也相应地发生了变化。详细介绍参见 MMEngine 中的执行器教程。

79.2.4 运行设置

checkpoint_config 和 **log_config** 字段的变化：

checkpoint_config 被移动至 default_hooks.checkpoint，log_config 被移动至 default_hooks.logger。同时,我们将很多原先在训练脚本中隐式定义的钩子移动到了 default_hooks 字段。

```
default_hooks = dict(
    # 记录每轮迭代的耗时
    timer=dict(type='IterTimerHook'),

    # 每 100 轮迭代打印一次日志
    logger=dict(type='LoggerHook', interval=100),

    # 启用优化器参数规划器
    param_scheduler=dict(type='ParamSchedulerHook'),

    # 每个 epoch 保存一次模型权重文件，并且自动保存最优权重文件
    checkpoint=dict(type='CheckpointHook', interval=1, save_best='auto'),

    # 在分布式环境中设置采样器种子
    sampler_seed=dict(type='DistSamplerSeedHook'),

    # 可视化验证结果，将 `enable` 设为 True 来启用这一功能。
    visualization=dict(type='VisualizationHook', enable=False),
)
```

此外，我们将原来的日志功能拆分为日志记录和可视化器。日志记录负责按照指定间隔保存日志数据，以及进行数据平滑等处理，可视化器用于在不同的后端记录日志，如终端、TensorBoard 和 WandB。

```
log_config = dict(
    interval=100,
    hooks=[
        dict(type='TextLoggerHook'),
```

(下页继续)

(续上页)

```
        dict(type='TensorboardLoggerHook'),
    ])
```

```
default_hooks = dict(
    ...
    logger=dict(type='LoggerHook', interval=100),
)

visualizer = dict(
    type='ClsVisualizer',
    vis_backends=[dict(type='LocalVisBackend'), dict(type='TensorboardVisBackend')],
)
```

load_from 和 **resume_from** 字段的变动：

- resume_from 字段被移除。我们现在使用 resume 和 load_from 字段实现以下功能：
  - 如 resume=True 且 load_from 不为 None，从 load_from 指定的权重文件恢复训练。
  - 如 resume=True 且 load_from 为 None，尝试从工作目录中最新的权重文件恢复训练。
  - 如 resume=False 且 load_from 不为 None，仅加载指定的权重文件，不恢复训练。
  - 如 resume=False 且 load_from 为 None，不进行任何操作。

dist_params 字段的变动：dist_params 字段被移动为 env_cfg 字段的一个子字段。以下为 env_cfg 字段的所有配置项：

```
env_cfg = dict(
    # 是否启用 cudnn benchmark
    cudnn_benchmark=False,

    # 设置多进程相关参数
    mp_cfg=dict(mp_start_method='fork', opencv_num_threads=0),

    # 设置分布式相关参数
    dist_cfg=dict(backend='nccl'),
)
```

workflow 字段的变动：workflow 相关的功能现已被移除。

新字段 **visualizer**：可视化器是 OpenMMLab 2.0 架构中的新设计，我们使用可视化器进行日志、结果的可视化与多后端的存储。详见 MMEngine 中的可视化教程。

```
visualizer = dict(
    type='ClsVisualizer',
    vis_backends=[
```

(下页继续)

(续上页)

```
        dict(type='LocalVisBackend'),
        # 将下行取消注释，即可将日志和可视化结果保存至 TesnorBoard
        # dict(type='TensorboardVisBackend')
    ]
)
```

新字段 **default_scope**：指定所有注册器进行模块搜索默认的起点。MMClassification 中的 default_scope 字段为 mmcls，大部分情况下不需要修改。详见 MMengine 中的注册器教程。

79.3 模块变动

79.3.1 mmcls.apis

详见包文档

函数	变动
init_model	无变动
inference_model	无变动，但我们推荐使用功能更强的*mmcls.ImageClassificationInferencer*。
train_model	移除，直接使用 runner.train 进行训练。
multi_gpu_test	移除，直接使用 runner.test 进行测试。
single_gpu_test	移除，直接使用 runner.test 进行测试。
show_result_pyplot	移除，使用*mmcls.ImageClassificationInferencer* 进行模型推理和结果可视化。
set_random_seed	移除，使用 mmengine.runner.set_random_seed.
init_random_seed	移除，使用 mmengine.dist.sync_random_seed.

79.3.2 mmcls.core

mmcls.core 包被重命名为*mmcls.engine*

子包	变动
evaluation	移除，使用 *mmcls.evaluation*
hook	移动至*mmcls.engine.hooks*
optimizers	移动至*mmcls.engine.optimizers*
utils	移除，分布式环境相关的函数统一至 mmengine.dist 包
visualization	移除，其中可视化相关的功能被移动至 *mmcls.visualization.ClsVisualizer*

hooks 包中的 MMClsWandbHook 尚未实现。

hooks 包中的 CosineAnnealingCooldownLrUpdaterHook 被移除。我们现在支持使用学习率规划器的组合实现该功能。详见自定义训练优化策略。

79.3.3 mmcls.datasets

详见包文档

数据集类	变动
CustomDataset	增加了 data_root 参数，作为 data_prefix 和 ann_file 的共同根路径。
ImageNet	与 CustomDataset 相同。
ImageNet21k	与 CustomDataset 相同。
CIFAR10 & *CIFAR100*	test_mode 参数目前是必要参数。
MNIST & *FashionMNIST*	test_mode 参数目前是必要参数。
VOC	现在需要指定 data_root，image_set_path 和 test_mode 参数。
CUB	现在需要指定 data_root 和 test_mode 参数。

mmcls.datasets.pipelines 包被重命名为 mmcls.datasets.transforms

数据变换类	变动
LoadImageFromFile	移除，使用 mmcv.transforms.LoadImageFromFile
RandomFlip	移除，使用 mmcv.transforms.RandomFlip，其中 flip_prob 参数被重命名为 prob
RandomCrop	size 参数被重命名为 crop_size
RandomResizedCrop	size 参数被重命名为 scale；scale 参数被重命名为 crop_ratio_range；不再支持 efficientnet_style，请使用 *EfficientNetRandomCrop*
CenterCrop	移除，使用 mmcv.transforms.CenterCrop；不再支持 efficientnet_style，请使用 *EfficientNetCenterCrop*
Resize	移除，使用 mmcv.transforms.Resize；size 参数被重命名为 scale，且不再支持形如 (256, -1) 参数，使用 *ResizeEdge*
AutoAugment & RandomAugment	policies 参数现在支持使用字符串指定预设的策略集。
Compose	移除，使用 mmcv.transforms.Compose

