MMSelfSup

发布 *0.9.2*

MMSelfSup Authors

2022 年 07 月 28 日

开始你的第一步

中文文档在持续翻译中，敬请期待，同时我们也鼓励社区开发者们参与到翻译中来

CHAPTER 1

前提

在这一节中，我们展示了如何用 PyTorch 准备环境。

MMselfSup 可在 Linux 上运行 (Windows 和 macOS 平台不完全支持)。要求 Python 3.6+, CUDA 9.2+ 和 PyTorch 1.5+。

如果您对 PyTorch 很熟悉，或者已经安装了它，可以忽略这部分并转到下一节，不然你可以按照下列步骤进行准备。

Step 0. 从 官方网址 下载并安装 Miniconda。

Step 1. 创建 conda 环境并激活

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

Step 2. 按照 官方教程 安装 PyTorch，例如

GPU 平台:

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

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优方案来安装 MMSelfSup，不过整体流程也可以是自定义的，可参考自定义安装章节

2.1 最优方案

Step 0. 使用 MIM 安装 MMCV。

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

Step 1. 安装 MMSelfSup.

实例 a: 如果您直接或者开发 MMSelfSup，从源安装:

```
git clone https://github.com/open-mmlab/mmselfsup.git
cd mmselfsup
pip install -v -e .
# "-v" means verbose, or more output
# "-e" means installing a project in editable mode,
# thus any local modifications made to the code will take effect without␣
↪reinstallation.
```

实例 b: 如果您以 mmselfsup 为依赖项或者第三方库，可使用 pip 安装：

```
pip install mmselfsup
```

2.2 安装校验

走完上面的步骤，为了确保您正确安装了 MMSelfSup 以及其各种依赖库，请使用下面脚本来完成校验：

```
import torch

from mmselfsup.models import build_algorithm

model_config = dict(
    type='Classification',
    backbone=dict(
        type='ResNet',
        depth=50,
        in_channels=3,
        num_stages=4,
        strides=(1, 2, 2, 2),
        dilations=(1, 1, 1, 1),
        out_indices=[4],  # 0: conv-1, x: stage-x
        norm_cfg=dict(type='BN'),
        frozen_stages=-1),
    head=dict(
        type='ClsHead', with_avg_pool=True, in_channels=2048,
        num_classes=1000))

model = build_algorithm(model_config).cuda()

image = torch.randn((1, 3, 224, 224)).cuda()
label = torch.tensor([1]).cuda()

loss = model.forward_train(image, label)
```

如果您能顺利运行上面脚本，恭喜您已成功配置好所有环境。

2.3 自定义安装

2.3.1 基准测试

依照最优方案 可以保证基本功能, 如果您需要一些下游任务来对您的预训练模型进行评测，例如检测或者分割，请安装 MMDetection 和 MMSegmentation。

如果您不运行 MMDetection 和 MMSegmentation 基准测试，可以不进行安装。

您可以使用以下命令进行安装：

```
pip install mmdet mmsegmentation
```

若需要更详细的信息，您可以参考 MMDetection 和 MMSegmentation 的安装指导页面。

2.3.2 CUDA 版本

在安装 PyTorch 时，您需要确认 CUDA 版本。若您对此不清楚，可以按照我们的建议：

- 对于安培架构的 NVIDIA GPUs，例如 GeForce 30 系列或者 NVIDIA A100, CUDA 11 是必须的。
- 对于较老版本的 NVIDIA GPUs，CUDA 11 是兼容的，但是 CUDA 10.2 具有更好的兼容性以及更加轻量化。

请确认您的 GPU 驱动满足最小版本需求。请参考 此表 获取更多信息。

注解: 如果您按照我们的最优方案安装 CUDA runtime 库是足够的，因为本地不会编译 CUDA 代码。但是如果您希望从源编译 MMCV 或开发其它 CUDA 算子，您需要安装完整的 CUDA 工具包，从 NVIDIA 的网站，https://developer.nvidia.com/cuda-downloads，并它的版本需要和 PyTorch 的 CUDA 版本相匹配。如准确的 cudatoolkit 版本在 `conda install` 命令中。

2.3.3 不使用 MIM 安装 MMCV

MMCV 包含了 C++ 和 CUDA 扩展, 因此以一种复杂的方式依赖于 PyTorch。MIM 自动解决了这种依赖关系，并使安装更加容易，然而，这不是必须的。

使用 pip 安装 MMCV，而不是 MIM, 请参考 MMCV 安装指南。这需要根据 PyTorch 版本及其 CUDA 版本手动指定一个链接。

例如, 下列命令安装了 mmcv-full，基于 PyTorch 1.10.x 和 CUDA 11.3。

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

2.3.4 另一种选择: 使用 Docker

我们提供了一个配置好所有环境的 Dockerfile。

```
# build an image with PyTorch 1.6.0, CUDA 10.1, CUDNN 7.
docker build -f ./docker/Dockerfile --rm -t mmselfsup:torch1.10.0-cuda11.3-cudnn8 .
```

重要: 请确保您安装了 nvidia-container-toolkit。

运行下面命令:

```
docker run --gpus all --shm-size=8g -it -v {DATA_DIR}:/workspace/mmselfsup/data␣
↪mmselfsup:torch1.10.0-cuda11.3-cudnn8 /bin/bash
```

`{DATA_DIR}` 是保存你所有数据集的根目录。

2.3.5 在 Google Colab 上安装

Google Colab 一般已经安装了 PyTorch，因此，我们只需要使用以下命令安装 MMCV 和 MMSeflSup。

Step 0. 使用 MIM 安装 MMCV。

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

Step 1. 安装 MMSelfSup

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

Step 2. 安装校验

```
import mmselfsup
print(mmselfsup.__version__)
# Example output: 0.9.0
```

注解: Within Jupyter, the exclamation mark `!` is used to call external executables and `%cd` is a magic command to change the current working directory of Python.

2.4 问题解答

如果您在安装过程中遇到了什么问题，请先查阅 FAQ 页面. 您也可以在 GitHub 创建 issue，如果您没找到答案。

CHAPTER 3

使用不同版本的 MMSelfSup

如果在您本地安装了多个版本的 MMSelfSup, 我们推荐您为这多个版本创建不同的虚拟环境。

另外一个方式就是在您程序的入口脚本处，插入以下代码片段 (`train.py`, `test.py` 或则其他任何程序入口脚本)

```
import os.path as osp
import sys
sys.path.insert(0, osp.join(osp.dirname(osp.abspath(__file__)), '../'))
```

或则在不同版本的 MMSelfSup 的主目录中运行以下命令:

```
export PYTHONPATH="$(pwd)":$PYTHONPATH
```

CHAPTER 4

准备数据集

MMSelfSup 支持多个数据集。请遵循相应的数据准备指南。建议将您的数据集根目录软链接到 `$MMSELFSUP/data`。如果您的文件夹结构不同，您可能需要更改配置文件中的相应路径。


```
mmselfsup
├── mmselfsup
├── tools
├── configs
├── docs
├── data
|   ├── imagenet
|   |   ├── meta
|   |   ├── train
```

(下页继续)

(续上页)

```
|   |   ├── val
|   ├── places205
|   |   ├── meta
|   |   ├── train
|   |   ├── val
|   ├── inaturalist2018
|   |   ├── meta
|   |   ├── train
|   |   ├── val
|   ├── VOCdevkit
|   |   ├── VOC2007
|   ├── cifar
|   |   ├── cifar-10-batches-py
```

4.1 准备 ImageNet 数据集

对于 ImageNet，它有多个版本，但最常用的是 ILSVRC 2012。可以通过以下步骤得到：

1. 注册账号并登录 下载页面
2. 找到 ILSVRC2012 的下载链接，下载以下两个文件
   - ILSVRC2012_img_train.tar (~138GB)
   - ILSVRC2012_img_val.tar (~6.3GB)
3. 解压下载的文件
4. 使用这个 脚本 下载元数据

4.2 准备 Places205 数据集

对于 Places205，您需要：

1. 注册账号并登录 下载页面
2. 下载 Places205 经过缩放的图片以及训练集和验证集的图片列表
3. 解压下载的文件

4.3 准备 iNaturalist2018 数据集

对于 iNaturalist2018，您需要：

1. 从 下载页面 下载训练集和验证集图像及标注
2. 解压下载的文件
3. 使用脚本 `tools/data_converters/convert_inaturalist.py` 将原来的 json 标注格式转换为列表格式

4.4 准备 PASCAL VOC 数据集

假设您通常将数据集存储在 `$YOUR_DATA_ROOT` 中。下面的命令会自动将 PASCAL VOC 2007 下载到 `$YOUR_DATA_ROOT` 中，准备好所需的文件，在 `$MMSELFSUP` 下创建一个文件夹 `data`，并制作一个软链接 `VOCdevkit`。

```
bash tools/data_converters/prepare_voc07_cls.sh $YOUR_DATA_ROOT
```

4.5 准备 CIFAR10 数据集

如果没有找到 CIFAR10 系统将会自动下载。此外，由 `MMSelfSup` 实现的 `dataset` 也会自动将 CIFAR10 转换为适当的格式。

4.6 准备检测和分割数据集

4.6.1 检测

您可以参考 mmdet 来准备 COCO，VOC2007 和 VOC2012 检测数据集。

4.6.2 分割

您可以参考 mmseg 来准备 VOC2012AUG 和 Cityscapes 分割数据集。

CHAPTER 5

基础教程

本文档提供 MMSelfSup 相关用法的基础教程。如果您对如何安装 MMSelfSup 以及其相关依赖库有疑问, 请参考安装文档.

5.1 训练已有的算法

注意: 当您启动一个任务的时候，默认会使用 8 块显卡. 如果您想使用少于或多余 8 块显卡, 那么你的 batch size 也会同比例缩放，同时您的学习率服从一个线性缩放原则, 那么您可以使用以下公式来调整您的学习率: `new_lr = old_lr * new_ngpus / old_ngpus`. 除此之外，我们推荐您使用 `tools/dist_train.sh` 来启动训练任务，即便您只使用一块显卡, 因为 MMSelfSup 中有些算法不支持非分布式训练。

5.1.1 使用 CPU 训练

```
export CUDA_VISIBLE_DEVICES=-1
python tools/train.py ${CONFIG_FILE}
```

注意: 我们不推荐用户使用 CPU 进行训练，因为 CPU 的训练速度很慢，一些算法仅支持分布式训练, 例如 `SyncBN`，该方法需要分布式进行训练，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。

5.1.2 使用单张/多张显卡训练

```
sh tools/dist_train.sh ${CONFIG_FILE} ${GPUS} --work-dir ${YOUR_WORK_DIR} [optional␣
↪arguments]
```

可选参数:

- `--resume-from ${CHECKPOINT_FILE}`: 从某个 checkpoint 处继续训练.
- `--deterministic`: 开启 “deterministic” 模式, 虽然开启会使得训练速度降低，但是会保证结果可复现。

例如:

```
# checkpoints and logs saved in WORK_DIR=work_dirs/selfsup/odc/odc_resnet50_8xb64-
↪steplr-440e_in1k/
sh tools/dist_train.sh configs/selfsup/odc/odc_resnet50_8xb64-steplr-440e_in1k.py 8 --
↪work_dir work_dirs/selfsup/odc/odc_resnet50_8xb64-steplr-440e_in1k/
```

注意: 在训练过程中, checkpoints 和 logs 被保存在同一目录层级下.

此外, 如果您在一个被 slurm 管理的集群中训练，您可以使用以下的脚本开展训练:

```
GPUS_PER_NODE=${GPUS_PER_NODE} GPUS=${GPUS} SRUN_ARGS=${SRUN_ARGS} sh tools/slurm_
↪train.sh ${PARTITION} ${JOB_NAME} ${CONFIG_FILE} ${YOUR_WORK_DIR} [optional␣
↪arguments]
```

例如:

```
GPUS_PER_NODE=8 GPUS=8 sh tools/slurm_train.sh Dummy Test_job configs/selfsup/odc/odc_
↪resnet50_8xb64-steplr-440e_in1k.py work_dirs/selfsup/odc/odc_resnet50_8xb64-steplr-
↪440e_in1k/
```

5.1.3 使用多台机器训练

如果您想使用由 ethernet 连接起来的多台机器，您可以使用以下命令:

在第一台机器上:

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

在第二台机器上:

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

但是，如果您不使用高速网路连接这几台机器的话，训练将会非常慢。

如果您使用的是 slurm 来管理多台机器，您可以使用同在单台机器上一样的命令来启动任务，但是您必须得设置合适的环境变量和参数，具体可以参考 slurm_train.sh。

5.1.4 在一台机器上启动多个任务

如果您想在一台机器上启动多个任务，比如说，您启动两个 4 卡的任务在一台 8 卡的机器上，您需要为每个任务指定不懂的端口来防止端口冲突。

如果您使用 `dist_train.sh` 来启动训练任务:

```
CUDA_VISIBLE_DEVICES=0,1,2,3 PORT=29500 sh tools/dist_train.sh ${CONFIG_FILE} 4 --
↪work-dir tmp_work_dir_1
CUDA_VISIBLE_DEVICES=4,5,6,7 PORT=29501 sh tools/dist_train.sh ${CONFIG_FILE} 4 --
↪work-dir tmp_work_dir_2
```

如果您使用 slurm 来启动训练任务，你有两种方式来为每个任务设置不同的端口:

方法 1:

在 `config1.py` 中, 做如下修改:

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

在 `config2.py` 中，做如下修改:

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

然后您可以通过 config1.py 和 config2.py 来启动两个不同的任务.

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

方法 2:

除了修改配置文件之外, 您可以设置 `cfg-options` 来重写默认的端口号:

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

5.2 基准测试

我们同时提供多种命令来评估您的预训练模型, 具体您可以参考*Benchmarks*。

5.3 工具和建议

5.3.1 统计模型的参数

```
python tools/analysis_tools/count_parameters.py ${CONFIG_FILE}
```

5.3.2 发布模型

当你发布一个模型之前，您可能想做以下几件事情

- 将模型的参数转为 CPU tensor.
- 删除 optimizer 的状态参数.
- 计算 checkpoint 文件的哈希值，并将其添加到 checkpoint 的文件名中.

