MMAction2

发布 *0.24.1*

MMAction2 Authors

2023 年 06 月 19 日

Contents

您可以在页面左下角切换中英文文档。

You can change the documentation language at the lower-left corner of the page.

CHAPTER 1

安装

本文档提供了安装 MMAction2 的相关步骤。

1.1 安装依赖包

- Linux (Windows 系统暂未有官方支持)
- Python 3.7+
- PyTorch 1.3+
- CUDA 9.2+ (如果要从源码对 PyTorch 进行编译, CUDA 9.0 版本同样可以兼容)

- GCC 5+
- mmcv 1.1.1+
- Numpy
- ffmpeg (4.2 版本最佳)
- decord (可选项, 0.4.1+)：使用 `pip install decord==0.4.1` 命令安装其 CPU 版本，GPU 版本需从源码进行编译。
- PyAV (可选项)：`conda install av -c conda-forge -y`。
- PyTurboJPEG (可选项)：`pip install PyTurboJPEG`。
- denseflow (可选项)：可参考 这里 获取简便安装步骤。
- moviepy (可选项)：`pip install moviepy`. 官方安装步骤可参考 这里。**特别地**，如果安装过程碰到 这个问题，可参考：
  1. 对于 Windows 用户，ImageMagick 将不会被 MoviePy 自动检测到，用户需要对 `moviepy/config_defaults.py` 文件进行修改，以提供 ImageMagick 的二进制文件（即，`magick`）的路径，如 `IMAGEMAGICK_BINARY = "C:\\Program Files\\ImageMagick_VERSION\\magick.exe"`
  2. 对于 Linux 用户，如果 ImageMagick 没有被 `moviepy` 检测到，用于需要对 `/etc/ImageMagick-6/policy.xml` 文件进行修改，把文件中的 `<policy domain="path" rights="none" pattern="@*" />` 代码行修改为 `<!-- <policy domain="path" rights="none" pattern="@*" /> -->`。
- Pillow-SIMD (可选项)：可使用如下脚本进行安装：

```
conda uninstall -y --force pillow pil jpeg libtiff libjpeg-turbo
pip   uninstall -y         pillow pil jpeg libtiff libjpeg-turbo
conda install -yc conda-forge libjpeg-turbo
CFLAGS="${CFLAGS} -mavx2" pip install --upgrade --no-cache-dir --force-reinstall --no-
↪binary :all: --compile pillow-simd
conda install -y jpeg libtiff
```

注意：用户需要首先运行 `pip uninstall mmcv` 命令，以确保 mmcv 被成功安装。如果 mmcv 和 mmcv-full 同时被安装, 会报 `ModuleNotFoundError` 的错误。

1.2 准备环境

a. 创建并激活 conda 虚拟环境，如：

```
conda create -n open-mmlab python=3.7 -y
conda activate open-mmlab
```

b. 根据 官方文档 进行 PyTorch 和 torchvision 的安装，如：

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

注：确保 CUDA 的编译版本和 CUDA 的运行版本相匹配。用户可以参照 PyTorch 官网 对预编译包所支持的 CUDA 版本进行核对。

例 1：如果用户的 `/usr/local/cuda` 文件夹下已安装 CUDA 10.1 版本，并且想要安装 PyTorch 1.5 版本，则需要安装 CUDA 10.1 下预编译的 PyTorch。

```
conda install pytorch cudatoolkit=10.1 torchvision -c pytorch
```

例 2：如果用户的 `/usr/local/cuda` 文件夹下已安装 CUDA 9.2 版本，并且想要安装 PyTorch 1.3.1 版本，则需要安装 CUDA 9.2 下预编译的 PyTorch。

```
conda install pytorch=1.3.1 cudatoolkit=9.2 torchvision=0.4.2 -c pytorch
```

如果 PyTorch 是由源码进行编译安装（而非直接下载预编译好的安装包），则可以使用更多的 CUDA 版本（如 9.0 版本）。

1.3 MMAction2 的安装步骤

这里推荐用户使用 MIM 安装 MMAction2。

```
pip install git+https://github.com/open-mmlab/mim.git
mim install mmaction2 -f https://github.com/open-mmlab/mmaction2.git
```

MIM 可以自动安装 OpenMMLab 项目及其依赖。

或者，用户也可以通过以下步骤手动安装 MMAction2。

a. 安装 mmcv-full，我们推荐您安装以下预构建包：

```
# pip install mmcv-full -f https://download.openmmlab.com/mmcv/dist/{cu_version}/
↪{torch_version}/index.html
pip install mmcv-full -f https://download.openmmlab.com/mmcv/dist/cu102/torch1.10.0/
↪index.html
```

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

```
# 我们可以忽略 PyTorch 的小版本号
pip install mmcv-full -f https://download.openmmlab.com/mmcv/dist/cu102/torch1.10/
↪index.html
```

可查阅 这里 以参考不同版本的 MMCV 所兼容的 PyTorch 和 CUDA 版本。

另外，用户也可以通过使用以下命令从源码进行编译：

```
git clone https://github.com/open-mmlab/mmcv.git
cd mmcv
MMCV_WITH_OPS=1 pip install -e .  # mmcv-full 包含一些 cuda 算子，执行该步骤会安装 mmcv-full
(而非 mmcv)
# 或者使用 pip install -e .  # 这个命令安装的 mmcv 将不包含 cuda ops，通常适配 CPU (无 GPU) 环境
cd ..
```

或者直接运行脚本：

```
pip install mmcv-full
```

注意：如果 mmcv 已经被安装，用户需要使用 `pip uninstall mmcv` 命令进行卸载。如果 mmcv 和 mmcv-full 同时被安装，会报 `ModuleNotFoundError` 的错误。

b. 克隆 MMAction2 库。

```
git clone https://github.com/open-mmlab/mmaction2.git
cd mmaction2
```

c. 安装依赖包和 MMAction2。

```
pip install -r requirements/build.txt
pip install -v -e .  # or "python setup.py develop"
```

如果是在 macOS 环境安装 MMAction2，则需使用如下命令：

```
CC=clang CXX=clang++ CFLAGS='-stdlib=libc++' pip install -e .
```

d. 安装 mmdetection 以支持时空检测任务。

如果用户不想做时空检测相关任务，这部分步骤可以选择跳过。

可参考 这里 进行 mmdetection 的安装。

注意：

1. 在步骤 b 中，git commit 的 id 将会被写到版本号中，如 0.6.0+2e7045c。这个版本号也会被保存到训练好的模型中。这里推荐用户每次在步骤 b 中对本地代码和 github 上的源码进行同步。如果 C++/CUDA 代

码被修改，就必须进行这一步骤。

2. 根据上述步骤，MMAction2 就会以 dev 模式被安装，任何本地的代码修改都会立刻生效，不需要再重新安装一遍（除非用户提交了 commits，并且想更新版本号）。

3. 如果用户想使用 opencv-python-headless 而不是 opencv-python，可再安装 MMCV 前安装 opencv-python-headless。

4. 如果用户想使用 PyAV，可以通过 conda install av -c conda-forge -y 进行安装。

5. 一些依赖包是可选的。运行 python setup.py develop 将只会安装运行代码所需的最小要求依赖包。要想使用一些可选的依赖包，如 decord，用户需要通过 pip install -r requirements/optional.txt 进行安装，或者通过调用 pip（如 pip install -v -e .[optional]，这里的 [optional] 可替换为 all,tests,build 或 optional）指定安装对应的依赖包，如 pip install -v -e .[tests,build]。

1.4 CPU 环境下的安装步骤

MMAction2 可以在只有 CPU 的环境下安装（即无法使用 GPU 的环境）。

在 CPU 模式下，用户可以运行 demo/demo.py 的代码。

1.5 利用 Docker 镜像安装 MMAction2

MMAction2 提供一个 Dockerfile 用户创建 docker 镜像。

```
# 创建拥有 PyTorch 1.6.0, CUDA 10.1, CUDNN 7 配置的 docker 镜像.
docker build -f ./docker/Dockerfile --rm -t mmaction2 .
```

注意：用户需要确保已经安装了 nvidia-container-toolkit。

运行以下命令：

```
docker run --gpus all --shm-size=8g -it -v {DATA_DIR}:/mmaction2/data mmaction2
```

1.6 源码安装 MMAction2

这里提供了 conda 下安装 MMAction2 并链接数据集路径的完整脚本（假设 Kinetics-400 数据的路径在 $KINETICS400_ROOT）。

```
conda create -n open-mmlab python=3.7 -y
conda activate open-mmlab
```

(下页继续)

(续上页)

```
# 安装最新的，使用默认版本的 CUDA 版本（一般为最新版本）预编译的 PyTorch 包
conda install -c pytorch pytorch torchvision -y

# 安装最新版本的 mmcv 或 mmcv-full，这里以 mmcv 为例
pip install mmcv

# 安装 mmaction2
git clone https://github.com/open-mmlab/mmaction2.git
cd mmaction2
pip install -r requirements/build.txt
python setup.py develop

mkdir data
ln -s $KINETICS400_ROOT data
```

1.7 在多个 MMAction2 版本下进行开发

MMAction2 的训练和测试脚本已经修改了 PYTHONPATH 变量，以确保其能够运行当前目录下的 MMAction2。

如果想要运行环境下默认的 MMAction2，用户需要在训练和测试脚本中去除这一行：

```
PYTHONPATH="$(dirname $0)/..":$PYTHONPATH
```

1.8 安装验证

为了验证 MMAction2 和所需的依赖包是否已经安装成功，用户可以运行以下的 python 代码，以测试其是否能成功地初始化动作识别器，并进行演示视频的推理：

```
import torch
from mmaction.apis import init_recognizer, inference_recognizer

config_file = 'configs/recognition/tsn/tsn_r50_video_inference_1x1x3_100e_kinetics400_
↪rgb.py'
device = 'cuda:0' # 或 'cpu'
device = torch.device(device)

model = init_recognizer(config_file, device=device)
# 进行演示视频的推理
inference_recognizer(model, 'demo/demo.mp4')
```

CHAPTER 2

基础教程

本文档提供 MMAction2 相关用法的基本教程。对于安装说明，请参阅安装指南。

2.1 数据集

MMAction2 建议用户将数据集根目录链接到 `$MMACTION2/data` 下。如果用户的文件夹结构与默认结构不同，则需要在配置文件中进行对应路径的修改。

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
│   ├── kinetics400
│   │   ├── rawframes_train
│   │   ├── rawframes_val
│   │   ├── kinetics_train_list.txt
│   │   ├── kinetics_val_list.txt
│   ├── ucf101
│   │   ├── rawframes_train
│   │   ├── rawframes_val
│   │   ├── ucf101_train_list.txt
│   │   ├── ucf101_val_list.txt
│   ├── ...
```

请参阅*数据集准备* 获取数据集准备的相关信息。

对于用户自定义数据集的准备，请参阅*教程 3：如何增加新数据集*

2.2 使用预训练模型进行推理

MMAction2 提供了一些脚本用于测试数据集（如 Kinetics-400，Something-Something V1&V2，(Multi-)Moments in Time，等），并提供了一些高级 API，以便更好地兼容其他项目。

MMAction2 支持仅使用 CPU 进行测试。然而，这样做的速度**非常慢**，用户应仅使用其作为无 GPU 机器上的 debug 手段。如需使用 CPU 进行测试，用户需要首先使用命令 `export CUDA_VISIBLE_DEVICES=-1` 禁用机器上的 GPU （如有），然后使用命令 `python tools/test.py {OTHER_ARGS}` 直接调用测试脚本。

2.2.1 测试某个数据集

- [x] 支持单 GPU
- [x] 支持单节点，多 GPU
- [x] 支持多节点

用户可使用以下命令进行数据集测试

```
# 单 GPU 测试
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--out ${RESULT_FILE}] [--eval
↪${EVAL_METRICS}] \
    [--gpu-collect] [--tmpdir ${TMPDIR}] [--options ${OPTIONS}] [--average-clips $
↪{AVG_TYPE}] \
    [--launcher ${JOB_LAUNCHER}] [--local_rank ${LOCAL_RANK}] [--onnx] [--tensorrt]

# 多 GPU 测试
./tools/dist_test.sh ${CONFIG_FILE} ${CHECKPOINT_FILE} ${GPU_NUM} [--out ${RESULT_
↪FILE}] [--eval ${EVAL_METRICS}] \
    [--gpu-collect] [--tmpdir ${TMPDIR}] [--options ${OPTIONS}] [--average-clips $
↪{AVG_TYPE}] \
    [--launcher ${JOB_LAUNCHER}] [--local_rank ${LOCAL_RANK}]
```

可选参数:

- `RESULT_FILE`：输出结果文件名。如果没有被指定，则不会保存测试结果。
- `EVAL_METRICS`：测试指标。其可选值与对应数据集相关，如 `top_k_accuracy`，`mean_class_accuracy` 适用于所有动作识别数据集，`mmit_mean_average_precision` 适用于 Multi-Moments in Time 数据集，`mean_average_precision` 适用于 Multi-Moments in Time 和单类 HVU 数据集，`AR@AN` 适用于 ActivityNet 数据集等。
- `--gpu-collect`：如果被指定，动作识别结果将会通过 GPU 通信进行收集。否则，它将被存储到不同 GPU 上的 `TMPDIR` 文件夹中，并在 rank 0 的进程中被收集。
- `TMPDIR`：用于存储不同进程收集的结果文件的临时文件夹。该变量仅当 `--gpu-collect` 没有被指定时有效。
- `OPTIONS`：用于验证过程的自定义选项。其可选值与对应数据集的 `evaluate` 函数变量有关。
- `AVG_TYPE`：用于平均测试片段结果的选项。如果被设置为 `prob`，则会在平均测试片段结果之前施加 softmax 函数。否则，会直接进行平均。
- `JOB_LAUNCHER`：分布式任务初始化启动器选项。可选值有 `none`，`pytorch`，`slurm`，`mpi`。特别地，如果被设置为 `none`, 则会以非分布式模式进行测试。
- `LOCAL_RANK`：本地 rank 的 ID。如果没有被指定，则会被设置为 0。
- `--onnx`: 如果指定，将通过 onnx 模型推理获取预测结果，输入参数 `CHECKPOINT_FILE` 应为 onnx 模型文件。Onnx 模型文件由 `/tools/deployment/pytorch2onnx.py` 脚本导出。目前，不支持多 GPU 测试以及动态张量形状（Dynamic shape）。请注意，数据集输出与模型输入张量的形状应保持一致。同时，不建议使用测试时数据增强，如 `ThreeCrop`，`TenCrop`，`twice_sample` 等。
- `--tensorrt`: 如果指定，将通过 TensorRT 模型推理获取预测结果，输入参数 `CHECKPOINT_FILE` 应为 TensorRT 模型文件。TensorRT 模型文件由导出的 onnx 模型以及 TensorRT 官方模型转换工具生成。目前，不支持多 GPU 测试以及动态张量形状（Dynamic shape）。请注意，数据集输出与模型输入张量的形状应保持一致。同时，不建议使用测试时数据增强，如 `ThreeCrop`，`TenCrop`，`twice_sample`

等。

例子:

假定用户将下载的模型权重文件放置在 `checkpoints/` 目录下。

1. 在 Kinetics-400 数据集下测试 TSN （不存储测试结果为文件），并验证 `top-k accuracy` 和 `mean class accuracy` 指标

```
python tools/test.py configs/recognition/tsn/tsn_r50_1x1x3_100e_kinetics400_rgb.
↪py \
    checkpoints/SOME_CHECKPOINT.pth \
    --eval top_k_accuracy mean_class_accuracy
```

2. 使用 8 块 GPU 在 Something-Something V1 下测试 TSN，并验证 `top-k accuracy` 指标

```
./tools/dist_test.sh configs/recognition/tsn/tsn_r50_1x1x8_50e_sthv1_rgb.py \
    checkpoints/SOME_CHECKPOINT.pth \
    8 --out results.pkl --eval top_k_accuracy
```

3. 在 slurm 分布式环境中测试 TSN 在 Kinetics-400 数据集下的 `top-k accuracy` 指标

```
python tools/test.py configs/recognition/tsn/tsn_r50_1x1x3_100e_kinetics400_rgb.
↪py \
    checkpoints/SOME_CHECKPOINT.pth \
    --launcher slurm --eval top_k_accuracy
```

4. 在 Something-Something V1 下测试 onnx 格式的 TSN 模型，并验证 `top-k accuracy` 指标

```
python tools/test.py configs/recognition/tsn/tsn_r50_1x1x3_100e_kinetics400_rgb.
↪py \
    checkpoints/SOME_CHECKPOINT.onnx \
    --eval top_k_accuracy --onnx
```

2.2.2 使用高级 API 对视频和帧文件夹进行测试

这里举例说明如何构建模型并测试给定视频

```
import torch

from mmaction.apis import init_recognizer, inference_recognizer

config_file = 'configs/recognition/tsn/tsn_r50_video_inference_1x1x3_100e_kinetics400_
↪rgb.py'
# 从模型库中下载检测点，并把它放到 `checkpoints/` 文件夹下
checkpoint_file = 'checkpoints/tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.
↪pth'
```

(下页继续)

(续上页)

```
# 指定设备
device = 'cuda:0' # or 'cpu'
device = torch.device(device)

 # 根据配置文件和检查点来建立模型
model = init_recognizer(config_file, checkpoint_file, device=device)

# 测试单个视频并显示其结果
video = 'demo/demo.mp4'
labels = 'tools/data/kinetics/label_map_k400.txt'
results = inference_recognizer(model, video)

# 显示结果
labels = open('tools/data/kinetics/label_map_k400.txt').readlines()
labels = [x.strip() for x in labels]
results = [(labels[k[0]], k[1]) for k in results]

print(f'The top-5 labels with corresponding scores are:')
for result in results:
    print(f'{result[0]}: ', result[1])
```

这里举例说明如何构建模型并测试给定帧文件夹

```
import torch

from mmaction.apis import init_recognizer, inference_recognizer

config_file = 'configs/recognition/tsn/tsn_r50_inference_1x1x3_100e_kinetics400_rgb.py
↪'
# 从模型库中下载检测点，并把它放到 `checkpoints/` 文件夹下
checkpoint_file = 'checkpoints/tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.
↪pth'

# 指定设备
device = 'cuda:0' # or 'cpu'
device = torch.device(device)

 # 根据配置文件和检查点来建立模型
model = init_recognizer(config_file, checkpoint_file, device=device)

# 测试单个视频的帧文件夹并显示其结果
video = 'SOME_DIR_PATH/'
labels = 'tools/data/kinetics/label_map_k400.txt'
```

(下页继续)

(续上页)

```
results = inference_recognizer(model, video)

# 显示结果
labels = open('tools/data/kinetics/label_map_k400.txt').readlines()
labels = [x.strip() for x in labels]
results = [(labels[k[0]], k[1]) for k in results]

print(f'The top-5 labels with corresponding scores are:')
for result in results:
    print(f'{result[0]}: ', result[1])
```

这里举例说明如何构建模型并通过 url 测试给定视频

```
import torch

from mmaction.apis import init_recognizer, inference_recognizer

config_file = 'configs/recognition/tsn/tsn_r50_video_inference_1x1x3_100e_kinetics400_
↪rgb.py'
# 从模型库中下载检测点，并把它放到 `checkpoints/` 文件夹下
checkpoint_file = 'checkpoints/tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.
↪pth'

# 指定设备
device = 'cuda:0' # or 'cpu'
device = torch.device(device)

 # 根据配置文件和检查点来建立模型
model = init_recognizer(config_file, checkpoint_file, device=device)

# 测试单个视频的 url 并显示其结果
video = 'https://www.learningcontainer.com/wp-content/uploads/2020/05/sample-mp4-file.
↪mp4'
labels = 'tools/data/kinetics/label_map_k400.txt'
results = inference_recognizer(model, video)

# 显示结果
labels = open('tools/data/kinetics/label_map_k400.txt').readlines()
labels = [x.strip() for x in labels]
results = [(labels[k[0]], k[1]) for k in results]

print(f'The top-5 labels with corresponding scores are:')
for result in results:
    print(f'{result[0]}: ', result[1])
```

注意：MMAction2 在默认提供的推理配置文件（inference configs）中定义 `data_prefix` 变量，并将其设置为 None 作为默认值。如果 `data_prefix` 不为 None，则要获取的视频文件（或帧文件夹）的路径将为 `data_prefix/video`。在这里，`video` 是上述脚本中的同名变量。可以在 `rawframe_dataset.py` 文件和 `video_dataset.py` 文件中找到此详细信息。例如，

- 当视频（帧文件夹）路径为 `SOME_DIR_PATH/VIDEO.mp4`（`SOME_DIR_PATH/VIDEO_NAME/img_xxxxx.jpg`），并且配置文件中的 `data_prefix` 为 None，则 `video` 变量应为 `SOME_DIR_PATH/VIDEO.mp4`（`SOME_DIR_PATH/VIDEO_NAME`）。
- 当视频（帧文件夹）路径为 `SOME_DIR_PATH/VIDEO.mp4`（`SOME_DIR_PATH/VIDEO_NAME/img_xxxxx.jpg`），并且配置文件中的 `data_prefix` 为 `SOME_DIR_PATH`，则 `video` 变量应为 `VIDEO.mp4`（`VIDEO_NAME`）。
- 当帧文件夹路径为 `VIDEO_NAME/img_xxxxx.jpg`，并且配置文件中的 `data_prefix` 为 None，则 `video` 变量应为 `VIDEO_NAME`。
- 当传递参数为视频 url 而非本地路径，则需使用 OpenCV 作为视频解码后端。

在 demo/demo.ipynb 中有提供相应的 notebook 演示文件。

2.3 如何建立模型

2.3.1 使用基本组件建立模型

MMAction2 将模型组件分为 4 种基础模型：

- 识别器（recognizer）：整个识别器模型管道，通常包含一个主干网络（backbone）和分类头（cls_head）。
- 主干网络（backbone）：通常为一个用于提取特征的 FCN 网络，例如 ResNet，BNInception。
- 分类头（cls_head）：用于分类任务的组件，通常包括一个带有池化层的 FC 层。
- 时序检测器（localizer）：用于时序检测的模型，目前有的检测器包含 BSN，BMN，SSN。

用户可参照给出的配置文件里的基础模型搭建流水线（如 `Recognizer2D`）

如果想创建一些新的组件，如 TSM: Temporal Shift Module for Efficient Video Understanding 中的 temporal shift backbone 结构，则需：

1. 创建 `mmaction/models/backbones/resnet_tsm.py` 文件

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

@BACKBONES.register_module()
class ResNetTSM(ResNet):

  def __init__(self,
```

(下页继续)

(续上页)

```
                depth,
                num_segments=8,
                is_shift=True,
                shift_div=8,
                shift_place='blockres',
                temporal_pool=False,
                **kwargs):
        pass

    def forward(self, x):
        # implementation is ignored
        pass
```

2. 从 mmaction/models/backbones/__init__.py 中导入模型

```
from .resnet_tsm import ResNetTSM
```

3. 修改模型文件

```
backbone=dict(
    type='ResNet',
    pretrained='torchvision://resnet50',
    depth=50,
    norm_eval=False)
```

修改为

```
backbone=dict(
    type='ResNetTSM',
    pretrained='torchvision://resnet50',
    depth=50,
    norm_eval=False,
    shift_div=8)
```

2.3.2 构建新模型

要编写一个新的动作识别器流水线，用户需要继承 BaseRecognizer，其定义了如下抽象方法

- forward_train(): 训练模式下的前向方法
- forward_test(): 测试模式下的前向方法

具体可参照 Recognizer2D 和 Recognizer3D

2.4 如何训练模型

2.4.1 推理流水线

MMAction2 使用 `MMDistributedDataParallel` 进行分布式训练，使用 `MMDataParallel` 进行非分布式训练。

对于单机多卡与多台机器的情况，MMAction2 使用分布式训练。假设服务器有 8 块 GPU，则会启动 8 个进程，并且每台 GPU 对应一个进程。

每个进程拥有一个独立的模型，以及对应的数据加载器和优化器。模型参数同步只发生于最开始。之后，每经过一次前向与后向计算，所有 GPU 中梯度都执行一次 allreduce 操作，而后优化器将更新模型参数。由于梯度执行了 allreduce 操作，因此不同 GPU 中模型参数将保持一致。

2.4.2 训练配置

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

默认情况下，MMAction2 在每个周期后会在验证集上评估模型，可以通过在训练配置中修改 `interval` 参数来更改评估间隔

```
evaluation = dict(interval=5)  # 每 5 个周期进行一次模型评估
```

根据 Linear Scaling Rule，当 GPU 数量或每个 GPU 上的视频批大小改变时，用户可根据批大小按比例地调整学习率，如，当 4 GPUs x 2 video/gpu 时，lr=0.01；当 16 GPUs x 4 video/gpu 时，lr=0.08。

MMAction2 支持仅使用 CPU 进行训练。然而，这样做的速度**非常慢**，用户应仅使用其作为无 GPU 机器上的 debug 手段。如需使用 CPU 进行训练，用户需要首先使用命令 `export CUDA_VISIBLE_DEVICES=-1` 禁用机器上的 GPU（如有），然后使用命令 `python tools/train.py {OTHER_ARGS}` 直接调用训练脚本。

2.4.3 使用单个 GPU 进行训练

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

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

2.4.4 使用多个 GPU 进行训练

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

可选参数为：

- `--validate` (**强烈建议**)：在训练期间每 k 个周期进行一次验证（默认值为 5，可通过修改每个配置文件中的 `evaluation` 字典变量的 `interval` 值进行改变）。
- `--test-last`：在训练结束后使用最后一个检查点的参数进行测试，将测试结果存储在 `${WORK_DIR}/last_pred.pkl` 中。
- `--test-best`：在训练结束后使用效果最好的检查点的参数进行测试，将测试结果存储在 `${WORK_DIR}/best_pred.pkl` 中。
- `--work-dir ${WORK_DIR}`：覆盖配置文件中指定的工作目录。
- `--resume-from ${CHECKPOINT_FILE}`：从以前的模型权重文件恢复训练。
- `--gpus ${GPU_NUM}`：使用的 GPU 数量，仅适用于非分布式训练。
- `--gpu-ids ${GPU_IDS}`：使用的 GPU ID，仅适用于非分布式训练。
- `--seed ${SEED}`：设置 python，numpy 和 pytorch 里的种子 ID，已用于生成随机数。
- `--deterministic`：如果被指定，程序将设置 CUDNN 后端的确定化选项。
- `JOB_LAUNCHER`：分布式任务初始化启动器选项。可选值有 `none`，`pytorch`，`slurm`，`mpi`。特别地，如果被设置为 `none`, 则会以非分布式模式进行测试。
- `LOCAL_RANK`：本地 rank 的 ID。如果没有被指定，则会被设置为 0。

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

这里提供一个使用 8 块 GPU 加载 TSN 模型权重文件的例子。

```
./tools/dist_train.sh configs/recognition/tsn/tsn_r50_1x1x3_100e_kinetics400_rgb.py 8␣
↪--resume-from work_dirs/tsn_r50_1x1x3_100e_kinetics400_rgb/latest.pth
```

2.4.5 使用多台机器进行训练

如果用户在 slurm 集群上运行 MMAction2，可使用 `slurm_train.sh` 脚本。（该脚本也支持单台机器上进行训练）

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

这里给出一个在 slurm 集群上的 dev 分区使用 16 块 GPU 训练 TSN 的例子。（使用 GPUS_PER_NODE=8 参数来指定一个有 8 块 GPUS 的 slurm 集群节点）

```
GPUS=16 ./tools/slurm_train.sh dev tsn_r50_k400 configs/recognition/tsn/tsn_r50_1x1x3_
↪100e_kinetics400_rgb.py --work-dir work_dirs/tsn_r50_1x1x3_100e_kinetics400_rgb
```

用户可以查看 slurm_train.sh 文件来检查完整的参数和环境变量。

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。

2.4.6 使用单台机器启动多个任务

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

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

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

如果用户在 slurm 集群下启动多个训练任务，则需要修改配置文件（通常是配置文件的倒数第 6 行）中的 dist_params 变量，以设置不同的通信端口。

在 config1.py 中，

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

在 config2.py 中，

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

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

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

2.5 详细教程

目前, MMAction2 提供以下几种更详细的教程:

- 如何编写配置文件
- 如何微调模型
- 如何增加新数据集
- 如何设计数据处理流程
- 如何增加新模块
- 如何导出模型为 *onnx* 格式
- 如何自定义模型运行参数

CHAPTER 3

Demo 示例

3.1 目录

3.2 预测视频的动作标签

MMAction2 提供如下脚本以预测视频的动作标签。为得到 [0, 1] 间的动作分值，请确保在配置文件中设定 `model['test_cfg'] = dict(average_clips='prob')`。

```
python demo/demo.py ${CONFIG_FILE} ${CHECKPOINT_FILE} ${VIDEO_FILE} {LABEL_FILE} [--
↪use-frames] \
    [--device ${DEVICE_TYPE}] [--fps {FPS}] [--font-scale {FONT_SCALE}] [--font-color
↪{FONT_COLOR}] \
    [--target-resolution ${TARGET_RESOLUTION}] [--resize-algorithm {RESIZE_ALGORITHM}
↪] [--out-filename {OUT_FILE}]
```

可选参数：

- `--use-frames`: 如指定，代表使用帧目录作为输入；否则代表使用视频作为输入。
- `DEVICE_TYPE`: 指定脚本运行设备，支持 cuda 设备（如 `cuda:0`）或 cpu（`cpu`）。默认为 `cuda:0`。
- `FPS`: 使用帧目录作为输入时，代表输入的帧率。默认为 30。
- `FONT_SCALE`: 输出视频上的字体缩放比例。默认为 0.5。
- `FONT_COLOR`: 输出视频上的字体颜色，默认为白色（`white`）。
- `TARGET_RESOLUTION`: 输出视频的分辨率，如未指定，使用输入视频的分辨率。
- `RESIZE_ALGORITHM`: 缩放视频时使用的插值方法，默认为 `bicubic`。
- `OUT_FILE`: 输出视频的路径，如未指定，则不会生成输出视频。

示例：

以下示例假设用户的当前目录为 `$MMACTION2`，并已经将所需的模型权重文件下载至目录 `checkpoints/` 下，用户也可以使用所提供的 URL 来直接加载模型权重，文件将会被默认下载至 `$HOME/.cache/torch/checkpoints`。

1. 在 cuda 设备上，使用 TSN 模型进行视频识别：

```
# demo.mp4 及 label_map_k400.txt 均来自 Kinetics-400 数据集
python demo/demo.py configs/recognition/tsn/tsn_r50_video_inference_1x1x3_100e_
↪kinetics400_rgb.py \
    checkpoints/tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.pth \
    demo/demo.mp4 tools/data/kinetics/label_map_k400.txt
```

2. 在 cuda 设备上，使用 TSN 模型进行视频识别，并利用 URL 加载模型权重文件：

```
# demo.mp4 及 label_map_k400.txt 均来自 Kinetics-400 数据集
python demo/demo.py configs/recognition/tsn/tsn_r50_video_inference_1x1x3_100e_
↪kinetics400_rgb.py \
```

(下页继续)

(续上页)

```
    https://download.openmmlab.com/mmaction/recognition/tsn/tsn_r50_1x1x3_100e_
↪kinetics400_rgb/tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.pth \
    demo/demo.mp4 tools/data/kinetics/label_map_k400.txt
```

3. 在 CPU 上，使用 TSN 模型进行视频识别，输入为视频抽好的帧：

```
python demo/demo.py configs/recognition/tsn/tsn_r50_inference_1x1x3_100e_
↪kinetics400_rgb.py \
    checkpoints/tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.pth \
    PATH_TO_FRAMES/ LABEL_FILE --use-frames --device cpu
```

4. 使用 TSN 模型进行视频识别，输出 MP4 格式的识别结果：

```
# demo.mp4 及 label_map_k400.txt 均来自 Kinetics-400 数据集
python demo/demo.py configs/recognition/tsn/tsn_r50_video_inference_1x1x3_100e_
↪kinetics400_rgb.py \
    checkpoints/tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.pth \
    demo/demo.mp4 tools/data/kinetics/label_map_k400.txt --out-filename demo/demo_
↪out.mp4
```

5. 使用 TSN 模型进行视频识别，输入为视频抽好的帧，将识别结果存为 GIF 格式：

```
python demo/demo.py configs/recognition/tsn/tsn_r50_inference_1x1x3_100e_
↪kinetics400_rgb.py \
    checkpoints/tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.pth \
    PATH_TO_FRAMES/ LABEL_FILE --use-frames --out-filename demo/demo_out.gif
```

6. 使用 TSN 模型进行视频识别，输出 MP4 格式的识别结果，并指定输出视频分辨率及缩放视频时使用的插值方法：

```
# demo.mp4 及 label_map_k400.txt 均来自 Kinetics-400 数据集
python demo/demo.py configs/recognition/tsn/tsn_r50_video_inference_1x1x3_100e_
↪kinetics400_rgb.py \
    checkpoints/tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.pth \
    demo/demo.mp4 tools/data/kinetics/label_map_k400.txt --target-resolution 340␣
↪256 --resize-algorithm bilinear \
    --out-filename demo/demo_out.mp4
```

```
# demo.mp4 及 label_map_k400.txt 均来自 Kinetics-400 数据集
# 若 TARGET_RESOLUTION 的任一维度被设置为 -1，视频帧缩放时将保持长宽比
# 如设定 --target-resolution 为 170 -1，原先长宽为 (340, 256) 的视频帧将被缩放至 (170,␣
↪128)
python demo/demo.py configs/recognition/tsn/tsn_r50_video_inference_1x1x3_100e_
↪kinetics400_rgb.py \
```

(下页继续)

(续上页)

```
    checkpoints/tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.pth \
    demo/demo.mp4 tools/data/kinetics/label_map_k400.txt --target-resolution 170 -
↪1 --resize-algorithm bilinear \
    --out-filename demo/demo_out.mp4
```

7. 使用 TSN 模型进行视频识别，输出 MP4 格式的识别结果，指定输出视频中使用红色文字，字体大小为 10 像素：

```
# demo.mp4 及 label_map_k400.txt 均来自 Kinetics-400 数据集
python demo/demo.py configs/recognition/tsn/tsn_r50_video_inference_1x1x3_100e_
↪kinetics400_rgb.py \
    checkpoints/tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.pth \
    demo/demo.mp4 tools/data/kinetics/label_map_k400.txt --font-size 10 --font-
↪color red \
    --out-filename demo/demo_out.mp4
```

8. 使用 TSN 模型进行视频识别，输入为视频抽好的帧，将识别结果存为 MP4 格式，帧率设置为 24fps：

```
python demo/demo.py configs/recognition/tsn/tsn_r50_inference_1x1x3_100e_
↪kinetics400_rgb.py \
    checkpoints/tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.pth \
    PATH_TO_FRAMES/ LABEL_FILE --use-frames --fps 24 --out-filename demo/demo_out.
↪gif
```

3.3 预测视频的时空检测结果

MMAction2 提供如下脚本以预测视频的时空检测结果。

```
python demo/demo_spatiotemporal_det.py --video ${VIDEO_FILE} \
    [--config ${SPATIOTEMPORAL_ACTION_DETECTION_CONFIG_FILE}] \
    [--checkpoint ${SPATIOTEMPORAL_ACTION_DETECTION_CHECKPOINT}] \
    [--det-config ${HUMAN_DETECTION_CONFIG_FILE}] \
    [--det-checkpoint ${HUMAN_DETECTION_CHECKPOINT}] \
    [--det-score-thr ${HUMAN_DETECTION_SCORE_THRESHOLD}] \
    [--action-score-thr ${ACTION_DETECTION_SCORE_THRESHOLD}] \
    [--label-map ${LABEL_MAP}] \
    [--device ${DEVICE}] \
    [--out-filename ${OUTPUT_FILENAME}] \
    [--predict-stepsize ${PREDICT_STEPSIZE}] \
    [--output-stepsize ${OUTPUT_STEPSIZE}] \
    [--output-fps ${OUTPUT_FPS}]
```

可选参数：

- SPATIOTEMPORAL_ACTION_DETECTION_CONFIG_FILE: 时空检测配置文件路径。
- SPATIOTEMPORAL_ACTION_DETECTION_CHECKPOINT: 时空检测模型权重文件路径。
- HUMAN_DETECTION_CONFIG_FILE: 人体检测配置文件路径。
- HUMAN_DETECTION_CHECKPOINT: 人体检测模型权重文件路径。
- HUMAN_DETECTION_SCORE_THRE: 人体检测分数阈值，默认为 0.9。
- ACTION_DETECTION_SCORE_THRESHOLD: 动作检测分数阈值，默认为 0.5。
- LABEL_MAP: 所使用的标签映射文件，默认为 tools/data/ava/label_map.txt。
- DEVICE: 指定脚本运行设备，支持 cuda 设备（如 cuda:0）或 cpu（cpu）。默认为 cuda:0。
- OUTPUT_FILENAME: 输出视频的路径，默认为 demo/stdet_demo.mp4。
- PREDICT_STEPSIZE: 每 N 帧进行一次预测（以节约计算资源），默认值为 8。
- OUTPUT_STEPSIZE: 对于输入视频的每 N 帧，输出 1 帧至输出视频中，默认值为 4，注意需满足 PREDICT_STEPSIZE % OUTPUT_STEPSIZE == 0。
- OUTPUT_FPS: 输出视频的帧率，默认值为 6。

示例：

以下示例假设用户的当前目录为 $MMACTION2，并已经将所需的模型权重文件下载至目录 checkpoints/ 下，用户也可以使用所提供的 URL 来直接加载模型权重，文件将会被默认下载至 $HOME/.cache/torch/checkpoints。

1. 使用 Faster RCNN 作为人体检测器，SlowOnly-8x8-R101 作为动作检测器。每 8 帧进行一次预测，原视频中每 4 帧输出 1 帧至输出视频中，设置输出视频的帧率为 6。

```
python demo/demo_spatiotemporal_det.py --video demo/demo.mp4 \
    --config configs/detection/ava/slowonly_omnisource_pretrained_r101_8x8x1_20e_ava_
↪rgb.py \
    --checkpoint https://download.openmmlab.com/mmaction/detection/ava/slowonly_
↪omnisource_pretrained_r101_8x8x1_20e_ava_rgb/slowonly_omnisource_pretrained_r101_
↪8x8x1_20e_ava_rgb_20201217-16378594.pth \
    --det-config demo/faster_rcnn_r50_fpn_2x_coco.py \
    --det-checkpoint http://download.openmmlab.com/mmdetection/v2.0/faster_rcnn/
↪faster_rcnn_r50_fpn_2x_coco/faster_rcnn_r50_fpn_2x_coco_bbox_mAP-0.384_20200504_
↪210434-a5d8aa15.pth \
    --det-score-thr 0.9 \
    --action-score-thr 0.5 \
    --label-map tools/data/ava/label_map.txt \
    --predict-stepsize 8 \
    --output-stepsize 4 \
    --output-fps 6
```

3.4 可视化输入视频的 GradCAM

MMAction2 提供如下脚本以可视化输入视频的 GradCAM。

```
python demo/demo_gradcam.py ${CONFIG_FILE} ${CHECKPOINT_FILE} ${VIDEO_FILE} [--use-
↪frames] \
    [--device ${DEVICE_TYPE}] [--target-layer-name ${TARGET_LAYER_NAME}] [--fps {FPS}
↪] \
    [--target-resolution ${TARGET_RESOLUTION}] [--resize-algorithm {RESIZE_ALGORITHM}
↪] [--out-filename {OUT_FILE}]
```

可选参数：

- `--use-frames`: 如指定，代表使用帧目录作为输入；否则代表使用视频作为输入。
- `DEVICE_TYPE`: 指定脚本运行设备，支持 cuda 设备（如 `cuda:0`）或 cpu（`cpu`）。默认为 `cuda:0`。
- `TARGET_LAYER_NAME`: 需要生成 GradCAM 可视化的网络层名称。
- `FPS`: 使用帧目录作为输入时，代表输入的帧率。默认为 30。
- `TARGET_RESOLUTION`: 输出视频的分辨率，如未指定，使用输入视频的分辨率。
- `RESIZE_ALGORITHM`: 缩放视频时使用的插值方法，默认为 `bilinear`。
- `OUT_FILE`: 输出视频的路径，如未指定，则不会生成输出视频。

示例：

以下示例假设用户的当前目录为 `$MMACTION2`，并已经将所需的模型权重文件下载至目录 `checkpoints/` 下，用户也可以使用所提供的 URL 来直接加载模型权重，文件将会被默认下载至 `$HOME/.cache/torch/checkpoints`。

1. 对于 I3D 模型进行 GradCAM 的可视化，使用视频作为输入，并输出一帧率为 10 的 GIF 文件：

```
python demo/demo_gradcam.py configs/recognition/i3d/i3d_r50_video_inference_
↪32x2x1_100e_kinetics400_rgb.py \
    checkpoints/i3d_r50_video_32x2x1_100e_kinetics400_rgb_20200826-e31c6f52.pth␣
↪demo/demo.mp4 \
    --target-layer-name backbone/layer4/1/relu --fps 10 \
    --out-filename demo/demo_gradcam.gif
```

2. 对于 I3D 模型进行 GradCAM 的可视化，使用视频作为输入，并输出一 GIF 文件，此示例利用 URL 加载模型权重文件：

```
python demo/demo_gradcam.py configs/recognition/tsm/tsm_r50_video_inference_1x1x8_
↪100e_kinetics400_rgb.py \
    https://download.openmmlab.com/mmaction/recognition/tsm/tsm_r50_video_1x1x8_
↪100e_kinetics400_rgb/tsm_r50_video_1x1x8_100e_kinetics400_rgb_20200702-a77f4328.
↪pth \
```

(下页继续)

(续上页)

```
    demo/demo.mp4 --target-layer-name backbone/layer4/1/relu --out-filename demo/
↪demo_gradcam_tsm.gif
```

3.5 使用网络摄像头的实时动作识别

MMAction2 提供如下脚本来进行使用网络摄像头的实时动作识别。为得到 [0, 1] 间的动作分值，请确保在配置文件中设定 `model['test_cfg'] = dict(average_clips='prob')` 。

```
python demo/webcam_demo.py ${CONFIG_FILE} ${CHECKPOINT_FILE} ${LABEL_FILE} \
    [--device ${DEVICE_TYPE}] [--camera-id ${CAMERA_ID}] [--threshold ${THRESHOLD}] \
    [--average-size ${AVERAGE_SIZE}] [--drawing-fps ${DRAWING_FPS}] [--inference-fps $
↪{INFERENCE_FPS}]
```

可选参数：

- `DEVICE_TYPE`: 指定脚本运行设备，支持 cuda 设备（如 `cuda:0`）或 cpu（`cpu`）。默认为 `cuda:0`。
- `CAMERA_ID`: 摄像头设备的 ID，默认为 0。
- `THRESHOLD`: 动作识别的分数阈值，只有分数大于阈值的动作类型会被显示，默认为 0。
- `AVERAGE_SIZE`: 使用最近 N 个片段的平均结果作为预测，默认为 1。
- `DRAWING_FPS`: 可视化结果时的最高帧率，默认为 20。
- `INFERENCE_FPS`: 进行推理时的最高帧率，默认为 4。

注：若用户的硬件配置足够，可增大可视化帧率和推理帧率以带来更好体验。

示例：

以下示例假设用户的当前目录为 `$MMACTION2`，并已经将所需的模型权重文件下载至目录 `checkpoints/` 下，用户也可以使用所提供的 URL 来直接加载模型权重，文件将会被默认下载至 `$HOME/.cache/torch/checkpoints`。

1. 使用 TSN 模型进行利用网络摄像头的实时动作识别，平均最近 5 个片段结果作为预测，输出大于阈值 0.2 的动作类别：

```
   python demo/webcam_demo.py configs/recognition/tsn/tsn_r50_video_inference_1x1x3_
↪100e_kinetics400_rgb.py \
     checkpoints/tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.pth tools/data/
↪kinetics/label_map_k400.txt --average-size 5 \
     --threshold 0.2 --device cpu
```

2. 使用 TSN 模型在 CPU 上进行利用网络摄像头的实时动作识别，平均最近 5 个片段结果作为预测，输出大于阈值 0.2 的动作类别，此示例利用 URL 加载模型权重文件：

```
    python demo/webcam_demo.py configs/recognition/tsn/tsn_r50_video_inference_1x1x3_
↪100e_kinetics400_rgb.py \
      https://download.openmmlab.com/mmaction/recognition/tsn/tsn_r50_1x1x3_100e_
↪kinetics400_rgb/tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.pth \
      tools/data/kinetics/label_map_k400.txt --average-size 5 --threshold 0.2 --
↪device cpu
```

3. 使用 I3D 模型在 GPU 上进行利用网络摄像头的实时动作识别，平均最近 5 个片段结果作为预测，输出大于阈值 0.2 的动作类别：

```
    python demo/webcam_demo.py configs/recognition/i3d/i3d_r50_video_inference_32x2x1_
↪100e_kinetics400_rgb.py \
      checkpoints/i3d_r50_32x2x1_100e_kinetics400_rgb_20200614-c25ef9a4.pth tools/
↪data/kinetics/label_map_k400.txt \
      --average-size 5 --threshold 0.2
```

注: 考虑到用户所使用的推理设备具有性能差异，可进行如下改动在用户设备上取得更好效果：

1). 更改配置文件中的 `test_pipeline` 下 `SampleFrames` 步骤（特别是 `clip_len` 与 `num_clips`）。2). 更改配置文件中的 `test_pipeline` 下的裁剪方式类型（可选项含：`TenCrop`, `ThreeCrop`, `CenterCrop`）。3). 调低 `AVERAGE_SIZE` 以加快推理。

3.6 滑动窗口预测长视频中不同动作类别

MMAction2 提供如下脚本来预测长视频中的不同动作类别。为得到 [0, 1] 间的动作分值，请确保在配置文件中设定 `model['test_cfg'] = dict(average_clips='prob')` 。

```
python demo/long_video_demo.py ${CONFIG_FILE} ${CHECKPOINT_FILE} ${VIDEO_FILE} $
↪{LABEL_FILE} \
    ${OUT_FILE} [--input-step ${INPUT_STEP}] [--device ${DEVICE_TYPE}] [--threshold $
↪{THRESHOLD}]
```

可选参数：

- `OUT_FILE`: 输出视频的路径。
- `INPUT_STEP`: 在视频中的每 N 帧中选取一帧作为输入，默认为 1。
- `DEVICE_TYPE`: 指定脚本运行设备，支持 cuda 设备（如 `cuda:0`）或 cpu（`cpu`）。默认为 `cuda:0`。
- `THRESHOLD`: 动作识别的分数阈值，只有分数大于阈值的动作类型会被显示，默认为 0.01。
- `STRIDE`: 默认情况下，脚本为每帧给出单独预测，较为耗时。可以设定 `STRIDE` 参数进行加速，此时脚本将会为每 `STRIDE x sample_length` 帧做一次预测（`sample_length` 指模型采帧时的时间窗大小，等于 `clip_len x frame_interval`）。例如，若 sample_length 为 64 帧且 `STRIDE` 设定为 0.5，

模型将每 32 帧做一次预测。若 STRIDE 设为 0，模型将为每帧做一次预测。STRIDE 的理想取值为 (0, 1] 间，若大于 1，脚本亦可正常执行。STRIDE 默认值为 0。

示例：

以下示例假设用户的当前目录为 $MMACTION2，并已经将所需的模型权重文件下载至目录 checkpoints/ 下，用户也可以使用所提供的 URL 来直接加载模型权重，文件将会被默认下载至 $HOME/.cache/torch/checkpoints。

1. 利用 TSN 模型在 CPU 上预测长视频中的不同动作类别，设置 INPUT_STEP 为 3（即每 3 帧随机选取 1 帧作为输入），输出分值大于 0.2 的动作类别：

```
python demo/long_video_demo.py configs/recognition/tsn/tsn_r50_video_inference_
↪1x1x3_100e_kinetics400_rgb.py \
    checkpoints/tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.pth PATH_TO_
↪LONG_VIDEO tools/data/kinetics/label_map_k400.txt PATH_TO_SAVED_VIDEO \
    --input-step 3 --device cpu --threshold 0.2
```

2. 利用 TSN 模型在 CPU 上预测长视频中的不同动作类别，设置 INPUT_STEP 为 3，输出分值大于 0.2 的动作类别，此示例利用 URL 加载模型权重文件：

```
python demo/long_video_demo.py configs/recognition/tsn/tsn_r50_video_inference_
↪1x1x3_100e_kinetics400_rgb.py \
    https://download.openmmlab.com/mmaction/recognition/tsn/tsn_r50_1x1x3_100e_
↪kinetics400_rgb/tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.pth \
    PATH_TO_LONG_VIDEO tools/data/kinetics/label_map_k400.txt PATH_TO_SAVED_VIDEO --
↪input-step 3 --device cpu --threshold 0.2
```

3. 利用 TSN 模型在 CPU 上预测网络长视频（利用 URL 读取）中的不同动作类别，设置 INPUT_STEP 为 3，输出分值大于 0.2 的动作类别，此示例利用 URL 加载模型权重文件：

```
python demo/long_video_demo.py configs/recognition/tsn/tsn_r50_video_inference_
↪1x1x3_100e_kinetics400_rgb.py \
    https://download.openmmlab.com/mmaction/recognition/tsn/tsn_r50_1x1x3_100e_
↪kinetics400_rgb/tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.pth \
    https://www.learningcontainer.com/wp-content/uploads/2020/05/sample-mp4-file.mp4␣
↪\
    tools/data/kinetics/label_map_k400.txt PATH_TO_SAVED_VIDEO --input-step 3 --
↪device cpu --threshold 0.2
```

4. 利用 I3D 模型在 GPU 上预测长视频中的不同动作类别，设置 INPUT_STEP 为 3，动作识别的分数阈值为 0.01：

```
python demo/long_video_demo.py configs/recognition/i3d/i3d_r50_video_inference_
↪32x2x1_100e_kinetics400_rgb.py \
  checkpoints/i3d_r50_256p_32x2x1_100e_kinetics400_rgb_20200801-7d9f44de.pth PATH_
↪TO_LONG_VIDEO tools/data/kinetics/label_map_k400.txt PATH_TO_SAVED_VIDEO \
```

(下页继续)

(续上页)

3.7 基于网络摄像头的实时时空动作检测

MMAction2 提供本脚本实现基于网络摄像头的实时时空动作检测。

```
python demo/webcam_demo_spatiotemporal_det.py \
    [--config ${SPATIOTEMPORAL_ACTION_DETECTION_CONFIG_FILE}] \
    [--checkpoint ${SPATIOTEMPORAL_ACTION_DETECTION_CHECKPOINT}] \
    [--action-score-thr ${ACTION_DETECTION_SCORE_THRESHOLD}] \
    [--det-config ${HUMAN_DETECTION_CONFIG_FILE}] \
    [--det-checkpoint ${HUMAN_DETECTION_CHECKPOINT}] \
    [--det-score-thr ${HUMAN_DETECTION_SCORE_THRESHOLD}] \
    [--input-video] ${INPUT_VIDEO} \
    [--label-map ${LABEL_MAP}] \
    [--device ${DEVICE}] \
    [--output-fps ${OUTPUT_FPS}] \
    [--out-filename ${OUTPUT_FILENAME}] \
    [--show] \
    [--display-height] ${DISPLAY_HEIGHT} \
    [--display-width] ${DISPLAY_WIDTH} \
    [--predict-stepsize ${PREDICT_STEPSIZE}] \
    [--clip-vis-length] ${CLIP_VIS_LENGTH}
```

