MMClassification Documentation

发布 *0.25.0*

MMClassification Authors

2022 年 12 月 30 日

开始你的第一步

You can switch between Chinese and English documentation in the lower-left corner of the layout.

您可以在页面左下角切换中英文文档。

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 3.6+、CUDA 9.2+ 和 PyTorch 1.5+。

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

第 1 步使用 MIM 安装 MMCV

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

第 2 步安装 MMClassification

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 https://github.com/open-mmlab/mmclassification.git
cd mmclassification
pip install -v -e .
# "-v" 表示输出更多安装相关的信息
# "-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
```

另外，如果你希望向 MMClassification 贡献代码，或者使用试验中的功能，请签出到 dev 分支。

```
git checkout dev
```

2.1.2 作为 Python 包安装

直接使用 pip 安装即可。

```
pip install mmcls
```

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。另外如果你拥有图形界面（而不是使用远程终端），那么可以启用 --show 选项，将示例图像和对应的预测结果在窗口中进行显示。

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.apis import init_model, inference_model

config_file = 'resnet50_8xb32_in1k.py'
```

(下页继续)

(续上页)

```
checkpoint_file = 'resnet50_8xb32_in1k_20210831-ea4938fc.pth'
model = init_model(config_file, checkpoint_file, 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。

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 不使用 MIM 安装 MMCV

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

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

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

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

2.3.3 在 CPU 环境中安装

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

在 CPU 模式下，MMCV 的部分功能将不可用，通常是一些 GPU 编译的算子。不过不用担心，MMClassification 中几乎所有的模型都不会依赖这些算子。

2.3.4 在 Google Colab 中安装

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

第 1 步使用 MIM 安装 MMCV

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

第 2 步从源码安装 MMClassification

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

第 3 步验证

```
import mmcls
print(mmcls.__version__)
# 预期输出：  0.23.0 或更新的版本号
```

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

2.3.5 通过 Docker 使用 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

基础教程

本文档提供 MMClassification 相关用法的基本教程。

3.1 准备数据集

MMClassification 建议用户将数据集根目录链接到 `$MMCLASSIFICATION/data` 下。如果用户的文件夹结构与默认结构不同，则需要在配置文件中进行对应路径的修改。

```
mmclassification
├── mmcls
├── tools
├── configs
├── docs
├── data
|   ├── imagenet
|   |   ├── meta
|   |   ├── train
|   |   ├── val
|   ├── cifar
|   |   ├── cifar-10-batches-py
|   ├── mnist
|   |   ├── train-images-idx3-ubyte
|   |   ├── train-labels-idx1-ubyte
|   |   ├── t10k-images-idx3-ubyte
```

(下页继续)

(续上页)

```
|   |   ├── t10k-labels-idx1-ubyte
```

对于 ImageNet，其存在多个版本，但最为常用的一个是 ILSVRC 2012，可以通过以下步骤获取该数据集。

1. 注册账号并登录 下载页面
2. 获取 ILSVRC2012 下载链接并下载以下文件
   - ILSVRC2012_img_train.tar (~138GB)
   - ILSVRC2012_img_val.tar (~6.3GB)
3. 解压下载的文件
4. 使用 该脚本 获取元数据

对于 MNIST，CIFAR10 和 CIFAR100，程序将会在需要的时候自动下载数据集。

对于用户自定义数据集的准备，请参阅*教程 3：如何自定义数据集*

3.2 使用预训练模型进行推理

MMClassification 提供了一些脚本用于进行单张图像的推理、数据集的推理和数据集的测试（如 ImageNet 等）

3.2.1 单张图像的推理

```
python demo/image_demo.py ${IMAGE_FILE} ${CONFIG_FILE} ${CHECKPOINT_FILE}

# Example
python demo/image_demo.py demo/demo.JPEG configs/resnet/resnet50_8xb32_in1k.py \
  https://download.openmmlab.com/mmclassification/v0/resnet/resnet50_8xb32_in1k_
↪20210831-ea4938fc.pth
```

3.2.2 数据集的推理与测试

- 支持单 GPU
- 支持 CPU
- 支持单节点多 GPU
- 支持多节点

用户可使用以下命令进行数据集的推理：

```
# 单 GPU
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--metrics ${METRICS}] [--out $
↪{RESULT_FILE}]

# CPU: 禁用 GPU 并运行单 GPU 测试脚本
export CUDA_VISIBLE_DEVICES=-1
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--metrics ${METRICS}] [--out $
↪{RESULT_FILE}]

# 多 GPU
./tools/dist_test.sh ${CONFIG_FILE} ${CHECKPOINT_FILE} ${GPU_NUM} [--metrics $
↪{METRICS}] [--out ${RESULT_FILE}]

# 基于 slurm 分布式环境的多节点
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--metrics ${METRICS}] [--out $
↪{RESULT_FILE}] --launcher slurm
```

可选参数：

- RESULT_FILE：输出结果的文件名。如果未指定，结果将不会保存到文件中。支持 json, yaml, pickle 格式。
- METRICS：数据集测试指标，如准确率 (accuracy), 精确率 (precision), 召回率 (recall) 等

例子：

在 ImageNet 验证集上，使用 ResNet-50 进行推理并获得预测标签及其对应的预测得分。

```
python tools/test.py configs/resnet/resnet50_8xb16_cifar10.py \
  https://download.openmmlab.com/mmclassification/v0/resnet/resnet50_b16x8_cifar10_
↪20210528-f54bfad9.pth \
  --out result.pkl
```

3.3 模型训练

MMClassification 使用 MMDistributedDataParallel 进行分布式训练，使用 MMDataParallel 进行非分布式训练。

所有的输出（日志文件和模型权重文件）会被将保存到工作目录下。工作目录通过配置文件中的参数 work_dir 指定。

默认情况下，MMClassification 在每个周期后会在验证集上评估模型，可以通过在训练配置中修改 interval 参数来更改评估间隔

```
evaluation = dict(interval=12)  # 每进行 12 轮训练后评估一次模型
```

3.3.1 使用单个 GPU 进行训练

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

如果用户想在命令中指定工作目录，则需要增加参数 `--work-dir ${YOUR_WORK_DIR}`

3.3.2 使用 CPU 训练

使用 CPU 训练的流程和使用单 GPU 训练的流程一致，我们仅需要在训练流程开始前禁用 GPU。

```
export CUDA_VISIBLE_DEVICES=-1
```

之后运行单 GPU 训练脚本即可。

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

3.3.3 使用单台机器多个 GPU 进行训练

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

可选参数为：

- `--no-validate` (**不建议**): 默认情况下，程序将会在训练期间的每 k（默认为 1) 个周期进行一次验证。要禁用这一功能，使用 `--no-validate`
- `--work-dir ${WORK_DIR}`：覆盖配置文件中指定的工作目录。
- `--resume-from ${CHECKPOINT_FILE}`：从以前的模型权重文件恢复训练。

`resume-from` 和 `load-from` 的不同点：`resume-from` 加载模型参数和优化器状态，并且保留检查点所在的周期数，常被用于恢复意外被中断的训练。`load-from` 只加载模型参数，但周期数从 0 开始计数，常被用于微调模型。

3.3.4 使用多台机器进行训练

如果您想使用由 ethernet 连接起来的多台机器，您可以使用以下命令:

在第一台机器上:

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

在第二台机器上:

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

但是，如果您不使用高速网路连接这几台机器的话，训练将会非常慢。

如果用户在 slurm 集群上运行 MMClassification，可使用 `slurm_train.sh` 脚本。(该脚本也支持单台机器上进行训练)

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

用户可以在 slurm_train.sh 中检查所有的参数和环境变量

如果用户的多台机器通过 Ethernet 连接，则可以参考 pytorch launch utility。如果用户没有高速网络，如 InfiniBand，速度将会非常慢。

3.3.5 使用单台机器启动多个任务

如果用使用单台机器启动多个任务，如在有 8 块 GPU 的单台机器上启动 2 个需要 4 块 GPU 的训练任务，则需要为每个任务指定不同端口，以避免通信冲突。

如果用户使用 `dist_train.sh` 脚本启动训练任务，则可以通过以下命令指定端口

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

如果用户在 slurm 集群下启动多个训练任务，则需要修改配置文件中的 `dist_params` 变量，以设置不同的通信端口。

在 `config1.py` 中，

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

在 `config2.py` 中，

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

之后便可启动两个任务，分别对应 config1.py 和 config2.py。

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

3.4 实用工具

我们在 tools/ 目录下提供的一些对训练和测试十分有用的工具

3.4.1 计算 FLOPs 和参数量（试验性的）

我们根据 flops-counter.pytorch 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}]
```

用户将获得如下结果：

```
==============================
Input shape: (3, 224, 224)
Flops: 4.12 GFLOPs
Params: 25.56 M
==============================
```

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 GN 和自定义运算。详细信息请参考 mmcv.cnn.get_model_complexity_info()

3.4.2 模型发布

在发布模型之前，你也许会需要

1. 转换模型权重至 CPU 张量
2. 删除优化器状态
3. 计算模型权重文件的哈希值，并添加至文件名之后

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

例如：

```
python tools/convert_models/publish_model.py work_dirs/resnet50/latest.pth imagenet_
↪resnet50.pth
```

最终输出的文件名将会是 `imagenet_resnet50_{date}-{hash id}.pth`

3.5 详细教程

目前，MMClassification 提供以下几种更详细的教程：

- 如何编写配置文件
- 如何微调模型
- 如何增加新数据集
- 如何设计数据处理流程
- 如何增加新模块
- 如何自定义优化策略
- 如何自定义运行参数。

CHAPTER 4

教程 1：如何编写配置文件

MMClassification 主要使用 python 文件作为配置文件。其配置文件系统的设计将模块化与继承整合进来，方便用户进行各种实验。所有配置文件都放置在 `configs` 文件夹下，主要包含 `_base_` 原始配置文件夹以及 `resnet`, `swin_transformer`, `vision_transformer` 等诸多算法文件夹。

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。

4.1 配置文件以及权重命名规则

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 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；

4.1.1 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

4.1.2 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

4.1.3 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

4.1.4 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`

4.1.5 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。

备注： 部分配置文件目前还没有遵循此命名规范，相关文件命名近期会更新。

4.1.6 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
```

4.2 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。由来自 _base_ 的组件组成的配置称为 *primitive*。

为了帮助用户对 MMClassification 检测系统中的完整配置和模块有一个基本的了解，我们使用 ResNet50 原始配置文件 作为案例进行说明并注释每一行含义。更详细的用法和各个模块对应的替代方案，请参考 API 文档。

```
_base_ = [
    '../_base_/models/resnet50.py',           # 模型
    '../_base_/datasets/imagenet_bs32.py',    # 数据
    '../_base_/schedules/imagenet_bs256.py',  # 训练策略
    '../_base_/default_runtime.py'            # 默认运行设置
]
```

下面对这四个部分分别进行说明，仍然以上述 ResNet50 原始配置文件作为案例。

4.2.1 模型

模型参数 model 在配置文件中为一个 python 字典，主要包括网络结构、损失函数等信息：

- type： 分类器名称, 目前 MMClassification 只支持 ImageClassifier，参考 API 文档。
- backbone： 主干网类型，可用选项参考 API 文档。
- neck： 颈网络类型，目前 MMClassification 只支持 GlobalAveragePooling，参考 API 文档。
- head： 头网络类型，包括单标签分类与多标签分类头网络，可用选项参考 API 文档。
  - loss： 损失函数类型，支持 CrossEntropyLoss, LabelSmoothLoss 等，可用选项参考 API 文档。

- train_cfg： 训练配置, 支持 mixup, cutmix 等训练增强。

备注： 配置文件中的‘type’不是构造时的参数，而是类名。

```
model = dict(
    type='ImageClassifier',      # 分类器类型
    backbone=dict(
        type='ResNet',           # 主干网络类型
        depth=50,                # 主干网网络深度， ResNet 一般有18, 34, 50, 101, 152␣
↪可以选择
        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。越远离输入图像，索引越大
        frozen_stages=-1,        #␣
↪网络微调时，冻结网络的stage（训练时不执行反相传播算法），若num_
↪stages=4，backbone包含stem 与 4 个 stages。frozen_stages为-1时，不冻结网络；␣
↪为0时，冻结 stem； 为1时，冻结 stem 和 stage1； 为4时，冻结整个backbone
        style='pytorch'),        # 主干网络的风格，'pytorch' 意思是步长为2的层为 3x3␣
↪卷积， 'caffe' 意思是步长为2的层为 1x1 卷积。
    neck=dict(type='GlobalAveragePooling'),    # 颈网络类型
    head=dict(
        type='LinearClsHead',      # 线性分类头，
        num_classes=1000,          # 输出类别数，这与数据集的类别数一致
        in_channels=2048,          # 输入通道数，这与 neck 的输出通道一致
        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
    ))
```

4.2.2 数据

数据参数 data 在配置文件中为一个 python 字典，主要包含构造数据集加载器 (dataloader) 配置信息：

- samples_per_gpu：构建 dataloader 时，每个 GPU 的 Batch Size
- workers_per_gpu：构建 dataloader 时，每个 GPU 的线程数
- train | val | test：构造数据集
  - type：数据集类型，MMClassification 支持 ImageNet、Cifar 等，参考API 文档
  - data_prefix：数据集根目录
  - 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

评估参数 evaluation 也是一个字典，为 evaluation hook 的配置信息，主要包括评估间隔、评估指标等。

```
# dataset settings
dataset_type = 'ImageNet'  # 数据集名称，
img_norm_cfg = dict(       #图像归一化配置，用来归一化输入的图像。
    mean=[123.675, 116.28, 103.53],  # 预训练里用于预训练主干网络模型的平均值。
    std=[58.395, 57.12, 57.375],     # 预训练里用于预训练主干网络模型的标准差。
    to_rgb=True)                     # 是否反转通道，使用 cv2, mmcv 读取图片默认为␣
↪BGR 通道顺序，这里 Normalize 均值方差数组的数值是以 RGB 通道顺序，␣
↪因此需要反转通道顺序。
# 训练数据流水线
train_pipeline = [
    dict(type='LoadImageFromFile'),                 # 读取图片
    dict(type='RandomResizedCrop', size=224),       # 随机缩放抠图
    dict(type='RandomFlip', flip_prob=0.5, direction='horizontal'),  # 以概率为0.
↪5随机水平翻转图片
    dict(type='Normalize', **img_norm_cfg),         # 归一化
    dict(type='ImageToTensor', keys=['img']),       # image 转为 torch.Tensor
    dict(type='ToTensor', keys=['gt_label']),       # gt_label 转为 torch.Tensor
    dict(type='Collect', keys=['img', 'gt_label']) #␣
↪决定数据中哪些键应该传递给检测器的流程
]
# 测试数据流水线
test_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='Resize', size=(256, -1)),
    dict(type='CenterCrop', crop_size=224),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='ImageToTensor', keys=['img']),
    dict(type='Collect', keys=['img'])              # test 时不传递 gt_label
]
data = dict(
    samples_per_gpu=32,     # 单个 GPU 的 Batch size
    workers_per_gpu=2,      # 单个 GPU 的 线程数
    train=dict(             # 训练数据信息
        type=dataset_type,                  # 数据集名称
        data_prefix='data/imagenet/train',  # 数据集目录，当不存在 ann_file␣
↪时，类别信息从文件夹自动获取
        pipeline=train_pipeline),           # 数据集需要经过的 数据流水线
    val=dict(               # 验证数据集信息
        type=dataset_type,
        data_prefix='data/imagenet/val',
        ann_file='data/imagenet/meta/val.txt',    # 标注文件路径，存在 ann_file␣
↪时，不通过文件夹自动获取类别信息
```

(下页继续)

(续上页)

```
        pipeline=test_pipeline),
    test=dict(              # 测试数据集信息
        type=dataset_type,
        data_prefix='data/imagenet/val',
        ann_file='data/imagenet/meta/val.txt',
        pipeline=test_pipeline))
evaluation = dict(        # evaluation hook 的配置
    interval=1,           # 验证期间的间隔，单位为 epoch 或者 iter，取决于 runner␣
↪类型。
    metric='accuracy')    # 验证期间使用的指标。
```

4.2.3 训练策略

主要包含优化器设置、optimizer hook 设置、学习率策略和 runner 设置:

- optimizer : 优化器设置信息, 支持 pytorch 所有的优化器，参考相关 mmcv 文档
- optimizer_config : optimizer hook 的配置文件, 如设置梯度限制，参考相关 mmcv 代码
- lr_config : 学习率策略，支持 “CosineAnnealing”、“Step”、“Cyclic” 等等，参考相关 mmcv 文档
- runner : 有关 runner 可以参考 mmcv 对于 runner 介绍文档

```
# 用于构建优化器的配置文件。支持 PyTorch 中的所有优化器，同时它们的参数与 PyTorch␣
↪里的优化器参数一致。
optimizer = dict(type='SGD',          # 优化器类型
                 lr=0.1,              # 优化器的学习率，参数的使用细节请参照对应的␣
↪PyTorch 文档。
                 momentum=0.9,        # 动量(Momentum)
                 weight_decay=0.0001) # 权重衰减系数(weight decay)。
 # optimizer hook 的配置文件
optimizer_config = dict(grad_clip=None)  # 大多数方法不使用梯度限制(grad_clip)。
# 学习率调整配置，用于注册 LrUpdater hook。
lr_config = dict(policy='step',           # 调度流程(scheduler)的策略，也支持␣
↪CosineAnnealing, Cyclic, 等。
                 step=[30, 60, 90])       # 在 epoch 为 30, 60, 90 时，lr 进行衰减
runner = dict(type='EpochBasedRunner',    # 将使用的 runner 的类别，如 IterBasedRunner␣
↪或 EpochBasedRunner。
              max_epochs=100)             # runner 总回合数，对于 IterBasedRunner␣
↪使用 `max_iters`
```

4.2.4 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。

```
# Checkpoint hook 的配置文件。
checkpoint_config = dict(interval=1)    # 保存的间隔是 1，单位会根据 runner␣
↪不同变动，可以为 epoch 或者 iter。
# 日志配置信息。
log_config = dict(
    interval=100,                       # 打印日志的间隔， 单位 iters
    hooks=[
        dict(type='TextLoggerHook'),           # 用于记录训练过程的文本记录器(logger)。
        # dict(type='TensorboardLoggerHook')  # 同样支持 Tensorboard 日志
    ])

dist_params = dict(backend='nccl')    # 用于设置分布式训练的参数，端口也同样可被设置。
log_level = 'INFO'                    # 日志的输出级别。
resume_from = None                    #␣
↪从给定路径里恢复检查点(checkpoints)，训练模式将从检查点保存的轮次开始恢复训练。
workflow = [('train', 1)]             # runner 的工作流程，[('train', 1)]␣
↪表示只有一个工作流且工作流仅执行一次。
work_dir = 'work_dir'                 # 用于保存当前实验的模型检查点和日志的目录文件地址。
```

4.3 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:

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

runner = dict(max_epochs=300)
lr_config = dict(step=[150, 200, 250])

data = dict(
    train=dict(data_prefix='mydata/imagenet/train'),
```

(下页继续)

(续上页)

```
    val=dict(data_prefix='mydata/imagenet/train', ),
    test=dict(data_prefix='mydata/imagenet/train', )
)
```

4.3.1 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 `train_pipeline` / `test_pipeline`，然后将它们传递到 `data` 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。

```
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='RandomResizedCrop', size=384, backend='pillow',),
    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'])
]
test_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='Resize', size=384, backend='pillow'),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='ImageToTensor', keys=['img']),
    dict(type='Collect', keys=['img'])
]
data = dict(
    train=dict(pipeline=train_pipeline),
    val=dict(pipeline=test_pipeline),
    test=dict(pipeline=test_pipeline))
```

4.3.2 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。可以参照 mmcv 来获得一些简单的指导。

以下是一个简单应用案例。如果在上述 ResNet50 案例中使用 cosine schedule，使用继承并直接修改会报 `get unexcepected keyword 'step'` 错，因为基础配置文件 lr_config 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 `lr_config` 相关域内容：

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

lr_config = dict(
    _delete_=True,
    policy='CosineAnnealing',
    min_lr=0,
    warmup='linear',
    by_epoch=True,
    warmup_iters=5,
    warmup_ratio=0.1
)
```

4.3.3 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。可以参照 mmcv 来获得一些简单的指导。

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/_base_/datasets/imagenet_bs64_autoaug.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_ = ['./pipelines/auto_aug.py']

# dataset settings
dataset_type = 'ImageNet'
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='RandomResizedCrop', size=224),
    dict(type='RandomFlip', flip_prob=0.5, direction='horizontal'),
    dict(type='AutoAugment', policies={{_base_.auto_increasing_policies}}),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='ImageToTensor', keys=['img']),
```

(下页继续)

(续上页)

```
    dict(type='ToTensor', keys=['gt_label']),
    dict(type='Collect', keys=['img', 'gt_label'])
]
test_pipeline = [...]
data = dict(
    samples_per_gpu=64,
    workers_per_gpu=2,
    train=dict(..., pipeline=train_pipeline),
    val=dict(..., pipeline=test_pipeline))
evaluation = dict(interval=1, metric='accuracy')
```

4.4 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 `workflow=[('train', 1)]`，用户如果想更改，需要指定 `--cfg-options workflow="[(train,1),(val,1)]"`。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。

4.5 导入用户自定义模块

备注： 本部分仅在当将 MMClassification 当作库构建自己项目时可能用到，初学者可跳过。

在学习完后续教程 如何添加新数据集、如何设计数据处理流程 、如何增加新模块 后，您可能使用 MMClassification 完成自己的项目并在项目中自定义了数据集、模型、数据增强等。为了精简代码，可以将 MMClas-

sification 作为一个第三方库，只需要保留自己的额外的代码，并在配置文件中导入自定义的模块。案例可以参考 OpenMMLab 算法大赛项目。

只需要在你的配置文件中添加以下代码：

```
custom_imports = dict(
    imports=['your_dataset_class',
             'your_transforme_class',
             'your_model_class',
             'your_module_class'],
    allow_failed_imports=False)
```

4.6 常见问题

- 无

CHAPTER 5

教程 2：如何微调模型

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。

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*教程 3：如何自定义数据集* 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。

5.1 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。要微调 ResNet-50 模型，新配置需要继承 `_base_/models/resnet50.py` 来搭建模型的基本结构。为了使用 CIFAR10 数据集，新的配置文件可以直接继承 `_base_/datasets/cifar10.py`。而为了保留运行相关设置，比如训练调整器，新的配置文件需要继承 `_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.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`。

5.2 修改模型

在进行模型微调是，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'` 来告诉初始化函数移除权重文件中键值名称的前缀，比如把 `backbone.conv1` 变成 `conv1`。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，有时我们在进行微调时会希望冻结主干网络前面几层的参数，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',
```

(下页继续)

(续上页)

```
            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。

5.3 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）来实现。

```
img_norm_cfg = dict(
    mean=[125.307, 122.961, 113.8575],
    std=[51.5865, 50.847, 51.255],
    to_rgb=False,
)
train_pipeline = [
    dict(type='RandomCrop', size=32, padding=4),
    dict(type='RandomFlip', flip_prob=0.5, direction='horizontal'),
    dict(type='Resize', size=224),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='ImageToTensor', keys=['img']),
    dict(type='ToTensor', keys=['gt_label']),
    dict(type='Collect', keys=['img', 'gt_label']),
]
test_pipeline = [
    dict(type='Resize', size=224),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='ImageToTensor', keys=['img']),
    dict(type='Collect', keys=['img']),
]
data = dict(
    train=dict(pipeline=train_pipeline),
    val=dict(pipeline=test_pipeline),
    test=dict(pipeline=test_pipeline),
```

(下页继续)

(续上页)

```
)
```

5.4 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和较少的训练时间。

```
# 用于批大小为 128 的优化器学习率
optimizer = dict(type='SGD', lr=0.01, momentum=0.9, weight_decay=0.0001)
optimizer_config = dict(grad_clip=None)
# 学习率衰减策略
lr_config = dict(policy='step', step=[15])
runner = dict(type='EpochBasedRunner', max_epochs=200)
log_config = dict(interval=100)
```

5.5 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_/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
img_norm_cfg = dict(
    mean=[125.307, 122.961, 113.8575],
    std=[51.5865, 50.847, 51.255],
```

(下页继续)

(续上页)

```
    to_rgb=False,
)
train_pipeline = [
    dict(type='RandomCrop', size=32, padding=4),
    dict(type='RandomFlip', flip_prob=0.5, direction='horizontal'),
    dict(type='Resize', size=224),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='ImageToTensor', keys=['img']),
    dict(type='ToTensor', keys=['gt_label']),
    dict(type='Collect', keys=['img', 'gt_label']),
]
test_pipeline = [
    dict(type='Resize', size=224),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='ImageToTensor', keys=['img']),
    dict(type='Collect', keys=['img']),
]
data = dict(
    train=dict(pipeline=train_pipeline),
    val=dict(pipeline=test_pipeline),
    test=dict(pipeline=test_pipeline),
)

# 训练策略设置
# 用于批大小为 128 的优化器学习率
optimizer = dict(type='SGD', lr=0.01, momentum=0.9, weight_decay=0.0001)
optimizer_config = dict(grad_clip=None)
# 学习率衰减策略
lr_config = dict(policy='step', step=[15])
runner = dict(type='EpochBasedRunner', max_epochs=200)
log_config = dict(interval=100)
```

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：

```
data = dict(
    samples_per_gpu=128,
```

(下页继续)

(续上页)

```
    train=dict(pipeline=train_pipeline),
    val=dict(pipeline=test_pipeline),
    test=dict(pipeline=test_pipeline),
)
```

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 `samples_per_gpu=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 `samples_per_gpu=128` 来匹配训练策略。

CHAPTER 6

教程 3：如何自定义数据集

我们支持许多常用的图像分类领域公开数据集，你可以在 此页面中找到它们。

在本节中，我们将介绍如何使用自己的数据集以及如何使用数据集包装。

6.1 使用自己的数据集

6.1.1 将数据集重新组织为已有格式

想要使用自己的数据集，最简单的方法就是将数据集转换为现有的数据集格式。

对于多分类任务，我们推荐使用 `CustomDataset` 格式。

`CustomDataset` 支持两种类型的数据格式：

1. 提供一个标注文件，其中每一行表示一张样本图片。

   样本图片可以以任意的结构进行组织，比如：

```
train/
├── folder_1
│   ├── xxx.png
│   ├── xxy.png
│   └── ...
├── 123.png
```

(下页继续)

(续上页)

```
├── nsdf3.png
└── ...
```

而标注文件则记录了所有样本图片的文件路径以及相应的类别序号。其中第一列表示图像相对于主目录（本例中为 `train` 目录）的路径，第二列表示类别序号：

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

备注： 类别序号的值应当属于 `[0, num_classes - 1]` 范围。

2. 将所有样本文件按如下结构进行组织：

```
train/
├── cat
│   ├── xxx.png
│   ├── xxy.png
│   └── ...
│       └── xxz.png
├── bird
│   ├── bird1.png
│   ├── bird2.png
│   └── ...
└── dog
    ├── 123.png
    ├── nsdf3.png
    ├── ...
    └── asd932_.png
```

这种情况下，你不需要提供标注文件，所有位于 `cat` 目录下的图片文件都会被视为 `cat` 类别的样本。

通常而言，我们会将整个数据集分为三个子数据集：`train`，`val` 和 `test`，分别用于训练、验证和测试。**每一个**子数据集都需要被组织成如上的一种结构。

举个例子，完整的数据集结构如下所示（使用第一种组织结构）：

```
mmclassification
└── data
    └── my_dataset
        ├── meta
```

(下页继续)

(续上页)

```
        |   ├── train.txt
        |   ├── val.txt
        |   └── test.txt
        ├── train
        ├── val
        └── test
```

之后在你的配置文件中，可以修改其中的 data 字段为如下格式：

```
...
dataset_type = 'CustomDataset'
classes = ['cat', 'bird', 'dog']  # 数据集中各类别的名称

data = dict(
    train=dict(
        type=dataset_type,
        data_prefix='data/my_dataset/train',
        ann_file='data/my_dataset/meta/train.txt',
        classes=classes,
        pipeline=train_pipeline
    ),
    val=dict(
        type=dataset_type,
        data_prefix='data/my_dataset/val',
        ann_file='data/my_dataset/meta/val.txt',
        classes=classes,
        pipeline=test_pipeline
    ),
    test=dict(
        type=dataset_type,
        data_prefix='data/my_dataset/test',
        ann_file='data/my_dataset/meta/test.txt',
        classes=classes,
        pipeline=test_pipeline
    )
)
...
```

6.1.2 创建一个新的数据集类

用户可以编写一个继承自 BasesDataset 的新数据集类，并重载 load_annotations(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
```

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。

```
import mmcv
import numpy as np

from .builder import DATASETS
from .base_dataset import BaseDataset


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

    def load_annotations(self):
        assert isinstance(self.ann_file, str)

        data_infos = []
        with open(self.ann_file) as f:
            samples = [x.strip().split(' ') for x in f.readlines()]
            for filename, gt_label in samples:
                info = {'img_prefix': self.data_prefix}
                info['img_info'] = {'filename': filename}
                info['gt_label'] = np.array(gt_label, dtype=np.int64)
                data_infos.append(info)
            return data_infos
```

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

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

(下页继续)

(续上页)

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

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

6.2 使用数据集包装

数据集包装是一种可以改变数据集类行为的类，比如将数据集中的样本进行重复，或是将不同类别的数据进行再平衡。

6.2.1 重复数据集

我们使用 RepeatDataset 作为一个重复数据集的封装。举个例子，假设原始数据集是 Dataset_A，为了重复它，我们需要如下的配置文件：

```
data = dict(
    train=dict(
        type='RepeatDataset',
        times=N,
        dataset=dict(  # 这里是 Dataset_A 的原始配置
            type='Dataset_A',
            ...
            pipeline=train_pipeline
        )
    )
    ...
)
```

6.2.2 类别平衡数据集

我们使用 `ClassBalancedDataset` 作为根据类别频率对数据集进行重复采样的封装类。进行重复采样的数据集需要实现函数 `self.get_cat_ids(idx)` 以支持 `ClassBalancedDataset`。