您可以使用以下命令来完整上面几件事情:

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

5.3.3 使用 t-SNE 来做模型可视化

我们提供了一个开箱即用的来做图片向量可视化的方法:

```
python tools/analysis_tools/visualize_tsne.py ${CONFIG_FILE} --checkpoint ${CKPT_PATH}
↪ --work-dir ${WORK_DIR} [optional arguments]
```

参数:

- CONFIG_FILE: 训练预训练模型的参数配置文件.
- CKPT_PATH: 预训练模型的路径.
- WORK_DIR: 保存可视化结果的路径.
- [optional arguments]: 可选参数，具体可以参考 visualize_tsne.py

5.3.4 MAE 可视化

我们提供了一个对 MAE 掩码效果和重建效果可视化可视化的方法:

```
python tools/misc/mae_visualization.py ${IMG} ${CONFIG_FILE} ${CKPT_PATH} --device $
↪{DEVICE}
```

参数:

- IMG: 用于可视化的图片
- CONFIG_FILE: 训练预训练模型的参数配置文件.
- CKPT_PATH: 预训练模型的路径.
- DEVICE: 用于推理的设备.

示例:

```
python tools/misc/mae_visualization.py tests/data/color.jpg configs/selfsup/mae/mae_
↪vit-base-p16_8xb512-coslr-400e_in1k.py mae_epoch_400.pth --device 'cuda:0'
```

5.3.5 可复现性

如果您想确保模型精度的可复现性，您可以设置 --deterministic 参数。但是，开启 --deterministic 意味着关闭 torch.backends.cudnn.benchmark, 所以会使模型的训练速度变慢。

CHAPTER 6

模型库

所有模型和部分基准测试如下。

6.1 预训练模型

备注：

- 训练细节记录在配置文件名中。
- 可以点击算法名获得更加全面的信息。

6.2 基准测试

在下列表格中，我们只展示了基于 ImageNet 数据集的线性评估，COCO17 数据集的目标检测和实例分割以及 PASCAL VOC12 Aug 数据集的语义分割任务，您可以点击预训练模型表格中的算法名查看更多基准测试结果。

6.2.1 ImageNet 线性评估

如果没有特殊说明，下列实验采用 MoCo 的设置，或者采用的训练设置写在备注中。

6.2.2 ImageNet 微调

6.2.3 COCO17 目标检测和实例分割

在 COCO17 数据集的目标检测和实例分割任务中，我们选用 MoCo 的评估设置，基于 Mask-RCNN FPN 网络架构，下列结果通过同样的 配置文件 训练得到。

6.2.4 Pascal VOC12 Aug 语义分割

在 Pascal VOC12 Aug 语义分割任务中，我们选用 MMSeg 的评估设置, 基于 FCN 网络架构, 下列结果通过同样的 配置文件 训练得到。

CHAPTER 7

教程 0: 学习配置

MMSelfSup 主要使用 python 文件作为配置。我们设计的配置文件系统集成了模块化和继承性，方便用户实施各种实验。所有的配置文件都放在 `configs` 文件夹。如果你想概要地审视配置文件，你可以执行 `python tools/misc/print_config.py` 查看完整配置。

7.1 配置文件与检查点命名约定

我们遵循下述约定来命名配置文件并建议贡献者也遵循该命名风格。配置文件名字被分成 4 部分：算法信息、模块信息、训练信息和数据信息。逻辑上，不同部分用下划线连接 `'_'`，同一部分中的单词使用破折线 `'-'` 连接。

```
{algorithm}_{module}_{training_info}_{data_info}.py
```

- `algorithm info`：包含算法名字的算法信息，例如 simclr，mocov2 等；
- `module info`：模块信息，用来表示一些 backbone，neck 和 head 信息；
- `training info`：训练信息，即一些训练调度，包括批大小，学习率调度，数据增强等；
- `data info`：数据信息：数据集名字，输入大小等，例如 imagenet，cifar 等。

7.1.1 算法信息

```
{algorithm}-{misc}
```

`Algorithm` 表示论文中的算法缩写和版本。例如：

- `relative-loc`：不同单词之间使用破折线连接 `'-'`
- `simclr`
- `mocov2`

`misc` 提供一些其他算法相关信息。例如：

- `npid-ensure-neg`
- `deepcluster-sobel`

7.1.2 模块信息

```
{backbone setting}-{neck setting}-{head_setting}
```

模块信息主要包含 backboe 信息。例如：

- `resnet50`
- `vit`（将会用在 mocov3 中）

或者其他一些需要在配置名字中强调的特殊的设置。例如：

- resnet50-nofrz：在一些下游任务的训练中，该 backbone 不会冻结 stages

7.1.3 训练信息

训练相关的配置，包括 batch size, lr schedule, data augment 等。

- Batch size，格式是 {gpu x batch_per_gpu}，例如 8xb32；
- Training recipe，该方法以如下顺序组织：{pipeline aug}-{train aug}-{loss trick}-{scheduler}-{epochs}

例如：

- 8xb32-mcrop-2-6-coslr-200e：mcrop 是 SwAV 提出的 pipeline 中的名为 multi-crop 的一部分。2 和 6 表示 2 个 pipeline 分别输出 2 个和 6 个裁剪图，而且裁剪信息记录在数据信息中；
- 8xb32-accum16-coslr-200e：accum16 表示权重会在梯度累积 16 个迭代之后更新。

7.1.4 数据信息

数据信息包含数据集，输入大小等。例如：

- in1k：ImageNet1k 数据集，默认使用的输入图像大小是 224x224
- in1k-384px：表示输入图像大小是 384x384
- cifar10
- inat18：iNaturalist2018 数据集，包含 8142 类
- places205

7.1.5 配置文件命名示例

```
swav_resnet50_8xb32-mcrop-2-6-coslr-200e_in1k-224-96.py
```

- swav：算法信息
- resnet50：模块信息
- 8xb32-mcrop-2-6-coslr-200e：训练信息
  - 8xb32：共使用 8 张 GPU，每张 GPU 上的 batch size 是 32
  - mcrop-2-6：使用 multi-crop 数据增强方法
  - coslr：使用余弦学习率调度器
  - 200e：训练模型 200 个周期
- in1k-224-96：数据信息，在 ImageNet1k 数据集上训练，输入大小是 224x224 和 96x96

7.1.6 检查点命名约定

权重的命名主要包括配置文件名字，日期和哈希值。

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

7.2 配置文件结构

在 configs/_base_ 文件中，有 4 种类型的基础组件文件，即

- models
- datasets
- schedules
- runtime

你可以通过继承一些基础配置文件快捷地构建你自己的配置。由 _base_ 下的组件组成的配置被称为 *原始配置*（*primitive*）。

为了易于理解，我们使用 MoCo v2 作为一个例子，并对它的每一行做出注释。若想了解更多细节，请参考 API 文档。

配置文件 configs/selfsup/mocov2/mocov2_resnet50_8xb32-coslr-200e_in1k.py 如下所述。

```
_base_ = [
    '../_base_/models/mocov2.py',                   # 模型
    '../_base_/datasets/imagenet_mocov2.py',        # 数据
    '../_base_/schedules/sgd_coslr-200e_in1k.py',   # 训练调度
    '../_base_/default_runtime.py',                 # 运行时设置
]

# 在这里，我们继承运行时设置并修改 max_keep_ckpts。
# max_keep_ckpts 控制在你的 work_dirs 中最大的 ckpt 文件的数量
# 如果它是 3，当 CheckpointHook (在 mmcv 中) 保存第 4 个 ckpt 时，
# 它会移除最早的那个，使总的 ckpt 文件个数保持为 3
checkpoint_config = dict(interval=10, max_keep_ckpts=3)
```

注解： 配置文件中的‘type’是一个类名，而不是参数的一部分。

../_base_/models/mocov2.py 是 MoCo v2 的基础模型配置。

```
model = dict(
    type='MoCo',  # 算法名字
```

(下页继续)

(续上页)

```
    queue_len=65536,  # 队列中维护的负样本数量
    feat_dim=128,  # 紧凑特征向量的维度，等于 neck 的 out_channels
    momentum=0.999,  # 动量更新编码器的动量系数
    backbone=dict(
        type='ResNet',  # Backbone name
        depth=50,  # backbone 深度，ResNet 可以选择 18、34、50、101、 152
        in_channels=3,  # 输入图像的通道数
        out_indices=[4],  # 输出特征图的输出索引，0 表示 conv-1，x 表示 stage-x
        norm_cfg=dict(type='BN')),  # 构建一个字典并配置 norm 层
    neck=dict(
        type='MoCoV2Neck',  # Neck name
        in_channels=2048,  # 输入通道数
        hid_channels=2048,  # 隐层通道数
        out_channels=128,  # 输出通道数
        with_avg_pool=True),  # 是否在 backbone 之后使用全局平均池化
    head=dict(
        type='ContrastiveHead',  # Head name，表示 MoCo v2 使用 contrastive loss
        temperature=0.2))  # 控制分布聚集程度的温度超参数
```

../_base_/datasets/imagenet_mocov2.py 是 MoCo v2 的基础数据集配置。

```
# 数据集配置
data_source = 'ImageNet'  # 数据源名字
dataset_type = 'MultiViewDataset' # 组成 pipeline 的数据集类型
img_norm_cfg = dict(
    mean=[0.485, 0.456, 0.406],  # 用来预训练预训练 backboe 模型的均值
    std=[0.229, 0.224, 0.225])  # 用来预训练预训练 backbone 模型的标准差
# mocov2 和 mocov1 之间的差异在于 pipeline 中的 transforms
train_pipeline = [
    dict(type='RandomResizedCrop', size=224, scale=(0.2, 1.)),  # RandomResizedCrop
    dict(
        type='RandomAppliedTrans',  # 以 0.8 的概率随机使用 ColorJitter 增强方法
        transforms=[
            dict(
                type='ColorJitter',
                brightness=0.4,
                contrast=0.4,
                saturation=0.4,
                hue=0.1)
        ],
        p=0.8),
    dict(type='RandomGrayscale', p=0.2),  # 0.2 概率的 RandomGrayscale
    dict(type='GaussianBlur', sigma_min=0.1, sigma_max=2.0, p=0.5),  # 0.5 概率的随机␣
↪GaussianBlur
```

(下页继续)

(续上页)

```
    dict(type='RandomHorizontalFlip'),  # 随机水平翻转图像
]

# prefetch
prefetch = False  # 是否使用 prefetch 加速 pipeline
if not prefetch:
    train_pipeline.extend(
        [dict(type='ToTensor'),
         dict(type='Normalize', **img_norm_cfg)])

# 数据集汇总
data = dict(
    samples_per_gpu=32,  # 单张 GPU 的批大小，共 32*8=256
    workers_per_gpu=4,  # 每张 GPU 用来 pre-fetch 数据的 worker 个数
    drop_last=True,  # 是否丢弃最后一个 batch 的数据
    train=dict(
        type=dataset_type,  # 数据集名字
        data_source=dict(
            type=data_source,  # 数据源名字
            data_prefix='data/imagenet/train',  # 数据集根目录，当 ann_file 不存在时，类别信
息自动从该根目录自动获取
            ann_file='data/imagenet/meta/train.txt',  #  若 ann_file 存在，类别信息从该文件
获取
        ),
        num_views=[2],  # pipeline 中不同的视图个数
        pipelines=[train_pipeline],  # 训练 pipeline
        prefetch=prefetch,  # 布尔值
    ))
```

../_base_/schedules/sgd_coslr-200e_in1k.py 是 MoCo v2 的基础调度配置。

```
# 优化器
optimizer = dict(
    type='SGD',  # 优化器类型
    lr=0.03,  # 优化器的学习率，参数的详细使用请参阅 PyTorch 文档
    weight_decay=1e-4,  # 动量参数
    momentum=0.9)  # SGD 的权重衰减
# 用来构建优化器钩子的配置，请参考 https://github.com/open-mmlab/mmcv/blob/master/mmcv/
↪runner/hooks/optimizer.py#L8 中的实现细节。
optimizer_config = dict()  # 这个配置可以设置 grad_clip, coalesce, bucket_size_mb 等。

# 学习策略
# 用来注册 LrUpdater 钩子的学习率调度配置
lr_config = dict(
```

(下页继续)

(续上页)

```
    policy='CosineAnnealing',  # 调度器策略，也支持 Step, Cyclic 等。 LrUpdater 支持的细节请
参考 https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/lr_updater.py#L9。
    min_lr=0.)  # CosineAnnealing 中的最小学习率设置

# 运行时设置
runner = dict(
    type='EpochBasedRunner',  # 使用的 runner 的类型 (例如 IterBasedRunner 或
↪EpochBasedRunner)
    max_epochs=200) # 运行工作流周期总数的 Runner 的 max_epochs，对于 IterBasedRunner 使用
↪`max_iters`
```

../_base_/default_runtime.py 是运行时的默认配置。

```
# 保存检查点
checkpoint_config = dict(interval=10)  # 保存间隔是 10

# yapf:disable
log_config = dict(
    interval=50,  # 打印日志的间隔
    hooks=[
        dict(type='TextLoggerHook'),  # 也支持 Tensorboard logger
        # dict(type='TensorboardLoggerHook'),
    ])
# yapf:enable

# 运行时设置
dist_params = dict(backend='nccl') # 设置分布式训练的参数，端口也支持设置。
log_level = 'INFO'  # 日志的输出 level。
load_from = None  # 加载 ckpt
resume_from = None  # 从给定的路径恢复检查点，将会从检查点保存时的周期恢复训练。
workflow = [('train', 1)]  # Workflow for runner. [('train', 1)] 表示有一个 workflow，该
↪workflow 名字是 'train' 且执行一次。
persistent_workers = True  # Dataloader 中设置 persistent_workers 的布尔值，详细信息请参考
↪PyTorch 文档
```

7.3 继承和修改配置文件

为了易于理解，我们推荐贡献者从现有方法继承。

对于同一个文件夹下的所有配置，我们推荐只使用一个 *原始*（*primitive*）配置。其他所有配置应当从 *原始*（*primitive*）配置继承，这样最大的继承层次为 3。