可选参数：

- SPATIOTEMPORAL_ACTION_DETECTION_CONFIG_FILE: 时空检测配置文件路径。
- SPATIOTEMPORAL_ACTION_DETECTION_CHECKPOINT: 时空检测模型权重文件路径。
- ACTION_DETECTION_SCORE_THRESHOLD: 动作检测分数阈值，默认为 0.4。
- HUMAN_DETECTION_CONFIG_FILE: 人体检测配置文件路径。
- HUMAN_DETECTION_CHECKPOINT: 人体检测模型权重文件路径。
- HUMAN_DETECTION_SCORE_THRE: 人体检测分数阈值，默认为 0.9。
- INPUT_VIDEO: 网络摄像头编号或本地视频文件路径，默认为 0。
- LABEL_MAP: 所使用的标签映射文件，默认为 tools/data/ava/label_map.txt。
- DEVICE: 指定脚本运行设备，支持 cuda 设备（如 cuda:0）或 cpu（cpu），默认为 cuda:0。
- OUTPUT_FPS: 输出视频的帧率，默认为 15。
- OUTPUT_FILENAME: 输出视频的路径，默认为 None。
- --show: 是否通过 cv2.imshow 展示预测结果。

- DISPLAY_HEIGHT: 输出结果图像高度，默认为 0。
- DISPLAY_WIDTH: 输出结果图像宽度，默认为 0。若 DISPLAY_HEIGHT <= 0 and DISPLAY_WIDTH <= 0，则表示输出图像形状与输入视频形状相同。
- PREDICT_STEPSIZE: 每 N 帧进行一次预测（以控制计算资源），默认为 8。
- CLIP_VIS_LENGTH: 预测结果可视化持续帧数，即每次预测结果将可视化到 CLIP_VIS_LENGTH 帧中，默认为 8。

小技巧：

- 如何设置 --output-fps 的数值?
  - --output-fps 建议设置为视频读取线程的帧率。
  - 视频读取线程帧率已通过日志输出，格式为 DEBUG:__main__:Read Thread: {duration} ms, {fps} fps。
- 如何设置 --predict-stepsize 的数值?
  - 该参数选择与模型选型有关。
  - 模型输入构建时间（视频读取线程）应大于等于模型推理时间（主线程）。
  - 模型输入构建时间与模型推理时间均已通过日志输出。
  - --predict-stepsize 数值越大，模型输入构建时间越长。
  - 可降低 --predict-stepsize 数值增加模型推理频率，从而充分利用计算资源。

示例：

以下示例假设用户的当前目录为 $MMACTION2，并已经将所需的模型权重文件下载至目录 checkpoints/ 下，用户也可以使用所提供的 URL 来直接加载模型权重，文件将会被默认下载至 $HOME/.cache/torch/checkpoints。

1. 使用 Faster RCNN 作为人体检测器，SlowOnly-8x8-R101 作为动作检测器，每 8 帧进行一次预测，设置输出视频的帧率为 20，并通过 cv2.imshow 展示预测结果。

```
python demo/webcam_demo_spatiotemporal_det.py \
    --input-video 0 \
    --config configs/detection/ava/slowonly_omnisource_pretrained_r101_8x8x1_20e_ava_
↪rgb.py \
    --checkpoint https://download.openmmlab.com/mmaction/detection/ava/slowonly_
↪omnisource_pretrained_r101_8x8x1_20e_ava_rgb/slowonly_omnisource_pretrained_r101_
↪8x8x1_20e_ava_rgb_20201217-16378594.pth \
    --det-config demo/faster_rcnn_r50_fpn_2x_coco.py \
    --det-checkpoint http://download.openmmlab.com/mmdetection/v2.0/faster_rcnn/
↪faster_rcnn_r50_fpn_2x_coco/faster_rcnn_r50_fpn_2x_coco_bbox_mAP-0.384_20200504_
↪210434-a5d8aa15.pth \
    --det-score-thr 0.9 \
    --action-score-thr 0.5 \
```

(下页继续)

(续上页)

```
    --label-map tools/data/ava/label_map.txt \
    --predict-stepsize 40 \
    --output-fps 20 \
    --show
```

3.8 基于人体姿态预测动作标签

MMAction2 提供本脚本实现基于人体姿态的动作标签预测。

```
python demo/demo_skeleton.py ${VIDEO_FILE} ${OUT_FILENAME} \
    [--config ${SKELETON_BASED_ACTION_RECOGNITION_CONFIG_FILE}] \
    [--checkpoint ${SKELETON_BASED_ACTION_RECOGNITION_CHECKPOINT}] \
    [--det-config ${HUMAN_DETECTION_CONFIG_FILE}] \
    [--det-checkpoint ${HUMAN_DETECTION_CHECKPOINT}] \
    [--det-score-thr ${HUMAN_DETECTION_SCORE_THRESHOLD}] \
    [--pose-config ${HUMAN_POSE_ESTIMATION_CONFIG_FILE}] \
    [--pose-checkpoint ${HUMAN_POSE_ESTIMATION_CHECKPOINT}] \
    [--label-map ${LABEL_MAP}] \
    [--device ${DEVICE}] \
    [--short-side] ${SHORT_SIDE}
```

可选参数:

- SKELETON_BASED_ACTION_RECOGNITION_CONFIG_FILE: 基于人体姿态的动作识别模型配置文件路径。
- SKELETON_BASED_ACTION_RECOGNITION_CHECKPOINT: 基于人体姿态的动作识别模型权重文件路径。
- HUMAN_DETECTION_CONFIG_FILE: 人体检测配置文件路径。
- HUMAN_DETECTION_CHECKPOINT: 人体检测模型权重文件路径。
- HUMAN_DETECTION_SCORE_THRE: 人体检测分数阈值，默认为 0.9。
- HUMAN_POSE_ESTIMATION_CONFIG_FILE: 人体姿态估计模型配置文件路径 (需在 COCO-keypoint 数据集上训练)。
- HUMAN_POSE_ESTIMATION_CHECKPOINT: 人体姿态估计模型权重文件路径 (需在 COCO-keypoint 数据集上训练).
- LABEL_MAP: 所使用的标签映射文件，默认为 tools/data/skeleton/label_map_ntu120.txt。
- DEVICE: 指定脚本运行设备，支持 cuda 设备（如 cuda:0）或 cpu（cpu），默认为 cuda:0。
- SHORT_SIDE: 视频抽帧时使用的短边长度，默认为 480。

示例：

以下示例假设用户的当前目录为 $MMACTION2。

1. 使用 Faster RCNN 作为人体检测器，HRNetw32 作为人体姿态估计模型，PoseC3D-NTURGB+D-120-Xsub-keypoint 作为基于人体姿态的动作识别模型。

```
python demo/demo_skeleton.py demo/ntu_sample.avi demo/skeleton_demo.mp4 \
    --config configs/skeleton/posec3d/slowonly_r50_u48_240e_ntu120_xsub_keypoint.py \
    --checkpoint https://download.openmmlab.com/mmaction/skeleton/posec3d/slowonly_
↪r50_u48_240e_ntu120_xsub_keypoint/slowonly_r50_u48_240e_ntu120_xsub_keypoint-
↪6736b03f.pth \
    --det-config demo/faster_rcnn_r50_fpn_2x_coco.py \
    --det-checkpoint http://download.openmmlab.com/mmdetection/v2.0/faster_rcnn/
↪faster_rcnn_r50_fpn_2x_coco/faster_rcnn_r50_fpn_2x_coco_bbox_mAP-0.384_20200504_
↪210434-a5d8aa15.pth \
    --det-score-thr 0.9 \
    --pose-config demo/hrnet_w32_coco_256x192.py \
    --pose-checkpoint https://download.openmmlab.com/mmpose/top_down/hrnet/hrnet_w32_
↪coco_256x192-c78dce93_20200708.pth \
    --label-map tools/data/skeleton/label_map_ntu120.txt
```

2. 使用 Faster RCNN 作为人体检测器，HRNetw32 作为人体姿态估计模型，STGCN-NTURGB+D-60-Xsub-keypoint 作为基于人体姿态的动作识别模型。

```
python demo/demo_skeleton.py demo/ntu_sample.avi demo/skeleton_demo.mp4 \
    --config configs/skeleton/stgcn/stgcn_80e_ntu60_xsub_keypoint.py \
    --checkpoint https://download.openmmlab.com/mmaction/skeleton/stgcn/stgcn_80e_
↪ntu60_xsub_keypoint/stgcn_80e_ntu60_xsub_keypoint-e7bb9653.pth \
    --det-config demo/faster_rcnn_r50_fpn_2x_coco.py \
    --det-checkpoint http://download.openmmlab.com/mmdetection/v2.0/faster_rcnn/
↪faster_rcnn_r50_fpn_2x_coco/faster_rcnn_r50_fpn_2x_coco_bbox_mAP-0.384_20200504_
↪210434-a5d8aa15.pth \
    --det-score-thr 0.9 \
    --pose-config demo/hrnet_w32_coco_256x192.py \
    --pose-checkpoint https://download.openmmlab.com/mmpose/top_down/hrnet/hrnet_w32_
↪coco_256x192-c78dce93_20200708.pth \
    --label-map tools/data/skeleton/label_map_ntu120.txt
```

3.9 视频结构化预测

MMAction2 提供本脚本实现基于人体姿态和 RGB 的视频结构化预测。

```
python demo/demo_video_structuralize.py
    [--rgb-stdet-config ${RGB_BASED_SPATIO_TEMPORAL_ACTION_DETECTION_CONFIG_FILE}] \
    [--rgb-stdet-checkpoint ${RGB_BASED_SPATIO_TEMPORAL_ACTION_DETECTION_CHECKPOINT}]␣
↪\
    [--skeleton-stdet-checkpoint ${SKELETON_BASED_SPATIO_TEMPORAL_ACTION_DETECTION_
↪CHECKPOINT}] \
    [--det-config ${HUMAN_DETECTION_CONFIG_FILE}] \
    [--det-checkpoint ${HUMAN_DETECTION_CHECKPOINT}] \
    [--pose-config ${HUMAN_POSE_ESTIMATION_CONFIG_FILE}] \
    [--pose-checkpoint ${HUMAN_POSE_ESTIMATION_CHECKPOINT}] \
    [--skeleton-config ${SKELETON_BASED_ACTION_RECOGNITION_CONFIG_FILE}] \
    [--skeleton-checkpoint ${SKELETON_BASED_ACTION_RECOGNITION_CHECKPOINT}] \
    [--rgb-config ${RGB_BASED_ACTION_RECOGNITION_CONFIG_FILE}] \
    [--rgb-checkpoint ${RGB_BASED_ACTION_RECOGNITION_CHECKPOINT}] \
    [--use-skeleton-stdet ${USE_SKELETON_BASED_SPATIO_TEMPORAL_DETECTION_METHOD}] \
    [--use-skeleton-recog ${USE_SKELETON_BASED_ACTION_RECOGNITION_METHOD}] \
    [--det-score-thr ${HUMAN_DETECTION_SCORE_THRE}] \
    [--action-score-thr ${ACTION_DETECTION_SCORE_THRE}] \
    [--video ${VIDEO_FILE}] \
    [--label-map-stdet ${LABEL_MAP_FOR_SPATIO_TEMPORAL_ACTION_DETECTION}] \
    [--device ${DEVICE}] \
    [--out-filename ${OUTPUT_FILENAME}] \
    [--predict-stepsize ${PREDICT_STEPSIZE}] \
    [--output-stepsize ${OUTPU_STEPSIZE}] \
    [--output-fps ${OUTPUT_FPS}] \
    [--cfg-options]
```

可选参数:

- RGB_BASED_SPATIO_TEMPORAL_ACTION_DETECTION_CONFIG_FILE: 基于 RGB 的时空检测配置文件路径。
- RGB_BASED_SPATIO_TEMPORAL_ACTION_DETECTION_CHECKPOINT: 基于 RGB 的时空检测模型权重文件路径。
- SKELETON_BASED_SPATIO_TEMPORAL_ACTION_DETECTION_CHECKPOINT: 基于人体姿态的时空检测模型权重文件路径。
- HUMAN_DETECTION_CONFIG_FILE: 人体检测配置文件路径。
- HUMAN_DETECTION_CHECKPOINT: The human detection checkpoint URL.
- HUMAN_POSE_ESTIMATION_CONFIG_FILE: 人体姿态估计模型配置文件路径 (需在 COCO-keypoint

数据集上训练)。

- HUMAN_POSE_ESTIMATION_CHECKPOINT: 人体姿态估计模型权重文件路径 (需在 COCO-keypoint 数据集上训练)。
- SKELETON_BASED_ACTION_RECOGNITION_CONFIG_FILE: 基于人体姿态的动作识别模型配置文件路径。
- SKELETON_BASED_ACTION_RECOGNITION_CHECKPOINT: 基于人体姿态的动作识别模型权重文件路径。
- RGB_BASED_ACTION_RECOGNITION_CONFIG_FILE: 基于 RGB 的行为识别配置文件路径。
- RGB_BASED_ACTION_RECOGNITION_CHECKPOINT: 基于 RGB 的行为识别模型权重文件路径。
- USE_SKELETON_BASED_SPATIO_TEMPORAL_DETECTION_METHOD: 使用基于人体姿态的时空检测方法。
- USE_SKELETON_BASED_ACTION_RECOGNITION_METHOD: 使用基于人体姿态的行为识别方法。
- HUMAN_DETECTION_SCORE_THRE: 人体检测分数阈值，默认为 0.9。
- ACTION_DETECTION_SCORE_THRE: 动作检测分数阈值，默认为 0.5。
- LABEL_MAP_FOR_SPATIO_TEMPORAL_ACTION_DETECTION: 时空动作检测所使用的标签映射文件，默认为: tools/data/ava/label_map.txt。
- LABEL_MAP: 行为识别所使用的标签映射文件，默认为: tools/data/kinetics/label_map_k400.txt。
- DEVICE: 指定脚本运行设备，支持 cuda 设备（如 cuda:0）或 cpu（cpu），默认为 cuda:0。
- OUTPUT_FILENAME: 输出视频的路径，默认为 demo/test_stdet_recognition_output.mp4。
- PREDICT_STEPSIZE: 每 N 帧进行一次预测（以节约计算资源），默认值为 8。
- OUTPUT_STEPSIZE: 对于输入视频的每 N 帧，输出 1 帧至输出视频中，默认值为 1，注意需满足 PREDICT_STEPSIZE % OUTPUT_STEPSIZE == 0。
- OUTPUT_FPS: 输出视频的帧率，默认为 24。

示例：

以下示例假设用户的当前目录为 $MMACTION2。

1. 使用 Faster RCNN 作为人体检测器，HRNetw32 作为人体姿态估计模型，PoseC3D 作为基于人体姿态的动作识别模型和时空动作检测器。每 8 帧进行一次预测，原视频中每 1 帧输出 1 帧至输出视频中，设置输出视频的帧率为 24。

```
python demo/demo_video_structuralize.py
    --skeleton-stdet-checkpoint https://download.openmmlab.com/mmaction/skeleton/
↪posec3d/posec3d_ava.pth \
    --det-config demo/faster_rcnn_r50_fpn_2x_coco.py \
```

(下页继续)

(续上页)

```
    --det-checkpoint http://download.openmmlab.com/mmdetection/v2.0/faster_rcnn/
↪faster_rcnn_r50_fpn_2x_coco/faster_rcnn_r50_fpn_2x_coco_bbox_mAP-0.384_20200504_
↪210434-a5d8aa15.pth \
    --pose-config demo/hrnet_w32_coco_256x192.py
    --pose-checkpoint https://download.openmmlab.com/mmpose/top_down/hrnet/
    hrnet_w32_coco_256x192-c78dce93_20200708.pth \
    --skeleton-config configs/skeleton/posec3d/slowonly_r50_u48_240e_ntu120_xsub_
↪keypoint.py \
    --skeleton-checkpoint https://download.openmmlab.com/mmaction/skeleton/posec3d/
    posec3d_k400.pth \
    --use-skeleton-stdet \
    --use-skeleton-recog \
    --label-map-stdet tools/data/ava/label_map.txt \
    --label-map tools/data/kinetics/label_map_k400.txt
```

2. 使用 Faster RCNN 作为人体检测器，TSN-R50-1x1x3 作为动作识别模型, SlowOnly-8x8-R101 作为时空动检测器。每 8 帧进行一次预测，原视频中每 1 帧输出 1 帧至输出视频中，设置输出视频的帧率为 24。

```
python demo/demo_video_structuralize.py
    --rgb-stdet-config configs/detection/ava/slowonly_omnisource_pretrained_r101_
↪8x8x1_20e_ava_rgb.py \
    --rgb-stdet-checkpoint  https://download.openmmlab.com/mmaction/detection/ava/
↪slowonly_omnisource_pretrained_r101_8x8x1_20e_ava_rgb/slowonly_omnisource_
↪pretrained_r101_8x8x1_20e_ava_rgb_20201217-16378594.pth \
    --det-config demo/faster_rcnn_r50_fpn_2x_coco.py \
    --det-checkpoint http://download.openmmlab.com/mmdetection/v2.0/faster_rcnn/
↪faster_rcnn_r50_fpn_2x_coco/faster_rcnn_r50_fpn_2x_coco_bbox_mAP-0.384_20200504_
↪210434-a5d8aa15.pth \
    --rgb-config configs/recognition/tsn/
    tsn_r50_video_inference_1x1x3_100e_kinetics400_rgb.py \
    --rgb-checkpoint https://download.openmmlab.com/mmaction/recognition/
    tsn/tsn_r50_1x1x3_100e_kinetics400_rgb/
    tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.pth \
    --label-map-stdet tools/data/ava/label_map.txt \
    --label-map tools/data/kinetics/label_map_k400.txt
```

3. 使用 Faster RCNN 作为人体检测器，HRNetw32 作为人体姿态估计模型，PoseC3D 作为基于人体姿态的动作识别模型, SlowOnly-8x8-R101 作为时空动作检测器。每 8 帧进行一次预测，原视频中每 1 帧输出 1 帧至输出视频中，设置输出视频的帧率为 24。

```
python demo/demo_video_structuralize.py
    --rgb-stdet-config configs/detection/ava/slowonly_omnisource_pretrained_r101_
↪8x8x1_20e_ava_rgb.py \
    --rgb-stdet-checkpoint  https://download.openmmlab.com/mmaction/detection/ava/
↪slowonly_omnisource_pretrained_r101_8x8x1_20e_ava_rgb/slowonly_omnisource_
↪pretrained_r101_8x8x1_20e_ava_rgb_20201217-16378594.pth \
```

(下页继续)

(续上页)

```
    --det-config demo/faster_rcnn_r50_fpn_2x_coco.py \
    --det-checkpoint http://download.openmmlab.com/mmdetection/v2.0/faster_rcnn/
↪faster_rcnn_r50_fpn_2x_coco/faster_rcnn_r50_fpn_2x_coco_bbox_mAP-0.384_20200504_
↪210434-a5d8aa15.pth \
    --pose-config demo/hrnet_w32_coco_256x192.py
    --pose-checkpoint https://download.openmmlab.com/mmpose/top_down/hrnet/
    hrnet_w32_coco_256x192-c78dce93_20200708.pth \
    --skeleton-config configs/skeleton/posec3d/slowonly_r50_u48_240e_ntu120_xsub_
↪keypoint.py \
    --skeleton-checkpoint https://download.openmmlab.com/mmaction/skeleton/posec3d/
    posec3d_k400.pth \
    --use-skeleton-recog \
    --label-map-stdet tools/data/ava/label_map.txt \
    --label-map tools/data/kinetics/label_map_k400.txt
```

4. 使用 Faster RCNN 作为人体检测器，HRNetw32 作为人体姿态估计模型，TSN-R50-1x1x3 作为动作识别模型, PoseC3D 作为基于人体姿态的时空动作检测器。每 8 帧进行一次预测，原视频中每 1 帧输出 1 帧至输出视频中，设置输出视频的帧率为 24。

```
python demo/demo_video_structuralize.py
    --skeleton-stdet-checkpoint https://download.openmmlab.com/mmaction/skeleton/
↪posec3d/posec3d_ava.pth \
    --det-config demo/faster_rcnn_r50_fpn_2x_coco.py \
    --det-checkpoint http://download.openmmlab.com/mmdetection/v2.0/faster_rcnn/
↪faster_rcnn_r50_fpn_2x_coco/faster_rcnn_r50_fpn_2x_coco_bbox_mAP-0.384_20200504_
↪210434-a5d8aa15.pth \
    --pose-config demo/hrnet_w32_coco_256x192.py
    --pose-checkpoint https://download.openmmlab.com/mmpose/top_down/hrnet/
    hrnet_w32_coco_256x192-c78dce93_20200708.pth \
    --skeleton-config configs/skeleton/posec3d/slowonly_r50_u48_240e_ntu120_xsub_
↪keypoint.py \
    --rgb-config configs/recognition/tsn/
    tsn_r50_video_inference_1x1x3_100e_kinetics400_rgb.py \
    --rgb-checkpoint https://download.openmmlab.com/mmaction/recognition/
    tsn/tsn_r50_1x1x3_100e_kinetics400_rgb/
    tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.pth \
    --use-skeleton-stdet \
    --label-map-stdet tools/data/ava/label_map.txt \
    --label-map tools/data/kinetics/label_map_k400.txt
```

3.10 基于音频的动作识别

本脚本可用于进行基于音频特征的动作识别。

脚本 `extract_audio.py` 可被用于从视频中提取音频，脚本 `build_audio_features.py` 可被用于基于音频文件提取音频特征。

```
python demo/demo_audio.py ${CONFIG_FILE} ${CHECKPOINT_FILE} ${AUDIO_FILE} {LABEL_FILE}
↪ [--device ${DEVICE}]
```

可选参数：

- `DEVICE`: 指定脚本运行设备，支持 cuda 设备（如 `cuda:0`）或 cpu（`cpu`），默认为 `cuda:0`。

示例：

以下示例假设用户的当前目录为 $MMACTION2。

1. 在 GPU 上，使用 TSN 模型进行基于音频特征的动作识别。

```
python demo/demo_audio.py \
    configs/recognition_audio/resnet/tsn_r18_64x1x1_100e_kinetics400_audio_
↪feature.py \
    https://download.openmmlab.com/mmaction/recognition/audio_recognition/tsn_r18_
↪64x1x1_100e_kinetics400_audio_feature/tsn_r18_64x1x1_100e_kinetics400_audio_
↪feature_20201012-bf34df6c.pth \
    audio_feature.npy label_map_k400.txt
```

CHAPTER 4

基准测试

这里将 MMAction2 与其他流行的代码框架和官方开源代码的速度性能进行对比

4.1 配置

4.1.1 硬件环境

- 8 NVIDIA Tesla V100 (32G) GPUs
- Intel(R) Xeon(R) Gold 6146 CPU @ 3.20GHz

4.1.2 软件环境

- Python 3.7
- PyTorch 1.4
- CUDA 10.1
- CUDNN 7.6.03
- NCCL 2.4.08

4.1.3 评测指标

这里测量的时间是一轮训练迭代的平均时间，包括数据处理和模型训练。训练速度以 s/iter 为单位，其值越低越好。注意，这里跳过了前 50 个迭代时间，因为它们可能包含设备的预热时间。

4.1.4 比较规则

这里以一轮训练迭代时间为基准，使用了相同的数据和模型设置对 MMAction2 和其他的视频理解工具箱进行比较。参与评测的其他代码库包括

- MMAction: commit id 7f3490d(1/5/2020)
- Temporal-Shift-Module: commit id 8d53d6f(5/5/2020)
- PySlowFast: commit id 8299c98(7/7/2020)
- BSN(boundary sensitive network): commit id f13707f(12/12/2018)
- BMN(boundary matching network): commit id 45d0514(17/10/2019)

为了公平比较，这里基于相同的硬件环境和数据进行对比实验。使用的视频帧数据集是通过 数据准备工具 生成的，使用的视频数据集是通过 该脚本 生成的，以快速解码为特点的，”短边 256，密集关键帧编码“的视频数据集。正如以下表格所示，在对比正常的短边 256 视频时，可以观察到速度上的显著提升，尤其是在采样特别稀疏的情况下，如 TSN。

4.2 主要结果

4.2.1 行为识别器

4.2.2 时序动作检测器

4.3 比较细节

4.3.1 TSN

- **MMAction2**

```
# 处理视频帧
bash tools/slurm_train.sh ${PARTATION_NAME} benchmark_tsn configs/recognition/tsn/tsn_
↪r50_1x1x3_100e_kinetics400_rgb.py --work-dir work_dirs/benchmark_tsn_rawframes

# 处理视频
bash tools/slurm_train.sh ${PARTATION_NAME} benchmark_tsn configs/recognition/tsn/tsn_
↪r50_video_1x1x3_100e_kinetics400_rgb.py --work-dir work_dirs/benchmark_tsn_video
```

- **MMAction**

```
python -u tools/train_recognizer.py configs/TSN/tsn_kinetics400_2d_rgb_r50_seg3_f1s1.
↪py
```

- **Temporal-Shift-Module**

```
python main.py kinetics RGB --arch resnet50 --num_segments 3 --gd 20 --lr 0.02 --wd␣
↪1e-4 --lr_steps 20 40 --epochs 1 --batch-size 256 -j 32 --dropout 0.5 --consensus_
↪type=avg --eval-freq=10 --npb --print-freq 1
```

4.3.2 I3D

- **MMAction2**

```
# 处理视频帧
bash tools/slurm_train.sh ${PARTATION_NAME} benchmark_i3d configs/recognition/i3d/i3d_
↪r50_32x2x1_100e_kinetics400_rgb.py --work-dir work_dirs/benchmark_i3d_rawframes

# 处理视频
bash tools/slurm_train.sh ${PARTATION_NAME} benchmark_i3d configs/recognition/i3d/i3d_
↪r50_video_heavy_8x8x1_100e_kinetics400_rgb.py --work-dir work_dirs/benchmark_i3d_
↪video
```

- **MMAction**

```
python -u tools/train_recognizer.py configs/I3D_RGB/i3d_kinetics400_3d_rgb_r50_c3d_
↪inflate3x1x1_seg1_f32s2.py
```

- **PySlowFast**

```
python tools/run_net.py   --cfg configs/Kinetics/I3D_8x8_R50.yaml   DATA.PATH_TO_DATA_
↪DIR ${DATA_ROOT}   NUM_GPUS 8 TRAIN.BATCH_SIZE 64 TRAIN.AUTO_RESUME False LOG_
↪PERIOD 1 SOLVER.MAX_EPOCH 1 > pysf_i3d_r50_8x8_video.log
```

可以通过编写一个简单的脚本对日志文件的‘time_diff’域进行解析，以复现对应的结果。

4.3.3 SlowFast

- **MMAction2**

```
bash tools/slurm_train.sh ${PARTATION_NAME} benchmark_slowfast configs/recognition/
↪slowfast/slowfast_r50_video_4x16x1_256e_kinetics400_rgb.py --work-dir work_dirs/
↪benchmark_slowfast_video
```

- **MMAction**

```
python tools/run_net.py   --cfg configs/Kinetics/SLOWFAST_4x16_R50.yaml   DATA.PATH_
↪TO_DATA_DIR ${DATA_ROOT}   NUM_GPUS 8 TRAIN.BATCH_SIZE 64 TRAIN.AUTO_RESUME False␣
↪LOG_PERIOD 1 SOLVER.MAX_EPOCH 1 > pysf_slowfast_r50_4x16_video.log
```

可以通过编写一个简单的脚本对日志文件的‘time_diff’域进行解析，以复现对应的结果。

4.3.4 SlowOnly

- **MMAction2**

```
bash tools/slurm_train.sh ${PARTATION_NAME} benchmark_slowonly configs/recognition/
↪slowonly/slowonly_r50_video_4x16x1_256e_kinetics400_rgb.py --work-dir work_dirs/
↪benchmark_slowonly_video
```

- **PySlowFast**

```
python tools/run_net.py   --cfg configs/Kinetics/SLOW_4x16_R50.yaml   DATA.PATH_TO_
↪DATA_DIR ${DATA_ROOT}   NUM_GPUS 8 TRAIN.BATCH_SIZE 64 TRAIN.AUTO_RESUME False LOG_
↪PERIOD 1 SOLVER.MAX_EPOCH 1 > pysf_slowonly_r50_4x16_video.log
```

可以通过编写一个简单的脚本对日志文件的‘time_diff’域进行解析，以复现对应的结果。

4.3.5 R2plus1D

- **MMAction2**

```
bash tools/slurm_train.sh ${PARTATION_NAME} benchmark_r2plus1d configs/recognition/
↪r2plus1d/r2plus1d_r34_video_8x8x1_180e_kinetics400_rgb.py --work-dir work_dirs/
↪benchmark_r2plus1d_video
```

CHAPTER 5

总览

- 论文数量: 14
  - DATASET: 14

有关受支持的视频理解算法，可参见模型总览。

5.1 支持的数据集

- 论文数量: 14
  - [DATASET] Activitynet: A Large-Scale Video Benchmark for Human Activity Understanding (ActivityNet ->, ActivityNet ->, ActivityNet ->)
  - [DATASET] Ava: A Video Dataset of Spatio-Temporally Localized Atomic Visual Actions (AVA ->, AVA ->, AVA ->)
  - [DATASET] Finegym: A Hierarchical Video Dataset for Fine-Grained Action Understanding (GYM ->, GYM ->, GYM ->)
  - [DATASET] Hmdb: A Large Video Database for Human Motion Recognition (HMDB51 ->, HMDB51 ->, HMDB51 ->)
  - [DATASET] Large Scale Holistic Video Understanding (HVU ->, HVU ->, HVU ->)
  - [DATASET] Moments in Time Dataset: One Million Videos for Event Understanding (Moments in Time ->, Moments in Time ->, Moments in Time ->)

- [DATASET] Multi-Moments in Time: Learning and Interpreting Models for Multi-Action Video Understanding (Multi-Moments in Time ->, Multi-Moments in Time ->, Multi-Moments in Time ->)
- [DATASET] Omni-Sourced Webly-Supervised Learning for Video Recognition (OmniSource ->, OmniSource ->, OmniSource ->)
- [DATASET] Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset (Kinetics-[400/600/700] ->, Kinetics-[400/600/700] ->, Kinetics-[400/600/700] ->)
- [DATASET] Resound: Towards Action Recognition Without Representation Bias (Diving48 ->, Diving48 ->, Diving48 ->)
- [DATASET] The "Something Something" Video Database for Learning and Evaluating Visual Common Sense (Something-Something V2 ->, Something-Something V1 ->, Something-Something V2 ->, Something-Something V1 ->, Something-Something V2 ->, Something-Something V1 ->)
- [DATASET] The Jester Dataset: A Large-Scale Video Dataset of Human Gestures (Jester ->, Jester ->, Jester ->)
- [DATASET] Towards Understanding Action Recognition (JHMDB ->, JHMDB ->, JHMDB ->)
- [DATASET] {Thumos (THUMOS' 14 ->, THUMOS' 14 ->, THUMOS' 14 ->)

CHAPTER 6

准备数据

本文为 MMAction2 的数据准备提供一些指南。

6.1 视频格式数据的一些注意事项

MMAction2 支持两种数据类型：原始帧和视频。前者在过去的项目中经常出现，如 TSN。如果能把原始帧存储在固态硬盘上，处理帧格式的数据是非常快的，但对于大规模的数据集来说，原始帧需要占据大量的磁盘空间。（举例来说，最新版本的 Kinetics 有 650K 个视频，其所有原始帧需要占据几个 TB 的磁盘空间。）视频格式的数据能够节省很多空间，但在运行模型时，必须进行视频解码，算力开销很大。为了加速视频解码，MMAction2 支持了若干种高效的视频加载库，如 decord, PyAV 等。

6.2 获取数据

本文介绍如何构建自定义数据集。与上述数据集相似，推荐用户把数据放在 `$MMACTION2/data/$DATASET` 中。

6.2.1 准备视频

请参照官网或官方脚本准备视频。注意，应该按照下面两种方法之一来组织视频数据文件夹结构：

(1) 形如 `${CLASS_NAME}/${VIDEO_ID}` 的两级文件目录结构，这种结构推荐在动作识别数据集中使用（如 UCF101 和 Kinetics）

(2) 单级文件目录结构，这种结构推荐在动作检测数据集或者多标签数据集中使用（如 THUMOS14）

6.2.2 提取帧

若想同时提取帧和光流，可以使用 OpenMMLab 准备的 denseflow 工具。因为不同的帧提取工具可能产生不同数量的帧，建议使用同一工具来提取 RGB 帧和光流，以避免它们的数量不同。

```
python build_rawframes.py ${SRC_FOLDER} ${OUT_FOLDER} [--task ${TASK}] [--level ${LEVEL}] \
    [--num-worker ${NUM_WORKER}] [--flow-type ${FLOW_TYPE}] [--out-format ${OUT_FORMAT}] \
    [--ext ${EXT}] [--new-width ${NEW_WIDTH}] [--new-height ${NEW_HEIGHT}] [--new-short ${NEW_SHORT}] \
    [--resume] [--use-opencv] [--mixed-ext]
```

- `SRC_FOLDER`: 视频源文件夹
- `OUT_FOLDER`: 存储提取出的帧和光流的根文件夹
- `TASK`: 提取任务，说明提取帧，光流，还是都提取，选项为 `rgb`, `flow`, `both`
- `LEVEL`: 目录层级。1 指单级文件目录，2 指两级文件目录
- `NUM_WORKER`: 提取原始帧的线程数
- `FLOW_TYPE`: 提取的光流类型，如 `None`, `tvl1`, `warp_tvl1`, `farn`, `brox`
- `OUT_FORMAT`: 提取帧的输出文件类型，如 `jpg`, `h5`, `png`
- `EXT`: 视频文件后缀名，如 `avi`, `mp4`
- `NEW_WIDTH`: 调整尺寸后，输出图像的宽
- `NEW_HEIGHT`: 调整尺寸后，输出图像的高
- `NEW_SHORT`: 等比例缩放图片后，输出图像的短边长
- `--resume`: 是否接续之前的光流提取任务，还是覆盖之前的输出结果重新提取

- `--use-opencv`: 是否使用 OpenCV 提取 RGB 帧
- `--mixed-ext`: 说明是否处理不同文件类型的视频文件

根据实际经验，推荐设置为：

1. 将 `$OUT_FOLDER` 设置为固态硬盘上的文件夹。
2. 软连接 `$OUT_FOLDER` 到 `$MMACTION2/data/$DATASET/rawframes`
3. 使用 `new-short` 而不是 `new-width` 和 `new-height` 来调整图像尺寸

```
ln -s ${YOUR_FOLDER} $MMACTION2/data/$DATASET/rawframes
```

denseflow 的替代项

如果用户因依赖要求（如 Nvidia 显卡驱动版本），无法安装 denseflow，或者只需要一些关于光流提取的快速演示，可用 Python 脚本 `tools/misc/flow_extraction.py` 替代 denseflow。这个脚本可用于一个或多个视频提取 RGB 帧和光流。注意，由于该脚本时在 CPU 上运行光流算法，其速度比 denseflow 慢很多。

```
python tools/misc/flow_extraction.py --input ${INPUT} [--prefix ${PREFIX}] [--dest $
↪{DEST}] [--rgb-tmpl ${RGB_TMPL}] \
    [--flow-tmpl ${FLOW_TMPL}] [--start-idx ${START_IDX}] [--method ${METHOD}] [--
↪bound ${BOUND}] [--save-rgb]
```

- `INPUT`: 用于提取帧的视频，可以是单个视频或一个视频列表，视频列表应该是一个 txt 文件，并且只包含视频文件名，不包含目录
- `PREFIX`: 输入视频的前缀，当输入是一个视频列表时使用
- `DEST`: 保存提取出的帧的位置
- `RGB_TMPL`: RGB 帧的文件名格式
- `FLOW_TMPL`: 光流的文件名格式
- `START_IDX`: 提取帧的开始索引
- `METHOD`: 用于生成光流的方法
- `BOUND`: 光流的最大值
- `SAVE_RGB`: 同时保存提取的 RGB 帧

6.2.3 生成文件列表

MMAction2 提供了便利的脚本用于生成文件列表。在完成视频下载（或更进一步完成视频抽帧）后，用户可以使用如下的脚本生成文件列表。

```
cd $MMACTION2
python tools/data/build_file_list.py ${DATASET} ${SRC_FOLDER} [--rgb-prefix ${RGB_
↪PREFIX}] \
    [--flow-x-prefix ${FLOW_X_PREFIX}] [--flow-y-prefix ${FLOW_Y_PREFIX}] [--num-
↪split ${NUM_SPLIT}] \
    [--subset ${SUBSET}] [--level ${LEVEL}] [--format ${FORMAT}] [--out-root-path $
↪{OUT_ROOT_PATH}] \
    [--seed ${SEED}] [--shuffle]
```

- DATASET: 所要准备的数据集，例如：`ucf101`，`kinetics400`，`thumos14`，`sthv1`，`sthv2` 等。
- SRC_FOLDER: 存放对应格式的数据的目录：
  - 如目录为 “$MMACTION2/data/$DATASET/rawframes”，则需设置 `--format rawframes`。
  - 如目录为 “$MMACTION2/data/$DATASET/videos”，则需设置 `--format videos`。
- RGB_PREFIX: RGB 帧的文件前缀。
- FLOW_X_PREFIX: 光流 x 分量帧的文件前缀。
- FLOW_Y_PREFIX: 光流 y 分量帧的文件前缀。
- NUM_SPLIT: 数据集总共的划分个数。
- SUBSET: 需要生成文件列表的子集名称。可选项为 `train`, `val`, `test`。
- LEVEL: 目录级别数量，1 表示一级目录（数据集中所有视频或帧文件夹位于同一目录），2 表示二级目录（数据集中所有视频或帧文件夹按类别存放于各子目录）。
- FORMAT: 需要生成文件列表的源数据格式。可选项为 `rawframes`, `videos`。
- OUT_ROOT_PATH: 生成文件的根目录。
- SEED: 随机种子。
- `--shuffle`: 是否打乱生成的文件列表。

至此为止，用户可参考基础教程 来进行模型的训练及测试。

6.2.4 准备音频

MMAction2 还提供如下脚本来提取音频的波形并生成梅尔频谱。

```
cd $MMACTION2
python tools/data/extract_audio.py ${ROOT} ${DST_ROOT} [--ext ${EXT}] [--num-workers $
↪{N_WORKERS}] \
    [--level ${LEVEL}]
```

- ROOT: 视频的根目录。
- DST_ROOT: 存放生成音频的根目录。
- EXT: 视频的后缀名，如 mp4。
- N_WORKERS: 使用的进程数量。

成功提取出音频后，用户可参照 配置文件在线解码并生成梅尔频谱。如果音频文件的目录结构与帧文件夹一致，用户可以直接使用帧数据所用的标注文件作为音频数据的标注文件。在线解码的缺陷在于速度较慢，因此，MMAction2 也提供如下脚本用于离线地生成梅尔频谱。

```
cd $MMACTION2
python tools/data/build_audio_features.py ${AUDIO_HOME_PATH} ${SPECTROGRAM_SAVE_PATH}␣
↪[--level ${LEVEL}] \
    [--ext $EXT] [--num-workers $N_WORKERS] [--part $PART]
```

- AUDIO_HOME_PATH: 音频文件的根目录。
- SPECTROGRAM_SAVE_PATH: 存放生成音频特征的根目录。
- EXT: 音频的后缀名，如 m4a。
- N_WORKERS: 使用的进程数量。
- PART: 将完整的解码任务分为几部分并执行其中一份。如 2/5 表示将所有待解码数据分成 5 份，并对其中的第 2 份进行解码。这一选项在用户有多台机器时发挥作用。

梅尔频谱特征所对应的标注文件与帧文件夹一致，用户可以直接复制 dataset_[train/val]_list_rawframes.txt 并将其重命名为 dataset_[train/val]_list_audio_feature.txt。

CHAPTER 7

支持的数据集

- 支持的动作识别数据集：
  - *UCF101* [主页].
  - *HMDB51* [主页].
  - *Kinetics-[400/600/700]* [主页]
  - *Something-Something V1* [主页]
  - *Something-Something V2* [主页]
  - *Moments in Time* [主页]
  - *Multi-Moments in Time* [主页]
  - *HVU* [主页]
  - *Jester* [主页]
  - *GYM* [主页]
  - *ActivityNet* [主页]
- 支持的时序动作检测数据集：
  - *ActivityNet* [主页]
  - *THUMOS14* [主页]
- 支持的时空动作检测数据集：
  - *AVA* [主页]

  - *UCF101-24* [主页]
  - *JHMDB* [主页]

- 基于人体骨架的动作识别数据集：
  - *PoseC3D Skeleton Dataset* [主页]

MMAction2 目前支持的数据集如上所列。MMAction2 在 `$MMACTION2/tools/data/` 路径下提供数据集准备脚本。每个数据集的详细准备教程也在 Readthedocs 中给出。

7.1 ActivityNet

7.1.1 简介

```
@article{Heilbron2015ActivityNetAL,
  title={ActivityNet: A large-scale video benchmark for human activity understanding},
  author={Fabian Caba Heilbron and Victor Escorcia and Bernard Ghanem and Juan Carlos
↪Niebles},
  journal={2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)},
  year={2015},
  pages={961-970}
}
```

用户可参考该数据集的 官网，以获取数据集相关的基本信息。对于时序动作检测任务，用户可以使用这个 代码库 提供的缩放过（rescaled）的 ActivityNet 特征，或者使用 MMAction2 进行特征提取（这将具有更高的精度）。MMAction2 同时提供了以上所述的两种数据使用流程。在数据集准备前，请确保命令行当前路径为 `$MMACTION2/tools/data/activitynet/`。

7.1.2 选项 1：用户可以使用这个代码库提供的特征

步骤 1. 下载标注文件

首先，用户可以使用以下命令下载标注文件。

```
bash download_feature_annotations.sh
```

步骤 2. 准备视频特征

之后，用户可以使用以下命令下载 ActivityNet 特征。

```
bash download_features.sh
```

步骤 3. 处理标注文件

之后，用户可以使用以下命令处理下载的标注文件，以便于训练和测试。该脚本会首先合并两个标注文件，然后再将其分为 `train`, `val` 和 `test` 三个部分。

```
python process_annotations.py
```

7.1.3 选项 2：使用 MMAction2 对官网提供的视频进行特征抽取

步骤 1. 下载标注文件

首先，用户可以使用以下命令下载标注文件。

```
bash download_annotations.sh
```

步骤 2. 准备视频

之后，用户可以使用以下脚本准备视频数据。该代码参考自 官方爬虫，该过程将会耗费较多时间。

```
bash download_videos.sh
```

由于 ActivityNet 数据集中的一些视频已经在 YouTube 失效，官网 在谷歌网盘和百度网盘提供了完整的数据集数据。如果用户想要获取失效的数据集，则需要填写 下载页面 中提供的 需求表格 以获取 7 天的下载权限。

MMAction2 同时也提供了 BSN 代码库 的标注文件的下载步骤。

```
bash download_bsn_videos.sh
```

对于这种情况，该下载脚本将在下载后更新此标注文件，以确保每个视频都存在。

步骤 3. 抽取 RGB 帧和光流

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

可使用以下命令抽取视频帧和光流。

```
bash extract_frames.sh
```

以上脚本将会生成短边 256 分辨率的视频。如果用户想生成短边 320 分辨率的视频（即 320p），或者 340x256 的固定分辨率，用户可以通过改变参数由 `--new-short 256` 至 `--new-short 320`，或者 `--new-width 340 --new-height 256` 进行设置更多细节可参考 数据准备指南

步骤 4. 生成用于 ActivityNet 微调的文件列表

根据抽取的帧，用户可以生成视频级别（video-level）或者片段级别（clip-level）的文件列表，其可用于微调 ActivityNet。

```
python generate_rawframes_filelist.py
```

步骤 5. 在 ActivityNet 上微调 TSN 模型

用户可使用 `configs/recognition/tsn` 目录中的 ActivityNet 配置文件进行 TSN 模型微调。用户需要使用 Kinetics 相关模型（同时支持 RGB 模型与光流模型）进行预训练。

步骤 6. 使用预训练模型进行 ActivityNet 特征抽取

在 ActivityNet 上微调 TSN 模型之后，用户可以使用该模型进行 RGB 特征和光流特征的提取。

```
python tsn_feature_extraction.py --data-prefix ../../../data/ActivityNet/rawframes --
↪data-list ../../../data/ActivityNet/anet_train_video.txt --output-prefix ../../../
↪data/ActivityNet/rgb_feat --modality RGB --ckpt /path/to/rgb_checkpoint.pth

python tsn_feature_extraction.py --data-prefix ../../../data/ActivityNet/rawframes --
↪data-list ../../../data/ActivityNet/anet_val_video.txt --output-prefix ../../../
↪data/ActivityNet/rgb_feat --modality RGB --ckpt /path/to/rgb_checkpoint.pth

python tsn_feature_extraction.py --data-prefix ../../../data/ActivityNet/rawframes --
↪data-list ../../../data/ActivityNet/anet_train_video.txt --output-prefix ../../../
↪data/ActivityNet/flow_feat --modality Flow --ckpt /path/to/flow_checkpoint.pth

python tsn_feature_extraction.py --data-prefix ../../../data/ActivityNet/rawframes --
↪data-list ../../../data/ActivityNet/anet_val_video.txt --output-prefix ../../../
↪data/ActivityNet/flow_feat --modality Flow --ckpt /path/to/flow_checkpoint.pth
```

在提取完特征后，用户可以使用后处理脚本整合 RGB 特征和光流特征，生成 100-t X 400-d 维度的特征用于时序动作检测。

```
python activitynet_feature_postprocessing.py --rgb ../../../data/ActivityNet/rgb_feat␣
↪--flow ../../../data/ActivityNet/flow_feat --dest ../../../data/ActivityNet/
↪mmaction_feat
```

7.1.4 最后一步：检查文件夹结构

在完成所有 ActivityNet 数据集准备流程后，用户可以获得对应的特征文件，RGB + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，ActivityNet 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── ActivityNet

(若根据选项 1 进行数据处理)
|   |   ├── anet_anno_{train,val,test,full}.json
|   |   ├── anet_anno_action.json
|   |   ├── video_info_new.csv
|   |   ├── activitynet_feature_cuhk
|   |   |   ├── csv_mean_100
|   |   |   |   ├── v___c8enCfzqw.csv
|   |   |   |   ├── v___dXUJsj3yo.csv
|   |   |   |   ├── ..