举个例子，按照 `oversample_thr=1e-3` 对 `Dataset_A` 进行重复采样，需要如下的配置文件：

```
data = dict(
    train = dict(
        type='ClassBalancedDataset',
        oversample_thr=1e-3,
        dataset=dict(  # 这里是 Dataset_A 的原始配置
            type='Dataset_A',
            ...
            pipeline=train_pipeline
        )
    )
    ...
)
```

更加具体的细节，请参考 API 文档。

CHAPTER 7

教程 4：如何设计数据处理流程

7.1 设计数据流水线

按照典型的用法，我们通过 `Dataset` 和 `DataLoader` 来使用多个 worker 进行数据加载。对 `Dataset` 的索引操作将返回一个与模型的 `forward` 方法的参数相对应的字典。

数据流水线和数据集在这里是解耦的。通常，数据集定义如何处理标注文件，而数据流水线定义所有准备数据字典的步骤。流水线由一系列操作组成。每个操作都将一个字典作为输入，并输出一个字典。

这些操作分为数据加载，预处理和格式化。

这里使用 ResNet-50 在 ImageNet 数据集上的数据流水线作为示例。

```
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='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'])
]
test_pipeline = [
    dict(type='LoadImageFromFile'),
```

(下页继续)

(续上页)

```
    dict(type='Resize', size=256),
    dict(type='CenterCrop', crop_size=224),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='ImageToTensor', keys=['img']),
    dict(type='Collect', keys=['img'])
]
```

对于每个操作，我们列出了添加、更新、删除的相关字典字段。在流水线的最后，我们使用 `Collect` 仅保留进行模型 `forward` 方法所需的项。

7.1.1 数据加载

`LoadImageFromFile` - 从文件中加载图像

- 添加：img, img_shape, ori_shape

默认情况下，`LoadImageFromFile` 将会直接从硬盘加载图像，但对于一些效率较高、规模较小的模型，这可能会导致 IO 瓶颈。MMCV 支持多种数据加载后端来加速这一过程。例如，如果训练设备上配置了 memcached，那么我们按照如下方式修改配置文件。

```
memcached_root = '/mnt/xxx/memcached_client/'
train_pipeline = [
    dict(
        type='LoadImageFromFile',
        file_client_args=dict(
            backend='memcached',
            server_list_cfg=osp.join(memcached_root, 'server_list.conf'),
            client_cfg=osp.join(memcached_root, 'client.conf'))),
]
```

更多支持的数据加载后端，可以参见 mmcv.fileio.FileClient。

7.1.2 预处理

`Resize` - 缩放图像尺寸

- 添加：scale, scale_idx, pad_shape, scale_factor, keep_ratio
- 更新：img, img_shape

`RandomFlip` - 随机翻转图像

- 添加：flip, flip_direction
- 更新：img

`RandomCrop` - 随机裁剪图像

- 更新：img, pad_shape

`Normalize` - 图像数据归一化

- 添加：img_norm_cfg
- 更新：img

7.1.3 格式化

`ToTensor` - 转换（标签）数据至 `torch.Tensor`

- 更新：根据参数 `keys` 指定

`ImageToTensor` - 转换图像数据至 `torch.Tensor`

- 更新：根据参数 `keys` 指定

`Collect` - 保留指定键值

- 删除：除了参数 `keys` 指定以外的所有键值对

7.2 扩展及使用自定义流水线

1. 编写一个新的数据处理操作，并放置在 `mmcls/datasets/pipelines/` 目录下的任何一个文件中，例如 `my_pipeline.py`。这个类需要重载 `__call__` 方法，接受一个字典作为输入，并返回一个字典。

```
from mmcls.datasets import PIPELINES

@PIPELINES.register_module()
class MyTransform(object):

    def __call__(self, results):
        # 对 results['img'] 进行变换操作
        return results
```

2. 在 `mmcls/datasets/pipelines/__init__.py` 中导入这个新的类。

```
...
from .my_pipeline import MyTransform

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

3. 在数据流水线的配置中添加这一操作。

```
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='RandomResizedCrop', size=224),
    dict(type='RandomFlip', flip_prob=0.5, direction='horizontal'),
    dict(type='MyTransform'),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='ImageToTensor', keys=['img']),
    dict(type='ToTensor', keys=['gt_label']),
    dict(type='Collect', keys=['img', 'gt_label'])
]
```

7.3 流水线可视化

设计好数据流水线后，可以使用可视化工具查看具体的效果。

CHAPTER 8

教程 5：如何增加新模块

8.1 开发新组件

我们基本上将模型组件分为 3 种类型。

- 主干网络：通常是一个特征提取网络，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

8.1.1 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，将 ResNet 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。

它继承自 `ResNet` 并只修改了 stem 层。

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

```
import torch.nn as nn

from ..builder import BACKBONES
from .resnet import ResNet
```

(下页继续)

(续上页)

```
@BACKBONES.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):  # 需要返回一个元组
        pass  # 此处省略了网络的前向实现

    def init_weights(self, pretrained=None):
        pass  # 如果有必要的话，重载基类 ResNet 的参数初始化函数

    def train(self, mode=True):
        pass  # 如果有必要的话，重载基类 ResNet 的训练状态函数
```

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),
    ...
```

8.1.2 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

```
import torch.nn as nn

from ..builder import NECKS

@NECKS.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'),
)
```

8.1.3 添加新的头部组件

在此，我们以 LinearClsHead 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 forward_train 函数，它接受来自颈部或主干网络的特征图作为输入，并基于真实标签计算。

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

```
from ..builder import HEADS
from .cls_head import ClsHead


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

    def __init__(self,
                 num_classes,
                 in_channels,
                 loss=dict(type='CrossEntropyLoss', loss_weight=1.0),
                 topk=(1, )):
        super(LinearClsHead, self).__init__(loss=loss, topk=topk)
        self.in_channels = in_channels
        self.num_classes = num_classes

        if self.num_classes <= 0:
            raise ValueError(
                f'num_classes={num_classes} must be a positive integer')

        self._init_layers()
```

(下页继续)

(续上页)

```
    def _init_layers(self):
        self.fc = nn.Linear(self.in_channels, self.num_classes)

    def init_weights(self):
        normal_init(self.fc, mean=0, std=0.01, bias=0)

    def forward_train(self, x, gt_label):
        cls_score = self.fc(x)
        losses = self.loss(cls_score, gt_label)
        return losses
```

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

```
...
from .linear_head import LinearClsHead

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

3. 修改配置文件以使用新的头部组件。

连同 GlobalAveragePooling 颈部组件，完整的模型配置如下：

```
model = dict(
    type='ImageClassifier',
    backbone=dict(
        type='ResNet',
        depth=50,
        num_stages=4,
        out_indices=(3, ),
        style='pytorch'),
    neck=dict(type='GlobalAveragePooling'),
    head=dict(
        type='LinearClsHead',
        num_classes=1000,
        in_channels=2048,
        loss=dict(type='CrossEntropyLoss', loss_weight=1.0),
        topk=(1, 5),
    ))
```

8.1.4 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。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 ..builder import LOSSES
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

@LOSSES.register_module()
class L1Loss(nn.Module):

    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, l1_loss
```

(下页继续)

(续上页)

```
__all__ = [
    ..., 'L1Loss', 'l1_loss'
]
```

3. 修改配置文件中的 `loss` 字段以使用新的损失函数

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

CHAPTER 9

教程 6：如何自定义优化策略

在本教程中，我们将介绍如何在运行自定义模型时，进行构造优化器、定制学习率及动量调整策略、梯度裁剪、梯度累计以及用户自定义优化方法等。

9.1 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“optimizer”字段。例如，如果要使用“SGD”，则修改如下。

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

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。

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。

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

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

9.2 定制学习率调整策略

9.2.1 定制学习率衰减曲线

深度学习研究中，广泛应用学习率衰减来提高网络的性能。要使用学习率衰减，可以在配置中设置 `lr_confg` 字段。

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，配置文件为：

```
lr_config = dict(policy='step', step=[100, 150])
```

在训练过程中，程序会周期性地调用 MMCV 中的 `StepLRHook` 来进行学习率更新。

此外，我们也支持其他学习率调整方法，如 `CosineAnnealing` 和 `Poly` 等。详情可见 这里

- ConsineAnnealing:

  ```
  lr_config = dict(policy='CosineAnnealing', min_lr_ratio=1e-5)
  ```

- Poly:

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

9.2.2 定制学习率预热策略

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。通过预热，学习率将会从一个很小的值逐步提高到预定值。

在 MMClassification 中，我们同样使用 `lr_config` 配置学习率预热策略，主要的参数有以下几个：

- `warmup`：学习率预热曲线类别，必须为‘constant’、‘linear’，‘exp’或者 `None` 其一，如果为 `None`，则不使用学习率预热策略。
- `warmup_by_epoch`：是否以轮次（epoch）为单位进行预热。
- `warmup_iters`：预热的迭代次数，当 `warmup_by_epoch=True` 时，单位为轮次（epoch）；当 `warmup_by_epoch=False` 时，单位为迭代次数（iter）。
- `warmup_ratio`：预测的初始学习率 `lr = lr * warmup_ratio`。

例如：

1. 逐**迭代次数**地**线性**预热

```
lr_config = dict(
    policy='CosineAnnealing',
    by_epoch=False,
    min_lr_ratio=1e-2,
    warmup='linear',
    warmup_ratio=1e-3,
    warmup_iters=20 * 1252,
    warmup_by_epoch=False)
```

2. 逐**轮次**地**指数**预热

```
lr_config = dict(
    policy='CosineAnnealing',
    min_lr=0,
    warmup='exp',
    warmup_iters=5,
    warmup_ratio=0.1,
    warmup_by_epoch=True)
```

小技巧： 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。

9.3 定制动量调整策略

MMClassification 支持动量调整器根据学习率修改模型的动量，从而使模型收敛更快。

动量调整程序通常与学习率调整器一起使用，例如，以下配置用于加速收敛。更多细节可参考 CyclicLrUpdater 和 CyclicMomentumUpdater。

这里是一个用例：

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

9.4 参数化精细配置

一些模型可能具有一些特定于参数的设置以进行优化，例如 BatchNorm 层不添加权重衰减或者对不同的网络层使用不同的学习率。在 MMClassification 中，我们通过 `optimizer` 的 `paramwise_cfg` 参数进行配置，可以参考MMCV。

- 使用指定选项

  MMClassification 提供了包括 `bias_lr_mult`、`bias_decay_mult`、`norm_decay_mult`、`dwconv_decay_mult`、`dcn_offset_lr_mult` 和 `bypass_duplicate` 选项，指定相关所有的 `bais`、`norm`、`dwconv`、`dcn` 和 `bypass` 参数。例如令模型中所有的 BN 不进行参数衰减：

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

- 使用 `custom_keys` 指定参数

  MMClassification 可通过 `custom_keys` 指定不同的参数使用不同的学习率或者权重衰减，例如对特定的参数不使用权重衰减：

```
paramwise_cfg = dict(
    custom_keys={
        'backbone.cls_token': dict(decay_mult=0.0),
        'backbone.pos_embed': dict(decay_mult=0.0)
    })

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

对 backbone 使用更小的学习率与衰减系数：

```
optimizer = dict(
    type='SGD',
    lr=0.8,
    weight_decay=1e-4,
    # backbone 的 'lr' and 'weight_decay' 分别为 0.1 * lr 和 0.9 * weight_decay
    paramwise_cfg = dict(custom_keys={'backbone': dict(lr_mult=0.1, decay_mult=0.
↪9)}))
```

9.5 梯度裁剪与梯度累计

除了 PyTorch 优化器的基本功能，我们还提供了一些对优化器的增强功能，例如梯度裁剪、梯度累计等，参考 MMCV。

9.5.1 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 `optimizer_config` 字段中添加 `grad_clip` 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：

```
# norm_type: 使用的范数类型，此处使用范数2。
optimizer_config = dict(grad_clip=dict(max_norm=35, norm_type=2))
```

当使用继承并修改基础配置方式时，如果基础配置中 `grad_clip=None`，需要添加 `_delete_=True`。有关 `_delete_` 可以参考教程 1：如何编写配置文件。案例如下：

```
_base_ = [./_base_/schedules/imagenet_bs256_coslr.py]

optimizer_config = dict(grad_clip=dict(max_norm=35, norm_type=2), _delete_=True, type=
↪'OptimizerHook')
# 当 type 为 'OptimizerHook'，可以省略 type；其他情况下，此处必须指明 type=
↪'xxxOptimizerHook'。
```

9.5.2 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。

用例如下：

```
data = dict(samples_per_gpu=64)
optimizer_config = dict(type="GradientCumulativeOptimizerHook", cumulative_iters=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：

```
data = dict(samples_per_gpu=256)
optimizer_config = dict(type="OptimizerHook")
```

备注： 当在 `optimizer_config` 不指定优化器钩子类型时，默认使用 `OptimizerHook`。

9.6 用户自定义优化方法

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。

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。

9.6.1 自定义优化器

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/core/optimizer 的文件夹，并在目录下的一个文件，如 mmcls/core/optimizer/my_optimizer.py 中实现该自定义优化器：

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


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

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

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/core/optimizer/__init__.py，将其导入至 optimizer 包；再修改 mmcls/core/__init__.py 以导入 optimizer 包

  创建 mmcls/core/optimizer/__init__.py 文件。新定义的模块应导入到 mmcls/core/optimizer/__init__.py 中，以便注册器能找到新模块并将其添加：

```
# 在 mmcls/core/optimizer/__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 mmcls/core/__init__.py 中
...
from .optimizer import *  # noqa: F401, F403
```

- 在配置中使用 custom_imports 手动导入

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

mmcls.core.optimizer.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，只有包含 MyOptmizer 类的包会被导入。mmcls.core.optimizer.my_optimizer.MyOptimizer **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 optimizer 域中使用 MyOptimizer。在配置中，优化器由 “optimizer” 字段定义，如下所示：

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

要使用自定义的优化器，可以将该字段更改为

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

9.6.2 自定义优化器构造器

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 BatchNorm 层的权重衰减。

虽然我们的 DefaultOptimizerConstructor 已经提供了这些强大的功能，但可能仍然无法覆盖需求。此时我们可以通过自定义优化器构造函数来进行其他细粒度的参数调整。

```
from mmcv.runner.optimizer import OPTIMIZER_BUILDERS


@OPTIMIZER_BUILDERS.register_module()
class MyOptimizerConstructor:

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

    def __call__(self, model):
        ...    # 在这里实现自己的优化器构造器。
        return my_optimizer
```

这里是我们默认的优化器构造器的实现，可以作为新优化器构造器实现的模板。

CHAPTER 10

教程 7：如何自定义模型运行参数

在本教程中，我们将介绍如何在运行自定义模型时，进行自定义工作流和钩子的方法。

10.1 定制工作流

工作流是一个形如 (任务名，周期数) 的列表，用于指定运行顺序和周期。这里“周期数”的单位由执行器的类型来决定。

比如在 MMClassification 中，我们默认使用基于**轮次**的执行器（`EpochBasedRunner`），那么“周期数”指的就是对应的任务在一个周期中要执行多少个轮次。通常，我们只希望执行训练任务，那么只需要使用以下设置：

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

有时我们可能希望在训练过程中穿插检查模型在验证集上的一些指标（例如，损失，准确性）。

在这种情况下，可以将工作流程设置为：

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

这样一来，程序会一轮训练一轮测试地反复执行。

需要注意的是，默认情况下，我们并不推荐用这种方式来进行模型验证，而是推荐在训练中使用 **EvalHook** 进行模型验证。使用上述工作流的方式进行模型验证只是一个替代方案。

备注:

1. 在验证周期时不会更新模型参数。
2. 配置文件内的关键词 `max_epochs` 控制训练时期数，并且不会影响验证工作流程。
3. 工作流 `[('train', 1), ('val', 1)]` 和 `[('train', 1)]` 不会改变 `EvalHook` 的行为。因为 `EvalHook` 由 `after_train_epoch` 调用，而验证工作流只会影响 `after_val_epoch` 调用的钩子。因此，`[('train', 1), ('val', 1)]` 和 `[('train', 1)]` 的区别在于，runner 在完成每一轮训练后，会计算验证集上的损失。

10.2 钩子

钩子机制在 OpenMMLab 开源算法库中应用非常广泛，结合执行器可以实现对训练过程的整个生命周期进行管理，可以通过相关文章进一步理解钩子。

钩子只有在构造器中被注册才起作用，目前钩子主要分为两类：

- 默认训练钩子

默认训练钩子由运行器默认注册，一般为一些基础型功能的钩子，已经有确定的优先级，一般不需要修改优先级。

- 定制钩子

定制钩子通过 `custom_hooks` 注册，一般为一些增强型功能的钩子，需要在配置文件中指定优先级，不指定该钩子的优先级将默被设定为‘NORMAL’。

优先级列表

Level	Value
HIGHEST	0
VERY_HIGH	10
HIGH	30
ABOVE_NORMAL	40
NORMAL(default)	50
BELOW_NORMAL	60
LOW	70
VERY_LOW	90
LOWEST	100

优先级确定钩子的执行顺序，每次训练前，日志会打印出各个阶段钩子的执行顺序，方便调试。

10.2.1 默认训练钩子

有一些常见的钩子未通过 `custom_hooks` 注册，但会在运行器（`Runner`）中默认注册，它们是：

Hooks	Priority
`LrUpdaterHook`	VERY_HIGH (10)
`MomentumUpdaterHook`	HIGH (30)
`OptimizerHook`	ABOVE_NORMAL (40)
`CheckpointHook`	NORMAL (50)
`IterTimerHook`	LOW (70)
`EvalHook`	LOW (70)
`LoggerHook(s)`	VERY_LOW (90)

`OptimizerHook`，`MomentumUpdaterHook` 和 `LrUpdaterHook` 在优化策略 部分进行了介绍，`IterTimerHook` 用于记录所用时间，目前不支持修改；

下面介绍如何使用去定制 `CheckpointHook`、`LoggerHooks` 以及 `EvalHook`。

权重文件钩子（CheckpointHook）

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

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

用户可以设置 “max_keep_ckpts” 来仅保存少量模型权重文件，或者通过 “save_optimizer” 决定是否存储优化器的状态字典。更多细节可参考 这里。

日志钩子（LoggerHooks）

log_config 包装了多个记录器钩子，并可以设置间隔。目前，MMCV 支持 TextLoggerHook、WandbLoggerHook、MlflowLoggerHook 和 TensorboardLoggerHook。更多细节可参考这里。

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

验证钩子（EvalHook）

配置中的 evaluation 字段将用于初始化 EvalHook。

EvalHook 有一些保留参数，如 interval，save_best 和 start 等。其他的参数，如 “metrics” 将被传递给 dataset.evaluate()。

```
evaluation = dict(interval=1, metric='accuracy', metric_options={'topk': (1, )})
```

我们可以通过参数 save_best 保存取得最好验证结果时的模型权重：

```
# "auto" 表示自动选择指标来进行模型的比较。也可以指定一个特定的 key 比如 "accuracy_
↪top-1"。
evaluation = dict(interval=1, save_best=True, metric='accuracy', metric_options={'topk
↪': (1, )})
```

在跑一些大型实验时，可以通过修改参数 start 跳过训练靠前轮次时的验证步骤，以节约时间。如下：

```
evaluation = dict(interval=1, start=200, metric='accuracy', metric_options={'topk':␣
↪(1, )})
```

表示在第 200 轮之前，只执行训练流程，不执行验证；从轮次 200 开始，在每一轮训练之后进行验证。

备注： 在 MMClassification 的默认配置文件中，evaluation 字段一般被放在 datasets 基础配置文件中。

10.2.2 使用内置钩子

一些钩子已在 MMCV 和 MMClassification 中实现：

- EMAHook
- SyncBuffersHook
- EmptyCacheHook
- ProfilerHook
- ……

可以直接修改配置以使用该钩子，如下格式：

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

例如使用 EMAHook，进行一次 EMA 的间隔是 100 个迭代：

```
custom_hooks = [
    dict(type='EMAHook', interval=100, priority='HIGH')
]
```

10.3 自定义钩子

10.3.1 创建一个新钩子

这里举一个在 MMClassification 中创建一个新钩子，并在训练中使用它的示例：

```
from mmcv.runner import HOOKS, Hook


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

    def __init__(self, a, b):
        pass

    def before_run(self, runner):
        pass

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

(下页继续)

(续上页)

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

    def after_epoch(self, runner):
        pass

    def before_iter(self, runner):
        pass

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

根据钩子的功能，用户需要指定钩子在训练的每个阶段将要执行的操作，比如 before_run，after_run，before_epoch，after_epoch，before_iter 和 after_iter。

10.3.2 注册新钩子

之后，需要导入 MyHook。假设该文件在 mmcls/core/utils/my_hook.py，有两种办法导入它：

- 修改 mmcls/core/utils/__init__.py 进行导入

  新定义的模块应导入到 mmcls/core/utils/__init__py 中，以便注册器能找到并添加新模块：

```
from .my_hook import MyHook

__all__ = ['MyHook']
```

- 使用配置文件中的 custom_imports 变量手动导入

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

10.3.3 修改配置

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

还可通过 priority 参数设置钩子优先级，如下所示：

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

默认情况下，在注册过程中，钩子的优先级设置为“NORMAL”。

10.4 常见问题

10.4.1 1. resume_from，load_from，init_cfg.Pretrained 区别

- `load_from`： 仅仅加载模型权重，主要用于加载预训练或者训练好的模型；
- `resume_from`： 不仅导入模型权重，还会导入优化器信息，当前轮次（epoch）信息，主要用于从断点继续训练。
- `init_cfg.Pretrained`： 在权重初始化期间加载权重，您可以指定要加载的模块。这通常在微调模型时使用，请参阅*教程 2：如何微调模型*

CHAPTER 11

模型库统计

- 论文数量：34
  - ALGORITHM: 34
- 模型权重文件数量：224
  - [ALGORITHM] *Conformer: Local Features Coupling Global Representations for Visual Recognition* (4 ckpts)
  - [ALGORITHM] *Patches Are All You Need?* (3 ckpts)
  - [ALGORITHM] *A ConvNet for the 2020s* (13 ckpts)
  - [ALGORITHM] *CSPNet: A New Backbone that can Enhance Learning Capability of CNN* (3 ckpts)
  - [ALGORITHM] *Residual Attention: A Simple but Effective Method for Multi-Label Recognition* (1 ckpts)
  - [ALGORITHM] *Training data-efficient image transformers & distillation through attention* (9 ckpts)
  - [ALGORITHM] *Densely Connected Convolutional Networks* (4 ckpts)
  - [ALGORITHM] *EfficientFormer: Vision Transformers at MobileNet Speed* (3 ckpts)
  - [ALGORITHM] *Rethinking Model Scaling for Convolutional Neural Networks* (23 ckpts)
  - [ALGORITHM] *HorNet: Efficient High-Order Spatial Interactions with Recursive Gated Convolutions* (9 ckpts)
  - [ALGORITHM] *Deep High-Resolution Representation Learning for Visual Recognition* (9 ckpts)
  - [ALGORITHM] *MLP-Mixer: An all-MLP Architecture for Vision* (2 ckpts)
  - [ALGORITHM] *MobileNetV2: Inverted Residuals and Linear Bottlenecks* (1 ckpts)

- [ALGORITHM] *Searching for MobileNetV3* (2 ckpts)
- [ALGORITHM] *MViTv2: Improved Multiscale Vision Transformers for Classification and Detection* (4 ckpts)
- [ALGORITHM] *MetaFormer is Actually What You Need for Vision* (5 ckpts)
- [ALGORITHM] *Designing Network Design Spaces* (16 ckpts)
- [ALGORITHM] *RepMLP: Re-parameterizing Convolutions into Fully-connected Layers forImage Recognition* (2 ckpts)
- [ALGORITHM] *Repvgg: Making vgg-style convnets great again* (12 ckpts)
- [ALGORITHM] *Res2Net: A New Multi-scale Backbone Architecture* (3 ckpts)
- [ALGORITHM] *Deep Residual Learning for Image Recognition* (26 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] *Swin Transformer V2: Scaling Up Capacity and Resolution* (12 ckpts)
- [ALGORITHM] *Tokens-to-Token ViT: Training Vision Transformers from Scratch on ImageNet* (3 ckpts)
- [ALGORITHM] *Transformer in Transformer* (1 ckpts)
- [ALGORITHM] *Twins: Revisiting the Design of Spatial Attention in Vision Transformers* (6 ckpts)
- [ALGORITHM] *Visual Attention Network* (8 ckpts)
- [ALGORITHM] *Very Deep Convolutional Networks for Large-Scale Image Recognition* (8 ckpts)
- [ALGORITHM] *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale* (7 ckpts)
- [ALGORITHM] *Wide Residual Networks* (3 ckpts)

CHAPTER 12

Model Zoo

12.1 ImageNet

ImageNet has multiple versions, but the most commonly used one is ILSVRC 2012. The ResNet family models below are trained by standard data augmentations, i.e., RandomResizedCrop, RandomHorizontalFlip and Normalize.

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	C
VGG-11	132.86	7.63	68.75	88.87	co
VGG-13	133.05	11.34	70.02	89.46	co
VGG-16	138.36	15.5	71.62	90.49	co
VGG-19	143.67	19.67	72.41	90.80	co
VGG-11-BN	132.87	7.64	70.75	90.12	co
VGG-13-BN	133.05	11.36	72.15	90.71	co
VGG-16-BN	138.37	15.53	73.72	91.68	co
VGG-19-BN	143.68	19.7	74.70	92.24	co
RepVGG-A0*	9.11 （train) | 8.31 (deploy)	1.52 (train) | 1.36 (deploy)	72.41	90.50	co
RepVGG-A1*	14.09 (train) | 12.79 (deploy)	2.64 (train) | 2.37 (deploy)	74.47	91.85	co
RepVGG-A2*	28.21 (train) | 25.5 (deploy)	5.7 (train) | 5.12 (deploy)	76.48	93.01	co
RepVGG-B0*	15.82 (train) | 14.34 (deploy)	3.42 (train) | 3.06 (deploy)	75.14	92.42	co
RepVGG-B1*	57.42 (train) | 51.83 (deploy)	13.16 (train) | 11.82 (deploy)	78.37	94.11	co
RepVGG-B1g2*	45.78 (train) | 41.36 (deploy)	9.82 (train) | 8.82 (deploy)	77.79	93.88	co
RepVGG-B1g4*	39.97 (train) | 36.13 (deploy)	8.15 (train) | 7.32 (deploy)	77.58	93.84	co

表 1 – 续上页

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	C
RepVGG-B2*	89.02 (train) | 80.32 (deploy)	20.46 (train) | 18.39 (deploy)	78.78	94.42	co
RepVGG-B2g4*	61.76 (train) | 55.78 (deploy)	12.63 (train) | 11.34 (deploy)	79.38	94.68	co
RepVGG-B3*	123.09 (train) | 110.96 (deploy)	29.17 (train) | 26.22 (deploy)	80.52	95.26	co
RepVGG-B3g4*	83.83 (train) | 75.63 (deploy)	17.9 (train) | 16.08 (deploy)	80.22	95.10	co
RepVGG-D2se*	133.33 (train) | 120.39 (deploy)	36.56 (train) | 32.85 (deploy)	81.81	95.94	co
ResNet-18	11.69	1.82	70.07	89.44	co
ResNet-34	21.8	3.68	73.85	91.53	co
ResNet-50 (rsb-a1)	25.56	4.12	80.12	94.78	co
ResNet-101	44.55	7.85	78.18	94.03	co
ResNet-152	60.19	11.58	78.63	94.16	co
Res2Net-50-14w-8s*	25.06	4.22	78.14	93.85	co
Res2Net-50-26w-8s*	48.40	8.39	79.20	94.36	co
Res2Net-101-26w-4s*	45.21	8.12	79.19	94.44	co
ResNeSt-50*	27.48	5.41	81.13	95.59	co
ResNeSt-101*	48.28	10.27	82.32	96.24	co
ResNeSt-200*	70.2	17.53	82.41	96.22	co
ResNeSt-269*	110.93	22.58	82.70	96.28	co
ResNetV1D-50	25.58	4.36	77.54	93.57	co
ResNetV1D-101	44.57	8.09	78.93	94.48	co
ResNetV1D-152	60.21	11.82	79.41	94.7	co
ResNeXt-32x4d-50	25.03	4.27	77.90	93.66	co
ResNeXt-32x4d-101	44.18	8.03	78.71	94.12	co
ResNeXt-32x8d-101	88.79	16.5	79.23	94.58	co
ResNeXt-32x4d-152	59.95	11.8	78.93	94.41	co
SE-ResNet-50	28.09	4.13	77.74	93.84	co
SE-ResNet-101	49.33	7.86	78.26	94.07	co
RegNetX-400MF	5.16	0.41	72.56	90.78	co
RegNetX-800MF	7.26	0.81	74.76	92.32	co
RegNetX-1.6GF	9.19	1.63	76.84	93.31	co
RegNetX-3.2GF	15.3	3.21	78.09	94.08	co
RegNetX-4.0GF	22.12	4.0	78.60	94.17	co
RegNetX-6.4GF	26.21	6.51	79.38	94.65	co
RegNetX-8.0GF	39.57	8.03	79.12	94.51	co
RegNetX-12GF	46.11	12.15	79.67	95.03	co
ShuffleNetV1 1.0x (group=3)	1.87	0.146	68.13	87.81	co
ShuffleNetV2 1.0x	2.28	0.149	69.55	88.92	co
MobileNet V2	3.5	0.319	71.86	90.42	co

表 1 – 续上页

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	C
ViT-B/16*	86.86	33.03	85.43	97.77	co
ViT-B/32*	88.3	8.56	84.01	97.08	co
ViT-L/16*	304.72	116.68	85.63	97.63	co
Swin-Transformer tiny	28.29	4.36	81.18	95.61	co
Swin-Transformer small	49.61	8.52	83.02	96.29	co
Swin-Transformer base	87.77	15.14	83.36	96.44	co
Transformer in Transformer small*	23.76	3.36	81.52	95.73	co
T2T-ViT_t-14	21.47	4.34	81.83	95.84	co
T2T-ViT_t-19	39.08	7.80	82.63	96.18	co
T2T-ViT_t-24	64.00	12.69	82.71	96.09	co
Mixer-B/16*	59.88	12.61	76.68	92.25	co
Mixer-L/16*	208.2	44.57	72.34	88.02	co
DeiT-tiny	5.72	1.08	74.50	92.24	co
DeiT-tiny distilled*	5.72	1.08	74.51	91.90	co
DeiT-small	22.05	4.24	80.69	95.06	co
DeiT-small distilled*	22.05	4.24	81.17	95.40	co
DeiT-base	86.57	16.86	81.76	95.81	co
DeiT-base distilled*	86.57	16.86	83.33	96.49	co
DeiT-base 384px*	86.86	49.37	83.04	96.31	co
DeiT-base distilled 384px*	86.86	49.37	85.55	97.35	co
Conformer-tiny-p16*	23.52	4.90	81.31	95.60	co
Conformer-small-p32*	38.85	7.09	81.96	96.02	co
Conformer-small-p16*	37.67	10.31	83.32	96.46	co
Conformer-base-p16*	83.29	22.89	83.82	96.59	co
PCPVT-small*	24.11	3.67	81.14	95.69	co
PCPVT-base*	43.83	6.45	82.66	96.26	co
PCPVT-large*	60.99	9.51	83.09	96.59	co
SVT-small*	24.06	2.82	81.77	95.57	co
SVT-base*	56.07	8.35	83.13	96.29	co
SVT-large*	99.27	14.82	83.60	96.50	co
EfficientNet-B0*	5.29	0.02	76.74	93.17	co
EfficientNet-B0 (AA)*	5.29	0.02	77.26	93.41	co
EfficientNet-B0 (AA + AdvProp)*	5.29	0.02	77.53	93.61	co
EfficientNet-B1*	7.79	0.03	78.68	94.28	co
EfficientNet-B1 (AA)*	7.79	0.03	79.20	94.42	co
EfficientNet-B1 (AA + AdvProp)*	7.79	0.03	79.52	94.43	co
EfficientNet-B2*	9.11	0.03	79.64	94.80	co

表 1 – 续上页

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	C
EfficientNet-B2 (AA)*	9.11	0.03	80.21	94.96	co
EfficientNet-B2 (AA + AdvProp)*	9.11	0.03	80.45	95.07	co
EfficientNet-B3*	12.23	0.06	81.01	95.34	co
EfficientNet-B3 (AA)*	12.23	0.06	81.58	95.67	co
EfficientNet-B3 (AA + AdvProp)*	12.23	0.06	81.81	95.69	co
EfficientNet-B4*	19.34	0.12	82.57	96.09	co
EfficientNet-B4 (AA)*	19.34	0.12	82.95	96.26	co
EfficientNet-B4 (AA + AdvProp)*	19.34	0.12	83.25	96.44	co
EfficientNet-B5*	30.39	0.24	83.18	96.47	co
EfficientNet-B5 (AA)*	30.39	0.24	83.82	96.76	co
EfficientNet-B5 (AA + AdvProp)*	30.39	0.24	84.21	96.98	co
EfficientNet-B6 (AA)*	43.04	0.41	84.05	96.82	co
EfficientNet-B6 (AA + AdvProp)*	43.04	0.41	84.74	97.14	co
EfficientNet-B7 (AA)*	66.35	0.72	84.38	96.88	co
EfficientNet-B7 (AA + AdvProp)*	66.35	0.72	85.14	97.23	co
EfficientNet-B8 (AA + AdvProp)*	87.41	1.09	85.38	97.28	co
ConvNeXt-T*	28.59	4.46	82.05	95.86	co
ConvNeXt-S*	50.22	8.69	83.13	96.44	co
ConvNeXt-B*	88.59	15.36	83.85	96.74	co
ConvNeXt-B*	88.59	15.36	85.81	97.86	co
ConvNeXt-L*	197.77	34.37	84.30	96.89	co
ConvNeXt-L*	197.77	34.37	86.61	98.04	co
ConvNeXt-XL*	350.20	60.93	86.97	98.20	co
HRNet-W18*	21.30	4.33	76.75	93.44	co
HRNet-W30*	37.71	8.17	78.19	94.22	co
HRNet-W32*	41.23	8.99	78.44	94.19	co
HRNet-W40*	57.55	12.77	78.94	94.47	co
HRNet-W44*	67.06	14.96	78.88	94.37	co
HRNet-W48*	77.47	17.36	79.32	94.52	co
HRNet-W64*	128.06	29.00	79.46	94.65	co
HRNet-W18 (ssld)*	21.30	4.33	81.06	95.70	co
HRNet-W48 (ssld)*	77.47	17.36	83.63	96.79	co
WRN-50*	68.88	11.44	81.45	95.53	co
WRN-101*	126.89	22.81	78.84	94.28	co
CSPDarkNet50*	27.64	5.04	80.05	95.07	co
CSPResNet50*	21.62	3.48	79.55	94.68	co
CSPResNeXt50*	20.57	3.11	79.96	94.96	co

表 1 – 续上页

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	C
DenseNet121*	7.98	2.88	74.96	92.21	co
DenseNet169*	14.15	3.42	76.08	93.11	co
DenseNet201*	20.01	4.37	77.32	93.64	co
DenseNet161*	28.68	7.82	77.61	93.83	co
VAN-T*	4.11	0.88	75.41	93.02	co
VAN-S*	13.86	2.52	81.01	95.63	co
VAN-B*	26.58	5.03	82.80	96.21	co
VAN-L*	44.77	8.99	83.86	96.73	co
MViTv2-tiny*	24.17	4.70	82.33	96.15	co
MViTv2-small*	34.87	7.00	83.63	96.51	co
MViTv2-base*	51.47	10.20	84.34	96.86	co
MViTv2-large*	217.99	42.10	85.25	97.14	co
EfficientFormer-l1*	12.19	1.30	80.46	94.99	co
EfficientFormer-l3*	31.41	3.93	82.45	96.18	co
EfficientFormer-l7*	82.23	10.16	83.40	96.60	co

*Models with * are converted from other repos, others are trained by ourselves.*

12.2 CIFAR10

Model	Params(M)	Flops(G)	Top-1 (%)	Config	Download
ResNet-18-b16x8	11.17	0.56	94.82		config
ResNet-34-b16x8	21.28	1.16	95.34		config
ResNet-50-b16x8	23.52	1.31	95.55		config
ResNet-101-b16x8	42.51	2.52	95.58		config
ResNet-152-b16x8	58.16	3.74	95.76		config

CHAPTER 13

Conformer

Conformer: Local Features Coupling Global Representations for Visual Recognition

13.1 Abstract

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.

13.2 Results and models

13.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
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.*

13.3 Citation