例如，如果你的配置文件是基于 MoCo v2 做一些修改，首先你可以通过指定 `_base_ ='./mocov2_resnet50_8xb32-coslr-200e_in1k.py.py'`（相对于你的配置文件的路径）继承基本的 MoCo v2 结构，数据集和其他训练设置，接着在配置文件中修改一些必要的参数。现在，我们举一个更具体的例子，我们想使用 `configs/selfsup/mocov2/mocov2_resnet50_8xb32-coslr-200e_in1k.py.py` 中几乎所有的配置，但是将训练周期数从 200 修改为 800，修改学习率衰减的时机和数据集路径，你可以创建一个名为 `configs/selfsup/mocov2/mocov2_resnet50_8xb32-coslr-800e_in1k.py.py` 的新配置文件，内容如下：

```
_base_ = './mocov2_resnet50_8xb32-coslr-200e_in1k.py'

runner = dict(max_epochs=800)
```

7.3.1 使用配置中的中间变量

在配置文件中使用一些中间变量会使配置文件更加清晰和易于修改。

例如：数据中的中间变量有 `data_source`, `dataset_type`, `train_pipeline`, `prefetch`. 我们先定义它们再将它们传进 `data`。

```
data_source = 'ImageNet'
dataset_type = 'MultiViewDataset'
img_norm_cfg = dict(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
train_pipeline = [...]

# prefetch
prefetch = False  # 是否使用 prefetch 加速 pipeline
if not prefetch:
    train_pipeline.extend(
        [dict(type='ToTensor'),
         dict(type='Normalize', **img_norm_cfg)])

# dataset summary
data = dict(
    samples_per_gpu=32,
    workers_per_gpu=4,
    drop_last=True,
    train=dict(type=dataset_type, type=data_source, data_prefix=...),
```

(下页继续)

(续上页)

```
        num_views=[2],
        pipelines=[train_pipeline],
        prefetch=prefetch,
    ))
```

7.3.2 忽略基础配置中的字段

有时候，你需要设置 _delete_=True 来忽略基础配置文件中一些域的内容。你可以参考 mmcv 获得更多说明。

接下来是一个例子。如果你希望在 simclr 的设置中使用 MoCoV2Neck，仅仅继承并直接修改将会报 get unexcepected keyword 'num_layers' 错误，因为在 model.neck 域信息中，基础配置 'num_layers' 字段被保存下来了，你需要添加 _delete_=True 来忽略 model.neck 在基础配置文件中的有关字段的内容。

```
_base_ = 'simclr_resnet50_8xb32-coslr-200e_in1k.py'

model = dict(
    neck=dict(
        _delete_=True,
        type='MoCoV2Neck',
        in_channels=2048,
        hid_channels=2048,
        out_channels=128,
        with_avg_pool=True))
```

7.3.3 使用基础配置中的字段

有时候，你可能引用 _base_ 配置中一些字段，以避免重复定义。你可以参考mmcv 获取更多的说明。

下面是在训练数据预处理 pipeline 中使用 auto augment 的一个例子，请参考 configs/selfsup/odc/odc_resnet50_8xb64-steplr-440e_in1k.py。当定义 num_classes 时，只需要将 auto augment 的定义文件名添入到 _base_，并使用 {{_base_.num_classes}} 来引用这些变量：

```
_base_ = [
    '../_base_/models/odc.py',
    '../_base_/datasets/imagenet_odc.py',
    '../_base_/schedules/sgd_steplr-200e_in1k.py',
    '../_base_/default_runtime.py',
]

# model settings
```

(下页继续)

(续上页)

```
model = dict(
    head=dict(num_classes={{_base_.num_classes}}),
    memory_bank=dict(num_classes={{_base_.num_classes}}),
)

# optimizer
optimizer = dict(
    type='SGD',
    lr=0.06,
    momentum=0.9,
    weight_decay=1e-5,
    paramwise_options={'\\Ahead.': dict(momentum=0.)})

# learning policy
lr_config = dict(policy='step', step=[400], gamma=0.4)

# runtime settings
runner = dict(type='EpochBasedRunner', max_epochs=440)
# max_keep_ckpts 控制在你的 work_dirs 中保存的 ckpt 的最大数目
# 如果它等于 3，CheckpointHook (在 mmcv 中) 在保存第 4 个 ckpt 时，
# 它会移除最早的那个，使总的 ckpt 文件个数保持为 3
checkpoint_config = dict(interval=10, max_keep_ckpts=3)
```

7.4 通过脚本参数修改配置

当用户使用脚本 "tools/train.py" 或 "tools/test.py" 提交任务，或者其他工具时，可以通过指定 `--cfg-options` 参数来直接修改配置文件中内容。

- 更新字典链中的配置的键

  配置项可以通过遵循原始配置中键的层次顺序指定。例如，`--cfg-options model.backbone.norm_eval=False` 改变模型 backbones 中的所有 BN 模块为 `train` 模式。

- 更新列表中配置的键

  你的配置中的一些配置字典是由列表组成。例如，训练 pipeline `data.train.pipeline` 通常是一个列表。例如 `[dict(type='LoadImageFromFile'), dict(type='TopDownRandomFlip', flip_prob=0.5), ...]`。如果你想要在 pipeline 中将 `'flip_prob=0.5'` 修改为 `'flip_prob=0.0'`，你可以指定 `--cfg-options data.train.pipeline.1.flip_prob=0.0`

- 更新 list/tuples 中的值

  如果想要更新的值是一个列表或者元组，例如：配置文件通常设置 `workflow=[('train', 1)]`。如果你想要改变这个键，你可以指定 `--cfg-options workflow="[(train,1),(val,1)]"`。注意：对于 list/tuple 数据类型，引号" 是必须的，并且在指定值的时候，在引号中 **NO** 空白字符。

7.5 导入用户定义模块

注解： 这部分内容初学者可以跳过，只在使用其他 MM-codebase 时会用到，例如使用 mmcls 作为第三方库来构建你的工程。

你可能使用其他的 MM-codebase 来完成你的工程，并在工程中创建新的数据集类，模型类，数据增强类等。为了简化代码，你可以使用 MM-codebase 作为第三方库，只需要保存你自己额外的代码，并在配置文件中导入自定义模块。你可以参考 OpenMMLab Algorithm Competition Project 中的例子。

在你自己的配置文件中添加如下所述的代码：

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

CHAPTER 8

教程 1: 添加新的数据格式

在本节教程中，我们将介绍创建自定义数据格式的基本步骤：

如果你的算法不需要任何定制的数据格式，你可以使用 datasets 目录中这些现成的数据格式。但是要使用这些现有的数据格式，你必须将你的数据集转换为现有的数据格式。

8.1 自定义数据格式示例

假设你的数据集的注释文件格式是：

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

要编写一个新的数据格式，你需要实现：

- 子类 `DataSource`：继承自父类 `BaseDataSource`——负责加载注释文件和读取图像。
- 子类 `Dataset`：继承自父类 `BaseDataset` ——负责对图像进行转换和打包。

8.2 创建 DataSource 子类

假设你基于父类 DataSource 创建的子类名为 NewDataSource，你可以在 mmselfsup/datasets/data_sources 目录下创建一个文件，文件名为 new_data_source.py ，并在这个文件中实现 NewDataSource 创建。

```
import mmcv
import numpy as np

from ..builder import DATASOURCES
from .base import BaseDataSource


@DATASOURCES.register_module()
class NewDataSource(BaseDataSource):

    def load_annotations(self):

        assert isinstance(self.ann_file, str)
        data_infos = []
        # writing your code here.
        return data_infos
```

然后，在 mmselfsup/dataset/data_sources/__init__.py 中添加 NewDataSource。

```
from .base import BaseDataSource
...
from .new_data_source import NewDataSource

__all__ = [
    'BaseDataSource', ..., 'NewDataSource'
]
```

8.3 创建 Dataset 子类

假设你基于父类 Dataset 创建的子类名为 NewDataset，你可以在 mmselfsup/datasets 目录下创建一个文件，文件名为 new_dataset.py ，并在这个文件中实现 NewDataset 创建。

```
# Copyright (c) OpenMMLab. All rights reserved.
import torch
from mmcv.utils import build_from_cfg
from torchvision.transforms import Compose
```

(下页继续)

(续上页)

```
from .base import BaseDataset
from .builder import DATASETS, PIPELINES, build_datasource
from .utils import to_numpy


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

    def __init__(self, data_source, num_views, pipelines, prefetch=False):
        # writing your code here
    def __getitem__(self, idx):
        # writing your code here
        return dict(img=img)

    def evaluate(self, results, logger=None):
        return NotImplemented
```

然后，在 mmselfsup/dataset/__init__.py 中添加 NewDataset。

```
from .base import BaseDataset
...
from .new_dataset import NewDataset

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

8.4 修改配置文件

为了使用 NewDataset，你可以修改配置如下：

```
train=dict(
        type='NewDataset',
        data_source=dict(
            type='NewDataSource',
        ),
        num_views=[2],
        pipelines=[train_pipeline],
        prefetch=prefetch,
    ))
```

CHAPTER 9

教程 2：自定义数据管道

- 教程 2：自定义数据管道
  - Pipeline 概览
  - 在 Pipeline 中创建新的数据增强

9.1 Pipeline 概览

DataSource 和 Pipeline 是 Dataset 的两个重要组件。我们已经在*add_new_dataset* 中介绍了 DataSource 。Pipeline 负责对图像进行一系列的数据增强，例如随机翻转。

这是用于 SimCLR 训练的 Pipeline 的配置示例：

```
train_pipeline = [
    dict(type='RandomResizedCrop', size=224),
    dict(type='RandomHorizontalFlip'),
    dict(
        type='RandomAppliedTrans',
        transforms=[
            dict(
                type='ColorJitter',
                brightness=0.8,
                contrast=0.8,
                saturation=0.8,
```

(下页继续)

(续上页)

```
                hue=0.2)
        ],
        p=0.8),
    dict(type='RandomGrayscale', p=0.2),
    dict(type='GaussianBlur', sigma_min=0.1, sigma_max=2.0, p=0.5)
]
```

Pipeline 中的每个增强都接收一张图像作为输入，并输出一张增强后的图像。

9.2 在 Pipeline 中创建新的数据增强

1. 在 transforms.py 中编写一个新的数据增强函数，并覆盖 __call__ 函数，该函数接收一张 Pillow 图像作为输入：

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

    def __call__(self, img):
        # apply transforms on img
        return img
```

2. 在配置文件中使用它。我们重新使用上面的配置文件，并在其中添加 MyTransform。

```
train_pipeline = [
    dict(type='RandomResizedCrop', size=224),
    dict(type='RandomHorizontalFlip'),
    dict(type='MyTransform'),
    dict(
        type='RandomAppliedTrans',
        transforms=[
            dict(
                type='ColorJitter',
                brightness=0.8,
                contrast=0.8,
                saturation=0.8,
                hue=0.2)
        ],
        p=0.8),
    dict(type='RandomGrayscale', p=0.2),
    dict(type='GaussianBlur', sigma_min=0.1, sigma_max=2.0, p=0.5)
]
```

CHAPTER 10

教程 3：添加新的模块

在自监督学习领域，每个模型可以被分为以下四个部分：

- backbone：用于提取图像特征。
- projection head：将 backbone 提取的特征映射到另一空间。
- loss：用于模型优化的损失函数。
- memory bank（可选）：一些方法（例如 `odc`），需要额外的 memory bank 用于存储图像特征。

10.1 添加新的 backbone

假设我们要创建一个自定义的 backbone `CustomizedBackbone`。

1. 创建新文件 `mmselfsup/models/backbones/customized_backbone.py` 并在其中实现 `CustomizedBackbone` 。

```
import torch.nn as nn
from ..builder import BACKBONES

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

    def __init__(self, **kwargs):

        ## TODO

    def forward(self, x):

        ## TODO

    def init_weights(self, pretrained=None):

        ## TODO

    def train(self, mode=True):

        ## TODO
```

2. 在 mmselfsup/models/backbones/__init__.py 中导入自定义的 backbone。

```
from .customized_backbone import CustomizedBackbone

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

3. 在你的配置文件中使用它。

```
model = dict(
    ...
    backbone=dict(
        type='CustomizedBackbone',
        ...),
    ...
)
```

10.2 添加新的 Necks

我们在 mmselfsup/models/necks 中包含了所有的 projection heads。假设我们要创建一个 CustomizedProjHead 。

1. 创建一个新文件 mmselfsup/models/necks/customized_proj_head.py 并在其中实现 CustomizedProjHead 。

```
import torch.nn as nn
from mmcv.runner import BaseModule

from ..builder import NECKS


@NECKS.register_module()
class CustomizedProjHead(BaseModule):

    def __init__(self, *args, **kwargs):
        super(CustomizedProjHead, self).__init__(init_cfg)
        ## TODO
    def forward(self, x):
        ## TODO
```

你需要实现前向函数，该函数从 backbone 中获取特征，并输出映射后的特征。

2. 在 mmselfsup/models/necks/__init__ 中导入 CustomizedProjHead 。

```
from .customized_proj_head import CustomizedProjHead

__all__ = [
    ...,
    CustomizedProjHead,
    ...
]
```

3. 在你的配置文件中使用它。

```
model = dict(
    ...,
    neck=dict(
        type='CustomizedProjHead',
        ...),
   ...)
```

10.3 添加新的损失

为了增加一个新的损失函数，我们主要在损失模块中实现 `forward` 函数。

1. 创建一个新的文件 `mmselfsup/models/heads/customized_head.py` 并在其中实现你自定义的 `CustomizedHead` 。

```
import torch
import torch.nn as nn
from mmcv.runner import BaseModule

from ..builder import HEADS


@HEADS.register_module()
class CustomizedHead(BaseModule):

    def __init__(self, *args, **kwargs):
        super(CustomizedHead, self).__init__()

        ## TODO

    def forward(self, *args, **kwargs):

        ## TODO
```

2. 在 `mmselfsup/models/heads/__init__.py` 中导入该模块。

```
from .customized_head import CustomizedHead

__all__ = [..., CustomizedHead, ...]
```

3. 在你的配置文件中使用它。

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

10.4 合并所有改动

在创建了上述每个组件后，我们需要创建一个 CustomizedAlgorithm 来有逻辑的将他们组织到一起。CustomizedAlgorithm 接收原始图像作为输入，并将损失输出给优化器。

1. 创建一个新文件 mmselfsup/models/algorithms/customized_algorithm.py 并在其中实现 CustomizedAlgorithm。

```
# Copyright (c) OpenMMLab. All rights reserved.
import torch

from ..builder import ALGORITHMS, build_backbone, build_head, build_neck
from ..utils import GatherLayer
from .base import BaseModel