(若根据选项 2 进行数据处理)
|   |   ├── anet_train_video.txt
|   |   ├── anet_val_video.txt
|   |   ├── anet_train_clip.txt
|   |   ├── anet_val_clip.txt
|   |   ├── activity_net.v1-3.min.json
|   |   ├── mmaction_feat
|   |   |   ├── v___c8enCfzqw.csv
|   |   |   ├── v___dXUJsj3yo.csv
|   |   |   ├── ..
|   |   ├── rawframes
|   |   |   ├── v___c8enCfzqw
|   |   |   |   ├── img_00000.jpg
```

(下页继续)

(续上页)

```
|   |   |   |   ├── flow_x_00000.jpg
|   |   |   |   ├── flow_y_00000.jpg
|   |   |   |   ├── ..
|   |   |   ├── ..
```

关于对 ActivityNet 进行训练和验证，可以参考 基础教程.

7.2 AVA

7.2.1 简介

```
@inproceedings{gu2018ava,
  title={Ava: A video dataset of spatio-temporally localized atomic visual actions},
  author={Gu, Chunhui and Sun, Chen and Ross, David A and Vondrick, Carl and␣
↪Pantofaru, Caroline and Li, Yeqing and Vijayanarasimhan, Sudheendra and Toderici,␣
↪George and Ricco, Susanna and Sukthankar, Rahul and others},
  booktitle={Proceedings of the IEEE Conference on Computer Vision and Pattern␣
↪Recognition},
  pages={6047--6056},
  year={2018}
}
```

请参照 官方网站 以获取数据集基本信息。在开始之前，用户需确保当前目录为 `$MMACTION2/tools/data/ava/`。

7.2.2 1. 准备标注文件

首先，用户可以使用如下脚本下载标注文件并进行预处理：

```
bash download_annotations.sh
```

这一命令将下载 `ava_v2.1.zip` 以得到 AVA v2.1 标注文件。如用户需要 AVA v2.2 标注文件，可使用以下脚本：

```
VERSION=2.2 bash download_annotations.sh
```

7.2.3 2. 下载视频

用户可以使用以下脚本准备视频，视频准备代码修改自 官方爬虫。注意这一步骤将花费较长时间。

```
bash download_videos.sh
```

亦可使用以下脚本，使用 python 并行下载 AVA 数据集视频：

```
bash download_videos_parallel.sh
```

7.2.4 3. 截取视频

截取每个视频中的 15 到 30 分钟，设定帧率为 30。

```
bash cut_videos.sh
```

7.2.5 4. 提取 RGB 帧和光流

在提取之前，请参考 安装教程 安装 denseflow。

如果用户有足够的 SSD 空间，那么建议将视频抽取为 RGB 帧以提升 I/O 性能。用户可以使用以下脚本为抽取得到的帧文件夹建立软连接：

```
## 执行以下脚本 (假设 SSD 被挂载在 "/mnt/SSD/")
mkdir /mnt/SSD/ava_extracted/
ln -s /mnt/SSD/ava_extracted/ ../data/ava/rawframes/
```

如果用户只使用 RGB 帧（由于光流提取非常耗时），可执行以下脚本使用 denseflow 提取 RGB 帧：

```
bash extract_rgb_frames.sh
```

如果用户未安装 denseflow，可执行以下脚本使用 ffmpeg 提取 RGB 帧：

```
bash extract_rgb_frames_ffmpeg.sh
```

如果同时需要 RGB 帧和光流，可使用如下脚本抽帧：

```
bash extract_frames.sh
```

7.2.6 5. 下载 AVA 上人体检测结果

以下脚本修改自 Long-Term Feature Banks。

可使用以下脚本下载 AVA 上预先计算的人体检测结果：

```
bash fetch_ava_proposals.sh
```

7.2.7 6. 目录结构

在完整完成 AVA 的数据处理后，将得到帧文件夹（RGB 帧和光流帧），视频以及标注文件。

在整个项目目录下（仅针对 AVA），最简目录结构如下所示：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── ava
|   |   ├── annotations
|   |   |   ├── ava_dense_proposals_train.FAIR.recall_93.9.pkl
|   |   |   ├── ava_dense_proposals_val.FAIR.recall_93.9.pkl
|   |   |   ├── ava_dense_proposals_test.FAIR.recall_93.9.pkl
|   |   |   ├── ava_train_v2.1.csv
|   |   |   ├── ava_val_v2.1.csv
|   |   |   ├── ava_train_excluded_timestamps_v2.1.csv
|   |   |   ├── ava_val_excluded_timestamps_v2.1.csv
|   |   |   ├── ava_action_list_v2.1_for_activitynet_2018.pbtxt
|   |   ├── videos
|   |   |   ├── 053oq2xB3oU.mkv
|   |   |   ├── 0f39OWEqJ24.mp4
|   |   |   ├── ...
|   |   ├── videos_15min
|   |   |   ├── 053oq2xB3oU.mkv
|   |   |   ├── 0f39OWEqJ24.mp4
|   |   |   ├── ...
|   |   ├── rawframes
|   |   |   ├── 053oq2xB3oU
|   |   |   |   ├── img_00001.jpg
|   |   |   |   ├── img_00002.jpg
|   |   |   |   ├── ...
```

关于 AVA 数据集上的训练与测试，请参照 基础教程。

7.3 Diving48

7.3.1 简介

```
@inproceedings{li2018resound,
  title={Resound: Towards action recognition without representation bias},
  author={Li, Yingwei and Li, Yi and Vasconcelos, Nuno},
  booktitle={Proceedings of the European Conference on Computer Vision (ECCV)},
  pages={513--528},
  year={2018}
}
```

用户可参考该数据集的 官网，以获取数据集相关的基本信息。在数据集准备前，请确保命令行当前路径为 `$MMACTION2/tools/data/diving48/`。

7.3.2 步骤 1. 下载标注文件

用户可以使用以下命令下载标注文件（考虑到标注的准确性，这里仅下载 V2 版本）。

```
bash download_annotations.sh
```

7.3.3 步骤 2. 准备视频

用户可以使用以下命令下载视频。

```
bash download_videos.sh
```

7.3.4 Step 3. 抽取 RGB 帧和光流

如果用户只想使用视频加载训练，则该部分是 **可选项**。

官网提供的帧压缩包并不完整。若想获取完整的数据，可以使用以下步骤解帧。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果拥有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 中。

可以运行以下命令为 SSD 建立软链接。

```
## 执行这两行进行抽取 (假设 SSD 挂载在 "/mnt/SSD/")
mkdir /mnt/SSD/diving48_extracted/
ln -s /mnt/SSD/diving48_extracted/ ../../../data/diving48/rawframes
```

如果用户需要抽取 RGB 帧（因为抽取光流的过程十分耗时），可以考虑运行以下命令使用 denseflow **只抽取 RGB 帧**。

```
cd $MMACTION2/tools/data/diving48/
bash extract_rgb_frames.sh
```

如果用户没有安装 denseflow，则可以运行以下命令使用 OpenCV 抽取 RGB 帧。然而，该方法只能抽取与原始视频分辨率相同的帧。

```
cd $MMACTION2/tools/data/diving48/
bash extract_rgb_frames_opencv.sh
```

如果用户想抽取 RGB 帧和光流，则可以运行以下脚本进行抽取。

```
cd $MMACTION2/tools/data/diving48/
bash extract_frames.sh
```

7.3.5 步骤 4. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
bash generate_videos_filelist.sh
bash generate_rawframes_filelist.sh
```

7.3.6 步骤 5. 检查文件夹结构

在完成所有 Diving48 数据集准备流程后，用户可以获得对应的 RGB + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，Diving48 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── diving48
|   |   ├── diving48_{train,val}_list_rawframes.txt
|   |   ├── diving48_{train,val}_list_videos.txt
|   |   ├── annotations
|   |   |   ├── Diving48_V2_train.json
|   |   |   ├── Diving48_V2_test.json
|   |   |   ├── Diving48_vocab.json
|   |   ├── videos
```

(下页继续)

(续上页)

```
|   |   |   ├── _8Vy3dlHg2w_00000.mp4
|   |   |   ├── _8Vy3dlHg2w_00001.mp4
|   |   |   ├── ...
|   |   ├── rawframes
|   |   |   ├── 2x00lRzlTVQ_00000
|   |   |   |   ├── img_00001.jpg
|   |   |   |   ├── img_00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_x_00001.jpg
|   |   |   |   ├── flow_x_00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_y_00001.jpg
|   |   |   |   ├── flow_y_00002.jpg
|   |   |   |   ├── ...
|   |   |   ├── 2x00lRzlTVQ_00001
|   |   |   ├── ...
```

关于对 Diving48 进行训练和验证，可以参考 基础教程。

7.4 GYM

7.4.1 简介

```
@inproceedings{shao2020finegym,
  title={Finegym: A hierarchical video dataset for fine-grained action understanding},
  author={Shao, Dian and Zhao, Yue and Dai, Bo and Lin, Dahua},
  booktitle={Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern␣
↪Recognition},
  pages={2616--2625},
  year={2020}
}
```

请参照 项目主页 及 原论文 以获取数据集基本信息。MMAction2 当前支持 GYM99 的数据集预处理。在开始之前，用户需确保当前目录为 `$MMACTION2/tools/data/gym/`。

7.4.2 1. 准备标注文件

首先，用户可以使用如下脚本下载标注文件并进行预处理：

```
bash download_annotations.sh
```

7.4.3 2. 准备视频

用户可以使用以下脚本准备视频，视频准备代码修改自 ActivityNet 爬虫。注意这一步骤将花费较长时间。

```
bash download_videos.sh
```

7.4.4 3. 裁剪长视频至动作级别

用户首先需要使用以下脚本将 GYM 中的长视频依据标注文件裁剪至动作级别。

```
python trim_event.py
```

7.4.5 4. 裁剪动作视频至分动作级别

随后，用户需要使用以下脚本将 GYM 中的动作视频依据标注文件裁剪至分动作级别。将视频的裁剪分成两个级别可以带来更高的效率（在长视频中裁剪多个极短片段异常耗时）。

```
python trim_subaction.py
```

7.4.6 5. 提取 RGB 帧和光流

如果用户仅使用 video loader，则可以跳过本步。

在提取之前，请参考 安装教程 安装 denseflow。

用户可使用如下脚本同时抽取 RGB 帧和光流（提取光流时使用 tvl1 算法）：

```
bash extract_frames.sh
```

7.4.7 6. 基于提取出的分动作生成文件列表

用户可使用以下脚本为 GYM99 生成训练及测试的文件列表：

```
python generate_file_list.py
```

7.4.8 7. 目录结构

在完整完成 GYM 的数据处理后，将得到帧文件夹（RGB 帧和光流帧），动作视频片段，分动作视频片段以及训练测试所用标注文件。

在整个项目目录下（仅针对 GYM），完整目录结构如下所示：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── gym
|   |   ├── annotations
|   |   |   ├── gym99_train_org.txt
|   |   |   ├── gym99_val_org.txt
|   |   |   ├── gym99_train.txt
|   |   |   ├── gym99_val.txt
|   |   |   ├── annotation.json
|   |   |   └── event_annotation.json
|   |   ├── videos
|   |   |   ├── 0LtLS9wROrk.mp4
|   |   |   ├── ...
|   |   |   └── zfqS-wCJSsw.mp4
|   |   ├── events
|   |   |   ├── 0LtLS9wROrk_E_002407_002435.mp4
|   |   |   ├── ...
|   |   |   └── zfqS-wCJSsw_E_006732_006824.mp4
|   |   ├── subactions
|   |   |   ├── 0LtLS9wROrk_E_002407_002435_A_0003_0005.mp4
|   |   |   ├── ...
|   |   |   └── zfqS-wCJSsw_E_006244_006252_A_0000_0007.mp4
|   |   └── subaction_frames
```

关于 GYM 数据集上的训练与测试，请参照 基础教程。

7.5 HMDB51

7.5.1 简介

```
@article{Kuehne2011HMDBAL,
  title={HMDB: A large video database for human motion recognition},
  author={Hilde Kuehne and Hueihan Jhuang and E. Garrote and T. Poggio and Thomas
↪Serre},
  journal={2011 International Conference on Computer Vision},
  year={2011},
  pages={2556-2563}
}
```

用户可以参照数据集 官网，获取数据集相关的基本信息。在准备数据集前，请确保命令行当前路径为 `$MMACTION2/tools/data/hmdb51/`。

为运行下面的 bash 脚本，需要安装 `unrar`。用户可运行 `sudo apt-get install unrar` 安装，或参照 setup，运行 `zzunrar.sh` 脚本实现无管理员权限下的简易安装。

7.5.2 步骤 1. 下载标注文件

首先，用户可使用以下命令下载标注文件。

```
bash download_annotations.sh
```

7.5.3 步骤 2. 下载视频

之后，用户可使用以下指令下载视频

```
bash download_videos.sh
```

7.5.4 步骤 3. 抽取帧和光流

如果用户只想使用视频加载训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果用户有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 上。用户可使用以下命令为 SSD 建立软链接。

```
## 执行这两行指令进行抽取（假设 SSD 挂载在 "/mnt/SSD/" 上）
mkdir /mnt/SSD/hmdb51_extracted/
ln -s /mnt/SSD/hmdb51_extracted/ ../../../data/hmdb51/rawframes
```

如果用户需要抽取 RGB 帧（因为抽取光流的过程十分耗时），可以考虑运行以下命令使用 denseflow **只抽取 RGB 帧**。

```
bash extract_rgb_frames.sh
```

如果用户没有安装 denseflow，则可以运行以下命令使用 OpenCV 抽取 RGB 帧。然而，该方法只能抽取与原始视频分辨率相同的帧。

```
bash extract_rgb_frames_opencv.sh
```

如果用户想抽取 RGB 帧和光流，则可以运行以下脚本，使用“tvl1”算法进行抽取。

```
bash extract_frames.sh
```

7.5.5 步骤 4. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
bash generate_rawframes_filelist.sh
bash generate_videos_filelist.sh
```

7.5.6 步骤 5. 检查目录结构

在完成 HMDB51 数据集准备流程后，用户可以得到 HMDB51 的 RGB 帧 + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，HMDB51 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
│   ├── hmdb51
│   │   ├── hmdb51_{train,val}_split_{1,2,3}_rawframes.txt
│   │   ├── hmdb51_{train,val}_split_{1,2,3}_videos.txt
│   │   ├── annotations
│   │   ├── videos
│   │   │   ├── brush_hair
```

(下页继续)

(续上页)

```
|   |   |   |   ├── April_09_brush_hair_u_nm_np1_ba_goo_0.avi

|   |   |   ├── wave
|   |   |   |   ├── 20060723sfjffbartsinger_wave_f_cm_np1_ba_med_0.avi
|   |   ├── rawframes
|   |   |   ├── brush_hair
|   |   |   |   ├── April_09_brush_hair_u_nm_np1_ba_goo_0
|   |   |   |   |   ├── img_00001.jpg
|   |   |   |   |   ├── img_00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── flow_x_00001.jpg
|   |   |   |   |   ├── flow_x_00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── flow_y_00001.jpg
|   |   |   |   |   ├── flow_y_00002.jpg
|   |   |   ├── ...
|   |   |   ├── wave
|   |   |   |   ├── 20060723sfjffbartsinger_wave_f_cm_np1_ba_med_0
|   |   |   |   ├── ...
|   |   |   |   ├── winKen_wave_u_cm_np1_ri_bad_1
```

关于对 HMDB51 进行训练和验证，可以参照 基础教程。

7.6 HVU

7.6.1 简介

```
@article{Diba2019LargeSH,
  title={Large Scale Holistic Video Understanding},
  author={Ali Diba and M. Fayyaz and Vivek Sharma and Manohar Paluri and Jurgen Gall␣
↪and R. Stiefelhagen and L. Gool},
  journal={arXiv: Computer Vision and Pattern Recognition},
  year={2019}
}
```

请参照 官方项目 及 原论文 以获取数据集基本信息。在开始之前，用户需确保当前目录为 `$MMACTION2/tools/data/hvu/`。

7.6.2 1. 准备标注文件

首先，用户可以使用如下脚本下载标注文件并进行预处理：

```
bash download_annotations.sh
```

此外，用户可使用如下命令解析 HVU 的标签列表：

```
python parse_tag_list.py
```

7.6.3 2. 准备视频

用户可以使用以下脚本准备视频，视频准备代码修改自 ActivityNet 爬虫。注意这一步骤将花费较长时间。

```
bash download_videos.sh
```

7.6.4 3. 提取 RGB 帧和光流

如果用户仅使用 video loader，则可以跳过本步。

在提取之前，请参考 安装教程 安装 denseflow。

用户可使用如下脚本同时抽取 RGB 帧和光流：

```
bash extract_frames.sh
```

该脚本默认生成短边长度为 256 的帧，可参考 数据准备 获得更多细节。

7.6.5 4. 生成文件列表

用户可以使用以下两个脚本分别为视频和帧文件夹生成文件列表：

```
bash generate_videos_filelist.sh
## 为帧文件夹生成文件列表
bash generate_rawframes_filelist.sh
```

7.6.6 5. 为每个 tag 种类生成文件列表

若用户需要为 HVU 数据集的每个 tag 种类训练识别模型，则需要进行此步骤。

步骤 4 中生成的文件列表包含不同类型的标签，仅支持使用 HVUDataset 进行涉及多个标签种类的多任务学习。加载数据的过程中需要使用 `LoadHVULabel` 类进行多类别标签的加载，训练过程中使用 `HVULoss` 作为损失函数。

如果用户仅需训练某一特定类别的标签，例如训练一识别模型用于识别 HVU 中 `action` 类别的标签，则建议使用如下脚本为特定标签种类生成文件列表。新生成的列表将只含有特定类别的标签，因此可使用 `VideoDataset` 或 `RawframeDataset` 进行加载。训训练过程中使用 `BCELossWithLogits` 作为损失函数。

以下脚本为类别为 ${category} 的标签生成文件列表，注意仅支持 HVU 数据集包含的 6 种标签类别: action, attribute, concept, event, object, scene。

```
python generate_sub_file_list.py path/to/filelist.json ${category}
```

对于类别 ${category}，生成的标签列表文件名中将使用 `hvu_${category}` 替代 `hvu`。例如，若原指定文件名为 `hvu_train.json`，则对于类别 action，生成的文件列表名为 `hvu_action_train.json`。

7.6.7 6. 目录结构

在完整完成 HVU 的数据处理后，将得到帧文件夹（RGB 帧和光流帧），视频以及标注文件。

在整个项目目录下（仅针对 HVU），完整目录结构如下所示：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
│   ├── hvu
│   │   ├── hvu_train_video.json
│   │   ├── hvu_val_video.json
│   │   ├── hvu_train.json
│   │   ├── hvu_val.json
│   │   ├── annotations
│   │   ├── videos_train
│   │   │   ├── OLpWTpTC4P8_000570_000670.mp4
│   │   │   ├── xsPKW4tZZBc_002330_002430.mp4
│   │   │   ├── ...
│   │   ├── videos_val
│   │   ├── rawframes_train
│   │   ├── rawframes_val
```

关于 HVU 数据集上的训练与测试，请参照 基础教程。

7.7 Jester

7.7.1 简介

```
@InProceedings{Materzynska_2019_ICCV,
  author = {Materzynska, Joanna and Berger, Guillaume and Bax, Ingo and Memisevic,␣
↪Roland},
  title = {The Jester Dataset: A Large-Scale Video Dataset of Human Gestures},
  booktitle = {Proceedings of the IEEE/CVF International Conference on Computer␣
↪Vision (ICCV) Workshops},
  month = {Oct},
  year = {2019}
}
```

用户可以参照数据集 官网，获取数据集相关的基本信息。在准备数据集前，请确保命令行当前路径为 `$MMACTION2/tools/data/jester/`。

7.7.2 步骤 1. 下载标注文件

首先，用户需要在 官网 完成注册，才能下载标注文件。下载好的标注文件需要放在 `$MMACTION2/data/jester/annotations` 文件夹下。

7.7.3 步骤 2. 准备 RGB 帧

jester 官网 并未提供原始视频文件，只提供了对原视频文件进行抽取得到的 RGB 帧，用户可在 jester 官网 直接下载。

将下载好的压缩文件放在 `$MMACTION2/data/jester/` 文件夹下，并使用以下脚本进行解压。

```
cd $MMACTION2/data/jester/
cat 20bn-jester-v1-?? | tar zx
cd $MMACTION2/tools/data/jester/
```

如果用户只想使用 RGB 帧，则可以跳过中间步骤至步骤 5 以直接生成视频帧的文件列表。由于官网的 JPG 文件名形如 “%05d.jpg”（比如，” 00001.jpg”），需要在配置文件的 `data.train`, `data.val` 和 `data.test` 处添加 `"filename_tmpl='{:05}.jpg'"` 代码，以修改文件名模板。

```
data = dict(
    videos_per_gpu=16,
```

(下页继续)

(续上页)

```
    workers_per_gpu=2,
    train=dict(
        type=dataset_type,
        ann_file=ann_file_train,
        data_prefix=data_root,
        filename_tmpl='{:05}.jpg',
        pipeline=train_pipeline),
    val=dict(
        type=dataset_type,
        ann_file=ann_file_val,
        data_prefix=data_root_val,
        filename_tmpl='{:05}.jpg',
        pipeline=val_pipeline),
    test=dict(
        type=dataset_type,
        ann_file=ann_file_test,
        data_prefix=data_root_val,
        filename_tmpl='{:05}.jpg',
        pipeline=test_pipeline))
```

7.7.4 步骤 3. 抽取光流

如果用户只想使用 RGB 帧训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果拥有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 中。

可以运行以下命令为 SSD 建立软链接。

```
## 执行这两行进行抽取 (假设 SSD 挂载在 "/mnt/SSD/")
mkdir /mnt/SSD/jester_extracted/
ln -s /mnt/SSD/jester_extracted/ ../../../data/jester/rawframes
```

如果想抽取光流，则可以运行以下脚本从 RGB 帧中抽取出光流。

```
cd $MMACTION2/tools/data/jester/
bash extract_flow.sh
```

7.7.5 步骤 4: 编码视频

如果用户只想使用 RGB 帧训练，则该部分是 **可选项**。

用户可以运行以下命令进行视频编码。

```
cd $MMACTION2/tools/data/jester/
bash encode_videos.sh
```

7.7.6 步骤 5. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
cd $MMACTION2/tools/data/jester/
bash generate_{rawframes, videos}_filelist.sh
```

7.7.7 步骤 6. 检查文件夹结构

在完成所有 Jester 数据集准备流程后，用户可以获得对应的 RGB + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，Jester 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── jester
|   |   ├── jester_{train,val}_list_rawframes.txt
|   |   ├── jester_{train,val}_list_videos.txt
|   |   ├── annotations
|   |   ├── videos
|   |   |   ├── 1.mp4
|   |   |   ├── 2.mp4
|   |   |   ├──...
|   |   ├── rawframes
|   |   |   ├── 1
|   |   |   |   ├── 00001.jpg
|   |   |   |   ├── 00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_x_00001.jpg
|   |   |   |   ├── flow_x_00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_y_00001.jpg
```

(下页继续)

(续上页)

```
|   |   |   |   ├── flow_y_00002.jpg
|   |   |   |   ├── ...
|   |   |   ├── 2
|   |   |   ├── ...
```

关于对 jester 进行训练和验证，可以参考 基础教程。

7.8 JHMDB

7.8.1 简介

```
@inproceedings{Jhuang:ICCV:2013,
    title = {Towards understanding action recognition},
    author = {H. Jhuang and J. Gall and S. Zuffi and C. Schmid and M. J. Black},
    booktitle = {International Conf. on Computer Vision (ICCV)},
    month = Dec,
    pages = {3192-3199},
    year = {2013}
}
```

用户可参考该数据集的 官网，以获取数据集相关的基本信息。在数据集准备前，请确保命令行当前路径为 `$MMACTION2/tools/data/jhmdb/`。

7.8.2 下载和解压

用户可以从 这里 下载 RGB 帧，光流和真实标签文件。该数据由 MOC 代码库提供，参考自 act-detector。

用户在下载 `JHMDB.tar.gz` 文件后，需将其放置在 `$MMACTION2/tools/data/jhmdb/` 目录下，并使用以下指令进行解压：

```
tar -zxvf JHMDB.tar.gz
```

如果拥有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 中。

可以运行以下命令为 SSD 建立软链接。

```
## 执行这两行进行抽取 (假设 SSD 挂载在 "/mnt/SSD/")
mkdir /mnt/SSD/JHMDB/
ln -s /mnt/SSD/JHMDB/ ../../../data/jhmdb
```

7.8.3 检查文件夹结构

完成解压后，用户将得到 FlowBrox04 文件夹，Frames 文件夹和 JHMDB-GT.pkl 文件。

在整个 MMAction2 文件夹下，JHMDB 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── jhmdb
|   |   ├── FlowBrox04
|   |   |   ├── brush_hair
|   |   |   |   ├── April_09_brush_hair_u_nm_np1_ba_goo_0
|   |   |   |   |   ├── 00001.jpg
|   |   |   |   |   ├── 00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── 00039.jpg
|   |   |   |   |   ├── 00040.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── Trannydude___Brushing_SyntheticHair___OhNOES!__those_fukin_knots!_
↪brush_hair_u_nm_np1_fr_goo_2
|   |   |   ├── ...
|   |   |   ├── wave
|   |   |   |   ├── 21_wave_u_nm_np1_fr_goo_5
|   |   |   |   ├── ...
|   |   |   |   ├── Wie_man_winkt!!_wave_u_cm_np1_fr_med_0
|   |   ├── Frames
|   |   |   ├── brush_hair
|   |   |   |   ├── April_09_brush_hair_u_nm_np1_ba_goo_0
|   |   |   |   |   ├── 00001.png
|   |   |   |   |   ├── 00002.png
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── 00039.png
|   |   |   |   |   ├── 00040.png
|   |   |   |   ├── ...
|   |   |   |   ├── Trannydude___Brushing_SyntheticHair___OhNOES!__those_fukin_knots!_
↪brush_hair_u_nm_np1_fr_goo_2
|   |   |   ├── ...
|   |   |   ├── wave
|   |   |   |   ├── 21_wave_u_nm_np1_fr_goo_5
|   |   |   |   ├── ...
|   |   |   |   ├── Wie_man_winkt!!_wave_u_cm_np1_fr_med_0
|   |   ├── JHMDB-GT.pkl
```

(下页继续)

(续上页)

```
```

注意：`JHMDB-GT.pkl` 作为一个缓存文件，它包含 6 个项目：

1. `labels` (list)：21 个行为类别名称组成的列表
2. `gttubes` (dict)：每个视频对应的基准 tubes 组成的字典 **gttube** 是由标签索引和 tube 列表组成的字典 **tube** 是一个 `nframes` 行和 5 列的 numpy array，每一列的形式如 `<frame index> <x1> <y1> <x2> <y2>`
3. `nframes` (dict)：用以表示每个视频对应的帧数，如 `'walk/Panic_in_the_Streets_walk_u_cm_np1_ba_med_5': 16`
4. `train_videos` (list)：包含 `nsplits=1` 的元素，每一项都包含了训练视频的列表
5. `test_videos` (list)：包含 `nsplits=1` 的元素，每一项都包含了测试视频的列表
6. `resolution` (dict)：每个视频对应的分辨率（形如 (h,w)），如 `'pour/Bartender_School_Students_Practice_pour_u_cm_np1_fr_med_1': (240, 320)`

7.9 Kinetics-[400/600/700]

7.9.1 简介

```
@inproceedings{inproceedings,
  author = {Carreira, J. and Zisserman, Andrew},
  year = {2017},
  month = {07},
  pages = {4724-4733},
  title = {Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset},
  doi = {10.1109/CVPR.2017.502}
}
```

请参照 官方网站 以获取数据集基本信息。此脚本用于准备数据集 kinetics400，kinetics600，kinetics700。为准备 kinetics 数据集的不同版本，用户需将脚本中的 `${DATASET}` 赋值为数据集对应版本名称，可选项为 `kinetics400`，`kinetics600`，`kinetics700`。在开始之前，用户需确保当前目录为 `$MMACTION2/tools/data/${DATASET}/`。

注：由于部分 YouTube 链接失效，爬取的 Kinetics 数据集大小可能与原版不同。以下是我们所使用 Kinetics 数据集的大小：

7.9.2 1. 准备标注文件

首先，用户可以使用如下脚本从 Kinetics 数据集官网下载标注文件并进行预处理：

```
bash download_annotations.sh ${DATASET}
```

由于部分视频的 URL 不可用，当前官方标注中所含视频数量可能小于初始版本。所以 MMAction2 提供了另一种方式以获取初始版本标注作为参考。在这其中，Kinetics400 和 Kinetics600 的标注文件来自 官方爬虫，Kinetics700 的标注文件于 05/02/2021 下载自 网站。

```
bash download_backup_annotations.sh ${DATASET}
```

7.9.3 2. 准备视频

用户可以使用以下脚本准备视频，视频准备代码修改自 官方爬虫。注意这一步骤将花费较长时间。

```
bash download_videos.sh ${DATASET}
```

重要提示：如果在此之前已下载好 Kinetics 数据集的视频，还需使用重命名脚本来替换掉类名中的空格：

```
bash rename_classnames.sh ${DATASET}
```

为提升解码速度，用户可以使用以下脚本将原始视频缩放至更小的分辨率（利用稠密编码）：

```
python ../resize_videos.py ../../../data/${DATASET}/videos_train/ ../../../data/$
↪{DATASET}/videos_train_256p_dense_cache --dense --level 2
```

也可以从 Academic Torrents 中下载短边长度为 256 的 kinetics400 和 kinetics700，或从 Common Visual Data Foundation 维护的 cvdfoundation/kinetics-dataset 中下载 Kinetics400/Kinetics600/Kinetics-700-2020。

7.9.4 3. 提取 RGB 帧和光流

如果用户仅使用 video loader，则可以跳过本步。

在提取之前，请参考 安装教程 安装 denseflow。

如果用户有足够的 SSD 空间，那么建议将视频抽取为 RGB 帧以提升 I/O 性能。用户可以使用以下脚本为抽取得到的帧文件夹建立软连接：

```
## 执行以下脚本 (假设 SSD 被挂载在 "/mnt/SSD/")
mkdir /mnt/SSD/${DATASET}_extracted_train/
ln -s /mnt/SSD/${DATASET}_extracted_train/ ../../../data/${DATASET}/rawframes_train/
mkdir /mnt/SSD/${DATASET}_extracted_val/
ln -s /mnt/SSD/${DATASET}_extracted_val/ ../../../data/${DATASET}/rawframes_val/
```

如果用户只使用 RGB 帧（由于光流提取非常耗时），可以考虑执行以下脚本，仅用 denseflow 提取 RGB 帧：

```
bash extract_rgb_frames.sh ${DATASET}
```

如果用户未安装 denseflow，以下脚本可以使用 OpenCV 进行 RGB 帧的提取，但视频原分辨率大小会被保留：

```
bash extract_rgb_frames_opencv.sh ${DATASET}
```

如果同时需要 RGB 帧和光流，可使用如下脚本抽帧：

```
bash extract_frames.sh ${DATASET}
```

以上的命令生成短边长度为 256 的 RGB 帧和光流帧。如果用户需要生成短边长度为 320 的帧 (320p)，或是固定分辨率为 340 x 256 的帧，可改变参数 `--new-short 256` 为 `--new-short 320` 或 `--new-width 340 --new-height 256`。更多细节可以参考 数据准备。

7.9.5 4. 生成文件列表

用户可以使用以下两个脚本分别为视频和帧文件夹生成文件列表：

```
bash generate_videos_filelist.sh ${DATASET}
## 为帧文件夹生成文件列表
bash generate_rawframes_filelist.sh ${DATASET}
```

7.9.6 5. 目录结构

在完整完成 Kinetics 的数据处理后，将得到帧文件夹（RGB 帧和光流帧），视频以及标注文件。

在整个项目目录下（仅针对 Kinetics），最简目录结构如下所示：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── ${DATASET}
|   |   ├── ${DATASET}_train_list_videos.txt
|   |   ├── ${DATASET}_val_list_videos.txt
|   |   ├── annotations
|   |   ├── videos_train
|   |   ├── videos_val
|   |   |   ├── abseiling
|   |   |   |   ├── 0wR5jVB-WPk_000417_000427.mp4
|   |   |   |   ├── ...
```

(下页继续)

(续上页)

```
|   |   |   ├── ...
|   |   |   ├── wrapping_present
|   |   |   ├── ...
|   |   |   ├── zumba
|   |   ├── rawframes_train
|   |   ├── rawframes_val
```

关于 Kinetics 数据集上的训练与测试，请参照 基础教程。

7.10 Moments in Time

7.10.1 简介

```
@article{monfortmoments,
    title={Moments in Time Dataset: one million videos for event understanding},
    author={Monfort, Mathew and Andonian, Alex and Zhou, Bolei and Ramakrishnan,␣
↪Kandan and Bargal, Sarah Adel and Yan, Tom and Brown, Lisa and Fan, Quanfu and␣
↪Gutfruend, Dan and Vondrick, Carl and others},
    journal={IEEE Transactions on Pattern Analysis and Machine Intelligence},
    year={2019},
    issn={0162-8828},
    pages={1--8},
    numpages={8},
    doi={10.1109/TPAMI.2019.2901464},
}
```

用户可以参照数据集 官网，获取数据集相关的基本信息。在准备数据集前，请确保命令行当前路径为 `$MMACTION2/tools/data/mit/`。

7.10.2 步骤 1. 准备标注文件和视频文件

首先，用户需要访问官网，填写申请表来下载数据集。在得到下载链接后，用户可以使用 `bash preprocess_data.sh` 来准备标注文件和视频。请注意此脚本并没有下载标注和视频文件，用户需要根据脚本文件中的注释，提前下载好数据集，并放/软链接到合适的位置。

为加快视频解码速度，用户需要缩小原视频的尺寸，可使用以下命令获取密集编码版视频：

```
python ../resize_videos.py ../../../data/mit/videos/ ../../../data/mit/videos_256p_
↪dense_cache --dense --level 2
```

7.10.3 Step 2. 抽取帧和光流

如果用户只想使用视频加载训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果用户有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 上。用户可使用以下命令为 SSD 建立软链接。

```
## 执行这两行指令进行抽取（假设 SSD 挂载在 "/mnt/SSD/" 上）
mkdir /mnt/SSD/mit_extracted/
ln -s /mnt/SSD/mit_extracted/ ../../../data/mit/rawframes
```

如果用户需要抽取 RGB 帧（因为抽取光流的过程十分耗时），可以考虑运行以下命令使用 denseflow **只抽取 RGB 帧**。

```
bash extract_rgb_frames.sh
```

如果用户没有安装 denseflow，则可以运行以下命令使用 OpenCV 抽取 RGB 帧。然而，该方法只能抽取与原始视频分辨率相同的帧。

```
bash extract_rgb_frames_opencv.sh
```

如果用户想抽取 RGB 帧和光流，则可以运行以下脚本进行抽取。

```
bash extract_frames.sh
```

7.10.4 步骤 3. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
bash generate_{rawframes, videos}_filelist.sh
```

7.10.5 步骤 4. 检查目录结构

在完成 Moments in Time 数据集准备流程后，用户可以得到 Moments in Time 的 RGB 帧 + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，Moments in Time 的文件结构如下：

```
mmaction2
├── data
│   └── mit
│       ├── annotations
```

(下页继续)

(续上页)

```
|       |   ├── license.txt
|       |   ├── moments_categories.txt
|       |   ├── README.txt
|       |   ├── trainingSet.csv
|       |   └── validationSet.csv
|       ├── mit_train_rawframe_anno.txt
|       ├── mit_train_video_anno.txt
|       ├── mit_val_rawframe_anno.txt
|       ├── mit_val_video_anno.txt
|       ├── rawframes
|       |   ├── training
|       |   |   ├── adult+female+singing
|       |   |   |   ├── 0P3XG_vf91c_35
|       |   |   |   |   ├── flow_x_00001.jpg
|       |   |   |   |   ├── flow_x_00002.jpg
|       |   |   |   |   ├── ...
|       |   |   |   |   ├── flow_y_00001.jpg
|       |   |   |   |   ├── flow_y_00002.jpg
|       |   |   |   |   ├── ...
|       |   |   |   |   ├── img_00001.jpg
|       |   |   |   |   └── img_00002.jpg
|       |   |   |   └── yt-zxQfALnTdfc_56
|       |   |   |   |   ├── ...
|       |   |   └── yawning
|       |   |       ├── _8zmP1e-EjU_2
|       |   |       |   ├── ...
|       |   └── validation
|       |   |       ├── ...
|       └── videos
|           ├── training
|           |   ├── adult+female+singing
|           |   |   ├── 0P3XG_vf91c_35.mp4
|           |   |   ├── ...
|           |   |   └── yt-zxQfALnTdfc_56.mp4
|           |   └── yawning
|           |       ├── ...
|           └── validation
|           |   ├── ...
└── mmaction
└── ...
```

关于对 Moments in Times 进行训练和验证，可以参照 基础教程。

7.11 Multi-Moments in Time

7.11.1 简介

```
@misc{monfort2019multimoments,
    title={Multi-Moments in Time: Learning and Interpreting Models for Multi-Action Video Understanding},
    author={Mathew Monfort and Kandan Ramakrishnan and Alex Andonian and Barry A McNamara and Alex Lascelles, Bowen Pan, Quanfu Fan, Dan Gutfreund, Rogerio Feris, Aude Oliva},
    year={2019},
    eprint={1911.00232},
    archivePrefix={arXiv},
    primaryClass={cs.CV}
}
```

用户可以参照数据集 官网，获取数据集相关的基本信息。在准备数据集前，请确保命令行当前路径为 `$MMACTION2/tools/data/mmit/`。

7.11.2 步骤 1. Prepare Annotations and Videos

首先，用户需要访问官网，填写申请表来下载数据集。在得到下载链接后，用户可以使用 `bash preprocess_data.sh` 来准备标注文件和视频。请注意此脚本并没有下载标注和视频文件，用户需要根据脚本文件中的注释，提前下载好数据集，并放/软链接到合适的位置。

为加快视频解码速度，用户需要缩小原视频的尺寸，可使用以下命令获取密集编码版视频：

```
python ../resize_videos.py ../../../data/mmit/videos/ ../../../data/mmit/videos_256p_dense_cache --dense --level 2
```

7.11.3 Step 2. 抽取帧和光流

如果用户只想使用视频加载训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果用户有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 上。用户可使用以下命令为 SSD 建立软链接。

```
## 执行这两行指令进行抽取（假设 SSD 挂载在 "/mnt/SSD/" 上）
mkdir /mnt/SSD/mmit_extracted/
ln -s /mnt/SSD/mmit_extracted/ ../../../data/mmit/rawframes
```

如果用户需要抽取 RGB 帧（因为抽取光流的过程十分耗时），可以考虑运行以下命令使用 denseflow **只抽取 RGB 帧**。

```
bash extract_rgb_frames.sh
```

如果用户没有安装 denseflow，则可以运行以下命令使用 OpenCV 抽取 RGB 帧。然而，该方法只能抽取与原始视频分辨率相同的帧。

```
bash extract_rgb_frames_opencv.sh
```

如果用户想抽取 RGB 帧和光流，则可以运行以下脚本进行抽取。

```
bash extract_frames.sh
```

7.11.4 步骤 3. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
bash generate_rawframes_filelist.sh
bash generate_videos_filelist.sh
```

7.11.5 步骤 4. 检查目录结构

在完成 Multi-Moments in Time 数据集准备流程后，用户可以得到 Multi-Moments in Time 的 RGB 帧 + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，Multi-Moments in Time 的文件结构如下：

```
mmaction2/
└── data
    └── mmit
        ├── annotations
        │   ├── moments_categories.txt
        │   ├── trainingSet.txt
        │   └── validationSet.txt
        ├── mmit_train_rawframes.txt
        ├── mmit_train_videos.txt
        ├── mmit_val_rawframes.txt
        ├── mmit_val_videos.txt
        ├── rawframes
        │   ├── 0-3-6-2-9-1-2-6-14603629126_5
        │   │   ├── flow_x_00001.jpg
        │   │   ├── flow_x_00002.jpg
```

(下页继续)

(续上页)

```
        |   |   ├── ...
        |   |   ├── flow_y_00001.jpg
        |   |   ├── flow_y_00002.jpg
        |   |   ├── ...
        |   |   ├── img_00001.jpg
        |   |   └── img_00002.jpg
        |   |   ├── ...
        |   └── yt-zxQfALnTdfc_56
        |   |   ├── ...
        |   └── ...

        └── videos
            └── adult+female+singing
                ├── 0-3-6-2-9-1-2-6-14603629126_5.mp4
                └── yt-zxQfALnTdfc_56.mp4
            └── ...
```

关于对 Multi-Moments in Time 进行训练和验证，可以参照 基础教程。

7.12 OmniSource

7.12.1 简介

```
@article{duan2020omni,
  title={Omni-sourced Webly-supervised Learning for Video Recognition},
  author={Duan, Haodong and Zhao, Yue and Xiong, Yuanjun and Liu, Wentao and Lin,␣
↪Dahua},
  journal={arXiv preprint arXiv:2003.13042},
  year={2020}
}
```

MMAction2 中发布了 OmniSource 网络数据集的一个子集 (来自论文 Omni-sourced Webly-supervised Learning for Video Recognition)。OmniSource 数据集中所有类别均来自 Kinetics-400。MMAction2 所提供的子集包含属于 Mini-Kinetics 数据集 200 类动作的网络数据 (Mini-inetics 数据集由论文 Rethinking Spatiotemporal Feature Learning: Speed-Accuracy Trade-offs in Video Classification 提出)。

MMAction2 提供所有数据源中属于 Mini-Kinetics 200 类动作的数据，这些数据源包含：Kinetics 数据集，Kinetics 原始数据集（未经裁剪的长视频），来自 Google 和 Instagram 的网络图片，来自 Instagram 的网络视频。为获取这一数据集，用户需先填写 数据申请表。在接收到申请后，下载链接将被发送至用户邮箱。由于发布的数据集均为爬取所得的原始数据，数据集较大，下载需要一定时间。下表中提供了 OmniSource 数据集各个分量的统计信息。

MMAction2 所发布的 OmniSource 数据集目录结构如下所示：

```
OmniSource/
├── annotations
│   ├── googleimage_200
│   │   ├── googleimage_200.txt                          从 Google 爬取到的所有图片列表
│   │   ├── tsn_8seg_googleimage_200_duplicate.txt       从 Google 爬取到的，疑似与 k200-val␣
↪中样本重复的正样本列表
│   │   ├── tsn_8seg_googleimage_200.txt                 从 Google 爬取到的，经过 teacher 模
型过滤的正样本列表
│   │   └── tsn_8seg_googleimage_200_wodup.txt           从 Google 爬取到的，经过 teacher 模
型过滤及去重的正样本列表
│   ├── insimage_200
│   │   ├── insimage_200.txt
│   │   ├── tsn_8seg_insimage_200_duplicate.txt
│   │   ├── tsn_8seg_insimage_200.txt
│   │   └── tsn_8seg_insimage_200_wodup.txt
│   ├── insvideo_200
│   │   ├── insvideo_200.txt
│   │   ├── slowonly_8x8_insvideo_200_duplicate.txt
│   │   ├── slowonly_8x8_insvideo_200.txt
│   │   └── slowonly_8x8_insvideo_200_wodup.txt
│   ├── k200_actions.txt                                 MiniKinetics 中 200 类动作的名称
│   ├── K400_to_MiniKinetics_classidx_mapping.json       Kinetics 中的类索引至␣
↪MiniKinetics 中的类索引的映射
│   ├── kinetics_200
│   │   ├── k200_train.txt
│   │   └── k200_val.txt
│   └── kinetics_raw_200
│       └── slowonly_8x8_kinetics_raw_200.json           经 teacher 模型过滤后的 Kinetics 原
始视频片段
├── googleimage_200                                      共 10 卷
│   ├── vol_0.tar
│   ├── ...
│   └── vol_9.tar
├── insimage_200                                         共 10 卷
│   ├── vol_0.tar
│   ├── ...
│   └── vol_9.tar
├── insvideo_200                                         共 20 卷
│   ├── vol_00.tar
│   ├── ...
│   └── vol_19.tar
├── kinetics_200_train
│   └── kinetics_200_train.tar
├── kinetics_200_val
```

(下页继续)

(续上页)

```
|   └── kinetics_200_val.tar
└── kinetics_raw_200_train                          共 16 卷
    ├── vol_0.tar
    ├── ...
    └── vol_15.tar
```

7.12.2 数据准备

用户需要首先完成数据下载，对于 kinetics_200 和三个网络数据集 googleimage_200, insimage_200, insvideo_200，用户仅需解压各压缩卷并将其合并至一处。

对于 Kinetics 原始视频，由于直接读取长视频非常耗时，用户需要先将其分割为小段。MMAction2 提供了名为 trim_raw_video.py 的脚本，用于将长视频分割至 10 秒的小段（分割完成后删除长视频）。用户可利用这一脚本分割长视频。

所有数据应位于 data/OmniSource/ 目录下。完成数据准备后，data/OmniSource/ 目录的结构应如下所示（为简洁，省去了训练及测试时未使用的文件）：

```
data/OmniSource/
├── annotations
|   ├── googleimage_200
|   |   └── tsn_8seg_googleimage_200_wodup.txt    Positive file list of images␣
↪crawled from Google, filtered by the teacher model, after de-duplication.
|   ├── insimage_200
|   |   └── tsn_8seg_insimage_200_wodup.txt
|   ├── insvideo_200
|   |   └── slowonly_8x8_insvideo_200_wodup.txt
|   ├── kinetics_200
|   |   ├── k200_train.txt
|   |   └── k200_val.txt
|   ├── kinetics_raw_200
|   |   └── slowonly_8x8_kinetics_raw_200.json    Kinetics Raw Clips filtered by the␣
↪teacher model.
|   └── webimage_200
|       └── tsn_8seg_webimage_200_wodup.txt       The union of `tsn_8seg_googleimage_
↪200_wodup.txt` and `tsn_8seg_insimage_200_wodup.txt`
├── googleimage_200
|   ├── 000
|   |   ├── 00
|   |   |   ├── 000001.jpg
|   |   |   ├── ...
|   |   |   └── 000901.jpg
|   |   ├── ...
```

(下页继续)

(续上页)

```
|   |   ├── 19
|   ├── ...
|   └── 199
├── insimage_200
|   ├── 000
|   |   ├── abseil
|   |   |   ├── 1J9tKWCNgV_0.jpg
|   |   |   ├── ...
|   |   |   └── 1J9tKWCNgV_0.jpg
|   |   ├── abseiling
|   ├── ...
|   └── 199
├── insvideo_200
|   ├── 000
|   |   ├── abseil
|   |   |   ├── B00arxogubl.mp4
|   |   |   ├── ...
|   |   |   └── BzYsP0HIvbt.mp4
|   |   ├── abseiling
|   ├── ...
|   └── 199
├── kinetics_200_train
|   ├── 0074cdXclLU.mp4
|   ├── ...
|   ├── zzzlyL61Fyo.mp4
├── kinetics_200_val
|   ├── 01fAWEHzudA.mp4
|   ├── ...
|   ├── zymA_6jZIz4.mp4
└── kinetics_raw_200_train
|   ├── pref_
|   |   ├── ___dTOdxzXY
|   |   |   ├── part_0.mp4
|   |   |   ├── ...
|   |   |   ├── part_6.mp4
|   |   ├── ...
|   |   └── _zygwGDE2EM
|   ├── ...
|   └── prefZ
```

7.13 骨架数据集

```
@misc{duan2021revisiting,
      title={Revisiting Skeleton-based Action Recognition},
      author={Haodong Duan and Yue Zhao and Kai Chen and Dian Shao and Dahua Lin and
↪Bo Dai},
      year={2021},
      eprint={2104.13586},
      archivePrefix={arXiv},
      primaryClass={cs.CV}
}
```

7.13.1 简介

MMAction2 发布 Revisiting Skeleton-based Action Recognition 论文中所使用的骨架标注。默认使用 Faster-RCNN 作为人体检测器，使用 HRNet-w32 作为单人姿态估计模型。对于 FineGYM 数据集，MMAction2 使用的是运动员的真实框标注，而非检测器所出的框。目前，MMAction2 已发布 FineGYM 和 NTURGB-D Xsub 部分的骨架标注，其他数据集的标注也将很快发布。

7.13.2 标注文件

目前，MMAction2 支持 HMDB51, UCF101, FineGYM 和 NTURGB+D 数据集。对于 FineGYM 数据集，用户可以使用以下脚本下载标注文件。

```
bash download_annotations.sh ${DATASET}
```

由于 NTURGB+D 数据集的 使用条例，MMAction2 并未直接发布实验中所使用的标注文件。因此，这里提供生成 NTURGB+D 数据集中视频的姿态标注文件，这将生成一个 dict 数据并将其保存为一个 pickle 文件。用户可以生成一个 list 用以包含对应视频的 dict 数据，并将其保存为一个 pickle 文件。之后，用户可以获得 `ntu60_xsub_train.pkl`, `ntu60_xsub_val.pkl`, `ntu120_xsub_train.pkl`, `ntu120_xsub_val.pkl` 文件用于训练。

对于无法进行姿态提取的用户，这里提供了上述流程的输出结果，分别对应 NTURGB-D 数据集的 4 个部分：

- ntu60_xsub_train: https://download.openmmlab.com/mmaction/posec3d/ntu60_xsub_train.pkl
- ntu60_xsub_val: https://download.openmmlab.com/mmaction/posec3d/ntu60_xsub_val.pkl
- ntu120_xsub_train: https://download.openmmlab.com/mmaction/posec3d/ntu120_xsub_train.pkl
- ntu120_xsub_val: https://download.openmmlab.com/mmaction/posec3d/ntu120_xsub_val.pkl
- hmdb51: https://download.openmmlab.com/mmaction/posec3d/hmdb51.pkl
- ucf101: https://download.openmmlab.com/mmaction/posec3d/ucf101.pkl

若想生成单个视频的 2D 姿态标注文件，首先，用户需要由源码安装 mmdetection 和 mmpose。之后，用户需要在 `ntu_pose_extraction.py` 中指定 `mmdet_root` 和 `mmpose_root` 变量。最后，用户可使用以下脚本进行 NTURGB+D 视频的姿态提取：

```
python ntu_pose_extraction.py S001C001P001R001A001_rgb.avi S001C001P001R001A001.pkl
```

在用户获得数据集某部分所有视频的姿态标注文件（如 `ntu60_xsub_val`）后，可以将其集合成一个 list 数据并保存为 `ntu60_xsub_val.pkl`。用户可用这些大型 pickle 文件进行训练和测试。

7.13.3 PoseC3D 的标注文件格式

这里简单介绍 PoseC3D 的标注文件格式。以 `gym_train.pkl` 为例：`gym_train.pkl` 存储一个长度为 20484 的 list，list 的每一项为单个视频的骨架标注 dict。每个 dict 的内容如下：

- keypoint：关键点坐标，大小为 N（## 人数）x T（时序长度）x K（# 关键点, 这里为 17）x 2 （x，y 坐标）的 numpy array 数据类型
- keypoint_score：关键点的置信分数，大小为 N（## 人数）x T（时序长度）x K（# 关键点, 这里为 17）的 numpy array 数据类型
- frame_dir: 对应视频名
- label: 动作类别
- img_shape: 每一帧图像的大小
- original_shape: 同 `img_shape`
- total_frames: 视频时序长度

如用户想使用自己的数据集训练 PoseC3D，可以参考 Custom Dataset Training。

7.13.4 可视化

为了可视化骨架数据，用户需要准备 RGB 的视频。详情可参考 visualize_heatmap_volume。这里提供一些 NTU-60 和 FineGYM 上的例子

7.13.5 如何将 NTU RGB+D 原始数据转化为 MMAction2 格式（转换好的标注文件目前仅适用于 GCN 模型）

这里介绍如何将 NTU RGB+D 原始数据转化为 MMAction2 格式。首先，需要从 https://github.com/shahroudy/NTURGB-D 下载原始 NTU-RGBD 60 和 NTU-RGBD 120 数据集的原始骨架数据。

对于 NTU-RGBD 60 数据集，可使用以下脚本

```
python gen_ntu_rgbd_raw.py --data-path your_raw_nturgbd60_skeleton_path --ignored-
↪sample-path NTU_RGBD_samples_with_missing_skeletons.txt --out-folder your_nturgbd60_
↪output_path --task ntu60
```

对于 NTU-RGBD 120 数据集，可使用以下脚本

```
python gen_ntu_rgbd_raw.py --data-path your_raw_nturgbd120_skeleton_path --ignored-
↪sample-path NTU_RGBD120_samples_with_missing_skeletons.txt --out-folder your_
↪nturgbd120_output_path --task ntu120
```

7.13.6 转换其他第三方项目的骨骼标注

MMAction2 提供脚本以将其他第三方项目的骨骼标注转至 MMAction2 格式，如：

- BABEL: `babel2mma2.py`

待办项：

- [x] FineGYM
- [x] NTU60_XSub
- [x] NTU120_XSub
- [x] NTU60_XView
- [x] NTU120_XSet
- [x] UCF101
- [x] HMDB51
- [ ] Kinetics

7.14 Something-Something V1

7.14.1 简介

```
@misc{goyal2017something,
      title={The "something something" video database for learning and evaluating␣
↪visual common sense},
      author={Raghav Goyal and Samira Ebrahimi Kahou and Vincent Michalski and Joanna␣
↪Materzyńska and Susanne Westphal and Heuna Kim and Valentin Haenel and Ingo Fruend␣
↪and Peter Yianilos and Moritz Mueller-Freitag and Florian Hoppe and Christian␣
↪Thurau and Ingo Bax and Roland Memisevic},
      year={2017},
```

(下页继续)

(续上页)

```
      eprint={1706.04261},
      archivePrefix={arXiv},
      primaryClass={cs.CV}
}
```

用户可参考该数据集的 官网，以获取数据集相关的基本信息。在数据集准备前，请确保命令行当前路径为 `$MMACTION2/tools/data/sthv1/`。

7.14.2 步骤 1. 下载标注文件

由于 Something-Something V1 的官方网站已经失效，用户需要通过第三方源下载原始数据集。下载好的标注文件需要放在 `$MMACTION2/data/sthv1/annotations` 文件夹下。

7.14.3 步骤 2. 准备 RGB 帧

官方数据集并未提供原始视频文件，只提供了对原视频文件进行抽取得到的 RGB 帧，用户可在第三方源直接下载视频帧。

将下载好的压缩文件放在 `$MMACTION2/data/sthv1/` 文件夹下，并使用以下脚本进行解压。

```
cd $MMACTION2/data/sthv1/
cat 20bn-something-something-v1-?? | tar zx
cd $MMACTION2/tools/data/sthv1/
```