```
@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 14

ConvMixer

Patches Are All You Need?

14.1 Abstract

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.

14.2 Results and models

14.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
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.*

14.3 Citation

```
@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 15

ConvNeXt

A ConvNet for the 2020s

15.1 Abstract

The "Roaring 20s" of visual recognition began with the introduction of Vision Transformers (ViTs), which quickly superseded ConvNets as the state-of-the-art image classification model. A vanilla ViT, on the other hand, faces difficulties when applied to general computer vision tasks such as object detection and semantic segmentation. It is the hierarchical Transformers (e.g., Swin Transformers) that reintroduced several ConvNet priors, making Transformers practically viable as a generic vision backbone and demonstrating remarkable performance on a wide variety of vision tasks. However, the effectiveness of such hybrid approaches is still largely credited to the intrinsic superiority of Transformers, rather than the inherent inductive biases of convolutions. In this work, we reexamine the design spaces and test the limits of what a pure ConvNet can achieve. We gradually "modernize" a standard ResNet toward the design of a vision Transformer, and discover several key components that contribute to the performance difference along the way. The outcome of this exploration is a family of pure ConvNet models dubbed ConvNeXt. Constructed entirely from standard ConvNet modules, ConvNeXts compete favorably with Transformers in terms of accuracy and scalability, achieving 87.8% ImageNet top-1 accuracy and outperforming Swin Transformers on COCO detection and ADE20K segmentation, while maintaining the simplicity and efficiency of standard ConvNets.

15.2 Results and models

15.2.1 ImageNet-1k

Model	Pretrain	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
ConvNeXt-T*	From scratch	28.59	4.46	82.05	95.86	config	model
ConvNeXt-S*	From scratch	50.22	8.69	83.13	96.44	config	model
ConvNeXt-B*	From scratch	88.59	15.36	83.85	96.74	config	model
ConvNeXt-B*	ImageNet-21k	88.59	15.36	85.81	97.86	config	model
ConvNeXt-L*	From scratch	197.77	34.37	84.30	96.89	config	model
ConvNeXt-L*	ImageNet-21k	197.77	34.37	86.61	98.04	config	model
ConvNeXt-XL*	ImageNet-21k	350.20	60.93	86.97	98.20	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.*

15.2.2 Pre-trained Models

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

Model	Training Data	Params(M)	Flops(G)	Download
ConvNeXt-T*	ImageNet-1k	28.59	4.46	model
ConvNeXt-S*	ImageNet-1k	50.22	8.69	model
ConvNeXt-B*	ImageNet-1k	88.59	15.36	model
ConvNeXt-B*	ImageNet-21k	88.59	15.36	model
ConvNeXt-L*	ImageNet-21k	197.77	34.37	model
ConvNeXt-XL*	ImageNet-21k	350.20	60.93	model

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

15.3 Citation

```
@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},
```

(下页继续)

(续上页)

```
  year    = {2022},
}
```

CHAPTER 16

CSPNet

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

16.1 Abstract

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.

16.2 Results and models

16.2.1 ImageNet-1k

Model	Pretrain	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
CSPDarkNet50*	From scratch	27.64	5.04	80.05	95.07	config	model
CSPResNet50*	From scratch	21.62	3.48	79.55	94.68	config	model
CSPResNeXt50*	From scratch	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.*

16.3 Citation

```
@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 17

CSRA

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

17.1 Abstract

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.

17.2 Results and models

17.2.1 VOC2007

Model	Pretrain	Params(M)	Flops(G)	mAP	OF1 (%)	CF1 (%)	Config	Download
Resnet101-CSRA	ImageNet-1k	23.55	4.12	94.98	90.80	89.16	config	model | log

17.3 Citation

```
@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 18

DeiT

Training data-efficient image transformers & distillation through attention

18.1 Abstract

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.

18.2 Results and models

18.2.1 ImageNet-1k

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

Model	Pretrain	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Con-fig	Down-load
DeiT-tiny	From scratch	5.72	1.08	74.50	92.24	config	model | log
DeiT-tiny distilled*	From scratch	5.72	1.08	74.51	91.90	config	model
DeiT-small	From scratch	22.05	4.24	80.69	95.06	config	model | log
DeiT-small distilled*	From scratch	22.05	4.24	81.17	95.40	config	model
DeiT-base	From scratch	86.57	16.86	81.76	95.81	config	model | log
DeiT-base*	From scratch	86.57	16.86	81.79	95.59	config	model
DeiT-base distilled*	From scratch	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.

18.3 Citation

```
@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},
```

(下页继续)

(续上页)

```
  month =      {July}
}
```

CHAPTER 19

DenseNet

Densely Connected Convolutional Networks

19.1 Abstract

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 other 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.

19.2 Results and models

19.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
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.*

19.3 Citation

```
@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 20

EfficientFormer

EfficientFormer: Vision Transformers at MobileNet Speed

20.1 Abstract

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.

20.2 Results and models

20.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
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.*

20.3 Citation

```
@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 21

EfficientNet

Rethinking Model Scaling for Convolutional Neural Networks

21.1 Abstract

Convolutional Neural Networks (ConvNets) are commonly developed at a fixed resource budget, and then scaled up for better accuracy if more resources are available. In this paper, we systematically study model scaling and identify that carefully balancing network depth, width, and resolution can lead to better performance. Based on this observation, we propose a new scaling method that uniformly scales all dimensions of depth/width/resolution using a simple yet highly effective compound coefficient. We demonstrate the effectiveness of this method on scaling up MobileNets and ResNet. To go even further, we use neural architecture search to design a new baseline network and scale it up to obtain a family of models, called EfficientNets, which achieve much better accuracy and efficiency than previous ConvNets. In particular, our EfficientNet-B7 achieves state-of-the-art 84.3% top-1 accuracy on ImageNet, while being 8.4x smaller and 6.1x faster on inference than the best existing ConvNet. Our EfficientNets also transfer well and achieve state-of-the-art accuracy on CIFAR-100 (91.7%), Flowers (98.8%), and 3 other transfer learning datasets, with an order of magnitude fewer parameters.

21.2 Results and models

21.2.1 ImageNet-1k

In the result table, AA means trained with AutoAugment pre-processing, more details can be found in the paper, and AdvProp is a method to train with adversarial examples, more details can be found in the paper.

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

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
EfficientNet-B0*	5.29	0.02	76.74	93.17	config	model
EfficientNet-B0 (AA)*	5.29	0.02	77.26	93.41	config	model
EfficientNet-B0 (AA + AdvProp)*	5.29	0.02	77.53	93.61	config	model
EfficientNet-B1*	7.79	0.03	78.68	94.28	config	model
EfficientNet-B1 (AA)*	7.79	0.03	79.20	94.42	config	model
EfficientNet-B1 (AA + AdvProp)*	7.79	0.03	79.52	94.43	config	model
EfficientNet-B2*	9.11	0.03	79.64	94.80	config	model
EfficientNet-B2 (AA)*	9.11	0.03	80.21	94.96	config	model
EfficientNet-B2 (AA + AdvProp)*	9.11	0.03	80.45	95.07	config	model
EfficientNet-B3*	12.23	0.06	81.01	95.34	config	model
EfficientNet-B3 (AA)*	12.23	0.06	81.58	95.67	config	model
EfficientNet-B3 (AA + AdvProp)*	12.23	0.06	81.81	95.69	config	model
EfficientNet-B4*	19.34	0.12	82.57	96.09	config	model
EfficientNet-B4 (AA)*	19.34	0.12	82.95	96.26	config	model
EfficientNet-B4 (AA + AdvProp)*	19.34	0.12	83.25	96.44	config	model
EfficientNet-B5*	30.39	0.24	83.18	96.47	config	model
EfficientNet-B5 (AA)*	30.39	0.24	83.82	96.76	config	model
EfficientNet-B5 (AA + AdvProp)*	30.39	0.24	84.21	96.98	config	model
EfficientNet-B6 (AA)*	43.04	0.41	84.05	96.82	config	model
EfficientNet-B6 (AA + AdvProp)*	43.04	0.41	84.74	97.14	config	model
EfficientNet-B7 (AA)*	66.35	0.72	84.38	96.88	config	model
EfficientNet-B7 (AA + AdvProp)*	66.35	0.72	85.14	97.23	config	model
EfficientNet-B8 (AA + AdvProp)*	87.41	1.09	85.38	97.28	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.*

21.3 Citation

```
@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 22

HorNet

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

22.1 Abstract

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.

22.2 Results and models

22.2.1 ImageNet-1k

Model	Pretrain	resolution	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
HorNet-T*	From scratch	224x224	22.41	3.98	82.84	96.24	config	model
HorNet-T-GF*	From scratch	224x224	22.99	3.9	82.98	96.38	config	model
HorNet-S*	From scratch	224x224	49.53	8.83	83.79	96.75	config	model
HorNet-S-GF*	From scratch	224x224	50.4	8.71	83.98	96.77	config	model
HorNet-B*	From scratch	224x224	87.26	15.59	84.24	96.94	config	model
HorNet-B-GF*	From scratch	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.

22.2.2 Pre-trained Models

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

Model	Pretrain	resolution	Params(M)	Flops(G)	Download
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.

22.3 Citation

```
@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 23

HRNet

Deep High-Resolution Representation Learning for Visual Recognition

23.1 Abstract

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.

23.2 Results and models

23.3 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
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.*

23.4 Citation

```
@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 24

Mlp-Mixer

MLP-Mixer: An all-MLP Architecture for Vision

24.1 Abstract

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.

24.2 Results and models

24.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
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.*

24.3 Citation

```
@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 25

MobileNet V2

MobileNetV2: Inverted Residuals and Linear Bottlenecks

25.1 Abstract

In this paper we describe a new mobile architecture, MobileNetV2, that improves the state of the art performance of mobile models on multiple tasks and benchmarks as well as across a spectrum of different model sizes. We also describe efficient ways of applying these mobile models to object detection in a novel framework we call SSDLite. Additionally, we demonstrate how to build mobile semantic segmentation models through a reduced form of DeepLabv3 which we call Mobile DeepLabv3.

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

25.2 Results and models

25.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
MobileNet V2	3.5	0.319	71.86	90.42	config	model | log

25.3 Citation

```
@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 26

MobileNet V3

Searching for MobileNetV3

26.1 Abstract

We present the next generation of MobileNets based on a combination of complementary search techniques as well as a novel architecture design. MobileNetV3 is tuned to mobile phone CPUs through a combination of hardware-aware network architecture search (NAS) complemented by the NetAdapt algorithm and then subsequently improved through novel architecture advances. This paper starts the exploration of how automated search algorithms and network design can work together to harness complementary approaches improving the overall state of the art. Through this process we 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. For the task of semantic segmentation (or any dense pixel prediction), we propose a new efficient segmentation decoder Lite Reduced Atrous Spatial Pyramid Pooling (LR-ASPP). We achieve new state of the art results for mobile classification, detection and segmentation. MobileNetV3-Large is 3.2% more accurate on ImageNet classification while reducing latency by 15% compared to MobileNetV2. MobileNetV3-Small is 4.6% more accurate while reducing latency by 5% compared to MobileNetV2. MobileNetV3-Large detection is 25% faster at roughly the same accuracy as MobileNetV2 on COCO detection. MobileNetV3-Large LR-ASPP is 30% faster than MobileNetV2 R-ASPP at similar accuracy for Cityscapes segmentation.

26.2 Results and models

26.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
MobileNetV3-Small*	2.54	0.06	67.66	87.41	config	model
MobileNetV3-Large*	5.48	0.23	74.04	91.34	config	model

*Models with * are converted from torchvision. 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.*

26.3 Citation

```
@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 27

MViT V2

MViTv2: Improved Multiscale Vision Transformers for Classification and Detection

27.1 Abstract

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.

27.2 Results and models

27.2.1 ImageNet-1k

Model	Pretrain	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
MViTv2-tiny*	From scratch	24.17	4.70	82.33	96.15	config	model
MViTv2-small*	From scratch	34.87	7.00	83.63	96.51	config	model
MViTv2-base*	From scratch	51.47	10.20	84.34	96.86	config	model
MViTv2-large*	From scratch	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.*

27.3 Citation

```
@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 28

PoolFormer

MetaFormer is Actually What You Need for Vision

28.1 Abstract

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.

28.2 Results and models

28.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
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.*

28.3 Citation

```
@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 29

RegNet

Designing Network Design Spaces

29.1 Abstract

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.

29.2 Results and models

29.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
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.*

29.3 Citation

```
@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 30

RepMLP

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

30.1 Abstract

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).

30.2 Results and models

30.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
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.*

30.3 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.

30.3.1 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
```

30.3.2 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()
```

30.4 Citation

```
@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 31

RepVGG

Repvgg: Making vgg-style convnets great again

31.1 Abstract

We present a simple but powerful architecture of convolutional neural network, which has a VGG-like inference-time body composed of nothing but a stack of 3x3 convolution and ReLU, while the training-time model has a multi-branch topology. Such decoupling of the training-time and inference-time architecture is realized by a structural re-parameterization technique so that the model is named RepVGG. On ImageNet, RepVGG reaches over 80% top-1 accuracy, which is the first time for a plain model, to the best of our knowledge. On NVIDIA 1080Ti GPU, RepVGG models run 83% faster than ResNet-50 or 101% faster than ResNet-101 with higher accuracy and show favorable accuracy-speed trade-off compared to the state-of-the-art models like EfficientNet and RegNet.

31.2 Results and models

31.2.1 ImageNet-1k

Model	Epochs	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
RepVGG-A0*	120	9.11（train）| 8.31 (deploy)	1.52 (train) | 1.36 (deploy)	72.41	90.50	config (train) | config (deploy)	model
RepVGG-A1*	120	14.09 (train) | 12.79 (deploy)	2.64 (train) | 2.37 (deploy)	74.47	91.85	config (train) | config (deploy)	model
RepVGG-A2*	120	28.21 (train) | 25.5 (deploy)	5.7 (train) | 5.12 (deploy)	76.48	93.01	config (train) |config (deploy)	model
RepVGG-B0*	120	15.82 (train) | 14.34 (deploy)	3.42 (train) | 3.06 (deploy)	75.14	92.42	config (train) |config (deploy)	model
RepVGG-B1*	120	57.42 (train) | 51.83 (deploy)	13.16 (train) | 11.82 (deploy)	78.37	94.11	config (train) |config (deploy)	model
RepVGG-B1g2*	120	45.78 (train) | 41.36 (deploy)	9.82 (train) | 8.82 (deploy)	77.79	93.88	config (train) |config (deploy)	model
RepVGG-B1g4*	120	39.97 (train) | 36.13 (deploy)	8.15 (train) | 7.32 (deploy)	77.58	93.84	config (train) |config (deploy)	model
RepVGG-B2*	120	89.02 (train) | 80.32 (deploy)	20.46 (train) | 18.39 (deploy)	78.78	94.42	config (train) |config (deploy)	model
RepVGG-B2g4*	200	61.76 (train) | 55.78 (deploy)	12.63 (train) | 11.34 (deploy)	79.38	94.68	config (train) |config (deploy)	model
RepVGG-B3*	200	123.09 (train) | 110.96 (deploy)	29.17 (train) | 26.22 (deploy)	80.52	95.26	config (train) |config (deploy)	model
RepVGG-B3g4*	200	83.83 (train) | 75.63 (deploy)	17.9 (train) | 16.08 (deploy)	80.22	95.10	config (train) |config (deploy)	model
RepVGG-D2se*	200	133.33 (train) | 120.39 (deploy)	36.56 (train) | 32.85 (deploy)	81.81	95.94	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.*

31.3 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.

31.3.1 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
```

31.3.2 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='RepVGG',arch='A0'),
backbone = build_backbone(backbone_cfg)
backbone.switch_to_deploy()
```

or

```
from mmcls.models import build_classifier

cfg = dict(
    type='ImageClassifier',
    backbone=dict(
        type='RepVGG',
        arch='A0'),
    neck=dict(type='GlobalAveragePooling'),
    head=dict(
        type='LinearClsHead',
        num_classes=1000,
```

(下页继续)

(续上页)

```
        in_channels=1280,
        loss=dict(type='CrossEntropyLoss', loss_weight=1.0),
        topk=(1, 5),
    ))

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

31.4 Citation

```
@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 32

Res2Net

Res2Net: A New Multi-scale Backbone Architecture

32.1 Abstract

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.

32.2 Results and models

32.2.1 ImageNet-1k

Model	resolution	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
Res2Net-50-14w-8s*	224x224	25.06	4.22	78.14	93.85	config	model | log
Res2Net-50-26w-8s*	224x224	48.40	8.39	79.20	94.36	config	model | log
Res2Net-101-26w-4s*	224x224	45.21	8.12	79.19	94.44	config	model | log

*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.*

32.3 Citation

```
@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 33

ResNet

Deep Residual Learning for Image Recognition

33.1 Abstract

Deeper neural networks are more difficult to train. We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. We explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. We provide comprehensive empirical evidence showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth. On the ImageNet dataset we evaluate residual nets with a depth of up to 152 layers—8x deeper than VGG nets but still having lower complexity. An ensemble of these residual nets achieves 3.57% error on the ImageNet test set. This result won the 1st place on the ILSVRC 2015 classification task. We also present analysis on CIFAR-10 with 100 and 1000 layers.

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.

33.2 Results and models

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

Model	resolution	Params(M)	Flops(G)	Download
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.

33.2.1 Cifar10

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
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

33.2.2 Cifar100

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
ResNet-50	23.71	1.31	79.90	95.19	config	model | log

33.2.3 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
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.*

33.2.4 CUB-200-2011

Model	Pretrain	resolution	Params(M)	Flops(G)	Top-1 (%)	Config	Download
ResNet-50	ImageNet-21k-mill	448x448	23.92	16.48	88.45	config	model | log

33.2.5 Stanford-Cars

Model	Pretrain	resolution	Params(M)	Flops(G)	Top-1 (%)	Config	Download
ResNet-50	ImageNet-21k-mill	448x448	23.92	16.48	92.82	config	model | log

33.3 Citation

```
@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 34

ResNeXt

Aggregated Residual Transformations for Deep Neural Networks

34.1 Abstract

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.

34.2 Results and models

34.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
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

34.3 Citation

```
@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 35

SE-ResNet

Squeeze-and-Excitation Networks

35.1 Abstract

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%.

35.2 Results and models

35.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
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

35.3 Citation

```
@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 36

ShuffleNet V1

ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices

36.1 Abstract

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.

36.2 Results and models

36.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
ShuffleNetV1 1.0x (group=3)	1.87	0.146	68.13	87.81	config	model | log

36.3 Citation

```
@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 37

ShuffleNet V2

Shufflenet v2: Practical guidelines for efficient cnn architecture design

37.1 Abstract

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.

37.2 Results and models

37.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
ShuffleNetV2 1.0x	2.28	0.149	69.55	88.92	config	model | log

37.3 Citation

```
@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 38

Swin Transformer

Swin Transformer: Hierarchical Vision Transformer using Shifted Windows

38.1 Abstract

This paper presents a new vision Transformer, called Swin Transformer, that capably serves as a general-purpose backbone for computer vision. Challenges in adapting Transformer from language to vision arise from differences between the two domains, such as large variations in the scale of visual entities and the high resolution of pixels in images compared to words in text. To address these differences, we propose a hierarchical Transformer whose representation is computed with **S**hifted **win**dows. The shifted windowing scheme brings greater efficiency by limiting self-attention computation to non-overlapping local windows while also allowing for cross-window connection. This hierarchical architecture has the flexibility to model at various scales and has linear computational complexity with respect to image size. These qualities of Swin Transformer make it compatible with a broad range of vision tasks, including image classification (87.3 top-1 accuracy on ImageNet-1K) and dense prediction tasks such as object detection (58.7 box AP and 51.1 mask AP on COCO test-dev) and semantic segmentation (53.5 mIoU on ADE20K val). Its performance surpasses the previous state-of-the-art by a large margin of +2.7 box AP and +2.6 mask AP on COCO, and +3.2 mIoU on ADE20K, demonstrating the potential of Transformer-based models as vision backbones. The hierarchical design and the shifted window approach also prove beneficial for all-MLP architectures.

38.2 Results and models

38.2.1 ImageNet-21k

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

Model	resolution	Params(M)	Flops(G)	Download
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

38.2.2 ImageNet-1k

Model	Pretrain	resolution	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
Swin-T	From scratch	224x224	28.29	4.36	81.18	95.61	config	model | log
Swin-S	From scratch	224x224	49.61	8.52	83.02	96.29	config	model | log
Swin-B	From scratch	224x224	87.77	15.14	83.36	96.44	config	model | log
Swin-S*	From scratch	224x224	49.61	8.52	83.21	96.25	config	model
Swin-B*	From scratch	224x224	87.77	15.14	83.42	96.44	config	model
Swin-B*	From scratch	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.*

38.2.3 CUB-200-2011

Model	Pretrain	resolution	Params(M)	Flops(G)	Top-1 (%)	Config	Download
Swin-L	ImageNet-21k	384x384	195.51	100.04	91.87	config	model | log

38.3 Citation

```
@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 39

Swin Transformer V2

Swin Transformer V2: Scaling Up Capacity and Resolution

39.1 Abstract

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.

39.2 Results and models

39.2.1 ImageNet-21k

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

Model	resolution	Params(M)	Flops(G)	Download
Swin-B*	192x192	87.92	8.51	model
Swin-L*	192x192	196.74	19.04	model

39.2.2 ImageNet-1k

Model	Pretrain	resolution	window	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
Swin-T*	From scratch	256x256	8x8	28.35	4.35	81.76	95.87	config	model
Swin-T*	From scratch	256x256	16x16	28.35	4.4	82.81	96.23	config	model
Swin-S*	From scratch	256x256	8x8	49.73	8.45	83.74	96.6	config	model
Swin-S*	From scratch	256x256	16x16	49.73	8.57	84.13	96.83	config	model
Swin-B*	From scratch	256x256	8x8	87.92	14.99	84.2	96.86	config	model
Swin-B*	From scratch	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.

39.3 Citation

```
@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},
  title = {Swin Transformer V2: Scaling Up Capacity and Resolution},
  publisher = {arXiv},
  year = {2021},
  copyright = {Creative Commons Attribution 4.0 International}
}
```

CHAPTER 40

Tokens-to-Token ViT

Tokens-to-Token ViT: Training Vision Transformers from Scratch on ImageNet

40.1 Abstract

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.

40.2 Results and models

40.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
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.

40.3 Citation

```
@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 41

TNT

Transformer in Transformer

41.1 Abstract

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.

41.2 Results and models

41.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
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.*

41.3 Citation

```
@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 42

Twins

Twins: Revisiting the Design of Spatial Attention in Vision Transformers

42.1 Abstract

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.

42.2 Results and models

42.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
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*

42.3 Citation

```
@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 43

Visual Attention Network

Visual Attention Network

43.1 Abstract

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.

43.2 Results and models

43.2.1 ImageNet-1k

Model	Pretrain	resolution	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
VAN-B0*	From scratch	224x224	4.11	0.88	75.41	93.02	config	model
VAN-B1*	From scratch	224x224	13.86	2.52	81.01	95.63	config	model
VAN-B2*	From scratch	224x224	26.58	5.03	82.80	96.21	config	model
VAN-B3*	From scratch	224x224	44.77	8.99	83.86	96.73	config	model
VAN-B4*	From scratch	224x224	60.28	12.22	84.13	96.86	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.

43.2.2 Pre-trained Models

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

Model	Pretrain	resolution	Params(M)	Flops(G)	Download
VAN-B4*	ImageNet-21k	224x224	60.28	12.22	model
VAN-B5*	ImageNet-21k	224x224	89.97	17.21	model
VAN-B6*	ImageNet-21k	224x224	283.9	55.28	model

*Models with * are converted from the official repo.

43.3 Citation

```
@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},
```

(下页继续)

(续上页)

```
  year={2022}
}
```

CHAPTER 44

VGG

Very Deep Convolutional Networks for Large-Scale Image Recognition

44.1 Abstract

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.

44.2 Results and models

44.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
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

44.3 Citation

```
@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 45

Vision Transformer

An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

45.1 Abstract

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.