@ALGORITHMS.register_module()
class CustomizedAlgorithm(BaseModel):

    def __init__(self, backbone, neck=None, head=None, init_cfg=None):
        super(SimCLR, self).__init__(init_cfg)

        ## TODO

    def forward_train(self, img, **kwargs):

        ## TODO
```

2. 在 mmselfsup/models/algorithms/__init__.py 中导入该模块。

```
from .customized_algorithm import CustomizedAlgorithm

__all__ = [..., CustomizedAlgorithm, ...]
```

3. 在你的配置文件中使用它。

```
model = dict(
    type='CustomizedAlgorightm',
    backbone=...,
    neck=...,
    head=...)
```

CHAPTER 11

教程 4：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、动量调整策略、参数化精细配置、梯度裁剪、梯度累计以及用户自定义优化方法等。

11.1 构造 PyTorch 内置优化器

我们已经支持使用 PyTorch 实现的所有优化器，要使用和修改这些优化器，请修改配置文件中的 `optimizer` 字段。

例如，如果您想使用 SGD，可以进行如下修改。

```
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。

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)`，则需要进行如下配置

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

除了 PyTorch 实现的优化器之外，我们还在 `mmselfsup/core/optimizer/optimizers.py` 中构造了一个LARS。

11.2 定制学习率调整策略

11.2.1 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。

例如，在 SimCLR 网络训练中，我们使用 CosineAnnealing 的学习率衰减策略，配置文件为：

```
lr_config = dict(
    policy='CosineAnnealing',
    ...)
```

在训练过程中，程序会周期性地调用 MMCV 中的 CosineAnealingLrUpdaterHook 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 `Poly` 等。详情可见 这里

11.2.2 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 MMSelfSup 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，默认为 True 。如果被设置为 False ，则以 iter 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`：预热的初始学习率 `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)
```

11.2.3 定制动量调整策略

我们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。

动量调整策略通常与学习率调整策略一起使用，例如，以下配置用于加速收敛。更多细节可参考 CyclicLrUpdater 和 CyclicMomentumUpdater。

例如：

```
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,
)
```

11.2.4 参数化精细配置

一些模型的优化策略，包含作用于特定参数的精细设置，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。为了进行精细配置，我们通过 `optimizer` 中的 `paramwise_options` 参数进行配置。

例如，如果我们不想对 BatchNorm 或 GroupNorm 的参数以及各层的 bias 应用权重衰减，我们可以使用以下配置文件：

```
optimizer = dict(
    type=...,
    lr=...,
    paramwise_options={
        '(bn|gn)(\\d+)?.(weight|bias)':
        dict(weight_decay=0.),
        'bias': dict(weight_decay=0.)
    })
```

11.3 梯度裁剪与梯度累计

11.3.1 梯度裁剪

除了 PyTorch 优化器的基本功能，我们还提供了一些增强功能，例如梯度裁剪、梯度累计等。更多细节参考 MMCV。

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：

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

当使用继承并修改基础配置时，如果基础配置中 `grad_clip=None`，需要添加 `_delete_=True`。

11.3.2 梯度累计

计算资源缺乏时，每个批次的大小（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="DistOptimizerHook", update_interval=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")
```

11.4 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 MMSelfSup 未实现的优化方法，可以通过以下方法添加。

在 `mmselfsup/core/optimizer/optimizers.py` 中实现您的 `CustomizedOptim` 。

```
import torch
from torch.optim import *  # noqa: F401,F403
from torch.optim.optimizer import Optimizer, required
```

(下页继续)

(续上页)

```
from mmcv.runner.optimizer.builder import OPTIMIZERS

@OPTIMIZER.register_module()
class CustomizedOptim(Optimizer):

    def __init__(self, *args, **kwargs):

        ## TODO

    @torch.no_grad()
    def step(self):

        ## TODO
```

修改 mmselfsup/core/optimizer/__init__.py，将其导入

```
from .optimizers import CustomizedOptim
from .builder import build_optimizer

__all__ = ['CustomizedOptim', 'build_optimizer', ...]
```

在配置文件中指定优化器

```
optimizer = dict(
    type='CustomizedOptim',
    ...
)
```

CHAPTER 12

教程 5：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。

12.1 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，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，准确率）。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验证地反复执行。

默认情况下，我们更推荐在每个训练轮次后使用 **EvalHook** 进行模型验证。

12.2 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。

钩子只有被注册进执行器才起作用，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

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。

12.2.1 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：

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、NeptuneLoggerHook、DvcliveLoggerHook 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="auto", 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。

12.3 使用其他内置钩子

一些钩子已在 MMCV 和 MMClassification 中实现：

- EMAHook
- SyncBuffersHook
- EmptyCacheHook
- ProfilerHook
- ……

如果要用的钩子已经在 MMCV 中实现，可以直接修改配置以使用该钩子，如下格式：

```
mmcv_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 iter：

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

12.4 自定义钩子

12.4.1 1. 创建一个新钩子

这里举一个在 MMSelfSup 中创建一个新钩子的示例：

```
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

    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`。

12.4.2 2. 导入新钩子

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

- 修改 mmselfsup/core/hooks/__init__.py 进行导入，如下：

```
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=['mmselfsup.core.hooks.my_hook'], allow_failed_
↪imports=False)
```

12.4.3 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='ABOVE_NORMAL')
]
```

默认情况下，在注册过程中，钩子的优先级设置为 NORMAL 。

CHAPTER 13

教程 6：运行基准评测

在 MMSelfSup 中，我们提供了许多基准评测，因此模型可以在不同的下游任务中进行评估。这里提供了全面的教程和例子来解释如何用 MMSelfSup 运行所有的基准。

首先，你应该通过 `tools/model_converters/extract_backbone_weights.py` 提取你的 backbone 权重。

```
python ./tools/model_converters/extract_backbone_weights.py {CHECKPOINT} {MODEL_FILE}
```

参数：

- `CHECKPOINT`：selfsup 方法的权重文件，名称为 epoch_*.pth 。
- `MODEL_FILE`：输出的 backbone 权重文件。如果没有指定，下面的 `PRETRAIN` 会使用这个提取的模型文件。

13.1 分类

关于分类，我们在 tools/benchmarks/classification/文件夹中提供了脚本，其中有 4 个 .sh 文件，1 个用于 VOC SVM 相关的分类任务的文件夹，1 个用于 ImageNet 最邻近分类任务的文件夹。

13.1.1 VOC SVM / Low-shot SVM

为了运行这个基准评测，你应该首先准备你的 VOC 数据集，数据准备的细节请参考prepare_data.md。

为了评估预训练的模型，你可以运行以下命令。

```
# 分布式版本
bash tools/benchmarks/classification/svm_voc07/dist_test_svm_pretrain.sh ${SELFSUP_
↪CONFIG} ${GPUS} ${PRETRAIN} ${FEATURE_LIST}

# slurm 版本
bash tools/benchmarks/classification/svm_voc07/slurm_test_svm_pretrain.sh ${PARTITION}
↪ ${JOB_NAME} ${SELFSUP_CONFIG} ${PRETRAIN} ${FEATURE_LIST}
```

此外，如果你想评估 runner 保存的 ckpt 文件，你可以运行下面的命令。

```
# 分布式版本
bash tools/benchmarks/classification/svm_voc07/dist_test_svm_epoch.sh ${SELFSUP_
↪CONFIG} ${EPOCH} ${FEATURE_LIST}

# slurm 版本
bash tools/benchmarks/classification/svm_voc07/slurm_test_svm_epoch.sh ${PARTITION} $
↪{JOB_NAME} ${SELFSUP_CONFIG} ${EPOCH} ${FEATURE_LIST}
```

用 ckpt 测试时，代码使用 epoch_*.pth 文件，不需要提取权重。

备注：

- ${SELFSUP_CONFIG} 是自监督实验的配置文件。
- ${FEATURE_LIST} 是一个字符串，指定 layer1 到 layer5 的特征用于评估；例如，如果你只想评估 layer5，那么 FEATURE_LIST 是 “feat5”，如果你想评估所有的特征，那么 FEATURE_LIST 是 “feat1 feat2 feat3 feat4 feat5”（用空格分隔）。如果留空，默认 FEATURE_LIST 为 “feat5”。
- PRETRAIN：预训练的模型文件。
- 如果你想改变 GPU 的数量，你可以在命令的开头加上 GPUS_PER_NODE=4 GPUS=4。
- EPOCH 是你要测试的 ckpt 的 epoch 数。

13.1.2 线性评估

线性评估是最通用的基准评测之一，我们整合了几篇论文的配置设置，也包括多头线性评估。我们在自己的代码库中为多头功能编写分类模型，因此，为了运行线性评估，我们仍然使用 `.sh` 脚本来启动训练。支持的数据集是 **ImageNet**、**Places205** 和 **iNaturalist18**。

```
# 分布式版本
bash tools/benchmarks/classification/dist_train_linear.sh ${CONFIG} ${PRETRAIN}

# slurm 版本
bash tools/benchmarks/classification/slurm_train_linear.sh ${PARTITION} ${JOB_NAME} $
↪{CONFIG} ${PRETRAIN}
```

备注：

- 默认的 GPU 数量是 8，当改变 GPUS 时，也请相应改变配置文件中的 `samples_per_gpu`，以确保总 batch size 为 256。
- `CONFIG`: 使用 `configs/benchmarks/classification/` 下的配置文件。具体有 `imagenet`（不包括 `imagenet_*percent` 文件夹），`places205` 和 `inaturalist2018`。
- `PRETRAIN`：预训练的模型文件。

13.1.3 ImageNet 半监督分类

为了运行 ImageNet 半监督分类，我们仍然使用 `.sh` 脚本来启动训练。

```
# 分布式版本
bash tools/benchmarks/classification/dist_train_semi.sh ${CONFIG} ${PRETRAIN}

# slurm 版本
bash tools/benchmarks/classification/slurm_train_semi.sh ${PARTITION} ${JOB_NAME} $
↪{CONFIG} ${PRETRAIN}
```

备注：

- 默认的 GPU 数量是 4。
- `CONFIG`：使用 `configs/benchmarks/classification/imagenet/` 下的配置文件，名为 `imagenet_*percent` 文件夹。
- `PRETRAIN`：预训练的模型文件。

13.1.4 ImageNet 最邻近分类

为了使用最邻近基准评测来评估预训练的模型，你可以运行以下命令。

```
# 分布式版本
bash tools/benchmarks/classification/knn_imagenet/dist_test_knn_pretrain.sh ${SELFSUP_
↪CONFIG} ${PRETRAIN}

# slurm 版本
bash tools/benchmarks/classification/knn_imagenet/slurm_test_knn_pretrain.sh $
↪{PARTITION} ${JOB_NAME} ${SELFSUP_CONFIG} ${PRETRAIN}
```

此外，如果你想评估 runner 保存的 ckpt 文件，你可以运行下面的命令。

```
# 分布式版本
bash tools/benchmarks/classification/knn_imagenet/dist_test_knn_epoch.sh ${SELFSUP_
↪CONFIG} ${EPOCH}

# slurm 版本
bash tools/benchmarks/classification/knn_imagenet/slurm_test_knn_epoch.sh ${PARTITION}
↪ ${JOB_NAME} ${SELFSUP_CONFIG} ${EPOCH}
```

用 ckpt 测试时，代码使用 epoch_*.pth 文件，不需要提取权重。

备注：

- `${SELFSUP_CONFIG}` 是自监督实验的配置文件。
- `PRETRAIN`：预训练的模型文件。
- 如果你想改变 GPU 的数量，你可以在命令的开头加上 `GPUS_PER_NODE=4 GPUS=4`。
- `EPOCH` 是你要测试的 ckpt 的 epoch 数。

13.2 检测

在这里，我们倾向于使用 MMDetection 来完成检测任务。首先，确保你已经安装了MIM，它也是 OpenMMLab 的一个项目。

```
pip install openmim
```

安装该软件包非常容易。

此外，请参考 MMDet 的安装和数据准备

安装完成后，你可以用简单的命令运行 MMDet

```
# 分布式版本
bash tools/benchmarks/mmdetection/mim_dist_train.sh ${CONFIG} ${PRETRAIN} ${GPUS}

# slurm 版本
bash tools/benchmarks/mmdetection/mim_slurm_train.sh ${PARTITION} ${CONFIG} $
↪{PRETRAIN}
```

备注:

- CONFIG: 使用 configs/benchmarks/mmdetection/ 下的配置文件或编写你自己的配置文件。
- PRETRAIN: 预训练的模型文件。

或者如果你想用detectron2做检测任务，我们也提供一些配置文件。请参考INSTALL.md进行安装，并按照目录结构来准备 detectron2 所需的数据集。

```
conda activate detectron2 # 在这里使用 detectron2 环境，否则使用 open-mmlab 环境
cd benchmarks/detection
python convert-pretrain-to-detectron2.py ${WEIGHT_FILE} ${OUTPUT_FILE} # 必须使用 .pkl␣
↪作为输出文件扩展名
bash run.sh ${DET_CFG} ${OUTPUT_FILE}
```

```

## 分割

对于语义分割任务，我们使用的是 MMSegmentation 。首先，确保你已经安装了[MIM](https://github.com/
↪open-mmlab/mim)，它也是 OpenMMLab 的一个项目。

```shell
pip install openmim
```
安装该软件包非常容易。

此外，请参考 MMSeg 的[安装](https://github.com/open-mmlab/mmsegmentation/blob/master/docs/
↪get_started.md)和[数据准备](https://github.com/open-mmlab/mmsegmentation/blob/master/
↪docs/dataset_prepare.md#prepare-datasets)。

安装后，你可以用简单的命令运行 MMSeg

```shell
# 分布式版本
bash tools/benchmarks/mmsegmentation/mim_dist_train.sh ${CONFIG} ${PRETRAIN} ${GPUS}

# slurm 版本
bash tools/benchmarks/mmsegmentation/mim_slurm_train.sh ${PARTITION} ${CONFIG} $
↪{PRETRAIN}
```

(下页继续)
```

(续上页)