如果用户只想使用 RGB 帧，则可以跳过中间步骤至步骤 5 以直接生成视频帧的文件列表。由于官网的 JPG 文件名形如 “%05d.jpg”（比如，” 00001.jpg”），需要在配置文件的 `data.train`, `data.val` 和 `data.test` 处添加 `"filename_tmpl='{:05}.jpg'"` 代码，以修改文件名模板。

```
data = dict(
    videos_per_gpu=16,
    workers_per_gpu=2,
    train=dict(
        type=dataset_type,
        ann_file=ann_file_train,
        data_prefix=data_root,
        filename_tmpl='{:05}.jpg',
        pipeline=train_pipeline),
    val=dict(
        type=dataset_type,
        ann_file=ann_file_val,
        data_prefix=data_root_val,
        filename_tmpl='{:05}.jpg',
```

(下页继续)

(续上页)

```
        pipeline=val_pipeline),
    test=dict(
        type=dataset_type,
        ann_file=ann_file_test,
        data_prefix=data_root_val,
        filename_tmpl='{:05}.jpg',
        pipeline=test_pipeline))
```

7.14.4 步骤 3. 抽取光流

如果用户只想使用原 RGB 帧加载训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果拥有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 中。

可以运行以下命令为 SSD 建立软链接。

```
## 执行这两行进行抽取（假设 SSD 挂载在 "/mnt/SSD/"）
mkdir /mnt/SSD/sthv1_extracted/
ln -s /mnt/SSD/sthv1_extracted/ ../../../data/sthv1/rawframes
```

如果想抽取光流，则可以运行以下脚本从 RGB 帧中抽取出光流。

```
cd $MMACTION2/tools/data/sthv1/
bash extract_flow.sh
```

7.14.5 步骤 4: 编码视频

如果用户只想使用 RGB 帧加载训练，则该部分是 **可选项**。

用户可以运行以下命令进行视频编码。

```
cd $MMACTION2/tools/data/sthv1/
bash encode_videos.sh
```

7.14.6 步骤 5. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
cd $MMACTION2/tools/data/sthv1/
bash generate_{rawframes, videos}_filelist.sh
```

7.14.7 步骤 6. 检查文件夹结构

在完成所有 Something-Something V1 数据集准备流程后，用户可以获得对应的 RGB + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，Something-Something V1 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── sthv1
|   |   ├── sthv1_{train,val}_list_rawframes.txt
|   |   ├── sthv1_{train,val}_list_videos.txt
|   |   ├── annotations
|   |   ├── videos
|   |   |   ├── 1.mp4
|   |   |   ├── 2.mp4
|   |   |   ├──...
|   |   ├── rawframes
|   |   |   ├── 1
|   |   |   |   ├── 00001.jpg
|   |   |   |   ├── 00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_x_00001.jpg
|   |   |   |   ├── flow_x_00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_y_00001.jpg
|   |   |   |   ├── flow_y_00002.jpg
|   |   |   |   ├── ...
|   |   |   ├── 2
|   |   |   ├── ...
```

关于对 Something-Something V1 进行训练和验证，可以参考 基础教程。

7.15 Something-Something V2

7.15.1 简介

```
@misc{goyal2017something,
      title={The "something something" video database for learning and evaluating visual common sense},
      author={Raghav Goyal and Samira Ebrahimi Kahou and Vincent Michalski and Joanna Materzyńska and Susanne Westphal and Heuna Kim and Valentin Haenel and Ingo Fruend and Peter Yianilos and Moritz Mueller-Freitag and Florian Hoppe and Christian Thurau and Ingo Bax and Roland Memisevic},
      year={2017},
      eprint={1706.04261},
      archivePrefix={arXiv},
      primaryClass={cs.CV}
}
```

用户可参考该数据集的 官网，以获取数据集相关的基本信息。在数据集准备前，请确保命令行当前路径为 `$MMACTION2/tools/data/sthv2/`。

7.15.2 步骤 1. 下载标注文件

首先，用户需要在 官网 完成注册，才能下载标注文件。下载好的标注文件需要放在 `$MMACTION2/data/sthv2/annotations` 文件夹下。

```
cd $MMACTION2/data/sthv2/annotations
unzip 20bn-something-something-download-package-labels.zip
find ./labels -name "*.json" -exec sh -c 'cp "$1" "something-something-v2-$(basename $1)"' _ {} \;
```

7.15.3 步骤 2. 准备视频

之后，用户可将下载好的压缩文件放在 `$MMACTION2/data/sthv2/` 文件夹下，并且使用以下指令进行解压。

```
cd $MMACTION2/data/sthv2/
cat 20bn-something-something-v2-?? | tar zx
cd $MMACTION2/tools/data/sthv2/
```

7.15.4 步骤 3. 抽取 RGB 帧和光流

如果用户只想使用视频加载训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果拥有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 中。

可以运行以下命令为 SSD 建立软链接。

```
## 执行这两行进行抽取（假设 SSD 挂载在 "/mnt/SSD/"）
mkdir /mnt/SSD/sthv2_extracted/
ln -s /mnt/SSD/sthv2_extracted/ ../../../data/sthv2/rawframes
```

如果用户需要抽取 RGB 帧（因为抽取光流的过程十分耗时），可以考虑运行以下命令使用 denseflow **只抽取 RGB 帧**。

```
cd $MMACTION2/tools/data/sthv2/
bash extract_rgb_frames.sh
```

如果用户没有安装 denseflow，则可以运行以下命令使用 OpenCV 抽取 RGB 帧。然而，该方法只能抽取与原始视频分辨率相同的帧。

```
cd $MMACTION2/tools/data/sthv2/
bash extract_rgb_frames_opencv.sh
```

如果用户想抽取 RGB 帧和光流，则可以运行以下脚本进行抽取。

```
cd $MMACTION2/tools/data/sthv2/
bash extract_frames.sh
```

7.15.5 步骤 4. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
cd $MMACTION2/tools/data/sthv2/
bash generate_{rawframes, videos}_filelist.sh
```

7.15.6 步骤 5. 检查文件夹结构

在完成所有 Something-Something V2 数据集准备流程后，用户可以获得对应的 RGB + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，Something-Something V2 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── sthv2
|   |   ├── sthv2_{train,val}_list_rawframes.txt
|   |   ├── sthv2_{train,val}_list_videos.txt
|   |   ├── annotations
|   |   ├── videos
|   |   |   ├── 1.mp4
|   |   |   ├── 2.mp4
|   |   |   ├──...
|   |   ├── rawframes
|   |   |   ├── 1
|   |   |   |   ├── img_00001.jpg
|   |   |   |   ├── img_00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_x_00001.jpg
|   |   |   |   ├── flow_x_00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_y_00001.jpg
|   |   |   |   ├── flow_y_00002.jpg
|   |   |   |   ├── ...
|   |   |   ├── 2
|   |   |   ├── ...
```

关于对 Something-Something V2 进行训练和验证，可以参考 基础教程。

7.16 THUMOS'14

7.16.1 简介

```
@misc{THUMOS14,
    author = {Jiang, Y.-G. and Liu, J. and Roshan Zamir, A. and Toderici, G. and
↪Laptev,
    I. and Shah, M. and Sukthankar, R.},
    title = {{THUMOS} Challenge: Action Recognition with a Large
    Number of Classes},
    howpublished = "\url{http://crcv.ucf.edu/THUMOS14/}",
    Year = {2014}
}
```

用户可以参照数据集 官网，获取数据集相关的基本信息。在准备数据集前，请确保命令行当前路径为 `$MMACTION2/tools/data/thumos14/`。

7.16.2 步骤 1. 下载标注文件

首先，用户可使用以下命令下载标注文件。

```
cd $MMACTION2/tools/data/thumos14/
bash download_annotations.sh
```

7.16.3 步骤 2. 下载视频

之后，用户可使用以下指令下载视频

```
cd $MMACTION2/tools/data/thumos14/
bash download_videos.sh
```

7.16.4 步骤 3. 抽取帧和光流

如果用户只想使用视频加载训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果用户有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 上。用户可使用以下命令为 SSD 建立软链接。

```
## 执行这两行指令进行抽取（假设 SSD 挂载在 "/mnt/SSD/" 上）
mkdir /mnt/SSD/thumos14_extracted/
ln -s /mnt/SSD/thumos14_extracted/ ../data/thumos14/rawframes/
```

如果用户需要抽取 RGB 帧（因为抽取光流的过程十分耗时），可以考虑运行以下命令使用 denseflow **只抽取 RGB 帧**。

```
cd $MMACTION2/tools/data/thumos14/
bash extract_rgb_frames.sh
```

如果用户没有安装 denseflow，则可以运行以下命令使用 OpenCV 抽取 RGB 帧。然而，该方法只能抽取与原始视频分辨率相同的帧。

```
cd $MMACTION2/tools/data/thumos14/
bash extract_rgb_frames_opencv.sh
```

如果用户想抽取 RGB 帧和光流，则可以运行以下脚本进行抽取。

```
cd $MMACTION2/tools/data/thumos14/
bash extract_frames.sh tvl1
```

7.16.5 步骤 4. 生成文件列表

如果用户不使用 SSN 模型，则该部分是 **可选项**。

可使用运行以下脚本下载预先计算的候选标签。

```
cd $MMACTION2/tools/data/thumos14/
bash fetch_tag_proposals.sh
```

7.16.6 步骤 5. 去规范化候选文件

如果用户不使用 SSN 模型，则该部分是 **可选项**。

可运行以下脚本，来根据本地原始帧的实际数量，去规范化预先计算的候选标签。

```
cd $MMACTION2/tools/data/thumos14/
bash denormalize_proposal_file.sh
```

7.16.7 步骤 6. 检查目录结构

在完成 THUMOS’14 数据集准备流程后，用户可以得到 THUMOS’14 的 RGB 帧 + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，THUMOS’14 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── thumos14
|   |   ├── proposals
|   |   |   ├── thumos14_tag_val_normalized_proposal_list.txt
|   |   |   ├── thumos14_tag_test_normalized_proposal_list.txt
|   |   ├── annotations_val
|   |   ├── annotations_test
|   |   ├── videos
|   |   |   ├── val
|   |   |   |   ├── video_validation_0000001.mp4
|   |   |   |   ├── ...
|   |   |   ├── test
|   |   |   |   ├── video_test_0000001.mp4
|   |   |   |   ├── ...
|   |   ├── rawframes
|   |   |   ├── val
|   |   |   |   ├── video_validation_0000001
|   |   |   |   |   ├── img_00001.jpg
|   |   |   |   |   ├── img_00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── flow_x_00001.jpg
|   |   |   |   |   ├── flow_x_00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── flow_y_00001.jpg
|   |   |   |   |   ├── flow_y_00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   ├── ...
|   |   |   ├── test
|   |   |   |   ├── video_test_0000001
```

关于对 THUMOS’14 进行训练和验证，可以参照 基础教程。

7.17 UCF-101

7.17.1 简介

```
@article{Soomro2012UCF101AD,
  title={UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild},
  author={K. Soomro and A. Zamir and M. Shah},
  journal={ArXiv},
  year={2012},
  volume={abs/1212.0402}
}
```

用户可参考该数据集的 官网，以获取数据集相关的基本信息。在数据集准备前，请确保命令行当前路径为 `$MMACTION2/tools/data/ucf101/`。

7.17.2 步骤 1. 下载标注文件

首先，用户可运行以下脚本下载标注文件。

```
bash download_annotations.sh
```

7.17.3 步骤 2. 准备视频文件

之后，用户可运行以下脚本准备视频文件。

```
bash download_videos.sh
```

用户可使用以下脚本，对原视频进行裁剪，得到密集编码且更小尺寸的视频。

```
python ../resize_videos.py ../../../data/ucf101/videos/ ../../../data/ucf101/videos_
↪256p_dense_cache --dense --level 2 --ext avi
```

7.17.4 步骤 3. 抽取视频帧和光流

如果用户只想使用视频加载训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果拥有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 中。所抽取的视频帧和光流约占据 100 GB 的存储空间。

可以运行以下命令为 SSD 建立软链接。

```
## 执行这两行进行抽取（假设 SSD 挂载在 "/mnt/SSD/"）
mkdir /mnt/SSD/ucf101_extracted/
ln -s /mnt/SSD/ucf101_extracted/ ../../../data/ucf101/rawframes
```

如果用户需要抽取 RGB 帧（因为抽取光流的过程十分耗时），可以考虑运行以下命令使用 denseflow **只抽取 RGB 帧**。

```
bash extract_rgb_frames.sh
```

如果用户没有安装 denseflow，则可以运行以下命令使用 OpenCV 抽取 RGB 帧。然而，该方法只能抽取与原始视频分辨率相同的帧。

```
bash extract_rgb_frames_opencv.sh
```

如果用户想抽取 RGB 帧和光流，则可以运行以下脚本使用“tvl1”算法进行抽取。

```
bash extract_frames.sh
```

7.17.5 步骤 4. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
bash generate_videos_filelist.sh
bash generate_rawframes_filelist.sh
```

7.17.6 步骤 5. 检查文件夹结构

在完成所有 UCF-101 数据集准备流程后，用户可以获得对应的 RGB + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，UCF-101 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── ucf101
|   |   ├── ucf101_{train,val}_split_{1,2,3}_rawframes.txt
|   |   ├── ucf101_{train,val}_split_{1,2,3}_videos.txt
|   |   ├── annotations
|   |   ├── videos
|   |   |   ├── ApplyEyeMakeup
|   |   |   |   ├── v_ApplyEyeMakeup_g01_c01.avi
```

(下页继续)

(续上页)

```
|   |   |   ├── YoYo
|   |   |   |   ├── v_YoYo_g25_c05.avi
|   |   ├── rawframes
|   |   |   ├── ApplyEyeMakeup
|   |   |   |   ├── v_ApplyEyeMakeup_g01_c01
|   |   |   |   |   ├── img_00001.jpg
|   |   |   |   |   ├── img_00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── flow_x_00001.jpg
|   |   |   |   |   ├── flow_x_00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── flow_y_00001.jpg
|   |   |   |   |   ├── flow_y_00002.jpg
|   |   |   ├── ...
|   |   |   ├── YoYo
|   |   |   |   ├── v_YoYo_g01_c01
|   |   |   |   ├── ...
|   |   |   |   ├── v_YoYo_g25_c05
```

关于对 UCF-101 进行训练和验证，可以参考 基础教程。

7.18 UCF101-24

7.18.1 简介

```
@article{Soomro2012UCF101AD,
  title={UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild},
  author={K. Soomro and A. Zamir and M. Shah},
  journal={ArXiv},
  year={2012},
  volume={abs/1212.0402}
}
```

用户可参考该数据集的 官网，以获取数据集相关的基本信息。在数据集准备前，请确保命令行当前路径为 `$MMACTION2/tools/data/ucf101_24/`。

7.18.2 下载和解压

用户可以从 这里 下载 RGB 帧，光流和标注文件。该数据由 MOC 代码库提供，参考自 act-detector 和 corrected-UCF101-Annots。

注意：UCF101-24 的标注文件来自于 这里，该标注文件相对于其他标注文件更加准确。

用户在下载 `UCF101_v2.tar.gz` 文件后，需将其放置在 `$MMACTION2/tools/data/ucf101_24/` 目录下，并使用以下指令进行解压：

```
tar -zxvf UCF101_v2.tar.gz
```

7.18.3 检查文件夹结构

经过解压后，用户将得到 `rgb-images` 文件夹，`brox-images` 文件夹和 `UCF101v2-GT.pkl` 文件。

在整个 MMAction2 文件夹下，UCF101_24 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── ucf101_24
|   |   ├── brox-images
|   |   |   ├── Basketball
|   |   |   |   ├── v_Basketball_g01_c01
|   |   |   |   |   ├── 00001.jpg
|   |   |   |   |   ├── 00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── 00140.jpg
|   |   |   |   |   ├── 00141.jpg
|   |   |   ├── ...
|   |   |   ├── WalkingWithDog
|   |   |   |   ├── v_WalkingWithDog_g01_c01
|   |   |   |   ├── ...
|   |   |   |   ├── v_WalkingWithDog_g25_c04
|   |   ├── rgb-images
|   |   |   ├── Basketball
|   |   |   |   ├── v_Basketball_g01_c01
|   |   |   |   |   ├── 00001.jpg
|   |   |   |   |   ├── 00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── 00140.jpg
|   |   |   |   |   ├── 00141.jpg
```

(下页继续)

(续上页)

```
|   |   |   ├── ...
|   |   |   ├── WalkingWithDog
|   |   |   |   ├── v_WalkingWithDog_g01_c01
|   |   |   |   ├── ...
|   |   |   |   ├── v_WalkingWithDog_g25_c04
|   |   ├── UCF101v2-GT.pkl
```

注意：`UCF101v2-GT.pkl` 作为一个缓存文件，它包含 6 个项目：

1. `labels` (list)：24 个行为类别名称组成的列表
2. `gttubes` (dict)：每个视频对应的基准 tubes 组成的字典 **gttube** 是由标签索引和 tube 列表组成的字典 **tube** 是一个 `nframes` 行和 5 列的 numpy array，每一列的形式如 `<frame index> <x1> <y1> <x2> <y2>`
3. `nframes` (dict)：用以表示每个视频对应的帧数，如 `'HorseRiding/v_HorseRiding_g05_c02': 151`
4. `train_videos` (list)：包含 `nsplits=1` 的元素，每一项都包含了训练视频的列表
5. `test_videos` (list)：包含 `nsplits=1` 的元素，每一项都包含了测试视频的列表
6. `resolution` (dict)：每个视频对应的分辨率（形如 (h,w)），如 `'FloorGymnastics/v_FloorGymnastics_g09_c03': (240, 320)`

7.19 ActivityNet

7.19.1 简介

```
@article{Heilbron2015ActivityNetAL,
  title={ActivityNet: A large-scale video benchmark for human activity understanding},
  author={Fabian Caba Heilbron and Victor Escorcia and Bernard Ghanem and Juan Carlos
↪Niebles},
  journal={2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)},
  year={2015},
  pages={961-970}
}
```

用户可参考该数据集的 官网，以获取数据集相关的基本信息。对于时序动作检测任务，用户可以使用这个 代码库 提供的缩放过（rescaled）的 ActivityNet 特征，或者使用 MMAction2 进行特征提取（这将具有更高的精度）。MMAction2 同时提供了以上所述的两种数据使用流程。在数据集准备前，请确保命令行当前路径为 `$MMACTION2/tools/data/activitynet/`。

7.19.2 选项 1：用户可以使用这个代码库提供的特征

步骤 1. 下载标注文件

首先，用户可以使用以下命令下载标注文件。

```
bash download_feature_annotations.sh
```

步骤 2. 准备视频特征

之后，用户可以使用以下命令下载 ActivityNet 特征。

```
bash download_features.sh
```

步骤 3. 处理标注文件

之后，用户可以使用以下命令处理下载的标注文件，以便于训练和测试。该脚本会首先合并两个标注文件，然后再将其分为 `train`, `val` 和 `test` 三个部分。

```
python process_annotations.py
```

7.19.3 选项 2：使用 MMAction2 对官网提供的视频进行特征抽取

步骤 1. 下载标注文件

首先，用户可以使用以下命令下载标注文件。

```
bash download_annotations.sh
```

步骤 2. 准备视频

之后，用户可以使用以下脚本准备视频数据。该代码参考自 官方爬虫，该过程将会耗费较多时间。

```
bash download_videos.sh
```

由于 ActivityNet 数据集中的一些视频已经在 YouTube 失效，官网 在谷歌网盘和百度网盘提供了完整的数据集数据。如果用户想要获取失效的数据集，则需要填写 下载页面 中提供的 需求表格 以获取 7 天的下载权限。

MMAction2 同时也提供了 BSN 代码库 的标注文件的下载步骤。

```
bash download_bsn_videos.sh
```

对于这种情况，该下载脚本将在下载后更新此标注文件，以确保每个视频都存在。

步骤 3. 抽取 RGB 帧和光流

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

可使用以下命令抽取视频帧和光流。

```
bash extract_frames.sh
```

以上脚本将会生成短边 256 分辨率的视频。如果用户想生成短边 320 分辨率的视频（即 320p），或者 340x256 的固定分辨率，用户可以通过改变参数由 `--new-short 256` 至 `--new-short 320`，或者 `--new-width 340 --new-height 256` 进行设置更多细节可参考 数据准备指南

步骤 4. 生成用于 ActivityNet 微调的文件列表

根据抽取的帧，用户可以生成视频级别（video-level）或者片段级别（clip-level）的文件列表，其可用于微调 ActivityNet。

```
python generate_rawframes_filelist.py
```

步骤 5. 在 ActivityNet 上微调 TSN 模型

用户可使用 `configs/recognition/tsn` 目录中的 ActivityNet 配置文件进行 TSN 模型微调。用户需要使用 Kinetics 相关模型（同时支持 RGB 模型与光流模型）进行预训练。

步骤 6. 使用预训练模型进行 ActivityNet 特征抽取

在 ActivityNet 上微调 TSN 模型之后，用户可以使用该模型进行 RGB 特征和光流特征的提取。

```
python tsn_feature_extraction.py --data-prefix ../../../data/ActivityNet/rawframes --
↪data-list ../../../data/ActivityNet/anet_train_video.txt --output-prefix ../../../
↪data/ActivityNet/rgb_feat --modality RGB --ckpt /path/to/rgb_checkpoint.pth

python tsn_feature_extraction.py --data-prefix ../../../data/ActivityNet/rawframes --
↪data-list ../../../data/ActivityNet/anet_val_video.txt --output-prefix ../../../
↪data/ActivityNet/rgb_feat --modality RGB --ckpt /path/to/rgb_checkpoint.pth

python tsn_feature_extraction.py --data-prefix ../../../data/ActivityNet/rawframes --
↪data-list ../../../data/ActivityNet/anet_train_video.txt --output-prefix ../../../
↪data/ActivityNet/flow_feat --modality Flow --ckpt /path/to/flow_checkpoint.pth

python tsn_feature_extraction.py --data-prefix ../../../data/ActivityNet/rawframes --
↪data-list ../../../data/ActivityNet/anet_val_video.txt --output-prefix ../../../
↪data/ActivityNet/flow_feat --modality Flow --ckpt /path/to/flow_checkpoint.pth
```

(下页继续)

(续上页)

在提取完特征后，用户可以使用后处理脚本整合 RGB 特征和光流特征，生成 100-t X 400-d 维度的特征用于时序动作检测。

```
python activitynet_feature_postprocessing.py --rgb ../../../data/ActivityNet/rgb_feat␣
↪--flow ../../../data/ActivityNet/flow_feat --dest ../../../data/ActivityNet/
↪mmaction_feat
```

7.19.4 最后一步：检查文件夹结构

在完成所有 ActivityNet 数据集准备流程后，用户可以获得对应的特征文件，RGB + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，ActivityNet 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
│   ├── ActivityNet

(若根据选项 1 进行数据处理)
│   │   ├── anet_anno_{train,val,test,full}.json
│   │   ├── anet_anno_action.json
│   │   ├── video_info_new.csv
│   │   ├── activitynet_feature_cuhk
│   │   │   ├── csv_mean_100
│   │   │   │   ├── v___c8enCfzqw.csv
│   │   │   │   ├── v___dXUJsj3yo.csv
│   │   │   │   ├── ..

(若根据选项 2 进行数据处理)
│   │   ├── anet_train_video.txt
│   │   ├── anet_val_video.txt
│   │   ├── anet_train_clip.txt
│   │   ├── anet_val_clip.txt
│   │   ├── activity_net.v1-3.min.json
│   │   ├── mmaction_feat
│   │   │   ├── v___c8enCfzqw.csv
│   │   │   ├── v___dXUJsj3yo.csv
│   │   │   ├── ..
```

(下页继续)

(续上页)

```
|   |   ├── rawframes
|   |   |   ├── v___c8enCfzqw
|   |   |   |   ├── img_00000.jpg
|   |   |   |   ├── flow_x_00000.jpg
|   |   |   |   ├── flow_y_00000.jpg
|   |   |   |   ├── ..
|   |   |   ├── ..
```

关于对 ActivityNet 进行训练和验证，可以参考 基础教程.

7.20 AVA

7.20.1 简介

```
@inproceedings{gu2018ava,
  title={Ava: A video dataset of spatio-temporally localized atomic visual actions},
  author={Gu, Chunhui and Sun, Chen and Ross, David A and Vondrick, Carl and␣
↪Pantofaru, Caroline and Li, Yeqing and Vijayanarasimhan, Sudheendra and Toderici,␣
↪George and Ricco, Susanna and Sukthankar, Rahul and others},
  booktitle={Proceedings of the IEEE Conference on Computer Vision and Pattern␣
↪Recognition},
  pages={6047--6056},
  year={2018}
}
```

请参照 官方网站 以获取数据集基本信息。在开始之前，用户需确保当前目录为 `$MMACTION2/tools/data/ava/`。

7.20.2 1. 准备标注文件

首先，用户可以使用如下脚本下载标注文件并进行预处理：

```
bash download_annotations.sh
```

这一命令将下载 `ava_v2.1.zip` 以得到 AVA v2.1 标注文件。如用户需要 AVA v2.2 标注文件，可使用以下脚本：

```
VERSION=2.2 bash download_annotations.sh
```

7.20.3 2. 下载视频

用户可以使用以下脚本准备视频，视频准备代码修改自 官方爬虫。注意这一步骤将花费较长时间。

```
bash download_videos.sh
```

亦可使用以下脚本，使用 python 并行下载 AVA 数据集视频：

```
bash download_videos_parallel.sh
```

7.20.4 3. 截取视频

截取每个视频中的 15 到 30 分钟，设定帧率为 30。

```
bash cut_videos.sh
```

7.20.5 4. 提取 RGB 帧和光流

在提取之前，请参考 安装教程 安装 denseflow。

如果用户有足够的 SSD 空间，那么建议将视频抽取为 RGB 帧以提升 I/O 性能。用户可以使用以下脚本为抽取得到的帧文件夹建立软连接：

```
## 执行以下脚本 (假设 SSD 被挂载在 "/mnt/SSD/")
mkdir /mnt/SSD/ava_extracted/
ln -s /mnt/SSD/ava_extracted/ ../data/ava/rawframes/
```

如果用户只使用 RGB 帧（由于光流提取非常耗时），可执行以下脚本使用 denseflow 提取 RGB 帧：

```
bash extract_rgb_frames.sh
```

如果用户未安装 denseflow，可执行以下脚本使用 ffmpeg 提取 RGB 帧：

```
bash extract_rgb_frames_ffmpeg.sh
```

如果同时需要 RGB 帧和光流，可使用如下脚本抽帧：

```
bash extract_frames.sh
```

7.20.6 5. 下载 AVA 上人体检测结果

以下脚本修改自 Long-Term Feature Banks。

可使用以下脚本下载 AVA 上预先计算的人体检测结果：

```
bash fetch_ava_proposals.sh
```

7.20.7 6. 目录结构

在完整完成 AVA 的数据处理后，将得到帧文件夹（RGB 帧和光流帧），视频以及标注文件。

在整个项目目录下（仅针对 AVA），最简目录结构如下所示：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── ava
|   |   ├── annotations
|   |   |   ├── ava_dense_proposals_train.FAIR.recall_93.9.pkl
|   |   |   ├── ava_dense_proposals_val.FAIR.recall_93.9.pkl
|   |   |   ├── ava_dense_proposals_test.FAIR.recall_93.9.pkl
|   |   |   ├── ava_train_v2.1.csv
|   |   |   ├── ava_val_v2.1.csv
|   |   |   ├── ava_train_excluded_timestamps_v2.1.csv
|   |   |   ├── ava_val_excluded_timestamps_v2.1.csv
|   |   |   ├── ava_action_list_v2.1_for_activitynet_2018.pbtxt
|   |   ├── videos
|   |   |   ├── 053oq2xB3oU.mkv
|   |   |   ├── 0f39OWEqJ24.mp4
|   |   |   ├── ...
|   |   ├── videos_15min
|   |   |   ├── 053oq2xB3oU.mkv
|   |   |   ├── 0f39OWEqJ24.mp4
|   |   |   ├── ...
|   |   ├── rawframes
|   |   |   ├── 053oq2xB3oU
|   |   |   |   ├── img_00001.jpg
|   |   |   |   ├── img_00002.jpg
|   |   |   |   ├── ...
```

关于 AVA 数据集上的训练与测试，请参照 基础教程。

7.21 Diving48

7.21.1 简介

```
@inproceedings{li2018resound,
  title={Resound: Towards action recognition without representation bias},
  author={Li, Yingwei and Li, Yi and Vasconcelos, Nuno},
  booktitle={Proceedings of the European Conference on Computer Vision (ECCV)},
  pages={513--528},
  year={2018}
}
```

用户可参考该数据集的 官网，以获取数据集相关的基本信息。在数据集准备前，请确保命令行当前路径为 `$MMACTION2/tools/data/diving48/`。

7.21.2 步骤 1. 下载标注文件

用户可以使用以下命令下载标注文件（考虑到标注的准确性，这里仅下载 V2 版本）。

```
bash download_annotations.sh
```

7.21.3 步骤 2. 准备视频

用户可以使用以下命令下载视频。

```
bash download_videos.sh
```

7.21.4 Step 3. 抽取 RGB 帧和光流

如果用户只想使用视频加载训练，则该部分是 **可选项**。

官网提供的帧压缩包并不完整。若想获取完整的数据，可以使用以下步骤解帧。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果拥有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 中。

可以运行以下命令为 SSD 建立软链接。

```
## 执行这两行进行抽取 (假设 SSD 挂载在 "/mnt/SSD/")
mkdir /mnt/SSD/diving48_extracted/
ln -s /mnt/SSD/diving48_extracted/ ../../../data/diving48/rawframes
```

如果用户需要抽取 RGB 帧（因为抽取光流的过程十分耗时），可以考虑运行以下命令使用 denseflow **只抽取 RGB 帧**。

```
cd $MMACTION2/tools/data/diving48/
bash extract_rgb_frames.sh
```

如果用户没有安装 denseflow，则可以运行以下命令使用 OpenCV 抽取 RGB 帧。然而，该方法只能抽取与原始视频分辨率相同的帧。

```
cd $MMACTION2/tools/data/diving48/
bash extract_rgb_frames_opencv.sh
```

如果用户想抽取 RGB 帧和光流，则可以运行以下脚本进行抽取。

```
cd $MMACTION2/tools/data/diving48/
bash extract_frames.sh
```

7.21.5 步骤 4. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
bash generate_videos_filelist.sh
bash generate_rawframes_filelist.sh
```

7.21.6 步骤 5. 检查文件夹结构

在完成所有 Diving48 数据集准备流程后，用户可以获得对应的 RGB + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，Diving48 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── diving48
|   |   ├── diving48_{train,val}_list_rawframes.txt
|   |   ├── diving48_{train,val}_list_videos.txt
|   |   ├── annotations
|   |   |   ├── Diving48_V2_train.json
|   |   |   ├── Diving48_V2_test.json
|   |   |   ├── Diving48_vocab.json
|   |   ├── videos
```

(下页继续)

(续上页)

```
|   |   |   ├── _8Vy3dlHg2w_00000.mp4
|   |   |   ├── _8Vy3dlHg2w_00001.mp4
|   |   |   ├── ...
|   |   ├── rawframes
|   |   |   ├── 2x00lRzlTVQ_00000
|   |   |   |   ├── img_00001.jpg
|   |   |   |   ├── img_00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_x_00001.jpg
|   |   |   |   ├── flow_x_00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_y_00001.jpg
|   |   |   |   ├── flow_y_00002.jpg
|   |   |   |   ├── ...
|   |   |   ├── 2x00lRzlTVQ_00001
|   |   |   ├── ...
```

关于对 Diving48 进行训练和验证，可以参考 基础教程。

7.22 GYM

7.22.1 简介

```
@inproceedings{shao2020finegym,
  title={Finegym: A hierarchical video dataset for fine-grained action understanding},
  author={Shao, Dian and Zhao, Yue and Dai, Bo and Lin, Dahua},
  booktitle={Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern␣
↪Recognition},
  pages={2616--2625},
  year={2020}
}
```

请参照 项目主页 及 原论文 以获取数据集基本信息。MMAction2 当前支持 GYM99 的数据集预处理。在开始之前，用户需确保当前目录为 `$MMACTION2/tools/data/gym/`。

7.22.2 1. 准备标注文件

首先，用户可以使用如下脚本下载标注文件并进行预处理：

```
bash download_annotations.sh
```

7.22.3 2. 准备视频

用户可以使用以下脚本准备视频，视频准备代码修改自 ActivityNet 爬虫。注意这一步骤将花费较长时间。

```
bash download_videos.sh
```

7.22.4 3. 裁剪长视频至动作级别

用户首先需要使用以下脚本将 GYM 中的长视频依据标注文件裁剪至动作级别。

```
python trim_event.py
```

7.22.5 4. 裁剪动作视频至分动作级别

随后，用户需要使用以下脚本将 GYM 中的动作视频依据标注文件裁剪至分动作级别。将视频的裁剪分成两个级别可以带来更高的效率（在长视频中裁剪多个极短片段异常耗时）。

```
python trim_subaction.py
```

7.22.6 5. 提取 RGB 帧和光流

如果用户仅使用 video loader，则可以跳过本步。

在提取之前，请参考 安装教程 安装 denseflow。

用户可使用如下脚本同时抽取 RGB 帧和光流（提取光流时使用 tvl1 算法）：

```
bash extract_frames.sh
```

7.22.7 6. 基于提取出的分动作生成文件列表

用户可使用以下脚本为 GYM99 生成训练及测试的文件列表：

```
python generate_file_list.py
```

7.22.8 7. 目录结构

在完整完成 GYM 的数据处理后，将得到帧文件夹（RGB 帧和光流帧），动作视频片段，分动作视频片段以及训练测试所用标注文件。

在整个项目目录下（仅针对 GYM），完整目录结构如下所示：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── gym
|   |   ├── annotations
|   |   |   ├── gym99_train_org.txt
|   |   |   ├── gym99_val_org.txt
|   |   |   ├── gym99_train.txt
|   |   |   ├── gym99_val.txt
|   |   |   ├── annotation.json
|   |   |   └── event_annotation.json
|   |   ├── videos
|   |   |   ├── 0LtLS9wROrk.mp4
|   |   |   ├── ...
|   |   |   └── zfqS-wCJSsw.mp4
|   |   ├── events
|   |   |   ├── 0LtLS9wROrk_E_002407_002435.mp4
|   |   |   ├── ...
|   |   |   └── zfqS-wCJSsw_E_006732_006824.mp4
|   |   ├── subactions
|   |   |   ├── 0LtLS9wROrk_E_002407_002435_A_0003_0005.mp4
|   |   |   ├── ...
|   |   |   └── zfqS-wCJSsw_E_006244_006252_A_0000_0007.mp4
|   |   └── subaction_frames
```

关于 GYM 数据集上的训练与测试，请参照 基础教程。

7.23 HMDB51

7.23.1 简介

```
@article{Kuehne2011HMDBAL,
  title={HMDB: A large video database for human motion recognition},
  author={Hilde Kuehne and Hueihan Jhuang and E. Garrote and T. Poggio and Thomas
↪Serre},
  journal={2011 International Conference on Computer Vision},
  year={2011},
  pages={2556-2563}
}
```

用户可以参照数据集 官网，获取数据集相关的基本信息。在准备数据集前，请确保命令行当前路径为 `$MMACTION2/tools/data/hmdb51/`。

为运行下面的 bash 脚本，需要安装 `unrar`。用户可运行 `sudo apt-get install unrar` 安装，或参照 setup，运行 `zzunrar.sh` 脚本实现无管理员权限下的简易安装。

7.23.2 步骤 1. 下载标注文件

首先，用户可使用以下命令下载标注文件。

```
bash download_annotations.sh
```

7.23.3 步骤 2. 下载视频

之后，用户可使用以下指令下载视频

```
bash download_videos.sh
```

7.23.4 步骤 3. 抽取帧和光流

如果用户只想使用视频加载训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果用户有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 上。用户可使用以下命令为 SSD 建立软链接。

```
## 执行这两行指令进行抽取（假设 SSD 挂载在 "/mnt/SSD/" 上）
mkdir /mnt/SSD/hmdb51_extracted/
ln -s /mnt/SSD/hmdb51_extracted/ ../../../data/hmdb51/rawframes
```

如果用户需要抽取 RGB 帧（因为抽取光流的过程十分耗时），可以考虑运行以下命令使用 denseflow **只抽取 RGB 帧**。

```
bash extract_rgb_frames.sh
```

如果用户没有安装 denseflow，则可以运行以下命令使用 OpenCV 抽取 RGB 帧。然而，该方法只能抽取与原始视频分辨率相同的帧。

```
bash extract_rgb_frames_opencv.sh
```

如果用户想抽取 RGB 帧和光流，则可以运行以下脚本，使用“tvl1”算法进行抽取。

```
bash extract_frames.sh
```

7.23.5 步骤 4. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
bash generate_rawframes_filelist.sh
bash generate_videos_filelist.sh
```

7.23.6 步骤 5. 检查目录结构

在完成 HMDB51 数据集准备流程后，用户可以得到 HMDB51 的 RGB 帧 + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，HMDB51 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
│   ├── hmdb51
│   │   ├── hmdb51_{train,val}_split_{1,2,3}_rawframes.txt
│   │   ├── hmdb51_{train,val}_split_{1,2,3}_videos.txt
│   │   ├── annotations
│   │   ├── videos
│   │   │   ├── brush_hair
```

(下页继续)

(续上页)

```
|   |   |   |   ├── April_09_brush_hair_u_nm_np1_ba_goo_0.avi

|   |   |   ├── wave
|   |   |   |   ├── 20060723sfjffbartsinger_wave_f_cm_np1_ba_med_0.avi
|   |   ├── rawframes
|   |   |   ├── brush_hair
|   |   |   |   ├── April_09_brush_hair_u_nm_np1_ba_goo_0
|   |   |   |   |   ├── img_00001.jpg
|   |   |   |   |   ├── img_00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── flow_x_00001.jpg
|   |   |   |   |   ├── flow_x_00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── flow_y_00001.jpg
|   |   |   |   |   ├── flow_y_00002.jpg
|   |   |   ├── ...
|   |   |   ├── wave
|   |   |   |   ├── 20060723sfjffbartsinger_wave_f_cm_np1_ba_med_0
|   |   |   |   ├── ...
|   |   |   |   ├── winKen_wave_u_cm_np1_ri_bad_1
```

关于对 HMDB51 进行训练和验证，可以参照 基础教程。

7.24 HVU

7.24.1 简介

```
@article{Diba2019LargeSH,
  title={Large Scale Holistic Video Understanding},
  author={Ali Diba and M. Fayyaz and Vivek Sharma and Manohar Paluri and Jurgen Gall␣
↪and R. Stiefelhagen and L. Gool},
  journal={arXiv: Computer Vision and Pattern Recognition},
  year={2019}
}
```

请参照 官方项目 及 原论文 以获取数据集基本信息。在开始之前，用户需确保当前目录为 `$MMACTION2/tools/data/hvu/`。

7.24.2 1. 准备标注文件

首先，用户可以使用如下脚本下载标注文件并进行预处理：

```
bash download_annotations.sh
```

此外，用户可使用如下命令解析 HVU 的标签列表：

```
python parse_tag_list.py
```

7.24.3 2. 准备视频

用户可以使用以下脚本准备视频，视频准备代码修改自 ActivityNet 爬虫。注意这一步骤将花费较长时间。

```
bash download_videos.sh
```

7.24.4 3. 提取 RGB 帧和光流

如果用户仅使用 video loader，则可以跳过本步。

在提取之前，请参考 安装教程 安装 denseflow。

用户可使用如下脚本同时抽取 RGB 帧和光流：

```
bash extract_frames.sh
```

该脚本默认生成短边长度为 256 的帧，可参考 数据准备 获得更多细节。

7.24.5 4. 生成文件列表

用户可以使用以下两个脚本分别为视频和帧文件夹生成文件列表：

```
bash generate_videos_filelist.sh
## 为帧文件夹生成文件列表
bash generate_rawframes_filelist.sh
```

7.24.6 5. 为每个 tag 种类生成文件列表

若用户需要为 HVU 数据集的每个 tag 种类训练识别模型，则需要进行此步骤。

步骤 4 中生成的文件列表包含不同类型的标签，仅支持使用 HVUDataset 进行涉及多个标签种类的多任务学习。加载数据的过程中需要使用 `LoadHVULabel` 类进行多类别标签的加载，训练过程中使用 `HVULoss` 作为损失函数。

如果用户仅需训练某一特定类别的标签，例如训练一识别模型用于识别 HVU 中 `action` 类别的标签，则建议使用如下脚本为特定标签种类生成文件列表。新生成的列表将只含有特定类别的标签，因此可使用 `VideoDataset` 或 `RawframeDataset` 进行加载。训训练过程中使用 `BCELossWithLogits` 作为损失函数。

以下脚本为类别为 ${category} 的标签生成文件列表，注意仅支持 HVU 数据集包含的 6 种标签类别: action, attribute, concept, event, object, scene。

```
python generate_sub_file_list.py path/to/filelist.json ${category}
```

对于类别 ${category}，生成的标签列表文件名中将使用 `hvu_${category}` 替代 `hvu`。例如，若原指定文件名为 `hvu_train.json`，则对于类别 action，生成的文件列表名为 `hvu_action_train.json`。

7.24.7 6. 目录结构

在完整完成 HVU 的数据处理后，将得到帧文件夹（RGB 帧和光流帧），视频以及标注文件。

在整个项目目录下（仅针对 HVU），完整目录结构如下所示：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
│   ├── hvu
│   │   ├── hvu_train_video.json
│   │   ├── hvu_val_video.json
│   │   ├── hvu_train.json
│   │   ├── hvu_val.json
│   │   ├── annotations
│   │   ├── videos_train
│   │   │   ├── OLpWTpTC4P8_000570_000670.mp4
│   │   │   ├── xsPKW4tZZBc_002330_002430.mp4
│   │   │   ├── ...
│   │   ├── videos_val
│   │   ├── rawframes_train
│   │   ├── rawframes_val
```

关于 HVU 数据集上的训练与测试，请参照 基础教程。

7.25 Jester

7.25.1 简介

```
@InProceedings{Materzynska_2019_ICCV,
  author = {Materzynska, Joanna and Berger, Guillaume and Bax, Ingo and Memisevic,␣
↪Roland},
  title = {The Jester Dataset: A Large-Scale Video Dataset of Human Gestures},
  booktitle = {Proceedings of the IEEE/CVF International Conference on Computer␣
↪Vision (ICCV) Workshops},
  month = {Oct},
  year = {2019}
}
```

用户可以参照数据集 官网，获取数据集相关的基本信息。在准备数据集前，请确保命令行当前路径为 `$MMACTION2/tools/data/jester/`。

7.25.2 步骤 1. 下载标注文件

首先，用户需要在 官网 完成注册，才能下载标注文件。下载好的标注文件需要放在 `$MMACTION2/data/jester/annotations` 文件夹下。

7.25.3 步骤 2. 准备 RGB 帧

jester 官网 并未提供原始视频文件，只提供了对原视频文件进行抽取得到的 RGB 帧，用户可在 jester 官网 直接下载。

将下载好的压缩文件放在 `$MMACTION2/data/jester/` 文件夹下，并使用以下脚本进行解压。

```
cd $MMACTION2/data/jester/
cat 20bn-jester-v1-?? | tar zx
cd $MMACTION2/tools/data/jester/
```

如果用户只想使用 RGB 帧，则可以跳过中间步骤至步骤 5 以直接生成视频帧的文件列表。由于官网的 JPG 文件名形如 “%05d.jpg” (比如，” 00001.jpg”)，需要在配置文件的 `data.train`, `data.val` 和 `data.test` 处添加 `"filename_tmpl='{:05}.jpg'"` 代码，以修改文件名模板。

```
data = dict(
    videos_per_gpu=16,
```

(下页继续)

(续上页)

```
    workers_per_gpu=2,
    train=dict(
        type=dataset_type,
        ann_file=ann_file_train,
        data_prefix=data_root,
        filename_tmpl='{:05}.jpg',
        pipeline=train_pipeline),
    val=dict(
        type=dataset_type,
        ann_file=ann_file_val,
        data_prefix=data_root_val,
        filename_tmpl='{:05}.jpg',
        pipeline=val_pipeline),
    test=dict(
        type=dataset_type,
        ann_file=ann_file_test,
        data_prefix=data_root_val,
        filename_tmpl='{:05}.jpg',
        pipeline=test_pipeline))
```

7.25.4 步骤 3. 抽取光流

如果用户只想使用 RGB 帧训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果拥有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 中。

可以运行以下命令为 SSD 建立软链接。

```
## 执行这两行进行抽取 (假设 SSD 挂载在 "/mnt/SSD/")
mkdir /mnt/SSD/jester_extracted/
ln -s /mnt/SSD/jester_extracted/ ../../../data/jester/rawframes
```

如果想抽取光流，则可以运行以下脚本从 RGB 帧中抽取出光流。

```
cd $MMACTION2/tools/data/jester/
bash extract_flow.sh
```

7.25.5 步骤 4: 编码视频

如果用户只想使用 RGB 帧训练，则该部分是 **可选项**。

用户可以运行以下命令进行视频编码。

```
cd $MMACTION2/tools/data/jester/
bash encode_videos.sh
```

7.25.6 步骤 5. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
cd $MMACTION2/tools/data/jester/
bash generate_{rawframes, videos}_filelist.sh
```

7.25.7 步骤 6. 检查文件夹结构

在完成所有 Jester 数据集准备流程后，用户可以获得对应的 RGB + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，Jester 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── jester
|   |   ├── jester_{train,val}_list_rawframes.txt
|   |   ├── jester_{train,val}_list_videos.txt
|   |   ├── annotations
|   |   ├── videos
|   |   |   ├── 1.mp4
|   |   |   ├── 2.mp4
|   |   |   ├──...
|   |   ├── rawframes
|   |   |   ├── 1
|   |   |   |   ├── 00001.jpg
|   |   |   |   ├── 00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_x_00001.jpg
|   |   |   |   ├── flow_x_00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_y_00001.jpg
```

(下页继续)

(续上页)

```
|   |   |   |   ├── flow_y_00002.jpg
|   |   |   |   ├── ...
|   |   |   ├── 2
|   |   |   ├── ...
```

关于对 jester 进行训练和验证，可以参考 基础教程。

7.26 JHMDB

7.26.1 简介

```
@inproceedings{Jhuang:ICCV:2013,
    title = {Towards understanding action recognition},
    author = {H. Jhuang and J. Gall and S. Zuffi and C. Schmid and M. J. Black},
    booktitle = {International Conf. on Computer Vision (ICCV)},
    month = Dec,
    pages = {3192-3199},
    year = {2013}
}
```

用户可参考该数据集的 官网，以获取数据集相关的基本信息。在数据集准备前，请确保命令行当前路径为 `$MMACTION2/tools/data/jhmdb/`。

7.26.2 下载和解压

用户可以从 这里 下载 RGB 帧，光流和真实标签文件。该数据由 MOC 代码库提供，参考自 act-detector。

用户在下载 `JHMDB.tar.gz` 文件后，需将其放置在 `$MMACTION2/tools/data/jhmdb/` 目录下，并使用以下指令进行解压：

```
tar -zxvf JHMDB.tar.gz
```

如果拥有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 中。

可以运行以下命令为 SSD 建立软链接。

```
## 执行这两行进行抽取 (假设 SSD 挂载在 "/mnt/SSD/")
mkdir /mnt/SSD/JHMDB/
ln -s /mnt/SSD/JHMDB/ ../../../data/jhmdb
```

7.26.3 检查文件夹结构

完成解压后，用户将得到 FlowBrox04 文件夹，Frames 文件夹和 JHMDB-GT.pkl 文件。

在整个 MMAction2 文件夹下，JHMDB 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── jhmdb
|   |   ├── FlowBrox04
|   |   |   ├── brush_hair
|   |   |   |   ├── April_09_brush_hair_u_nm_np1_ba_goo_0
|   |   |   |   |   ├── 00001.jpg
|   |   |   |   |   ├── 00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── 00039.jpg
|   |   |   |   |   ├── 00040.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── Trannydude___Brushing_SyntheticHair___OhNOES!__those_fukin_knots!_
↪brush_hair_u_nm_np1_fr_goo_2
|   |   |   ├── ...
|   |   |   ├── wave
|   |   |   |   ├── 21_wave_u_nm_np1_fr_goo_5
|   |   |   |   ├── ...
|   |   |   |   ├── Wie_man_winkt!!_wave_u_cm_np1_fr_med_0
|   |   ├── Frames
|   |   |   ├── brush_hair
|   |   |   |   ├── April_09_brush_hair_u_nm_np1_ba_goo_0
|   |   |   |   |   ├── 00001.png
|   |   |   |   |   ├── 00002.png
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── 00039.png
|   |   |   |   |   ├── 00040.png
|   |   |   |   ├── ...
|   |   |   |   ├── Trannydude___Brushing_SyntheticHair___OhNOES!__those_fukin_knots!_
↪brush_hair_u_nm_np1_fr_goo_2
|   |   |   ├── ...
|   |   |   ├── wave
|   |   |   |   ├── 21_wave_u_nm_np1_fr_goo_5
|   |   |   |   ├── ...
|   |   |   |   ├── Wie_man_winkt!!_wave_u_cm_np1_fr_med_0
|   |   ├── JHMDB-GT.pkl
```

(下页继续)

(续上页)

```
```

注意：`JHMDB-GT.pkl` 作为一个缓存文件，它包含 6 个项目：

1. `labels` (list)：21 个行为类别名称组成的列表
2. `gttubes` (dict)：每个视频对应的基准 tubes 组成的字典 **gttube** 是由标签索引和 tube 列表组成的字典 **tube** 是一个 `nframes` 行和 5 列的 numpy array，每一列的形式如 `<frame index> <x1> <y1> <x2> <y2>`
3. `nframes` (dict)：用以表示每个视频对应的帧数，如 `'walk/Panic_in_the_Streets_walk_u_cm_np1_ba_med_5': 16`
4. `train_videos` (list)：包含 `nsplits=1` 的元素，每一项都包含了训练视频的列表
5. `test_videos` (list)：包含 `nsplits=1` 的元素，每一项都包含了测试视频的列表
6. `resolution` (dict)：每个视频对应的分辨率（形如 (h,w)），如 `'pour/Bartender_School_Students_Practice_pour_u_cm_np1_fr_med_1': (240, 320)`

7.27 Kinetics-[400/600/700]

7.27.1 简介

```
@inproceedings{inproceedings,
  author = {Carreira, J. and Zisserman, Andrew},
  year = {2017},
  month = {07},
  pages = {4724-4733},
  title = {Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset},
  doi = {10.1109/CVPR.2017.502}
}
```

请参照 官方网站 以获取数据集基本信息。此脚本用于准备数据集 kinetics400，kinetics600，kinetics700。为准备 kinetics 数据集的不同版本，用户需将脚本中的 `${DATASET}` 赋值为数据集对应版本名称，可选项为 `kinetics400`，`kinetics600`，`kinetics700`。在开始之前，用户需确保当前目录为 `$MMACTION2/tools/data/${DATASET}/`。

注：由于部分 YouTube 链接失效，爬取的 Kinetics 数据集大小可能与原版不同。以下是我们所使用 Kinetics 数据集的大小：

7.27.2 1. 准备标注文件

首先，用户可以使用如下脚本从 Kinetics 数据集官网下载标注文件并进行预处理：