79.3.4 mmcls.models

详见包文档，**backbones**、**necks** 和 **losses** 的结构没有变动。

ImageClassifier 的变动：

分类器的方法	变动
extract_feat	无变动
forward	现在需要三个输入：inputs、data_samples 和 mode。详见文档
forward_train	变更为 loss 方法。
simple_test	变更为 predict 方法。
train_step	optimizer 参数被修改为 optim_wrapper，接受 OptimWrapper
val_step	原先的 val_step 与 train_step 一致，现在该方法将会调用 predict
test_step	新方法，与 val_step 一致。

heads 中的变动：

分类头的方法	变动
pre_logits	无变动
forward_train	变更为 loss 方法。
simple_test	变更为 predict 方法。
loss	现在接受 data_samples 参数，而不是 gt_labels，data_samples 参数应当接受 *ClsDataSample* 的列表。
forward	新方法，它将返回分类头的输出，不会进行任何后处理（包括 softmax 或 sigmoid）。

79.3.5 mmcls.utils

详见包文档

函数	变动
collect_env	无变动
get_root_logger	移除，使用 mmengine.logging.MMLogger.get_current_instance
load_json_log	输出格式发生变化。
setup_multi_processes	移除，使用 mmengine.utils.dl_utils.set_multi_processing
wrap_non_distributed_model	移除，现在 runner 会自动包装模型。
wrap_distributed_model	移除，现在 runner 会自动包装模型。
auto_select_device	移除，现在 runner 会自动选择设备。

79.3.6 其他变动

- 我们将所有注册器的定义从各个包移动到了 `mmcls.registry` 包。

CHAPTER 80

mmcls.apis

该包提供了一些用于分类任务的高阶 API

mmcls.apis

- *Model*
- 推理

80.1 Model

list_models	List all models available in MMClassification.
get_model	Get a pre-defined model by the name of model.
init_model	从配置文件初始化一个分类器

80.1.1 mmcls.apis.list_models

mmcls.apis.**list_models**(*pattern=None*)

List all models available in MMClassification.

参数 **pattern** (*str | None*) –A wildcard pattern to match model names.

返回 a list of model names.

返回类型 List[str]

使用示例

List all models:

```
>>> from mmcls import list_models
>>> print(list_models())
```

List ResNet-50 models on ImageNet-1k dataset:

```
>>> from mmcls import list_models
>>> print(list_models('resnet*in1k'))
['resnet50_8xb32_in1k',
 'resnet50_8xb32-fp16_in1k',
 'resnet50_8xb256-rsb-a1-600e_in1k',
 'resnet50_8xb256-rsb-a2-300e_in1k',
 'resnet50_8xb256-rsb-a3-100e_in1k']
```

80.1.2 mmcls.apis.get_model

mmcls.apis.**get_model**(*model_name*, *pretrained=False*, *device=None*, ***kwargs*)

Get a pre-defined model by the name of model.

参数

- **model_name** (*str*) –The name of model.
- **pretrained** (*bool | str*) –If True, load the pre-defined pretrained weights. If a string, load the weights from it. Defaults to False.
- **device** (*str | torch.device | None*) –Transfer the model to the target device. Defaults to None.
- ****kwargs** –Other keyword arguments of the model config.

返回 The result model.

返回类型 mmengine.model.BaseModel

使用示例

Get a ResNet-50 model and extract images feature:

```
>>> import torch
>>> from mmcls import get_model
>>> inputs = torch.rand(16, 3, 224, 224)
>>> model = get_model('resnet50_8xb32_in1k', pretrained=True, backbone=dict(out_
↪indices=(0, 1, 2, 3)))
>>> feats = model.extract_feat(inputs)
>>> for feat in feats:
...     print(feat.shape)
torch.Size([16, 256])
torch.Size([16, 512])
torch.Size([16, 1024])
torch.Size([16, 2048])
```

Get Swin-Transformer model with pre-trained weights and inference:

```
>>> from mmcls import get_model, inference_model
>>> model = get_model('swin-base_16xb64_in1k', pretrained=True)
>>> result = inference_model(model, 'demo/demo.JPEG')
>>> print(result['pred_class'])
'sea snake'
```

80.1.3 mmcls.apis.init_model

mmcls.apis.**init_model**(*config*, *checkpoint=None*, *device=None*, ***kwargs*)

从配置文件初始化一个分类器

参数

- **config** (str | mmengine.Config) –Config file path or the config object.
- **checkpoint** (*str, optional*) –Checkpoint path. If left as None, the model will not load any weights.
- **device** (*str | torch.device | None*) –Transfer the model to the target device. Defaults to None.
- ****kwargs** –Other keyword arguments of the model config.

返回 The constructed model.

返回类型 nn.Module

80.2 推理

ImageClassificationInferencer	The inferencer for image classification.

80.2.1 ImageClassificationInferencer

class mmcls.apis.**ImageClassificationInferencer**(*model*, *weights=None*, *device=None*, *classes=None*)

The inferencer for image classification.

参数

- **model** (*BaseModel | str | Config*) –A model name or a path to the confi file, or a BaseModel object. The model name can be found by ImageClassificationInferencer.list_models() and you can also query it in 模型库统计.
- **weights** (*str, optional*) –Path to the checkpoint. If None, it will try to find a pre-defined weight from the model you specified (only work if the model is a model name). Defaults to None.
- **device** (*str, optional*) –Device to run inference. If None, use CPU or the device of the input model. Defaults to None.

示例

1. Use a pre-trained model in MMClassification to inference an image.

```
>>> from mmcls import ImageClassificationInferencer
>>> inferencer = ImageClassificationInferencer('resnet50_8xb32_in1k')
>>> inferencer('demo/demo.JPEG')
[{'pred_score': array([...]),
  'pred_label': 65,
  'pred_score': 0.6649367809295654,
  'pred_class': 'sea snake'}]
```

2. Use a config file and checkpoint to inference multiple images on GPU, and save the visualization results in a folder.

```
>>> from mmcls import ImageClassificationInferencer
>>> inferencer = ImageClassificationInferencer(
        model='configs/resnet/resnet50_8xb32_in1k.py',
        weights='https://download.openmmlab.com/mmclassification/v0/resnet/
↪resnet50_8xb32_in1k_20210831-ea4938fc.pth',
```

(下页继续)

(续上页)

```
        device='cuda')
>>> inferencer(['demo/dog.jpg', 'demo/bird.JPEG'], show_dir="./visualize/")
```

__call__(*inputs*, *return_datasamples=False*, *batch_size=1*, ***kwargs*)

Call the inferencer.