```
```

备注:
- `CONFIG`: 使用 `configs/benchmarks/mmsegmentation/` 下的配置文件或编写自己的配置文件。
- `PRETRAIN`: 预训练的模型文件。
```
```

CHAPTER 14

BYOL

Bootstrap your own latent: A new approach to self-supervised Learning

14.1 Abstract

Bootstrap **Y**our **O**wn **L**atent (BYOL) is a new approach to self-supervised image representation learning. BYOL relies on two neural networks, referred to as online and target networks, that interact and learn from each other. From an augmented view of an image, we train the online network to predict the target network representation of the same image under a different augmented view. At the same time, we update the target network with a slow-moving average of the online network.

14.2 Results and Models

Back to model_zoo.md to download models.

In this page, we provide benchmarks as much as possible to evaluate our pre-trained models. If not mentioned, all models are pre-trained on ImageNet-1k dataset.

14.2.1 Classification

The classification benchmarks includes 4 downstream task datasets, **VOC**, **ImageNet**, **iNaturalist2018** and **Places205**. If not specified, the results are Top-1 (%).

VOC SVM / Low-shot SVM

The **Best Layer** indicates that the best results are obtained from which layers feature map. For example, if the **Best Layer** is **feature3**, its best result is obtained from the second stage of ResNet (1 for stem layer, 2-5 for 4 stage layers).

Besides, k=1 to 96 indicates the hyper-parameter of Low-shot SVM.

ImageNet Linear Evaluation

The **Feature1 - Feature5** don't have the GlobalAveragePooling, the feature map is pooled to the specific dimensions and then follows a Linear layer to do the classification. Please refer to resnet50_mhead_linear-8xb32-steplr-90e_in1k for details of config.

The **AvgPool** result is obtained from Linear Evaluation with GlobalAveragePooling. Please refer to resnet50_linear-8xb512-coslr-90e_in1k for details of config.

Places205 Linear Evaluation

The **Feature1 - Feature5** don't have the GlobalAveragePooling, the feature map is pooled to the specific dimensions and then follows a Linear layer to do the classification. Please refer to resnet50_mhead_8xb32-steplr-28e_places205.py for details of config.

ImageNet Nearest-Neighbor Classification

The results are obtained from the features after GlobalAveragePooling. Here, k=10 to 200 indicates different number of nearest neighbors.

14.2.2 Detection

The detection benchmarks includes 2 downstream task datasets, **Pascal VOC 2007 + 2012** and **COCO2017**. This benchmark follows the evluation protocols set up by MoCo.

Pascal VOC 2007 + 2012

Please refer to faster_rcnn_r50_c4_mstrain_24k_voc0712.py for details of config.

COCO2017

Please refer to mask_rcnn_r50_fpn_mstrain_1x_coco.py for details of config.

14.2.3 Segmentation

The segmentation benchmarks includes 2 downstream task datasets, **Cityscapes** and **Pascal VOC 2012 + Aug**. It follows the evluation protocols set up by MMSegmentation.

Pascal VOC 2012 + Aug

Please refer to fcn_r50-d8_512x512_20k_voc12aug.py for details of config.

14.3 Citation

```
@inproceedings{grill2020bootstrap,
  title={Bootstrap your own latent: A new approach to self-supervised learning},
  author={Grill, Jean-Bastien and Strub, Florian and Altch{\'e}, Florent and Tallec,␣
↪Corentin and Richemond, Pierre H and Buchatskaya, Elena and Doersch, Carl and Pires,
↪ Bernardo Avila and Guo, Zhaohan Daniel and Azar, Mohammad Gheshlaghi and others},
  booktitle={NeurIPS},
  year={2020}
}
```

CHAPTER 15

DeepCluster

Deep Clustering for Unsupervised Learning of Visual Features

15.1 Abstract

Clustering is a class of unsupervised learning methods that has been extensively applied and studied in computer vision. Little work has been done to adapt it to the end-to-end training of visual features on large scale datasets. In this work, we present DeepCluster, a clustering method that jointly learns the parameters of a neural network and the cluster assignments of the resulting features. DeepCluster iteratively groups the features with a standard clustering algorithm, k-means, and uses the subsequent assignments as supervision to update the weights of the network.

15.2 Results and Models

Back to model_zoo.md to download models.

In this page, we provide benchmarks as much as possible to evaluate our pre-trained models. If not mentioned, all models are pre-trained on ImageNet-1k dataset.

15.2.1 Classification

The classification benchmarks includes 4 downstream task datasets, **VOC**, **ImageNet**, **iNaturalist2018** and **Places205**. If not specified, the results are Top-1 (%).

VOC SVM / Low-shot SVM

The **Best Layer** indicates that the best results are obtained from which layers feature map. For example, if the **Best Layer** is **feature3**, its best result is obtained from the second stage of ResNet (1 for stem layer, 2-5 for 4 stage layers).

Besides, k=1 to 96 indicates the hyper-parameter of Low-shot SVM.

ImageNet Linear Evaluation

The **Feature1 - Feature5** don't have the GlobalAveragePooling, the feature map is pooled to the specific dimensions and then follows a Linear layer to do the classification. Please refer to resnet50_mhead_linear-8xb32-steplr-90e_in1k for details of config.

The **AvgPool** result is obtained from Linear Evaluation with GlobalAveragePooling. Please refer to resnet50_linear-8xb32-steplr-100e_in1k for details of config.

Places205 Linear Evaluation

The **Feature1 - Feature5** don't have the GlobalAveragePooling, the feature map is pooled to the specific dimensions and then follows a Linear layer to do the classification. Please refer to resnet50_mhead_8xb32-steplr-28e_places205.py for details of config.

15.3 Citation

```
@inproceedings{caron2018deep,
  title={Deep clustering for unsupervised learning of visual features},
  author={Caron, Mathilde and Bojanowski, Piotr and Joulin, Armand and Douze,␣
↪Matthijs},
  booktitle={ECCV},
  year={2018}
}
```

CHAPTER 16

DenseCL

Dense Contrastive Learning for Self-Supervised Visual Pre-Training

16.1 Abstract

To date, most existing self-supervised learning methods are designed and optimized for image classification. These pre-trained models can be sub-optimal for dense prediction tasks due to the discrepancy between image-level prediction and pixel-level prediction. To fill this gap, we aim to design an effective, dense self-supervised learning method that directly works at the level of pixels (or local features) by taking into account the correspondence between local features. We present dense contrastive learning (DenseCL), which implements self-supervised learning by optimizing a pairwise contrastive (dis)similarity loss at the pixel level between two views of input images.

16.2 Results and Models

Back to model_zoo.md to download models.

In this page, we provide benchmarks as much as possible to evaluate our pre-trained models. If not mentioned, all models are pre-trained on ImageNet-1k dataset.

16.2.1 Classification

The classification benchmarks includes 4 downstream task datasets, **VOC**, **ImageNet**, **iNaturalist2018** and **Places205**. If not specified, the results are Top-1 (%).

VOC SVM / Low-shot SVM

The **Best Layer** indicates that the best results are obtained from which layers feature map. For example, if the **Best Layer** is **feature3**, its best result is obtained from the second stage of ResNet (1 for stem layer, 2-5 for 4 stage layers).

Besides, k=1 to 96 indicates the hyper-parameter of Low-shot SVM.

ImageNet Linear Evaluation

The **Feature1 - Feature5** don't have the GlobalAveragePooling, the feature map is pooled to the specific dimensions and then follows a Linear layer to do the classification. Please refer to resnet50_mhead_linear-8xb32-steplr-90e_in1k for details of config.

The **AvgPool** result is obtained from Linear Evaluation with GlobalAveragePooling. Please refer to resnet50_linear-8xb32-steplr-100e_in1k for details of config.

Places205 Linear Evaluation

The **Feature1 - Feature5** don't have the GlobalAveragePooling, the feature map is pooled to the specific dimensions and then follows a Linear layer to do the classification. Please refer to resnet50_mhead_8xb32-steplr-28e_places205.py for details of config.

ImageNet Nearest-Neighbor Classification

The results are obtained from the features after GlobalAveragePooling. Here, k=10 to 200 indicates different number of nearest neighbors.

16.2.2 Detection

The detection benchmarks includes 2 downstream task datasets, **Pascal VOC 2007 + 2012** and **COCO2017**. This benchmark follows the evluation protocols set up by MoCo.

Pascal VOC 2007 + 2012

Please refer to faster_rcnn_r50_c4_mstrain_24k_voc0712.py for details of config.

COCO2017

Please refer to mask_rcnn_r50_fpn_mstrain_1x_coco.py for details of config.

16.2.3 Segmentation

The segmentation benchmarks includes 2 downstream task datasets, **Cityscapes** and **Pascal VOC 2012 + Aug**. It follows the evluation protocols set up by MMSegmentation.

Pascal VOC 2012 + Aug

Please refer to fcn_r50-d8_512x512_20k_voc12aug.py for details of config.

16.3 Citation

```
@inproceedings{wang2021dense,
  title={Dense contrastive learning for self-supervised visual pre-training},
  author={Wang, Xinlong and Zhang, Rufeng and Shen, Chunhua and Kong, Tao and Li, Lei}
↪,
  booktitle={CVPR},
  year={2021}
}
```

CHAPTER 17

MoCo v2

Improved Baselines with Momentum Contrastive Learning

17.1 Abstract

Contrastive unsupervised learning has recently shown encouraging progress, e.g., in Momentum Contrast (MoCo) and SimCLR. In this note, we verify the effectiveness of two of SimCLR' s design improvements by implementing them in the MoCo framework. With simple modifications to MoCo—namely, using an MLP projection head and more data augmentation—we establish stronger baselines that outperform SimCLR and do not require large training batches. We hope this will make state-of-the-art unsupervised learning research more accessible.

17.2 Results and Models

Back to model_zoo.md to download models.

In this page, we provide benchmarks as much as possible to evaluate our pre-trained models. If not mentioned, all models are pre-trained on ImageNet-1k dataset.

17.2.1 Classification

The classification benchmarks includes 4 downstream task datasets, **VOC**, **ImageNet**, **iNaturalist2018** and **Places205**. If not specified, the results are Top-1 (%).

VOC SVM / Low-shot SVM

The **Best Layer** indicates that the best results are obtained from which layers feature map. For example, if the **Best Layer** is **feature3**, its best result is obtained from the second stage of ResNet (1 for stem layer, 2-5 for 4 stage layers).

Besides, k=1 to 96 indicates the hyper-parameter of Low-shot SVM.

ImageNet Linear Evaluation

The **Feature1 - Feature5** don't have the GlobalAveragePooling, the feature map is pooled to the specific dimensions and then follows a Linear layer to do the classification. Please refer to resnet50_mhead_linear-8xb32-steplr-90e_in1k for details of config.

The **AvgPool** result is obtained from Linear Evaluation with GlobalAveragePooling. Please refer to resnet50_linear-8xb32-steplr-100e_in1k for details of config.

Places205 Linear Evaluation

The **Feature1 - Feature5** don't have the GlobalAveragePooling, the feature map is pooled to the specific dimensions and then follows a Linear layer to do the classification. Please refer to resnet50_mhead_8xb32-steplr-28e_places205.py for details of config.

ImageNet Nearest-Neighbor Classification

The results are obtained from the features after GlobalAveragePooling. Here, k=10 to 200 indicates different number of nearest neighbors.

17.2.2 Detection

The detection benchmarks includes 2 downstream task datasets, **Pascal VOC 2007 + 2012** and **COCO2017**. This benchmark follows the evluation protocols set up by MoCo.

Pascal VOC 2007 + 2012

Please refer to faster_rcnn_r50_c4_mstrain_24k_voc0712.py for details of config.

COCO2017

Please refer to mask_rcnn_r50_fpn_mstrain_1x_coco.py for details of config.

17.2.3 Segmentation

The segmentation benchmarks includes 2 downstream task datasets, **Cityscapes** and **Pascal VOC 2012 + Aug**. It follows the evluation protocols set up by MMSegmentation.

Pascal VOC 2012 + Aug

Please refer to fcn_r50-d8_512x512_20k_voc12aug.py for details of config.

17.3 Citation

```
@article{chen2020improved,
  title={Improved baselines with momentum contrastive learning},
  author={Chen, Xinlei and Fan, Haoqi and Girshick, Ross and He, Kaiming},
  journal={arXiv preprint arXiv:2003.04297},
  year={2020}
}
```

CHAPTER 18

NPID

Unsupervised Feature Learning via Non-Parametric Instance Discrimination

18.1 Abstract

Neural net classifiers trained on data with annotated class labels can also capture apparent visual similarity among categories without being directed to do so. We study whether this observation can be extended beyond the conventional domain of supervised learning: Can we learn a good feature representation that captures apparent similar- ity among instances, instead of classes, by merely asking the feature to be discriminative of individual instances?

We formulate this intuition as a non-parametric classification problem at the instance-level, and use noise-contrastive estimation to tackle the computational challenges imposed by the large number of instance classes. Our experimental results demonstrate that, under unsupervised learning settings, our method surpasses the state-of-the-art on ImageNet classification by a large margin.

Our method is also remarkable for consistently improving test performance with more training data and better network architectures. By fine-tuning the learned feature, we further obtain competitive results for semi-supervised learning and object detection tasks. Our non-parametric model is highly compact: With 128 features per image, our method requires only 600MB storage for a million images, enabling fast nearest neighbour retrieval at the run time.

18.2 Results and Models

Back to model_zoo.md to download models.

In this page, we provide benchmarks as much as possible to evaluate our pre-trained models. If not mentioned, all models are pre-trained on ImageNet-1k dataset.

18.2.1 Classification

The classification benchmarks includes 4 downstream task datasets, **VOC**, **ImageNet**, **iNaturalist2018** and **Places205**. If not specified, the results are Top-1 (%).

VOC SVM / Low-shot SVM

The **Best Layer** indicates that the best results are obtained from which layers feature map. For example, if the **Best Layer** is **feature3**, its best result is obtained from the second stage of ResNet (1 for stem layer, 2-5 for 4 stage layers).

Besides, k=1 to 96 indicates the hyper-parameter of Low-shot SVM.

ImageNet Linear Evaluation

The **Feature1 - Feature5** don't have the GlobalAveragePooling, the feature map is pooled to the specific dimensions and then follows a Linear layer to do the classification. Please refer to resnet50_mhead_linear-8xb32-steplr-90e_in1k for details of config.

The **AvgPool** result is obtained from Linear Evaluation with GlobalAveragePooling. Please refer to resnet50_linear-8xb32-steplr-100e_in1k for details of config.

Places205 Linear Evaluation

The **Feature1 - Feature5** don't have the GlobalAveragePooling, the feature map is pooled to the specific dimensions and then follows a Linear layer to do the classification. Please refer to resnet50_mhead_8xb32-steplr-28e_places205.py for details of config.

ImageNet Nearest-Neighbor Classification

The results are obtained from the features after GlobalAveragePooling. Here, k=10 to 200 indicates different number of nearest neighbors.

18.2.2 Detection

The detection benchmarks includes 2 downstream task datasets, **Pascal VOC 2007 + 2012** and **COCO2017**. This benchmark follows the evluation protocols set up by MoCo.

Pascal VOC 2007 + 2012

Please refer to faster_rcnn_r50_c4_mstrain_24k_voc0712.py for details of config.

COCO2017

Please refer to mask_rcnn_r50_fpn_mstrain_1x_coco.py for details of config.

18.2.3 Segmentation

The segmentation benchmarks includes 2 downstream task datasets, **Cityscapes** and **Pascal VOC 2012 + Aug**. It follows the evluation protocols set up by MMSegmentation.

Pascal VOC 2012 + Aug

Please refer to fcn_r50-d8_512x512_20k_voc12aug.py for details of config.

18.3 Citation