45.2 Results and models

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.

45.2.1 ImageNet-21k

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

Model	resolution	Params(M)	Flops(G)	Download
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.*

45.2.2 ImageNet-1k

Model	Pretrain	resolution	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
ViT-B16*	ImageNet-21k	384x384	86.86	33.03	85.43	97.77	config	model
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
ViT-B16 (IPU)	ImageNet-21k	224x224	86.86	33.03	81.22	95.56	config	model | log

*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.*

45.3 Citation

```
@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 46

Wide-ResNet

Wide Residual Networks

46.1 Abstract

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.

46.2 Results and models

46.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
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.*

46.3 Citation

```
@INPROCEEDINGS{Zagoruyko2016WRN,
    author = {Sergey Zagoruyko and Nikos Komodakis},
    title = {Wide Residual Networks},
    booktitle = {BMVC},
    year = {2016}}
```

CHAPTER 47

Pytorch 转 ONNX （试验性的）

47.1 如何将模型从 PyTorch 转换到 ONNX

47.1.1 准备工作

1. 请参照 安装指南 从源码安装 MMClassification。
2. 安装 onnx 和 onnxruntime。

```
pip install onnx onnxruntime==1.5.1
```

47.1.2 使用方法

```
python tools/deployment/pytorch2onnx.py \
    ${CONFIG_FILE} \
    --checkpoint ${CHECKPOINT_FILE} \
    --output-file ${OUTPUT_FILE} \
    --shape ${IMAGE_SHAPE} \
    --opset-version ${OPSET_VERSION} \
    --dynamic-shape \
    --show \
    --simplify \
    --verify \
```

所有参数的说明:

- `config` : 模型配置文件的路径。
- `--checkpoint` : 模型权重文件的路径。
- `--output-file`: ONNX 模型的输出路径。如果没有指定，默认为当前脚本执行路径下的 `tmp.onnx`。
- `--shape`: 模型输入的高度和宽度。如果没有指定，默认为 `224 224`。
- `--opset-version` : ONNX 的 opset 版本。如果没有指定，默认为 `11`。
- `--dynamic-shape` : 是否以动态输入尺寸导出 ONNX。如果没有指定，默认为 `False`。
- `--show`: 是否打印导出模型的架构。如果没有指定，默认为 `False`。
- `--simplify`: 是否精简导出的 ONNX 模型。如果没有指定，默认为 `False`。
- `--verify`: 是否验证导出模型的正确性。如果没有指定，默认为 `False`。

示例:

```
python tools/deployment/pytorch2onnx.py \
    configs/resnet/resnet18_8xb16_cifar10.py \
    --checkpoint checkpoints/resnet/resnet18_b16x8_cifar10.pth \
    --output-file checkpoints/resnet/resnet18_b16x8_cifar10.onnx \
    --dynamic-shape \
    --show \
    --simplify \
    --verify \
```

47.2 支持导出至 ONNX 的模型列表

下表列出了保证可导出至 ONNX，并在 ONNX Runtime 中运行的模型。

模型	配置文件	批推理	动态输入尺寸	备注
MobileNetV2	configs/mobilenet_v2/mobilenet-v2_8xb32_in1k.py	Y	Y	
ResNet	configs/resnet/resnet18_8xb16_cifar10.py	Y	Y	
ResNeXt	configs/resnext/resnext50-32x4d_8xb32_in1k.py	Y	Y	
SE-ResNet	configs/seresnet/seresnet50_8xb32_in1k.py	Y	Y	
ShuffleNetV1	configs/shufflenet_v1/shufflenet-v1-1x_16xb64_in1k.py	Y	Y	
ShuffleNetV2	configs/shufflenet_v2/shufflenet-v2-1x_16xb64_in1k.py	Y	Y	

注：

- *以上所有模型转换测试基于 Pytorch==1.6.0 进行*

47.3 提示

- 如果你在上述模型的转换中遇到问题，请在 GitHub 中创建一个 issue，我们会尽快处理。未在上表中列出的模型，由于资源限制，我们可能无法提供很多帮助，如果遇到问题，请尝试自行解决。

47.4 常见问题

- 无

CHAPTER 48

ONNX 转 TensorRT（试验性的）

48.1 如何将模型从 ONNX 转换到 TensorRT

48.1.1 准备工作

1. 请参照 安装指南 从源码安装 MMClassification。
2. 使用我们的工具*pytorch2onnx.md* 将 PyTorch 模型转换至 ONNX。

48.1.2 使用方法

```
python tools/deployment/onnx2tensorrt.py \
    ${MODEL} \
    --trt-file ${TRT_FILE} \
    --shape ${IMAGE_SHAPE} \
    --workspace-size {WORKSPACE_SIZE} \
    --show \
    --verify \
```

所有参数的说明:

- `model` : ONNX 模型的路径。
- `--trt-file`: TensorRT 引擎文件的输出路径。如果没有指定，默认为当前脚本执行路径下的 `tmp.trt`。
- `--shape`: 模型输入的高度和宽度。如果没有指定，默认为 `224 224`。
- `--workspace-size` : 构建 TensorRT 引擎所需要的 GPU 空间大小，单位为 GiB。如果没有指定，默认为 `1` GiB。
- `--show`: 是否展示模型的输出。如果没有指定，默认为 `False`。
- `--verify`: 是否使用 ONNXRuntime 和 TensorRT 验证模型转换的正确性。如果没有指定，默认为 `False`。

示例:

```
python tools/deployment/onnx2tensorrt.py \
    checkpoints/resnet/resnet18_b16x8_cifar10.onnx \
    --trt-file checkpoints/resnet/resnet18_b16x8_cifar10.trt \
    --shape 224 224 \
    --show \
    --verify \
```

48.2 支持转换至 TensorRT 的模型列表

下表列出了保证可转换为 TensorRT 的模型。

模型	配置文件	状态
MobileNetV2	`configs/mobilenet_v2/mobilenet-v2_8xb32_in1k.py`	Y
ResNet	`configs/resnet/resnet18_8xb16_cifar10.py`	Y
ResNeXt	`configs/resnext/resnext50-32x4d_8xb32_in1k.py`	Y
ShuffleNetV1	`configs/shufflenet_v1/shufflenet-v1-1x_16xb64_in1k.py`	Y
ShuffleNetV2	`configs/shufflenet_v2/shufflenet-v2-1x_16xb64_in1k.py`	Y

注:

- 以上所有模型转换测试基于 *Pytorch==1.6.0* 和 *TensorRT-7.2.1.6.Ubuntu-16.04.x86_64-gnu.cuda-10.2.cudnn8.0* 进行

48.3 提示

- 如果你在上述模型的转换中遇到问题，请在 GitHub 中创建一个 issue，我们会尽快处理。未在上表中列出的模型，由于资源限制，我们可能无法提供很多帮助，如果遇到问题，请尝试自行解决。

48.4 常见问题

- 无

CHAPTER 49

Pytorch 转 TorchScript （试验性的）

49.1 如何将 PyTorch 模型转换至 TorchScript

49.1.1 使用方法

```
python tools/deployment/pytorch2torchscript.py \
    ${CONFIG_FILE} \
    --checkpoint ${CHECKPOINT_FILE} \
    --output-file ${OUTPUT_FILE} \
    --shape ${IMAGE_SHAPE} \
    --verify \
```

所有参数的说明:

- `config` : 模型配置文件的路径。
- `--checkpoint` : 模型权重文件的路径。
- `--output-file`: TorchScript 模型的输出路径。如果没有指定，默认为当前脚本执行路径下的 `tmp.pt`。

- `--shape`: 模型输入的高度和宽度。如果没有指定，默认为 `224 224`。
- `--verify`: 是否验证导出模型的正确性。如果没有指定，默认为 `False`。

示例：

```
python tools/deployment/pytorch2torchscript.py \
    configs/resnet/resnet18_8xb16_cifar10.py \
    --checkpoint checkpoints/resnet/resnet18_b16x8_cifar10.pth \
    --output-file checkpoints/resnet/resnet18_b16x8_cifar10.pt \
    --verify \
```

注：

- 所有模型基于 *Pytorch==1.8.1* 通过了转换测试

49.2 提示

- 由于 `torch.jit.is_tracing()` 只在 PyTorch 1.6 之后的版本中得到支持，对于 PyTorch 1.3-1.5 的用户，我们建议手动提前返回结果。
- 如果你在本仓库的模型转换中遇到问题，请在 GitHub 中创建一个 issue，我们会尽快处理。

49.3 常见问题

- 无

CHAPTER 50

模型部署至 TorchServe

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

50.1 1. 转换 MMClassification 模型至 TorchServe

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

50.2 2. 构建 mmcls-serve docker 镜像

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

50.3 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。

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

示例：

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

CHAPTER 51

可视化

51.1 数据流水线可视化

```
python tools/visualizations/vis_pipeline.py \
    ${CONFIG_FILE} \
    [--output-dir ${OUTPUT_DIR}] \
    [--phase ${DATASET_PHASE}] \
    [--number ${BUNBER_IMAGES_DISPLAY}] \
    [--skip-type ${SKIP_TRANSFORM_TYPE}] \
    [--mode ${DISPLAY_MODE}] \
    [--show] \
    [--adaptive] \
    [--min-edge-length ${MIN_EDGE_LENGTH}] \
    [--max-edge-length ${MAX_EDGE_LENGTH}] \
    [--bgr2rgb] \
    [--window-size ${WINDOW_SIZE}] \
    [--cfg-options ${CFG_OPTIONS}]
```

所有参数的说明:

- config : 模型配置文件的路径。
- --output-dir: 保存图片文件夹，如果没有指定，默认为 ''，表示不保存。
- --phase: 可视化数据集的阶段，只能为 [train, val, test] 之一，默认为 train。
- --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。
- --skip-type: 预设跳过的数据流水线过程。如果没有指定，默认为 ['ToTensor', 'Normalize', 'ImageToTensor', 'Collect']。
- --mode: 可视化的模式，只能为 [original, transformed, concat, pipeline] 之一，如果没有指定，默认为 concat。
- --show: 将可视化图片以弹窗形式展示。
- --adaptive: 自动调节可视化图片的大小。
- --min-edge-length: 最短边长度，当使用了 --adaptive 时有效。当图片任意边小于 ${MIN_EDGE_LENGTH} 时，会保持长宽比不变放大图片，短边对齐至 ${MIN_EDGE_LENGTH}，默认为 200。
- --max-edge-length: 最长边长度，当使用了 --adaptive 时有效。当图片任意边大于 ${MAX_EDGE_LENGTH} 时，会保持长宽比不变缩小图片，短边对齐至 ${MAX_EDGE_LENGTH}，默认为 1000。
- --bgr2rgb: 将图片的颜色通道翻转。
- --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，参考教程 1：如何编写配置文件。

备注:

1. 如果不指定 --mode，默认设置为 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 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 pipeline，则获得数据流水线所有中间过程图片。
2. 当指定了 --adaptive 选项时，会自动的调整尺寸过大和过小的图片，你可以通过设定 --min-edge-length 与 --max-edge-length 来指定自动调整的图片尺寸。

示例:

1. **'original'** 模式，可视化 CIFAR100 验证集中的 100 张原始图片，显示并保存在 ./tmp 文件夹下:

```
python ./tools/visualizations/vis_pipeline.py configs/resnet/resnet50_8xb16_cifar100.
↪py --phase val --output-dir tmp --mode original --number 100 --show --adaptive --
↪bgr2rgb
```

2. **‘transformed’**模式，可视化 ImageNet 训练集的所有经过预处理的图片，并以弹窗形式显示：

```
python ./tools/visualizations/vis_pipeline.py ./configs/resnet/resnet50_8xb32_in1k.py␣
↪--show --mode transformed
```

3. **‘concat’**模式，可视化 ImageNet 训练集的 10 张原始图片与预处理后图片对比图，保存在 ./tmp 文件夹下：

```
python ./tools/visualizations/vis_pipeline.py configs/swin_transformer/swin_base_224_
↪b16x64_300e_imagenet.py --phase train --output-dir tmp --number 10 --adaptive
```

4. **‘pipeline’**模式，可视化 ImageNet 训练集经过数据流水线的过程图像：

```
python ./tools/visualizations/vis_pipeline.py configs/swin_transformer/swin_base_224_
↪b16x64_300e_imagenet.py --phase train --adaptive --mode pipeline --show
```

51.2 学习率策略可视化

```
python tools/visualizations/vis_lr.py \
    ${CONFIG_FILE} \
    [--dataset-size ${Dataset_Size}] \
    [--ngpus ${NUM_GPUs}] \
    [--save-path ${SAVE_PATH}] \
    [--title ${TITLE}] \
    [--style ${STYLE}] \
    [--window-size ${WINDOW_SIZE}] \
    [--cfg-options ${CFG_OPTIONS}] \
```

所有参数的说明：

- config：模型配置文件的路径。
- --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。
- --ngpus：使用 GPU 的数量。
- --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，参考教程 1：如何编写配置文件。

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

示例：

```
python tools/visualizations/vis_lr.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_lr.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
```

51.3 类别激活图可视化

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} \
    [--target-layers ${TARGET-LAYERS}] \
    [--preview-model] \
    [--method ${METHOD}] \
    [--target-category ${TARGET-CATEGORY}] \
    [--save-path ${SAVE_PATH}] \
```

(下页继续)

(续上页)

```
    [--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，参考教程 1：如何编写配置文件。

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;

示例（CNN）:

`--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_8xb32_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

示例（Transformer）：

`--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

51.4 常见问题

- 无

CHAPTER 52

分析

52.1 日志分析

52.1.1 绘制曲线图

指定一个训练日志文件，可通过 `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}]  \
```

(下页继续)

(续上页)

```
    [--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`： 图例名（可同时传入多个，使用空格分开，数目与 `${JSON_LOGS} * ${KEYS}` 数目一致）。默认使用 `"${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： `--style` 选项依赖于第三方库 `seaborn`，需要设置绘图风格请现安装该库。

例如:

- 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_top-1 accuracy_top-5  --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 run1 run2
```

备注： 本工具会自动根据关键字段选择从日志的训练部分还是验证部分读取，因此如果你添加了自定义的验

证指标，请把相对应的关键字段加入到本工具的 TEST_METRICS 变量中。

52.1.2 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轮迭代的记录（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/some_exp/
↪20200422_153324.log.json
```

预计输出结果如下所示：

```
-----Analyze train time of work_dirs/some_exp/20200422_153324.log.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
```

52.2 结果分析

利用 tools/test.py 的 --out 参数，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。

52.2.1 评估结果

tools/analysis_tools/eval_metric.py 可以用来再次计算评估结果。

```
python tools/analysis_tools/eval_metric.py \
      ${CONFIG} \
      ${RESULT} \
      [--metrics ${METRICS}]  \
      [--cfg-options ${CFG_OPTIONS}] \
      [--metric-options ${METRIC_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。
- metrics： 评估的衡量指标，可接受的值取决于数据集类。
- --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。
- --metric-options: 如果指定了，这些选项将被传递给数据集 evaluate 函数的 metric_options 参数。

备注： 在 tools/test.py 中，我们支持使用 --out-items 选项来选择保存哪些结果。为了使用本工具，请确保结果文件中包含“class_scores”。

示例:

```
python tools/analysis_tools/eval_metric.py configs/t2t_vit/t2t-vit-t-14_8xb64_in1k.py␣
↪./result.pkl --metrics accuracy --metric-options "topk=(1,5)"
```

52.2.2 查看典型结果

`tools/analysis_tools/analyze_results.py` 可以保存预测成功/失败，同时得分最高的 k 个图像。

```
python tools/analysis_tools/analyze_results.py \
      ${CONFIG} \
      ${RESULT} \
      [--out-dir ${OUT_DIR}] \
      [--topk ${TOPK}] \
      [--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`。
- `--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。

备注： 在 `tools/test.py` 中，我们支持使用 `--out-items` 选项来选择保存哪些结果。为了使用本工具，请确保结果文件中包含“pred_score”、”pred_label”和“pred_class”。

示例:

```
python tools/analysis_tools/analyze_results.py \
       configs/resnet/resnet50_xxxx.py \
       result.pkl \
       --out-dir results \
       --topk 50
```

52.3 模型复杂度分析

52.3.1 计算 FLOPs 和参数量（试验性的）

我们根据 flops-counter.pytorch 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`。

用户将获得如下结果：

```
==============================
Input shape: (3, 224, 224)
Flops: 4.12 GFLOPs
Params: 25.56 M
==============================
```

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 GN 和自定义运算。详细信息请参考 `mmcv.cnn.get_model_complexity_info()`

52.4 常见问题

- 无

CHAPTER 53

其他工具

53.1 打印完整配置

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

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

所有参数说明:

- config： 配置文件的路径。
- --cfg-options: 额外的配置选项，会被合入配置文件，参考教程 1：如何编写配置文件。

示例:

```
python tools/misc/print_config.py configs/t2t_vit/t2t-vit-t-14_8xb64_in1k.py
```

53.2 检查数据集

`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。

```
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。

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
```

53.3 常见问题

- 无

CHAPTER 54

参与贡献 OpenMMLab

欢迎任何类型的贡献，包括但不限于

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

54.1 工作流程

1. fork 并 pull 最新的 OpenMMLab 仓库 (MMClassification)
2. 签出到一个新分支（不要使用 master 分支提交 PR）
3. 进行修改并提交至 fork 出的自己的远程仓库
4. 在我们的仓库中创建一个 PR

备注： 如果你计划添加一些新的功能，并引入大量改动，请尽量首先创建一个 issue 来进行讨论。

54.2 代码风格

54.2.1 Python

我们采用 PEP8 作为统一的代码风格。

我们使用下列工具来进行代码风格检查与格式化:

- flake8: Python 官方发布的代码规范检查工具，是多个检查工具的封装
- isort: 自动调整模块导入顺序的工具
- yapf: 一个 Python 文件的格式化工具。
- codespell: 检查单词拼写是否有误
- mdformat: 检查 markdown 文件的工具
- docformatter: 一个 docstring 格式化工具。

yapf 和 isort 的格式设置位于 setup.cfg

我们使用 pre-commit hook 来保证每次提交时自动进行代码检查和格式化，启用的功能包括 `flake8`, `yapf`, `isort`, `trailing whitespaces`, `markdown files`, 修复 `end-of-files`, `double-quoted-strings`, `python-encoding-pragma`, `mixed-line-ending`, 对 `requirments.txt` 的排序等。pre-commit hook 的配置文件位于 .pre-commit-config

在你克隆仓库后，你需要按照如下步骤安装并初始化 pre-commit hook。

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

在仓库文件夹中执行

```
pre-commit install
```

在此之后，每次提交，代码规范检查和格式化工具都将被强制执行。

重要: 在创建 PR 之前，请确保你的代码完成了代码规范检查，并经过了 yapf 的格式化。

54.2.2 C++ 和 CUDA

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

CHAPTER 55

mmcls.apis

These are some high-level APIs for classification tasks.

mmcls.apis

- *Train*
- *Test*
- *Inference*

55.1 Train

`init_random_seed`	Initialize random seed.
`set_random_seed`	Set random seed.
`train_model`	Train a model.

55.1.1 mmcls.apis.init_random_seed

mmcls.apis.**init_random_seed**(*seed=None*, *device=None*)

Initialize random seed.

If the seed is not set, the seed will be automatically randomized, and then broadcast to all processes to prevent some potential bugs.

参数

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

返回 Seed to be used.

返回类型 int

55.1.2 mmcls.apis.set_random_seed

mmcls.apis.**set_random_seed**(*seed*, *deterministic=False*)

Set random seed.

参数

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

55.1.3 mmcls.apis.train_model

mmcls.apis.**train_model**(*model*, *dataset*, *cfg*, *distributed=False*, *validate=False*, *timestamp=None*, *device=None*, *meta=None*)

Train a model.

This method will build dataloaders, wrap the model and build a runner according to the provided config.

参数

- **model** (torch.nn.Module) –The model to be run.
- **dataset** (*mmcls.datasets.BaseDataset* | List[BaseDataset]) –The dataset used to train the model. It can be a single dataset, or a list of dataset with the same length as workflow.
- **cfg** (mmcv.utils.Config) –The configs of the experiment.

- **distributed** (*bool*) –Whether to train the model in a distributed environment. Defaults to False.
- **validate** (*bool*) –Whether to do validation with `mmcv.runner.EvalHook`. Defaults to False.
- **timestamp** (*str, optional*) –The timestamp string to auto generate the name of log files. Defaults to None.
- **device** (*str, optional*) –TODO
- **meta** (*dict, optional*) –A dict records some import information such as environment info and seed, which will be logged in logger hook. Defaults to None.

55.2 Test

single_gpu_test	Test model with local single gpu.
multi_gpu_test	Test model with multiple gpus.

55.2.1 mmcls.apis.single_gpu_test

mmcls.apis.**single_gpu_test**(*model*, *data_loader*, *show=False*, *out_dir=None*, ***show_kwargs*)

Test model with local single gpu.

This method tests model with a single gpu and supports showing results.

参数

- **model** (`torch.nn.Module`) –Model to be tested.
- **data_loader** (`torch.utils.data.DataLoader`) –Pytorch data loader.
- **show** (*bool*) –Whether to show the test results. Defaults to False.
- **out_dir** (*str*) –The output directory of result plots of all samples. Defaults to None, which means not to write output files.
- ****show_kwargs** –Any other keyword arguments for showing results.

返回 The prediction results.

返回类型 list

55.2.2 mmcls.apis.multi_gpu_test

mmcls.apis.**multi_gpu_test**(*model*, *data_loader*, *tmpdir=None*, *gpu_collect=False*)

Test model with multiple gpus.

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

参数

- **model** (*nn.Module*) –Model to be tested.
- **data_loader** (*nn.Dataloader*) –Pytorch data loader.
- **tmpdir** (*str*) –Path of directory to save the temporary results from different gpus under cpu mode.
- **gpu_collect** (*bool*) –Option to use either gpu or cpu to collect results.

返回 The prediction results.

返回类型 list

55.3 Inference

init_model	Initialize a classifier from config file.
inference_model	Inference image(s) with the classifier.
show_result_pyplot	Visualize the classification results on the image.

55.3.1 mmcls.apis.init_model

mmcls.apis.**init_model**(*config*, *checkpoint=None*, *device='cuda:0'*, *options=None*)

Initialize a classifier from config file.

参数

- **config** (str or mmcv.Config) –Config file path or the config object.
- **checkpoint** (*str, optional*) –Checkpoint path. If left as None, the model will not load any weights.
- **options** (*dict*) –Options to override some settings in the used config.

返回 The constructed classifier.

返回类型 nn.Module

55.3.2 mmcls.apis.inference_model

mmcls.apis.**inference_model**(*model*, *img*)

Inference image(s) with the classifier.

参数

- **model** (*nn.Module*) –The loaded classifier.
- **img** (*str/ndarray*) –The image filename or loaded image.

返回

The classification results that contains *class_name*, *pred_label* and *pred_score*.

返回类型 result (dict)

55.3.3 mmcls.apis.show_result_pyplot

mmcls.apis.**show_result_pyplot**(*model*, *img*, *result*, *fig_size=(15, 10)*, *title='result'*, *wait_time=0*)

Visualize the classification results on the image.

参数

- **model** (*nn.Module*) –The loaded classifier.
- **img** (*str or np.ndarray*) –Image filename or loaded image.
- **result** (*list*) –The classification result.
- **fig_size** (*tuple*) –Figure size of the pyplot figure. Defaults to (15, 10).
- **title** (*str*) –Title of the pyplot figure. Defaults to 'result'.
- **wait_time** (*int*) –How many seconds to display the image. Defaults to 0.

CHAPTER 56

mmcls.core

This package includes some runtime components. These components are useful in classification tasks but not supported by MMCV yet.

备注： Some components may be moved to MMCV in the future.

mmcls.core

56.1 Evaluation

Evaluation metrics calculation functions

precision	Calculate precision according to the prediction and target.
recall	Calculate recall according to the prediction and target.
f1_score	Calculate F1 score according to the prediction and target.
precision_recall_f1	Calculate precision, recall and f1 score according to the prediction and target.
average_precision	Calculate the average precision for a single class.
mAP	Calculate the mean average precision with respect of classes.
support	Calculate the total number of occurrences of each label according to the prediction and target.
average_performance	Calculate CP, CR, CF1, OP, OR, OF1, where C stands for per-class average, O stands for overall average, P stands for precision, R stands for recall and F1 stands for F1-score.
calculate_confusion_matrix	Calculate confusion matrix according to the prediction and target.

56.1.1 mmcls.core.precision

mmcls.core.**precision**(*pred*, *target*, *average_mode='macro'*, *thrs=0.0*)

Calculate precision according to the prediction and target.

参数

- **pred** (*torch.Tensor | np.array*) –The model prediction with shape (N, C).
- **target** (*torch.Tensor | np.array*) –The target of each prediction with shape (N, 1) or (N,).
- **average_mode** (*str*) –The type of averaging performed on the result. Options are 'macro' and 'none'. If 'none', the scores for each class are returned. If 'macro', calculate metrics for each class, and find their unweighted mean. Defaults to 'macro'.
- **thrs** (*Number | tuple[Number], optional*) –Predictions with scores under the thresholds are considered negative. Default to 0.

返回 Precision.

返回类型

float | np.array | list[float | np.array]

Args	thrs is number	thrs is tuple
average_mode = "macro"	float	list[float]
average_mode = "none"	np.array	list[np.array]

56.1.2 mmcls.core.recall

mmcls.core.**recall**(*pred*, *target*, *average_mode='macro'*, *thrs=0.0*)

Calculate recall according to the prediction and target.

参数

- **pred** (*torch.Tensor | np.array*) –The model prediction with shape (N, C).
- **target** (*torch.Tensor | np.array*) –The target of each prediction with shape (N, 1) or (N,).
- **average_mode** (*str*) –The type of averaging performed on the result. Options are 'macro' and 'none'. If 'none', the scores for each class are returned. If 'macro', calculate metrics for each class, and find their unweighted mean. Defaults to 'macro'.
- **thrs** (*Number | tuple[Number], optional*) –Predictions with scores under the thresholds are considered negative. Default to 0.

返回 Recall.

返回类型

float | np.array | list[float | np.array]

Args	thrs is number	thrs is tuple
average_mode = "macro"	float	list[float]
average_mode = "none"	np.array	list[np.array]

56.1.3 mmcls.core.f1_score

mmcls.core.**f1_score**(*pred*, *target*, *average_mode='macro'*, *thrs=0.0*)

Calculate F1 score according to the prediction and target.

参数

- **pred** (*torch.Tensor | np.array*) –The model prediction with shape (N, C).
- **target** (*torch.Tensor | np.array*) –The target of each prediction with shape (N, 1) or (N,).
- **average_mode** (*str*) –The type of averaging performed on the result. Options are 'macro' and 'none'. If 'none', the scores for each class are returned. If 'macro', calculate metrics for each class, and find their unweighted mean. Defaults to 'macro'.
- **thrs** (*Number | tuple[Number], optional*) –Predictions with scores under the thresholds are considered negative. Default to 0.

返回 F1 score.

返回类型

float | np.array | list[float | np.array]

Args	`thrs` is number	`thrs` is tuple
`average_mode` = “macro”	float	list[float]
`average_mode` = “none”	np.array	list[np.array]

56.1.4 mmcls.core.precision_recall_f1

mmcls.core.**precision_recall_f1**(*pred*, *target*, *average_mode='macro'*, *thrs=0.0*)

Calculate precision, recall and f1 score according to the prediction and target.

参数

- **pred** (*torch.Tensor | np.array*) –The model prediction with shape (N, C).
- **target** (*torch.Tensor | np.array*) –The target of each prediction with shape (N, 1) or (N,).
- **average_mode** (*str*) –The type of averaging performed on the result. Options are ‘macro’ and ‘none’. If ‘none’, the scores for each class are returned. If ‘macro’, calculate metrics for each class, and find their unweighted mean. Defaults to ‘macro’.
- **thrs** (*Number | tuple[Number], optional*) –Predictions with scores under the thresholds are considered negative. Default to 0.

返回

tuple containing precision, recall, f1 score.

The type of precision, recall, f1 score is one of the following:

Args	`thrs` is number	`thrs` is tuple
`average_mode` = “macro”	float	list[float]
`average_mode` = “none”	np.array	list[np.array]

返回类型 tuple

56.1.5 mmcls.core.average_precision

mmcls.core.**average_precision**(*pred*, *target*)

Calculate the average precision for a single class.

AP summarizes a precision-recall curve as the weighted mean of maximum precisions obtained for any r' >r, where r is the recall:

$$\text{AP} = \sum_n (R_n - R_{n-1}) P_n$$

Note that no approximation is involved since the curve is piecewise constant.

参数

- **pred** (*np.ndarray*) –The model prediction with shape (N,).
- **target** (*np.ndarray*) –The target of each prediction with shape (N,).

返回 a single float as average precision value.

返回类型 float

56.1.6 mmcls.core.mAP

mmcls.core.**mAP**(*pred*, *target*)

Calculate the mean average precision with respect of classes.

参数

- **pred** (*torch.Tensor | np.ndarray*) –The model prediction with shape (N, C), where C is the number of classes.
- **target** (*torch.Tensor | np.ndarray*) –The target of each prediction with shape (N, C), where C is the number of classes. 1 stands for positive examples, 0 stands for negative examples and -1 stands for difficult examples.

返回 A single float as mAP value.

返回类型 float

56.1.7 mmcls.core.support

mmcls.core.**support**(*pred*, *target*, *average_mode='macro'*)

Calculate the total number of occurrences of each label according to the prediction and target.

参数

- **pred** (*torch.Tensor | np.array*) –The model prediction with shape (N, C).

- **target** (*torch.Tensor | np.array*) –The target of each prediction with shape (N, 1) or (N,).
- **average_mode** (*str*) –The type of averaging performed on the result. Options are ‘macro’ and ‘none’. If ‘none’, the scores for each class are returned. If ‘macro’, calculate metrics for each class, and find their unweighted sum. Defaults to ‘macro’.

返回

Support.

- If the `average_mode` is set to macro, the function returns a single float.
- If the `average_mode` is set to none, the function returns a np.array with shape C.

返回类型 float | np.array

56.1.8 mmcls.core.average_performance

`mmcls.core.`**`average_performance`**(*pred*, *target*, *thr=None*, *k=None*)

Calculate CP, CR, CF1, OP, OR, OF1, where C stands for per-class average, O stands for overall average, P stands for precision, R stands for recall and F1 stands for F1-score.

参数

- **pred** (*torch.Tensor | np.ndarray*) –The model prediction with shape (N, C), where C is the number of classes.
- **target** (*torch.Tensor | np.ndarray*) –The target of each prediction with shape (N, C), where C is the number of classes. 1 stands for positive examples, 0 stands for negative examples and -1 stands for difficult examples.
- **thr** (*float*) –The confidence threshold. Defaults to None.
- **k** (*int*) –Top-k performance. Note that if thr and k are both given, k will be ignored. Defaults to None.

返回 (CP, CR, CF1, OP, OR, OF1)

返回类型 tuple

56.1.9 mmcls.core.calculate_confusion_matrix

`mmcls.core.`**`calculate_confusion_matrix`**(*pred*, *target*)

Calculate confusion matrix according to the prediction and target.

参数

- **pred** (*torch.Tensor | np.array*) –The model prediction with shape (N, C).

- **target** (*torch.Tensor | np.array*) –The target of each prediction with shape (N, 1) or (N,).

返回

Confusion matrix The shape is (C, C), where C is the number of classes.

返回类型 torch.Tensor

56.2 Hook

ClassNumCheckHook	

PreciseBNHook	Precise BN hook.
CosineAnnealingCooldownLrUpdaterHook	Cosine annealing learning rate scheduler with cooldown.
MMClsWandbHook	Enhanced Wandb logger hook for classification.

56.2.1 mmcls.core.ClassNumCheckHook

class mmcls.core.**ClassNumCheckHook**

56.2.2 mmcls.core.PreciseBNHook

class mmcls.core.**PreciseBNHook**(*num_samples: int = 8192*, *interval: int = 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. Defaults to 1.

56.2.3 mmcls.core.CosineAnnealingCooldownLrUpdaterHook

class mmcls.core.**CosineAnnealingCooldownLrUpdaterHook**(*min_lr=None*, *min_lr_ratio=None*, *cool_down_ratio=0.1*, *cool_down_time=10*, ***kwargs*)

Cosine annealing learning rate scheduler with cooldown.

参数

- **min_lr** (*float, optional*) –The minimum learning rate after annealing. Defaults to None.
- **min_lr_ratio** (*float, optional*) –The minimum learning ratio after nnealing. Defaults to None.
- **cool_down_ratio** (*float*) –The cooldown ratio. Defaults to 0.1.
- **cool_down_time** (*int*) –The cooldown time. Defaults to 10.
- **by_epoch** (*bool*) –If True, the learning rate changes epoch by epoch. If False, the learning rate changes iter by iter. Defaults to True.
- **warmup** (*string, optional*) –Type of warmup used. It can be None (use no warmup), 'constant', 'linear' or 'exp'. Defaults to None.
- **warmup_iters** (*int*) –The number of iterations or epochs that warmup lasts. Defaults to 0.
- **warmup_ratio** (*float*) –LR used at the beginning of warmup equals to `warmup_ratio * initial_lr`. Defaults to 0.1.
- **warmup_by_epoch** (*bool*) –If True, the `warmup_iters` means the number of epochs that warmup lasts, otherwise means the number of iteration that warmup lasts. Defaults to False.

备注: You need to set one and only one of `min_lr` and `min_lr_ratio`.

56.2.4 mmcls.core.MMClsWandbHook

class mmcls.core.**MMClsWandbHook**(*init_kwargs=None*, *interval=10*, *log_checkpoint=False*, *log_checkpoint_metadata=False*, *num_eval_images=100*, ***kwargs*)

Enhanced Wandb logger hook for classification.

Comparing with the **:cls:'mmcv.runner.WandbLoggerHook'**, this hook can not only automatically log all information in `log_buffer` but also log the following extra information.

- **Checkpoints**: If `log_checkpoint` is True, the checkpoint saved at every checkpoint interval will be saved as W&B Artifacts. This depends on the : class:*mmcv.runner.CheckpointHook* whose priority is higher than this hook. Please refer to https://docs.wandb.ai/guides/artifacts/model-versioning to learn more about model versioning with W&B Artifacts.
- **Checkpoint Metadata**: If `log_checkpoint_metadata` is True, every checkpoint artifact will have a metadata associated with it. The metadata contains the evaluation metrics computed on validation data with that checkpoint along with the current epoch/iter. It depends on `EvalHook` whose priority is higher than this hook.
- **Evaluation**: At every interval, this hook logs the model prediction as interactive W&B Tables. The number of samples logged is given by `num_eval_images`. Currently, this hook logs the predicted labels along with the ground truth at every evaluation interval. This depends on the `EvalHook` whose priority is higher than this hook. Also note that the data is just logged once and subsequent evaluation tables uses reference to the logged data to save memory usage. Please refer to https://docs.wandb.ai/guides/data-vis to learn more about W&B Tables.

Here is a config example:

```
checkpoint_config = dict(interval=10)

# To log checkpoint metadata, the interval of checkpoint saving should
# be divisible by the interval of evaluation.
evaluation = dict(interval=5)

log_config = dict(
    ...
    hooks=[
        ...
        dict(type='MMClsWandbHook',
             init_kwargs={
                 'entity': "YOUR_ENTITY",
                 'project': "YOUR_PROJECT_NAME"
             },
             log_checkpoint=True,
             log_checkpoint_metadata=True,
             num_eval_images=100)
    ])
```

参数

- **init_kwargs** (*dict*) –A dict passed to wandb.init to initialize a W&B run. Please refer to https://docs.wandb.ai/ref/python/init for possible key-value pairs.
- **interval** (*int*) –Logging interval (every k iterations). Defaults to 10.
- **log_checkpoint** (*bool*) –Save the checkpoint at every checkpoint interval as W&B Ar-

tifacts. Use this for model versioning where each version is a checkpoint. Defaults to False.

- **log_checkpoint_metadata** (*bool*) –Log the evaluation metrics computed on the validation data with the checkpoint, along with current epoch as a metadata to that checkpoint. Defaults to True.
- **num_eval_images** (*int*) –The number of validation images to be logged. If zero, the evaluation won't be logged. Defaults to 100.

56.3 Optimizers

Lamb	A pure pytorch variant of FuseLAMB (NvLamb variant) optimizer.

56.3.1 mmcls.core.Lamb

class mmcls.core.**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)

CHAPTER 57

mmcls.models

The `models` package contains several sub-packages for addressing the different components of a model.

- *Classifier*: The top-level module which defines the whole process of a classification model.
- *Backbones*: Usually a feature extraction network, e.g., ResNet, MobileNet.
- *Necks*: The component between backbones and heads, e.g., GlobalAveragePooling.
- *Heads*: The component for specific tasks. In MMClassification, we provides heads for classification.
- *Losses*: Loss functions.

build_classifier	

build_backbone	Build backbone.
build_neck	Build neck.
build_head	Build head.
build_loss	Build loss.

57.1 mmcls.models.build_classifier

mmcls.models.**build_classifier**(*cfg*)

57.2 mmcls.models.build_backbone

mmcls.models.**build_backbone**(*cfg*)

Build backbone.

57.3 mmcls.models.build_neck

mmcls.models.**build_neck**(*cfg*)

Build neck.

57.4 mmcls.models.build_head

mmcls.models.**build_head**(*cfg*)

Build head.

57.5 mmcls.models.build_loss

mmcls.models.**build_loss**(*cfg*)

Build loss.

57.6 Classifier

BaseClassifier	Base class for classifiers.
ImageClassifier	

57.6.1 mmcls.models.BaseClassifier

class mmcls.models.**BaseClassifier**(*init_cfg=None*)

Base class for classifiers.

57.6.2 mmcls.models.ImageClassifier

class mmcls.models.**ImageClassifier**(*backbone*, *neck=None*, *head=None*, *pretrained=None*, *train_cfg=None*, *init_cfg=None*)

57.7 Backbones

AlexNet	AlexNet backbone.
CSPDarkNet	CSP-Darknet backbone used in YOLOv4.
CSPNet	The abstract CSP Network class.
CSPResNeXt	CSP-ResNeXt backbone.
CSPResNet	CSP-ResNet backbone.
Conformer	Conformer backbone.
ConvMixer	ConvMixer.
ConvNeXt	ConvNeXt.
DenseNet	DenseNet.
DistilledVisionTransformer	Distilled Vision Transformer.
EfficientNet	EfficientNet backbone.
HRNet	HRNet backbone.
LeNet5	LeNet5 backbone.
MlpMixer	Mlp-Mixer backbone.
MobileNetV2	MobileNetV2 backbone.
MobileNetV3	MobileNetV3 backbone.
PCPVT	The backbone of Twins-PCPVT.
PoolFormer	PoolFormer.
RegNet	RegNet backbone.
RepMLPNet	RepMLPNet backbone.
RepVGG	RepVGG backbone.
Res2Net	Res2Net backbone.
ResNeSt	ResNeSt backbone.
ResNeXt	ResNeXt backbone.
ResNet	ResNet backbone.

下页继续

表 1 – 续上页

ResNetV1c	ResNetV1c backbone.
ResNetV1d	ResNetV1d backbone.
ResNet_CIFAR	ResNet backbone for CIFAR.
SEResNeXt	SEResNeXt backbone.
SEResNet	SEResNet backbone.
SVT	The backbone of Twins-SVT.
ShuffleNetV1	ShuffleNetV1 backbone.
ShuffleNetV2	ShuffleNetV2 backbone.
SwinTransformer	Swin Transformer.
T2T_ViT	Tokens-to-Token Vision Transformer (T2T-ViT)
TIMMBackbone	Wrapper to use backbones from timm library.
TNT	Transformer in Transformer.
VAN	Visual Attention Network.
VGG	VGG backbone.
VisionTransformer	Vision Transformer.
EfficientFormer	EfficientFormer.
HorNet	A PyTorch impl of : *HorNet: Efficient High-Order Spatial Interactions with Recursive Gated Convolutions*

57.7.1 mmcls.models.AlexNet

class mmcls.models.**AlexNet**(*num_classes=-1*)

AlexNet backbone.

The input for AlexNet is a 224x224 RGB image.

参数 **num_classes** (*int*) –number of classes for classification. The default value is -1, which uses the backbone as a feature extractor without the top classifier.

57.7.2 mmcls.models.CSPDarkNet

class mmcls.models.**CSPDarkNet**(*depth*, *in_channels=3*, *out_indices=(4,)*, *frozen_stages=-1*, *conv_cfg=None*, *norm_cfg={'eps': 1e-05, 'type': 'BN'}*, *act_cfg={'inplace': True, 'type': 'LeakyReLU'}*, *norm_eval=False*, *init_cfg={'a': 2.23606797749979, 'distribution': 'uniform', 'layer': 'Conv2d', 'mode': 'fan_in', 'nonlinearity': 'leaky_relu', 'type': 'Kaiming'}*)

CSP-Darknet backbone used in YOLOv4.

参数

- **depth** (*int*) –Depth of CSP-Darknet. Default: 53.

- **in_channels** (*int*) –Number of input image channels. Default: 3.
- **out_indices** (*Sequence[int]*) –Output from which stages. Default: (3,).
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Default: -1.
- **conv_cfg** (*dict*) –Config dict for convolution layer. Default: None.
- **norm_cfg** (*dict*) –Dictionary to construct and config norm layer. Default: dict(type='BN', requires_grad=True).
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type='LeakyReLU', negative_slope=0.1).
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None.

示例

```
>>> from mmcls.models import CSPDarkNet
>>> import torch
>>> model = CSPDarkNet(depth=53, out_indices=(0, 1, 2, 3, 4))
>>> model.eval()
>>> inputs = torch.rand(1, 3, 416, 416)
>>> level_outputs = model(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
...
(1, 64, 208, 208)
(1, 128, 104, 104)
(1, 256, 52, 52)
(1, 512, 26, 26)
(1, 1024, 13, 13)
```

57.7.3 mmcls.models.CSPNet

class mmcls.models.**CSPNet**(*arch*, *stem_fn*, *in_channels=3*, *out_indices=-1*, *frozen_stages=-1*, *drop_path_rate=0.0*, *conv_cfg=None*, *norm_cfg={'eps': 1e-05, 'type': 'BN'}*, *act_cfg={'inplace': True, 'type': 'LeakyReLU'}*, *norm_eval=False*, *init_cfg={'layer': 'Conv2d', 'type': 'Kaiming'}*)

The abstract CSP Network class.

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

This class is an abstract class because the Cross Stage Partial Network (CSPNet) is a kind of universal network structure, and you network block to implement networks like CSPResNet, CSPResNeXt and CSPDarkNet.

参数

- **arch** (*dict*) –The architecture of the CSPNet. It should have the following keys:
  - block_fn (Callable): A function or class to return a block module, and it should accept at least `in_channels`, `out_channels`, `expansion`, `drop_path_rate`, `norm_cfg` and `act_cfg`.
  - in_channels (Tuple[int]): The number of input channels of each stage.
  - out_channels (Tuple[int]): The number of output channels of each stage.
  - num_blocks (Tuple[int]): The number of blocks in each stage.
  - expansion_ratio (float | Tuple[float]): The expansion ratio in the expand convolution of each stage. Defaults to 0.5.
  - bottle_ratio (float | Tuple[float]): The expansion ratio of blocks in each stage. Defaults to 2.
  - has_downsampler (bool | Tuple[bool]): Whether to add a downsample convolution in each stage. Defaults to True
  - down_growth (bool | Tuple[bool]): Whether to expand the channels in the downsampler layer of each stage. Defaults to False.
  - block_args (dict | Tuple[dict], optional): The extra arguments to the blocks in each stage. Defaults to None.
- **stem_fn** (*Callable*) –A function or class to return a stem module. And it should accept `in_channels`.
- **in_channels** (*int*) –Number of input image channels. Defaults to 3.
- **out_indices** (*int | Sequence[int]*) –Output from which stages. Defaults to -1, which means the last stage.
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Defaults to -1.
- **conv_cfg** (*dict, optional*) –The config dict for conv layers in blocks. Defaults to None, which means use Conv2d.
- **norm_cfg** (*dict*) –The config dict for norm layers. Defaults to `dict(type='BN', eps=1e-5)`.
- **act_cfg** (*dict*) –The config dict for activation functions. Defaults to `dict(type='LeakyReLU', inplace=True)`.

- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Defaults to False.
- **init_cfg** (*dict, optional*) –The initialization settings. Defaults to `dict(type='Kaiming', layer='Conv2d'))`.

示例

```
>>> from functools import partial
>>> import torch
>>> import torch.nn as nn
>>> from mmcls.models import CSPNet
>>> from mmcls.models.backbones.resnet import Bottleneck
>>>
>>> # A simple example to build CSPNet.
>>> arch = dict(
...     block_fn=Bottleneck,
...     in_channels=[32, 64],
...     out_channels=[64, 128],
...     num_blocks=[3, 4]
... )
>>> stem_fn = partial(nn.Conv2d, out_channels=32, kernel_size=3)
>>> model = CSPNet(arch=arch, stem_fn=stem_fn, out_indices=(0, 1))
>>> inputs = torch.rand(1, 3, 224, 224)
>>> outs = model(inputs)
>>> for out in outs:
...     print(out.shape)
...
(1, 64, 111, 111)
(1, 128, 56, 56)
```

57.7.4 mmcls.models.CSPResNeXt

class mmcls.models.**CSPResNeXt**(**args*, ***kwargs*)

CSP-ResNeXt backbone.

参数

- **depth** (*int*) –Depth of CSP-ResNeXt. Default: 50.
- **out_indices** (*Sequence[int]*) –Output from which stages. Default: (4,).
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Default: -1.
- **conv_cfg** (*dict*) –Config dict for convolution layer. Default: None.

- **norm_cfg** (*dict*) –Dictionary to construct and config norm layer. Default: dict(type=' BN' , requires_grad=True).
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type=' LeakyReLU' , negative_slope=0.1).
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None.

示例

```
>>> from mmcls.models import CSPResNeXt
>>> import torch
>>> model = CSPResNeXt(depth=50, out_indices=(0, 1, 2, 3))
>>> model.eval()
>>> inputs = torch.rand(1, 3, 224, 224)
>>> level_outputs = model(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
...
(1, 256, 56, 56)
(1, 512, 28, 28)
(1, 1024, 14, 14)
(1, 2048, 7, 7)
```

57.7.5 mmcls.models.CSPResNet

class mmcls.models.**CSPResNet**(*depth*, *in_channels=3*, *out_indices=(3,)*, *frozen_stages=- 1*, *deep_stem=False*, *conv_cfg=None*, *norm_cfg={'eps': 1e-05, 'type': 'BN'}*, *act_cfg={'inplace': True, 'type': 'LeakyReLU'}*, *norm_eval=False*, *init_cfg={'layer': 'Conv2d', 'type': 'Kaiming'}*)

CSP-ResNet backbone.

参数

- **depth** (*int*) –Depth of CSP-ResNet. Default: 50.
- **out_indices** (*Sequence[int]*) –Output from which stages. Default: (4,).
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Default: -1.
- **conv_cfg** (*dict*) –Config dict for convolution layer. Default: None.

- **norm_cfg** (*dict*) –Dictionary to construct and config norm layer. Default: dict(type='BN', requires_grad=True).
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type='LeakyReLU', negative_slope=0.1).
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Default: None.

示例

```
>>> from mmcls.models import CSPResNet
>>> import torch
>>> model = CSPResNet(depth=50, out_indices=(0, 1, 2, 3))
>>> model.eval()
>>> inputs = torch.rand(1, 3, 416, 416)
>>> level_outputs = model(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
...
(1, 128, 104, 104)
(1, 256, 52, 52)
(1, 512, 26, 26)
(1, 1024, 13, 13)
```

57.7.6 mmcls.models.Conformer

class mmcls.models.**Conformer**(*arch='tiny', patch_size=16, base_channels=64, mlp_ratio=4.0, qkv_bias=True, with_cls_token=True, drop_path_rate=0.0, norm_eval=True, frozen_stages=0, out_indices=- 1, init_cfg=None*)

Conformer backbone.

A PyTorch implementation of : Conformer: Local Features Coupling Global Representations for Visual Recognition

参数

- **arch** (*str | dict*) –Conformer architecture. Defaults to 'tiny'.
- **patch_size** (*int*) –The patch size. Defaults to 16.
- **base_channels** (*int*) –The base number of channels in CNN network. Defaults to 64.

- **mlp_ratio** (*float*) –The expansion ratio of FFN network in transformer block. Defaults to 4.
- **with_cls_token** (*bool*) –Whether use class token or not. Defaults to True.
- **drop_path_rate** (*float*) –stochastic depth rate. Defaults to 0.
- **out_indices** (*Sequence | int*) –Output from which stages. Defaults to -1, means the last stage.
- **init_cfg** (*dict, optional*) –Initialization config dict. Defaults to None.

57.7.7 mmcls.models.ConvMixer

class mmcls.models.**ConvMixer**(*arch='768/32'*, *in_channels=3*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'GELU'}*, *out_indices=- 1*, *frozen_stages=0*, *init_cfg=None*)

ConvMixer. .

A PyTorch implementation of : Patches Are All You Need?

Modified from the official repo and timm.

参数

- **arch** (*str | dict*) –The model's architecture. If string, it should be one of architecture in `ConvMixer.arch_settings`. And if dict, it should include the following two keys:
  - embed_dims (int): The dimensions of patch embedding.
  - depth (int): Number of repetitions of ConvMixer Layer.
  - patch_size (int): The patch size.
  - kernel_size (int): The kernel size of depthwise conv layers.

  Defaults to '768/32'.
- **in_channels** (*int*) –Number of input image channels. Defaults to 3.
- **patch_size** (*int*) –The size of one patch in the patch embed layer. Defaults to 7.
- **norm_cfg** (*dict*) –The config dict for norm layers. Defaults to `dict(type='BN')`.
- **act_cfg** (*dict*) –The config dict for activation after each convolution. Defaults to `dict(type='GELU')`.
- **out_indices** (*Sequence | int*) –Output from which stages. Defaults to -1, means the last stage.
- **frozen_stages** (*int*) –Stages to be frozen (all param fixed). Defaults to 0, which means not freezing any parameters.
- **init_cfg** (*dict, optional*) –Initialization config dict.

57.7.8 mmcls.models.ConvNeXt

class mmcls.models.**ConvNeXt**(*arch='tiny'*, *in_channels=3*, *stem_patch_size=4*, *norm_cfg={'eps': 1e-06, 'type': 'LN2d'}*, *act_cfg={'type': 'GELU'}*, *linear_pw_conv=True*, *drop_path_rate=0.0*, *layer_scale_init_value=1e-06*, *out_indices=- 1*, *frozen_stages=0*, *gap_before_final_norm=True*, *with_cp=False*, *init_cfg=None*)

ConvNeXt.

A PyTorch implementation of : A ConvNet for the 2020s

Modified from the official repo and timm.

参数

- **arch** (*str | dict*) –The model's architecture. If string, it should be one of architecture in `ConvNeXt.arch_settings`. And if dict, it should include the following two keys:
  - depths (list[int]): Number of blocks at each stage.
  - channels (list[int]): The number of channels at each stage.

  Defaults to 'tiny'.
- **in_channels** (*int*) –Number of input image channels. Defaults to 3.
- **stem_patch_size** (*int*) –The size of one patch in the stem layer. Defaults to 4.
- **norm_cfg** (*dict*) –The config dict for norm layers. Defaults to `dict(type='LN2d', eps=1e-6)`.
- **act_cfg** (*dict*) –The config dict for activation between pointwise convolution. Defaults to `dict(type='GELU')`.
- **linear_pw_conv** (*bool*) –Whether to use linear layer to do pointwise convolution. Defaults to True.
- **drop_path_rate** (*float*) –Stochastic depth rate. Defaults to 0.
- **layer_scale_init_value** (*float*) –Init value for Layer Scale. Defaults to 1e-6.
- **out_indices** (*Sequence | int*) –Output from which stages. Defaults to -1, means the last stage.
- **frozen_stages** (*int*) –Stages to be frozen (all param fixed). Defaults to 0, which means not freezing any parameters.
- **gap_before_final_norm** (*bool*) –Whether to globally average the feature map before the final norm layer. In the official repo, it's only used in classification task. Defaults to True.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Defaults to False.

- **init_cfg** (*dict, optional*) –Initialization config dict

57.7.9 mmcls.models.DenseNet

class mmcls.models.**DenseNet**(*arch='121', in_channels=3, bn_size=4, drop_rate=0, compression_factor=0.5, memory_efficient=False, norm_cfg={'type': 'BN'}, act_cfg={'type': 'ReLU'}, out_indices=- 1, frozen_stages=0, init_cfg=None*)

DenseNet.

A PyTorch implementation of : Densely Connected Convolutional Networks

Modified from the official repo and pytorch.

参数

- **arch** (*str | dict*) –The model's architecture. If string, it should be one of architecture in `DenseNet.arch_settings`. And if dict, it should include the following two keys:
  - growth_rate (int): Each layer of DenseBlock produce *k* feature maps. Here refers *k* as the growth rate of the network.
  - depths (list[int]): Number of repeated layers in each DenseBlock.
  - init_channels (int): The output channels of stem layers.

  Defaults to '121'.
- **in_channels** (*int*) –Number of input image channels. Defaults to 3.
- **bn_size** (*int*) –Refers to channel expansion parameter of 1x1 convolution layer. Defaults to 4.
- **drop_rate** (*float*) –Drop rate of Dropout Layer. Defaults to 0.
- **compression_factor** (*float*) –The reduction rate of transition layers. Defaults to 0.5.
- **memory_efficient** (*bool*) –If True, uses checkpointing. Much more memory efficient, but slower. Defaults to False. See "paper".
- **norm_cfg** (*dict*) –The config dict for norm layers. Defaults to `dict(type='BN')`.
- **act_cfg** (*dict*) –The config dict for activation after each convolution. Defaults to `dict(type='ReLU')`.
- **out_indices** (*Sequence | int*) –Output from which stages. Defaults to -1, means the last stage.
- **frozen_stages** (*int*) –Stages to be frozen (all param fixed). Defaults to 0, which means not freezing any parameters.
- **init_cfg** (*dict, optional*) –Initialization config dict.

57.7.10 mmcls.models.DistilledVisionTransformer

class mmcls.models.**DistilledVisionTransformer**(*arch='deit-base'*, **args*, ***kwargs*)

Distilled Vision Transformer.

A PyTorch implement of : Training data-efficient image transformers & distillation through attention

参数

- **arch** (*str | dict*) –Vision Transformer architecture. If use string, choose from 'small', 'base', 'large', 'deit-tiny', 'deit-small' and 'deit-base'. If use dict, it should have below keys:
  - **embed_dims** (int): The dimensions of embedding.
  - **num_layers** (int): The number of transformer encoder layers.
  - **num_heads** (int): The number of heads in attention modules.
  - **feedforward_channels** (int): The hidden dimensions in feedforward modules.

  Defaults to 'deit-base'.
- **img_size** (*int | tuple*) –The expected input image shape. Because we support dynamic input shape, just set the argument to the most common input image shape. Defaults to 224.
- **patch_size** (*int | tuple*) –The patch size in patch embedding. Defaults to 16.
- **in_channels** (*int*) –The num of input channels. Defaults to 3.
- **out_indices** (*Sequence | int*) –Output from which stages. Defaults to -1, means the last stage.
- **drop_rate** (*float*) –Probability of an element to be zeroed. Defaults to 0.
- **drop_path_rate** (*float*) –stochastic depth rate. Defaults to 0.
- **qkv_bias** (*bool*) –Whether to add bias for qkv in attention modules. Defaults to True.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Defaults to `dict(type='LN')`.
- **final_norm** (*bool*) –Whether to add a additional layer to normalize final feature map. Defaults to True.
- **with_cls_token** (*bool*) –Whether concatenating class token into image tokens as transformer input. Defaults to True.
- **output_cls_token** (*bool*) –Whether output the cls_token. If set True, `with_cls_token` must be True. Defaults to True.
- **interpolate_mode** (*str*) –Select the interpolate mode for position embeding vector resize. Defaults to "bicubic".

- **patch_cfg** (*dict*) –Configs of patch embeding. Defaults to an empty dict.
- **layer_cfgs** (*Sequence | dict*) –Configs of each transformer layer in encoder. Defaults to an empty dict.
- **init_cfg** (*dict, optional*) –Initialization config dict. Defaults to None.

57.7.11 mmcls.models.EfficientNet

class mmcls.models.**EfficientNet**(*arch='b0', drop_path_rate=0.0, out_indices=(6,), frozen_stages=0, conv_cfg={'type': 'Conv2dAdaptivePadding'}, norm_cfg={'eps': 0.001, 'type': 'BN'}, act_cfg={'type': 'Swish'}, norm_eval=False, with_cp=False, init_cfg=[{'type': 'Kaiming', 'layer': 'Conv2d'}, {'type': 'Constant', 'layer': ['_BatchNorm', 'GroupNorm'], 'val': 1}]*)

EfficientNet backbone.

参数

- **arch** (*str*) –Architecture of efficientnet. Defaults to b0.
- **out_indices** (*Sequence[int]*) –Output from which stages. Defaults to (6,).
- **frozen_stages** (*int*) –Stages to be frozen (all param fixed). Defaults to 0, which means not freezing any parameters.
- **conv_cfg** (*dict*) –Config dict for convolution layer. Defaults to None, which means using conv2d.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Defaults to dict(type='BN').
- **act_cfg** (*dict*) –Config dict for activation layer. Defaults to dict(type='Swish').
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Defaults to False.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Defaults to False.

57.7.12 mmcls.models.HRNet

class mmcls.models.**HRNet**(*arch='w32', extra=None, in_channels=3, conv_cfg=None, norm_cfg={'type': 'BN'}, norm_eval=False, with_cp=False, zero_init_residual=False, multiscale_output=True, init_cfg=[{'type': 'Kaiming', 'layer': 'Conv2d'}, {'type': 'Constant', 'val': 1, 'layer': ['_BatchNorm', 'GroupNorm']}]*)

HRNet backbone.

High-Resolution Representations for Labeling Pixels and Regions.

参数

- **arch** (*str*) –The preset HRNet architecture, includes ‘w18’, ‘w30’, ‘w32’, ‘w40’, ‘w44’, ‘w48’, ‘w64’. It will only be used if extra is `None`. Defaults to ‘w32’.
- **extra** (*dict, optional*) –Detailed configuration for each stage of HRNet. There must be 4 stages, the configuration for each stage must have 5 keys:
  - num_modules (int): The number of HRModule in this stage.
  - num_branches (int): The number of branches in the HRModule.
  - block (str): The type of convolution block. Please choose between ‘BOTTLENECK’ and ‘BASIC’.
  - num_blocks (tuple): The number of blocks in each branch. The length must be equal to num_branches.
  - num_channels (tuple): The number of base channels in each branch. The length must be equal to num_branches.

  Defaults to None.
- **in_channels** (*int*) –Number of input image channels. Defaults to 3.
- **conv_cfg** (*dict, optional*) –Dictionary to construct and config conv layer. Defaults to None.
- **norm_cfg** (*dict*) –Dictionary to construct and config norm layer. Defaults to `dict(type='BN')`.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Defaults to False.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Defaults to False.
- **zero_init_residual** (*bool*) –Whether to use zero init for last norm layer in resblocks to let them behave as identity. Defaults to False.
- **multiscale_output** (*bool*) –Whether to output multi-level features produced by multiple branches. If False, only the first level feature will be output. Defaults to True.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Defaults to None.

示例