参数

- **inputs** (*InputsType*) –Inputs for the inferencer.
- **return_datasamples** (*bool*) –Whether to return results as `BaseDataElement`. Defaults to False.
- **batch_size** (*int*) –Batch size. Defaults to 1.
- **rescale_factor** (*float, optional*) –Rescale the image by the rescale factor for visualization. This is helpful when the image is too large or too small for visualization. Defaults to None.
- **draw_score** (*bool*) –Whether to draw the prediction scores of prediction categories. Defaults to True.
- **show** (*bool*) –Whether to display the visualization result in a window. Defaults to False.
- **show_dir** (*str, optional*) –If not None, save the visualization results in the specified directory. Defaults to None.

返回 The inference results.

返回类型 list

static list_models(*pattern=None*)

List all available model names.

参数 **pattern** (*str | None*) –A wildcard pattern to match model names.

返回 a list of model names.

返回类型 List[str]

inference_model	Inference an image with the classifier.

80.2.2 mmcls.apis.inference_model

mmcls.apis.**inference_model**(*model*, *img*, *device=None*)

Inference an image with the classifier.

参数

- **model** (*BaseModel | str | Config*) –The loaded classifier or the model name or the config of the model.
- **img** (*str | ndarray*) –The image filename or loaded image.
- **device** (*str, optional*) –Device to run inference. If None, use CPU or the device of the input model. Defaults to None.

返回

The classification results that contains:

- `pred_scores`: The classification scores of all categories.
- `pred_class`: The predicted category.
- `pred_label`: The predicted index of the category.
- `pred_score`: The score of the predicted category.

返回类型 result (dict)

备注: This function is reserved for compatibility and demo on a single image. We suggest to use *ImageClassificationInferencer*, which is more powerful and configurable.

CHAPTER 81

mmcls.engine

该包中包含了一些运行时组件，如钩子（hook）、执行器（runner）、优化器（optimizer）和循环执行器（loop）。这些组件在分类任务中需要用到，而还未被 MMEngine 支持。

备注： 部分组件未来可能会被移动到 MMEngine 中。

mmcls.engine

81.1 Hooks

ClassNumCheckHook	Class Number Check HOOK.
PreciseBNHook	Precise BN hook.
VisualizationHook	Classification Visualization Hook.
PrepareProtoBeforeValLoopHook	The hook to prepare the prototype in retrievers.
SetAdaptiveMarginsHook	Set adaptive-margins in ArcFaceClsHead based on the power of category-wise count.
EMAHook	A Hook to apply Exponential Moving Average (EMA) on the model during training.

81.1.1 ClassNumCheckHook

class mmcls.engine.hooks.**ClassNumCheckHook**

Class Number Check HOOK.

before_test(*runner*)

Check whether the test dataset is compatible with head.

参数 **(obj** (*runner*) –*IterBasedRunner*): Iter based Runner.

before_train(*runner*)

Check whether the training dataset is compatible with head.

参数 **(obj** (*runner*) –*IterBasedRunner*): Iter based Runner.

before_val(*runner*)

Check whether the validation dataset is compatible with head.

参数 **(obj** (*runner*) –*IterBasedRunner*): Iter based Runner.

81.1.2 PreciseBNHook

class mmcls.engine.hooks.**PreciseBNHook**(*num_samples=8192*, *interval=1*)

Precise BN hook.

Recompute and update the batch norm stats to make them more precise. During training both BN stats and the weight are changing after every iteration, so the running average can not precisely reflect the actual stats of the current model.

With this hook, the BN stats are recomputed with fixed weights, to make the running average more precise. Specifically, it computes the true average of per-batch mean/variance instead of the running average. See Sec. 3 of the paper *Rethinking Batch in BatchNorm <https://arxiv.org/abs/2105.07576>* for details.

This hook will update BN stats, so it should be executed before `CheckpointHook` and `EMAHook`, generally set its priority to "ABOVE_NORMAL".

参数

- **num_samples** (*int*) –The number of samples to update the bn stats. Defaults to 8192.
- **interval** (*int*) –Perform precise bn interval. If the train loop is
- **by_epoch=True** (*EpochBasedTrainLoop or*) –train loop is *IterBasedTrainLoop* or *by_epoch=False*, its unit is 'iter'. Defaults to 1.
- **the** (*its unit is 'epoch'; if*) –train loop is *IterBasedTrainLoop* or *by_epoch=False*, its unit is 'iter'. Defaults to 1.

after_train_epoch(*runner*)

Calculate prcise BN and broadcast BN stats across GPUs.

参数 **(obj** (*runner*) –*Runner*): The runner of the training process.

after_train_iter(*runner*, *batch_idx*, *data_batch=None*, *outputs=None*)

Calculate prcise BN and broadcast BN stats across GPUs.

参数

- **(obj** (*runner*) –*Runner*): The runner of the training process.
- **batch_idx** (*int*) –The index of the current batch in the train loop.
- **data_batch** (*Sequence[dict], optional*) –Data from dataloader. Defaults to None.

81.1.3 VisualizationHook

class mmcls.engine.hooks.**VisualizationHook**(*enable=False*, *interval=5000*, *show=False*, *out_dir=None*, ***kwargs*)

Classification Visualization Hook. Used to visualize validation and testing prediction results.

- If `out_dir` is specified, all storage backends are ignored and save the image to the `out_dir`.
- If `show` is True, plot the result image in a window, please confirm you are able to access the graphical interface.

参数

- **enable** (*bool*) –Whether to enable this hook. Defaults to False.
- **interval** (*int*) –The interval of samples to visualize. Defaults to 5000.
- **show** (*bool*) –Whether to display the drawn image. Defaults to False.

- **out_dir** (*str, optional*) –directory where painted images will be saved in the testing process. If None, handle with the backends of the visualizer. Defaults to None.
- ****kwargs** –other keyword arguments of `mmcls.visualization.ClsVisualizer.add_datasample()`.

after_test_iter(*runner*, *batch_idx*, *data_batch*, *outputs*)

Visualize every `self.interval` samples during test.