```
@inproceedings{wu2018unsupervised,
  title={Unsupervised feature learning via non-parametric instance discrimination},
  author={Wu, Zhirong and Xiong, Yuanjun and Yu, Stella X and Lin, Dahua},
  booktitle={CVPR},
  year={2018}
}
```

CHAPTER 19

ODC

Online Deep Clustering for Unsupervised Representation Learning

19.1 Abstract

Joint clustering and feature learning methods have shown remarkable performance in unsupervised representation learning. However, the training schedule alternating between feature clustering and network parameters update leads to unstable learning of visual representations. To overcome this challenge, we propose Online Deep Clustering (ODC) that performs clustering and network update simultaneously rather than alternatingly. Our key insight is that the cluster centroids should evolve steadily in keeping the classifier stably updated. Specifically, we design and maintain two dynamic memory modules, i.e., samples memory to store samples' labels and features, and centroids memory for centroids evolution. We break down the abrupt global clustering into steady memory update and batch-wise label re-assignment. The process is integrated into network update iterations. In this way, labels and the network evolve shoulder-to-shoulder rather than alternatingly. Extensive experiments demonstrate that ODC stabilizes the training process and boosts the performance effectively.

19.2 Results and Models

Back to model_zoo.md to download models.

In this page, we provide benchmarks as much as possible to evaluate our pre-trained models. If not mentioned, all models are pre-trained on ImageNet-1k dataset.

19.2.1 Classification

The classification benchmarks includes 4 downstream task datasets, **VOC**, **ImageNet**, **iNaturalist2018** and **Places205**. If not specified, the results are Top-1 (%).

VOC SVM / Low-shot SVM

The **Best Layer** indicates that the best results are obtained from which layers feature map. For example, if the **Best Layer** is **feature3**, its best result is obtained from the second stage of ResNet (1 for stem layer, 2-5 for 4 stage layers).

Besides, k=1 to 96 indicates the hyper-parameter of Low-shot SVM.

ImageNet Linear Evaluation

The **Feature1 - Feature5** don't have the GlobalAveragePooling, the feature map is pooled to the specific dimensions and then follows a Linear layer to do the classification. Please refer to resnet50_mhead_linear-8xb32-steplr-90e_in1k for details of config.

The **AvgPool** result is obtained from Linear Evaluation with GlobalAveragePooling. Please refer to resnet50_linear-8xb32-steplr-100e_in1k for details of config.

Places205 Linear Evaluation

The **Feature1 - Feature5** don't have the GlobalAveragePooling, the feature map is pooled to the specific dimensions and then follows a Linear layer to do the classification. Please refer to resnet50_mhead_8xb32-steplr-28e_places205.py for details of config.

ImageNet Nearest-Neighbor Classification

The results are obtained from the features after GlobalAveragePooling. Here, k=10 to 200 indicates different number of nearest neighbors.

19.3 Citation

```
@inproceedings{zhan2020online,
  title={Online deep clustering for unsupervised representation learning},
  author={Zhan, Xiaohang and Xie, Jiahao and Liu, Ziwei and Ong, Yew-Soon and Loy,␣
↪Chen Change},
  booktitle={CVPR},
  year={2020}
}
```

CHAPTER 20

Relative Location

Unsupervised Visual Representation Learning by Context Prediction

20.1 Abstract

This work explores the use of spatial context as a source of free and plentiful supervisory signal for training a rich visual representation. Given only a large, unlabeled image collection, we extract random pairs of patches from each image and train a convolutional neural net to predict the position of the second patch relative to the first. We argue that doing well on this task requires the model to learn to recognize objects and their parts. We demonstrate that the feature representation learned using this within-image context indeed captures visual similarity across images. For example, this representation allows us to perform unsupervised visual discovery of objects like cats, people, and even birds from the Pascal VOC 2011 detection dataset. Furthermore, we show that the learned ConvNet can be used in the RCNN framework and provides a significant boost over a randomly-initialized ConvNet, resulting in state-of-the-art performance among algorithms which use only Pascal-provided training set annotations.

20.2 Results and Models

Back to model_zoo.md to download models.

In this page, we provide benchmarks as much as possible to evaluate our pre-trained models. If not mentioned, all models are pre-trained on ImageNet-1k dataset.

20.2.1 Classification

The classification benchmarks includes 4 downstream task datasets, **VOC**, **ImageNet**, **iNaturalist2018** and **Places205**. If not specified, the results are Top-1 (%).

VOC SVM / Low-shot SVM

The **Best Layer** indicates that the best results are obtained from which layers feature map. For example, if the **Best Layer** is **feature3**, its best result is obtained from the second stage of ResNet (1 for stem layer, 2-5 for 4 stage layers).

Besides, k=1 to 96 indicates the hyper-parameter of Low-shot SVM.

ImageNet Linear Evaluation

The **Feature1 - Feature5** don't have the GlobalAveragePooling, the feature map is pooled to the specific dimensions and then follows a Linear layer to do the classification. Please refer to resnet50_mhead_linear-8xb32-steplr-90e_in1k for details of config.

The **AvgPool** result is obtained from Linear Evaluation with GlobalAveragePooling. Please refer to resnet50_linear-8xb32-steplr-100e_in1k for details of config.

Places205 Linear Evaluation

The **Feature1 - Feature5** don't have the GlobalAveragePooling, the feature map is pooled to the specific dimensions and then follows a Linear layer to do the classification. Please refer to resnet50_mhead_8xb32-steplr-28e_places205.py for details of config.

ImageNet Nearest-Neighbor Classification

The results are obtained from the features after GlobalAveragePooling. Here, k=10 to 200 indicates different number of nearest neighbors.

20.2.2 Detection

The detection benchmarks includes 2 downstream task datasets, **Pascal VOC 2007 + 2012** and **COCO2017**. This benchmark follows the evluation protocols set up by MoCo.

Pascal VOC 2007 + 2012

Please refer to faster_rcnn_r50_c4_mstrain_24k_voc0712.py for details of config.

COCO2017

Please refer to mask_rcnn_r50_fpn_mstrain_1x_coco.py for details of config.

20.2.3 Segmentation

The segmentation benchmarks includes 2 downstream task datasets, **Cityscapes** and **Pascal VOC 2012 + Aug**. It follows the evluation protocols set up by MMSegmentation.

Pascal VOC 2012 + Aug

Please refer to fcn_r50-d8_512x512_20k_voc12aug.py for details of config.

20.3 Citation

```
@inproceedings{doersch2015unsupervised,
  title={Unsupervised visual representation learning by context prediction},
  author={Doersch, Carl and Gupta, Abhinav and Efros, Alexei A},
  booktitle={ICCV},
  year={2015}
}
```

CHAPTER 21

Rotation Prediction

Unsupervised Representation Learning by Predicting Image Rotation

21.1 Abstract

Over the last years, deep convolutional neural networks (ConvNets) have transformed the field of computer vision thanks to their unparalleled capacity to learn high level semantic image features. However, in order to successfully learn those features, they usually require massive amounts of manually labeled data, which is both expensive and impractical to scale. Therefore, unsupervised semantic feature learning, i.e., learning without requiring manual annotation effort, is of crucial importance in order to successfully harvest the vast amount of visual data that are available today. In our work we propose to learn image features by training ConvNets to recognize the 2d rotation that is applied to the image that it gets as input. We demonstrate both qualitatively and quantitatively that this apparently simple task actually provides a very powerful supervisory signal for semantic feature learning. We exhaustively evaluate our method in various unsupervised feature learning benchmarks and we exhibit in all of them state-of-the-art performance. Specifically, our results on those benchmarks demonstrate dramatic improvements w.r.t. prior state-of-the-art approaches in unsupervised representation learning and thus significantly close the gap with supervised feature learning.

21.2 Results and Models

Back to model_zoo.md to download models.

In this page, we provide benchmarks as much as possible to evaluate our pre-trained models. If not mentioned, all models are pre-trained on ImageNet-1k dataset.

21.2.1 Classification

The classification benchmarks includes 4 downstream task datasets, **VOC**, **ImageNet**, **iNaturalist2018** and **Places205**. If not specified, the results are Top-1 (%).

VOC SVM / Low-shot SVM

The **Best Layer** indicates that the best results are obtained from which layers feature map. For example, if the **Best Layer** is **feature3**, its best result is obtained from the second stage of ResNet (1 for stem layer, 2-5 for 4 stage layers).

Besides, k=1 to 96 indicates the hyper-parameter of Low-shot SVM.

ImageNet Linear Evaluation

The **Feature1 - Feature5** don't have the GlobalAveragePooling, the feature map is pooled to the specific dimensions and then follows a Linear layer to do the classification. Please refer to resnet50_mhead_linear-8xb32-steplr-90e_in1k for details of config.

The **AvgPool** result is obtained from Linear Evaluation with GlobalAveragePooling. Please refer to resnet50_linear-8xb32-steplr-100e_in1k.py for details of config.

Places205 Linear Evaluation

The **Feature1 - Feature5** don't have the GlobalAveragePooling, the feature map is pooled to the specific dimensions and then follows a Linear layer to do the classification. Please refer to resnet50_mhead_8xb32-steplr-28e_places205.py for details of config.

ImageNet Nearest-Neighbor Classification

The results are obtained from the features after GlobalAveragePooling. Here, k=10 to 200 indicates different number of nearest neighbors.

21.2.2 Detection

The detection benchmarks includes 2 downstream task datasets, **Pascal VOC 2007 + 2012** and **COCO2017**. This benchmark follows the evluation protocols set up by MoCo.

Pascal VOC 2007 + 2012

Please refer to faster_rcnn_r50_c4_mstrain_24k_voc0712.py for details of config.

COCO2017

Please refer to mask_rcnn_r50_fpn_mstrain_1x_coco.py for details of config.

21.2.3 Segmentation

The segmentation benchmarks includes 2 downstream task datasets, **Cityscapes** and **Pascal VOC 2012 + Aug**. It follows the evluation protocols set up by MMSegmentation.

Pascal VOC 2012 + Aug

Please refer to fcn_r50-d8_512x512_20k_voc12aug.py for details of config.

21.3 Citation

```
@inproceedings{komodakis2018unsupervised,
  title={Unsupervised representation learning by predicting image rotations},
  author={Komodakis, Nikos and Gidaris, Spyros},
  booktitle={ICLR},
  year={2018}
}
```

CHAPTER 22

SimCLR

A Simple Framework for Contrastive Learning of Visual Representations

22.1 Abstract

This paper presents SimCLR: a simple framework for contrastive learning of visual representations. We simplify recently proposed contrastive self-supervised learning algorithms without requiring specialized architectures or a memory bank. In order to understand what enables the contrastive prediction tasks to learn useful representations, we systematically study the major components of our framework. We show that (1) composition of data augmentations plays a critical role in defining effective predictive tasks, (2) introducing a learnable nonlinear transformation between the representation and the contrastive loss substantially improves the quality of the learned representations, and (3) contrastive learning benefits from larger batch sizes and more training steps compared to supervised learning. By combining these findings, we are able to considerably outperform previous methods for self-supervised and semi-supervised learning on ImageNet. A linear classifier trained on self-supervised representations learned by SimCLR achieves 76.5% top-1 accuracy, which is a 7% relative improvement over previous state-of-the-art, matching the performance of a supervised ResNet-50.

22.2 Results and Models

Back to model_zoo.md to download models.

In this page, we provide benchmarks as much as possible to evaluate our pre-trained models. If not mentioned, all models are pre-trained on ImageNet-1k dataset.

22.2.1 Classification

The classification benchmarks includes 4 downstream task datasets, **VOC**, **ImageNet**, **iNaturalist2018** and **Places205**. If not specified, the results are Top-1 (%).

VOC SVM / Low-shot SVM

The **Best Layer** indicates that the best results are obtained from which layers feature map. For example, if the **Best Layer** is **feature3**, its best result is obtained from the second stage of ResNet (1 for stem layer, 2-5 for 4 stage layers).

Besides, k=1 to 96 indicates the hyper-parameter of Low-shot SVM.

ImageNet Linear Evaluation

The **Feature1 - Feature5** don't have the GlobalAveragePooling, the feature map is pooled to the specific dimensions and then follows a Linear layer to do the classification. Please refer to resnet50_mhead_linear-8xb32-steplr-90e_in1k for details of config.

The **AvgPool** result is obtained from Linear Evaluation with GlobalAveragePooling. Please refer to resnet50_linear-8xb512-coslr-90e_in1k for details of config.

Places205 Linear Evaluation

The **Feature1 - Feature5** don't have the GlobalAveragePooling, the feature map is pooled to the specific dimensions and then follows a Linear layer to do the classification. Please refer to resnet50_mhead_8xb32-steplr-28e_places205.py for details of config.

ImageNet Nearest-Neighbor Classification

The results are obtained from the features after GlobalAveragePooling. Here, k=10 to 200 indicates different number of nearest neighbors.

22.2.2 Detection

The detection benchmarks includes 2 downstream task datasets, **Pascal VOC 2007 + 2012** and **COCO2017**. This benchmark follows the evluation protocols set up by MoCo.

Pascal VOC 2007 + 2012

Please refer to faster_rcnn_r50_c4_mstrain_24k_voc0712.py for details of config.

COCO2017

Please refer to mask_rcnn_r50_fpn_mstrain_1x_coco.py for details of config.

22.2.3 Segmentation

The segmentation benchmarks includes 2 downstream task datasets, **Cityscapes** and **Pascal VOC 2012 + Aug**. It follows the evluation protocols set up by MMSegmentation.

Pascal VOC 2012 + Aug

Please refer to fcn_r50-d8_512x512_20k_voc12aug.py for details of config.

22.3 Citation