```
bash download_annotations.sh ${DATASET}
```

由于部分视频的 URL 不可用，当前官方标注中所含视频数量可能小于初始版本。所以 MMAction2 提供了另一种方式以获取初始版本标注作为参考。在这其中，Kinetics400 和 Kinetics600 的标注文件来自 官方爬虫，Kinetics700 的标注文件于 05/02/2021 下载自 网站。

```
bash download_backup_annotations.sh ${DATASET}
```

7.27.3 2. 准备视频

用户可以使用以下脚本准备视频，视频准备代码修改自 官方爬虫。注意这一步骤将花费较长时间。

```
bash download_videos.sh ${DATASET}
```

重要提示：如果在此之前已下载好 Kinetics 数据集的视频，还需使用重命名脚本来替换掉类名中的空格：

```
bash rename_classnames.sh ${DATASET}
```

为提升解码速度，用户可以使用以下脚本将原始视频缩放至更小的分辨率（利用稠密编码）：

```
python ../resize_videos.py ../../../data/${DATASET}/videos_train/ ../../../data/$
↪{DATASET}/videos_train_256p_dense_cache --dense --level 2
```

也可以从 Academic Torrents 中下载短边长度为 256 的 kinetics400 和 kinetics700，或从 Common Visual Data Foundation 维护的 cvdfoundation/kinetics-dataset 中下载 Kinetics400/Kinetics600/Kinetics-700-2020。

7.27.4 3. 提取 RGB 帧和光流

如果用户仅使用 video loader，则可以跳过本步。

在提取之前，请参考 安装教程 安装 denseflow。

如果用户有足够的 SSD 空间，那么建议将视频抽取为 RGB 帧以提升 I/O 性能。用户可以使用以下脚本为抽取得到的帧文件夹建立软连接：

```
## 执行以下脚本 (假设 SSD 被挂载在 "/mnt/SSD/")
mkdir /mnt/SSD/${DATASET}_extracted_train/
ln -s /mnt/SSD/${DATASET}_extracted_train/ ../../../data/${DATASET}/rawframes_train/
mkdir /mnt/SSD/${DATASET}_extracted_val/
ln -s /mnt/SSD/${DATASET}_extracted_val/ ../../../data/${DATASET}/rawframes_val/
```

如果用户只使用 RGB 帧（由于光流提取非常耗时），可以考虑执行以下脚本，仅用 denseflow 提取 RGB 帧：

```
bash extract_rgb_frames.sh ${DATASET}
```

如果用户未安装 denseflow，以下脚本可以使用 OpenCV 进行 RGB 帧的提取，但视频原分辨率大小会被保留：

```
bash extract_rgb_frames_opencv.sh ${DATASET}
```

如果同时需要 RGB 帧和光流，可使用如下脚本抽帧：

```
bash extract_frames.sh ${DATASET}
```

以上的命令生成短边长度为 256 的 RGB 帧和光流帧。如果用户需要生成短边长度为 320 的帧 (320p)，或是固定分辨率为 340 x 256 的帧，可改变参数 `--new-short 256` 为 `--new-short 320` 或 `--new-width 340 --new-height 256`。更多细节可以参考 数据准备。

7.27.5 4. 生成文件列表

用户可以使用以下两个脚本分别为视频和帧文件夹生成文件列表：

```
bash generate_videos_filelist.sh ${DATASET}
## 为帧文件夹生成文件列表
bash generate_rawframes_filelist.sh ${DATASET}
```

7.27.6 5. 目录结构

在完整完成 Kinetics 的数据处理后，将得到帧文件夹（RGB 帧和光流帧），视频以及标注文件。

在整个项目目录下（仅针对 Kinetics），最简目录结构如下所示：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── ${DATASET}
|   |   ├── ${DATASET}_train_list_videos.txt
|   |   ├── ${DATASET}_val_list_videos.txt
|   |   ├── annotations
|   |   ├── videos_train
|   |   ├── videos_val
|   |   |   ├── abseiling
|   |   |   |   ├── 0wR5jVB-WPk_000417_000427.mp4
|   |   |   |   ├── ...
```

(下页继续)

(续上页)

```
|   |   |   ├── ...
|   |   |   ├── wrapping_present
|   |   |   ├── ...
|   |   |   ├── zumba
|   |   ├── rawframes_train
|   |   ├── rawframes_val
```

关于 Kinetics 数据集上的训练与测试，请参照 基础教程。

7.28 Moments in Time

7.28.1 简介

```
@article{monfortmoments,
    title={Moments in Time Dataset: one million videos for event understanding},
    author={Monfort, Mathew and Andonian, Alex and Zhou, Bolei and Ramakrishnan,␣
↪Kandan and Bargal, Sarah Adel and Yan, Tom and Brown, Lisa and Fan, Quanfu and␣
↪Gutfruend, Dan and Vondrick, Carl and others},
    journal={IEEE Transactions on Pattern Analysis and Machine Intelligence},
    year={2019},
    issn={0162-8828},
    pages={1--8},
    numpages={8},
    doi={10.1109/TPAMI.2019.2901464},
}
```

用户可以参照数据集 官网，获取数据集相关的基本信息。在准备数据集前，请确保命令行当前路径为 `$MMACTION2/tools/data/mit/`。

7.28.2 步骤 1. 准备标注文件和视频文件

首先，用户需要访问官网，填写申请表来下载数据集。在得到下载链接后，用户可以使用 `bash preprocess_data.sh` 来准备标注文件和视频。请注意此脚本并没有下载标注和视频文件，用户需要根据脚本文件中的注释，提前下载好数据集，并放/软链接到合适的位置。

为加快视频解码速度，用户需要缩小原视频的尺寸，可使用以下命令获取密集编码版视频：

```
python ../resize_videos.py ../../../data/mit/videos/ ../../../data/mit/videos_256p_
↪dense_cache --dense --level 2
```

7.28.3 Step 2. 抽取帧和光流

如果用户只想使用视频加载训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果用户有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 上。用户可使用以下命令为 SSD 建立软链接。

```
## 执行这两行指令进行抽取（假设 SSD 挂载在 "/mnt/SSD/" 上）
mkdir /mnt/SSD/mit_extracted/
ln -s /mnt/SSD/mit_extracted/ ../../../data/mit/rawframes
```

如果用户需要抽取 RGB 帧（因为抽取光流的过程十分耗时），可以考虑运行以下命令使用 denseflow **只抽取 RGB 帧**。

```
bash extract_rgb_frames.sh
```

如果用户没有安装 denseflow，则可以运行以下命令使用 OpenCV 抽取 RGB 帧。然而，该方法只能抽取与原始视频分辨率相同的帧。

```
bash extract_rgb_frames_opencv.sh
```

如果用户想抽取 RGB 帧和光流，则可以运行以下脚本进行抽取。

```
bash extract_frames.sh
```

7.28.4 步骤 3. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
bash generate_{rawframes, videos}_filelist.sh
```

7.28.5 步骤 4. 检查目录结构

在完成 Moments in Time 数据集准备流程后，用户可以得到 Moments in Time 的 RGB 帧 + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，Moments in Time 的文件结构如下：

```
mmaction2
├── data
│   └── mit
│       ├── annotations
```

(下页继续)

(续上页)

```
|       |   ├── license.txt
|       |   ├── moments_categories.txt
|       |   ├── README.txt
|       |   ├── trainingSet.csv
|       |   └── validationSet.csv
|       ├── mit_train_rawframe_anno.txt
|       ├── mit_train_video_anno.txt
|       ├── mit_val_rawframe_anno.txt
|       ├── mit_val_video_anno.txt
|       ├── rawframes
|       |   ├── training
|       |   |   ├── adult+female+singing
|       |   |   |   ├── 0P3XG_vf91c_35
|       |   |   |   |   ├── flow_x_00001.jpg
|       |   |   |   |   ├── flow_x_00002.jpg
|       |   |   |   |   ├── ...
|       |   |   |   |   ├── flow_y_00001.jpg
|       |   |   |   |   ├── flow_y_00002.jpg
|       |   |   |   |   ├── ...
|       |   |   |   |   ├── img_00001.jpg
|       |   |   |   |   └── img_00002.jpg
|       |   |   |   └── yt-zxQfALnTdfc_56
|       |   |   |   |   ├── ...
|       |   |   └── yawning
|       |   |       ├── _8zmP1e-EjU_2
|       |   |       |   ├── ...
|       |   └── validation
|       |   |       ├── ...
|       └── videos
|           ├── training
|           |   ├── adult+female+singing
|           |   |   ├── 0P3XG_vf91c_35.mp4
|           |   |   ├── ...
|           |   |   └── yt-zxQfALnTdfc_56.mp4
|           |   └── yawning
|           |       ├── ...
|           └── validation
|           |   ├── ...
└── mmaction
└── ...
```

关于对 Moments in Times 进行训练和验证，可以参照 基础教程。

7.29 Multi-Moments in Time

7.29.1 简介

```
@misc{monfort2019multimoments,
    title={Multi-Moments in Time: Learning and Interpreting Models for Multi-Action Video Understanding},
    author={Mathew Monfort and Kandan Ramakrishnan and Alex Andonian and Barry A McNamara and Alex Lascelles, Bowen Pan, Quanfu Fan, Dan Gutfreund, Rogerio Feris, Aude Oliva},
    year={2019},
    eprint={1911.00232},
    archivePrefix={arXiv},
    primaryClass={cs.CV}
}
```

用户可以参照数据集 官网，获取数据集相关的基本信息。在准备数据集前，请确保命令行当前路径为 `$MMACTION2/tools/data/mmit/`。

7.29.2 步骤 1. Prepare Annotations and Videos

首先，用户需要访问官网，填写申请表来下载数据集。在得到下载链接后，用户可以使用 `bash preprocess_data.sh` 来准备标注文件和视频。请注意此脚本并没有下载标注和视频文件，用户需要根据脚本文件中的注释，提前下载好数据集，并放/软链接到合适的位置。

为加快视频解码速度，用户需要缩小原视频的尺寸，可使用以下命令获取密集编码版视频：

```
python ../resize_videos.py ../../../data/mmit/videos/ ../../../data/mmit/videos_256p_dense_cache --dense --level 2
```

7.29.3 Step 2. 抽取帧和光流

如果用户只想使用视频加载训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果用户有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 上。用户可使用以下命令为 SSD 建立软链接。

```
## 执行这两行指令进行抽取（假设 SSD 挂载在 "/mnt/SSD/" 上）
mkdir /mnt/SSD/mmit_extracted/
ln -s /mnt/SSD/mmit_extracted/ ../../../data/mmit/rawframes
```

如果用户需要抽取 RGB 帧（因为抽取光流的过程十分耗时），可以考虑运行以下命令使用 denseflow **只抽取 RGB 帧**。

```
bash extract_rgb_frames.sh
```

如果用户没有安装 denseflow，则可以运行以下命令使用 OpenCV 抽取 RGB 帧。然而，该方法只能抽取与原始视频分辨率相同的帧。

```
bash extract_rgb_frames_opencv.sh
```

如果用户想抽取 RGB 帧和光流，则可以运行以下脚本进行抽取。

```
bash extract_frames.sh
```

7.29.4 步骤 3. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
bash generate_rawframes_filelist.sh
bash generate_videos_filelist.sh
```

7.29.5 步骤 4. 检查目录结构

在完成 Multi-Moments in Time 数据集准备流程后，用户可以得到 Multi-Moments in Time 的 RGB 帧 + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，Multi-Moments in Time 的文件结构如下：

```
mmaction2/
└── data
    └── mmit
        ├── annotations
        │   ├── moments_categories.txt
        │   ├── trainingSet.txt
        │   └── validationSet.txt
        ├── mmit_train_rawframes.txt
        ├── mmit_train_videos.txt
        ├── mmit_val_rawframes.txt
        ├── mmit_val_videos.txt
        ├── rawframes
        │   ├── 0-3-6-2-9-1-2-6-14603629126_5
        │   │   ├── flow_x_00001.jpg
        │   │   ├── flow_x_00002.jpg
```

(下页继续)

(续上页)

```
        |   |   ├── ...
        |   |   ├── flow_y_00001.jpg
        |   |   ├── flow_y_00002.jpg
        |   |   ├── ...
        |   |   ├── img_00001.jpg
        |   |   └── img_00002.jpg
        |   |   ├── ...
        |   └── yt-zxQfALnTdfc_56
        |   |   ├── ...
        |   └── ...

        └── videos
            └── adult+female+singing
                ├── 0-3-6-2-9-1-2-6-14603629126_5.mp4
                └── yt-zxQfALnTdfc_56.mp4
            └── ...
```

关于对 Multi-Moments in Time 进行训练和验证，可以参照 基础教程。

7.30 OmniSource

7.30.1 简介

```
@article{duan2020omni,
  title={Omni-sourced Webly-supervised Learning for Video Recognition},
  author={Duan, Haodong and Zhao, Yue and Xiong, Yuanjun and Liu, Wentao and Lin,␣
↪Dahua},
  journal={arXiv preprint arXiv:2003.13042},
  year={2020}
}
```

MMAction2 中发布了 OmniSource 网络数据集的一个子集 (来自论文 Omni-sourced Webly-supervised Learning for Video Recognition)。OmniSource 数据集中所有类别均来自 Kinetics-400。MMAction2 所提供的子集包含属于 Mini-Kinetics 数据集 200 类动作的网络数据 (Mini-inetics 数据集由论文 Rethinking Spatiotemporal Feature Learning: Speed-Accuracy Trade-offs in Video Classification 提出)。

MMAction2 提供所有数据源中属于 Mini-Kinetics 200 类动作的数据，这些数据源包含：Kinetics 数据集，Kinetics 原始数据集（未经裁剪的长视频），来自 Google 和 Instagram 的网络图片，来自 Instagram 的网络视频。为获取这一数据集，用户需先填写 数据申请表。在接收到申请后，下载链接将被发送至用户邮箱。由于发布的数据集均为爬取所得的原始数据，数据集较大，下载需要一定时间。下表中提供了 OmniSource 数据集各个分量的统计信息。

MMAction2 所发布的 OmniSource 数据集目录结构如下所示：

```
OmniSource/
├── annotations
│   ├── googleimage_200
│   │   ├── googleimage_200.txt                          从 Google 爬取到的所有图片列表
│   │   ├── tsn_8seg_googleimage_200_duplicate.txt       从 Google 爬取到的，疑似与 k200-val␣
↪中样本重复的正样本列表
│   │   ├── tsn_8seg_googleimage_200.txt                 从 Google 爬取到的，经过 teacher 模
型过滤的正样本列表
│   │   └── tsn_8seg_googleimage_200_wodup.txt           从 Google 爬取到的，经过 teacher 模
型过滤及去重的正样本列表
│   ├── insimage_200
│   │   ├── insimage_200.txt
│   │   ├── tsn_8seg_insimage_200_duplicate.txt
│   │   ├── tsn_8seg_insimage_200.txt
│   │   └── tsn_8seg_insimage_200_wodup.txt
│   ├── insvideo_200
│   │   ├── insvideo_200.txt
│   │   ├── slowonly_8x8_insvideo_200_duplicate.txt
│   │   ├── slowonly_8x8_insvideo_200.txt
│   │   └── slowonly_8x8_insvideo_200_wodup.txt
│   ├── k200_actions.txt                                 MiniKinetics 中 200 类动作的名称
│   ├── K400_to_MiniKinetics_classidx_mapping.json       Kinetics 中的类索引至␣
↪MiniKinetics 中的类索引的映射
│   ├── kinetics_200
│   │   ├── k200_train.txt
│   │   └── k200_val.txt
│   └── kinetics_raw_200
│       └── slowonly_8x8_kinetics_raw_200.json           经 teacher 模型过滤后的 Kinetics 原
始视频片段
├── googleimage_200                                      共 10 卷
│   ├── vol_0.tar
│   ├── ...
│   └── vol_9.tar
├── insimage_200                                         共 10 卷
│   ├── vol_0.tar
│   ├── ...
│   └── vol_9.tar
├── insvideo_200                                         共 20 卷
│   ├── vol_00.tar
│   ├── ...
│   └── vol_19.tar
├── kinetics_200_train
│   └── kinetics_200_train.tar
├── kinetics_200_val
```

(下页继续)

(续上页)

```
|   └── kinetics_200_val.tar
└── kinetics_raw_200_train                          共 16 卷
    ├── vol_0.tar
    ├── ...
    └── vol_15.tar
```

7.30.2 数据准备

用户需要首先完成数据下载，对于 kinetics_200 和三个网络数据集 googleimage_200, insimage_200, insvideo_200，用户仅需解压各压缩卷并将其合并至一处。

对于 Kinetics 原始视频，由于直接读取长视频非常耗时，用户需要先将其分割为小段。MMAction2 提供了名为 trim_raw_video.py 的脚本，用于将长视频分割至 10 秒的小段（分割完成后删除长视频）。用户可利用这一脚本分割长视频。

所有数据应位于 data/OmniSource/ 目录下。完成数据准备后，data/OmniSource/ 目录的结构应如下所示（为简洁，省去了训练及测试时未使用的文件）：

```
data/OmniSource/
├── annotations
|   ├── googleimage_200
|   |   └── tsn_8seg_googleimage_200_wodup.txt    Positive file list of images␣
↪crawled from Google, filtered by the teacher model, after de-duplication.
|   ├── insimage_200
|   |   └── tsn_8seg_insimage_200_wodup.txt
|   ├── insvideo_200
|   |   └── slowonly_8x8_insvideo_200_wodup.txt
|   ├── kinetics_200
|   |   ├── k200_train.txt
|   |   └── k200_val.txt
|   ├── kinetics_raw_200
|   |   └── slowonly_8x8_kinetics_raw_200.json    Kinetics Raw Clips filtered by the␣
↪teacher model.
|   └── webimage_200
|       └── tsn_8seg_webimage_200_wodup.txt       The union of `tsn_8seg_googleimage_
↪200_wodup.txt` and `tsn_8seg_insimage_200_wodup.txt`
├── googleimage_200
|   ├── 000
|   |   ├── 00
|   |   |   ├── 000001.jpg
|   |   |   ├── ...
|   |   |   └── 000901.jpg
|   |   ├── ...
```

(下页继续)

(续上页)

```
|   |   ├── 19
|   ├── ...
|   └── 199
├── insimage_200
|   ├── 000
|   |   ├── abseil
|   |   |   ├── 1J9tKWCNgV_0.jpg
|   |   |   ├── ...
|   |   |   └── 1J9tKWCNgV_0.jpg
|   |   ├── abseiling
|   ├── ...
|   └── 199
├── insvideo_200
|   ├── 000
|   |   ├── abseil
|   |   |   ├── B00arxogubl.mp4
|   |   |   ├── ...
|   |   |   └── BzYsP0HIvbt.mp4
|   |   ├── abseiling
|   ├── ...
|   └── 199
├── kinetics_200_train
|   ├── 0074cdXclLU.mp4
|   ├── ...
|   ├── zzzlyL61Fyo.mp4
├── kinetics_200_val
|   ├── 01fAWEHzudA.mp4
|   ├── ...
|   ├── zymA_6jZIz4.mp4
└── kinetics_raw_200_train
|   ├── pref_
|   |   ├── ___dTOdxzXY
|   |   |   ├── part_0.mp4
|   |   |   ├── ...
|   |   |   ├── part_6.mp4
|   |   ├── ...
|   |   └── _zygwGDE2EM
|   ├── ...
|   └── prefZ
```

7.31 骨架数据集

```
@misc{duan2021revisiting,
      title={Revisiting Skeleton-based Action Recognition},
      author={Haodong Duan and Yue Zhao and Kai Chen and Dian Shao and Dahua Lin and
↪Bo Dai},
      year={2021},
      eprint={2104.13586},
      archivePrefix={arXiv},
      primaryClass={cs.CV}
}
```

7.31.1 简介

MMAction2 发布 Revisiting Skeleton-based Action Recognition 论文中所使用的骨架标注。默认使用 Faster-RCNN 作为人体检测器，使用 HRNet-w32 作为单人姿态估计模型。对于 FineGYM 数据集，MMAction2 使用的是运动员的真实框标注，而非检测器所出的框。目前，MMAction2 已发布 FineGYM 和 NTURGB-D Xsub 部分的骨架标注，其他数据集的标注也将很快发布。

7.31.2 标注文件

目前，MMAction2 支持 HMDB51, UCF101, FineGYM 和 NTURGB+D 数据集。对于 FineGYM 数据集，用户可以使用以下脚本下载标注文件。

```
bash download_annotations.sh ${DATASET}
```

由于 NTURGB+D 数据集的 使用条例，MMAction2 并未直接发布实验中所使用的标注文件。因此，这里提供生成 NTURGB+D 数据集中视频的姿态标注文件，这将生成一个 dict 数据并将其保存为一个 pickle 文件。用户可以生成一个 list 用以包含对应视频的 dict 数据，并将其保存为一个 pickle 文件。之后，用户可以获得 `ntu60_xsub_train.pkl`, `ntu60_xsub_val.pkl`, `ntu120_xsub_train.pkl`, `ntu120_xsub_val.pkl` 文件用于训练。

对于无法进行姿态提取的用户，这里提供了上述流程的输出结果，分别对应 NTURGB-D 数据集的 4 个部分：

- ntu60_xsub_train: https://download.openmmlab.com/mmaction/posec3d/ntu60_xsub_train.pkl
- ntu60_xsub_val: https://download.openmmlab.com/mmaction/posec3d/ntu60_xsub_val.pkl
- ntu120_xsub_train: https://download.openmmlab.com/mmaction/posec3d/ntu120_xsub_train.pkl
- ntu120_xsub_val: https://download.openmmlab.com/mmaction/posec3d/ntu120_xsub_val.pkl
- hmdb51: https://download.openmmlab.com/mmaction/posec3d/hmdb51.pkl
- ucf101: https://download.openmmlab.com/mmaction/posec3d/ucf101.pkl

若想生成单个视频的 2D 姿态标注文件，首先，用户需要由源码安装 mmdetection 和 mmpose。之后，用户需要在 `ntu_pose_extraction.py` 中指定 `mmdet_root` 和 `mmpose_root` 变量。最后，用户可使用以下脚本进行 NTURGB+D 视频的姿态提取：

```
python ntu_pose_extraction.py S001C001P001R001A001_rgb.avi S001C001P001R001A001.pkl
```

在用户获得数据集某部分所有视频的姿态标注文件（如 `ntu60_xsub_val`）后，可以将其集合成一个 list 数据并保存为 `ntu60_xsub_val.pkl`。用户可用这些大型 pickle 文件进行训练和测试。

7.31.3 PoseC3D 的标注文件格式

这里简单介绍 PoseC3D 的标注文件格式。以 `gym_train.pkl` 为例：`gym_train.pkl` 存储一个长度为 20484 的 list，list 的每一项为单个视频的骨架标注 dict。每个 dict 的内容如下：

- keypoint：关键点坐标，大小为 N（## 人数）x T（时序长度）x K（# 关键点, 这里为 17）x 2 （x，y 坐标）的 numpy array 数据类型
- keypoint_score：关键点的置信分数，大小为 N（## 人数）x T（时序长度）x K（# 关键点, 这里为 17）的 numpy array 数据类型
- frame_dir: 对应视频名
- label: 动作类别
- img_shape: 每一帧图像的大小
- original_shape: 同 `img_shape`
- total_frames: 视频时序长度

如用户想使用自己的数据集训练 PoseC3D，可以参考 Custom Dataset Training。

7.31.4 可视化

为了可视化骨架数据，用户需要准备 RGB 的视频。详情可参考 visualize_heatmap_volume。这里提供一些 NTU-60 和 FineGYM 上的例子

7.31.5 如何将 NTU RGB+D 原始数据转化为 MMAction2 格式（转换好的标注文件目前仅适用于 GCN 模型）

这里介绍如何将 NTU RGB+D 原始数据转化为 MMAction2 格式。首先，需要从 https://github.com/shahroudy/NTURGB-D 下载原始 NTU-RGBD 60 和 NTU-RGBD 120 数据集的原始骨架数据。

对于 NTU-RGBD 60 数据集，可使用以下脚本

```
python gen_ntu_rgbd_raw.py --data-path your_raw_nturgbd60_skeleton_path --ignored-
↪sample-path NTU_RGBD_samples_with_missing_skeletons.txt --out-folder your_nturgbd60_
↪output_path --task ntu60
```

对于 NTU-RGBD 120 数据集，可使用以下脚本

```
python gen_ntu_rgbd_raw.py --data-path your_raw_nturgbd120_skeleton_path --ignored-
↪sample-path NTU_RGBD120_samples_with_missing_skeletons.txt --out-folder your_
↪nturgbd120_output_path --task ntu120
```

7.31.6 转换其他第三方项目的骨骼标注

MMAction2 提供脚本以将其他第三方项目的骨骼标注转至 MMAction2 格式，如：

- BABEL: `babel2mma2.py`

待办项：

- [x] FineGYM
- [x] NTU60_XSub
- [x] NTU120_XSub
- [x] NTU60_XView
- [x] NTU120_XSet
- [x] UCF101
- [x] HMDB51
- [ ] Kinetics

7.32 Something-Something V1

7.32.1 简介

```
@misc{goyal2017something,
      title={The "something something" video database for learning and evaluating␣
↪visual common sense},
      author={Raghav Goyal and Samira Ebrahimi Kahou and Vincent Michalski and Joanna␣
↪Materzyńska and Susanne Westphal and Heuna Kim and Valentin Haenel and Ingo Fruend␣
↪and Peter Yianilos and Moritz Mueller-Freitag and Florian Hoppe and Christian␣
↪Thurau and Ingo Bax and Roland Memisevic},
      year={2017},
```

(下页继续)

(续上页)

```
      eprint={1706.04261},
      archivePrefix={arXiv},
      primaryClass={cs.CV}
}
```

用户可参考该数据集的 官网，以获取数据集相关的基本信息。在数据集准备前，请确保命令行当前路径为 `$MMACTION2/tools/data/sthv1/`。

7.32.2 步骤 1. 下载标注文件

由于 Something-Something V1 的官方网站已经失效，用户需要通过第三方源下载原始数据集。下载好的标注文件需要放在 `$MMACTION2/data/sthv1/annotations` 文件夹下。

7.32.3 步骤 2. 准备 RGB 帧

官方数据集并未提供原始视频文件，只提供了对原视频文件进行抽取得到的 RGB 帧，用户可在第三方源直接下载视频帧。

将下载好的压缩文件放在 `$MMACTION2/data/sthv1/` 文件夹下，并使用以下脚本进行解压。

```
cd $MMACTION2/data/sthv1/
cat 20bn-something-something-v1-?? | tar zx
cd $MMACTION2/tools/data/sthv1/
```

如果用户只想使用 RGB 帧，则可以跳过中间步骤至步骤 5 以直接生成视频帧的文件列表。由于官网的 JPG 文件名形如 “%05d.jpg”（比如，” 00001.jpg”），需要在配置文件的 `data.train`, `data.val` 和 `data.test` 处添加 `"filename_tmpl='{:05}.jpg'"` 代码，以修改文件名模板。

```
data = dict(
    videos_per_gpu=16,
    workers_per_gpu=2,
    train=dict(
        type=dataset_type,
        ann_file=ann_file_train,
        data_prefix=data_root,
        filename_tmpl='{:05}.jpg',
        pipeline=train_pipeline),
    val=dict(
        type=dataset_type,
        ann_file=ann_file_val,
        data_prefix=data_root_val,
        filename_tmpl='{:05}.jpg',
```

(下页继续)

(续上页)

```
        pipeline=val_pipeline),
    test=dict(
        type=dataset_type,
        ann_file=ann_file_test,
        data_prefix=data_root_val,
        filename_tmpl='{:05}.jpg',
        pipeline=test_pipeline))
```

7.32.4 步骤 3. 抽取光流

如果用户只想使用原 RGB 帧加载训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果拥有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 中。

可以运行以下命令为 SSD 建立软链接。

```
## 执行这两行进行抽取（假设 SSD 挂载在 "/mnt/SSD/"）
mkdir /mnt/SSD/sthv1_extracted/
ln -s /mnt/SSD/sthv1_extracted/ ../../../data/sthv1/rawframes
```

如果想抽取光流，则可以运行以下脚本从 RGB 帧中抽取出光流。

```
cd $MMACTION2/tools/data/sthv1/
bash extract_flow.sh
```

7.32.5 步骤 4: 编码视频

如果用户只想使用 RGB 帧加载训练，则该部分是 **可选项**。

用户可以运行以下命令进行视频编码。

```
cd $MMACTION2/tools/data/sthv1/
bash encode_videos.sh
```

7.32.6 步骤 5. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
cd $MMACTION2/tools/data/sthv1/
bash generate_{rawframes, videos}_filelist.sh
```

7.32.7 步骤 6. 检查文件夹结构

在完成所有 Something-Something V1 数据集准备流程后，用户可以获得对应的 RGB + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，Something-Something V1 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── sthv1
|   |   ├── sthv1_{train,val}_list_rawframes.txt
|   |   ├── sthv1_{train,val}_list_videos.txt
|   |   ├── annotations
|   |   ├── videos
|   |   |   ├── 1.mp4
|   |   |   ├── 2.mp4
|   |   |   ├──...
|   |   ├── rawframes
|   |   |   ├── 1
|   |   |   |   ├── 00001.jpg
|   |   |   |   ├── 00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_x_00001.jpg
|   |   |   |   ├── flow_x_00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_y_00001.jpg
|   |   |   |   ├── flow_y_00002.jpg
|   |   |   |   ├── ...
|   |   |   ├── 2
|   |   |   ├── ...
```

关于对 Something-Something V1 进行训练和验证，可以参考 基础教程。

7.33 Something-Something V2

7.33.1 简介

```
@misc{goyal2017something,
      title={The "something something" video database for learning and evaluating visual common sense},
      author={Raghav Goyal and Samira Ebrahimi Kahou and Vincent Michalski and Joanna Materzyńska and Susanne Westphal and Heuna Kim and Valentin Haenel and Ingo Fruend and Peter Yianilos and Moritz Mueller-Freitag and Florian Hoppe and Christian Thurau and Ingo Bax and Roland Memisevic},
      year={2017},
      eprint={1706.04261},
      archivePrefix={arXiv},
      primaryClass={cs.CV}
}
```

用户可参考该数据集的 官网，以获取数据集相关的基本信息。在数据集准备前，请确保命令行当前路径为 `$MMACTION2/tools/data/sthv2/`。

7.33.2 步骤 1. 下载标注文件

首先，用户需要在 官网 完成注册，才能下载标注文件。下载好的标注文件需要放在 `$MMACTION2/data/sthv2/annotations` 文件夹下。

```
cd $MMACTION2/data/sthv2/annotations
unzip 20bn-something-something-download-package-labels.zip
find ./labels -name "*.json" -exec sh -c 'cp "$1" "something-something-v2-$(basename $1)"' _ {} \;
```

7.33.3 步骤 2. 准备视频

之后，用户可将下载好的压缩文件放在 `$MMACTION2/data/sthv2/` 文件夹下，并且使用以下指令进行解压。

```
cd $MMACTION2/data/sthv2/
cat 20bn-something-something-v2-?? | tar zx
cd $MMACTION2/tools/data/sthv2/
```

7.33.4 步骤 3. 抽取 RGB 帧和光流

如果用户只想使用视频加载训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果拥有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 中。

可以运行以下命令为 SSD 建立软链接。

```
## 执行这两行进行抽取（假设 SSD 挂载在 "/mnt/SSD/"）
mkdir /mnt/SSD/sthv2_extracted/
ln -s /mnt/SSD/sthv2_extracted/ ../../../data/sthv2/rawframes
```

如果用户需要抽取 RGB 帧（因为抽取光流的过程十分耗时），可以考虑运行以下命令使用 denseflow **只抽取 RGB 帧**。

```
cd $MMACTION2/tools/data/sthv2/
bash extract_rgb_frames.sh
```

如果用户没有安装 denseflow，则可以运行以下命令使用 OpenCV 抽取 RGB 帧。然而，该方法只能抽取与原始视频分辨率相同的帧。

```
cd $MMACTION2/tools/data/sthv2/
bash extract_rgb_frames_opencv.sh
```

如果用户想抽取 RGB 帧和光流，则可以运行以下脚本进行抽取。

```
cd $MMACTION2/tools/data/sthv2/
bash extract_frames.sh
```

7.33.5 步骤 4. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
cd $MMACTION2/tools/data/sthv2/
bash generate_{rawframes, videos}_filelist.sh
```

7.33.6 步骤 5. 检查文件夹结构

在完成所有 Something-Something V2 数据集准备流程后，用户可以获得对应的 RGB + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，Something-Something V2 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── sthv2
|   |   ├── sthv2_{train,val}_list_rawframes.txt
|   |   ├── sthv2_{train,val}_list_videos.txt
|   |   ├── annotations
|   |   ├── videos
|   |   |   ├── 1.mp4
|   |   |   ├── 2.mp4
|   |   |   ├──...
|   |   ├── rawframes
|   |   |   ├── 1
|   |   |   |   ├── img_00001.jpg
|   |   |   |   ├── img_00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_x_00001.jpg
|   |   |   |   ├── flow_x_00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_y_00001.jpg
|   |   |   |   ├── flow_y_00002.jpg
|   |   |   |   ├── ...
|   |   |   ├── 2
|   |   |   ├── ...
```

关于对 Something-Something V2 进行训练和验证，可以参考 基础教程。

7.34 THUMOS'14

7.34.1 简介

```
@misc{THUMOS14,
    author = {Jiang, Y.-G. and Liu, J. and Roshan Zamir, A. and Toderici, G. and
↪Laptev,
    I. and Shah, M. and Sukthankar, R.},
    title = {{THUMOS} Challenge: Action Recognition with a Large
    Number of Classes},
    howpublished = "\url{http://crcv.ucf.edu/THUMOS14/}",
    Year = {2014}
}
```

用户可以参照数据集 官网，获取数据集相关的基本信息。在准备数据集前，请确保命令行当前路径为 `$MMACTION2/tools/data/thumos14/`。

7.34.2 步骤 1. 下载标注文件

首先，用户可使用以下命令下载标注文件。

```
cd $MMACTION2/tools/data/thumos14/
bash download_annotations.sh
```

7.34.3 步骤 2. 下载视频

之后，用户可使用以下指令下载视频

```
cd $MMACTION2/tools/data/thumos14/
bash download_videos.sh
```

7.34.4 步骤 3. 抽取帧和光流

如果用户只想使用视频加载训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果用户有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 上。用户可使用以下命令为 SSD 建立软链接。

```
## 执行这两行指令进行抽取（假设 SSD 挂载在 "/mnt/SSD/" 上）
mkdir /mnt/SSD/thumos14_extracted/
ln -s /mnt/SSD/thumos14_extracted/ ../data/thumos14/rawframes/
```

如果用户需要抽取 RGB 帧（因为抽取光流的过程十分耗时），可以考虑运行以下命令使用 denseflow **只抽取 RGB 帧**。

```
cd $MMACTION2/tools/data/thumos14/
bash extract_rgb_frames.sh
```

如果用户没有安装 denseflow，则可以运行以下命令使用 OpenCV 抽取 RGB 帧。然而，该方法只能抽取与原始视频分辨率相同的帧。

```
cd $MMACTION2/tools/data/thumos14/
bash extract_rgb_frames_opencv.sh
```

如果用户想抽取 RGB 帧和光流，则可以运行以下脚本进行抽取。

```
cd $MMACTION2/tools/data/thumos14/
bash extract_frames.sh tvl1
```

7.34.5 步骤 4. 生成文件列表

如果用户不使用 SSN 模型，则该部分是 **可选项**。

可使用运行以下脚本下载预先计算的候选标签。

```
cd $MMACTION2/tools/data/thumos14/
bash fetch_tag_proposals.sh
```

7.34.6 步骤 5. 去规范化候选文件

如果用户不使用 SSN 模型，则该部分是 **可选项**。

可运行以下脚本，来根据本地原始帧的实际数量，去规范化预先计算的候选标签。

```
cd $MMACTION2/tools/data/thumos14/
bash denormalize_proposal_file.sh
```

7.34.7 步骤 6. 检查目录结构

在完成 THUMOS’14 数据集准备流程后，用户可以得到 THUMOS’14 的 RGB 帧 + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，THUMOS’14 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── thumos14
|   |   ├── proposals
|   |   |   ├── thumos14_tag_val_normalized_proposal_list.txt
|   |   |   ├── thumos14_tag_test_normalized_proposal_list.txt
|   |   ├── annotations_val
|   |   ├── annotations_test
|   |   ├── videos
|   |   |   ├── val
|   |   |   |   ├── video_validation_0000001.mp4
|   |   |   |   ├── ...
|   |   |   ├── test
|   |   |   |   ├── video_test_0000001.mp4
|   |   |   |   ├── ...
|   |   ├── rawframes
|   |   |   ├── val
|   |   |   |   ├── video_validation_0000001
|   |   |   |   |   ├── img_00001.jpg
|   |   |   |   |   ├── img_00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── flow_x_00001.jpg
|   |   |   |   |   ├── flow_x_00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── flow_y_00001.jpg
|   |   |   |   |   ├── flow_y_00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   ├── ...
|   |   |   ├── test
|   |   |   |   ├── video_test_0000001
```

关于对 THUMOS’14 进行训练和验证，可以参照 基础教程。

7.35 UCF-101

7.35.1 简介

```
@article{Soomro2012UCF101AD,
  title={UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild},
  author={K. Soomro and A. Zamir and M. Shah},
  journal={ArXiv},
  year={2012},
  volume={abs/1212.0402}
}
```

用户可参考该数据集的 官网，以获取数据集相关的基本信息。在数据集准备前，请确保命令行当前路径为 `$MMACTION2/tools/data/ucf101/`。

7.35.2 步骤 1. 下载标注文件

首先，用户可运行以下脚本下载标注文件。

```
bash download_annotations.sh
```

7.35.3 步骤 2. 准备视频文件

之后，用户可运行以下脚本准备视频文件。

```
bash download_videos.sh
```

用户可使用以下脚本，对原视频进行裁剪，得到密集编码且更小尺寸的视频。

```
python ../resize_videos.py ../../../data/ucf101/videos/ ../../../data/ucf101/videos_
↪256p_dense_cache --dense --level 2 --ext avi
```

7.35.4 步骤 3. 抽取视频帧和光流

如果用户只想使用视频加载训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果拥有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 中。所抽取的视频帧和光流约占据 100 GB 的存储空间。

可以运行以下命令为 SSD 建立软链接。

```
## 执行这两行进行抽取（假设 SSD 挂载在 "/mnt/SSD/"）
mkdir /mnt/SSD/ucf101_extracted/
ln -s /mnt/SSD/ucf101_extracted/ ../../../data/ucf101/rawframes
```

如果用户需要抽取 RGB 帧（因为抽取光流的过程十分耗时），可以考虑运行以下命令使用 denseflow **只抽取 RGB 帧**。

```
bash extract_rgb_frames.sh
```

如果用户没有安装 denseflow，则可以运行以下命令使用 OpenCV 抽取 RGB 帧。然而，该方法只能抽取与原始视频分辨率相同的帧。

```
bash extract_rgb_frames_opencv.sh
```

如果用户想抽取 RGB 帧和光流，则可以运行以下脚本使用“tvl1”算法进行抽取。

```
bash extract_frames.sh
```

7.35.5 步骤 4. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
bash generate_videos_filelist.sh
bash generate_rawframes_filelist.sh
```

7.35.6 步骤 5. 检查文件夹结构

在完成所有 UCF-101 数据集准备流程后，用户可以获得对应的 RGB + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，UCF-101 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── ucf101
|   |   ├── ucf101_{train,val}_split_{1,2,3}_rawframes.txt
|   |   ├── ucf101_{train,val}_split_{1,2,3}_videos.txt
|   |   ├── annotations
|   |   ├── videos
|   |   |   ├── ApplyEyeMakeup
|   |   |   |   ├── v_ApplyEyeMakeup_g01_c01.avi
```

(下页继续)

(续上页)

```
|   |   |   ├── YoYo
|   |   |   |   ├── v_YoYo_g25_c05.avi
|   |   ├── rawframes
|   |   |   ├── ApplyEyeMakeup
|   |   |   |   ├── v_ApplyEyeMakeup_g01_c01
|   |   |   |   |   ├── img_00001.jpg
|   |   |   |   |   ├── img_00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── flow_x_00001.jpg
|   |   |   |   |   ├── flow_x_00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── flow_y_00001.jpg
|   |   |   |   |   ├── flow_y_00002.jpg
|   |   |   ├── ...
|   |   |   ├── YoYo
|   |   |   |   ├── v_YoYo_g01_c01
|   |   |   |   ├── ...
|   |   |   |   ├── v_YoYo_g25_c05
```

关于对 UCF-101 进行训练和验证，可以参考 基础教程。

7.36 UCF101-24

7.36.1 简介

```
@article{Soomro2012UCF101AD,
  title={UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild},
  author={K. Soomro and A. Zamir and M. Shah},
  journal={ArXiv},
  year={2012},
  volume={abs/1212.0402}
}
```

用户可参考该数据集的 官网，以获取数据集相关的基本信息。在数据集准备前，请确保命令行当前路径为 `$MMACTION2/tools/data/ucf101_24/`。

7.36.2 下载和解压

用户可以从 这里 下载 RGB 帧，光流和标注文件。该数据由 MOC 代码库提供，参考自 act-detector 和 corrected-UCF101-Annots。

注意：UCF101-24 的标注文件来自于 这里，该标注文件相对于其他标注文件更加准确。

用户在下载 `UCF101_v2.tar.gz` 文件后，需将其放置在 `$MMACTION2/tools/data/ucf101_24/` 目录下，并使用以下指令进行解压：

```
tar -zxvf UCF101_v2.tar.gz
```

7.36.3 检查文件夹结构

经过解压后，用户将得到 `rgb-images` 文件夹，`brox-images` 文件夹和 `UCF101v2-GT.pkl` 文件。

在整个 MMAction2 文件夹下，UCF101_24 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── ucf101_24
|   |   ├── brox-images
|   |   |   ├── Basketball
|   |   |   |   ├── v_Basketball_g01_c01
|   |   |   |   |   ├── 00001.jpg
|   |   |   |   |   ├── 00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── 00140.jpg
|   |   |   |   |   ├── 00141.jpg
|   |   |   ├── ...
|   |   |   ├── WalkingWithDog
|   |   |   |   ├── v_WalkingWithDog_g01_c01
|   |   |   |   ├── ...
|   |   |   |   ├── v_WalkingWithDog_g25_c04
|   |   ├── rgb-images
|   |   |   ├── Basketball
|   |   |   |   ├── v_Basketball_g01_c01
|   |   |   |   |   ├── 00001.jpg
|   |   |   |   |   ├── 00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── 00140.jpg
|   |   |   |   |   ├── 00141.jpg
```

(下页继续)

(续上页)

```
|   |   |   ├── ...
|   |   |   ├── WalkingWithDog
|   |   |   |   ├── v_WalkingWithDog_g01_c01
|   |   |   |   ├── ...
|   |   |   |   ├── v_WalkingWithDog_g25_c04
|   |   ├── UCF101v2-GT.pkl
```

注意：`UCF101v2-GT.pkl` 作为一个缓存文件，它包含 6 个项目：

1. `labels` (list)：24 个行为类别名称组成的列表
2. `gttubes` (dict)：每个视频对应的基准 tubes 组成的字典 **gttube** 是由标签索引和 tube 列表组成的字典 **tube** 是一个 `nframes` 行和 5 列的 numpy array，每一列的形式如 `<frame index> <x1> <y1> <x2> <y2>`
3. `nframes` (dict)：用以表示每个视频对应的帧数，如 `'HorseRiding/v_HorseRiding_g05_c02': 151`
4. `train_videos` (list)：包含 `nsplits=1` 的元素，每一项都包含了训练视频的列表
5. `test_videos` (list)：包含 `nsplits=1` 的元素，每一项都包含了测试视频的列表
6. `resolution` (dict)：每个视频对应的分辨率（形如 (h,w)），如 `'FloorGymnastics/v_FloorGymnastics_g09_c03': (240, 320)`

7.37 ActivityNet

7.37.1 简介

```
@article{Heilbron2015ActivityNetAL,
  title={ActivityNet: A large-scale video benchmark for human activity understanding},
  author={Fabian Caba Heilbron and Victor Escorcia and Bernard Ghanem and Juan Carlos␣
↪Niebles},
  journal={2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)},
  year={2015},
  pages={961-970}
}
```

用户可参考该数据集的 官网，以获取数据集相关的基本信息。对于时序动作检测任务，用户可以使用这个 代码库 提供的缩放过（rescaled）的 ActivityNet 特征，或者使用 MMAction2 进行特征提取（这将具有更高的精度）。MMAction2 同时提供了以上所述的两种数据使用流程。在数据集准备前，请确保命令行当前路径为 `$MMACTION2/tools/data/activitynet/`。

7.37.2 选项 1：用户可以使用这个代码库提供的特征

步骤 1. 下载标注文件

首先，用户可以使用以下命令下载标注文件。

```
bash download_feature_annotations.sh
```

步骤 2. 准备视频特征

之后，用户可以使用以下命令下载 ActivityNet 特征。

```
bash download_features.sh
```

步骤 3. 处理标注文件

之后，用户可以使用以下命令处理下载的标注文件，以便于训练和测试。该脚本会首先合并两个标注文件，然后再将其分为 `train`, `val` 和 `test` 三个部分。

```
python process_annotations.py
```

7.37.3 选项 2：使用 MMAction2 对官网提供的视频进行特征抽取

步骤 1. 下载标注文件

首先，用户可以使用以下命令下载标注文件。

```
bash download_annotations.sh
```

步骤 2. 准备视频

之后，用户可以使用以下脚本准备视频数据。该代码参考自 官方爬虫，该过程将会耗费较多时间。

```
bash download_videos.sh
```

由于 ActivityNet 数据集中的一些视频已经在 YouTube 失效，官网 在谷歌网盘和百度网盘提供了完整的数据集数据。如果用户想要获取失效的数据集，则需要填写 下载页面 中提供的 需求表格 以获取 7 天的下载权限。

MMAction2 同时也提供了 BSN 代码库 的标注文件的下载步骤。

```
bash download_bsn_videos.sh
```

对于这种情况，该下载脚本将在下载后更新此标注文件，以确保每个视频都存在。

步骤 3. 抽取 RGB 帧和光流

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

可使用以下命令抽取视频帧和光流。

```
bash extract_frames.sh
```

以上脚本将会生成短边 256 分辨率的视频。如果用户想生成短边 320 分辨率的视频（即 320p），或者 340x256 的固定分辨率，用户可以通过改变参数由 `--new-short 256` 至 `--new-short 320`，或者 `--new-width 340 --new-height 256` 进行设置更多细节可参考 数据准备指南

步骤 4. 生成用于 ActivityNet 微调的文件列表

根据抽取的帧，用户可以生成视频级别（video-level）或者片段级别（clip-level）的文件列表，其可用于微调 ActivityNet。

```
python generate_rawframes_filelist.py
```

步骤 5. 在 ActivityNet 上微调 TSN 模型

用户可使用 `configs/recognition/tsn` 目录中的 ActivityNet 配置文件进行 TSN 模型微调。用户需要使用 Kinetics 相关模型（同时支持 RGB 模型与光流模型）进行预训练。

步骤 6. 使用预训练模型进行 ActivityNet 特征抽取

在 ActivityNet 上微调 TSN 模型之后，用户可以使用该模型进行 RGB 特征和光流特征的提取。

```
python tsn_feature_extraction.py --data-prefix ../../../data/ActivityNet/rawframes --
↪data-list ../../../data/ActivityNet/anet_train_video.txt --output-prefix ../../../
↪data/ActivityNet/rgb_feat --modality RGB --ckpt /path/to/rgb_checkpoint.pth

python tsn_feature_extraction.py --data-prefix ../../../data/ActivityNet/rawframes --
↪data-list ../../../data/ActivityNet/anet_val_video.txt --output-prefix ../../../
↪data/ActivityNet/rgb_feat --modality RGB --ckpt /path/to/rgb_checkpoint.pth

python tsn_feature_extraction.py --data-prefix ../../../data/ActivityNet/rawframes --
↪data-list ../../../data/ActivityNet/anet_train_video.txt --output-prefix ../../../
↪data/ActivityNet/flow_feat --modality Flow --ckpt /path/to/flow_checkpoint.pth

python tsn_feature_extraction.py --data-prefix ../../../data/ActivityNet/rawframes --
↪data-list ../../../data/ActivityNet/anet_val_video.txt --output-prefix ../../../
↪data/ActivityNet/flow_feat --modality Flow --ckpt /path/to/flow_checkpoint.pth
```

(下页继续)

(续上页)

在提取完特征后，用户可以使用后处理脚本整合 RGB 特征和光流特征，生成 `100-t X 400-d` 维度的特征用于时序动作检测。

```
python activitynet_feature_postprocessing.py --rgb ../../../data/ActivityNet/rgb_feat␣
↪--flow ../../../data/ActivityNet/flow_feat --dest ../../../data/ActivityNet/
↪mmaction_feat
```

7.37.4 最后一步：检查文件夹结构

在完成所有 ActivityNet 数据集准备流程后，用户可以获得对应的特征文件，RGB + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，ActivityNet 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
│   ├── ActivityNet

(若根据选项 1 进行数据处理)
│   │   ├── anet_anno_{train,val,test,full}.json
│   │   ├── anet_anno_action.json
│   │   ├── video_info_new.csv
│   │   ├── activitynet_feature_cuhk
│   │   │   ├── csv_mean_100
│   │   │   │   ├── v___c8enCfzqw.csv
│   │   │   │   ├── v___dXUJsj3yo.csv
│   │   │   │   ├── ..