参数

- **runner** (`Runner`) –The runner of the testing process.
- **batch_idx** (*int*) –The index of the current batch in the test loop.
- **data_batch** (*dict*) –Data from dataloader.
- **outputs** (Sequence[`DetDataSample`]) –Outputs from model.

after_val_iter(*runner*, *batch_idx*, *data_batch*, *outputs*)

Visualize every `self.interval` samples during validation.

参数

- **runner** (`Runner`) –The runner of the validation process.
- **batch_idx** (*int*) –The index of the current batch in the val loop.
- **data_batch** (*dict*) –Data from dataloader.
- **outputs** (Sequence[`ClsDataSample`]) –Outputs from model.

81.1.4 PrepareProtoBeforeValLoopHook

class mmcls.engine.hooks.**PrepareProtoBeforeValLoopHook**

The hook to prepare the prototype in retrievers.

Since the encoders of the retriever changes during training, the prototype changes accordingly. So the *prototype_vecs* needs to be regenerated before validation loop.

81.1.5 SetAdaptiveMarginsHook

class mmcls.engine.hooks.**SetAdaptiveMarginsHook**(*margin_min=0.05*, *margin_max=0.5*, *power=- 0.25*)

Set adaptive-margins in ArcFaceClsHead based on the power of category-wise count.

A PyTorch implementation of paper Google Landmark Recognition 2020 Competition Third Place Solution. The margins will be f$(n) = (marginMax - marginMin) \cdot norm(n^p) + marginMin$. The *n* indicates the number of occurrences of a category.

参数

- **margin_min** (*float*) –Lower bound of margins. Defaults to 0.05.
- **margin_max** (*float*) –Upper bound of margins. Defaults to 0.5.
- **power** (*float*) –The power of category freqercy. Defaults to -0.25.

before_train(*runner*)

change the margins in ArcFaceClsHead.

参数 **(obj** (*runner*) –*Runner*): Runner.

81.1.6 EMAHook

class mmcls.engine.hooks.**EMAHook**(*ema_type='ExponentialMovingAverage'*, *strict_load=False*, *begin_iter=0*, *begin_epoch=0*, *evaluate_on_ema=True*, *evaluate_on_origin=False*, ***kwargs*)

A Hook to apply Exponential Moving Average (EMA) on the model during training.

Comparing with mmengine.hooks.EMAHook, this hook accepts evaluate_on_ema and evaluate_on_origin arguments. By default, the evaluate_on_ema is enabled, and if you want to do validation and testing on both original and EMA models, please set both arguments True.

备注:

- EMAHook takes priority over CheckpointHook.
- The original model parameters are actually saved in ema field after train.
- begin_iter and begin_epoch cannot be set at the same time.

参数

- **ema_type** (*str*) –The type of EMA strategy to use. You can find the supported strategies in mmengine.model.averaged_model. Defaults to 'ExponentialMovingAverage'.
- **strict_load** (*bool*) –Whether to strictly enforce that the keys of state_dict in checkpoint match the keys returned by self.module.state_dict. Defaults to False. Changed in v0.3.0.
- **begin_iter** (*int*) –The number of iteration to enable EMAHook. Defaults to 0.
- **begin_epoch** (*int*) –The number of epoch to enable EMAHook. Defaults to 0.
- **evaluate_on_ema** (*bool*) –Whether to evaluate (validate and test) on EMA model during val-loop and test-loop. Defaults to True.

- **evaluate_on_origin** (*bool*) –Whether to evaluate (validate and test) on the original model during val-loop and test-loop. Defaults to False.
- ****kwargs** –Keyword arguments passed to subclasses of `BaseAveragedModel`

after_load_checkpoint(*runner*, *checkpoint*)

Resume ema parameters from checkpoint.

参数 **runner** (*Runner*) –The runner of the testing process.

after_test_epoch(*runner*, *metrics=None*)

We recover source model's parameter from ema model after test.

参数

- **runner** (*Runner*) –The runner of the testing process.
- **metrics** (*Dict[str, float], optional*) –Evaluation results of all metrics on test dataset. The keys are the names of the metrics, and the values are corresponding results.

after_val_epoch(*runner*, *metrics=None*)

We recover source model's parameter from ema model after validation.

参数

- **runner** (*Runner*) –The runner of the validation process.
- **metrics** (*Dict[str, float], optional*) –Evaluation results of all metrics on validation dataset. The keys are the names of the metrics, and the values are corresponding results.

before_test_epoch(*runner*)

We load parameter values from ema model to source model before test.

参数 **runner** (*Runner*) –The runner of the training process.

before_val_epoch(*runner*)

We load parameter values from ema model to source model before validation.

参数 **runner** (*Runner*) –The runner of the training process.

81.2 Optimizers

Lamb	A pure pytorch variant of FuseLAMB (NvLamb variant) optimizer.

81.2.1 Lamb

class mmcls.engine.optimizers.**Lamb**(*params*, *lr=0.001*, *bias_correction=True*, *betas=(0.9, 0.999)*, *eps=1e-06*, *weight_decay=0.01*, *grad_averaging=True*, *max_grad_norm=1.0*, *trust_clip=False*, *always_adapt=False*)

A pure pytorch variant of FuseLAMB (NvLamb variant) optimizer.

This class is copied from timm. The LAMB was proposed in Large Batch Optimization for Deep Learning - Training BERT in 76 minutes.

参数

- **params** (*iterable*) –iterable of parameters to optimize or dicts defining
- **groups.** (*parameter*) –
- **lr** (*float, optional*) –learning rate. (default: 1e-3)
- **betas** (*Tuple[float, float], optional*) –coefficients used for computing running averages of gradient and its norm. (default: (0.9, 0.999))
- **eps** (*float, optional*) –term added to the denominator to improve numerical stability. (default: 1e-8)
- **weight_decay** (*float, optional*) –weight decay (L2 penalty) (default: 0)
- **grad_averaging** (*bool, optional*) –whether apply (1-beta2) to grad when calculating running averages of gradient. (default: True)
- **max_grad_norm** (*float, optional*) –value used to clip global grad norm (default: 1.0)
- **trust_clip** (*bool*) –enable LAMBC trust ratio clipping (default: False)
- **always_adapt** (*boolean, optional*) –Apply adaptive learning rate to 0.0 weight decay parameter (default: False)

step(*closure=None*)

Performs a single optimization step.

参数 **closure** (*callable, optional*) –A closure that reevaluates the model and returns the loss.