```
@inproceedings{chen2020simple,
  title={A simple framework for contrastive learning of visual representations},
  author={Chen, Ting and Kornblith, Simon and Norouzi, Mohammad and Hinton, Geoffrey},
  booktitle={ICML},
  year={2020},
}
```

CHAPTER 23

SimSiam

Exploring Simple Siamese Representation Learning

23.1 Abstract

Siamese networks have become a common structure in various recent models for unsupervised visual representation learning. These models maximize the similarity between two augmentations of one image, subject to certain conditions for avoiding collapsing solutions. In this paper, we report surprising empirical results that simple Siamese networks can learn meaningful representations even using none of the following: (i) negative sample pairs, (ii) large batches, (iii) momentum encoders. Our experiments show that collapsing solutions do exist for the loss and structure, but a stop-gradient operation plays an essential role in preventing collapsing. We provide a hypothesis on the implication of stop-gradient, and further show proof-of-concept experiments verifying it. Our "SimSiam" method achieves competitive results on ImageNet and downstream tasks. We hope this simple baseline will motivate people to rethink the roles of Siamese architectures for unsupervised representation learning.

23.2 Results and Models

Back to model_zoo.md to download models.

In this page, we provide benchmarks as much as possible to evaluate our pre-trained models. If not mentioned, all models are pre-trained on ImageNet-1k dataset.

23.2.1 Classification

The classification benchmarks includes 4 downstream task datasets, **VOC**, **ImageNet**, **iNaturalist2018** and **Places205**. If not specified, the results are Top-1 (%).

VOC SVM / Low-shot SVM

The **Best Layer** indicates that the best results are obtained from which layers feature map. For example, if the **Best Layer** is **feature3**, its best result is obtained from the second stage of ResNet (1 for stem layer, 2-5 for 4 stage layers).

Besides, k=1 to 96 indicates the hyper-parameter of Low-shot SVM.

ImageNet Linear Evaluation

The **Feature1 - Feature5** don't have the GlobalAveragePooling, the feature map is pooled to the specific dimensions and then follows a Linear layer to do the classification. Please refer to resnet50_mhead_linear-8xb32-steplr-90e_in1k for details of config.

The **AvgPool** result is obtained from Linear Evaluation with GlobalAveragePooling. Please refer to resnet50_linear-8xb512-coslr-90e_in1k for details of config.

Places205 Linear Evaluation

The **Feature1 - Feature5** don't have the GlobalAveragePooling, the feature map is pooled to the specific dimensions and then follows a Linear layer to do the classification. Please refer to resnet50_mhead_8xb32-steplr-28e_places205.py for details of config.

ImageNet Nearest-Neighbor Classification

The results are obtained from the features after GlobalAveragePooling. Here, k=10 to 200 indicates different number of nearest neighbors.

23.2.2 Detection

The detection benchmarks includes 2 downstream task datasets, **Pascal VOC 2007 + 2012** and **COCO2017**. This benchmark follows the evluation protocols set up by MoCo.

Pascal VOC 2007 + 2012

Please refer to faster_rcnn_r50_c4_mstrain_24k_voc0712.py for details of config.

COCO2017

Please refer to mask_rcnn_r50_fpn_mstrain_1x_coco.py for details of config.

23.2.3 Segmentation

The segmentation benchmarks includes 2 downstream task datasets, **Cityscapes** and **Pascal VOC 2012 + Aug**. It follows the evluation protocols set up by MMSegmentation.

Pascal VOC 2012 + Aug

Please refer to fcn_r50-d8_512x512_20k_voc12aug.py for details of config.

23.3 Citation

```
@inproceedings{chen2021exploring,
  title={Exploring simple siamese representation learning},
  author={Chen, Xinlei and He, Kaiming},
  booktitle={CVPR},
  year={2021}
}
```

CHAPTER 24

SwAV

Unsupervised Learning of Visual Features by Contrasting Cluster Assignments

24.1 Abstract

Unsupervised image representations have significantly reduced the gap with supervised pretraining, notably with the recent achievements of contrastive learning methods. These contrastive methods typically work online and rely on a large number of explicit pairwise feature comparisons, which is computationally challenging. In this paper, we propose an online algorithm, SwAV, that takes advantage of contrastive methods without requiring to compute pairwise comparisons. Specifically, our method simultaneously clusters the data while enforcing consistency between cluster assignments produced for different augmentations (or "views") of the same image, instead of comparing features directly as in contrastive learning. Simply put, we use a "swapped" prediction mechanism where we predict the code of a view from the representation of another view. Our method can be trained with large and small batches and can scale to unlimited amounts of data. Compared to previous contrastive methods, our method is more memory efficient since it does not require a large memory bank or a special momentum network. In addition, we also propose a new data augmentation strategy, multi-crop, that uses a mix of views with different resolutions in place of two full-resolution views, without increasing the memory or compute requirements.

24.2 Results and Models

Back to model_zoo.md to download models.

In this page, we provide benchmarks as much as possible to evaluate our pre-trained models. If not mentioned, all models are pre-trained on ImageNet-1k dataset.

24.2.1 Classification

The classification benchmarks includes 4 downstream task datasets, **VOC**, **ImageNet**, **iNaturalist2018** and **Places205**. If not specified, the results are Top-1 (%).

VOC SVM / Low-shot SVM

The **Best Layer** indicates that the best results are obtained from which layers feature map. For example, if the **Best Layer** is **feature3**, its best result is obtained from the second stage of ResNet (1 for stem layer, 2-5 for 4 stage layers).

Besides, k=1 to 96 indicates the hyper-parameter of Low-shot SVM.

ImageNet Linear Evaluation

The **Feature1 - Feature5** don't have the GlobalAveragePooling, the feature map is pooled to the specific dimensions and then follows a Linear layer to do the classification. Please refer to resnet50_mhead_linear-8xb32-steplr-90e_in1k for details of config.

The **AvgPool** result is obtained from Linear Evaluation with GlobalAveragePooling. Please refer to resnet50_linear-8xb32-coslr-100e_in1k for details of config.

Places205 Linear Evaluation

The **Feature1 - Feature5** don't have the GlobalAveragePooling, the feature map is pooled to the specific dimensions and then follows a Linear layer to do the classification. Please refer to resnet50_mhead_8xb32-steplr-28e_places205.py for details of config.

ImageNet Nearest-Neighbor Classification

The results are obtained from the features after GlobalAveragePooling. Here, k=10 to 200 indicates different number of nearest neighbors.

24.2.2 Detection

The detection benchmarks includes 2 downstream task datasets, **Pascal VOC 2007 + 2012** and **COCO2017**. This benchmark follows the evluation protocols set up by MoCo.

Pascal VOC 2007 + 2012

Please refer to faster_rcnn_r50_c4_mstrain_24k_voc0712.py for details of config.

COCO2017

Please refer to mask_rcnn_r50_fpn_mstrain_1x_coco.py for details of config.

24.2.3 Segmentation

The segmentation benchmarks includes 2 downstream task datasets, **Cityscapes** and **Pascal VOC 2012 + Aug**. It follows the evluation protocols set up by MMSegmentation.

Pascal VOC 2012 + Aug

Please refer to fcn_r50-d8_512x512_20k_voc12aug.py for details of config.

24.3 Citation

```
@article{caron2020unsupervised,
  title={Unsupervised Learning of Visual Features by Contrasting Cluster Assignments},
  author={Caron, Mathilde and Misra, Ishan and Mairal, Julien and Goyal, Priya and␣
↪Bojanowski, Piotr and Joulin, Armand},
  booktitle={NeurIPS},
  year={2020}
}
```

CHAPTER 25

MoCo v3

An Empirical Study of Training Self-Supervised Vision Transformers

25.1 Abstract

This paper does not describe a novel method. Instead, it studies a straightforward, incremental, yet must-know baseline given the recent progress in computer vision: self-supervised learning for Vision Transformers (ViT). While the training recipes for standard convolutional networks have been highly mature and robust, the recipes for ViT are yet to be built, especially in the self-supervised scenarios where training becomes more challenging. In this work, we go back to basics and investigate the effects of several fundamental components for training self-supervised ViT. We observe that instability is a major issue that degrades accuracy, and it can be hidden by apparently good results. We reveal that these results are indeed partial failure, and they can be improved when training is made more stable. We benchmark ViT results in MoCo v3 and several other self-supervised frameworks, with ablations in various aspects. We discuss the currently positive evidence as well as challenges and open questions. We hope that this work will provide useful data points and experience for future research.

25.2 Results and Models

Back to model_zoo.md to download models.

In this page, we provide benchmarks as much as possible to evaluate our pre-trained models. If not mentioned, all models are pre-trained on ImageNet-1k dataset.

25.2.1 Classification

The classification benchmarks includes 4 downstream task datasets, **VOC**, **ImageNet**, **iNaturalist2018** and **Places205**. If not specified, the results are Top-1 (%).

ImageNet Linear Evaluation

The **Linear Evaluation** result is obtained by training a linear head upon the pre-trained backbone. Please refer to vit-small-p16_8xb128-coslr-90e_in1k for details of config.

25.3 Citation

```
@InProceedings{Chen_2021_ICCV,
    title     = {An Empirical Study of Training Self-Supervised Vision Transformers},
    author    = {Chen, Xinlei and Xie, Saining and He, Kaiming},
    booktitle = {Proceedings of the IEEE/CVF International Conference on Computer␣
↪Vision (ICCV)},
    year      = {2021}
}
```

CHAPTER 26

MAE

Masked Autoencoders Are Scalable Vision Learners

26.1 Abstract

This paper shows that masked autoencoders (MAE) are scalable self-supervised learners for computer vision. Our MAE approach is simple: we mask random patches of the input image and reconstruct the missing pixels. It is based on two core designs. First, we develop an asymmetric encoder-decoder architecture, with an encoder that operates only on the visible subset of patches (without mask tokens), along with a lightweight decoder that reconstructs the original image from the latent representation and mask tokens. Second, we find that masking a high proportion of the input image, e.g., 75%, yields a nontrivial and meaningful self-supervisory task. Coupling these two designs enables us to train large models efficiently and effectively: we accelerate training (by 3× or more) and improve accuracy. Our scalable approach allows for learning high-capacity models that generalize well: e.g., a vanilla ViT-Huge model achieves the best accuracy (87.8%) among methods that use only ImageNet-1K data. Transfer performance in downstream tasks outperforms supervised pretraining and shows promising scaling behavior.

26.2 Models and Benchmarks

Here, we report the results of the model, which is pre-trained on ImageNet1K for 400 epochs, the details are below:

26.3 Citation

```
@article{He2021MaskedAA,
  title={Masked Autoencoders Are Scalable Vision Learners},
  author={Kaiming He and Xinlei Chen and Saining Xie and Yanghao Li and
  Piotr Doll'ar and Ross B. Girshick},
  journal={ArXiv},
  year={2021}
}
```

CHAPTER 27

SimMIM

SimMIM: A Simple Framework for Masked Image Modeling

27.1 Abstract

This paper presents SimMIM, a simple framework for masked image modeling. We simplify recently proposed related approaches without special designs such as blockwise masking and tokenization via discrete VAE or clustering. To study what let the masked image modeling task learn good representations, we systematically study the major components in our framework, and find that simple designs of each component have revealed very strong representation learning performance: 1) random masking of the input image with a moderately large masked patch size (e.g., 32) makes a strong pre-text task; 2) predicting raw pixels of RGB values by direct regression performs no worse than the patch classification approaches with complex designs; 3) the prediction head can be as light as a linear layer, with no worse performance than heavier ones. Using ViT-B, our approach achieves 83.8% top-1 fine-tuning accuracy on ImageNet-1K by pre-training also on this dataset, surpassing previous best approach by +0.6%. When applied on a larger model of about 650 million parameters, SwinV2H, it achieves 87.1% top-1 accuracy on ImageNet-1K using only ImageNet-1K data. We also leverage this approach to facilitate the training of a 3B model (SwinV2-G), that by 40× less data than that in previous practice, we achieve the state-of-the-art on four representative vision benchmarks. The code and models will be publicly available at https: //github.com/microsoft/SimMIM .

27.2 Models and Benchmarks

Here, we report the results of the model, and more results will be coming soon.

27.3 Citation

```
@inproceedings{xie2021simmim,
  title={SimMIM: A Simple Framework for Masked Image Modeling},
  author={Xie, Zhenda and Zhang, Zheng and Cao, Yue and Lin, Yutong and Bao, Jianmin␣
↪and Yao, Zhuliang and Dai, Qi and Hu, Han},
  booktitle={International Conference on Computer Vision and Pattern Recognition␣
↪(CVPR)},
  year={2022}
}
```

CHAPTER 28

BarlowTwins

Barlow Twins: Self-Supervised Learning via Redundancy Reduction

28.1 Abstract

Self-supervised learning (SSL) is rapidly closing the gap with supervised methods on large computer vision benchmarks. A successful approach to SSL is to learn embeddings which are invariant to distortions of the input sample. However, a recurring issue with this approach is the existence of trivial constant solutions. Most current methods avoid such solutions by careful implementation details. We propose an objective function that naturally avoids collapse by measuring the cross-correlation matrix between the outputs of two identical networks fed with distorted versions of a sample, and making it as close to the identity matrix as possible. This causes the embedding vectors of distorted versions of a sample to be similar, while minimizing the redundancy between the components of these vectors. The method is called Barlow Twins, owing to neuroscientist H. Barlow's redundancy-reduction principle applied to a pair of identical networks. Barlow Twins does not require large batches nor asymmetry between the network twins such as a predictor network, gradient stopping, or a moving average on the weight updates. Intriguingly it benefits from very high-dimensional output vectors. Barlow Twins outperforms previous methods on ImageNet for semi-supervised classification in the low-data regime, and is on par with current state of the art for ImageNet classification with a linear classifier head, and for transfer tasks of classification and object detection.

28.2 Results and Models

Back to model_zoo.md to download models.

In this page, we provide benchmarks as much as possible to evaluate our pre-trained models. If not mentioned, all models are pre-trained on ImageNet-1k dataset.

28.2.1 Classification

The classification benchmarks includes 1 downstream task datasets, **ImageNet**. If not specified, the results are Top-1 (%).

ImageNet Linear Evaluation

The **Feature1 - Feature5** don't have the GlobalAveragePooling, the feature map is pooled to the specific dimensions and then follows a Linear layer to do the classification. Please refer to resnet50_mhead_8xb32-steplr-90e.py for details of config.

The **AvgPool** result is obtained from Linear Evaluation with GlobalAveragePooling. Please refer to resnet50_8xb32-steplr-100e_in1k.py for details of config.

ImageNet Nearest-Neighbor Classification

The results are obtained from the features after GlobalAveragePooling. Here, k=10 to 200 indicates different number of nearest neighbors.

28.3 Citation