```
>>> import torch
>>> from mmcls.models import HRNet
>>> extra = dict(
>>>     stage1=dict(
>>>         num_modules=1,
>>>         num_branches=1,
>>>         block='BOTTLENECK',
>>>         num_blocks=(4, ),
>>>         num_channels=(64, )),
>>>     stage2=dict(
>>>         num_modules=1,
>>>         num_branches=2,
>>>         block='BASIC',
>>>         num_blocks=(4, 4),
>>>         num_channels=(32, 64)),
>>>     stage3=dict(
>>>         num_modules=4,
>>>         num_branches=3,
>>>         block='BASIC',
>>>         num_blocks=(4, 4, 4),
>>>         num_channels=(32, 64, 128)),
>>>     stage4=dict(
>>>         num_modules=3,
>>>         num_branches=4,
>>>         block='BASIC',
>>>         num_blocks=(4, 4, 4, 4),
>>>         num_channels=(32, 64, 128, 256)))
>>> self = HRNet(extra, in_channels=1)
>>> self.eval()
>>> inputs = torch.rand(1, 1, 32, 32)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 32, 8, 8)
(1, 64, 4, 4)
(1, 128, 2, 2)
(1, 256, 1, 1)
```

57.7.13 mmcls.models.LeNet5

class mmcls.models.**LeNet5**(*num_classes=-1*)

LeNet5 backbone.

The input for LeNet-5 is a 32×32 grayscale image.

参数 **num_classes** (*int*) –number of classes for classification. The default value is -1, which uses the backbone as a feature extractor without the top classifier.

57.7.14 mmcls.models.MlpMixer

class mmcls.models.**MlpMixer**(*arch='base'*, *img_size=224*, *patch_size=16*, *out_indices=-1*, *drop_rate=0.0*, *drop_path_rate=0.0*, *norm_cfg={'type': 'LN'}*, *act_cfg={'type': 'GELU'}*, *patch_cfg={}*, *layer_cfgs={}*, *init_cfg=None*)

Mlp-Mixer backbone.

Pytorch implementation of MLP-Mixer: An all-MLP Architecture for Vision

参数

- **arch** (*str | dict*) –MLP Mixer architecture. If use string, choose from 'small', 'base' and 'large'. If use dict, it should have below keys:
  - **embed_dims** (int): The dimensions of embedding.
  - **num_layers** (int): The number of MLP blocks.
  - **tokens_mlp_dims** (int): The hidden dimensions for tokens FFNs.
  - **channels_mlp_dims** (int): The The hidden dimensions for channels FFNs.

  Defaults to 'base'.
- **img_size** (*int | tuple*) –The input image shape. Defaults to 224.
- **patch_size** (*int | tuple*) –The patch size in patch embedding. Defaults to 16.
- **out_indices** (*Sequence | int*) –Output from which layer. Defaults to -1, means the last layer.
- **drop_rate** (*float*) –Probability of an element to be zeroed. Defaults to 0.
- **drop_path_rate** (*float*) –stochastic depth rate. Defaults to 0.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Defaults to `dict(type='LN')`.
- **act_cfg** (*dict*) –The activation config for FFNs. Default GELU.
- **patch_cfg** (*dict*) –Configs of patch embeding. Defaults to an empty dict.

- **layer_cfgs** (*Sequence | dict*) –Configs of each mixer block layer. Defaults to an empty dict.
- **init_cfg** (*dict, optional*) –Initialization config dict. Defaults to None.

57.7.15 mmcls.models.MobileNetV2

class mmcls.models.**MobileNetV2**(*widen_factor=1.0*, *out_indices=(7,)*, *frozen_stages=-1*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU6'}*, *norm_eval=False*, *with_cp=False*, *init_cfg=[{'type': 'Kaiming', 'layer': ['Conv2d']}, {'type': 'Constant', 'val': 1, 'layer': ['_BatchNorm', 'GroupNorm']}]*)

MobileNetV2 backbone.

参数

- **widen_factor** (*float*) –Width multiplier, multiply number of channels in each layer by this amount. Default: 1.0.
- **out_indices** (*None or Sequence[int]*) –Output from which stages. Default: (7,).
- **frozen_stages** (*int*) –Stages to be frozen (all param fixed). Default: -1, which means not freezing any parameters.
- **conv_cfg** (*dict, optional*) –Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type='BN').
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type='ReLU6').
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.

57.7.16 mmcls.models.MobileNetV3

class mmcls.models.**MobileNetV3**(*arch='small'*, *conv_cfg=None*, *norm_cfg={'eps': 0.001, 'momentum': 0.01, 'type': 'BN'}*, *out_indices=None*, *frozen_stages=-1*, *norm_eval=False*, *with_cp=False*, *init_cfg=[{'type': 'Kaiming', 'layer': ['Conv2d'], 'nonlinearity': 'leaky_relu'}, {'type': 'Normal', 'layer': ['Linear'], 'std': 0.01}, {'type': 'Constant', 'layer': ['BatchNorm2d'], 'val': 1}]*)

MobileNetV3 backbone.

参数

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

57.7.17 mmcls.models.PCPVT

class mmcls.models.**PCPVT**(*arch*, *in_channels=3*, *out_indices=(3,)*, *qkv_bias=False*, *drop_rate=0.0*, *attn_drop_rate=0.0*, *drop_path_rate=0.0*, *norm_cfg={'type': 'LN'}*, *norm_after_stage=False*, *init_cfg=None*)

The backbone of Twins-PCPVT.

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

参数

- **arch** (*dict, str*) –PCPVT architecture, a str value in arch zoo or a detailed configuration dict with 7 keys, and the length of all the values in dict should be the same:
  - depths (List[int]): The number of encoder layers in each stage.
  - embed_dims (List[int]): Embedding dimension in each stage.
  - patch_sizes (List[int]): The patch sizes in each stage.
  - num_heads (List[int]): Numbers of attention head in each stage.
  - strides (List[int]): The strides in each stage.
  - mlp_ratios (List[int]): The ratios of mlp in each stage.
  - **sr_ratios (List[int]): The ratios of GSA-encoder layers in each** stage.
- **in_channels** (*int*) –Number of input channels. Default: 3.
- **out_indices** (*tuple[int]*) –Output from which stages. Default: (3,).

- **qkv_bias** (*bool*) –Enable bias for qkv if True. Default: False.
- **drop_rate** (*float*) –Probability of an element to be zeroed. Default 0.
- **attn_drop_rate** (*float*) –The drop out rate for attention layer. Default 0.0
- **drop_path_rate** (*float*) –Stochastic depth rate. Default 0.0
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type='LN')
- **norm_after_stage** (*bool, List[bool]*) –Add extra norm after each stage. Default False.
- **init_cfg** (*dict, optional*) –The Config for initialization. Defaults to None.

实际案例

```
>>> from mmcls.models import PCPVT
>>> import torch
>>> pcpvt_cfg = {'arch': "small",
>>>              'norm_after_stage': [False, False, False, True]}
>>> model = PCPVT(**pcpvt_cfg)
>>> x = torch.rand(1, 3, 224, 224)
>>> outputs = model(x)
>>> print(outputs[-1].shape)
torch.Size([1, 512, 7, 7])
>>> pcpvt_cfg['norm_after_stage'] = [True, True, True, True]
>>> pcpvt_cfg['out_indices'] = (0, 1, 2, 3)
>>> model = PCPVT(**pcpvt_cfg)
>>> outputs = model(x)
>>> for feat in outputs:
>>>     print(feat.shape)
torch.Size([1, 64, 56, 56])
torch.Size([1, 128, 28, 28])
torch.Size([1, 320, 14, 14])
torch.Size([1, 512, 7, 7])
```

57.7.18 mmcls.models.PoolFormer

class mmcls.models.**PoolFormer**(*arch='s12', pool_size=3, norm_cfg={'num_groups': 1, 'type': 'GN'}, act_cfg={'type': 'GELU'}, in_patch_size=7, in_stride=4, in_pad=2, down_patch_size=3, down_stride=2, down_pad=1, drop_rate=0.0, drop_path_rate=0.0, out_indices=- 1, frozen_stages=0, init_cfg=None*)

PoolFormer.

A PyTorch implementation of PoolFormer introduced by: MetaFormer is Actually What You Need for Vision

Modified from the *official repo <https://github.com/sail-sg/poolformer/blob/main/models/poolformer.py>*.

参数

- **arch** (*str | dict*) –The model’s architecture. If string, it should be one of architecture in `PoolFormer.arch_settings`. And if dict, it should include the following two keys:
  - layers (list[int]): Number of blocks at each stage.
  - embed_dims (list[int]): The number of channels at each stage.
  - mlp_ratios (list[int]): Expansion ratio of MLPs.
  - layer_scale_init_value (float): Init value for Layer Scale.

  Defaults to ‘S12’.
- **norm_cfg** (*dict*) –The config dict for norm layers. Defaults to `dict(type='LN2d', eps=1e-6)`.
- **act_cfg** (*dict*) –The config dict for activation between pointwise convolution. Defaults to `dict(type='GELU')`.
- **in_patch_size** (*int*) –The patch size of input image patch embedding. Defaults to 7.
- **in_stride** (*int*) –The stride of input image patch embedding. Defaults to 4.
- **in_pad** (*int*) –The padding of input image patch embedding. Defaults to 2.
- **down_patch_size** (*int*) –The patch size of downsampling patch embedding. Defaults to 3.
- **down_stride** (*int*) –The stride of downsampling patch embedding. Defaults to 2.
- **down_pad** (*int*) –The padding of downsampling patch embedding. Defaults to 1.
- **drop_rate** (*float*) –Dropout rate. Defaults to 0.
- **drop_path_rate** (*float*) –Stochastic depth rate. Defaults to 0.
- **out_indices** (*Sequence | int*) –Output from which network position. Index 0-6 respectively corresponds to [stage1, downsampling, stage2, downsampling, stage3, downsampling, stage4] Defaults to -1, means the last stage.
- **frozen_stages** (*int*) –Stages to be frozen (all param fixed). Defaults to 0, which means not freezing any parameters.
- **init_cfg** (*dict, optional*) –Initialization config dict

57.7.19 mmcls.models.RegNet

class mmcls.models.**RegNet**(*arch*, *in_channels=3*, *stem_channels=32*, *base_channels=32*, *strides=(2, 2, 2, 2)*, *dilations=(1, 1, 1, 1)*, *out_indices=(3,)*, *style='pytorch'*, *deep_stem=False*, *avg_down=False*, *frozen_stages=-1*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *norm_eval=False*, *with_cp=False*, *zero_init_residual=True*, *init_cfg=None*)

RegNet backbone.

More details can be found in paper .

参数

- **arch** (*dict*) –The parameter of RegNets. - w0 (int): initial width - wa (float): slope of width - wm (float): quantization parameter to quantize the width - depth (int): depth of the backbone - group_w (int): width of group - bot_mul (float): bottleneck ratio, i.e. expansion of bottleneck.
- **strides** (*Sequence[int]*) –Strides of the first block of each stage.
- **base_channels** (*int*) –Base channels after stem layer.
- **in_channels** (*int*) –Number of input image channels. Default: 3.
- **dilations** (*Sequence[int]*) –Dilation of each stage.
- **out_indices** (*Sequence[int]*) –Output from which stages.
- **style** (*str*) –*pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer. Default: "pytorch".
- **frozen_stages** (*int*) –Stages to be frozen (all param fixed). -1 means not freezing any parameters. Default: -1.
- **norm_cfg** (*dict*) –dictionary to construct and config norm layer. Default: dict(type='BN', requires_grad=True).
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **zero_init_residual** (*bool*) –whether to use zero init for last norm layer in resblocks to let them behave as identity. Default: True.

示例

```
>>> from mmcls.models import RegNet
>>> import torch
>>> inputs = torch.rand(1, 3, 32, 32)
>>> # use str type 'arch'
>>> # Note that default out_indices is (3,)
>>> regnet_cfg = dict(arch='regnetx_4.0gf')
>>> model = RegNet(**regnet_cfg)
>>> model.eval()
>>> level_outputs = model(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
    (1, 1360, 1, 1)
>>> # use dict type 'arch'
>>> arch_cfg =dict(w0=88, wa=26.31, wm=2.25,
>>>                group_w=48, depth=25, bot_mul=1.0)
>>> regnet_cfg = dict(arch=arch_cfg, out_indices=(0, 1, 2, 3))
>>> model = RegNet(**regnet_cfg)
>>> model.eval()
>>> level_outputs = model(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
    (1, 96, 8, 8)
    (1, 192, 4, 4)
    (1, 432, 2, 2)
    (1, 1008, 1, 1)
```

57.7.20 mmcls.models.RepMLPNet

class mmcls.models.**RepMLPNet**(*arch*, *img_size=224*, *in_channels=3*, *patch_size=4*, *out_indices=(3,)*, *reparam_conv_kernels=(3,)*, *globalperceptron_ratio=4*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *patch_cfg={}*, *final_norm=True*, *deploy=False*, *init_cfg=None*)

RepMLPNet backbone.

A PyTorch impl of : RepMLP: Re-parameterizing Convolutions into Fully-connected Layers for Image Recognition

参数

- **arch** (*str | dict*) –RepMLP architecture. If use string, choose from 'base' and 'b'. If use dict, it should have below keys:
  - channels (List[int]): Number of blocks in each stage.

  - depths (List[int]): The number of blocks in each branch.
  - sharesets_nums (List[int]): RepVGG Block that declares the need to apply group convolution.

- **img_size** (*int | tuple*) –The size of input image. Defaults: 224.
- **in_channels** (*int*) –Number of input image channels. Default: 3.
- **patch_size** (*int | tuple*) –The patch size in patch embedding. Defaults to 4.
- **out_indices** (*Sequence[int]*) –Output from which stages. Default: `(3, )`.
- **reparam_conv_kernels** (*Squeue(int) | None*) –The conv kernels in the GlobalPerceptron. Default: None.
- **globalperceptron_ratio** (*int*) –The reducation ratio in the GlobalPerceptron. Default: 4.
- **num_sharesets** (*int*) –The number of sharesets in the PartitionPerceptron. Default 1.
- **conv_cfg** (*dict | None*) –The config dict for conv layers. Default: None.
- **norm_cfg** (*dict*) –The config dict for norm layers. Default: dict(type='BN', requires_grad=True).
- **patch_cfg** (*dict*) –Extra config dict for patch embedding. Defaults to an empty dict.
- **final_norm** (*bool*) –Whether to add a additional layer to normalize final feature map. Defaults to True.
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type='ReLU').
- **deploy** (*bool*) –Whether to switch the model structure to deployment mode. Default: False.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

57.7.21 mmcls.models.RepVGG

class mmcls.models.**RepVGG**(*arch*, *in_channels=3*, *base_channels=64*, *out_indices=(3,)*, *strides=(2, 2, 2, 2)*, *dilations=(1, 1, 1, 1)*, *frozen_stages=- 1*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *with_cp=False*, *deploy=False*, *norm_eval=False*, *add_ppf=False*, *init_cfg=[{'type': 'Kaiming', 'layer': ['Conv2d']}, {'type': 'Constant', 'val': 1, 'layer': ['_BatchNorm', 'GroupNorm']}]*)

RepVGG backbone.

A PyTorch impl of : RepVGG: Making VGG-style ConvNets Great Again

参数

- **arch** (*str | dict*) –RepVGG architecture. If use string, choose from 'A0', 'A1'', 'A2', 'B0', 'B1', 'B1g2', 'B1g4', 'B2', 'B2g2', 'B2g4', 'B3', 'B3g2', 'B3g4' or 'D2se'. If use dict,

  it should have below keys:

  - num_blocks (Sequence[int]): Number of blocks in each stage.
  - width_factor (Sequence[float]): Width deflator in each stage.
  - group_layer_map (dict | None): RepVGG Block that declares the need to apply group convolution.
  - se_cfg (dict | None): Se Layer config.
  - **stem_channels (int, optional): The stem channels, the final**

    stem channels will be `min(stem_channels, base_channels*width_factor[0])`.

    If not set here, 64 is used by default in the code.

- **in_channels** (*int*) –Number of input image channels. Default: 3.
- **base_channels** (*int*) –Base channels of RepVGG backbone, work with width_factor together. Defaults to 64.
- **out_indices** (*Sequence[int]*) –Output from which stages. Default: (3,).
- **strides** (*Sequence[int]*) –Strides of the first block of each stage. Default: (2, 2, 2, 2).
- **dilations** (*Sequence[int]*) –Dilation of each stage. Default: (1, 1, 1, 1).
- **frozen_stages** (*int*) –Stages to be frozen (all param fixed). -1 means not freezing any parameters. Default: -1.
- **conv_cfg** (*dict | None*) –The config dict for conv layers. Default: None.
- **norm_cfg** (*dict*) –The config dict for norm layers. Default: dict(type='BN').
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type='ReLU').
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **deploy** (*bool*) –Whether to switch the model structure to deployment mode. Default: False.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **add_ppf** (*bool*) –Whether to use the MTSPPF block. Default: False.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict.

57.7.22 mmcls.models.Res2Net

class mmcls.models.**Res2Net**(*scales=4*, *base_width=26*, *style='pytorch'*, *deep_stem=True*, *avg_down=True*, *init_cfg=None*, ***kwargs*)

Res2Net backbone.

A PyTorch implement of : Res2Net: A New Multi-scale Backbone Architecture

参数

- **depth** (*int*) –Depth of Res2Net, choose from {50, 101, 152}.
- **scales** (*int*) –Scales used in Res2Net. Defaults to 4.
- **base_width** (*int*) –Basic width of each scale. Defaults to 26.
- **in_channels** (*int*) –Number of input image channels. Defaults to 3.
- **num_stages** (*int*) –Number of Res2Net stages. Defaults to 4.
- **strides** (*Sequence[int]*) –Strides of the first block of each stage. Defaults to `(1, 2, 2, 2)`.
- **dilations** (*Sequence[int]*) –Dilation of each stage. Defaults to `(1, 1, 1, 1)`.
- **out_indices** (*Sequence[int]*) –Output from which stages. Defaults to `(3, )`.
- **style** (*str*) –"pytorch" or "caffe" . If set to "pytorch" , the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer. Defaults to "pytorch" .
- **deep_stem** (*bool*) –Replace 7x7 conv in input stem with 3 3x3 conv. Defaults to True.
- **avg_down** (*bool*) –Use AvgPool instead of stride conv when downsampling in the bottle2neck. Defaults to True.
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Defaults to -1.
- **norm_cfg** (*dict*) –Dictionary to construct and config norm layer. Defaults to `dict(type='BN', requires_grad=True)`.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Defaults to False.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Defaults to False.
- **zero_init_residual** (*bool*) –Whether to use zero init for last norm layer in resblocks to let them behave as identity. Defaults to True.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict. Defaults to None.

示例

```
>>> from mmcls.models import Res2Net
>>> import torch
>>> model = Res2Net(depth=50,
...                 scales=4,
...                 base_width=26,
...                 out_indices=(0, 1, 2, 3))
>>> model.eval()
>>> inputs = torch.rand(1, 3, 32, 32)
>>> level_outputs = model.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 256, 8, 8)
(1, 512, 4, 4)
(1, 1024, 2, 2)
(1, 2048, 1, 1)
```

57.7.23 mmcls.models.ResNeSt

class mmcls.models.**ResNeSt**(*depth*, *groups=1*, *width_per_group=4*, *radix=2*, *reduction_factor=4*, *avg_down_stride=True*, ***kwargs*)

ResNeSt backbone.

Please refer to the paper for details.

参数

- **depth** (*int*) –Network depth, from {50, 101, 152, 200}.
- **groups** (*int*) –Groups of conv2 in Bottleneck. Default: 32.
- **width_per_group** (*int*) –Width per group of conv2 in Bottleneck. Default: 4.
- **radix** (*int*) –Radix of SpltAtConv2d. Default: 2
- **reduction_factor** (*int*) –Reduction factor of SplitAttentionConv2d. Default: 4.
- **avg_down_stride** (*bool*) –Whether to use average pool for stride in Bottleneck. Default: True.
- **in_channels** (*int*) –Number of input image channels. Default: 3.
- **stem_channels** (*int*) –Output channels of the stem layer. Default: 64.
- **num_stages** (*int*) –Stages of the network. Default: 4.
- **strides** (*Sequence[int]*) –Strides of the first block of each stage. Default: `(1, 2, 2, 2)`.

- **dilations** (*Sequence[int]*) –Dilation of each stage. Default: `(1, 1, 1, 1)`.
- **out_indices** (*Sequence[int]*) –Output from which stages. If only one stage is specified, a single tensor (feature map) is returned, otherwise multiple stages are specified, a tuple of tensors will be returned. Default: `(3, )`.
- **style** (*str*) –*pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer.
- **deep_stem** (*bool*) –Replace 7x7 conv in input stem with 3 3x3 conv. Default: False.
- **avg_down** (*bool*) –Use AvgPool instead of stride conv when downsampling in the bottleneck. Default: False.
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Default: -1.
- **conv_cfg** (*dict | None*) –The config dict for conv layers. Default: None.
- **norm_cfg** (*dict*) –The config dict for norm layers.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **zero_init_residual** (*bool*) –Whether to use zero init for last norm layer in resblocks to let them behave as identity. Default: True.

57.7.24 mmcls.models.ResNeXt

class mmcls.models.**ResNeXt**(*depth*, *groups=32*, *width_per_group=4*, ***kwargs*)

ResNeXt backbone.

Please refer to the paper for details.

参数

- **depth** (*int*) –Network depth, from {50, 101, 152}.
- **groups** (*int*) –Groups of conv2 in Bottleneck. Default: 32.
- **width_per_group** (*int*) –Width per group of conv2 in Bottleneck. Default: 4.
- **in_channels** (*int*) –Number of input image channels. Default: 3.
- **stem_channels** (*int*) –Output channels of the stem layer. Default: 64.
- **num_stages** (*int*) –Stages of the network. Default: 4.
- **strides** (*Sequence[int]*) –Strides of the first block of each stage. Default: `(1, 2, 2, 2)`.

- **dilations** (*Sequence[int]*) –Dilation of each stage. Default: `(1, 1, 1, 1)`.
- **out_indices** (*Sequence[int]*) –Output from which stages. If only one stage is specified, a single tensor (feature map) is returned, otherwise multiple stages are specified, a tuple of tensors will be returned. Default: `(3, )`.
- **style** (*str*) –*pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer.
- **deep_stem** (*bool*) –Replace 7x7 conv in input stem with 3 3x3 conv. Default: False.
- **avg_down** (*bool*) –Use AvgPool instead of stride conv when downsampling in the bottleneck. Default: False.
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Default: -1.
- **conv_cfg** (*dict | None*) –The config dict for conv layers. Default: None.
- **norm_cfg** (*dict*) –The config dict for norm layers.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **zero_init_residual** (*bool*) –Whether to use zero init for last norm layer in resblocks to let them behave as identity. Default: True.

57.7.25 mmcls.models.ResNet

class mmcls.models.**ResNet**(*depth*, *in_channels=3*, *stem_channels=64*, *base_channels=64*, *expansion=None*, *num_stages=4*, *strides=(1, 2, 2, 2)*, *dilations=(1, 1, 1, 1)*, *out_indices=(3,)*, *style='pytorch'*, *deep_stem=False*, *avg_down=False*, *frozen_stages=-1*, *conv_cfg=None*, *norm_cfg={'requires_grad': True, 'type': 'BN'}*, *norm_eval=False*, *with_cp=False*, *zero_init_residual=True*, *init_cfg=[{'type': 'Kaiming', 'layer': ['Conv2d']}, {'type': 'Constant', 'val': 1, 'layer': ['_BatchNorm', 'GroupNorm']}]*, *drop_path_rate=0.0*)

ResNet backbone.

Please refer to the paper for details.

参数

- **depth** (*int*) –Network depth, from {18, 34, 50, 101, 152}.
- **in_channels** (*int*) –Number of input image channels. Default: 3.
- **stem_channels** (*int*) –Output channels of the stem layer. Default: 64.

- **base_channels** (*int*) –Middle channels of the first stage. Default: 64.
- **num_stages** (*int*) –Stages of the network. Default: 4.
- **strides** (*Sequence[int]*) –Strides of the first block of each stage. Default: `(1, 2, 2, 2)`.
- **dilations** (*Sequence[int]*) –Dilation of each stage. Default: `(1, 1, 1, 1)`.
- **out_indices** (*Sequence[int]*) –Output from which stages. Default: `(3, )`.
- **style** (*str*) –*pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer.
- **deep_stem** (*bool*) –Replace 7x7 conv in input stem with 3 3x3 conv. Default: False.
- **avg_down** (*bool*) –Use AvgPool instead of stride conv when downsampling in the bottleneck. Default: False.
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Default: -1.
- **conv_cfg** (*dict | None*) –The config dict for conv layers. Default: None.
- **norm_cfg** (*dict*) –The config dict for norm layers.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **zero_init_residual** (*bool*) –Whether to use zero init for last norm layer in resblocks to let them behave as identity. Default: True.

示例

```
>>> from mmcls.models import ResNet
>>> import torch
>>> self = ResNet(depth=18)
>>> self.eval()
>>> inputs = torch.rand(1, 3, 32, 32)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 64, 8, 8)
(1, 128, 4, 4)
(1, 256, 2, 2)
(1, 512, 1, 1)
```

57.7.26 mmcls.models.ResNetV1c

class mmcls.models.**ResNetV1c**(***kwargs*)

ResNetV1c backbone.

This variant is described in Bag of Tricks..

Compared with default ResNet(ResNetV1b), ResNetV1c replaces the 7x7 conv in the input stem with three 3x3 convs.

57.7.27 mmcls.models.ResNetV1d

class mmcls.models.**ResNetV1d**(***kwargs*)

ResNetV1d backbone.

This variant is described in Bag of Tricks..

Compared with default ResNet(ResNetV1b), ResNetV1d replaces the 7x7 conv in the input stem with three 3x3 convs. And in the downsampling block, a 2x2 avg_pool with stride 2 is added before conv, whose stride is changed to 1.

57.7.28 mmcls.models.ResNet_CIFAR

class mmcls.models.**ResNet_CIFAR**(*depth*, *deep_stem=False*, ***kwargs*)

ResNet backbone for CIFAR.

Compared to standard ResNet, it uses *kernel_size=3* and *stride=1* in conv1, and does not apply MaxPoolinng after stem. It has been proven to be more efficient than standard ResNet in other public codebase, e.g., *https://github.com/kuangliu/pytorch-cifar/blob/master/models/resnet.py*.

参数

- **depth** (*int*) –Network depth, from {18, 34, 50, 101, 152}.
- **in_channels** (*int*) –Number of input image channels. Default: 3.
- **stem_channels** (*int*) –Output channels of the stem layer. Default: 64.
- **base_channels** (*int*) –Middle channels of the first stage. Default: 64.
- **num_stages** (*int*) –Stages of the network. Default: 4.
- **strides** (*Sequence[int]*) –Strides of the first block of each stage. Default: `(1, 2, 2, 2)`.
- **dilations** (*Sequence[int]*) –Dilation of each stage. Default: `(1, 1, 1, 1)`.

- **out_indices** (*Sequence[int]*) –Output from which stages. If only one stage is specified, a single tensor (feature map) is returned, otherwise multiple stages are specified, a tuple of tensors will be returned. Default: `(3, )`.
- **style** (*str*) –*pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer.
- **deep_stem** (*bool*) –This network has specific designed stem, thus it is asserted to be False.
- **avg_down** (*bool*) –Use AvgPool instead of stride conv when downsampling in the bottleneck. Default: False.
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Default: -1.
- **conv_cfg** (*dict | None*) –The config dict for conv layers. Default: None.
- **norm_cfg** (*dict*) –The config dict for norm layers.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **zero_init_residual** (*bool*) –Whether to use zero init for last norm layer in resblocks to let them behave as identity. Default: True.

57.7.29 mmcls.models.SEResNeXt

class mmcls.models.**SEResNeXt**(*depth*, *groups=32*, *width_per_group=4*, ***kwargs*)

SEResNeXt backbone.

Please refer to the paper for details.

参数

- **depth** (*int*) –Network depth, from {50, 101, 152}.
- **groups** (*int*) –Groups of conv2 in Bottleneck. Default: 32.
- **width_per_group** (*int*) –Width per group of conv2 in Bottleneck. Default: 4.
- **se_ratio** (*int*) –Squeeze ratio in SELayer. Default: 16.
- **in_channels** (*int*) –Number of input image channels. Default: 3.
- **stem_channels** (*int*) –Output channels of the stem layer. Default: 64.
- **num_stages** (*int*) –Stages of the network. Default: 4.

- **strides** (*Sequence[int]*) –Strides of the first block of each stage. Default: `(1, 2, 2, 2)`.
- **dilations** (*Sequence[int]*) –Dilation of each stage. Default: `(1, 1, 1, 1)`.
- **out_indices** (*Sequence[int]*) –Output from which stages. If only one stage is specified, a single tensor (feature map) is returned, otherwise multiple stages are specified, a tuple of tensors will be returned. Default: `(3, )`.
- **style** (*str*) –*pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer.
- **deep_stem** (*bool*) –Replace 7x7 conv in input stem with 3 3x3 conv. Default: False.
- **avg_down** (*bool*) –Use AvgPool instead of stride conv when downsampling in the bottleneck. Default: False.
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Default: -1.
- **conv_cfg** (*dict | None*) –The config dict for conv layers. Default: None.
- **norm_cfg** (*dict*) –The config dict for norm layers.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **zero_init_residual** (*bool*) –Whether to use zero init for last norm layer in resblocks to let them behave as identity. Default: True.

57.7.30 mmcls.models.SEResNet

class mmcls.models.**SEResNet**(*depth*, *se_ratio=16*, ***kwargs*)

SEResNet backbone.

Please refer to the paper for details.

参数

- **depth** (*int*) –Network depth, from {50, 101, 152}.
- **se_ratio** (*int*) –Squeeze ratio in SELayer. Default: 16.
- **in_channels** (*int*) –Number of input image channels. Default: 3.
- **stem_channels** (*int*) –Output channels of the stem layer. Default: 64.
- **num_stages** (*int*) –Stages of the network. Default: 4.

- **strides** (*Sequence[int]*) –Strides of the first block of each stage. Default: `(1, 2, 2, 2)`.
- **dilations** (*Sequence[int]*) –Dilation of each stage. Default: `(1, 1, 1, 1)`.
- **out_indices** (*Sequence[int]*) –Output from which stages. If only one stage is specified, a single tensor (feature map) is returned, otherwise multiple stages are specified, a tuple of tensors will be returned. Default: `(3, )`.
- **style** (*str*) –*pytorch* or *caffe*. If set to "pytorch", the stride-two layer is the 3x3 conv layer, otherwise the stride-two layer is the first 1x1 conv layer.
- **deep_stem** (*bool*) –Replace 7x7 conv in input stem with 3 3x3 conv. Default: False.
- **avg_down** (*bool*) –Use AvgPool instead of stride conv when downsampling in the bottleneck. Default: False.
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Default: -1.
- **conv_cfg** (*dict | None*) –The config dict for conv layers. Default: None.
- **norm_cfg** (*dict*) –The config dict for norm layers.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.
- **zero_init_residual** (*bool*) –Whether to use zero init for last norm layer in resblocks to let them behave as identity. Default: True.

示例