CHAPTER 82

mmcls.datasets

dataset 包中包含了分类任务中常用的数据集，以及一些数据集封装。

mmcls.datasets

82.1 Custom Dataset

class mmcls.datasets.**CustomDataset**(*ann_file=''*, *metainfo=None*, *data_root=''*, *data_prefix=''*, *extensions=('.jpg', '.jpeg', '.png', '.ppm', '.bmp', '.pgm', '.tif')*, *lazy_init=False*, ***kwargs*)

Custom dataset for classification.

The dataset supports two kinds of annotation format.

1. An annotation file is provided, and each line indicates a sample:

   The sample files:

   ```
   data_prefix/
   ├── folder_1
   │   ├── xxx.png
   │   ├── xxy.png
   │   └── ...
   └── folder_2
       ├── 123.png
       ├── nsdf3.png
       └── ...
   ```

   The annotation file (the first column is the image path and the second column is the index of category):

   ```
   folder_1/xxx.png 0
   folder_1/xxy.png 1
   folder_2/123.png 5
   folder_2/nsdf3.png 3
   ...
   ```

   Please specify the name of categories by the argument `classes` or `metainfo`.

2. The samples are arranged in the specific way:

   ```
   data_prefix/
   ├── class_x
   │   ├── xxx.png
   │   ├── xxy.png
   │   └── ...
   │       └── xxz.png
   └── class_y
       ├── 123.png
       ├── nsdf3.png
       ├── ...
       └── asd932_.png
   ```

If the `ann_file` is specified, the dataset will be generated by the first way, otherwise, try the second way.

参数

- **ann_file** (*str*) –Annotation file path. Defaults to ''.
- **metainfo** (*dict, optional*) –Meta information for dataset, such as class information. Defaults to None.
- **data_root** (*str*) –The root directory for `data_prefix` and `ann_file`. Defaults to ''.
- **data_prefix** (*str | dict*) –Prefix for the data. Defaults to ''.
- **extensions** (*Sequence[str]*) –A sequence of allowed extensions. Defaults to ('.jpg', '.jpeg', '.png', '.ppm', '.bmp', '.pgm', '.tif').
- **lazy_init** (*bool*) –Whether to load annotation during instantiation. In some cases, such as visualization, only the meta information of the dataset is needed, which is not necessary to load annotation file. `Basedataset` can skip load annotations to save time by set `lazy_init=False`. Defaults to False.
- ****kwargs** –Other keyword arguments in *BaseDataset*.

82.2 ImageNet

class mmcls.datasets.**ImageNet**(*ann_file=''*, *metainfo=None*, *data_root=''*, *data_prefix=''*, ***kwargs*)

ImageNet Dataset.

The dataset supports two kinds of annotation format. More details can be found in *CustomDataset*.

参数

- **ann_file** (*str*) –Annotation file path. Defaults to ''.
- **metainfo** (*dict, optional*) –Meta information for dataset, such as class information. Defaults to None.
- **data_root** (*str*) –The root directory for `data_prefix` and `ann_file`. Defaults to ''.
- **data_prefix** (*str | dict*) –Prefix for training data. Defaults to ''.
- ****kwargs** –Other keyword arguments in *CustomDataset* and *BaseDataset*.

class mmcls.datasets.**ImageNet21k**(*ann_file=''*, *metainfo=None*, *data_root=''*, *data_prefix=''*, *multi_label=False*, ***kwargs*)

ImageNet21k Dataset.

Since the dataset ImageNet21k is extremely big, cantains 21k+ classes and 1.4B files. We won't provide the default categories list. Please specify it from the `classes` argument.

参数

- **ann_file** (*str*) –Annotation file path. Defaults to ''.
- **metainfo** (*dict, optional*) –Meta information for dataset, such as class information. Defaults to None.
- **data_root** (*str*) –The root directory for `data_prefix` and `ann_file`. Defaults to ''.
- **data_prefix** (*str | dict*) –Prefix for training data. Defaults to ''.
- **multi_label** (*bool*) –Not implement by now. Use multi label or not. Defaults to False.
- ****kwargs** –Other keyword arguments in *CustomDataset* and *BaseDataset*.

82.3 CIFAR

class mmcls.datasets.**CIFAR10**(*data_prefix*, *test_mode*, *metainfo=None*, *data_root=''*, *download=True*, ***kwargs*)

CIFAR10 Dataset.

This implementation is modified from https://github.com/pytorch/vision/blob/master/torchvision/datasets/cifar.py

参数

- **data_prefix** (*str*) –Prefix for data.
- **test_mode** (*bool*) –`test_mode=True` means in test phase. It determines to use the training set or test set.
- **metainfo** (*dict, optional*) –Meta information for dataset, such as categories information. Defaults to None.
- **data_root** (*str*) –The root directory for `data_prefix`. Defaults to ''.
- **download** (*bool*) –Whether to download the dataset if not exists. Defaults to True.
- ****kwargs** –Other keyword arguments in *BaseDataset*.

class mmcls.datasets.**CIFAR100**(*data_prefix*, *test_mode*, *metainfo=None*, *data_root=''*, *download=True*, ***kwargs*)

CIFAR100 Dataset.

参数

- **data_prefix** (*str*) –Prefix for data.
- **test_mode** (*bool*) –`test_mode=True` means in test phase. It determines to use the training set or test set.

- **metainfo** (*dict, optional*) –Meta information for dataset, such as categories information. Defaults to None.
- **data_root** (*str*) –The root directory for `data_prefix`. Defaults to "".
- **download** (*bool*) –Whether to download the dataset if not exists. Defaults to True.
- ****kwargs** –Other keyword arguments in *BaseDataset*.

82.4 MNIST

class mmcls.datasets.**MNIST**(*data_prefix*, *test_mode*, *metainfo=None*, *data_root=''*, *download=True*, ***kwargs*)

MNIST Dataset.

This implementation is modified from https://github.com/pytorch/vision/blob/master/torchvision/datasets/mnist.py

参数

- **data_prefix** (*str*) –Prefix for data.
- **test_mode** (*bool*) –`test_mode=True` means in test phase. It determines to use the training set or test set.
- **metainfo** (*dict, optional*) –Meta information for dataset, such as categories information. Defaults to None.
- **data_root** (*str*) –The root directory for `data_prefix`. Defaults to "".
- **download** (*bool*) –Whether to download the dataset if not exists. Defaults to True.
- ****kwargs** –Other keyword arguments in *BaseDataset*.

class mmcls.datasets.**FashionMNIST**(*data_prefix*, *test_mode*, *metainfo=None*, *data_root=''*, *download=True*, ***kwargs*)

Fashion-MNIST Dataset.