(若根据选项 2 进行数据处理)
│   │   ├── anet_train_video.txt
│   │   ├── anet_val_video.txt
│   │   ├── anet_train_clip.txt
│   │   ├── anet_val_clip.txt
│   │   ├── activity_net.v1-3.min.json
│   │   ├── mmaction_feat
│   │   │   ├── v___c8enCfzqw.csv
│   │   │   ├── v___dXUJsj3yo.csv
│   │   │   ├── ..
```

(下页继续)

(续上页)

```
|   |   ├── rawframes
|   |   |   ├── v___c8enCfzqw
|   |   |   |   ├── img_00000.jpg
|   |   |   |   ├── flow_x_00000.jpg
|   |   |   |   ├── flow_y_00000.jpg
|   |   |   |   ├── ..
|   |   |   ├── ..
```

关于对 ActivityNet 进行训练和验证，可以参考 基础教程.

7.38 AVA

7.38.1 简介

```
@inproceedings{gu2018ava,
  title={Ava: A video dataset of spatio-temporally localized atomic visual actions},
  author={Gu, Chunhui and Sun, Chen and Ross, David A and Vondrick, Carl and␣
↪Pantofaru, Caroline and Li, Yeqing and Vijayanarasimhan, Sudheendra and Toderici,␣
↪George and Ricco, Susanna and Sukthankar, Rahul and others},
  booktitle={Proceedings of the IEEE Conference on Computer Vision and Pattern␣
↪Recognition},
  pages={6047--6056},
  year={2018}
}
```

请参照 官方网站 以获取数据集基本信息。在开始之前，用户需确保当前目录为 `$MMACTION2/tools/data/ava/`。

7.38.2 1. 准备标注文件

首先，用户可以使用如下脚本下载标注文件并进行预处理：

```
bash download_annotations.sh
```

这一命令将下载 `ava_v2.1.zip` 以得到 AVA v2.1 标注文件。如用户需要 AVA v2.2 标注文件，可使用以下脚本：

```
VERSION=2.2 bash download_annotations.sh
```

7.38.3 2. 下载视频

用户可以使用以下脚本准备视频，视频准备代码修改自 官方爬虫。注意这一步骤将花费较长时间。

```
bash download_videos.sh
```

亦可使用以下脚本，使用 python 并行下载 AVA 数据集视频：

```
bash download_videos_parallel.sh
```

7.38.4 3. 截取视频

截取每个视频中的 15 到 30 分钟，设定帧率为 30。

```
bash cut_videos.sh
```

7.38.5 4. 提取 RGB 帧和光流

在提取之前，请参考 安装教程 安装 denseflow。

如果用户有足够的 SSD 空间，那么建议将视频抽取为 RGB 帧以提升 I/O 性能。用户可以使用以下脚本为抽取得到的帧文件夹建立软连接：

```
## 执行以下脚本 (假设 SSD 被挂载在 "/mnt/SSD/")
mkdir /mnt/SSD/ava_extracted/
ln -s /mnt/SSD/ava_extracted/ ../data/ava/rawframes/
```

如果用户只使用 RGB 帧（由于光流提取非常耗时），可执行以下脚本使用 denseflow 提取 RGB 帧：

```
bash extract_rgb_frames.sh
```

如果用户未安装 denseflow，可执行以下脚本使用 ffmpeg 提取 RGB 帧：

```
bash extract_rgb_frames_ffmpeg.sh
```

如果同时需要 RGB 帧和光流，可使用如下脚本抽帧：

```
bash extract_frames.sh
```

7.38.6 5. 下载 AVA 上人体检测结果

以下脚本修改自 Long-Term Feature Banks。

可使用以下脚本下载 AVA 上预先计算的人体检测结果：

```
bash fetch_ava_proposals.sh
```

7.38.7 6. 目录结构

在完整完成 AVA 的数据处理后，将得到帧文件夹（RGB 帧和光流帧），视频以及标注文件。

在整个项目目录下（仅针对 AVA），最简目录结构如下所示：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── ava
|   |   ├── annotations
|   |   |   ├── ava_dense_proposals_train.FAIR.recall_93.9.pkl
|   |   |   ├── ava_dense_proposals_val.FAIR.recall_93.9.pkl
|   |   |   ├── ava_dense_proposals_test.FAIR.recall_93.9.pkl
|   |   |   ├── ava_train_v2.1.csv
|   |   |   ├── ava_val_v2.1.csv
|   |   |   ├── ava_train_excluded_timestamps_v2.1.csv
|   |   |   ├── ava_val_excluded_timestamps_v2.1.csv
|   |   |   ├── ava_action_list_v2.1_for_activitynet_2018.pbtxt
|   |   ├── videos
|   |   |   ├── 053oq2xB3oU.mkv
|   |   |   ├── 0f39OWEqJ24.mp4
|   |   |   ├── ...
|   |   ├── videos_15min
|   |   |   ├── 053oq2xB3oU.mkv
|   |   |   ├── 0f39OWEqJ24.mp4
|   |   |   ├── ...
|   |   ├── rawframes
|   |   |   ├── 053oq2xB3oU
|   |   |   |   ├── img_00001.jpg
|   |   |   |   ├── img_00002.jpg
|   |   |   |   ├── ...
```

关于 AVA 数据集上的训练与测试，请参照 基础教程。

7.39 Diving48

7.39.1 简介

```
@inproceedings{li2018resound,
  title={Resound: Towards action recognition without representation bias},
  author={Li, Yingwei and Li, Yi and Vasconcelos, Nuno},
  booktitle={Proceedings of the European Conference on Computer Vision (ECCV)},
  pages={513--528},
  year={2018}
}
```

用户可参考该数据集的 官网，以获取数据集相关的基本信息。在数据集准备前，请确保命令行当前路径为 `$MMACTION2/tools/data/diving48/`。

7.39.2 步骤 1. 下载标注文件

用户可以使用以下命令下载标注文件（考虑到标注的准确性，这里仅下载 V2 版本）。

```
bash download_annotations.sh
```

7.39.3 步骤 2. 准备视频

用户可以使用以下命令下载视频。

```
bash download_videos.sh
```

7.39.4 Step 3. 抽取 RGB 帧和光流

如果用户只想使用视频加载训练，则该部分是 **可选项**。

官网提供的帧压缩包并不完整。若想获取完整的数据，可以使用以下步骤解帧。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果拥有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 中。

可以运行以下命令为 SSD 建立软链接。

```
## 执行这两行进行抽取 (假设 SSD 挂载在 "/mnt/SSD/")
mkdir /mnt/SSD/diving48_extracted/
ln -s /mnt/SSD/diving48_extracted/ ../../../data/diving48/rawframes
```

如果用户需要抽取 RGB 帧（因为抽取光流的过程十分耗时），可以考虑运行以下命令使用 denseflow **只抽取 RGB 帧**。

```
cd $MMACTION2/tools/data/diving48/
bash extract_rgb_frames.sh
```

如果用户没有安装 denseflow，则可以运行以下命令使用 OpenCV 抽取 RGB 帧。然而，该方法只能抽取与原始视频分辨率相同的帧。

```
cd $MMACTION2/tools/data/diving48/
bash extract_rgb_frames_opencv.sh
```

如果用户想抽取 RGB 帧和光流，则可以运行以下脚本进行抽取。

```
cd $MMACTION2/tools/data/diving48/
bash extract_frames.sh
```

7.39.5 步骤 4. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
bash generate_videos_filelist.sh
bash generate_rawframes_filelist.sh
```

7.39.6 步骤 5. 检查文件夹结构

在完成所有 Diving48 数据集准备流程后，用户可以获得对应的 RGB + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，Diving48 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── diving48
|   |   ├── diving48_{train,val}_list_rawframes.txt
|   |   ├── diving48_{train,val}_list_videos.txt
|   |   ├── annotations
|   |   |   ├── Diving48_V2_train.json
|   |   |   ├── Diving48_V2_test.json
|   |   |   ├── Diving48_vocab.json
|   |   ├── videos
```

(下页继续)

(续上页)

```
|   |   |   ├── _8Vy3dlHg2w_00000.mp4
|   |   |   ├── _8Vy3dlHg2w_00001.mp4
|   |   |   ├── ...
|   |   ├── rawframes
|   |   |   ├── 2x00lRzlTVQ_00000
|   |   |   |   ├── img_00001.jpg
|   |   |   |   ├── img_00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_x_00001.jpg
|   |   |   |   ├── flow_x_00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_y_00001.jpg
|   |   |   |   ├── flow_y_00002.jpg
|   |   |   |   ├── ...
|   |   |   ├── 2x00lRzlTVQ_00001
|   |   |   ├── ...
```

关于对 Diving48 进行训练和验证，可以参考 基础教程。

7.40 GYM

7.40.1 简介

```
@inproceedings{shao2020finegym,
  title={Finegym: A hierarchical video dataset for fine-grained action understanding},
  author={Shao, Dian and Zhao, Yue and Dai, Bo and Lin, Dahua},
  booktitle={Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern␣
↪Recognition},
  pages={2616--2625},
  year={2020}
}
```

请参照 项目主页 及 原论文 以获取数据集基本信息。MMAction2 当前支持 GYM99 的数据集预处理。在开始之前，用户需确保当前目录为 `$MMACTION2/tools/data/gym/`。

7.40.2 1. 准备标注文件

首先，用户可以使用如下脚本下载标注文件并进行预处理：

```
bash download_annotations.sh
```

7.40.3 2. 准备视频

用户可以使用以下脚本准备视频，视频准备代码修改自 ActivityNet 爬虫。注意这一步骤将花费较长时间。

```
bash download_videos.sh
```

7.40.4 3. 裁剪长视频至动作级别

用户首先需要使用以下脚本将 GYM 中的长视频依据标注文件裁剪至动作级别。

```
python trim_event.py
```

7.40.5 4. 裁剪动作视频至分动作级别

随后，用户需要使用以下脚本将 GYM 中的动作视频依据标注文件裁剪至分动作级别。将视频的裁剪分成两个级别可以带来更高的效率（在长视频中裁剪多个极短片段异常耗时）。

```
python trim_subaction.py
```

7.40.6 5. 提取 RGB 帧和光流

如果用户仅使用 video loader，则可以跳过本步。

在提取之前，请参考 安装教程 安装 denseflow。

用户可使用如下脚本同时抽取 RGB 帧和光流（提取光流时使用 tvl1 算法）：

```
bash extract_frames.sh
```

7.40.7 6. 基于提取出的分动作生成文件列表

用户可使用以下脚本为 GYM99 生成训练及测试的文件列表：

```
python generate_file_list.py
```

7.40.8 7. 目录结构

在完整完成 GYM 的数据处理后，将得到帧文件夹（RGB 帧和光流帧），动作视频片段，分动作视频片段以及训练测试所用标注文件。

在整个项目目录下（仅针对 GYM），完整目录结构如下所示：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── gym
|   |   ├── annotations
|   |   |   ├── gym99_train_org.txt
|   |   |   ├── gym99_val_org.txt
|   |   |   ├── gym99_train.txt
|   |   |   ├── gym99_val.txt
|   |   |   ├── annotation.json
|   |   |   └── event_annotation.json
|   |   ├── videos
|   |   |   ├── 0LtLS9wROrk.mp4
|   |   |   ├── ...
|   |   |   └── zfqS-wCJSsw.mp4
|   |   ├── events
|   |   |   ├── 0LtLS9wROrk_E_002407_002435.mp4
|   |   |   ├── ...
|   |   |   └── zfqS-wCJSsw_E_006732_006824.mp4
|   |   ├── subactions
|   |   |   ├── 0LtLS9wROrk_E_002407_002435_A_0003_0005.mp4
|   |   |   ├── ...
|   |   |   └── zfqS-wCJSsw_E_006244_006252_A_0000_0007.mp4
|   |   └── subaction_frames
```

关于 GYM 数据集上的训练与测试，请参照 基础教程。

7.41 HMDB51

7.41.1 简介

```
@article{Kuehne2011HMDBAL,
  title={HMDB: A large video database for human motion recognition},
  author={Hilde Kuehne and Hueihan Jhuang and E. Garrote and T. Poggio and Thomas
↪Serre},
  journal={2011 International Conference on Computer Vision},
  year={2011},
  pages={2556-2563}
}
```

用户可以参照数据集 官网，获取数据集相关的基本信息。在准备数据集前，请确保命令行当前路径为 `$MMACTION2/tools/data/hmdb51/`。

为运行下面的 bash 脚本，需要安装 `unrar`。用户可运行 `sudo apt-get install unrar` 安装，或参照 setup，运行 `zzunrar.sh` 脚本实现无管理员权限下的简易安装。

7.41.2 步骤 1. 下载标注文件

首先，用户可使用以下命令下载标注文件。

```
bash download_annotations.sh
```

7.41.3 步骤 2. 下载视频

之后，用户可使用以下指令下载视频

```
bash download_videos.sh
```

7.41.4 步骤 3. 抽取帧和光流

如果用户只想使用视频加载训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果用户有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 上。用户可使用以下命令为 SSD 建立软链接。

```
## 执行这两行指令进行抽取（假设 SSD 挂载在 "/mnt/SSD/" 上）
mkdir /mnt/SSD/hmdb51_extracted/
ln -s /mnt/SSD/hmdb51_extracted/ ../../../data/hmdb51/rawframes
```

如果用户需要抽取 RGB 帧（因为抽取光流的过程十分耗时），可以考虑运行以下命令使用 denseflow **只抽取 RGB 帧**。

```
bash extract_rgb_frames.sh
```

如果用户没有安装 denseflow，则可以运行以下命令使用 OpenCV 抽取 RGB 帧。然而，该方法只能抽取与原始视频分辨率相同的帧。

```
bash extract_rgb_frames_opencv.sh
```

如果用户想抽取 RGB 帧和光流，则可以运行以下脚本，使用“tvl1”算法进行抽取。

```
bash extract_frames.sh
```

7.41.5 步骤 4. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
bash generate_rawframes_filelist.sh
bash generate_videos_filelist.sh
```

7.41.6 步骤 5. 检查目录结构

在完成 HMDB51 数据集准备流程后，用户可以得到 HMDB51 的 RGB 帧 + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，HMDB51 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
│   ├── hmdb51
│   │   ├── hmdb51_{train,val}_split_{1,2,3}_rawframes.txt
│   │   ├── hmdb51_{train,val}_split_{1,2,3}_videos.txt
│   │   ├── annotations
│   │   ├── videos
│   │   │   ├── brush_hair
```

(下页继续)

(续上页)

```
|   |   |   |   ├── April_09_brush_hair_u_nm_np1_ba_goo_0.avi

|   |   |   ├── wave
|   |   |   |   ├── 20060723sfjffbartsinger_wave_f_cm_np1_ba_med_0.avi
|   |   ├── rawframes
|   |   |   ├── brush_hair
|   |   |   |   ├── April_09_brush_hair_u_nm_np1_ba_goo_0
|   |   |   |   |   ├── img_00001.jpg
|   |   |   |   |   ├── img_00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── flow_x_00001.jpg
|   |   |   |   |   ├── flow_x_00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── flow_y_00001.jpg
|   |   |   |   |   ├── flow_y_00002.jpg
|   |   |   ├── ...
|   |   |   ├── wave
|   |   |   |   ├── 20060723sfjffbartsinger_wave_f_cm_np1_ba_med_0
|   |   |   |   ├── ...
|   |   |   |   ├── winKen_wave_u_cm_np1_ri_bad_1
```

关于对 HMDB51 进行训练和验证，可以参照 基础教程。

7.42 HVU

7.42.1 简介

```
@article{Diba2019LargeSH,
  title={Large Scale Holistic Video Understanding},
  author={Ali Diba and M. Fayyaz and Vivek Sharma and Manohar Paluri and Jurgen Gall␣
↪and R. Stiefelhagen and L. Gool},
  journal={arXiv: Computer Vision and Pattern Recognition},
  year={2019}
}
```

请参照 官方项目 及 原论文 以获取数据集基本信息。在开始之前，用户需确保当前目录为 `$MMACTION2/tools/data/hvu/`。

7.42.2 1. 准备标注文件

首先，用户可以使用如下脚本下载标注文件并进行预处理：

```
bash download_annotations.sh
```

此外，用户可使用如下命令解析 HVU 的标签列表：

```
python parse_tag_list.py
```

7.42.3 2. 准备视频

用户可以使用以下脚本准备视频，视频准备代码修改自 ActivityNet 爬虫。注意这一步骤将花费较长时间。

```
bash download_videos.sh
```

7.42.4 3. 提取 RGB 帧和光流

如果用户仅使用 video loader，则可以跳过本步。

在提取之前，请参考 安装教程 安装 denseflow。

用户可使用如下脚本同时抽取 RGB 帧和光流：

```
bash extract_frames.sh
```

该脚本默认生成短边长度为 256 的帧，可参考 数据准备 获得更多细节。

7.42.5 4. 生成文件列表

用户可以使用以下两个脚本分别为视频和帧文件夹生成文件列表：

```
bash generate_videos_filelist.sh
## 为帧文件夹生成文件列表
bash generate_rawframes_filelist.sh
```

7.42.6 5. 为每个 tag 种类生成文件列表

若用户需要为 HVU 数据集的每个 tag 种类训练识别模型，则需要进行此步骤。

步骤 4 中生成的文件列表包含不同类型的标签，仅支持使用 HVUDataset 进行涉及多个标签种类的多任务学习。加载数据的过程中需要使用 `LoadHVULabel` 类进行多类别标签的加载，训练过程中使用 `HVULoss` 作为损失函数。

如果用户仅需训练某一特定类别的标签，例如训练一识别模型用于识别 HVU 中 `action` 类别的标签，则建议使用如下脚本为特定标签种类生成文件列表。新生成的列表将只含有特定类别的标签，因此可使用 `VideoDataset` 或 `RawframeDataset` 进行加载。训训练过程中使用 `BCELossWithLogits` 作为损失函数。

以下脚本为类别为 ${category} 的标签生成文件列表，注意仅支持 HVU 数据集包含的 6 种标签类别: action, attribute, concept, event, object, scene。

```
python generate_sub_file_list.py path/to/filelist.json ${category}
```

对于类别 ${category}，生成的标签列表文件名中将使用 `hvu_${category}` 替代 `hvu`。例如，若原指定文件名为 `hvu_train.json`，则对于类别 action，生成的文件列表名为 `hvu_action_train.json`。

7.42.7 6. 目录结构

在完整完成 HVU 的数据处理后，将得到帧文件夹（RGB 帧和光流帧），视频以及标注文件。

在整个项目目录下（仅针对 HVU），完整目录结构如下所示：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
│   ├── hvu
│   │   ├── hvu_train_video.json
│   │   ├── hvu_val_video.json
│   │   ├── hvu_train.json
│   │   ├── hvu_val.json
│   │   ├── annotations
│   │   ├── videos_train
│   │   │   ├── OLpWTpTC4P8_000570_000670.mp4
│   │   │   ├── xsPKW4tZZBc_002330_002430.mp4
│   │   │   ├── ...
│   │   ├── videos_val
│   │   ├── rawframes_train
│   │   ├── rawframes_val
```

关于 HVU 数据集上的训练与测试，请参照 基础教程。

7.43 Jester

7.43.1 简介

```
@InProceedings{Materzynska_2019_ICCV,
  author = {Materzynska, Joanna and Berger, Guillaume and Bax, Ingo and Memisevic,␣
↪Roland},
  title = {The Jester Dataset: A Large-Scale Video Dataset of Human Gestures},
  booktitle = {Proceedings of the IEEE/CVF International Conference on Computer␣
↪Vision (ICCV) Workshops},
  month = {Oct},
  year = {2019}
}
```

用户可以参照数据集 官网，获取数据集相关的基本信息。在准备数据集前，请确保命令行当前路径为 `$MMACTION2/tools/data/jester/`。

7.43.2 步骤 1. 下载标注文件

首先，用户需要在 官网 完成注册，才能下载标注文件。下载好的标注文件需要放在 `$MMACTION2/data/jester/annotations` 文件夹下。

7.43.3 步骤 2. 准备 RGB 帧

jester 官网 并未提供原始视频文件，只提供了对原视频文件进行抽取得到的 RGB 帧，用户可在 jester 官网 直接下载。

将下载好的压缩文件放在 `$MMACTION2/data/jester/` 文件夹下，并使用以下脚本进行解压。

```
cd $MMACTION2/data/jester/
cat 20bn-jester-v1-?? | tar zx
cd $MMACTION2/tools/data/jester/
```

如果用户只想使用 RGB 帧，则可以跳过中间步骤至步骤 5 以直接生成视频帧的文件列表。由于官网的 JPG 文件名形如 “%05d.jpg”（比如，” 00001.jpg”），需要在配置文件的 `data.train`, `data.val` 和 `data.test` 处添加 `"filename_tmpl='{:05}.jpg'"` 代码，以修改文件名模板。

```
data = dict(
    videos_per_gpu=16,
```

(下页继续)

(续上页)

```
    workers_per_gpu=2,
    train=dict(
        type=dataset_type,
        ann_file=ann_file_train,
        data_prefix=data_root,
        filename_tmpl='{:05}.jpg',
        pipeline=train_pipeline),
    val=dict(
        type=dataset_type,
        ann_file=ann_file_val,
        data_prefix=data_root_val,
        filename_tmpl='{:05}.jpg',
        pipeline=val_pipeline),
    test=dict(
        type=dataset_type,
        ann_file=ann_file_test,
        data_prefix=data_root_val,
        filename_tmpl='{:05}.jpg',
        pipeline=test_pipeline))
```

7.43.4 步骤 3. 抽取光流

如果用户只想使用 RGB 帧训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果拥有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 中。

可以运行以下命令为 SSD 建立软链接。

```
## 执行这两行进行抽取 (假设 SSD 挂载在 "/mnt/SSD/")
mkdir /mnt/SSD/jester_extracted/
ln -s /mnt/SSD/jester_extracted/ ../../../data/jester/rawframes
```

如果想抽取光流，则可以运行以下脚本从 RGB 帧中抽取出光流。

```
cd $MMACTION2/tools/data/jester/
bash extract_flow.sh
```

7.43.5 步骤 4: 编码视频

如果用户只想使用 RGB 帧训练，则该部分是 **可选项**。

用户可以运行以下命令进行视频编码。

```
cd $MMACTION2/tools/data/jester/
bash encode_videos.sh
```

7.43.6 步骤 5. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
cd $MMACTION2/tools/data/jester/
bash generate_{rawframes, videos}_filelist.sh
```

7.43.7 步骤 6. 检查文件夹结构

在完成所有 Jester 数据集准备流程后，用户可以获得对应的 RGB + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，Jester 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── jester
|   |   ├── jester_{train,val}_list_rawframes.txt
|   |   ├── jester_{train,val}_list_videos.txt
|   |   ├── annotations
|   |   ├── videos
|   |   |   ├── 1.mp4
|   |   |   ├── 2.mp4
|   |   |   ├──...
|   |   ├── rawframes
|   |   |   ├── 1
|   |   |   |   ├── 00001.jpg
|   |   |   |   ├── 00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_x_00001.jpg
|   |   |   |   ├── flow_x_00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_y_00001.jpg
```

(下页继续)

(续上页)

```
|   |   |   |   ├── flow_y_00002.jpg
|   |   |   |   ├── ...
|   |   |   ├── 2
|   |   |   ├── ...
```

关于对 jester 进行训练和验证，可以参考 基础教程。

7.44 JHMDB

7.44.1 简介

```
@inproceedings{Jhuang:ICCV:2013,
    title = {Towards understanding action recognition},
    author = {H. Jhuang and J. Gall and S. Zuffi and C. Schmid and M. J. Black},
    booktitle = {International Conf. on Computer Vision (ICCV)},
    month = Dec,
    pages = {3192-3199},
    year = {2013}
}
```

用户可参考该数据集的 官网，以获取数据集相关的基本信息。在数据集准备前，请确保命令行当前路径为 `$MMACTION2/tools/data/jhmdb/`。

7.44.2 下载和解压

用户可以从 这里 下载 RGB 帧，光流和真实标签文件。该数据由 MOC 代码库提供，参考自 act-detector。

用户在下载 `JHMDB.tar.gz` 文件后，需将其放置在 `$MMACTION2/tools/data/jhmdb/` 目录下，并使用以下指令进行解压：

```
tar -zxvf JHMDB.tar.gz
```

如果拥有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 中。

可以运行以下命令为 SSD 建立软链接。

```
## 执行这两行进行抽取 (假设 SSD 挂载在 "/mnt/SSD/")
mkdir /mnt/SSD/JHMDB/
ln -s /mnt/SSD/JHMDB/ ../../../data/jhmdb
```

7.44.3 检查文件夹结构

完成解压后，用户将得到 FlowBrox04 文件夹，Frames 文件夹和 JHMDB-GT.pkl 文件。

在整个 MMAction2 文件夹下，JHMDB 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── jhmdb
|   |   ├── FlowBrox04
|   |   |   ├── brush_hair
|   |   |   |   ├── April_09_brush_hair_u_nm_np1_ba_goo_0
|   |   |   |   |   ├── 00001.jpg
|   |   |   |   |   ├── 00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── 00039.jpg
|   |   |   |   |   ├── 00040.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── Trannydude___Brushing_SyntheticHair___OhNOES!__those_fukin_knots!_
↪brush_hair_u_nm_np1_fr_goo_2
|   |   |   ├── ...
|   |   |   ├── wave
|   |   |   |   ├── 21_wave_u_nm_np1_fr_goo_5
|   |   |   |   ├── ...
|   |   |   |   ├── Wie_man_winkt!!_wave_u_cm_np1_fr_med_0
|   |   ├── Frames
|   |   |   ├── brush_hair
|   |   |   |   ├── April_09_brush_hair_u_nm_np1_ba_goo_0
|   |   |   |   |   ├── 00001.png
|   |   |   |   |   ├── 00002.png
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── 00039.png
|   |   |   |   |   ├── 00040.png
|   |   |   |   ├── ...
|   |   |   |   ├── Trannydude___Brushing_SyntheticHair___OhNOES!__those_fukin_knots!_
↪brush_hair_u_nm_np1_fr_goo_2
|   |   |   ├── ...
|   |   |   ├── wave
|   |   |   |   ├── 21_wave_u_nm_np1_fr_goo_5
|   |   |   |   ├── ...
|   |   |   |   ├── Wie_man_winkt!!_wave_u_cm_np1_fr_med_0
|   |   ├── JHMDB-GT.pkl
```

(下页继续)

(续上页)

```
```

注意：`JHMDB-GT.pkl` 作为一个缓存文件，它包含 6 个项目：

1. `labels` (list)：21 个行为类别名称组成的列表
2. `gttubes` (dict)：每个视频对应的基准 tubes 组成的字典 **gttube** 是由标签索引和 tube 列表组成的字典 **tube** 是一个 `nframes` 行和 5 列的 numpy array，每一列的形式如 `<frame index> <x1> <y1> <x2> <y2>`
3. `nframes` (dict)：用以表示每个视频对应的帧数，如 `'walk/Panic_in_the_Streets_walk_u_cm_np1_ba_med_5': 16`
4. `train_videos` (list)：包含 `nsplits=1` 的元素，每一项都包含了训练视频的列表
5. `test_videos` (list)：包含 `nsplits=1` 的元素，每一项都包含了测试视频的列表
6. `resolution` (dict)：每个视频对应的分辨率（形如 (h,w)），如 `'pour/Bartender_School_Students_Practice_pour_u_cm_np1_fr_med_1': (240, 320)`

7.45 Kinetics-[400/600/700]

7.45.1 简介

```
@inproceedings{inproceedings,
  author = {Carreira, J. and Zisserman, Andrew},
  year = {2017},
  month = {07},
  pages = {4724-4733},
  title = {Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset},
  doi = {10.1109/CVPR.2017.502}
}
```

请参照 官方网站 以获取数据集基本信息。此脚本用于准备数据集 kinetics400，kinetics600，kinetics700。为准备 kinetics 数据集的不同版本，用户需将脚本中的 `${DATASET}` 赋值为数据集对应版本名称，可选项为 `kinetics400`，`kinetics600`，`kinetics700`。在开始之前，用户需确保当前目录为 `$MMACTION2/tools/data/${DATASET}/`。

注：由于部分 YouTube 链接失效，爬取的 Kinetics 数据集大小可能与原版不同。以下是我们所使用 Kinetics 数据集的大小：

7.45.2 1. 准备标注文件

首先，用户可以使用如下脚本从 Kinetics 数据集官网下载标注文件并进行预处理：

```
bash download_annotations.sh ${DATASET}
```

由于部分视频的 URL 不可用，当前官方标注中所含视频数量可能小于初始版本。所以 MMAction2 提供了另一种方式以获取初始版本标注作为参考。在这其中，Kinetics400 和 Kinetics600 的标注文件来自 官方爬虫，Kinetics700 的标注文件于 05/02/2021 下载自 网站。

```
bash download_backup_annotations.sh ${DATASET}
```

7.45.3 2. 准备视频

用户可以使用以下脚本准备视频，视频准备代码修改自 官方爬虫。注意这一步骤将花费较长时间。

```
bash download_videos.sh ${DATASET}
```

重要提示：如果在此之前已下载好 Kinetics 数据集的视频，还需使用重命名脚本来替换掉类名中的空格：

```
bash rename_classnames.sh ${DATASET}
```

为提升解码速度，用户可以使用以下脚本将原始视频缩放至更小的分辨率（利用稠密编码）：

```
python ../resize_videos.py ../../../data/${DATASET}/videos_train/ ../../../data/$
↪{DATASET}/videos_train_256p_dense_cache --dense --level 2
```

也可以从 Academic Torrents 中下载短边长度为 256 的 kinetics400 和 kinetics700，或从 Common Visual Data Foundation 维护的 cvdfoundation/kinetics-dataset 中下载 Kinetics400/Kinetics600/Kinetics-700-2020。

7.45.4 3. 提取 RGB 帧和光流

如果用户仅使用 video loader，则可以跳过本步。

在提取之前，请参考 安装教程 安装 denseflow。

如果用户有足够的 SSD 空间，那么建议将视频抽取为 RGB 帧以提升 I/O 性能。用户可以使用以下脚本为抽取得到的帧文件夹建立软连接：

```
## 执行以下脚本 (假设 SSD 被挂载在 "/mnt/SSD/")
mkdir /mnt/SSD/${DATASET}_extracted_train/
ln -s /mnt/SSD/${DATASET}_extracted_train/ ../../../data/${DATASET}/rawframes_train/
mkdir /mnt/SSD/${DATASET}_extracted_val/
ln -s /mnt/SSD/${DATASET}_extracted_val/ ../../../data/${DATASET}/rawframes_val/
```

如果用户只使用 RGB 帧（由于光流提取非常耗时），可以考虑执行以下脚本，仅用 denseflow 提取 RGB 帧：

```
bash extract_rgb_frames.sh ${DATASET}
```

如果用户未安装 denseflow，以下脚本可以使用 OpenCV 进行 RGB 帧的提取，但视频原分辨率大小会被保留：

```
bash extract_rgb_frames_opencv.sh ${DATASET}
```

如果同时需要 RGB 帧和光流，可使用如下脚本抽帧：

```
bash extract_frames.sh ${DATASET}
```

以上的命令生成短边长度为 256 的 RGB 帧和光流帧。如果用户需要生成短边长度为 320 的帧 (320p)，或是固定分辨率为 340 x 256 的帧，可改变参数 `--new-short 256` 为 `--new-short 320` 或 `--new-width 340 --new-height 256`。更多细节可以参考 数据准备。

7.45.5 4. 生成文件列表

用户可以使用以下两个脚本分别为视频和帧文件夹生成文件列表：

```
bash generate_videos_filelist.sh ${DATASET}
## 为帧文件夹生成文件列表
bash generate_rawframes_filelist.sh ${DATASET}
```

7.45.6 5. 目录结构

在完整完成 Kinetics 的数据处理后，将得到帧文件夹（RGB 帧和光流帧），视频以及标注文件。

在整个项目目录下（仅针对 Kinetics），最简目录结构如下所示：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── ${DATASET}
|   |   ├── ${DATASET}_train_list_videos.txt
|   |   ├── ${DATASET}_val_list_videos.txt
|   |   ├── annotations
|   |   ├── videos_train
|   |   ├── videos_val
|   |   |   ├── abseiling
|   |   |   |   ├── 0wR5jVB-WPk_000417_000427.mp4
|   |   |   |   ├── ...
```

(下页继续)

(续上页)

```
|   |   |   ├── ...
|   |   |   ├── wrapping_present
|   |   |   ├── ...
|   |   |   ├── zumba
|   |   ├── rawframes_train
|   |   ├── rawframes_val
```

关于 Kinetics 数据集上的训练与测试，请参照 基础教程。

7.46 Moments in Time

7.46.1 简介

```
@article{monfortmoments,
    title={Moments in Time Dataset: one million videos for event understanding},
    author={Monfort, Mathew and Andonian, Alex and Zhou, Bolei and Ramakrishnan,␣
↪Kandan and Bargal, Sarah Adel and Yan, Tom and Brown, Lisa and Fan, Quanfu and␣
↪Gutfruend, Dan and Vondrick, Carl and others},
    journal={IEEE Transactions on Pattern Analysis and Machine Intelligence},
    year={2019},
    issn={0162-8828},
    pages={1--8},
    numpages={8},
    doi={10.1109/TPAMI.2019.2901464},
}
```

用户可以参照数据集 官网，获取数据集相关的基本信息。在准备数据集前，请确保命令行当前路径为 `$MMACTION2/tools/data/mit/`。

7.46.2 步骤 1. 准备标注文件和视频文件

首先，用户需要访问官网，填写申请表来下载数据集。在得到下载链接后，用户可以使用 `bash preprocess_data.sh` 来准备标注文件和视频。请注意此脚本并没有下载标注和视频文件，用户需要根据脚本文件中的注释，提前下载好数据集，并放/软链接到合适的位置。

为加快视频解码速度，用户需要缩小原视频的尺寸，可使用以下命令获取密集编码版视频：

```
python ../resize_videos.py ../../../data/mit/videos/ ../../../data/mit/videos_256p_
↪dense_cache --dense --level 2
```

7.46.3 Step 2. 抽取帧和光流

如果用户只想使用视频加载训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果用户有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 上。用户可使用以下命令为 SSD 建立软链接。

```
## 执行这两行指令进行抽取（假设 SSD 挂载在 "/mnt/SSD/" 上）
mkdir /mnt/SSD/mit_extracted/
ln -s /mnt/SSD/mit_extracted/ ../../../data/mit/rawframes
```

如果用户需要抽取 RGB 帧（因为抽取光流的过程十分耗时），可以考虑运行以下命令使用 denseflow **只抽取 RGB 帧**。

```
bash extract_rgb_frames.sh
```

如果用户没有安装 denseflow，则可以运行以下命令使用 OpenCV 抽取 RGB 帧。然而，该方法只能抽取与原始视频分辨率相同的帧。

```
bash extract_rgb_frames_opencv.sh
```

如果用户想抽取 RGB 帧和光流，则可以运行以下脚本进行抽取。

```
bash extract_frames.sh
```

7.46.4 步骤 3. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
bash generate_{rawframes, videos}_filelist.sh
```

7.46.5 步骤 4. 检查目录结构

在完成 Moments in Time 数据集准备流程后，用户可以得到 Moments in Time 的 RGB 帧 + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，Moments in Time 的文件结构如下：

```
mmaction2
├── data
│   └── mit
│       ├── annotations
```

(下页继续)

(续上页)

```
|       |   ├── license.txt
|       |   ├── moments_categories.txt
|       |   ├── README.txt
|       |   ├── trainingSet.csv
|       |   └── validationSet.csv
|       ├── mit_train_rawframe_anno.txt
|       ├── mit_train_video_anno.txt
|       ├── mit_val_rawframe_anno.txt
|       ├── mit_val_video_anno.txt
|       ├── rawframes
|       |   ├── training
|       |   |   ├── adult+female+singing
|       |   |   |   ├── 0P3XG_vf91c_35
|       |   |   |   |   ├── flow_x_00001.jpg
|       |   |   |   |   ├── flow_x_00002.jpg
|       |   |   |   |   ├── ...
|       |   |   |   |   ├── flow_y_00001.jpg
|       |   |   |   |   ├── flow_y_00002.jpg
|       |   |   |   |   ├── ...
|       |   |   |   |   ├── img_00001.jpg
|       |   |   |   |   └── img_00002.jpg
|       |   |   |   └── yt-zxQfALnTdfc_56
|       |   |   |   |   ├── ...
|       |   |   └── yawning
|       |   |       ├── _8zmP1e-EjU_2
|       |   |       |   ├── ...
|       |   └── validation
|       |   |       ├── ...
|       └── videos
|           ├── training
|           |   ├── adult+female+singing
|           |   |   ├── 0P3XG_vf91c_35.mp4
|           |   |   ├── ...
|           |   |   └── yt-zxQfALnTdfc_56.mp4
|           |   └── yawning
|           |       ├── ...
|           └── validation
|           |   ├── ...
└── mmaction
└── ...
```

关于对 Moments in Times 进行训练和验证，可以参照 基础教程。

7.47 Multi-Moments in Time

7.47.1 简介

```
@misc{monfort2019multimoments,
    title={Multi-Moments in Time: Learning and Interpreting Models for Multi-Action Video Understanding},
    author={Mathew Monfort and Kandan Ramakrishnan and Alex Andonian and Barry A McNamara and Alex Lascelles, Bowen Pan, Quanfu Fan, Dan Gutfreund, Rogerio Feris, Aude Oliva},
    year={2019},
    eprint={1911.00232},
    archivePrefix={arXiv},
    primaryClass={cs.CV}
}
```

用户可以参照数据集 官网，获取数据集相关的基本信息。在准备数据集前，请确保命令行当前路径为 `$MMACTION2/tools/data/mmit/`。

7.47.2 步骤 1. Prepare Annotations and Videos

首先，用户需要访问官网，填写申请表来下载数据集。在得到下载链接后，用户可以使用 `bash preprocess_data.sh` 来准备标注文件和视频。请注意此脚本并没有下载标注和视频文件，用户需要根据脚本文件中的注释，提前下载好数据集，并放/软链接到合适的位置。

为加快视频解码速度，用户需要缩小原视频的尺寸，可使用以下命令获取密集编码版视频：

```
python ../resize_videos.py ../../../data/mmit/videos/ ../../../data/mmit/videos_256p_dense_cache --dense --level 2
```

7.47.3 Step 2. 抽取帧和光流

如果用户只想使用视频加载训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果用户有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 上。用户可使用以下命令为 SSD 建立软链接。

```
## 执行这两行指令进行抽取（假设 SSD 挂载在 "/mnt/SSD/" 上）
mkdir /mnt/SSD/mmit_extracted/
ln -s /mnt/SSD/mmit_extracted/ ../../../data/mmit/rawframes
```

如果用户需要抽取 RGB 帧（因为抽取光流的过程十分耗时），可以考虑运行以下命令使用 denseflow **只抽取 RGB 帧**。

```
bash extract_rgb_frames.sh
```

如果用户没有安装 denseflow，则可以运行以下命令使用 OpenCV 抽取 RGB 帧。然而，该方法只能抽取与原始视频分辨率相同的帧。

```
bash extract_rgb_frames_opencv.sh
```

如果用户想抽取 RGB 帧和光流，则可以运行以下脚本进行抽取。

```
bash extract_frames.sh
```

7.47.4 步骤 3. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
bash generate_rawframes_filelist.sh
bash generate_videos_filelist.sh
```

7.47.5 步骤 4. 检查目录结构

在完成 Multi-Moments in Time 数据集准备流程后，用户可以得到 Multi-Moments in Time 的 RGB 帧 + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，Multi-Moments in Time 的文件结构如下：

```
mmaction2/
└── data
    └── mmit
        ├── annotations
        │   ├── moments_categories.txt
        │   ├── trainingSet.txt
        │   └── validationSet.txt
        ├── mmit_train_rawframes.txt
        ├── mmit_train_videos.txt
        ├── mmit_val_rawframes.txt
        ├── mmit_val_videos.txt
        ├── rawframes
        │   ├── 0-3-6-2-9-1-2-6-14603629126_5
        │   │   ├── flow_x_00001.jpg
        │   │   ├── flow_x_00002.jpg
```

(下页继续)

(续上页)

```
        |   |   ├── ...
        |   |   ├── flow_y_00001.jpg
        |   |   ├── flow_y_00002.jpg
        |   |   ├── ...
        |   |   ├── img_00001.jpg
        |   |   └── img_00002.jpg
        |   |   ├── ...
        |   └── yt-zxQfALnTdfc_56
        |   |   ├── ...
        |   └── ...

        └── videos
            └── adult+female+singing
                ├── 0-3-6-2-9-1-2-6-14603629126_5.mp4
                └── yt-zxQfALnTdfc_56.mp4
            └── ...
```

关于对 Multi-Moments in Time 进行训练和验证，可以参照 基础教程。

7.48 OmniSource

7.48.1 简介

```
@article{duan2020omni,
  title={Omni-sourced Webly-supervised Learning for Video Recognition},
  author={Duan, Haodong and Zhao, Yue and Xiong, Yuanjun and Liu, Wentao and Lin,␣
↪Dahua},
  journal={arXiv preprint arXiv:2003.13042},
  year={2020}
}
```

MMAction2 中发布了 OmniSource 网络数据集的一个子集 (来自论文 Omni-sourced Webly-supervised Learning for Video Recognition)。OmniSource 数据集中所有类别均来自 Kinetics-400。MMAction2 所提供的子集包含属于 Mini-Kinetics 数据集 200 类动作的网络数据 (Mini-inetics 数据集由论文 Rethinking Spatiotemporal Feature Learning: Speed-Accuracy Trade-offs in Video Classification 提出)。

MMAction2 提供所有数据源中属于 Mini-Kinetics 200 类动作的数据，这些数据源包含：Kinetics 数据集，Kinetics 原始数据集（未经裁剪的长视频），来自 Google 和 Instagram 的网络图片，来自 Instagram 的网络视频。为获取这一数据集，用户需先填写 数据申请表。在接收到申请后，下载链接将被发送至用户邮箱。由于发布的数据集均为爬取所得的原始数据，数据集较大，下载需要一定时间。下表中提供了 OmniSource 数据集各个分量的统计信息。

MMAction2 所发布的 OmniSource 数据集目录结构如下所示：

```
OmniSource/
├── annotations
│   ├── googleimage_200
│   │   ├── googleimage_200.txt                          从 Google 爬取到的所有图片列表
│   │   ├── tsn_8seg_googleimage_200_duplicate.txt       从 Google 爬取到的，疑似与 k200-val␣
↪中样本重复的正样本列表
│   │   ├── tsn_8seg_googleimage_200.txt                 从 Google 爬取到的，经过 teacher 模
型过滤的正样本列表
│   │   └── tsn_8seg_googleimage_200_wodup.txt           从 Google 爬取到的，经过 teacher 模
型过滤及去重的正样本列表
│   ├── insimage_200
│   │   ├── insimage_200.txt
│   │   ├── tsn_8seg_insimage_200_duplicate.txt
│   │   ├── tsn_8seg_insimage_200.txt
│   │   └── tsn_8seg_insimage_200_wodup.txt
│   ├── insvideo_200
│   │   ├── insvideo_200.txt
│   │   ├── slowonly_8x8_insvideo_200_duplicate.txt
│   │   ├── slowonly_8x8_insvideo_200.txt
│   │   └── slowonly_8x8_insvideo_200_wodup.txt
│   ├── k200_actions.txt                                 MiniKinetics 中 200 类动作的名称
│   ├── K400_to_MiniKinetics_classidx_mapping.json       Kinetics 中的类索引至␣
↪MiniKinetics 中的类索引的映射
│   ├── kinetics_200
│   │   ├── k200_train.txt
│   │   └── k200_val.txt
│   └── kinetics_raw_200
│       └── slowonly_8x8_kinetics_raw_200.json           经 teacher 模型过滤后的 Kinetics 原
始视频片段
├── googleimage_200                                      共 10 卷
│   ├── vol_0.tar
│   ├── ...
│   └── vol_9.tar
├── insimage_200                                         共 10 卷
│   ├── vol_0.tar
│   ├── ...
│   └── vol_9.tar
├── insvideo_200                                         共 20 卷
│   ├── vol_00.tar
│   ├── ...
│   └── vol_19.tar
├── kinetics_200_train
│   └── kinetics_200_train.tar
├── kinetics_200_val
```

(下页继续)

(续上页)

```
|   └── kinetics_200_val.tar
└── kinetics_raw_200_train                          共 16 卷
    ├── vol_0.tar
    ├── ...
    └── vol_15.tar
```

7.48.2 数据准备

用户需要首先完成数据下载，对于 kinetics_200 和三个网络数据集 googleimage_200, insimage_200, insvideo_200，用户仅需解压各压缩卷并将其合并至一处。

对于 Kinetics 原始视频，由于直接读取长视频非常耗时，用户需要先将其分割为小段。MMAction2 提供了名为 trim_raw_video.py 的脚本，用于将长视频分割至 10 秒的小段（分割完成后删除长视频）。用户可利用这一脚本分割长视频。

所有数据应位于 data/OmniSource/ 目录下。完成数据准备后，data/OmniSource/ 目录的结构应如下所示（为简洁，省去了训练及测试时未使用的文件）：

```
data/OmniSource/
├── annotations
|   ├── googleimage_200
|   |   └── tsn_8seg_googleimage_200_wodup.txt    Positive file list of images␣
↪crawled from Google, filtered by the teacher model, after de-duplication.
|   ├── insimage_200
|   |   └── tsn_8seg_insimage_200_wodup.txt
|   ├── insvideo_200
|   |   └── slowonly_8x8_insvideo_200_wodup.txt
|   ├── kinetics_200
|   |   ├── k200_train.txt
|   |   └── k200_val.txt
|   ├── kinetics_raw_200
|   |   └── slowonly_8x8_kinetics_raw_200.json    Kinetics Raw Clips filtered by the␣
↪teacher model.
|   └── webimage_200
|       └── tsn_8seg_webimage_200_wodup.txt       The union of `tsn_8seg_googleimage_
↪200_wodup.txt` and `tsn_8seg_insimage_200_wodup.txt`
├── googleimage_200
|   ├── 000
|   |   ├── 00
|   |   |   ├── 000001.jpg
|   |   |   ├── ...
|   |   |   └── 000901.jpg
|   |   ├── ...
```

(下页继续)

(续上页)

```
|   |   ├── 19
|   ├── ...
|   └── 199
├── insimage_200
|   ├── 000
|   |   ├── abseil
|   |   |   ├── 1J9tKWCNgV_0.jpg
|   |   |   ├── ...
|   |   |   └── 1J9tKWCNgV_0.jpg
|   |   ├── abseiling
|   ├── ...
|   └── 199
├── insvideo_200
|   ├── 000
|   |   ├── abseil
|   |   |   ├── B00arxogubl.mp4
|   |   |   ├── ...
|   |   |   └── BzYsP0HIvbt.mp4
|   |   ├── abseiling
|   ├── ...
|   └── 199
├── kinetics_200_train
|   ├── 0074cdXclLU.mp4
|   ├── ...
|   ├── zzzlyL61Fyo.mp4
├── kinetics_200_val
|   ├── 01fAWEHzudA.mp4
|   ├── ...
|   ├── zymA_6jZIz4.mp4
└── kinetics_raw_200_train
|   ├── pref_
|   |   ├── ___dTOdxzXY
|   |   |   ├── part_0.mp4
|   |   |   ├── ...
|   |   |   ├── part_6.mp4
|   |   ├── ...
|   |   └── _zygwGDE2EM
|   ├── ...
|   └── prefZ
```

7.49 骨架数据集

```
@misc{duan2021revisiting,
      title={Revisiting Skeleton-based Action Recognition},
      author={Haodong Duan and Yue Zhao and Kai Chen and Dian Shao and Dahua Lin and
↪Bo Dai},
      year={2021},
      eprint={2104.13586},
      archivePrefix={arXiv},
      primaryClass={cs.CV}
}
```

7.49.1 简介

MMAction2 发布 Revisiting Skeleton-based Action Recognition 论文中所使用的骨架标注。默认使用 Faster-RCNN 作为人体检测器，使用 HRNet-w32 作为单人姿态估计模型。对于 FineGYM 数据集，MMAction2 使用的是运动员的真实框标注，而非检测器所出的框。目前，MMAction2 已发布 FineGYM 和 NTURGB-D Xsub 部分的骨架标注，其他数据集的标注也将很快发布。

7.49.2 标注文件

目前，MMAction2 支持 HMDB51, UCF101, FineGYM 和 NTURGB+D 数据集。对于 FineGYM 数据集，用户可以使用以下脚本下载标注文件。

```
bash download_annotations.sh ${DATASET}
```

由于 NTURGB+D 数据集的 使用条例，MMAction2 并未直接发布实验中所使用的标注文件。因此，这里提供生成 NTURGB+D 数据集中视频的姿态标注文件，这将生成一个 dict 数据并将其保存为一个 pickle 文件。用户可以生成一个 list 用以包含对应视频的 dict 数据，并将其保存为一个 pickle 文件。之后，用户可以获得 `ntu60_xsub_train.pkl`, `ntu60_xsub_val.pkl`, `ntu120_xsub_train.pkl`, `ntu120_xsub_val.pkl` 文件用于训练。

对于无法进行姿态提取的用户，这里提供了上述流程的输出结果，分别对应 NTURGB-D 数据集的 4 个部分：

- ntu60_xsub_train: https://download.openmmlab.com/mmaction/posec3d/ntu60_xsub_train.pkl
- ntu60_xsub_val: https://download.openmmlab.com/mmaction/posec3d/ntu60_xsub_val.pkl
- ntu120_xsub_train: https://download.openmmlab.com/mmaction/posec3d/ntu120_xsub_train.pkl
- ntu120_xsub_val: https://download.openmmlab.com/mmaction/posec3d/ntu120_xsub_val.pkl
- hmdb51: https://download.openmmlab.com/mmaction/posec3d/hmdb51.pkl
- ucf101: https://download.openmmlab.com/mmaction/posec3d/ucf101.pkl

若想生成单个视频的 2D 姿态标注文件，首先，用户需要由源码安装 mmdetection 和 mmpose。之后，用户需要在 `ntu_pose_extraction.py` 中指定 `mmdet_root` 和 `mmpose_root` 变量。最后，用户可使用以下脚本进行 NTURGB+D 视频的姿态提取：

```
python ntu_pose_extraction.py S001C001P001R001A001_rgb.avi S001C001P001R001A001.pkl
```

在用户获得数据集某部分所有视频的姿态标注文件（如 `ntu60_xsub_val`）后，可以将其集合成一个 list 数据并保存为 `ntu60_xsub_val.pkl`。用户可用这些大型 pickle 文件进行训练和测试。

7.49.3 PoseC3D 的标注文件格式

这里简单介绍 PoseC3D 的标注文件格式。以 `gym_train.pkl` 为例：`gym_train.pkl` 存储一个长度为 20484 的 list，list 的每一项为单个视频的骨架标注 dict。每个 dict 的内容如下：

- keypoint：关键点坐标，大小为 N（## 人数）x T（时序长度）x K（# 关键点, 这里为 17）x 2 （x，y 坐标）的 numpy array 数据类型
- keypoint_score：关键点的置信分数，大小为 N（## 人数）x T（时序长度）x K（# 关键点, 这里为 17）的 numpy array 数据类型
- frame_dir: 对应视频名
- label: 动作类别
- img_shape: 每一帧图像的大小
- original_shape: 同 `img_shape`
- total_frames: 视频时序长度

如用户想使用自己的数据集训练 PoseC3D，可以参考 Custom Dataset Training。

7.49.4 可视化

为了可视化骨架数据，用户需要准备 RGB 的视频。详情可参考 visualize_heatmap_volume。这里提供一些 NTU-60 和 FineGYM 上的例子

7.49.5 如何将 NTU RGB+D 原始数据转化为 MMAction2 格式（转换好的标注文件目前仅适用于 GCN 模型）

这里介绍如何将 NTU RGB+D 原始数据转化为 MMAction2 格式。首先，需要从 https://github.com/shahroudy/NTURGB-D 下载原始 NTU-RGBD 60 和 NTU-RGBD 120 数据集的原始骨架数据。