```
>>> from mmcls.models import SEResNet
>>> import torch
>>> self = SEResNet(depth=50)
>>> self.eval()
>>> inputs = torch.rand(1, 3, 224, 224)
>>> level_outputs = self.forward(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 64, 56, 56)
(1, 128, 28, 28)
(1, 256, 14, 14)
(1, 512, 7, 7)
```

57.7.31 mmcls.models.SVT

class mmcls.models.**SVT**(*arch*, *in_channels=3*, *out_indices=(3,)*, *qkv_bias=False*, *drop_rate=0.0*, *attn_drop_rate=0.0*, *drop_path_rate=0.0*, *norm_cfg={'type': 'LN'}*, *norm_after_stage=False*, *init_cfg=None*)

The backbone of Twins-SVT.

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

参数

- **arch** (*dict, str*) –SVT architecture, a str value in arch zoo or a detailed configuration dict with 8 keys, and the length of all the values in dict should be the same:
  - depths (List[int]): The number of encoder layers in each stage.
  - embed_dims (List[int]): Embedding dimension in each stage.
  - patch_sizes (List[int]): The patch sizes in each stage.
  - num_heads (List[int]): Numbers of attention head in each stage.
  - strides (List[int]): The strides in each stage.
  - mlp_ratios (List[int]): The ratios of mlp in each stage.
  - **sr_ratios (List[int]): The ratios of GSA-encoder layers in each** stage.
  - **windiow_sizes (List[int]): The window sizes in LSA-encoder layers** in each stage.
- **in_channels** (*int*) –Number of input channels. Default: 3.
- **out_indices** (*tuple[int]*) –Output from which stages. Default: (3,).
- **qkv_bias** (*bool*) –Enable bias for qkv if True. Default: False.
- **drop_rate** (*float*) –Dropout rate. Default 0.
- **attn_drop_rate** (*float*) –Dropout ratio of attention weight. Default 0.0
- **drop_path_rate** (*float*) –Stochastic depth rate. Default 0.2.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type='LN')
- **norm_after_stage** (*bool, List[bool]*) –Add extra norm after each stage. Default False.
- **init_cfg** (*dict, optional*) –The Config for initialization. Defaults to None.

实际案例

```
>>> from mmcls.models import SVT
>>> import torch
>>> svt_cfg = {'arch': "small",
>>>            'norm_after_stage': [False, False, False, True]}
>>> model = SVT(**svt_cfg)
>>> x = torch.rand(1, 3, 224, 224)
>>> outputs = model(x)
>>> print(outputs[-1].shape)
torch.Size([1, 512, 7, 7])
>>> svt_cfg["out_indices"] = (0, 1, 2, 3)
>>> svt_cfg["norm_after_stage"] = [True, True, True, True]
>>> model = SVT(**svt_cfg)
>>> output = model(x)
>>> for feat in output:
>>>     print(feat.shape)
torch.Size([1, 64, 56, 56])
torch.Size([1, 128, 28, 28])
torch.Size([1, 320, 14, 14])
torch.Size([1, 512, 7, 7])
```

57.7.32 mmcls.models.ShuffleNetV1

class mmcls.models.**ShuffleNetV1**(*groups=3*, *widen_factor=1.0*, *out_indices=(2,)*, *frozen_stages=-1*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *norm_eval=False*, *with_cp=False*, *init_cfg=None*)

ShuffleNetV1 backbone.

参数

- **groups** (*int*) –The number of groups to be used in grouped 1x1 convolutions in each ShuffleUnit. Default: 3.
- **widen_factor** (*float*) –Width multiplier - adjusts the number of channels in each layer by this amount. Default: 1.0.
- **out_indices** (*Sequence[int]*) –Output from which stages. Default: (2,)
- **frozen_stages** (*int*) –Stages to be frozen (all param fixed). Default: -1, which means not freezing any parameters.
- **conv_cfg** (*dict, optional*) –Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type='BN').

- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type=’ReLU’).
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.

57.7.33 mmcls.models.ShuffleNetV2

class mmcls.models.**ShuffleNetV2**(*widen_factor=1.0*, *out_indices=(3,)*, *frozen_stages=-1*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *norm_eval=False*, *with_cp=False*, *init_cfg=None*)

ShuffleNetV2 backbone.

参数

- **widen_factor** (*float*) –Width multiplier - adjusts the number of channels in each layer by this amount. Default: 1.0.
- **out_indices** (*Sequence[int]*) –Output from which stages. Default: (0, 1, 2, 3).
- **frozen_stages** (*int*) –Stages to be frozen (all param fixed). Default: -1, which means not freezing any parameters.
- **conv_cfg** (*dict, optional*) –Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type=’BN’).
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type=’ReLU’).
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.

57.7.34 mmcls.models.SwinTransformer

class mmcls.models.**SwinTransformer**(*arch='tiny'*, *img_size=224*, *patch_size=4*, *in_channels=3*, *window_size=7*, *drop_rate=0.0*, *drop_path_rate=0.1*, *out_indices=(3,)*, *out_after_downsample=False*, *use_abs_pos_embed=False*, *interpolate_mode='bicubic'*, *with_cp=False*, *frozen_stages=-1*, *norm_eval=False*, *pad_small_map=False*, *norm_cfg={'type': 'LN'}*, *stage_cfgs={}*, *patch_cfg={}*, *init_cfg=None*)

Swin Transformer.

A PyTorch implement of : Swin Transformer: Hierarchical Vision Transformer using Shifted Windows

Inspiration from https://github.com/microsoft/Swin-Transformer

参数

- **arch** (*str | dict*) –Swin Transformer architecture. If use string, choose from 'tiny', 'small', 'base' and 'large'. If use dict, it should have below keys:
  - **embed_dims** (int): The dimensions of embedding.
  - **depths** (List[int]): The number of blocks in each stage.
  - **num_heads** (List[int]): The number of heads in attention modules of each stage.

  Defaults to 'tiny'.
- **img_size** (*int | tuple*) –The expected input image shape. Because we support dynamic input shape, just set the argument to the most common input image shape. Defaults to 224.
- **patch_size** (*int | tuple*) –The patch size in patch embedding. Defaults to 4.
- **in_channels** (*int*) –The num of input channels. Defaults to 3.
- **window_size** (*int*) –The height and width of the window. Defaults to 7.
- **drop_rate** (*float*) –Dropout rate after embedding. Defaults to 0.
- **drop_path_rate** (*float*) –Stochastic depth rate. Defaults to 0.1.
- **out_after_downsample** (*bool*) –Whether to output the feature map of a stage after the following downsample layer. Defaults to False.
- **use_abs_pos_embed** (*bool*) –If True, add absolute position embedding to the patch embedding. Defaults to False.
- **interpolate_mode** (*str*) –Select the interpolate mode for absolute position embeding vector resize. Defaults to "bicubic".
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Defaults to False.
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Defaults to -1.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Defaults to False.
- **pad_small_map** (*bool*) –If True, pad the small feature map to the window size, which is common used in detection and segmentation. If False, avoid shifting window and shrink

the window size to the size of feature map, which is common used in classification. Defaults to False.

- **norm_cfg** (*dict*) –Config dict for normalization layer for all output features. Defaults to `dict(type='LN')`
- **stage_cfgs** (*Sequence[dict] | dict*) –Extra config dict for each stage. Defaults to an empty dict.
- **patch_cfg** (*dict*) –Extra config dict for patch embedding. Defaults to an empty dict.
- **init_cfg** (*dict, optional*) –The Config for initialization. Defaults to None.

实际案例

```
>>> from mmcls.models import SwinTransformer
>>> import torch
>>> extra_config = dict(
>>>     arch='tiny',
>>>     stage_cfgs=dict(downsample_cfg={'kernel_size': 3,
>>>                                     'expansion_ratio': 3}))
>>> self = SwinTransformer(**extra_config)
>>> inputs = torch.rand(1, 3, 224, 224)
>>> output = self.forward(inputs)
>>> print(output.shape)
(1, 2592, 4)
```

57.7.35 mmcls.models.T2T_ViT

class mmcls.models.**T2T_ViT**(*img_size=224*, *in_channels=3*, *embed_dims=384*, *num_layers=14*, *out_indices=- 1*, *drop_rate=0.0*, *drop_path_rate=0.0*, *norm_cfg={'type': 'LN'}*, *final_norm=True*, *with_cls_token=True*, *output_cls_token=True*, *interpolate_mode='bicubic'*, *t2t_cfg={}*, *layer_cfgs={}*, *init_cfg=None*)

Tokens-to-Token Vision Transformer (T2T-ViT)

A PyTorch implementation of Tokens-to-Token ViT: Training Vision Transformers from Scratch on ImageNet

参数

- **img_size** (*int | tuple*) –The expected input image shape. Because we support dynamic input shape, just set the argument to the most common input image shape. Defaults to 224.
- **in_channels** (*int*) –Number of input channels.
- **embed_dims** (*int*) –Embedding dimension.

- **num_layers** (*int*) –Num of transformer layers in encoder. Defaults to 14.
- **out_indices** (*Sequence | int*) –Output from which stages. Defaults to -1, means the last stage.
- **drop_rate** (*float*) –Dropout rate after position embedding. Defaults to 0.
- **drop_path_rate** (*float*) –stochastic depth rate. Defaults to 0.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Defaults to `dict(type='LN')`.
- **final_norm** (*bool*) –Whether to add a additional layer to normalize final feature map. Defaults to True.
- **with_cls_token** (*bool*) –Whether concatenating class token into image tokens as transformer input. Defaults to True.
- **output_cls_token** (*bool*) –Whether output the cls_token. If set True, `with_cls_token` must be True. Defaults to True.
- **interpolate_mode** (*str*) –Select the interpolate mode for position embeding vector resize. Defaults to "bicubic".
- **t2t_cfg** (*dict*) –Extra config of Tokens-to-Token module. Defaults to an empty dict.
- **layer_cfgs** (*Sequence | dict*) –Configs of each transformer layer in encoder. Defaults to an empty dict.
- **init_cfg** (*dict, optional*) –The Config for initialization. Defaults to None.

57.7.36 mmcls.models.TIMMBackbone

class mmcls.models.**TIMMBackbone**(*model_name*, *features_only=False*, *pretrained=False*, *checkpoint_path=''*, *in_channels=3*, *init_cfg=None*, ***kwargs*)

Wrapper to use backbones from timm library.

More details can be found in timm. See especially the document for feature extraction.

参数

- **model_name** (*str*) –Name of timm model to instantiate.
- **features_only** (*bool*) –Whether to extract feature pyramid (multi-scale feature maps from the deepest layer at each stride). For Vision Transformer models that do not support this argument, set this False. Defaults to False.
- **pretrained** (*bool*) –Whether to load pretrained weights. Defaults to False.
- **checkpoint_path** (*str*) –Path of checkpoint to load at the last of `timm.create_model`. Defaults to empty string, which means not loading.

- **in_channels** (*int*) –Number of input image channels. Defaults to 3.
- **init_cfg** (*dict or list[dict], optional*) –Initialization config dict of OpenMMLab projects. Defaults to None.
- ****kwargs** –Other timm & model specific arguments.

57.7.37 mmcls.models.TNT

class mmcls.models.**TNT**(*arch='b'*, *img_size=224*, *patch_size=16*, *in_channels=3*, *ffn_ratio=4*, *qkv_bias=False*, *drop_rate=0.0*, *attn_drop_rate=0.0*, *drop_path_rate=0.0*, *act_cfg={'type': 'GELU'}*, *norm_cfg={'type': 'LN'}*, *first_stride=4*, *num_fcs=2*, *init_cfg=[{'type': 'TruncNormal', 'layer': 'Linear', 'std': 0.02}, {'type': 'Constant', 'layer': 'LayerNorm', 'val': 1.0, 'bias': 0.0}]*)

Transformer in Transformer.

A PyTorch implement of: Transformer in Transformer

Inspiration from https://github.com/rwightman/pytorch-image-models/blob/master/timm/models/tnt.py

参数

- **arch** (*str | dict*) –Vision Transformer architecture Default: 'b'
- **img_size** (*int | tuple*) –Input image size. Default to 224
- **patch_size** (*int | tuple*) –The patch size. Deault to 16
- **in_channels** (*int*) –Number of input channels. Default to 3
- **ffn_ratio** (*int*) –A ratio to calculate the hidden_dims in ffn layer. Default: 4
- **qkv_bias** (*bool*) –Enable bias for qkv if True. Default False
- **drop_rate** (*float*) –Probability of an element to be zeroed after the feed forward layer. Default 0.
- **attn_drop_rate** (*float*) –The drop out rate for attention layer. Default 0.
- **drop_path_rate** (*float*) –stochastic depth rate. Default 0.
- **act_cfg** (*dict*) –The activation config for FFNs. Defaults to GELU.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default layer normalization
- **first_stride** (*int*) –The stride of the conv2d layer. We use a conv2d layer and a unfold layer to implement image to pixel embedding.
- **num_fcs** (*int*) –The number of fully-connected layers for FFNs. Default 2
- **init_cfg** (*dict, optional*) –Initialization config dict

57.7.38 mmcls.models.VAN

class mmcls.models.**VAN**(*arch='tiny'*, *patch_sizes=[7, 3, 3, 3]*, *in_channels=3*, *drop_rate=0.0*, *drop_path_rate=0.0*, *out_indices=(3,)*, *frozen_stages=-1*, *norm_eval=False*, *norm_cfg={'type': 'LN'}*, *block_cfgs={}*, *init_cfg=None*)

Visual Attention Network.

A PyTorch implement of : Visual Attention Network

Inspiration from https://github.com/Visual-Attention-Network/VAN-Classification

参数

- **arch** (*str | dict*) –Visual Attention Network architecture. If use string, choose from 'b0', 'b1', b2', b3' and etc., if use dict, it should have below keys:
  - **embed_dims** (List[int]): The dimensions of embedding.
  - **depths** (List[int]): The number of blocks in each stage.
  - **ffn_ratios** (List[int]): The number of expansion ratio of feedforward network hidden layer channels.

  Defaults to 'tiny'.
- **patch_sizes** (*List[int | tuple]*) –The patch size in patch embeddings. Defaults to [7, 3, 3, 3].
- **in_channels** (*int*) –The num of input channels. Defaults to 3.
- **drop_rate** (*float*) –Dropout rate after embedding. Defaults to 0.
- **drop_path_rate** (*float*) –Stochastic depth rate. Defaults to 0.1.
- **out_indices** (*Sequence[int]*) –Output from which stages. Default: `(3, )`.
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Defaults to -1.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Defaults to False.
- **norm_cfg** (*dict*) –Config dict for normalization layer for all output features. Defaults to `dict(type='LN')`
- **block_cfgs** (*Sequence[dict] | dict*) –The extra config of each block. Defaults to empty dicts.
- **init_cfg** (*dict, optional*) –The Config for initialization. Defaults to None.

实际案例

```
>>> from mmcls.models import VAN
>>> import torch
>>> model = VAN(arch='b0')
>>> inputs = torch.rand(1, 3, 224, 224)
>>> outputs = model(inputs)
>>> for out in outputs:
>>>     print(out.size())
(1, 256, 7, 7)
```

57.7.39 mmcls.models.VGG

class mmcls.models.**VGG**(*depth*, *num_classes=- 1*, *num_stages=5*, *dilations=(1, 1, 1, 1, 1)*, *out_indices=None*, *frozen_stages=- 1*, *conv_cfg=None*, *norm_cfg=None*, *act_cfg={'type': 'ReLU'}*, *norm_eval=False*, *ceil_mode=False*, *with_last_pool=True*, *init_cfg=[{'type': 'Kaiming', 'layer': ['Conv2d']}, {'type': 'Constant', 'val': 1.0, 'layer': ['_BatchNorm']}, {'type': 'Normal', 'std': 0.01, 'layer': ['Linear']}]*)

VGG backbone.

参数

- **depth** (*int*) –Depth of vgg, from {11, 13, 16, 19}.
- **with_norm** (*bool*) –Use BatchNorm or not.
- **num_classes** (*int*) –number of classes for classification.
- **num_stages** (*int*) –VGG stages, normally 5.
- **dilations** (*Sequence[int]*) –Dilation of each stage.
- **out_indices** (*Sequence[int], optional*) –Output from which stages. When it is None, the default behavior depends on whether num_classes is specified. If num_classes <= 0, the default value is (4,), output the last feature map before classifier. If num_classes > 0, the default value is (5,), output the classification score. Default: None.
- **frozen_stages** (*int*) –Stages to be frozen (all param fixed). -1 means not freezing any parameters.
- **norm_eval** (*bool*) –Whether to set norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: False.
- **ceil_mode** (*bool*) –Whether to use ceil_mode of MaxPool. Default: False.
- **with_last_pool** (*bool*) –Whether to keep the last pooling before classifier. Default: True.

57.7.40 mmcls.models.VisionTransformer

class mmcls.models.**VisionTransformer**(*arch='base'*, *img_size=224*, *patch_size=16*, *in_channels=3*, *out_indices=- 1*, *drop_rate=0.0*, *drop_path_rate=0.0*, *qkv_bias=True*, *norm_cfg={'eps': 1e-06, 'type': 'LN'}*, *final_norm=True*, *with_cls_token=True*, *output_cls_token=True*, *interpolate_mode='bicubic'*, *patch_cfg={}*, *layer_cfgs={}*, *init_cfg=None*)

Vision Transformer.

A PyTorch implement of : An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

参数

- **arch** (*str | dict*) –Vision Transformer architecture. If use string, choose from 'small', 'base', 'large', 'deit-tiny', 'deit-small' and 'deit-base'. If use dict, it should have below keys:
  - **embed_dims** (int): The dimensions of embedding.
  - **num_layers** (int): The number of transformer encoder layers.
  - **num_heads** (int): The number of heads in attention modules.
  - **feedforward_channels** (int): The hidden dimensions in feedforward modules.

  Defaults to 'base'.
- **img_size** (*int | tuple*) –The expected input image shape. Because we support dynamic input shape, just set the argument to the most common input image shape. Defaults to 224.
- **patch_size** (*int | tuple*) –The patch size in patch embedding. Defaults to 16.
- **in_channels** (*int*) –The num of input channels. Defaults to 3.
- **out_indices** (*Sequence | int*) –Output from which stages. Defaults to -1, means the last stage.
- **drop_rate** (*float*) –Probability of an element to be zeroed. Defaults to 0.
- **drop_path_rate** (*float*) –stochastic depth rate. Defaults to 0.
- **qkv_bias** (*bool*) –Whether to add bias for qkv in attention modules. Defaults to True.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Defaults to `dict(type='LN')`.
- **final_norm** (*bool*) –Whether to add a additional layer to normalize final feature map. Defaults to True.

- **with_cls_token** (*bool*) –Whether concatenating class token into image tokens as transformer input. Defaults to True.
- **output_cls_token** (*bool*) –Whether output the cls_token. If set True, `with_cls_token` must be True. Defaults to True.
- **interpolate_mode** (*str*) –Select the interpolate mode for position embeding vector resize. Defaults to "bicubic".
- **patch_cfg** (*dict*) –Configs of patch embeding. Defaults to an empty dict.
- **layer_cfgs** (*Sequence | dict*) –Configs of each transformer layer in encoder. Defaults to an empty dict.
- **init_cfg** (*dict, optional*) –Initialization config dict. Defaults to None.

57.7.41 mmcls.models.EfficientFormer

class mmcls.models.**EfficientFormer**(*arch='l1'*, *in_channels=3*, *pool_size=3*, *mlp_ratios=4*, *reshape_last_feat=False*, *out_indices=- 1*, *frozen_stages=- 1*, *act_cfg={'type': 'GELU'}*, *drop_rate=0.0*, *drop_path_rate=0.0*, *use_layer_scale=True*, *init_cfg=None*)

EfficientFormer.

A PyTorch implementation of EfficientFormer introduced by: EfficientFormer: Vision Transformers at MobileNet Speed

Modified from the *official repo <https://github.com/snap-research/EfficientFormer>*.

参数

- **arch** (*str | dict*) –The model's architecture. If string, it should be one of architecture in `EfficientFormer.arch_settings`. And if dict,

  it should include the following 4 keys:

  - layers (list[int]): Number of blocks at each stage.
  - embed_dims (list[int]): The number of channels at each stage.
  - downsamples (list[int]): Has downsample or not in the four stages.
  - vit_num (int): The num of vit blocks in the last stage.

  Defaults to 'l1'.
- **in_channels** (*int*) –The num of input channels. Defaults to 3.
- **pool_size** (*int*) –The pooling size of `Meta4D` blocks. Defaults to 3.

- **mlp_ratios** (*int*) –The dimension ratio of multi-head attention mechanism in `Meta4D` blocks. Defaults to 3.
- **reshape_last_feat** (*bool*) –Whether to reshape the feature map from (B, N, C) to (B, C, H, W) in the last stage, when the `vit-num`

  in `arch` is not 0. Defaults to False. Usually set to True in downstream tasks.
- **out_indices** (*Sequence[int]*) –Output from which stages. Defaults to -1.
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Defaults to -1.
- **act_cfg** (*dict*) –The config dict for activation between pointwise convolution. Defaults to `dict(type='GELU')`.
- **drop_rate** (*float*) –Dropout rate. Defaults to 0.
- **drop_path_rate** (*float*) –Stochastic depth rate. Defaults to 0.
- **use_layer_scale** (*bool*) –Whether to use use_layer_scale in MetaFormer block. Defaults to True.
- **init_cfg** (*dict, optional*) –Initialization config dict. Defaults to None.
- **Example** –

```
>>> from mmcls.models import EfficientFormer
>>> import torch
>>> inputs = torch.rand((1, 3, 224, 224))
>>> # build EfficientFormer backbone for classification task
>>> model = EfficientFormer(arch="l1")
>>> model.eval()
>>> level_outputs = model(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 448, 49)
>>> # build EfficientFormer backbone for downstream task
>>> model = EfficientFormer(
>>>    arch="l3",
>>>    out_indices=(0, 1, 2, 3),
>>>    reshape_last_feat=True)
>>> model.eval()
>>> level_outputs = model(inputs)
>>> for level_out in level_outputs:
...     print(tuple(level_out.shape))
(1, 64, 56, 56)
(1, 128, 28, 28)
```

(下页继续)

(续上页)