参数

- **data_prefix** (*str*) –Prefix for data.
- **test_mode** (*bool*) –`test_mode=True` means in test phase. It determines to use the training set or test set.
- **metainfo** (*dict, optional*) –Meta information for dataset, such as categories information. Defaults to None.
- **data_root** (*str*) –The root directory for `data_prefix`. Defaults to "".
- **download** (*bool*) –Whether to download the dataset if not exists. Defaults to True.

- ****kwargs** –Other keyword arguments in *BaseDataset*.

82.5 VOC

class mmcls.datasets.**VOC**(*data_root*, *image_set_path*, *data_prefix={'ann_path': 'Annotations', 'img_path': 'JPEGImages'}*, *test_mode=False*, *metainfo=None*, ***kwargs*)

Pascal VOC Dataset.

After decompression, the dataset directory structure is as follows:

VOC dataset directory:

```
VOC2007 (data_root)/
├── JPEGImages (data_prefix['img_path'])
|   ├── xxx.jpg
|   ├── xxy.jpg
|   └── ...
├── Annotations (data_prefix['ann_path'])
|   ├── xxx.xml
|   ├── xxy.xml
|   └── ...
└── ImageSets (directory contains various imageset file)
```

Extra difficult label is in VOC annotations, we will use *gt_label_difficult* to record the difficult labels in each sample and corresponding evaluation should take care of this field to calculate metrics. Usually, difficult labels are reckoned as negative in defaults.

参数

- **data_root** (*str*) –The root directory for VOC dataset.
- **image_set_path** (*str*) –The path of image set, The file which lists image ids of the sub dataset, and this path is relative to `data_root`.
- **data_prefix** (*dict*) –Prefix for data and annotation, keyword 'img_path' and 'ann_path' can be set. Defaults to be `dict(img_path='JPEGImages', ann_path='Annotations')`.
- **test_mode** (*bool*) –`test_mode=True` means in test phase. It determines to use the training set or test set.
- **metainfo** (*dict, optional*) –Meta information for dataset, such as categories information. Defaults to None.
- ****kwargs** –Other keyword arguments in *BaseDataset*.

82.6 CUB

class mmcls.datasets.**CUB**(*data_root*, *test_mode*, *ann_file='images.txt'*, *data_prefix='images'*, *image_class_labels_file='image_class_labels.txt'*, *train_test_split_file='train_test_split.txt'*, ***kwargs*)

The CUB-200-2011 Dataset.

Support the CUB-200-2011 Dataset. Comparing with the CUB-200 Dataset, there are much more pictures in *CUB-200-2011*. After downloading and decompression, the dataset directory structure is as follows.

CUB dataset directory:

```
CUB-200-2011 (data_root)/
├── images (data_prefix)
│   ├── class_x
│   │   ├── xx1.jpg
│   │   ├── xx2.jpg
│   │   └── ...
│   ├── class_y
│   │   ├── yy1.jpg
│   │   ├── yy2.jpg
│   │   └── ...
│   └── ...
├── images.txt (ann_file)
├── image_class_labels.txt (image_class_labels_file)
├── train_test_split.txt (train_test_split_file)
└── ....
```

参数

- **data_root** (*str*) –The root directory for CUB-200-2011 dataset.
- **test_mode** (*bool*) –`test_mode=True` means in test phase. It determines to use the training set or test set.
- **ann_file** (*str, optional*) –Annotation file path, path relative to `data_root`. Defaults to 'images.txt'.
- **data_prefix** (*str*) –Prefix for iamges, path relative to `data_root`. Defaults to 'images'.
- **image_class_labels_file** (*str, optional*) –The label file, path relative to `data_root`. Defaults to 'image_class_labels.txt'.
- **train_test_split_file** (*str, optional*) –The split file to split train and test dataset, path relative to `data_root`. Defaults to 'train_test_split_file.txt'.

使用示例

```
>>> from mmcls.datasets import CUB
>>> cub_train_cfg = dict(data_root='data/CUB_200_2011', test_mode=True)
>>> cub_train = CUB(**cub_train_cfg)
>>> cub_train
Dataset CUB
Number of samples:  5994
Number of categories:       200
Root of dataset:    data/CUB_200_2011
>>> cub_test_cfg = dict(data_root='data/CUB_200_2011', test_mode=True)
>>> cub_test = CUB(**cub_test_cfg)
>>> cub_test
Dataset CUB
Number of samples:  5794
Number of categories:       200
Root of dataset:    data/CUB_200_2011
```

82.7 Retrieval

class mmcls.datasets.**InShop**(*data_root*, *split='train'*, *data_prefix='Img'*, *ann_file='Eval/list_eval_partition.txt'*, ***kwargs*)

InShop Dataset for Image Retrieval.

Please download the images from the homepage 'https://mmlab.ie.cuhk.edu.hk/projects/DeepFashion/InShopRetrieval.html' (In-shop Clothes Retrieval Benchmark -> Img -> img.zip, Eval/list_eval_partition.txt), and organize them as follows way:

In-shop dataset directory:

```
In-shop Clothes Retrieval Benchmark (data_root)/
   ├── Eval /
   │    └── list_eval_partition.txt (ann_file)
   ├── Img (img_prefix)
   │    └── img/
   ├── README.txt
   └── .....
```

参数

- **data_root** (*str*) –The root directory for dataset.
- **split** (*str*) –Choose from 'train', 'query' and 'gallery'. Defaults to 'train'.

- **data_prefix** (*str | dict*) –Prefix for training data. Defaults to 'Img'.
- **ann_file** (*str*) –Annotation file path, path relative to `data_root`. Defaults to 'Eval/list_eval_partition.txt'.
- ****kwargs** –Other keyword arguments in *BaseDataset*.