```
@inproceedings{zbontar2021barlow,
  title={Barlow twins: Self-supervised learning via redundancy reduction},
  author={Zbontar, Jure and Jing, Li and Misra, Ishan and LeCun, Yann and Deny, St{\
↪'e}phane},
  booktitle={International Conference on Machine Learning},
  year={2021},
}
```

CHAPTER 29

CAE

Context Autoencoder for Self-Supervised Representation Learning

29.1 Abstract

We present a novel masked image modeling (MIM) approach, context autoencoder (CAE), for self-supervised learning. We randomly partition the image into two sets: visible patches and masked patches. The CAE architecture consists of: (i) an encoder that takes visible patches as input and outputs their latent representations, (ii) a latent context regressor that predicts the masked patch representations from the visible patch representations that are not updated in this regressor, (iii) a decoder that takes the estimated masked patch representations as input and makes predictions for the masked patches, and (iv) an alignment module that aligns the masked patch representation estimation with the masked patch representations computed from the encoder. In comparison to previous MIM methods that couple the encoding and decoding roles, e.g., using a single module in BEiT, our approach attempts to separate the encoding role (content understanding) from the decoding role (making predictions for masked patches) using different modules, improving the content understanding capability. In addition, our approach makes predictions from the visible patches to the masked patches in the latent representation space that is expected to take on semantics. In addition, we present the explanations about why contrastive pretraining and supervised pretraining perform similarly and why MIM potentially performs better. We demonstrate the effectiveness of our CAE through superior transfer performance in downstream tasks: semantic segmentation, and object detection and instance segmentation.

29.2 Prerequisite

Create a new folder `cae_ckpt` under the root directory and download the weights for `dalle` encoder to that folder

29.3 Models and Benchmarks

Here, we report the results of the model, which is pre-trained on ImageNet-1k for 300 epochs, the details are below:

29.4 Citation

```
@article{CAE,
  title={Context Autoencoder for Self-Supervised Representation Learning},
  author={Xiaokang Chen, Mingyu Ding, Xiaodi Wang, Ying Xin, Shentong Mo,
  Yunhao Wang, Shumin Han, Ping Luo, Gang Zeng, Jingdong Wang},
  journal={ArXiv},
  year={2022}
}
```

CHAPTER 30

更新日志

30.1 MMSelfSup

30.1.1 v0.9.2 (28/07/2022)

New Features

- 支持 MAE 重建图像的可视化 (#376)

Bug Fixes

- 修复 extract.py 文件中 cfg/args 路径问题，应用 cfg 中的路径进行处理 (#357)
- 修复 SimMIM 配置文件中掩码生成器类型名称的错误 (#360)

Improvements

- 更新 mdformat 设置 (#323)
- 添加 circle ci 配置 (#374)

Docs

- 修复语言更换链接问题 (#327)
- 更新 tutorials/4_schedule.md 中的文档链接 (#354)

30.1.2 v0.9.1 (31/05/2022)

亮点

- 更新 **BYOL** 模型和结果 (#319)
- 改进部分文档

新特性

- 更新 **BYOL** 模型和结果 (#319)

Bug 修复

- 对于 CAE 和 MAE 设置 qkv 偏置参数 (#303)
- 修复 MAE 配置文件拼写错误 (#307)

改进

- 修改文件名 (#304)
- 应用 mdformat (#311)

文档

- 改正教程中的打字错误 (#308)
- 配置 Myst-parser (#309)
- 更新文档算法简介 (#310)
- 改进安装文档 (#317)
- 改进首页 README (#318)

30.1.3 v0.9.0 (29/04/2022)

亮点

- 支持 **CAE** (#284)
- 支持 **Barlow Twins** (#207)

新特性

- 支持 CAE (#284)
- 支持 Barlow twins (#207)
- 增加 SimMIM 192 预训练及 224 微调的结果 (#280)
- 增加 MAE fp16 预训练设置 (#271)

Bug 修复

- 修复参数问题 (#290)
- 在 MAE 配置中修改 imgs_per_gpu 为 samples_per_gpu (#278)
- 使用 prefetch dataloader 时避免 GPU 内存溢出 (#277)
- 修复在注册自定义钩子时键值错误的问题 (#273)

改进

- 更新 SimCLR 模型和结果 (#295)
- 单元测试减少内存使用 (#291)
- 去除 pytorch 1.5 测试 (#288)
- 重命名线性评估配置文件 (#281)
- 为 api 增加单元测试 (#276)

文档

- 在模型库增加 SimMIM 并修复链接 (#272)

30.1.4 v0.8.0 (31/03/2022)

亮点

- 支持 **SimMIM** (#239)
- 增加 **KNN** 基准测试，支持中间 checkpoint 和提取的 backbone 权重进行评估 (#243)
- 支持 ImageNet-21k 数据集 (#225)

新特性

- 支持 SimMIM (#239)
- 增加 KNN 基准测试，支持中间 checkpoint 和提取的 backbone 权重进行评估 (#243)
- 支持 ImageNet-21k 数据集 (#225)
- 支持自动继续 checkpoint 文件的训练 (#245)

Bug 修复

- 在分布式 sampler 中增加种子 (#250)
- 修复 dist_test_svm_epoch.sh 中参数位置问题 (#260)
- 修复 prepare_voc07_cls.sh 中 mkdir 潜在错误 (#261)

改进

- 更新命令行参数模式 (#253)

文档

- 修复 6_benchmarks.md 中命令文档 (#263)
- 翻译 6_benchmarks.md 到中文 (#262)

30.1.5 v0.7.0 (03/03/2022)

亮点

- 支持 MAE 算法 (#221)
- 增加 Places205 下游基准测试 (#210)
- 在 CI 工作流中添加 Windows 测试 (#215)

新特性

- 支持 MAE 算法 (#221)
- 增加 Places205 下游基准测试 (#210)

Bug 修复

- 修复部分配置文件中的错误 (#200)
- 修复图像读取通道问题并更新相关结果 (#210)
- 修复在使用 prefetch 时，部分 dataset 输出格式不匹配的问题 (#218)
- 修复 t-sne ‘no init_cfg’的错误 (#222)

改进

- 配置文件中弃用 `imgs_per_gpu`，改用 `samples_per_gpu` (#204)
- 更新 MMCV 的安装方式 (#208)
- 为算法 readme 和代码版权增加 pre-commit 钩子 (#213)
- 在 CI 工作流中添加 Windows 测试 (#215)

文档

- 将 0_config.md 翻译成中文 (#216)
- 更新主页 OpenMMLab 项目和介绍 (#219)

30.1.6 v0.6.0 (02/02/2022)

亮点

- 支持基于 vision transformer 的 MoCo v3 (#194)
- 加速训练和启动时间 (#181)
- 支持 cpu 训练 (#188)

新特性

- 支持基于 vision transformer 的 MoCo v3 (#194)
- 支持 cpu 训练 (#188)

Bug 修复

- 修复问题 (#159, #160) 中提到的相关 bugs (#161)
- 修复 `RandomAppliedTrans` 中缺失的 prob 赋值 (#173)
- 修复 k-means losses 显示的 bug (#182)
- 修复非分布式多 gpu 训练/测试中的 bug (#189)
- 修复加载 cifar 数据集时的 bug (#191)
- 修复 `dataset.evaluate` 的参数 bug (#192)

改进

- 取消之前在 CI 中未完成的运行 (#145)
- 增强 MIM 功能 (#152)
- 更改某些特定文件时跳过 CI (#154)
- 在构建 eval 优化器时添加 `drop_last` 选项 (#158)
- 弃用对“python setup.py test”的支持 (#174)
- 加速训练和启动时间 (#181)
- 升级 `isort` 到 5.10.1 (#184)

文档

- 重构文档目录结构 (#146)
- 修复 readthedocs (#148, #149, #153)
- 修复一些文档中的拼写错误和无效链接 (#155, #180, #195)
- 更新模型库里的训练日志和基准测试结果 (#157, #165, #195)
- 更新部分文档并翻译成中文 (#163, #164, #165, #166, #167, #168, #169, #172, #176, #178, #179)
- 更新算法 README 到新格式 (#177)

30.1.7 v0.5.0 (16/12/2021)

亮点

- 代码重构后发版。
- 添加 3 个新的自监督学习算法。
- 支持 MMDet 和 MMSeg 的基准测试。
- 添加全面的文档。

重构

- 合并冗余数据集文件。
- 适配新版 MMCV，去除旧版相关代码。
- 继承 MMCV BaseModule。
- 优化目录结构。
- 重命名所有配置文件。

新特性

- 添加 SwAV、SimSiam、DenseCL 算法。
- 添加 t-SNE 可视化工具。
- 支持 MMCV 版本 fp16。

基准

- 更多基准测试结果，包括分类、检测和分割。
- 支持下游任务中的一些新数据集。
- 使用 MIM 启动 MMDet 和 MMSeg 训练。

文档

- 重构 README、getting_started、install、model_zoo 文档。
- 添加数据准备文档。
- 添加全面的教程。

30.2 OpenSelfSup (历史)

30.2.1 v0.3.0 (14/10/2020)

亮点

- 支持混合精度训练。
- 改进 GaussianBlur 使训练速度加倍。
- 更多基准测试结果。

Bug 修复

- 修复 moco v2 中的 bugs，现在结果可复现。
- 修复 byol 中的 bugs。

新特性

- 混合精度训练。
- 改进 GaussianBlur 使 MoCo V2、SimCLR、BYOL 的训练速度加倍。
- 更多基准测试结果，包括 Places、VOC、COCO。

30.2.2 v0.2.0 (26/6/2020)

亮点

- 支持 BYOL。
- 支持半监督基准测试。

Bug 修复

- 修复 publish_model.py 中的哈希 id。

新特性

- 支持 BYOL。
- 在线性和半监督评估中将训练和测试脚本分开。
- 支持半监督基准测试：benchmarks/dist_train_semi.sh。
- 将基准测试相关的配置文件移动到 configs/benchmarks/。
- 提供基准测试结果和模型下载链接。
- 支持每隔几次迭代更新网络。
- 支持带有 Nesterov 的 LARS 优化器。
- 支持 SimCLR 和 BYOL 从 LARS 适应和权重衰减中排除特定参数的需求。

CHAPTER 31

MMSelfSup 和 OpenSelfSup 的不同点

该文件记录了最新版本 MMSelfSup 和旧版以及 OpenSelfSup 之间的区别。

MMSelfSup 进行了重构并解决了许多遗留问题，它与 OpenSelfSup 并不兼容，如旧的配置文件需要更新，因为某些类或组件的名称已被修改。

主要的不同点为：代码库约定，模块化设计。

31.1 模块化设计

为了构建更加清晰的目录结构，MMSelfSup 重新设计了一些模块。

31.1.1 数据集

- MMSelfSup 合并了部分数据集，减少了冗余代码.
  - Classification, Extraction, NPID -> OneViewDataset
  - BYOL, Contrastive -> MultiViewDataset
- 重构了 `data_sources` 文件夹，现在的数据读取函数更加鲁棒。

另外，该部分仍然在重构中，会在接下里的某一版本中发布。

31.1.2 模型

- 注册机制已经更新。现在，models 文件夹下的各部分在构建时会有一个从 MMCV 中引入的父类 MMCV_MODELS。请查阅 mmselfsup/models/builder.py 和 mmcv/utils/registry.py 获取更多信息。
- models 文件夹包含 algorithms，backbones，necks，heads，memories 和一些所需的工具。algorithms 部分集成了其他主要的组件来构建自监督学习算法，就像 MMCls 中的 classifiers 或者 MMDet 中的 detectors。
- 在 OpenSelfSup 中，necks 的命名会有一些混乱并且实现在同一个 python 文件中。现在，necks 部分已经被重构，通过一个文件进行归类管理，并进行重新命名。请查阅 mmselfsup/models/necks 获取更多信息。

31.2 代码库约定

由于 OpenSelfSup 很久没有更新，MMSelfSup 更新了代码库约定。

31.2.1 配置文件

- MMSelfSup 对所有配置文件进行了重命名，并制定了命名规范，请查阅*0_config* 获取更多信息。
- 在配置文件中，一些类的参数命名或组件名字已被修改。
  - 一个算法名被修改：MOCO -> MoCo
  - 由于所有模型的组件继承自 MMCV 的 BaseModule，模型根据 init_cfg 进行初始化。请您依照该规范进行初始化设置，init_weights 仍然适用。
  - 请使用新的 necks 的命名来组合算，请在写配置文件前确认。
  - 归一化层通过 norm_cfg 进行管理。

31.2.2 脚本

- tools 的目录结构被修改，现在更加清晰，通过多个文件夹进行脚本分类和管理。例如，两个转换文件夹分为为了模型和数据格式。另外，和基准测试相关的脚本都在 benchmarks 文件夹中，它和 configs/benchmarks 拥有相同的目录结构。
- train.py 脚本中的参数已被更新，两个主要修改点为
  - 增加 --cfg-options 参数，通过命令行传参对配置文件进行修改。
  - 移除 --pretrained，通过 --cfg-options 设置预训练模型。

CHAPTER 32

English

CHAPTER 33

简体中文

CHAPTER 34

mmselfsup.apis

CHAPTER 35

mmselfsup.core

35.1 hooks

35.2 optimizer

CHAPTER 36

mmselfsup.datasets

36.1 data_sources

36.2 pipelines

36.3 samplers

36.4 datasets

CHAPTER 37

mmselfsup.models

CHAPTER 38

mmselfsup.utils

CHAPTER 39

Indices and tables

- genindex
- search