```
(1, 320, 14, 14)
(1, 512, 7, 7)
```

57.7.42 mmcls.models.HorNet

class mmcls.models.**HorNet**(*arch='tiny'*, *in_channels=3*, *drop_path_rate=0.0*, *scale=0.3333333333333333*, *use_layer_scale=True*, *out_indices=(3,)*, *frozen_stages=-1*, *with_cp=False*, *gap_before_final_norm=True*, *init_cfg=None*)

A PyTorch impl of : *HorNet: Efficient High-Order Spatial Interactions with Recursive Gated Convolutions*

Inspiration from https://github.com/raoyongming/HorNet

参数

- **arch** (*str | dict*) –HorNet architecture. If use string, choose from 'tiny', 'small', 'base' and 'large'. If use dict, it should have below keys: - **base_dim** (int): The base dimensions of embedding. - **depths** (List[int]): The number of blocks in each stage. - **orders** (List[int]): The number of order of gnConv in each

  stage.

  - **dw_cfg** (List[dict]): The Config for dw conv.

  Defaults to 'tiny'.
- **in_channels** (*int*) –Number of input image channels. Defaults to 3.
- **drop_path_rate** (*float*) –Stochastic depth rate. Defaults to 0.
- **scale** (*float*) –Scaling parameter of gflayer outputs. Defaults to 1/3.
- **use_layer_scale** (*bool*) –Whether to use use_layer_scale in HorNet block. Defaults to True.
- **out_indices** (*Sequence[int]*) –Output from which stages. Default: `(3, )`.
- **frozen_stages** (*int*) –Stages to be frozen (stop grad and set eval mode). -1 means not freezing any parameters. Defaults to -1.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Defaults to False.
- **gap_before_final_norm** (*bool*) –Whether to globally average the feature map before the final norm layer. In the official repo, it's only used in classification task. Defaults to True.
- **init_cfg** (*dict, optional*) –The Config for initialization. Defaults to None.

57.8 Necks

GlobalAveragePooling	Global Average Pooling neck.
GeneralizedMeanPooling	Generalized Mean Pooling neck.
HRFuseScales	Fuse feature map of multiple scales in HRNet.

57.8.1 mmcls.models.GlobalAveragePooling

class mmcls.models.**GlobalAveragePooling**(*dim=2*)

Global Average Pooling neck.

Note that we use *view* to remove extra channel after pooling. We do not use *squeeze* as it will also remove the batch dimension when the tensor has a batch dimension of size 1, which can lead to unexpected errors.

参数 **dim** (*int*) –Dimensions of each sample channel, can be one of {1, 2, 3}. Default: 2

57.8.2 mmcls.models.GeneralizedMeanPooling

class mmcls.models.**GeneralizedMeanPooling**(*p=3.0*, *eps=1e-06*, *clamp=True*)

Generalized Mean Pooling neck.

Note that we use *view* to remove extra channel after pooling. We do not use *squeeze* as it will also remove the batch dimension when the tensor has a batch dimension of size 1, which can lead to unexpected errors.

参数

- **p** (*float*) –Parameter value. Default: 3.
- **eps** (*float*) –epsilon. Default: 1e-6
- **clamp** (*bool*) –Use clamp before pooling. Default: True

57.8.3 mmcls.models.HRFuseScales

class mmcls.models.**HRFuseScales**(*in_channels*, *out_channels=2048*, *norm_cfg={'momentum': 0.1, 'type': 'BN'}*, *init_cfg={'layer': 'Linear', 'std': 0.01, 'type': 'Normal'}*)

Fuse feature map of multiple scales in HRNet.

参数

- **in_channels** (*list[int]*) –The input channels of all scales.
- **out_channels** (*int*) –The channels of fused feature map. Defaults to 2048.

- **norm_cfg** (*dict*) –dictionary to construct norm layers. Defaults to `dict(type='BN', momentum=0.1)`.
- **init_cfg** (*dict | list[dict], optional*) –Initialization config dict. Defaults to `dict(type='Normal', layer='Linear', std=0.01))`.

57.9 Heads

ClsHead	classification head.
LinearClsHead	Linear classifier head.
StackedLinearClsHead	Classifier head with several hidden fc layer and a output fc layer.
MultiLabelClsHead	Classification head for multilabel task.
MultiLabelLinearClsHead	Linear classification head for multilabel task.
VisionTransformerClsHead	Vision Transformer classifier head.
DeiTClsHead	Distilled Vision Transformer classifier head.
ConformerHead	Linear classifier head.

57.9.1 mmcls.models.ClsHead

class mmcls.models.**ClsHead**(*loss={'loss_weight': 1.0, 'type': 'CrossEntropyLoss'}*, *topk=(1,)*, *cal_acc=False*, *init_cfg=None*)

classification head.

参数

- **loss** (*dict*) –Config of classification loss.
- **topk** (*int | tuple*) –Top-k accuracy.
- **cal_acc** (*bool*) –Whether to calculate accuracy during training. If you use Mixup/CutMix or something like that during training, it is not reasonable to calculate accuracy. Defaults to False.

57.9.2 mmcls.models.LinearClsHead

class mmcls.models.**LinearClsHead**(*num_classes*, *in_channels*, *init_cfg={'layer': 'Linear', 'std': 0.01, 'type': 'Normal'}*, **args*, ***kwargs*)

Linear classifier head.

参数

- **num_classes** (*int*) –Number of categories excluding the background category.

- **in_channels** (*int*) –Number of channels in the input feature map.
- **init_cfg** (*dict | optional*) –The extra init config of layers. Defaults to use dict(type=’Normal’, layer=’Linear’, std=0.01).

57.9.3 mmcls.models.StackedLinearClsHead

class mmcls.models.**StackedLinearClsHead**(*num_classes: int*, *in_channels: int*, *mid_channels: Sequence*, *dropout_rate: float = 0.0*, *norm_cfg: Optional[Dict] = None*, *act_cfg: Dict = {'type': 'ReLU'}*, ***kwargs*)

Classifier head with several hidden fc layer and a output fc layer.

参数

- **num_classes** (*int*) –Number of categories.
- **in_channels** (*int*) –Number of channels in the input feature map.
- **mid_channels** (*Sequence*) –Number of channels in the hidden fc layers.
- **dropout_rate** (*float*) –Dropout rate after each hidden fc layer, except the last layer. Defaults to 0.
- **norm_cfg** (*dict, optional*) –Config dict of normalization layer after each hidden fc layer, except the last layer. Defaults to None.
- **act_cfg** (*dict, optional*) –Config dict of activation function after each hidden layer, except the last layer. Defaults to use “ReLU”.

57.9.4 mmcls.models.MultiLabelClsHead

class mmcls.models.**MultiLabelClsHead**(*loss={'loss_weight': 1.0, 'reduction': 'mean', 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, *init_cfg=None*)

Classification head for multilabel task.

参数 **loss** (*dict*) –Config of classification loss.

57.9.5 mmcls.models.MultiLabelLinearClsHead

class mmcls.models.**MultiLabelLinearClsHead**(*num_classes*, *in_channels*, *loss={'loss_weight': 1.0, 'reduction': 'mean', 'type': 'CrossEntropyLoss', 'use_sigmoid': True}*, *init_cfg={'layer': 'Linear', 'std': 0.01, 'type': 'Normal'}*)

Linear classification head for multilabel task.

参数

- **num_classes** (*int*) –Number of categories.
- **in_channels** (*int*) –Number of channels in the input feature map.
- **loss** (*dict*) –Config of classification loss.
- **init_cfg** (*dict | optional*) –The extra init config of layers. Defaults to use dict(type=’Normal’, layer=’Linear’, std=0.01).

57.9.6 mmcls.models.VisionTransformerClsHead

class mmcls.models.**VisionTransformerClsHead**(*num_classes*, *in_channels*, *hidden_dim=None*, *act_cfg={'type': 'Tanh'}*, *init_cfg={'layer': 'Linear', 'type': 'Constant', 'val': 0}*, **args*, ***kwargs*)

Vision Transformer classifier head.

参数

- **num_classes** (*int*) –Number of categories excluding the background category.
- **in_channels** (*int*) –Number of channels in the input feature map.
- **hidden_dim** (*int*) –Number of the dimensions for hidden layer. Defaults to None, which means no extra hidden layer.
- **act_cfg** (*dict*) –The activation config. Only available during pre-training. Defaults to `dict(type='Tanh')`.
- **init_cfg** (*dict*) –The extra initialization configs. Defaults to `dict(type='Constant', layer='Linear', val=0)`.

57.9.7 mmcls.models.DeiTClsHead

class mmcls.models.**DeiTClsHead**(**args*, ***kwargs*)

Distilled Vision Transformer classifier head.

Comparing with the *VisionTransformerClsHead*, this head adds an extra linear layer to handle the dist token. The final classification score is the average of both linear transformation results of `cls_token` and `dist_token`.

参数

- **num_classes** (*int*) –Number of categories excluding the background category.
- **in_channels** (*int*) –Number of channels in the input feature map.
- **hidden_dim** (*int*) –Number of the dimensions for hidden layer. Defaults to None, which means no extra hidden layer.

- **act_cfg** (*dict*) –The activation config. Only available during pre-training. Defaults to `dict(type='Tanh')`.
- **init_cfg** (*dict*) –The extra initialization configs. Defaults to `dict(type='Constant', layer='Linear', val=0)`.

57.9.8 mmcls.models.ConformerHead

class mmcls.models.**ConformerHead**(*num_classes*, *in_channels*, *init_cfg={'layer': 'Linear', 'std': 0.01, 'type': 'Normal'}*, **args*, ***kwargs*)

Linear classifier head.

参数

- **num_classes** (*int*) –Number of categories excluding the background category.
- **in_channels** (*int*) –Number of channels in the input feature map.
- **init_cfg** (*dict | optional*) –The extra init config of layers. Defaults to use `dict(type='Normal', layer='Linear', std=0.01)`.

57.10 Losses

Accuracy	

AsymmetricLoss	asymmetric loss.
CrossEntropyLoss	Cross entropy loss.
LabelSmoothLoss	Initializer for the label smoothed cross entropy loss.
FocalLoss	Focal loss.
SeesawLoss	Implementation of seesaw loss.

57.10.1 mmcls.models.Accuracy

class mmcls.models.**Accuracy**(*topk=(1,)*)

57.10.2 mmcls.models.AsymmetricLoss

class mmcls.models.**AsymmetricLoss**(*gamma_pos=0.0*, *gamma_neg=4.0*, *clip=0.05*, *reduction='mean'*, *loss_weight=1.0*, *use_sigmoid=True*, *eps=1e-08*)

asymmetric loss.

参数

- **gamma_pos** (*float*) –positive focusing parameter. Defaults to 0.0.
- **gamma_neg** (*float*) –Negative focusing parameter. We usually set gamma_neg > gamma_pos. Defaults to 4.0.
- **clip** (*float, optional*) –Probability margin. Defaults to 0.05.
- **reduction** (*str*) –The method used to reduce the loss into a scalar.
- **loss_weight** (*float*) –Weight of loss. Defaults to 1.0.
- **use_sigmoid** (*bool*) –Whether the prediction uses sigmoid instead of softmax. Defaults to True.
- **eps** (*float*) –The minimum value of the argument of logarithm. Defaults to 1e-8.

57.10.3 mmcls.models.CrossEntropyLoss

class mmcls.models.**CrossEntropyLoss**(*use_sigmoid=False*, *use_soft=False*, *reduction='mean'*, *loss_weight=1.0*, *class_weight=None*, *pos_weight=None*)

Cross entropy loss.

参数

- **use_sigmoid** (*bool*) –Whether the prediction uses sigmoid of softmax. Defaults to False.
- **use_soft** (*bool*) –Whether to use the soft version of CrossEntropyLoss. Defaults to False.
- **reduction** (*str*) –The method used to reduce the loss. Options are "none", "mean" and "sum". Defaults to 'mean'.
- **loss_weight** (*float*) –Weight of the loss. Defaults to 1.0.
- **class_weight** (*List[float], optional*) –The weight for each class with shape (C), C is the number of classes. Default None.
- **pos_weight** (*List[float], optional*) –The positive weight for each class with shape (C), C is the number of classes. Only enabled in BCE loss when `use_sigmoid` is True. Default None.

57.10.4 mmcls.models.LabelSmoothLoss

class mmcls.models.**LabelSmoothLoss**(*label_smooth_val*, *num_classes=None*, *mode='original'*, *reduction='mean'*, *loss_weight=1.0*)

Initializer for the label smoothed cross entropy loss.

Refers to Rethinking the Inception Architecture for Computer Vision

This decreases gap between output scores and encourages generalization. Labels provided to forward can be one-hot like vectors (NxC) or class indices (Nx1). And this accepts linear combination of one-hot like labels from mixup or cutmix except multi-label task.

参数

- **label_smooth_val** (*float*) –The degree of label smoothing.
- **num_classes** (*int, optional*) –Number of classes. Defaults to None.
- **mode** (*str*) –Refers to notes, Options are ‘original’, ‘classy_vision’, ‘multi_label’. Defaults to ‘original’
- **reduction** (*str*) –The method used to reduce the loss. Options are “none”, “mean” and “sum”. Defaults to ‘mean’.
- **loss_weight** (*float*) –Weight of the loss. Defaults to 1.0.

提示

if the mode is “original”, this will use the same label smooth method as the original paper as:

$$(1-\epsilon)\delta_{k,y}+\frac{\epsilon}{K}$$

where epsilon is the *label_smooth_val*, K is the num_classes and delta(k,y) is Dirac delta, which equals 1 for k=y and 0 otherwise.

if the mode is “classy_vision”, this will use the same label smooth method as the facebookresearch/ClassyVision repo as:

$$\frac{\delta_{k,y}+\epsilon/K}{1+\epsilon}$$

if the mode is “multi_label”, this will accept labels from multi-label task and smoothing them as:

$$(1-2\epsilon)\delta_{k,y}+\epsilon$$

57.10.5 mmcls.models.FocalLoss

class mmcls.models.**FocalLoss**(*gamma=2.0*, *alpha=0.25*, *reduction='mean'*, *loss_weight=1.0*)

Focal loss.

参数

- **gamma** (*float*) –Focusing parameter in focal loss. Defaults to 2.0.
- **alpha** (*float*) –The parameter in balanced form of focal loss. Defaults to 0.25.
- **reduction** (*str*) –The method used to reduce the loss into a scalar. Options are "none" and "mean". Defaults to 'mean'.
- **loss_weight** (*float*) –Weight of loss. Defaults to 1.0.

57.10.6 mmcls.models.SeesawLoss

class mmcls.models.**SeesawLoss**(*use_sigmoid=False*, *p=0.8*, *q=2.0*, *num_classes=1000*, *eps=0.01*, *reduction='mean'*, *loss_weight=1.0*)

Implementation of seesaw loss.

Refers to Seesaw Loss for Long-Tailed Instance Segmentation (CVPR 2021)

参数

- **use_sigmoid** (*bool*) –Whether the prediction uses sigmoid of softmax. Only False is supported. Defaults to False.
- **p** (*float*) –The `p` in the mitigation factor. Defaults to 0.8.
- **q** (*float*) –The `q` in the compenstation factor. Defaults to 2.0.
- **num_classes** (*int*) –The number of classes. Default to 1000 for the ImageNet dataset.
- **eps** (*float*) –The minimal value of divisor to smooth the computation of compensation factor, default to 1e-2.
- **reduction** (*str*) –The method that reduces the loss to a scalar. Options are "none", "mean" and "sum". Default to "mean".
- **loss_weight** (*float*) –The weight of the loss. Defaults to 1.0

CHAPTER 58

mmcls.models.utils

This package includes some helper functions and common components used in various networks.

mmcls.models.utils

58.1 Common Components

`InvertedResidual`	Inverted Residual Block.
`SELayer`	Squeeze-and-Excitation Module.
`ShiftWindowMSA`	Shift Window Multihead Self-Attention Module.
`MultiheadAttention`	Multi-head Attention Module.
`ConditionalPositionEncoding`	The Conditional Position Encoding (CPE) module.

58.1.1 mmcls.models.utils.InvertedResidual

class mmcls.models.utils.**InvertedResidual**(*in_channels*, *out_channels*, *mid_channels*, *kernel_size=3*, *stride=1*, *se_cfg=None*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *drop_path_rate=0.0*, *with_cp=False*, *init_cfg=None*)

Inverted Residual Block.

参数

- **in_channels** (*int*) –The input channels of this module.
- **out_channels** (*int*) –The output channels of this module.
- **mid_channels** (*int*) –The input channels of the depthwise convolution.
- **kernel_size** (*int*) –The kernel size of the depthwise convolution. Defaults to 3.
- **stride** (*int*) –The stride of the depthwise convolution. Defaults to 1.
- **se_cfg** (*dict, optional*) –Config dict for se layer. Defaults to None, which means no se layer.
- **conv_cfg** (*dict*) –Config dict for convolution layer. Defaults to None, which means using conv2d.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Defaults to `dict(type='BN')`.
- **act_cfg** (*dict*) –Config dict for activation layer. Defaults to `dict(type='ReLU')`.
- **drop_path_rate** (*float*) –stochastic depth rate. Defaults to 0.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Defaults to False.
- **init_cfg** (*dict | list[dict], optional*) –Initialization config dict.

58.1.2 mmcls.models.utils.SELayer

class mmcls.models.utils.**SELayer**(*channels*, *squeeze_channels=None*, *ratio=16*, *divisor=8*, *bias='auto'*, *conv_cfg=None*, *act_cfg=({'type': 'ReLU'}, {'type': 'Sigmoid'})*, *return_weight=False*, *init_cfg=None*)

Squeeze-and-Excitation Module.

参数

- **channels** (*int*) –The input (and output) channels of the SE layer.

- **squeeze_channels** (*None or int*) –The intermediate channel number of SElayer. Default: None, means the value of `squeeze_channels` is `make_divisible(channels // ratio, divisor)`.
- **ratio** (*int*) –Squeeze ratio in SELayer, the intermediate channel will be `make_divisible(channels // ratio, divisor)`. Only used when `squeeze_channels` is None. Default: 16.
- **divisor** (*int*) –The divisor to true divide the channel number. Only used when `squeeze_channels` is None. Default: 8.
- **conv_cfg** (*None or dict*) –Config dict for convolution layer. Default: None, which means using conv2d.
- **return_weight** (*bool*) –Whether to return the weight. Default: False.
- **act_cfg** (*dict or Sequence[dict]*) –Config dict for activation layer. If act_cfg is a dict, two activation layers will be configurated by this dict. If act_cfg is a sequence of dicts, the first activation layer will be configurated by the first dict and the second activation layer will be configurated by the second dict. Default: (dict(type='ReLU'), dict(type='Sigmoid'))

58.1.3 mmcls.models.utils.ShiftWindowMSA

class mmcls.models.utils.**ShiftWindowMSA**(*embed_dims*, *num_heads*, *window_size*, *shift_size=0*, *qkv_bias=True*, *qk_scale=None*, *attn_drop=0*, *proj_drop=0*, *dropout_layer={'drop_prob': 0.0, 'type': 'DropPath'}*, *pad_small_map=False*, *input_resolution=None*, *auto_pad=None*, *window_msa=<class 'mmcls.models.utils.attention.WindowMSA'>*, *msa_cfg={}*, *init_cfg=None*)

Shift Window Multihead Self-Attention Module.

参数

- **embed_dims** (*int*) –Number of input channels.
- **num_heads** (*int*) –Number of attention heads.
- **window_size** (*int*) –The height and width of the window.
- **shift_size** (*int, optional*) –The shift step of each window towards right-bottom. If zero, act as regular window-msa. Defaults to 0.
- **qkv_bias** (*bool, optional*) –If True, add a learnable bias to q, k, v. Defaults to True

- **qk_scale** (*float | None, optional*) –Override default qk scale of head_dim ** -0.5 if set. Defaults to None.
- **attn_drop** (*float, optional*) –Dropout ratio of attention weight. Defaults to 0.0.
- **proj_drop** (*float, optional*) –Dropout ratio of output. Defaults to 0.
- **dropout_layer** (*dict, optional*) –The dropout_layer used before output. Defaults to dict(type=’DropPath’, drop_prob=0.).
- **pad_small_map** (*bool*) –If True, pad the small feature map to the window size, which is common used in detection and segmentation. If False, avoid shifting window and shrink the window size to the size of feature map, which is common used in classification. Defaults to False.
- **version** (*str, optional*) –Version of implementation of Swin Transformers. Defaults to *v1*.
- **init_cfg** (*dict, optional*) –The extra config for initialization. Defaults to None.

58.1.4 mmcls.models.utils.MultiheadAttention

class mmcls.models.utils.**MultiheadAttention**(*embed_dims*, *num_heads*, *input_dims=None*, *attn_drop=0.0*, *proj_drop=0.0*, *dropout_layer={'drop_prob': 0.0, 'type': 'Dropout'}*, *qkv_bias=True*, *qk_scale=None*, *proj_bias=True*, *v_shortcut=False*, *init_cfg=None*)

Multi-head Attention Module.

This module implements multi-head attention that supports different input dims and embed dims. And it also supports a shortcut from `value`, which is useful if input dims is not the same with embed dims.

参数

- **embed_dims** (*int*) –The embedding dimension.
- **num_heads** (*int*) –Parallel attention heads.
- **input_dims** (*int, optional*) –The input dimension, and if None, use `embed_dims`. Defaults to None.
- **attn_drop** (*float*) –Dropout rate of the dropout layer after the attention calculation of query and key. Defaults to 0.
- **proj_drop** (*float*) –Dropout rate of the dropout layer after the output projection. Defaults to 0.
- **dropout_layer** (*dict*) –The dropout config before adding the shortcut. Defaults to `dict(type='Dropout', drop_prob=0.)`.

- **qkv_bias** (*bool*) –If True, add a learnable bias to q, k, v. Defaults to True.
- **qk_scale** (*float, optional*) –Override default qk scale of `head_dim ** -0.5` if set. Defaults to None.
- **proj_bias** (*bool*) –Defaults to True.
- **v_shortcut** (*bool*) –Add a shortcut from value to output. It's usually used if `input_dims` is different from `embed_dims`. Defaults to False.
- **init_cfg** (*dict, optional*) –The Config for initialization. Defaults to None.

58.1.5 mmcls.models.utils.ConditionalPositionEncoding

class mmcls.models.utils.**ConditionalPositionEncoding**(*in_channels*, *embed_dims=768*, *stride=1*, *init_cfg=None*)

The Conditional Position Encoding (CPE) module.

The CPE is the implementation of 'Conditional Positional Encodings for Vision Transformers <https://arxiv.org/abs/2102.10882>'_.

参数

- **in_channels** (*int*) –Number of input channels.
- **embed_dims** (*int*) –The feature dimension. Default: 768.
- **stride** (*int*) –Stride of conv layer. Default: 1.

58.2 Helper Functions

58.2.1 channel_shuffle

mmcls.models.utils.**channel_shuffle**(*x*, *groups*)

Channel Shuffle operation.

This function enables cross-group information flow for multiple groups convolution layers.

参数

- **x** (*Tensor*) –The input tensor.
- **groups** (*int*) –The number of groups to divide the input tensor in the channel dimension.

返回 The output tensor after channel shuffle operation.

返回类型 Tensor

58.2.2 make_divisible

mmcls.models.utils.**make_divisible**(*value*, *divisor*, *min_value=None*, *min_ratio=0.9*)

Make divisible function.

This function rounds the channel number down to the nearest value that can be divisible by the divisor.

参数

- **value** (*int*) –The original channel number.
- **divisor** (*int*) –The divisor to fully divide the channel number.
- **min_value** (*int, optional*) –The minimum value of the output channel. Default: None, means that the minimum value equal to the divisor.
- **min_ratio** (*float*) –The minimum ratio of the rounded channel number to the original channel number. Default: 0.9.

返回 The modified output channel number

返回类型 int

58.2.3 to_ntuple

mmcls.models.utils.**to_ntuple**(*n*)

A *to_tuple* function generator.

It returns a function, this function will repeat the input to a tuple of length `n` if the input is not an Iterable object, otherwise, return the input directly.

参数 **n** (*int*) –The number of the target length.

mmcls.models.utils.**to_2tuple**(*x*)

mmcls.models.utils.**to_3tuple**(*x*)

mmcls.models.utils.**to_4tuple**(*x*)

58.2.4 is_tracing

mmcls.models.utils.**is_tracing**() → bool

Determine whether the model is called during the tracing of code with `torch.jit.trace`.

CHAPTER 59

mmcls.datasets

The `datasets` package contains several usual datasets for image classification tasks and some dataset wrappers.

59.1 Custom Dataset

class mmcls.datasets.**CustomDataset**(*data_prefix: str*, *pipeline: Sequence = ()*, *classes: Optional[Union[str, Sequence[str]]] = None*, *ann_file: Optional[str] = None*, *extensions: Sequence[str] = ('.jpg', '.jpeg', '.png', '.ppm', '.bmp', '.pgm', '.tif')*, *test_mode: bool = False*, *file_client_args: Optional[dict] = None*)

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
```

(下页继续)

(续上页)

```
    ├── 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`.

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

- **data_prefix** (*str*) –The path of data directory.
- **pipeline** (*Sequence[dict]*) –A list of dict, where each element represents a operation defined in `mmcls.datasets.pipelines`. Defaults to an empty tuple.
- **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 `cls.CLASSES` or the names of sub folders (If use the second way to arrange samples).

  Defaults to None.
- **ann_file** (*str, optional*) –The annotation file. If is string, read samples paths from the ann_file. If is None, find samples in `data_prefix`. Defaults to None.

- **extensions** (*Sequence[str]*) –A sequence of allowed extensions. Defaults to ('.jpg', '.jpeg', '.png', '.ppm', '.bmp', '.pgm', '.tif').
- **test_mode** (*bool*) –In train mode or test mode. It's only a mark and won't be used in this class. Defaults to False.
- **file_client_args** (*dict, optional*) –Arguments to instantiate a FileClient. See mmcv.fileio.FileClient for details. If None, automatically inference from the specified path. Defaults to None.

59.2 ImageNet

class mmcls.datasets.**ImageNet**(*data_prefix: str*, *pipeline: Sequence = ()*, *classes: Optional[Union[str, Sequence[str]]] = None*, *ann_file: Optional[str] = None*, *test_mode: bool = False*, *file_client_args: Optional[dict] = None*)

ImageNet Dataset.

The dataset supports two kinds of annotation format. More details can be found in *CustomDataset*.

参数

- **data_prefix** (*str*) –The path of data directory.
- **pipeline** (*Sequence[dict]*) –A list of dict, where each element represents a operation defined in mmcls.datasets.pipelines. Defaults to an empty tuple.
- **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 the default ImageNet-1k classes names.

  Defaults to None.
- **ann_file** (*str, optional*) –The annotation file. If is string, read samples paths from the ann_file. If is None, find samples in data_prefix. Defaults to None.
- **extensions** (*Sequence[str]*) –A sequence of allowed extensions. Defaults to ('.jpg', '.jpeg', '.png', '.ppm', '.bmp', '.pgm', '.tif').
- **test_mode** (*bool*) –In train mode or test mode. It's only a mark and won't be used in this class. Defaults to False.
- **file_client_args** (*dict, optional*) –Arguments to instantiate a FileClient. See mmcv.fileio.FileClient for details. If None, automatically inference from the specified path. Defaults to None.

class mmcls.datasets.**ImageNet21k**(*data_prefix: str*, *pipeline: Sequence = ()*, *classes: Optional[Union[str, Sequence[str]]] = None*, *ann_file: Optional[str] = None*, *serialize_data: bool = True*, *multi_label: bool = False*, *recursion_subdir: bool = True*, *test_mode=False*, *file_client_args: Optional[dict] = None*)

ImageNet21k Dataset.

Since the dataset ImageNet21k is extremely big, cantains 21k+ classes and 1.4B files. This class has improved the following points on the basis of the class `ImageNet`, in order to save memory, we enable the `serialize_data` optional by default. With this option, the annotation won't be stored in the list `data_infos`, but be serialized as an array.

参数

- **data_prefix** (*str*) –The path of data directory.
- **pipeline** (*Sequence[dict]*) –A list of dict, where each element represents a operation defined in `mmcls.datasets.pipelines`. Defaults to an empty tuple.
- **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, the object won't have category information. (Not recommended)

  Defaults to None.
- **ann_file** (*str, optional*) –The annotation file. If is string, read samples paths from the ann_file. If is None, find samples in `data_prefix`. Defaults to None.
- **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.
- **multi_label** (*bool*) –Not implement by now. Use multi label or not. Defaults to False.
- **recursion_subdir** (*bool*) –Deprecated, and the dataset will recursively get all images now.
- **test_mode** (*bool*) –In train mode or test mode. It's only a mark and won't be used in this class. Defaults to False.
- **file_client_args** (*dict, optional*) –Arguments to instantiate a FileClient. See `mmcv.fileio.FileClient` for details. If None, automatically inference from the specified path. Defaults to None.

59.3 CIFAR

class mmcls.datasets.**CIFAR10**(*data_prefix*, *pipeline*, *classes=None*, *ann_file=None*, *test_mode=False*)

CIFAR10 Dataset.

This implementation is modified from https://github.com/pytorch/vision/blob/master/torchvision/datasets/cifar.py

class mmcls.datasets.**CIFAR100**(*data_prefix*, *pipeline*, *classes=None*, *ann_file=None*, *test_mode=False*)

CIFAR100 Dataset.

59.4 MNIST

class mmcls.datasets.**MNIST**(*data_prefix*, *pipeline*, *classes=None*, *ann_file=None*, *test_mode=False*)

MNIST Dataset.

This implementation is modified from https://github.com/pytorch/vision/blob/master/torchvision/datasets/mnist.py

class mmcls.datasets.**FashionMNIST**(*data_prefix*, *pipeline*, *classes=None*, *ann_file=None*, *test_mode=False*)

Fashion-MNIST Dataset.

59.5 VOC

class mmcls.datasets.**VOC**(*difficult_as_postive=None*, ***kwargs*)

Pascal VOC Dataset.

参数

- **data_prefix** (*str*) –the prefix of data path
- **pipeline** (*list*) –a list of dict, where each element represents a operation defined in *mmcls.datasets.pipelines*
- **ann_file** (*str | None*) –the annotation file. When ann_file is str, the subclass is expected to read from the ann_file. When ann_file is None, the subclass is expected to read according to data_prefix
- **difficult_as_postive** (*Optional[bool]*) –Whether to map the difficult labels as positive. If it set to True, map difficult examples to positive ones(1), If it set to False, map difficult examples to negative ones(0). Defaults to None, the difficult labels will be set to '-1'.

59.6 StanfordCars Cars

class mmcls.datasets.**StanfordCars**(*data_prefix: str*, *test_mode: bool*, *ann_file: Optional[str] = None*, ***kwargs*)

Stanford Cars Dataset.

After downloading and decompression, the dataset directory structure is as follows.

Stanford Cars dataset directory:

```
Stanford Cars
├── cars_train
│   ├── 00001.jpg
│   ├── 00002.jpg
│   └── ...
├── cars_test
│   ├── 00001.jpg
│   ├── 00002.jpg
│   └── ...
└── devkit
    ├── cars_meta.mat
    ├── cars_train_annos.mat
    ├── cars_test_annos.mat
    ├── cars_test_annoswithlabels.mat
    ├── eval_train.m
    └── train_perfect_preds.txt
```

参数

- **data_prefix** (*str*) –the prefix of data path
- **test_mode** (*bool*) –`test_mode=True` means in test phase. It determines to use the training set or test set.
- **ann_file** (*str, optional*) –The annotation file. If is string, read samples paths from the ann_file. If is None, read samples path from cars_{train|test}_annos.mat file. Defaults to None.

59.7 Base classes

class mmcls.datasets.**BaseDataset**(*data_prefix*, *pipeline*, *classes=None*, *ann_file=None*, *test_mode=False*)

Base dataset.

参数

- **data_prefix** (*str*) –the prefix of data path
- **pipeline** (*list*) –a list of dict, where each element represents a operation defined in *mmcls.datasets.pipelines*
- **ann_file** (*str | None*) –the annotation file. When ann_file is str, the subclass is expected to read from the ann_file. When ann_file is None, the subclass is expected to read according to data_prefix
- **test_mode** (*bool*) –in train mode or test mode

class mmcls.datasets.**MultiLabelDataset**(*data_prefix*, *pipeline*, *classes=None*, *ann_file=None*, *test_mode=False*)

Multi-label Dataset.

59.8 Dataset Wrappers

class mmcls.datasets.**ConcatDataset**(*datasets*, *separate_eval=True*)

A wrapper of concatenated dataset.

Same as `torch.utils.data.dataset.ConcatDataset`, but add *get_cat_ids* function.

参数

- **datasets** (list[*BaseDataset*]) –A list of datasets.
- **separate_eval** (*bool*) –Whether to evaluate the results separately if it is used as validation dataset. Defaults to True.

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

A wrapper of repeated dataset.

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

参数

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

class mmcls.datasets.**ClassBalancedDataset**(*dataset*, *oversample_thr*)

A wrapper of repeated dataset with repeat factor.

Suitable for training on class imbalanced datasets like LVIS. Following the sampling strategy in this paper, in each epoch, an image may appear multiple times based on its "repeat factor".

The repeat factor for an image is a function of the frequency the rarest category labeled in that image. The "frequency of category c" in [0, 1] is defined by the fraction of images in the training set (without repeats) in which category c appears.

The dataset needs to implement `self.get_cat_ids()` to support ClassBalancedDataset.

The repeat factor is computed as followed.

1. For each category c, compute the fraction $f(c)$ of images that contain it.
2. For each category c, compute the category-level repeat factor.

$$r(c) = \max(1, \sqrt{\frac{t}{f(c)}})$$

where t is *oversample_thr*.

3. For each image I and its labels $L(I)$, compute the image-level repeat factor.

$$r(I) = \max_{c \in L(I)} r(c)$$

Each image repeats $\lceil r(I) \rceil$ times.

参数

- **dataset** (*BaseDataset*) –The dataset to be repeated.
- **oversample_thr** (*float*) –frequency threshold below which data is repeated. For categories with `f_c` >= `oversample_thr`, there is no oversampling. For categories with `f_c` < `oversample_thr`, the degree of oversampling following the square-root inverse frequency heuristic above.

CHAPTER 60

Data Transformations

In MMClassification, the data preparation and the dataset is decomposed. The datasets only define how to get samples' basic information from the file system. These basic information includes the ground-truth label and raw images data / the paths of images.

To prepare the inputs data, we need to do some transformations on these basic information. These transformations includes loading, preprocessing and formatting. And a series of data transformations makes up a data pipeline. Therefore, you can find the a `pipeline` argument in the configs of dataset, for example:

```
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='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'])
]
test_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='Resize', size=256),
    dict(type='CenterCrop', crop_size=224),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='ImageToTensor', keys=['img']),
```

(下页继续)

(续上页)

```
    dict(type='Collect', keys=['img'])
]

data = dict(
    train=dict(..., pipeline=train_pipeline),
    val=dict(..., pipeline=test_pipeline),
    test=dict(..., pipeline=test_pipeline),
)
```

Every item of a pipeline list is one of the following data transformations class. And if you want to add a custom data transformation class, the tutorial *Custom Data Pipelines* will help you.

mmcls.datasets.pipelines

- *Loading*
  - *LoadImageFromFile*
- *Preprocessing and Augmentation*
  - *CenterCrop*
  - *Lighting*
  - *Normalize*
  - *Pad*
  - *Resize*
  - *RandomCrop*
  - *RandomErasing*
  - *RandomFlip*
  - *RandomGrayscale*
  - *RandomResizedCrop*
  - *ColorJitter*
  - *Composed Augmentation*
- *Formatting*
  - *Collect*
  - *ImageToTensor*
  - *ToNumpy*

- *ToPIL*
- *ToTensor*
- *Transpose*

60.1 Loading

60.1.1 LoadImageFromFile

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

Load an image from file.

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

参数

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

60.2 Preprocessing and Augmentation

60.2.1 CenterCrop

class mmcls.datasets.pipelines.**CenterCrop**(*crop_size*, *efficientnet_style=False*, *crop_padding=32*, *interpolation='bilinear'*, *backend='cv2'*)

Center crop the image.

参数

- **crop_size** (*int | tuple*) –Expected size after cropping with the format of (h, w).
- **efficientnet_style** (*bool*) –Whether to use efficientnet style center crop. Defaults to False.