使用示例

```
>>> from mmcls.datasets import InShop
>>>
>>> # build train InShop dataset
>>> inshop_train_cfg = dict(data_root='data/inshop', split='train')
>>> inshop_train = InShop(**inshop_train_cfg)
>>> inshop_train
Dataset InShop
    Number of samples:  25882
    The `CLASSES` meta info is not set.
    Root of dataset:    data/inshop
>>>
>>> # build query InShop dataset
>>> inshop_query_cfg =  dict(data_root='data/inshop', split='query')
>>> inshop_query = InShop(**inshop_query_cfg)
>>> inshop_query
Dataset InShop
    Number of samples:  14218
    The `CLASSES` meta info is not set.
    Root of dataset:    data/inshop
>>>
>>> # build gallery InShop dataset
>>> inshop_gallery_cfg = dict(data_root='data/inshop', split='gallery')
>>> inshop_gallery = InShop(**inshop_gallery_cfg)
>>> inshop_gallery
Dataset InShop
    Number of samples:  12612
    The `CLASSES` meta info is not set.
    Root of dataset:    data/inshop
```

82.8 Base classes

class mmcls.datasets.**BaseDataset**(*ann_file*, *metainfo=None*, *data_root=''*, *data_prefix=''*, *filter_cfg=None*, *indices=None*, *serialize_data=True*, *pipeline=()*, *test_mode=False*, *lazy_init=False*, *max_refetch=1000*, *classes=None*)

Base dataset for image classification task.

This dataset support annotation file in *OpenMMLab 2.0 style annotation format*.

Comparing with the `mmengine.BaseDataset`, this class implemented several useful methods.

参数

- **ann_file** (*str*) –Annotation file path.
- **metainfo** (*dict, optional*) –Meta information for dataset, such as class information. Defaults to None.
- **data_root** (*str*) –The root directory for `data_prefix` and `ann_file`. Defaults to `''`.
- **data_prefix** (*str | dict*) –Prefix for training data. Defaults to `''`.
- **filter_cfg** (*dict, optional*) –Config for filter data. Defaults to None.
- **indices** (*int or Sequence[int], optional*) –Support using first few data in annotation file to facilitate training/testing on a smaller dataset. Defaults to None, which means using all `data_infos`.
- **serialize_data** (*bool*) –Whether to hold memory using serialized objects, when enabled, data loader workers can use shared RAM from master process instead of making a copy. Defaults to True.
- **pipeline** (*Sequence*) –Processing pipeline. Defaults to an empty tuple.
- **test_mode** (*bool*) –`test_mode=True` means in test phase. Defaults to False.
- **lazy_init** (*bool*) –Whether to load annotation during instantiation. In some cases, such as visualization, only the meta information of the dataset is needed, which is not necessary to load annotation file. `Basedataset` can skip load annotations to save time by set `lazy_init=False`. Defaults to False.
- **max_refetch** (*int*) –If `Basedataset.prepare_data` get a None img. The maximum extra number of cycles to get a valid image. Defaults to 1000.
- **classes** (*str | Sequence[str], optional*) –Specify names of classes.
  - If is string, it should be a file path, and the every line of the file is a name of a class.
  - If is a sequence of string, every item is a name of class.

- If is None, use categories information in `metainfo` argument, annotation file or the class attribute `METAINFO`.

Defaults to None.

class mmcls.datasets.**MultiLabelDataset**(*ann_file*, *metainfo=None*, *data_root=''*, *data_prefix=''*, *filter_cfg=None*, *indices=None*, *serialize_data=True*, *pipeline=()*, *test_mode=False*, *lazy_init=False*, *max_refetch=1000*, *classes=None*)

Multi-label Dataset.

This dataset support annotation file in *OpenMMLab 2.0 style annotation format*.

The annotation format is shown as follows.

```
{
    "metainfo":
    {
      "classes":['A', 'B', 'C'....]
    },
    "data_list":
    [
      {
        "img_path": "test_img1.jpg",
        'img_label': [0, 1],
      },
      {
        "img_path": "test_img2.jpg",
        'img_label': [2],
      },
    ]
    ....
}
```

参数

- **ann_file** (*str*) –Annotation file path.
- **metainfo** (*dict, optional*) –Meta information for dataset, such as class information. Defaults to None.
- **data_root** (*str*) –The root directory for `data_prefix` and `ann_file`. Defaults to '' .
- **data_prefix** (*str | dict*) –Prefix for training data. Defaults to '' .
- **filter_cfg** (*dict, optional*) –Config for filter data. Defaults to None.

- **indices** (*int or Sequence[int], optional*) –Support using first few data in annotation file to facilitate training/testing on a smaller dataset. Defaults to None which means using all `data_infos`.
- **serialize_data** (*bool, optional*) –Whether to hold memory using serialized objects, when enabled, data loader workers can use shared RAM from master process instead of making a copy. Defaults to True.
- **pipeline** (*list, optional*) –Processing pipeline. Defaults to [].
- **test_mode** (*bool, optional*) –`test_mode=True` means in test phase. Defaults to False.
- **lazy_init** (*bool, optional*) –Whether to load annotation during instantiation. In some cases, such as visualization, only the meta information of the dataset is needed, which is not necessary to load annotation file. `Basedataset` can skip load annotations to save time by set `lazy_init=False`. Defaults to False.
- **max_refetch** (*int, optional*) –If `Basedataset.prepare_data` get a None img. The maximum extra number of cycles to get a valid image. Defaults to 1000.
- **classes** (*str | Sequence[str], optional*) –Specify names of classes.
  - If is string, it should be a file path, and the every line of the file is a name of a class.
  - If is a sequence of string, every item is a name of class.
  - If is None, use categories information in `metainfo` argument, annotation file or the class attribute `METAINFO`.

  Defaults to None.

82.9 Dataset Wrappers

class mmcls.datasets.**KFoldDataset**(*dataset*, *fold=0*, *num_splits=5*, *test_mode=False*, *seed=None*)

A wrapper of dataset for K-Fold cross-validation.

K-Fold cross-validation divides all the samples in groups of samples, called folds, of almost equal sizes. And we use k-1 of folds to do training and use the fold left to do validation.

参数

- **dataset** (`mmengine.dataset.BaseDataset` | dict) –The dataset to be divided
- **fold** (*int*) –The fold used to do validation. Defaults to 0.
- **num_splits** (*int*) –The number of all folds. Defaults to 5.
- **test_mode** (*bool*) –Use the training dataset or validation dataset. Defaults to False.

- **seed** (*int, optional*) –The seed to shuffle the dataset before splitting. If None, not shuffle the dataset. Defaults to None.

The dataset wrappers in the MMEngine can be directly used in MMClassification.

ConcatDataset	A wrapper of concatenated dataset.
RepeatDataset	A wrapper of repeated dataset.
ClassBalancedDataset	A wrapper of class balanced dataset.