对于 NTU-RGBD 60 数据集，可使用以下脚本

```
python gen_ntu_rgbd_raw.py --data-path your_raw_nturgbd60_skeleton_path --ignored-
↪sample-path NTU_RGBD_samples_with_missing_skeletons.txt --out-folder your_nturgbd60_
↪output_path --task ntu60
```

对于 NTU-RGBD 120 数据集，可使用以下脚本

```
python gen_ntu_rgbd_raw.py --data-path your_raw_nturgbd120_skeleton_path --ignored-
↪sample-path NTU_RGBD120_samples_with_missing_skeletons.txt --out-folder your_
↪nturgbd120_output_path --task ntu120
```

7.49.6 转换其他第三方项目的骨骼标注

MMAction2 提供脚本以将其他第三方项目的骨骼标注转至 MMAction2 格式，如：

- BABEL: `babel2mma2.py`

待办项：

- [x] FineGYM
- [x] NTU60_XSub
- [x] NTU120_XSub
- [x] NTU60_XView
- [x] NTU120_XSet
- [x] UCF101
- [x] HMDB51
- [ ] Kinetics

7.50 Something-Something V1

7.50.1 简介

```
@misc{goyal2017something,
      title={The "something something" video database for learning and evaluating␣
↪visual common sense},
      author={Raghav Goyal and Samira Ebrahimi Kahou and Vincent Michalski and Joanna␣
↪Materzyńska and Susanne Westphal and Heuna Kim and Valentin Haenel and Ingo Fruend␣
↪and Peter Yianilos and Moritz Mueller-Freitag and Florian Hoppe and Christian␣
↪Thurau and Ingo Bax and Roland Memisevic},
      year={2017},
```

(下页继续)

(续上页)

```
      eprint={1706.04261},
      archivePrefix={arXiv},
      primaryClass={cs.CV}
}
```

用户可参考该数据集的 官网，以获取数据集相关的基本信息。在数据集准备前，请确保命令行当前路径为 `$MMACTION2/tools/data/sthv1/`。

7.50.2 步骤 1. 下载标注文件

由于 Something-Something V1 的官方网站已经失效，用户需要通过第三方源下载原始数据集。下载好的标注文件需要放在 `$MMACTION2/data/sthv1/annotations` 文件夹下。

7.50.3 步骤 2. 准备 RGB 帧

官方数据集并未提供原始视频文件，只提供了对原视频文件进行抽取得到的 RGB 帧，用户可在第三方源直接下载视频帧。

将下载好的压缩文件放在 `$MMACTION2/data/sthv1/` 文件夹下，并使用以下脚本进行解压。

```
cd $MMACTION2/data/sthv1/
cat 20bn-something-something-v1-?? | tar zx
cd $MMACTION2/tools/data/sthv1/
```

如果用户只想使用 RGB 帧，则可以跳过中间步骤至步骤 5 以直接生成视频帧的文件列表。由于官网的 JPG 文件名形如 “%05d.jpg”（比如，” 00001.jpg”），需要在配置文件的 `data.train`, `data.val` 和 `data.test` 处添加 `"filename_tmpl='{:05}.jpg'"` 代码，以修改文件名模板。

```
data = dict(
    videos_per_gpu=16,
    workers_per_gpu=2,
    train=dict(
        type=dataset_type,
        ann_file=ann_file_train,
        data_prefix=data_root,
        filename_tmpl='{:05}.jpg',
        pipeline=train_pipeline),
    val=dict(
        type=dataset_type,
        ann_file=ann_file_val,
        data_prefix=data_root_val,
        filename_tmpl='{:05}.jpg',
```

(下页继续)

(续上页)

```
        pipeline=val_pipeline),
    test=dict(
        type=dataset_type,
        ann_file=ann_file_test,
        data_prefix=data_root_val,
        filename_tmpl='{:05}.jpg',
        pipeline=test_pipeline))
```

7.50.4 步骤 3. 抽取光流

如果用户只想使用原 RGB 帧加载训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果拥有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 中。

可以运行以下命令为 SSD 建立软链接。

```
## 执行这两行进行抽取（假设 SSD 挂载在 "/mnt/SSD/"）
mkdir /mnt/SSD/sthv1_extracted/
ln -s /mnt/SSD/sthv1_extracted/ ../../../data/sthv1/rawframes
```

如果想抽取光流，则可以运行以下脚本从 RGB 帧中抽取出光流。

```
cd $MMACTION2/tools/data/sthv1/
bash extract_flow.sh
```

7.50.5 步骤 4: 编码视频

如果用户只想使用 RGB 帧加载训练，则该部分是 **可选项**。

用户可以运行以下命令进行视频编码。

```
cd $MMACTION2/tools/data/sthv1/
bash encode_videos.sh
```

7.50.6 步骤 5. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
cd $MMACTION2/tools/data/sthv1/
bash generate_{rawframes, videos}_filelist.sh
```

7.50.7 步骤 6. 检查文件夹结构

在完成所有 Something-Something V1 数据集准备流程后，用户可以获得对应的 RGB + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，Something-Something V1 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── sthv1
|   |   ├── sthv1_{train,val}_list_rawframes.txt
|   |   ├── sthv1_{train,val}_list_videos.txt
|   |   ├── annotations
|   |   ├── videos
|   |   |   ├── 1.mp4
|   |   |   ├── 2.mp4
|   |   |   ├──...
|   |   ├── rawframes
|   |   |   ├── 1
|   |   |   |   ├── 00001.jpg
|   |   |   |   ├── 00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_x_00001.jpg
|   |   |   |   ├── flow_x_00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_y_00001.jpg
|   |   |   |   ├── flow_y_00002.jpg
|   |   |   |   ├── ...
|   |   |   ├── 2
|   |   |   ├── ...
```

关于对 Something-Something V1 进行训练和验证，可以参考 基础教程。

7.51 Something-Something V2

7.51.1 简介

```
@misc{goyal2017something,
      title={The "something something" video database for learning and evaluating visual common sense},
      author={Raghav Goyal and Samira Ebrahimi Kahou and Vincent Michalski and Joanna Materzyńska and Susanne Westphal and Heuna Kim and Valentin Haenel and Ingo Fruend and Peter Yianilos and Moritz Mueller-Freitag and Florian Hoppe and Christian Thurau and Ingo Bax and Roland Memisevic},
      year={2017},
      eprint={1706.04261},
      archivePrefix={arXiv},
      primaryClass={cs.CV}
}
```

用户可参考该数据集的 官网，以获取数据集相关的基本信息。在数据集准备前，请确保命令行当前路径为 `$MMACTION2/tools/data/sthv2/`。

7.51.2 步骤 1. 下载标注文件

首先，用户需要在 官网 完成注册，才能下载标注文件。下载好的标注文件需要放在 `$MMACTION2/data/sthv2/annotations` 文件夹下。

```
cd $MMACTION2/data/sthv2/annotations
unzip 20bn-something-something-download-package-labels.zip
find ./labels -name "*.json" -exec sh -c 'cp "$1" "something-something-v2-$(basename $1)"' _ {} \;
```

7.51.3 步骤 2. 准备视频

之后，用户可将下载好的压缩文件放在 `$MMACTION2/data/sthv2/` 文件夹下，并且使用以下指令进行解压。

```
cd $MMACTION2/data/sthv2/
cat 20bn-something-something-v2-?? | tar zx
cd $MMACTION2/tools/data/sthv2/
```

7.51.4 步骤 3. 抽取 RGB 帧和光流

如果用户只想使用视频加载训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果拥有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 中。

可以运行以下命令为 SSD 建立软链接。

```
## 执行这两行进行抽取（假设 SSD 挂载在 "/mnt/SSD/"）
mkdir /mnt/SSD/sthv2_extracted/
ln -s /mnt/SSD/sthv2_extracted/ ../../../data/sthv2/rawframes
```

如果用户需要抽取 RGB 帧（因为抽取光流的过程十分耗时），可以考虑运行以下命令使用 denseflow **只抽取 RGB 帧**。

```
cd $MMACTION2/tools/data/sthv2/
bash extract_rgb_frames.sh
```

如果用户没有安装 denseflow，则可以运行以下命令使用 OpenCV 抽取 RGB 帧。然而，该方法只能抽取与原始视频分辨率相同的帧。

```
cd $MMACTION2/tools/data/sthv2/
bash extract_rgb_frames_opencv.sh
```

如果用户想抽取 RGB 帧和光流，则可以运行以下脚本进行抽取。

```
cd $MMACTION2/tools/data/sthv2/
bash extract_frames.sh
```

7.51.5 步骤 4. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
cd $MMACTION2/tools/data/sthv2/
bash generate_{rawframes, videos}_filelist.sh
```

7.51.6 步骤 5. 检查文件夹结构

在完成所有 Something-Something V2 数据集准备流程后，用户可以获得对应的 RGB + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，Something-Something V2 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── sthv2
|   |   ├── sthv2_{train,val}_list_rawframes.txt
|   |   ├── sthv2_{train,val}_list_videos.txt
|   |   ├── annotations
|   |   ├── videos
|   |   |   ├── 1.mp4
|   |   |   ├── 2.mp4
|   |   |   ├──...
|   |   ├── rawframes
|   |   |   ├── 1
|   |   |   |   ├── img_00001.jpg
|   |   |   |   ├── img_00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_x_00001.jpg
|   |   |   |   ├── flow_x_00002.jpg
|   |   |   |   ├── ...
|   |   |   |   ├── flow_y_00001.jpg
|   |   |   |   ├── flow_y_00002.jpg
|   |   |   |   ├── ...
|   |   |   ├── 2
|   |   |   ├── ...
```

关于对 Something-Something V2 进行训练和验证，可以参考 基础教程。

7.52 THUMOS'14

7.52.1 简介

```
@misc{THUMOS14,
    author = {Jiang, Y.-G. and Liu, J. and Roshan Zamir, A. and Toderici, G. and
↪Laptev,
    I. and Shah, M. and Sukthankar, R.},
    title = {{THUMOS} Challenge: Action Recognition with a Large
    Number of Classes},
    howpublished = "\url{http://crcv.ucf.edu/THUMOS14/}",
    Year = {2014}
}
```

用户可以参照数据集 官网，获取数据集相关的基本信息。在准备数据集前，请确保命令行当前路径为 `$MMACTION2/tools/data/thumos14/`。

7.52.2 步骤 1. 下载标注文件

首先，用户可使用以下命令下载标注文件。

```
cd $MMACTION2/tools/data/thumos14/
bash download_annotations.sh
```

7.52.3 步骤 2. 下载视频

之后，用户可使用以下指令下载视频

```
cd $MMACTION2/tools/data/thumos14/
bash download_videos.sh
```

7.52.4 步骤 3. 抽取帧和光流

如果用户只想使用视频加载训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果用户有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 上。用户可使用以下命令为 SSD 建立软链接。

```
## 执行这两行指令进行抽取（假设 SSD 挂载在 "/mnt/SSD/" 上）
mkdir /mnt/SSD/thumos14_extracted/
ln -s /mnt/SSD/thumos14_extracted/ ../data/thumos14/rawframes/
```

如果用户需要抽取 RGB 帧（因为抽取光流的过程十分耗时），可以考虑运行以下命令使用 denseflow **只抽取 RGB 帧**。

```
cd $MMACTION2/tools/data/thumos14/
bash extract_rgb_frames.sh
```

如果用户没有安装 denseflow，则可以运行以下命令使用 OpenCV 抽取 RGB 帧。然而，该方法只能抽取与原始视频分辨率相同的帧。

```
cd $MMACTION2/tools/data/thumos14/
bash extract_rgb_frames_opencv.sh
```

如果用户想抽取 RGB 帧和光流，则可以运行以下脚本进行抽取。

```
cd $MMACTION2/tools/data/thumos14/
bash extract_frames.sh tvl1
```

7.52.5 步骤 4. 生成文件列表

如果用户不使用 SSN 模型，则该部分是 **可选项**。

可使用运行以下脚本下载预先计算的候选标签。

```
cd $MMACTION2/tools/data/thumos14/
bash fetch_tag_proposals.sh
```

7.52.6 步骤 5. 去规范化候选文件

如果用户不使用 SSN 模型，则该部分是 **可选项**。

可运行以下脚本，来根据本地原始帧的实际数量，去规范化预先计算的候选标签。

```
cd $MMACTION2/tools/data/thumos14/
bash denormalize_proposal_file.sh
```

7.52.7 步骤 6. 检查目录结构

在完成 THUMOS’14 数据集准备流程后，用户可以得到 THUMOS’14 的 RGB 帧 + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，THUMOS’14 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── thumos14
|   |   ├── proposals
|   |   |   ├── thumos14_tag_val_normalized_proposal_list.txt
|   |   |   ├── thumos14_tag_test_normalized_proposal_list.txt
|   |   ├── annotations_val
|   |   ├── annotations_test
|   |   ├── videos
|   |   |   ├── val
|   |   |   |   ├── video_validation_0000001.mp4
|   |   |   |   ├── ...
|   |   |   ├── test
|   |   |   |   ├── video_test_0000001.mp4
|   |   |   |   ├── ...
|   |   ├── rawframes
|   |   |   ├── val
|   |   |   |   ├── video_validation_0000001
|   |   |   |   |   ├── img_00001.jpg
|   |   |   |   |   ├── img_00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── flow_x_00001.jpg
|   |   |   |   |   ├── flow_x_00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── flow_y_00001.jpg
|   |   |   |   |   ├── flow_y_00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   ├── ...
|   |   |   ├── test
|   |   |   |   ├── video_test_0000001
```

关于对 THUMOS’14 进行训练和验证，可以参照 基础教程。

7.53 UCF-101

7.53.1 简介

```
@article{Soomro2012UCF101AD,
  title={UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild},
  author={K. Soomro and A. Zamir and M. Shah},
  journal={ArXiv},
  year={2012},
  volume={abs/1212.0402}
}
```

用户可参考该数据集的 官网，以获取数据集相关的基本信息。在数据集准备前，请确保命令行当前路径为 `$MMACTION2/tools/data/ucf101/`。

7.53.2 步骤 1. 下载标注文件

首先，用户可运行以下脚本下载标注文件。

```
bash download_annotations.sh
```

7.53.3 步骤 2. 准备视频文件

之后，用户可运行以下脚本准备视频文件。

```
bash download_videos.sh
```

用户可使用以下脚本，对原视频进行裁剪，得到密集编码且更小尺寸的视频。

```
python ../resize_videos.py ../../../data/ucf101/videos/ ../../../data/ucf101/videos_
↪256p_dense_cache --dense --level 2 --ext avi
```

7.53.4 步骤 3. 抽取视频帧和光流

如果用户只想使用视频加载训练，则该部分是 **可选项**。

在抽取视频帧和光流之前，请参考 安装指南 安装 denseflow。

如果拥有大量的 SSD 存储空间，则推荐将抽取的帧存储至 I/O 性能更优秀的 SSD 中。所抽取的视频帧和光流约占据 100 GB 的存储空间。

可以运行以下命令为 SSD 建立软链接。

```
## 执行这两行进行抽取（假设 SSD 挂载在 "/mnt/SSD/"）
mkdir /mnt/SSD/ucf101_extracted/
ln -s /mnt/SSD/ucf101_extracted/ ../../../data/ucf101/rawframes
```

如果用户需要抽取 RGB 帧（因为抽取光流的过程十分耗时），可以考虑运行以下命令使用 denseflow **只抽取 RGB 帧**。

```
bash extract_rgb_frames.sh
```

如果用户没有安装 denseflow，则可以运行以下命令使用 OpenCV 抽取 RGB 帧。然而，该方法只能抽取与原始视频分辨率相同的帧。

```
bash extract_rgb_frames_opencv.sh
```

如果用户想抽取 RGB 帧和光流，则可以运行以下脚本使用“tvl1”算法进行抽取。

```
bash extract_frames.sh
```

7.53.5 步骤 4. 生成文件列表

用户可以通过运行以下命令生成帧和视频格式的文件列表。

```
bash generate_videos_filelist.sh
bash generate_rawframes_filelist.sh
```

7.53.6 步骤 5. 检查文件夹结构

在完成所有 UCF-101 数据集准备流程后，用户可以获得对应的 RGB + 光流文件，视频文件以及标注文件。

在整个 MMAction2 文件夹下，UCF-101 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── ucf101
|   |   ├── ucf101_{train,val}_split_{1,2,3}_rawframes.txt
|   |   ├── ucf101_{train,val}_split_{1,2,3}_videos.txt
|   |   ├── annotations
|   |   ├── videos
|   |   |   ├── ApplyEyeMakeup
|   |   |   |   ├── v_ApplyEyeMakeup_g01_c01.avi
```

(下页继续)

(续上页)

```
|   |   |   ├── YoYo
|   |   |   |   ├── v_YoYo_g25_c05.avi
|   |   ├── rawframes
|   |   |   ├── ApplyEyeMakeup
|   |   |   |   ├── v_ApplyEyeMakeup_g01_c01
|   |   |   |   |   ├── img_00001.jpg
|   |   |   |   |   ├── img_00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── flow_x_00001.jpg
|   |   |   |   |   ├── flow_x_00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── flow_y_00001.jpg
|   |   |   |   |   ├── flow_y_00002.jpg
|   |   |   ├── ...
|   |   |   ├── YoYo
|   |   |   |   ├── v_YoYo_g01_c01
|   |   |   |   ├── ...
|   |   |   |   ├── v_YoYo_g25_c05
```

关于对 UCF-101 进行训练和验证，可以参考 基础教程。

7.54 UCF101-24

7.54.1 简介

```
@article{Soomro2012UCF101AD,
  title={UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild},
  author={K. Soomro and A. Zamir and M. Shah},
  journal={ArXiv},
  year={2012},
  volume={abs/1212.0402}
}
```

用户可参考该数据集的 官网，以获取数据集相关的基本信息。在数据集准备前，请确保命令行当前路径为 `$MMACTION2/tools/data/ucf101_24/`。

7.54.2 下载和解压

用户可以从 这里 下载 RGB 帧，光流和标注文件。该数据由 MOC 代码库提供，参考自 act-detector 和 corrected-UCF101-Annots。

注意：UCF101-24 的标注文件来自于 这里，该标注文件相对于其他标注文件更加准确。

用户在下载 `UCF101_v2.tar.gz` 文件后，需将其放置在 `$MMACTION2/tools/data/ucf101_24/` 目录下，并使用以下指令进行解压：

```
tar -zxvf UCF101_v2.tar.gz
```

7.54.3 检查文件夹结构

经过解压后，用户将得到 `rgb-images` 文件夹，`brox-images` 文件夹和 `UCF101v2-GT.pkl` 文件。

在整个 MMAction2 文件夹下，UCF101_24 的文件结构如下：

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
|   ├── ucf101_24
|   |   ├── brox-images
|   |   |   ├── Basketball
|   |   |   |   ├── v_Basketball_g01_c01
|   |   |   |   |   ├── 00001.jpg
|   |   |   |   |   ├── 00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── 00140.jpg
|   |   |   |   |   ├── 00141.jpg
|   |   |   ├── ...
|   |   |   ├── WalkingWithDog
|   |   |   |   ├── v_WalkingWithDog_g01_c01
|   |   |   |   ├── ...
|   |   |   |   ├── v_WalkingWithDog_g25_c04
|   |   ├── rgb-images
|   |   |   ├── Basketball
|   |   |   |   ├── v_Basketball_g01_c01
|   |   |   |   |   ├── 00001.jpg
|   |   |   |   |   ├── 00002.jpg
|   |   |   |   |   ├── ...
|   |   |   |   |   ├── 00140.jpg
|   |   |   |   |   ├── 00141.jpg
```

(下页继续)

(续上页)

```
|   |   |   ├── ...
|   |   |   ├── WalkingWithDog
|   |   |   |   ├── v_WalkingWithDog_g01_c01
|   |   |   |   ├── ...
|   |   |   |   ├── v_WalkingWithDog_g25_c04
|   |   ├── UCF101v2-GT.pkl
```

注意：`UCF101v2-GT.pkl` 作为一个缓存文件，它包含 6 个项目：

1. `labels` (list)：24 个行为类别名称组成的列表
2. `gttubes` (dict)：每个视频对应的基准 tubes 组成的字典 **gttube** 是由标签索引和 tube 列表组成的字典 **tube** 是一个 `nframes` 行和 5 列的 numpy array，每一列的形式如 `<frame index> <x1> <y1> <x2> <y2>`
3. `nframes` (dict)：用以表示每个视频对应的帧数，如 `'HorseRiding/v_HorseRiding_g05_c02': 151`
4. `train_videos` (list)：包含 `nsplits=1` 的元素，每一项都包含了训练视频的列表
5. `test_videos` (list)：包含 `nsplits=1` 的元素，每一项都包含了测试视频的列表
6. `resolution` (dict)：每个视频对应的分辨率（形如 (h,w)），如 `'FloorGymnastics/v_FloorGymnastics_g09_c03': (240, 320)`

CHAPTER 8

总览

- 模型权重文件数量: 220
- 配置文件数量: 196
- 论文数量: 27
  - ALGORITHM: 23
  - BACKBONE: 1
  - DATASET: 2
  - OTHERS: 1

有关受支持的数据集，可参见数据集总览。

8.1 时空动作检测模型

- 模型权重文件数量: 22
- 配置文件数量: 20
- 论文数量: 5
  - [ALGORITHM] Actor-Centric Relation Network (->)
  - [ALGORITHM] Long-Term Feature Banks for Detailed Video Understanding (->)
  - [ALGORITHM] Omni-Sourced Webly-Supervised Learning for Video Recognition (->)

  - [ALGORITHM] Slowfast Networks for Video Recognition (->)
  - [DATASET] Ava: A Video Dataset of Spatio-Temporally Localized Atomic Visual Actions (-> ->)

8.2 时序动作检测模型

- 模型权重文件数量: 7
- 配置文件数量: 3
- 论文数量: 4
  - [ALGORITHM] Bmn: Boundary-Matching Network for Temporal Action Proposal Generation (->)
  - [ALGORITHM] Bsn: Boundary Sensitive Network for Temporal Action Proposal Generation (->)
  - [ALGORITHM] Temporal Action Detection With Structured Segment Networks (->)
  - [DATASET] Cuhk & Ethz & Siat Submission to Activitynet Challenge 2017 (->)

8.3 动作识别模型

- 模型权重文件数量: 175
- 配置文件数量: 157
- 论文数量: 17
  - [ALGORITHM] A Closer Look at Spatiotemporal Convolutions for Action Recognition (->)
  - [ALGORITHM] Audiovisual Slowfast Networks for Video Recognition (->)
  - [ALGORITHM] Is Space-Time Attention All You Need for Video Understanding? (->)
  - [ALGORITHM] Learning Spatiotemporal Features With 3d Convolutional Networks (->)
  - [ALGORITHM] Omni-Sourced Webly-Supervised Learning for Video Recognition (->)
  - [ALGORITHM] Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset (->)
  - [ALGORITHM] Slowfast Networks for Video Recognition (-> ->)
  - [ALGORITHM] Tam: Temporal Adaptive Module for Video Recognition (->)
  - [ALGORITHM] Temporal Interlacing Network (->)
  - [ALGORITHM] Temporal Pyramid Network for Action Recognition (->)
  - [ALGORITHM] Temporal Relational Reasoning in Videos (->)
  - [ALGORITHM] Temporal Segment Networks: Towards Good Practices for Deep Action Recognition (->)
  - [ALGORITHM] Tsm: Temporal Shift Module for Efficient Video Understanding (->)

  - [ALGORITHM] Video Classification With Channel-Separated Convolutional Networks (->)
  - [ALGORITHM] X3d: Expanding Architectures for Efficient Video Recognition (->)
  - [BACKBONE] Non-Local Neural Networks (-> ->)
  - [OTHERS] Large-Scale Weakly-Supervised Pre-Training for Video Action Recognition (->)

8.4 骨骼动作识别模型

- 模型权重文件数量: 16
- 配置文件数量: 16
- 论文数量: 3
  - [ALGORITHM] Revisiting Skeleton-Based Action Recognition (->)
  - [ALGORITHM] Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition (->)
  - [ALGORITHM] Two-Stream Adaptive Graph Convolutional Networks for Skeleton-Based Action Recognition (->)

CHAPTER 9

动作识别模型

9.1 C3D

9.1.1 简介

```
@ARTICLE{2014arXiv1412.0767T,
author = {Tran, Du and Bourdev, Lubomir and Fergus, Rob and Torresani, Lorenzo and␣
↪Paluri, Manohar},
title = {Learning Spatiotemporal Features with 3D Convolutional Networks},
keywords = {Computer Science - Computer Vision and Pattern Recognition},
year = 2014,
month = dec,
eid = {arXiv:1412.0767}
}
```

9.1.2 模型库

UCF-101

注:

1. C3D 的原论文使用 UCF-101 的数据均值进行数据正则化，并且使用 SVM 进行视频分类。MMAction2 使用 ImageNet 的 RGB 均值进行数据正则化，并且使用线性分类器。

2. 这里的 **GPU 数量**指的是得到模型权重文件对应的 GPU 个数。默认地，MMAction2 所提供的配置文件对应使用 8 块 GPU 进行训练的情况。依据 线性缩放规则，当用户使用不同数量的 GPU 或者每块 GPU 处理不同视频个数时，需要根据批大小等比例地调节学习率。如，lr=0.01 对应 4 GPUs x 2 video/gpu，以及 lr=0.08 对应 16 GPUs x 4 video/gpu。
3. 这里的 **推理时间**是根据 基准测试脚本 获得的，采用测试时的采帧策略，且只考虑模型的推理时间，并不包括 IO 时间以及预处理时间。对于每个配置，MMAction2 使用 1 块 GPU 并设置批大小（每块 GPU 处理的视频个数）为 1 来计算推理时间。

对于数据集准备的细节，用户可参考 数据集准备文档 中的 UCF-101 部分。

9.1.3 如何训练

用户可以使用以下指令进行模型训练。

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

例如：以一个确定性的训练方式，辅以定期的验证过程进行 C3D 模型在 UCF-101 数据集上的训练。

```
python tools/train.py configs/recognition/c3d/c3d_sports1m_16x1x1_45e_ucf101_rgb.py \
    --validate --seed 0 --deterministic
```

更多训练细节，可参考 基础教程 中的 **训练配置**部分。

9.1.4 如何测试

用户可以使用以下指令进行模型测试。

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

例如：在 UCF-101 数据集上测试 C3D 模型，并将结果导出为一个 json 文件。

```
python tools/test.py configs/recognition/c3d/c3d_sports1m_16x1x1_45e_ucf101_rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --eval top_k_accuracy
```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

9.2 CSN

9.2.1 简介

```
@inproceedings{inproceedings,
author = {Wang, Heng and Feiszli, Matt and Torresani, Lorenzo},
year = {2019},
month = {10},
pages = {5551-5560},
title = {Video Classification With Channel-Separated Convolutional Networks},
doi = {10.1109/ICCV.2019.00565}
}
```

```
@inproceedings{ghadiyaram2019large,
  title={Large-scale weakly-supervised pre-training for video action recognition},
  author={Ghadiyaram, Deepti and Tran, Du and Mahajan, Dhruv},
  booktitle={Proceedings of the IEEE Conference on Computer Vision and Pattern
↪Recognition},
  pages={12046--12055},
  year={2019}
}
```

9.2.2 模型库

Kinetics-400

注：

1. 这里的 **GPU 数量**指的是得到模型权重文件对应的 GPU 个数。默认地，MMAction2 所提供的配置文件对应使用 8 块 GPU 进行训练的情况。依据 线性缩放规则，当用户使用不同数量的 GPU 或者每块 GPU 处理不同视频个数时，需要根据批大小等比例地调节学习率。如，lr=0.01 对应 4 GPUs x 2 video/gpu，以及 lr=0.08 对应 16 GPUs x 4 video/gpu。
2. 这里的 **推理时间**是根据 基准测试脚本 获得的，采用测试时的采帧策略，且只考虑模型的推理时间，并不包括 IO 时间以及预处理时间。对于每个配置，MMAction2 使用 1 块 GPU 并设置批大小（每块 GPU 处理的视频个数）为 1 来计算推理时间。
3. 这里使用的 Kinetics400 验证集包含 19796 个视频，用户可以从 验证集视频 下载这些视频。同时也提供了对应的 数据列表 （每行格式为：视频 ID，视频帧数目，类别序号）以及 标签映射 （类别序号到类别名称）。
4. 这里的 **infer_ckpt** 表示该模型权重文件是从 VMZ 导入的。

对于数据集准备的细节，用户可参考 数据集准备文档 中的 Kinetics400 部分。

9.2.3 如何训练

用户可以使用以下指令进行模型训练。

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

例如：以一个确定性的训练方式，辅以定期的验证过程进行 CSN 模型在 Kinetics400 数据集上的训练。

```
python tools/train.py configs/recognition/csn/ircsn_ig65m_pretrained_r152_32x2x1_58e_
↪kinetics400_rgb.py \
    --work-dir work_dirs/ircsn_ig65m_pretrained_r152_32x2x1_58e_kinetics400_rgb \
    --validate --seed 0 --deterministic
```

更多训练细节，可参考 基础教程 中的 **训练配置**部分。

9.2.4 如何测试

用户可以使用以下指令进行模型测试。

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

例如：在 Kinetics400 数据集上测试 CSN 模型，并将结果导出为一个 json 文件。

```
python tools/test.py configs/recognition/csn/ircsn_ig65m_pretrained_r152_32x2x1_58e_
↪kinetics400_rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --eval top_k_accuracy mean_class_accuracy \
    --out result.json --average-clips prob
```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

9.3 I3D

9.3.1 简介

```
@inproceedings{inproceedings,
  author = {Carreira, J. and Zisserman, Andrew},
  year = {2017},
  month = {07},
  pages = {4724-4733},
  title = {Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset},
  doi = {10.1109/CVPR.2017.502}
}
```

```
@article{NonLocal2018,
  author =    {Xiaolong Wang and Ross Girshick and Abhinav Gupta and Kaiming He},
  title =     {Non-local Neural Networks},
  journal =   {CVPR},
  year =      {2018}
}
```

9.3.2 模型库

Kinetics-400

注：

1. 这里的 **GPU 数量**指的是得到模型权重文件对应的 GPU 个数。默认地，MMAction2 所提供的配置文件对应使用 8 块 GPU 进行训练的情况。依据 线性缩放规则，当用户使用不同数量的 GPU 或者每块 GPU 处理不同视频个数时，需要根据批大小等比例地调节学习率。如，lr=0.01 对应 4 GPUs x 2 video/gpu，以及 lr=0.08 对应 16 GPUs x 4 video/gpu。
2. 这里的 **推理时间**是根据 基准测试脚本 获得的，采用测试时的采帧策略，且只考虑模型的推理时间，并不包括 IO 时间以及预处理时间。对于每个配置，MMAction2 使用 1 块 GPU 并设置批大小（每块 GPU 处理的视频个数）为 1 来计算推理时间。
3. 我们使用的 Kinetics400 验证集包含 19796 个视频，用户可以从 验证集视频 下载这些视频。同时也提供了对应的 数据列表 （每行格式为：视频 ID，视频帧数目，类别序号）以及 标签映射 （类别序号到类别名称）。

对于数据集准备的细节，用户可参考 数据集准备文档 中的 Kinetics400 部分。

9.3.3 如何训练

用户可以使用以下指令进行模型训练。

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

例如：以一个确定性的训练方式，辅以定期的验证过程进行 I3D 模型在 Kinetics400 数据集上的训练。

```
python tools/train.py configs/recognition/i3d/i3d_r50_32x2x1_100e_kinetics400_rgb.py \
    --work-dir work_dirs/i3d_r50_32x2x1_100e_kinetics400_rgb \
    --validate --seed 0 --deterministic
```

更多训练细节，可参考 基础教程 中的 **训练配置**部分。

9.3.4 如何测试

用户可以使用以下指令进行模型测试。

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

例如：在 Kinetics400 数据集上测试 I3D 模型，并将结果导出为一个 json 文件。

```
python tools/test.py configs/recognition/i3d/i3d_r50_32x2x1_100e_kinetics400_rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --eval top_k_accuracy mean_class_accuracy \
    --out result.json --average-clips prob
```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

9.4 Omni-sourced Webly-supervised Learning for Video Recognition

Haodong Duan, Yue Zhao, Yuanjun Xiong, Wentao Liu, Dahua Lin

In ECCV, 2020. Paper, Dataset

9.4.1 模型库

Kinetics-400

MMAction2 当前公开了 4 个 OmniSource 框架训练的模型，包含 2D 架构与 3D 架构。下表比较了使用或不适用 OmniSource 框架训练得的模型在 Kinetics-400 上的精度：

1. 我们使用的 Kinetics400 验证集包含 19796 个视频，用户可以从 验证集视频 下载这些视频。同时也提供了对应的 数据列表 （每行格式为：视频 ID，视频帧数目，类别序号）以及 标签映射 （类别序号到类别名称）。

9.4.2 Mini-Kinetics 上的基准测试

OmniSource 项目当前公开了所采集网络数据的一个子集，涉及 Mini-Kinetics 中的 200 个动作类别。OmniSource 数据集准备 中记录了这些数据集的详细统计信息。用户可以通过填写 申请表 获取这些数据，在完成填写后，数据下载链接会被发送至用户邮箱。更多关于 OmniSource 网络数据集的信息请参照 OmniSource 数据集准备。

MMAction2 在公开的数据集上进行了 OmniSource 框架的基准测试，下表记录了详细的结果（在 Mini-Kinetics 验证集上的精度），这些结果可以作为使用网络数据训练视频识别任务的基线。

TSN-8seg-ResNet50

SlowOnly-8x8-ResNet50

下表列出了原论文中在 Kinetics-400 上进行基准测试的结果供参考：

9.4.3 注：

如果 OmniSource 项目对您的研究有所帮助，请使用以下 BibTex 项进行引用：

```
@article{duan2020omni,
  title={Omni-sourced Webly-supervised Learning for Video Recognition},
  author={Duan, Haodong and Zhao, Yue and Xiong, Yuanjun and Liu, Wentao and Lin,␣
↪Dahua},
  journal={arXiv preprint arXiv:2003.13042},
  year={2020}
}
```

9.5 R2plus1D

9.5.1 简介

```
@inproceedings{tran2018closer,
  title={A closer look at spatiotemporal convolutions for action recognition},
  author={Tran, Du and Wang, Heng and Torresani, Lorenzo and Ray, Jamie and LeCun,␣
↪Yann and Paluri, Manohar},
  booktitle={Proceedings of the IEEE conference on Computer Vision and Pattern␣
↪Recognition},
  pages={6450--6459},
  year={2018}
}
```

9.5.2 模型库

Kinetics-400

注：

1. 这里的 **GPU 数量**指的是得到模型权重文件对应的 GPU 个数。默认地，MMAction2 所提供的配置文件对应使用 8 块 GPU 进行训练的情况。依据 线性缩放规则，当用户使用不同数量的 GPU 或者每块 GPU

处理不同视频个数时，需要根据批大小等比例地调节学习率。如，lr=0.01 对应 4 GPUs x 2 video/gpu，以及 lr=0.08 对应 16 GPUs x 4 video/gpu。

2. 这里的 **推理时间**是根据 基准测试脚本 获得的，采用测试时的采帧策略，且只考虑模型的推理时间，并不包括 IO 时间以及预处理时间。对于每个配置，MMAction2 使用 1 块 GPU 并设置批大小（每块 GPU 处理的视频个数）为 1 来计算推理时间。

3. 我们使用的 Kinetics400 验证集包含 19796 个视频，用户可以从 验证集视频 下载这些视频。同时也提供了对应的 数据列表 （每行格式为：视频 ID，视频帧数目，类别序号）以及 标签映射 （类别序号到类别名称）。

对于数据集准备的细节，用户可参考 数据集准备文档 中的 Kinetics400 部分。

9.5.3 如何训练

用户可以使用以下指令进行模型训练。

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

例如：以一个确定性的训练方式，辅以定期的验证过程进行 R(2+1)D 模型在 Kinetics400 数据集上的训练。

```
python tools/train.py configs/recognition/r2plus1d/r2plus1d_r34_8x8x1_180e_
↪kinetics400_rgb.py \
    --work-dir work_dirs/r2plus1d_r34_3d_8x8x1_180e_kinetics400_rgb \
    --validate --seed 0 --deterministic
```

更多训练细节，可参考 基础教程 中的 **训练配置**部分。

9.5.4 如何测试

用户可以使用以下指令进行模型测试。

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

例如：在 Kinetics400 数据集上测试 R(2+1)D 模型，并将结果导出为一个 json 文件。

```
python tools/test.py configs/recognition/r2plus1d/r2plus1d_r34_8x8x1_180e_kinetics400_
↪rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --eval top_k_accuracy mean_class_accuracy \
    --out result.json --average-clips=prob
```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

9.6 SlowFast

9.6.1 简介

```
@inproceedings{feichtenhofer2019slowfast,
  title={Slowfast networks for video recognition},
  author={Feichtenhofer, Christoph and Fan, Haoqi and Malik, Jitendra and He, Kaiming}
↪,
  booktitle={Proceedings of the IEEE international conference on computer vision},
  pages={6202--6211},
  year={2019}
}
```

9.6.2 模型库

Kinetics-400

Something-Something V1

注:

1. 这里的 **GPU 数量**指的是得到模型权重文件对应的 GPU 个数。默认地，MMAction2 所提供的配置文件对应使用 8 块 GPU 进行训练的情况。依据 线性缩放规则，当用户使用不同数量的 GPU 或者每块 GPU 处理不同视频个数时，需要根据批大小等比例地调节学习率。如，lr=0.01 对应 4 GPUs x 2 video/gpu，以及 lr=0.08 对应 16 GPUs x 4 video/gpu。
2. 这里的 **推理时间**是根据 基准测试脚本 获得的，采用测试时的采帧策略，且只考虑模型的推理时间，并不包括 IO 时间以及预处理时间。对于每个配置，MMAction2 使用 1 块 GPU 并设置批大小（每块 GPU 处理的视频个数）为 1 来计算推理时间。
3. 我们使用的 Kinetics400 验证集包含 19796 个视频，用户可以从 验证集视频 下载这些视频。同时也提供了对应的 数据列表 （每行格式为：视频 ID，视频帧数目，类别序号）以及 标签映射 （类别序号到类别名称）。

对于数据集准备的细节，用户可参考 数据集准备文档 中的 Kinetics400 部分。

9.6.3 如何训练

用户可以使用以下指令进行模型训练。

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

例如：以一个确定性的训练方式，辅以定期的验证过程进行 SlowFast 模型在 Kinetics400 数据集上的训练。

```
python tools/train.py configs/recognition/slowfast/slowfast_r50_4x16x1_256e_
↪kinetics400_rgb.py \
    --work-dir work_dirs/slowfast_r50_4x16x1_256e_kinetics400_rgb \
    --validate --seed 0 --deterministic
```

更多训练细节，可参考 基础教程 中的 **训练配置**部分。

9.6.4 如何测试

用户可以使用以下指令进行模型测试。

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

例如：在 SlowFast 数据集上测试 CSN 模型，并将结果导出为一个 json 文件。

```
python tools/test.py configs/recognition/slowfast/slowfast_r50_4x16x1_256e_
↪kinetics400_rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --eval top_k_accuracy mean_class_accuracy \
    --out result.json --average-clips=prob
```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

9.7 SlowOnly

9.7.1 简介

```
@inproceedings{feichtenhofer2019slowfast,
  title={Slowfast networks for video recognition},
  author={Feichtenhofer, Christoph and Fan, Haoqi and Malik, Jitendra and He, Kaiming}
↪,
  booktitle={Proceedings of the IEEE international conference on computer vision},
  pages={6202--6211},
  year={2019}
}
```

9.7.2 模型库

Kinetics-400

Kinetics-400 数据基准测试

在数据基准测试中，比较两种不同的数据预处理方法 (1) 视频分辨率为 340x256, (2) 视频分辨率为短边 320px, (3) 视频分辨率为短边 256px.

Kinetics-400 OmniSource Experiments

Kinetics-600

Kinetics-700

GYM99

Jester

HMDB51

UCF101

Something-Something V1

注：

1. 这里的 **GPU 数量**指的是得到模型权重文件对应的 GPU 个数。默认地，MMAction2 所提供的配置文件对应使用 8 块 GPU 进行训练的情况。依据 线性缩放规则，当用户使用不同数量的 GPU 或者每块 GPU 处理不同视频个数时，需要根据批大小等比例地调节学习率。如，lr=0.01 对应 4 GPUs x 2 video/gpu，以及 lr=0.08 对应 16 GPUs x 4 video/gpu。
2. 这里的 **推理时间**是根据 基准测试脚本 获得的，采用测试时的采帧策略，且只考虑模型的推理时间，并不包括 IO 时间以及预处理时间。对于每个配置，MMAction2 使用 1 块 GPU 并设置批大小（每块 GPU 处理的视频个数）为 1 来计算推理时间。
3. 我们使用的 Kinetics400 验证集包含 19796 个视频，用户可以从 验证集视频 下载这些视频。同时也提供了对应的 数据列表 （每行格式为：视频 ID，视频帧数目，类别序号）以及 标签映射 （类别序号到类别名称）。

对于数据集准备的细节，用户可参考 数据集准备文档 中的 Kinetics400 部分。

9.7.3 如何训练

用户可以使用以下指令进行模型训练。

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

例如：以一个确定性的训练方式，辅以定期的验证过程进行 SlowOnly 模型在 Kinetics400 数据集上的训练。

```
python tools/train.py configs/recognition/slowonly/slowonly_r50_4x16x1_256e_
↪kinetics400_rgb.py \
    --work-dir work_dirs/slowonly_r50_4x16x1_256e_kinetics400_rgb \
    --validate --seed 0 --deterministic
```

更多训练细节，可参考 基础教程 中的 **训练配置**部分。

9.7.4 如何测试

用户可以使用以下指令进行模型测试。

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

例如：在 Kinetics400 数据集上测试 SlowOnly 模型，并将结果导出为一个 json 文件。

```
python tools/test.py configs/recognition/slowonly/slowonly_r50_4x16x1_256e_
↪kinetics400_rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --eval top_k_accuracy mean_class_accuracy \
    --out result.json --average-clips=prob
```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

9.8 TANet

9.8.1 简介

```
@article{liu2020tam,
  title={TAM: Temporal Adaptive Module for Video Recognition},
  author={Liu, Zhaoyang and Wang, Limin and Wu, Wayne and Qian, Chen and Lu, Tong},
  journal={arXiv preprint arXiv:2005.06803},
  year={2020}
}
```

9.8.2 模型库

Kinetics-400

Something-Something V1

注：

1. 这里的 **GPU 数量**指的是得到模型权重文件对应的 GPU 个数。默认地，MMAction2 所提供的配置文件对应使用 8 块 GPU 进行训练的情况。依据 线性缩放规则，当用户使用不同数量的 GPU 或者每块 GPU 处理不同视频个数时，需要根据批大小等比例地调节学习率。如，lr=0.01 对应 4 GPUs x 2 video/gpu，以及 lr=0.08 对应 16 GPUs x 4 video/gpu。
2. 这里的 **推理时间**是根据 基准测试脚本 获得的，采用测试时的采帧策略，且只考虑模型的推理时间，并不包括 IO 时间以及预处理时间。对于每个配置，MMAction2 使用 1 块 GPU 并设置批大小（每块 GPU 处理的视频个数）为 1 来计算推理时间。
3. 参考代码的结果是通过使用相同的模型配置在原来的代码库上训练得到的。对应的模型权重文件可从 这里 下载。
4. 我们使用的 Kinetics400 验证集包含 19796 个视频，用户可以从 验证集视频 下载这些视频。同时也提供了对应的 数据列表 （每行格式为：视频 ID，视频帧数目，类别序号）以及 标签映射 （类别序号到类别名称）。

对于数据集准备的细节，用户可参考 数据集准备文档 中的 Kinetics400 部分。

9.8.3 如何训练

用户可以使用以下指令进行模型训练。

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

例如：以一个确定性的训练方式，辅以定期的验证过程进行 TANet 模型在 Kinetics400 数据集上的训练。

```
python tools/train.py configs/recognition/tanet/tanet_r50_dense_1x1x8_100e_
↪kinetics400_rgb.py \
    --work-dir work_dirs/tanet_r50_dense_1x1x8_100e_kinetics400_rgb \
    --validate --seed 0 --deterministic
```

更多训练细节，可参考 基础教程 中的 **训练配置**部分。

9.8.4 如何测试

用户可以使用以下指令进行模型测试。

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

例如：在 Kinetics400 数据集上测试 TANet 模型，并将结果导出为一个 json 文件。

```
python tools/test.py configs/recognition/tanet/tanet_r50_dense_1x1x8_100e_kinetics400_
↪rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --eval top_k_accuracy mean_class_accuracy \
    --out result.json
```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

9.9 TimeSformer

9.9.1 简介

```
@misc{bertasius2021spacetime,
    title   = {Is Space-Time Attention All You Need for Video Understanding?},
    author  = {Gedas Bertasius and Heng Wang and Lorenzo Torresani},
    year    = {2021},
    eprint  = {2102.05095},
    archivePrefix = {arXiv},
    primaryClass = {cs.CV}
}
```

9.9.2 模型库

Kinetics-400

注：

1. 这里的 **GPU 数量**指的是得到模型权重文件对应的 GPU 个数（32G V100）。默认地，MMAction2 所提供的配置文件对应使用 8 块 GPU 进行训练的情况。依据 线性缩放规则，当用户使用不同数量的 GPU 或者每块 GPU 处理不同视频个数时，需要根据批大小等比例地调节学习率。如，lr=0.005 对应 8 GPUs x 8 video/gpu，以及 lr=0.004375 对应 8 GPUs x 7 video/gpu。
2. MMAction2 保持与 原代码 的测试设置一致（three crop x 1 clip）。
3. TimeSformer 使用的预训练模型 `vit_base_patch16_224.pth` 转换自 vision_transformer。

对于数据集准备的细节，用户可参考 数据集准备文档 中的 Kinetics400 部分。

9.9.3 如何训练

用户可以使用以下指令进行模型训练。

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

例如：以一个确定性的训练方式，辅以定期的验证过程进行 TimeSformer 模型在 Kinetics400 数据集上的训练。

```
python tools/train.py configs/recognition/timesformer/timesformer_divST_8x32x1_15e_
↪kinetics400_rgb.py \
    --work-dir work_dirs/timesformer_divST_8x32x1_15e_kinetics400_rgb.py \
    --validate --seed 0 --deterministic
```

更多训练细节，可参考 基础教程 中的 **训练配置**部分。

9.9.4 如何测试

用户可以使用以下指令进行模型测试。

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

例如：在 Kinetics400 数据集上测试 TimeSformer 模型，并将结果导出为一个 json 文件。

```
python tools/test.py configs/recognition/timesformer/timesformer_divST_8x32x1_15e_
↪kinetics400_rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --eval top_k_accuracy mean_class_accuracy \
    --out result.json
```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

9.10 TIN

9.10.1 简介

```
@article{shao2020temporal,
    title={Temporal Interlacing Network},
    author={Hao Shao and Shengju Qian and Yu Liu},
    year={2020},
    journal={AAAI},
}
```

9.10.2 模型库

Something-Something V1

Something-Something V2

Kinetics-400

这里，MMAction2 使用 `finetune` 一词表示 TIN 模型使用 Kinetics400 上的 TSM 模型 进行微调。

注：

1. 参考代码的结果是通过 原始 repo 解决 AverageMeter 相关问题 后训练得到的，该问题会导致错误的精度计算。
2. 这里的 **GPU 数量**指的是得到模型权重文件对应的 GPU 个数。默认地，MMAction2 所提供的配置文件对应使用 8 块 GPU 进行训练的情况。依据 线性缩放规则，当用户使用不同数量的 GPU 或者每块 GPU 处理不同视频个数时，需要根据批大小等比例地调节学习率。如，lr=0.01 对应 4 GPUs x 2 video/gpu，以及 lr=0.08 对应 16 GPUs x 4 video/gpu。
3. 这里的 **推理时间**是根据 基准测试脚本 获得的，采用测试时的采帧策略，且只考虑模型的推理时间，并不包括 IO 时间以及预处理时间。对于每个配置，MMAction2 使用 1 块 GPU 并设置批大小（每块 GPU 处理的视频个数）为 1 来计算推理时间。
4. 参考代码的结果是通过使用相同的模型配置在原来的代码库上训练得到的。
5. 我们使用的 Kinetics400 验证集包含 19796 个视频，用户可以从 验证集视频 下载这些视频。同时也提供了对应的 数据列表 （每行格式为：视频 ID，视频帧数目，类别序号）以及 标签映射 （类别序号到类别名称）。

对于数据集准备的细节，用户可参考 数据集准备文档 中的 Kinetics400, Something-Something V1 and Something-Something V2 部分。

9.10.3 如何训练

用户可以使用以下指令进行模型训练。

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

例如：以一个确定性的训练方式，辅以定期的验证过程进行 TIN 模型在 Something-Something V1 数据集上的训练。

```
python tools/train.py configs/recognition/tin/tin_r50_1x1x8_40e_sthv1_rgb.py \
    --work-dir work_dirs/tin_r50_1x1x8_40e_sthv1_rgb \
    --validate --seed 0 --deterministic
```

更多训练细节，可参考 基础教程 中的 **训练配置**部分。

9.10.4 如何测试

用户可以使用以下指令进行模型测试。

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

例如：在 Something-Something V1 数据集上测试 TIN 模型，并将结果导出为一个 json 文件。

```
python tools/test.py configs/recognition/tin/tin_r50_1x1x8_40e_sthv1_rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --eval top_k_accuracy mean_class_accuracy \
    --out result.json
```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

9.11 TPN

9.11.1 简介