CHAPTER 83

数据处理

在 MMClassification 中，数据处理和数据集是解耦的。数据集只定义了如何从文件系统中获取样本的基本信息。这些基本信息包括分类标签和原始图像数据/图像的路径。完整的数据处理流程包括了数据变换（data transform）、数据预处理器（data preprocessor）及批量数据增强（batch augmentation）。

- 数据变换：数据变换包括了数据的加载、部分预处理/增强、数据格式化等操作
- 数据预处理器：主要负责批量数据的收集、归一化、堆叠、通道翻转等操作。
  - 批量数据增强：批量数据增强是数据预处理器的功能之一，负责处理涉及多个样本的数据增强操作，例如 Mixup 和 CutMix。

83.1 数据变换

为了准备输入数据，我们需要对数据集中保存的基本信息做一些变换。这些变换包括数据加载、部分预处理和增强、格式化。一系列的数据变换组成了数据流水线（data pipeline）。因此，在数据集的配置参数中通常存在一个 `pipeline` 参数，例如：

```
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='RandomResizedCrop', scale=224),
    dict(type='RandomFlip', prob=0.5, direction='horizontal'),
    dict(type='PackClsInputs'),
]
```

(下页继续)

(续上页)

```
train_dataloader = dict(
    ....
    dataset=dict(
        pipeline=train_pipeline,
        ....),
    ....
)
```

pipeline 列表中的每一项都是以下数据变换类之一。如果您想添加自定义数据变换类，可以参考自定义数据流水线教程。

- 组合式增强
- 格式化
- *MMCV* 中的数据变换

83.1.1 组合式增强

Albumentations	使用 Albumentations 库进行数据变换的封装类
ColorJitter	随机改变图像的亮度、对比度和饱和度
EfficientNetCenterCrop	EfficientNet 风格的中心裁剪
EfficientNetRandomCrop	EfficientNet 风格的随机缩放裁剪
Lighting	使用 AlexNet 风格的 PCA 抖动随机调整图像照明
RandomCrop	在随机位置裁剪给定图像
RandomErasing	在图像中随机选择一个矩形区域并擦除像素
RandomResizedCrop	将给定图像按照随机尺寸和纵横比进行裁剪
ResizeEdge	按照指定边长调整图像尺寸

Albumentations

class mmcls.datasets.transforms.**Albumentations**(*transforms*, *keymap=None*)

使用 Albumentations 库进行数据变换的封装类

Required Keys:

- img

Modified Keys:

- img
- img_shape

Adds custom transformations from albumentations library. More details can be found in Albumentations. An example of `transforms` is as followed:

```
[
    dict(
        type='ShiftScaleRotate',
        shift_limit=0.0625,
        scale_limit=0.0,
        rotate_limit=0,
        interpolation=1,
        p=0.5),
    dict(
        type='RandomBrightnessContrast',
        brightness_limit=[0.1, 0.3],
        contrast_limit=[0.1, 0.3],
        p=0.2),
    dict(type='ChannelShuffle', p=0.1),
    dict(
        type='OneOf',
        transforms=[
            dict(type='Blur', blur_limit=3, p=1.0),
            dict(type='MedianBlur', blur_limit=3, p=1.0)
        ],
        p=0.1),
]
```

参数

- **transforms** (*List[Dict]*) –List of albumentations transform configs.
- **keymap** (*Optional[Dict]*) –Mapping of mmcls to albumentations fields, in format {'input key':'albumentation-style key'}. Defaults to None.

示例

```
>>> import mmcv
>>> from mmcls.datasets import Albumentations
>>> transforms = [
...     dict(
...         type='ShiftScaleRotate',
...         shift_limit=0.0625,
...         scale_limit=0.0,
...         rotate_limit=0,
...         interpolation=1,
```

(下页继续)

(续上页)

```
...         p=0.5),
...     dict(
...         type='RandomBrightnessContrast',
...         brightness_limit=[0.1, 0.3],
...         contrast_limit=[0.1, 0.3],
...         p=0.2),
...     dict(type='ChannelShuffle', p=0.1),
...     dict(
...         type='OneOf',
...         transforms=[
...             dict(type='Blur', blur_limit=3, p=1.0),
...             dict(type='MedianBlur', blur_limit=3, p=1.0)
...         ],
...         p=0.1),
... ]
>>> albu = Albumentations(transforms)
>>> data = {'img': mmcv.imread('./demo/demo.JPEG')}
>>> data = albu(data)
>>> print(data['img'].shape)
(375, 500, 3)
```

transform(*results*)

Transform function to perform albumentations transforms.

参数 **results** (*dict*) –Result dict from loading pipeline.

返回

Transformed results, 'img' and 'img_shape' keys are updated in result dict.

返回类型 dict

ColorJitter

class mmcls.datasets.transforms.**ColorJitter**(*brightness=0.0*, *contrast=0.0*, *saturation=0.0*, *hue=0.0*)

随机改变图像的亮度、对比度和饱和度

Modified from https://github.com/pytorch/vision/blob/main/torchvision/transforms/transforms.py Licensed under the BSD 3-Clause License.

Required Keys:

- img

Modified Keys:

- img

参数

- **brightness** (*float | Sequence[float] (min, max)*) –How much to jitter brightness. brightness_factor is chosen uniformly from `[max(0, 1 - brightness), 1 + brightness]` or the given `[min, max]`. Should be non negative numbers. Defaults to 0.
- **contrast** (*float | Sequence[float] (min, max)*) –How much to jitter contrast. contrast_factor is chosen uniformly from `[max(0, 1 - contrast), 1 + contrast]` or the given `[min, max]`. Should be non negative numbers. Defaults to 0.
- **saturation** (*float | Sequence[float] (min, max)*) –How much to jitter saturation. saturation_factor is chosen uniformly from `[max(0, 1 - saturation), 1 + saturation]` or the given `[min, max]`. Should be non negative numbers. Defaults to 0.
- **hue** (*float | Sequence[float] (min, max)*) –How much to jitter hue. hue_factor is chosen uniformly from `[-hue, hue]` (0 <= hue <= 0.5) or the given `[min, max]` (-0.5 <= min <= max <= 0.5). Defaults to 0.

transform(*results*)

Transform function to resize images.

参数 **results** (*dict*) –Result dict from loading pipeline.

返回 ColorJitter results, ‘img’ key is updated in result dict.

返回类型 dict

EfficientNetCenterCrop

class mmcls.datasets.transforms.**EfficientNetCenterCrop**(*crop_size*, *crop_padding=32*, *interpolation='bicubic'*, *backend='cv2'*)

EfficientNet 风格的中心裁剪

Required Keys:

- img

Modified Keys:

- img
- img_shape