```
@inproceedings{yang2020tpn,
  title={Temporal Pyramid Network for Action Recognition},
  author={Yang, Ceyuan and Xu, Yinghao and Shi, Jianping and Dai, Bo and Zhou, Bolei},
  booktitle={Proceedings of the IEEE Conference on Computer Vision and Pattern␣
↪Recognition (CVPR)},
  year={2020},
}
```

9.11.2 模型库

Kinetics-400

Something-Something V1

| 配置文件 | GPU 数量 | 主干网络 | 预训练 | top1 准确率 | top5 准确率 | GPU 显存占用 (M) | ckpt | log| json| |:–|:–:|:–:|:–:|:–:|:–:|:–:|:–:|:–:|:–:|:–:|:–:|:–:|:–:|:–:| |tpn_tsm_r50_1x1x8_150e_sthv1_rgb|height 100|8x6| ResNet50 | TSM | 51.50 | 79.15 | 8828 |ckpt |log|json|

注：

1. 这里的 **GPU 数量**指的是得到模型权重文件对应的 GPU 个数。默认地，MMAction2 所提供的配置文件对应使用 8 块 GPU 进行训练的情况。依据 线性缩放规则，当用户使用不同数量的 GPU 或者每块 GPU 处理不同视频个数时，需要根据批大小等比例地调节学习率。如，lr=0.01 对应 4 GPUs x 2 video/gpu，以及 lr=0.08 对应 16 GPUs x 4 video/gpu。

2. 这里的 **推理时间**是根据 基准测试脚本 获得的，采用测试时的采帧策略，且只考虑模型的推理时间，并不包括 IO 时间以及预处理时间。对于每个配置，MMAction2 使用 1 块 GPU 并设置批大小（每块 GPU 处理的视频个数）为 1 来计算推理时间。
3. 参考代码的结果是通过使用相同的模型配置在原来的代码库上训练得到的。
4. 我们使用的 Kinetics400 验证集包含 19796 个视频，用户可以从 验证集视频 下载这些视频。同时也提供了对应的 数据列表 （每行格式为：视频 ID，视频帧数目，类别序号）以及 标签映射 （类别序号到类别名称）。

9.11.3 如何训练

用户可以使用以下指令进行模型训练。

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

例如：以一个确定性的训练方式，辅以定期的验证过程进行 TPN 模型在 Kinetics-400 数据集上的训练。

```
python tools/train.py configs/recognition/tpn/tpn_slowonly_r50_8x8x1_150e_kinetics_
↪rgb.py \
    --work-dir work_dirs/tpn_slowonly_r50_8x8x1_150e_kinetics_rgb [--validate --seed␣
↪0 --deterministic]
```

更多训练细节，可参考 基础教程 中的 **训练配置**部分。

9.11.4 如何测试

用户可以使用以下指令进行模型测试。

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

例如：在 Kinetics-400 数据集上测试 TPN 模型，并将结果导出为一个 json 文件。

```
python tools/test.py configs/recognition/tpn/tpn_slowonly_r50_8x8x1_150e_kinetics_rgb.
↪py \
    checkpoints/SOME_CHECKPOINT.pth --eval top_k_accuracy mean_class_accuracy \
    --out result.json --average-clips prob
```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

9.12 TRN

9.12.1 简介

```
@article{zhou2017temporalrelation,
    title = {Temporal Relational Reasoning in Videos},
    author = {Zhou, Bolei and Andonian, Alex and Oliva, Aude and Torralba, Antonio},
    journal={European Conference on Computer Vision},
    year={2018}
}
```

9.12.2 模型库

Something-Something V1

Something-Something V2

注：

1. 这里的 **GPU 数量**指的是得到模型权重文件对应的 GPU 个数。默认地，MMAction2 所提供的配置文件对应使用 8 块 GPU 进行训练的情况。依据 线性缩放规则，当用户使用不同数量的 GPU 或者每块 GPU 处理不同视频个数时，需要根据批大小等比例地调节学习率。如，lr=0.01 对应 4 GPUs x 2 video/gpu，以及 lr=0.08 对应 16 GPUs x 4 video/gpu。
2. 对于 Something-Something 数据集，有两种测试方案：efficient（对应 center crop x 1 clip）和 accurate（对应 Three crop x 2 clip）。
3. 在原代码库中，作者在 Something-Something 数据集上使用了随机水平翻转，但这种数据增强方法有一些问题，因为 Something-Something 数据集有一些方向性的动作，比如从左往右推。所以 MMAction2 把随机水平翻转改为带标签映射的水平翻转，同时修改了测试模型的数据处理方法，即把裁剪 `10` 个图像块（这里面包括 5 个翻转后的图像块）修改成采帧两次 `&` 裁剪 `3` 个图像块。
4. MMAction2 使用 `ResNet50` 代替 `BNInception` 作为 TRN 的主干网络。使用原代码，在 sthv1 数据集上训练 `TRN-ResNet50` 时，实验得到的 top1 (top5) 的准确度为 30.542 (58.627)，而 MMAction2 的精度为 31.62 (60.01)。

关于数据处理的更多细节，用户可以参照

- 准备 sthv1
- 准备 sthv2

9.12.3 如何训练

用户可以使用以下指令进行模型训练。

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

例如：以一个确定性的训练方式，辅以定期的验证过程进行 TRN 模型在 sthv1 数据集上的训练。

```
python tools/train.py configs/recognition/trn/trn_r50_1x1x8_50e_sthv1_rgb.py \
    --work-dir work_dirs/trn_r50_1x1x8_50e_sthv1_rgb \
    --validate --seed 0 --deterministic
```

更多训练细节，可参考 基础教程 中的 **训练配置**部分。

9.12.4 如何测试

用户可以使用以下指令进行模型测试。

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

例如：在 sthv1 数据集上测试 TRN 模型，并将结果导出为一个 json 文件。

```
python tools/test.py configs/recognition/trn/trn_r50_1x1x8_50e_sthv1_rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --eval top_k_accuracy mean_class_accuracy \
    --out result.json
```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

9.13 TSM

9.13.1 简介

```
@inproceedings{lin2019tsm,
  title={TSM: Temporal Shift Module for Efficient Video Understanding},
  author={Lin, Ji and Gan, Chuang and Han, Song},
  booktitle={Proceedings of the IEEE International Conference on Computer Vision},
  year={2019}
}
```

```
@article{NonLocal2018,
  author =    {Xiaolong Wang and Ross Girshick and Abhinav Gupta and Kaiming He},
  title =     {Non-local Neural Networks},
```

(下页继续)

(续上页)

```
  journal =  {CVPR},
  year =     {2018}
}
```

9.13.2 模型库

Kinetics-400

Diving48

Something-Something V1

Something-Something V2

Diving48

MixUp & CutMix on Something-Something V1

Jester

HMDB51

UCF101

注：

1. 这里的 **GPU 数量**指的是得到模型权重文件对应的 GPU 个数。默认地，MMAction2 所提供的配置文件对应使用 8 块 GPU 进行训练的情况。依据 线性缩放规则，当用户使用不同数量的 GPU 或者每块 GPU 处理不同视频个数时，需要根据批大小等比例地调节学习率。如，lr=0.01 对应 4 GPUs x 2 video/gpu，以及 lr=0.08 对应 16 GPUs x 4 video/gpu。
2. 这里的 **推理时间**是根据 基准测试脚本 获得的，采用测试时的采帧策略，且只考虑模型的推理时间，并不包括 IO 时间以及预处理时间。对于每个配置，MMAction2 使用 1 块 GPU 并设置批大小（每块 GPU 处理的视频个数）为 1 来计算推理时间。
3. 参考代码的结果是通过使用相同的模型配置在原来的代码库上训练得到的。对应的模型权重文件可从 这里 下载。
4. 对于 Something-Something 数据集，有两种测试方案：efficient（对应 center crop x 1 clip）和 accurate（对应 Three crop x 2 clip）。两种方案参考自 原始代码库。MMAction2 使用 efficient 方案作为配置文件中的默认选择，用户可以通过以下方式转变为 accurate 方案：

```
...
test_pipeline = [
    dict(
        type='SampleFrames',
        clip_len=1,
        frame_interval=1,
        num_clips=16,   ## 当使用 8 个 视频段时，设置 `num_clips = 8`
        twice_sample=True,    ## 设置 `twice_sample=True` 用于 accurate 方案中的 Twice␣
↪Sample
        test_mode=True),
    dict(type='RawFrameDecode'),
    dict(type='Resize', scale=(-1, 256)),
    ## dict(type='CenterCrop', crop_size=224), 用于 efficient 方案
    dict(type='ThreeCrop', crop_size=256),  ## 用于 accurate 方案
    dict(type='Normalize', **img_norm_cfg),
    dict(type='FormatShape', input_format='NCHW'),
    dict(type='Collect', keys=['imgs', 'label'], meta_keys=[]),
    dict(type='ToTensor', keys=['imgs'])
]
```

5. 当采用 Mixup 和 CutMix 的数据增强时，使用超参 `alpha=0.2`。
6. 我们使用的 Kinetics400 验证集包含 19796 个视频，用户可以从 验证集视频 下载这些视频。同时也提供了对应的 数据列表 （每行格式为：视频 ID，视频帧数目，类别序号）以及 标签映射 （类别序号到类别名称）。
7. 这里的 **infer_ckpt** 表示该模型权重文件是从 TSM 导入的。

对于数据集准备的细节，用户可参考 数据集准备文档 中的 Kinetics400, Something-Something V1 and Something-Something V2 部分。

9.13.3 如何训练

用户可以使用以下指令进行模型训练。

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

例如：以一个确定性的训练方式，辅以定期的验证过程进行 TSM 模型在 Kinetics-400 数据集上的训练。

```
python tools/train.py configs/recognition/tsm/tsm_r50_1x1x8_50e_kinetics400_rgb.py \
    --work-dir work_dirs/tsm_r50_1x1x8_100e_kinetics400_rgb \
    --validate --seed 0 --deterministic
```

更多训练细节，可参考 基础教程 中的 **训练配置**部分。

9.13.4 如何测试

用户可以使用以下指令进行模型测试。

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

例如：在 Kinetics-400 数据集上测试 TSM 模型，并将结果导出为一个 json 文件。

```
python tools/test.py configs/recognition/tsm/tsm_r50_1x1x8_50e_kinetics400_rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --eval top_k_accuracy mean_class_accuracy \
    --out result.json
```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

9.14 TSN

9.14.1 简介

```
@inproceedings{wang2016temporal,
  title={Temporal segment networks: Towards good practices for deep action␣
↪recognition},
  author={Wang, Limin and Xiong, Yuanjun and Wang, Zhe and Qiao, Yu and Lin, Dahua␣
↪and Tang, Xiaoou and Van Gool, Luc},
  booktitle={European conference on computer vision},
  pages={20--36},
  year={2016},
  organization={Springer}
}
```

9.14.2 模型库

UCF-101

[1] 这里汇报的是 UCF-101 的 split1 部分的结果。

Diving48

HMDB51

Kinetics-400

这里，MMAction2 使用 [1: 1] 表示以 1: 1 的比例融合 RGB 和光流两分支的融合结果（融合前不经过 softmax）

在 TSN 模型中使用第三方的主干网络

用户可在 MMAction2 的框架中使用第三方的主干网络训练 TSN，例如：

- [x] MMClassification 中的主干网络
- [x] TorchVision 中的主干网络
- [x] pytorch-image-models(timm) 中的主干网络

1. 由于多种原因，TIMM 中的一些模型未能收到支持，详情请参考 PR ##880。

Kinetics-400 数据基准测试 (8 块 GPU, ResNet50, ImageNet 预训练; 3 个视频段)

在数据基准测试中，比较：

1. 不同的数据预处理方法：(1) 视频分辨率为 340x256, (2) 视频分辨率为短边 320px, (3) 视频分辨率为短边 256px;
2. 不同的数据增强方法：(1) MultiScaleCrop, (2) RandomResizedCrop;
3. 不同的测试方法：(1) 25 帧 x 10 裁剪片段, (2) 25 frames x 3 裁剪片段.

Kinetics-400 OmniSource 实验

[1] MMAction2 使用 torch-hub 提供的 `resnet50_swsl` 预训练模型。

Kinetics-600

Kinetics-700

Something-Something V1

Something-Something V2

Moments in Time

Multi-Moments in Time

ActivityNet v1.3

HVU

[1] 简单起见，MMAction2 对每个 tag 类别训练特定的模型，作为 HVU 的基准模型。

[2] 这里 HATNet 和 HATNet-multi 的结果来自于 paper: Large Scale Holistic Video Understanding。HATNet 的时序动作候选是一个双分支的卷积网络（一个 2D 分支，一个 3D 分支），并且和 MMAction2 有相同的主干网络（ResNet18）。HATNet 的输入是 16 帧或 32 帧的长视频片段（这样的片段比 MMAction2 使用的要长），同时输入分辨率更粗糙（112px 而非 224px）。HATNet 是在每个独立的任务（对应每个 tag 类别）上进行训练的，HATNet-multi 是在多个任务上进行训练的。由于目前没有 HATNet 的开源代码和模型，这里仅汇报了原 paper 的精度。

注：

1. 这里的 **GPU 数量**指的是得到模型权重文件对应的 GPU 个数。默认地，MMAction2 所提供的配置文件对应使用 8 块 GPU 进行训练的情况。依据 线性缩放规则，当用户使用不同数量的 GPU 或者每块 GPU 处理不同视频个数时，需要根据批大小等比例地调节学习率。如，lr=0.01 对应 4 GPUs x 2 video/gpu，以及 lr=0.08 对应 16 GPUs x 4 video/gpu。
2. 这里的 **推理时间**是根据 基准测试脚本 获得的，采用测试时的采帧策略，且只考虑模型的推理时间，并不包括 IO 时间以及预处理时间。对于每个配置，MMAction2 使用 1 块 GPU 并设置批大小（每块 GPU 处理的视频个数）为 1 来计算推理时间。
3. 参考代码的结果是通过使用相同的模型配置在原来的代码库上训练得到的。
4. 我们使用的 Kinetics400 验证集包含 19796 个视频，用户可以从 验证集视频 下载这些视频。同时也提供了对应的 数据列表 （每行格式为：视频 ID，视频帧数目，类别序号）以及 标签映射 （类别序号到类别名称）。

对于数据集准备的细节，用户可参考：

- 准备 ucf101
- 准备 kinetics
- 准备 sthv1
- 准备 sthv2
- 准备 mit
- 准备 mmit
- 准备 hvu
- 准备 hmdb51

9.14.3 如何训练

用户可以使用以下指令进行模型训练。

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

例如：以一个确定性的训练方式，辅以定期的验证过程进行 TSN 模型在 Kinetics-400 数据集上的训练。

```
python tools/train.py configs/recognition/tsn/tsn_r50_1x1x3_100e_kinetics400_rgb.py \
    --work-dir work_dirs/tsn_r50_1x1x3_100e_kinetics400_rgb \
    --validate --seed 0 --deterministic
```

更多训练细节，可参考 基础教程 中的 **训练配置**部分。

9.14.4 如何测试

用户可以使用以下指令进行模型测试。

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

例如：在 Kinetics-400 数据集上测试 TSN 模型，并将结果导出为一个 json 文件。

```
python tools/test.py configs/recognition/tsn/tsn_r50_1x1x3_100e_kinetics400_rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --eval top_k_accuracy mean_class_accuracy \
    --out result.json
```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

9.15 X3D

9.15.1 简介

```
@misc{feichtenhofer2020x3d,
      title={X3D: Expanding Architectures for Efficient Video Recognition},
      author={Christoph Feichtenhofer},
      year={2020},
      eprint={2004.04730},
      archivePrefix={arXiv},
      primaryClass={cs.CV}
}
```

9.15.2 模型库

Kinetics-400

[1] 这里的模型是从 SlowFast 代码库中导入并在 MMAction2 使用的数据上进行测试的。目前仅支持 X3D 模型的测试，训练部分将会在近期提供。

注：

1. 参考代码的结果是通过使用相同的数据和原来的代码库所提供的模型进行测试得到的。
2. 我们使用的 Kinetics400 验证集包含 19796 个视频，用户可以从 验证集视频 下载这些视频。同时也提供了对应的 数据列表 （每行格式为：视频 ID，视频帧数目，类别序号）以及 标签映射 （类别序号到类别名称）。

对于数据集准备的细节，用户可参考 数据集准备文档 中的 Kinetics400 部分

9.15.3 如何测试

用户可以使用以下指令进行模型测试。

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

例如：在 Kinetics-400 数据集上测试 X3D 模型，并将结果导出为一个 json 文件。

```
python tools/test.py configs/recognition/x3d/x3d_s_13x6x1_facebook_kinetics400_rgb.py␣
↪\
    checkpoints/SOME_CHECKPOINT.pth --eval top_k_accuracy mean_class_accuracy \
    --out result.json --average-clips prob
```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

9.16 ResNet for Audio

9.16.1 简介

```
@article{xiao2020audiovisual,
  title={Audiovisual SlowFast Networks for Video Recognition},
  author={Xiao, Fanyi and Lee, Yong Jae and Grauman, Kristen and Malik, Jitendra and␣
↪Feichtenhofer, Christoph},
  journal={arXiv preprint arXiv:2001.08740},
  year={2020}
}
```

9.16.2 模型库

Kinetics-400

注：

1. 这里的 **GPU 数量**指的是得到模型权重文件对应的 GPU 个数。默认地，MMAction2 所提供的配置文件对应使用 8 块 GPU 进行训练的情况。依据 线性缩放规则，当用户使用不同数量的 GPU 或者每块 GPU 处理不同视频个数时，需要根据批大小等比例地调节学习率。如，lr=0.01 对应 4 GPUs x 2 video/gpu，以及 lr=0.08 对应 16 GPUs x 4 video/gpu。
2. 这里的 **推理时间**是根据 基准测试脚本 获得的，采用测试时的采帧策略，且只考虑模型的推理时间，并不包括 IO 时间以及预处理时间。对于每个配置，MMAction2 使用 1 块 GPU 并设置批大小（每块 GPU 处理的视频个数）为 1 来计算推理时间。
3. 我们使用的 Kinetics400 验证集包含 19796 个视频，用户可以从 验证集视频 下载这些视频。同时也提供了对应的 数据列表 （每行格式为：视频 ID，视频帧数目，类别序号）以及 标签映射 （类别序号到类别名称）。

对于数据集准备的细节，用户可参考 数据集准备文档 中的准备音频部分。

9.16.3 如何训练

用户可以使用以下指令进行模型训练。

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

Example: 以一个确定性的训练方式，辅以定期的验证过程进行 ResNet 模型在 Kinetics400 音频数据集上的训练。

```
python tools/train.py configs/audio_recognition/tsn_r50_64x1x1_100e_kinetics400_audio_
↪feature.py \
    --work-dir work_dirs/tsn_r50_64x1x1_100e_kinetics400_audio_feature \
    --validate --seed 0 --deterministic
```

更多训练细节，可参考 基础教程 中的 **训练配置**部分。

9.16.4 如何测试

用户可以使用以下指令进行模型测试。

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

例如：在 Kinetics400 音频数据集上测试 ResNet 模型，并将结果导出为一个 json 文件。

```
python tools/test.py configs/audio_recognition/tsn_r50_64x1x1_100e_kinetics400_audio_
↪feature.py \
    checkpoints/SOME_CHECKPOINT.pth --eval top_k_accuracy mean_class_accuracy \
    --out result.json
```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

9.16.5 融合

对于多模态融合，用户可以使用这个 脚本，其命令大致为：

```
python tools/analysis/report_accuracy.py --scores ${AUDIO_RESULT_PKL} ${VISUAL_RESULT_
↪PKL} --datalist data/kinetics400/kinetics400_val_list_rawframes.txt --coefficient 1␣
↪1
```

- AUDIO_RESULT_PKL: `tools/test.py` 脚本通过 `--out` 选项存储的输出文件。
- VISUAL_RESULT_PKL: `tools/test.py` 脚本通过 `--out` 选项存储的输出文件。

CHAPTER 10

时序动作检测模型

10.1 BMN

10.1.1 简介

```
@inproceedings{lin2019bmn,
  title={Bmn: Boundary-matching network for temporal action proposal generation},
  author={Lin, Tianwei and Liu, Xiao and Li, Xin and Ding, Errui and Wen, Shilei},
  booktitle={Proceedings of the IEEE International Conference on Computer Vision},
  pages={3889--3898},
  year={2019}
}
```

```
@article{zhao2017cuhk,
  title={Cuhk \& ethz \& siat submission to activitynet challenge 2017},
  author={Zhao, Y and Zhang, B and Wu, Z and Yang, S and Zhou, L and Yan, S and Wang,␣
↪L and Xiong, Y and Lin, D and Qiao, Y and others},
  journal={arXiv preprint arXiv:1710.08011},
  volume={8},
  year={2017}
}
```

10.1.2 模型库

ActivityNet feature

- 注：

1. 这里的 **GPU 数量**指的是得到模型权重文件对应的 GPU 个数。默认地，MMAction2 所提供的配置文件对应使用 8 块 GPU 进行训练的情况。依据 线性缩放规则，当用户使用不同数量的 GPU 或者每块 GPU 处理不同视频个数时，需要根据批大小等比例地调节学习率。如，lr=0.01 对应 4 GPUs x 2 video/gpu，以及 lr=0.08 对应 16 GPUs x 4 video/gpu。
2. 对于 **特征**这一列，`cuhk_mean_100` 表示所使用的特征为利用 anet2016-cuhk 代码库抽取的，被广泛利用的 CUHK ActivityNet 特征，`mmaction_video` 和 `mmaction_clip` 分布表示所使用的特征为利用 MMAction 抽取的，视频级别 ActivityNet 预训练模型的特征；视频片段级别 ActivityNet 预训练模型的特征。
3. MMAction2 使用 ActivityNet2017 未剪辑视频分类赛道上 anet_cuhk_2017 所提交的结果来为每个视频的时序动作候选指定标签，以用于 BMN 模型评估。

*MMAction2 在 原始代码库 上训练 BMN，并且在 anet_cuhk_2017 的对应标签上评估时序动作候选生成和时序检测的结果。

对于数据集准备的细节，用户可参考 数据集准备文档 中的 ActivityNet 特征部分。

10.1.3 如何训练

用户可以使用以下指令进行模型训练。

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

例如：在 ActivityNet 特征上训练 BMN。

```
python tools/train.py configs/localization/bmn/bmn_400x100_2x8_9e_activitynet_feature.
↪py
```

更多训练细节，可参考 基础教程 中的 **训练配置**部分。

10.1.4 如何测试

用户可以使用以下指令进行模型测试。

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

例如：在 ActivityNet 特征上测试 BMN 模型。

```
## 注：如果需要进行指标验证，需确测试数据的保标注文件包含真实标签
python tools/test.py configs/localization/bmn/bmn_400x100_2x8_9e_activitynet_feature.
↪py checkpoints/SOME_CHECKPOINT.pth --eval AR@AN --out results.json
```

用户也可以利用 anet_cuhk_2017 的预测文件评估模型时序检测的结果，并生成时序动作候选文件（即命令中的 `results.json`）

```
python tools/analysis/report_map.py --proposal path/to/proposal_file
```

注：

1. (可选项) 用户可以使用以下指令生成格式化的时序动作候选文件，该文件可被送入动作识别器中（目前只支持 SSN 和 P-GCN，不包括 TSN, I3D 等），以获得时序动作候选的分类结果。

   ```
   python tools/data/activitynet/convert_proposal_format.py
   ```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

10.2 BSN

10.2.1 简介

```
@inproceedings{lin2018bsn,
  title={Bsn: Boundary sensitive network for temporal action proposal generation},
  author={Lin, Tianwei and Zhao, Xu and Su, Haisheng and Wang, Chongjing and Yang,␣
↪Ming},
  booktitle={Proceedings of the European Conference on Computer Vision (ECCV)},
  pages={3--19},
  year={2018}
}
```

10.2.2 模型库

ActivityNet feature

注：

1. 这里的 **GPU 数量**指的是得到模型权重文件对应的 GPU 个数。默认地，MMAction2 所提供的配置文件对应使用 8 块 GPU 进行训练的情况。依据 线性缩放规则，当用户使用不同数量的 GPU 或者每块 GPU 处理不同视频个数时，需要根据批大小等比例地调节学习率。如，lr=0.01 对应 4 GPUs x 2 video/gpu，以及 lr=0.08 对应 16 GPUs x 4 video/gpu。

2. 对于 **特征**这一列，cuhk_mean_100 表示所使用的特征为利用 anet2016-cuhk 代码库抽取的，被广泛利用的 CUHK ActivityNet 特征，mmaction_video 和 mmaction_clip 分布表示所使用的特征为利用 MMAction 抽取的，视频级别 ActivityNet 预训练模型的特征；视频片段级别 ActivityNet 预训练模型的特征。

对于数据集准备的细节，用户可参考 数据集准备文档 中的 ActivityNet 特征部分。

10.2.3 如何训练

用户可以使用以下指令进行模型训练。

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

例如：

1. 在 ActivityNet 特征上训练 BSN(TEM) 模型。

```
python tools/train.py configs/localization/bsn/bsn_tem_400x100_1x16_20e_
↪activitynet_feature.py
```

2. 基于 PGM 的结果训练 BSN(PEM)。

```
python tools/train.py configs/localization/bsn/bsn_pem_400x100_1x16_20e_
↪activitynet_feature.py
```

更多训练细节，可参考 基础教程 中的 **训练配置**部分。

10.2.4 如何进行推理

用户可以使用以下指令进行模型推理。

1. 推理 TEM 模型。

```
## Note: This could not be evaluated.
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [optional arguments]
```

2. 推理 PGM 模型

```
python tools/misc/bsn_proposal_generation.py ${CONFIG_FILE} [--mode ${MODE}]
```

3. 推理 PEM 模型

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

例如

1. 利用预训练模型进行 BSN(TEM) 模型的推理。

```
python tools/test.py configs/localization/bsn/bsn_tem_400x100_1x16_20e_
↪activitynet_feature.py checkpoints/SOME_CHECKPOINT.pth
```

2. 利用预训练模型进行 BSN(PGM) 模型的推理

```
python tools/misc/bsn_proposal_generation.py configs/localization/bsn/bsn_pgm_
↪400x100_activitynet_feature.py --mode train
```

3. 推理 BSN(PEM) 模型，并计算‘AR@AN’指标，输出结果文件。

```
## 注：如果需要进行指标验证，需确测试数据的保标注文件包含真实标签
python tools/test.py configs/localization/bsn/bsn_pem_400x100_1x16_20e_
↪activitynet_feature.py  checkpoints/SOME_CHECKPOINT.pth  --eval AR@AN --out␣
↪results.json
```

10.2.5 如何测试

用户可以使用以下指令进行模型测试。

1. TEM

```
## 注：该命令无法进行指标验证
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [optional arguments]
```

2. PGM

```
python tools/misc/bsn_proposal_generation.py ${CONFIG_FILE} [--mode ${MODE}]
```

3. PEM

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

例如：

1. 在 ActivityNet 数据集上测试 TEM 模型。

```
python tools/test.py configs/localization/bsn/bsn_tem_400x100_1x16_20e_
↪activitynet_feature.py checkpoints/SOME_CHECKPOINT.pth
```

2. 在 ActivityNet 数据集上测试 PGM 模型。

```
python tools/misc/bsn_proposal_generation.py configs/localization/bsn/bsn_pgm_
↪400x100_activitynet_feature.py --mode test
```

3. 测试 PEM 模型，并计算‘AR@AN’指标，输出结果文件。

```
python tools/test.py configs/localization/bsn/bsn_pem_400x100_1x16_20e_
↪activitynet_feature.py checkpoints/SOME_CHECKPOINT.pth --eval AR@AN --out␣
↪results.json
```

注:

1. (可选项) 用户可以使用以下指令生成格式化的时序动作候选文件，该文件可被送入动作识别器中（目前只支持 SSN 和 P-GCN，不包括 TSN, I3D 等），以获得时序动作候选的分类结果。

```
python tools/data/activitynet/convert_proposal_format.py
```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

10.3 SSN

10.3.1 简介

```
@InProceedings{Zhao_2017_ICCV,
author = {Zhao, Yue and Xiong, Yuanjun and Wang, Limin and Wu, Zhirong and Tang,␣
↪Xiaoou and Lin, Dahua},
title = {Temporal Action Detection With Structured Segment Networks},
booktitle = {Proceedings of the IEEE International Conference on Computer Vision␣
↪(ICCV)},
month = {Oct},
year = {2017}
}
```

10.3.2 模型库

注:

1. 这里的 **GPU 数量**指的是得到模型权重文件对应的 GPU 个数。默认地，MMAction2 所提供的配置文件对应使用 8 块 GPU 进行训练的情况。依据 线性缩放规则，当用户使用不同数量的 GPU 或者每块 GPU 处理不同视频个数时，需要根据批大小等比例地调节学习率。如，lr=0.01 对应 4 GPUs x 2 video/gpu，以及 lr=0.08 对应 16 GPUs x 4 video/gpu。
2. 由于 SSN 在训练和测试阶段使用不同的结构化时序金字塔池化方法（structured temporal pyramid pooling methods），请分别参考 ssn_r50_450e_thumos14_rgb_train 和 ssn_r50_450e_thumos14_rgb_test。
3. MMAction2 使用 TAG 的时序动作候选进行 SSN 模型的精度验证。关于数据准备的更多细节，用户可参考 Data 数据集准备文档 准备 thumos14 的 TAG 时序动作候选。
4. 参考代码的 SSN 模型是和 MMAction2 一样在 `ResNet50` 主干网络上验证的。注意，这里的 SSN 的初始设置与原代码库的 `BNInception` 骨干网络的设置相同。

10.3.3 如何训练

用户可以使用以下指令进行模型训练。

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

例如：在 thumos14 数据集上训练 SSN 模型。

```
python tools/train.py configs/localization/ssn/ssn_r50_450e_thumos14_rgb_train.py
```

更多训练细节，可参考 基础教程 中的 **训练配置**部分。

10.3.4 如何测试

用户可以使用以下指令进行模型测试。

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

例如：在 ActivityNet 特征上测试 BMN。

```
## 注：如果需要进行指标验证，需确测试数据的保标注文件包含真实标签
python tools/test.py configs/localization/ssn/ssn_r50_450e_thumos14_rgb_test.py␣
↪checkpoints/SOME_CHECKPOINT.pth --eval mAP
```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

CHAPTER 11

时空动作检测模型

11.1 ACRN

11.1.1 简介

```
@inproceedings{gu2018ava,
  title={Ava: A video dataset of spatio-temporally localized atomic visual actions},
  author={Gu, Chunhui and Sun, Chen and Ross, David A and Vondrick, Carl and␣
↪Pantofaru, Caroline and Li, Yeqing and Vijayanarasimhan, Sudheendra and Toderici,␣
↪George and Ricco, Susanna and Sukthankar, Rahul and others},
  booktitle={Proceedings of the IEEE Conference on Computer Vision and Pattern␣
↪Recognition},
  pages={6047--6056},
  year={2018}
}
```

```
@inproceedings{sun2018actor,
  title={Actor-centric relation network},
  author={Sun, Chen and Shrivastava, Abhinav and Vondrick, Carl and Murphy, Kevin and␣
↪Sukthankar, Rahul and Schmid, Cordelia},
  booktitle={Proceedings of the European Conference on Computer Vision (ECCV)},
  pages={318--334},
  year={2018}
}
```

11.1.2 模型库

AVA2.1

AVA2.2

- 注：

1. 这里的 **GPU 数量**指的是得到模型权重文件对应的 GPU 个数。默认地，MMAction2 所提供的配置文件对应使用 8 块 GPU 进行训练的情况。依据 线性缩放规则，当用户使用不同数量的 GPU 或者每块 GPU 处理不同视频个数时，需要根据批大小等比例地调节学习率。如，lr=0.01 对应 4 GPUs x 2 video/gpu，以及 lr=0.08 对应 16 GPUs x 4 video/gpu。

对于数据集准备的细节，用户可参考 数据准备。

11.1.3 如何训练

用户可以使用以下指令进行模型训练。

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

例如：在 AVA 数据集上训练 ACRN 辅以 SlowFast 主干网络，并定期验证。

```
python tools/train.py configs/detection/acrn/slowfast_acrn_kinetics_pretrained_r50_
↪8x8x1_cosine_10e_ava22_rgb.py --validate
```

更多训练细节，可参考 基础教程 中的 **训练配置**部分。

11.1.4 如何测试

用户可以使用以下指令进行模型测试。

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

例如：在 AVA 上测试 ACRN 辅以 SlowFast 主干网络，并将结果存为 csv 文件。

```
python tools/test.py configs/detection/acrn/slowfast_acrn_kinetics_pretrained_r50_
↪8x8x1_cosine_10e_ava22_rgb.py checkpoints/SOME_CHECKPOINT.pth --eval mAP --out␣
↪results.csv
```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

11.2 AVA

11.2.1 简介

```
@inproceedings{gu2018ava,
  title={Ava: A video dataset of spatio-temporally localized atomic visual actions},
  author={Gu, Chunhui and Sun, Chen and Ross, David A and Vondrick, Carl and␣
↪Pantofaru, Caroline and Li, Yeqing and Vijayanarasimhan, Sudheendra and Toderici,␣
↪George and Ricco, Susanna and Sukthankar, Rahul and others},
  booktitle={Proceedings of the IEEE Conference on Computer Vision and Pattern␣
↪Recognition},
  pages={6047--6056},
  year={2018}
}
```

```
@article{duan2020omni,
  title={Omni-sourced Webly-supervised Learning for Video Recognition},
  author={Duan, Haodong and Zhao, Yue and Xiong, Yuanjun and Liu, Wentao and Lin,␣
↪Dahua},
  journal={arXiv preprint arXiv:2003.13042},
  year={2020}
}
```

```
@inproceedings{feichtenhofer2019slowfast,
  title={Slowfast networks for video recognition},
  author={Feichtenhofer, Christoph and Fan, Haoqi and Malik, Jitendra and He, Kaiming}
↪,
  booktitle={Proceedings of the IEEE international conference on computer vision},
  pages={6202--6211},
  year={2019}
}
```

11.2.2 模型库

AVA2.1

AVA2.2

注:

1. 这里的 **GPU 数量**指的是得到模型权重文件对应的 GPU 个数。默认地，MMAction2 所提供的配置文件对应使用 8 块 GPU 进行训练的情况。依据 线性缩放规则，当用户使用不同数量的 GPU 或者每块 GPU

处理不同视频个数时，需要根据批大小等比例地调节学习率。如，lr=0.01 对应 4 GPUs x 2 video/gpu，以及 lr=0.08 对应 16 GPUs x 4 video/gpu。

2. **Context** 表示同时使用 RoI 特征与全局特征进行分类，可带来约 1% mAP 的提升。

对于数据集准备的细节，用户可参考 数据准备。

11.2.3 如何训练

用户可以使用以下指令进行模型训练。

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

例如：在 AVA 数据集上训练 SlowOnly，并定期验证。

```
python tools/train.py configs/detection/ava/slowonly_kinetics_pretrained_r50_8x8x1_
↪20e_ava_rgb.py --validate
```

更多训练细节，可参考 基础教程 中的 **训练配置**部分。

训练 AVA 数据集中的自定义类别

用户可以训练 AVA 数据集中的自定义类别。AVA 中不同类别的样本量很不平衡：其中有超过 100000 样本的类别：`stand/listen to (a person)/talk to (e.g., self, a person, a group)/watch (a person)`，也有样本较少的类别（半数类别不足 500 样本）。大多数情况下，仅使用样本较少的类别进行训练将在这些类别上得到更好精度。

训练 AVA 数据集中的自定义类别包含 3 个步骤：

1. 从原先的类别中选择希望训练的类别，将其填写至配置文件的 `custom_classes` 域中。其中 `0` 不表示具体的动作类别，不应被选择。

2. 将 `num_classes` 设置为 `num_classes = len(custom_classes) + 1`。

   - 在新的类别到编号的对应中，编号 `0` 仍对应原类别 `0`，编号 `i (i > 0)` 对应原类别 `custom_classes[i-1]`。
   - 配置文件中 3 处涉及 `num_classes` 需要修改：`model -> roi_head -> bbox_head -> num_classes`，`data -> train -> num_classes`，`data -> val -> num_classes`.
   - 若 `num_classes <= 5`，配置文件 `BBoxHeadAVA` 中的 `topk` 参数应被修改。`topk` 的默认值为 `(3, 5)`，`topk` 中的所有元素应小于 `num_classes`。

3. 确认所有自定义类别在 `label_file` 中。

以 `slowonly_kinetics_pretrained_r50_4x16x1_20e_ava_rgb` 为例，这一配置文件训练所有 AP 在 `(0.1, 0.3)` 间的类别（这里的 AP 为 AVA 80 类训出模型的表现），即 `[3, 6, 10, 27, 29, 38, 41, 48, 51, 53, 54, 59, 61, 64, 70, 72]`。下表列出了自定义类别训练的模型精度：

11.2.4 如何测试

用户可以使用以下指令进行模型测试。

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

例如：在 AVA 上测试 SlowOnly 模型，并将结果存为 csv 文件。

```
python tools/test.py configs/detection/ava/slowonly_kinetics_pretrained_r50_8x8x1_20e_
↪ava_rgb.py checkpoints/SOME_CHECKPOINT.pth --eval mAP --out results.csv
```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

11.3 LFB

11.3.1 简介

```
@inproceedings{wu2019long,
  title={Long-term feature banks for detailed video understanding},
  author={Wu, Chao-Yuan and Feichtenhofer, Christoph and Fan, Haoqi and He, Kaiming␣
↪and Krahenbuhl, Philipp and Girshick, Ross},
  booktitle={Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern␣
↪Recognition},
  pages={284--293},
  year={2019}
}
```

11.3.2 模型库

AVA2.1

- 注:

1. 这里的 **GPU 数量**指的是得到模型权重文件对应的 GPU 个数。默认地，MMAction2 所提供的配置文件对应使用 8 块 GPU 进行训练的情况。依据 线性缩放规则，当用户使用不同数量的 GPU 或者每块 GPU 处理不同视频个数时，需要根据批大小等比例地调节学习率。如，lr=0.01 对应 4 GPUs x 2 video/gpu，以及 lr=0.08 对应 16 GPUs x 4 video/gpu。
2. 本 LFB 模型暂没有使用原论文中的 `I3D-R50-NL` 作为主干网络，而是用 `slowonly_r50_4x16x1` 替代，但取得了同样的提升效果：（本模型：20.1 -> 24.11 而原论文模型：22.1 -> 25.8）。
3. 因为测试时，长时特征是被随机采样的，所以测试精度可能有一些偏差。

4. 在训练或测试 LFB 之前，用户需要使用配置文件特征库 lfb_slowonly_r50_ava_infer.py 来推导长时特征库。有关推导长时特征库的更多细节，请参照训练部分。
5. 用户也可以直接从 AVA_train_val_float32_lfb 或者 AVA_train_val_float16_lfb 下载 float32 或 float16 的长时特征库，并把它们放在 `lfb_prefix_path` 上。

11.3.3 训练

a. 为训练 LFB 推导长时特征库

在训练或测试 LFB 之前，用户首先需要推导长时特征库。

具体来说，使用配置文件 lfb_slowonly_r50_ava_infer，在训练集、验证集、测试集上都运行一次模型测试。

配置文件的默认设置是推导训练集的长时特征库，用户需要将 `dataset_mode` 设置成 `'val'` 来推导验证集的长时特征库，在推导过程中。共享头 LFBInferHead 会生成长时特征库。

AVA 训练集和验证集的 float32 精度的长时特征库文件大约占 3.3 GB。如果以半精度来存储长时特征，文件大约占 1.65 GB。

用户可以使用以下命令来推导 AVA 训练集和验证集的长时特征库，而特征库会被存储为 `lfb_prefix_path/lfb_train.pkl` 和 `lfb_prefix_path/lfb_val.pkl`。

```
## 在 lfb_slowonly_r50_ava_infer.py 中 设置 `dataset_mode = 'train'`
python tools/test.py configs/detection/lfb/lfb_slowonly_r50_ava_infer.py \
    checkpoints/YOUR_BASELINE_CHECKPOINT.pth --eval mAP

## 在 lfb_slowonly_r50_ava_infer.py 中 设置 `dataset_mode = 'val'`
python tools/test.py configs/detection/lfb/lfb_slowonly_r50_ava_infer.py \
    checkpoints/YOUR_BASELINE_CHECKPOINT.pth --eval mAP
```

MMAction2 使用来自配置文件 slowonly_kinetics_pretrained_r50_4x16x1_20e_ava_rgb 的模型权重文件 slowonly_r50_4x16x1 checkpoint作为推导长时特征库的 LFB 模型的主干网络的预训练模型。

b. 训练 LFB

用户可以使用以下指令进行模型训练。

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

例如：使用半精度的长时特征库在 AVA 数据集上训练 LFB 模型。

```
python tools/train.py configs/detection/lfb/lfb_nl_kinetics_pretrained_slowonly_r50_
↪4x16x1_20e_ava_rgb.py \
  --validate --seed 0 --deterministic
```

更多训练细节，可参考 基础教程 中的 **训练配置**部分。

11.3.4 测试

a. 为测试 LFB 推导长时特征库

在训练或测试 LFB 之前，用户首先需要推导长时特征库。如果用户之前已经生成了特征库文件，可以跳过这一步。

这一步做法与训练部分中的 **为训练 LFB 推导长时特征库**相同。

b. 测试 LFB

用户可以使用以下指令进行模型测试。

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

例如：使用半精度的长时特征库在 AVA 数据集上测试 LFB 模型，并将结果导出为一个 json 文件。

```
python tools/test.py configs/detection/lfb/lfb_nl_kinetics_pretrained_slowonly_r50_
↪4x16x1_20e_ava_rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --eval mAP --out results.csv
```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

CHAPTER 12

骨骼动作识别模型

12.1 AGCN

12.1.1 简介

```
@inproceedings{shi2019two,
  title={Two-stream adaptive graph convolutional networks for skeleton-based action
↪recognition},
  author={Shi, Lei and Zhang, Yifan and Cheng, Jian and Lu, Hanqing},
  booktitle={Proceedings of the IEEE/CVF conference on computer vision and pattern
↪recognition},
  pages={12026--12035},
  year={2019}
}
```

12.1.2 模型库

NTU60_XSub

12.1.3 如何训练

用户可以使用以下指令进行模型训练。

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

例如：以一个确定性的训练方式，辅以定期的验证过程进行 AGCN 模型在 NTU60 数据集的骨骼数据上的训练。

```
python tools/train.py configs/skeleton/2s-agcn/2sagcn_80e_ntu60_xsub_keypoint_3d.py \
    --work-dir work_dirs/2sagcn_80e_ntu60_xsub_keypoint_3d \
    --validate --seed 0 --deterministic
```

例如：以一个确定性的训练方式，辅以定期的验证过程进行 AGCN 模型在 NTU60 数据集的关节数据上的训练。

```
python tools/train.py configs/skeleton/2s-agcn/2sagcn_80e_ntu60_xsub_bone_3d.py \
    --work-dir work_dirs/2sagcn_80e_ntu60_xsub_bone_3d \
    --validate --seed 0 --deterministic
```

更多训练细节，可参考 基础教程 中的 **训练配置**部分。

12.1.4 如何测试

用户可以使用以下指令进行模型测试。

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

例如：在 NTU60 数据集的骨骼数据上测试 AGCN 模型，并将结果导出为一个 pickle 文件。

```
python tools/test.py configs/skeleton/2s-agcn/2sagcn_80e_ntu60_xsub_keypoint_3d.py \
    checkpoints/SOME_CHECKPOINT.pth --eval top_k_accuracy mean_class_accuracy \
    --out joint_result.pkl
```

例如：在 NTU60 数据集的关节数据上测试 AGCN 模型，并将结果导出为一个 pickle 文件。

```
python tools/test.py configs/skeleton/2s-agcn/2sagcn_80e_ntu60_xsub_bone_3d.py \
    checkpoints/SOME_CHECKPOINT.pth --eval top_k_accuracy mean_class_accuracy \
    --out bone_result.pkl
```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

12.2 PoseC3D

12.2.1 简介

```
@misc{duan2021revisiting,
      title={Revisiting Skeleton-based Action Recognition},
      author={Haodong Duan and Yue Zhao and Kai Chen and Dian Shao and Dahua Lin and
↪Bo Dai},
      year={2021},
      eprint={2104.13586},
      archivePrefix={arXiv},
      primaryClass={cs.CV}
}
```

12.2.2 模型库

FineGYM

NTU60_XSub

NTU120_XSub

UCF101

HMDB51

注：

1. 这里的 **GPU 数量**指的是得到模型权重文件对应的 GPU 个数。默认地，MMAction2 所提供的配置文件对应使用 8 块 GPU 进行训练的情况。依据 线性缩放规则，当用户使用不同数量的 GPU 或者每块 GPU 处理不同视频个数时，需要根据批大小等比例地调节学习率。如，lr=0.2 对应 8 GPUs x 16 video/gpu，以及 lr=0.4 对应 16 GPUs x 16 video/gpu。
2. 用户可以参照 准备骨骼数据集 来获取以上配置文件使用的骨骼标注。

12.2.3 如何训练

用户可以使用以下指令进行模型训练。

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

Example: 以确定性的训练，加以定期的验证过程进行 PoseC3D 模型在 FineGYM 数据集上的训练。

```
python tools/train.py configs/skeleton/posec3d/slowonly_r50_u48_240e_gym_keypoint.py \
    --work-dir work_dirs/slowonly_r50_u48_240e_gym_keypoint \
    --validate --seed 0 --deterministic
```

有关自定义数据集上的训练，可以参考 Custom Dataset Training。

更多训练细节，可参考 基础教程 中的 **训练配置**部分。

12.2.4 如何测试

用户可以使用以下指令进行模型测试。

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

Example: 在 FineGYM 数据集上测试 PoseC3D 模型，并将结果导出为一个 pickle 文件。

```
python tools/test.py configs/skeleton/posec3d/slowonly_r50_u48_240e_gym_keypoint.py \
    checkpoints/SOME_CHECKPOINT.pth --eval top_k_accuracy mean_class_accuracy \
    --out result.pkl
```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

12.3 STGCN

12.3.1 简介

```
@inproceedings{yan2018spatial,
  title={Spatial temporal graph convolutional networks for skeleton-based action␣
↪recognition},
  author={Yan, Sijie and Xiong, Yuanjun and Lin, Dahua},
  booktitle={Thirty-second AAAI conference on artificial intelligence},
  year={2018}
}
```

12.3.2 模型库

NTU60_XSub

BABEL

* 注：此数字引自原 论文，实际公开的 模型权重 精度略低一些。

12.3.3 如何训练

用户可以使用以下指令进行模型训练。

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

例如：以一个确定性的训练方式，辅以定期的验证过程进行 STGCN 模型在 NTU60 数据集上的训练

```
python tools/train.py configs/skeleton/stgcn/stgcn_80e_ntu60_xsub_keypoint.py \
    --work-dir work_dirs/stgcn_80e_ntu60_xsub_keypoint \
    --validate --seed 0 --deterministic
```

更多训练细节，可参考 基础教程 中的 **训练配置**部分。

12.3.4 如何测试

用户可以使用以下指令进行模型测试。

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

例如：在 NTU60 数据集上测试 STGCN 模型，并将结果导出为一个 pickle 文件。

```
python tools/test.py configs/skeleton/stgcn/stgcn_80e_ntu60_xsub_keypoint.py \
    checkpoints/SOME_CHECKPOINT.pth --eval top_k_accuracy mean_class_accuracy \
    --out result.pkl
```

更多测试细节，可参考 基础教程 中的 **测试某个数据集**部分。

CHAPTER 13

教程 1：如何编写配置文件

MMAction2 使用 python 文件作为配置文件。其配置文件系统的设计将模块化与继承整合进来，方便用户进行各种实验。MMAction2 提供的所有配置文件都放置在 `$MMAction2/configs` 文件夹下，用户可以通过运行命令 `python tools/analysis/print_config.py /PATH/TO/CONFIG` 来查看完整的配置信息，从而方便检查所对应的配置文件。

13.1 通过命令行参数修改配置信息

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

- 更新配置文件内的字典

  用户可以按照原始配置中的字典键顺序来指定配置文件的设置。例如，`--cfg-options model.backbone.norm_eval=False` 会改变 `train` 模式下模型主干网络 backbone 中所有的 BN 模块。

- 更新配置文件内列表的键

  配置文件中，存在一些由字典组成的列表。例如，训练数据前处理流水线 data.train.pipeline 就是 python 列表。如，`[dict(type='SampleFrames'), ...]`。如果用户想更改其中的 `'SampleFrames'` 为 `'DenseSampleFrames'`，可以指定 `--cfg-options data.train.pipeline.0.type=DenseSampleFrames`。

- 更新列表/元组的值。

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

13.2 配置文件结构

在 `config/_base_` 文件夹下存在 3 种基本组件类型：模型（model），训练策略（schedule），运行时的默认设置（default_runtime）。许多方法都可以方便地通过组合这些组件进行实现，如 TSN，I3D，SlowOnly 等。其中，通过 `_base_` 下组件来构建的配置被称为 *原始配置*（*primitive*）。

对于在同一文件夹下的所有配置文件，MMAction2 推荐只存在 **一个**对应的 *原始配置* 文件。所有其他的配置文件都应该继承 *原始配置* 文件，这样就能保证配置文件的最大继承深度为 3。

为了方便理解，MMAction2 推荐用户继承现有方法的配置文件。例如，如需修改 TSN 的配置文件，用户应先通过 `_base_ = '../tsn/tsn_r50_1x1x3_100e_kinetics400_rgb.py'` 继承 TSN 配置文件的基本结构，并修改其中必要的内容以完成继承。

如果用户想实现一个独立于任何一个现有的方法结构的新方法，则需要像 `configs/recognition`，`configs/detection` 等一样，在 `configs/TASK` 中建立新的文件夹。

更多详细内容，请参考 mmcv。

13.3 配置文件命名规则

MMAction2 按照以下风格进行配置文件命名，代码库的贡献者需要遵循相同的命名规则。

```
{model}_[model setting]_{backbone}_[misc]_{data setting}_[gpu x batch_per_gpu]_
↪{schedule}_{dataset}_{modality}
```

其中，{xxx} 表示必要的命名域，[yyy] 表示可选的命名域。

- {model}：模型类型，如 tsn，i3d 等。
- [model setting]：一些模型上的特殊设置。
- {backbone}：主干网络类型，如 r50（ResNet-50）等。
- [misc]：模型的额外设置或插件，如 dense，320p，video 等。
- {data setting}：采帧数据格式，形如 {clip_len}x{frame_interval}x{num_clips}。
- [gpu x batch_per_gpu]：GPU 数量以及每个 GPU 上的采样。
- {schedule}：训练策略设置，如 20e 表示 20 个周期（epoch）。
- {dataset}：数据集名，如 kinetics400，mmit 等。
- {modality}：帧的模态，如 rgb, flow 等。

13.3.1 时序动作检测的配置文件系统

MMAction2 将模块化设计整合到配置文件系统中，以便于执行各种不同的实验。

- 以 BMN 为例

  为了帮助用户理解 MMAction2 的配置文件结构，以及时序动作检测系统中的一些模块，这里以 BMN 为例，给出其配置文件的注释。对于每个模块的详细用法以及对应参数的选择，请参照 API 文档。

```
# 模型设置
model = dict(  # 模型的配置
    type='BMN',  # 时序动作检测器的类型
    temporal_dim=100,  # 每个视频中所选择的帧数量
    boundary_ratio=0.5,  # 视频边界的决策几率
    num_samples=32,  # 每个候选的采样数
    num_samples_per_bin=3,  # 每个样本的直方图采样数
    feat_dim=400,  # 特征维度
    soft_nms_alpha=0.4,  # soft-NMS 的 alpha 值
    soft_nms_low_threshold=0.5,  # soft-NMS 的下界
    soft_nms_high_threshold=0.9,  # soft-NMS 的上界
    post_process_top_k=100)  # 后处理得到的最好的 K 个 proposal
# 模型训练和测试的设置
```

(下页继续)

(续上页)

```
train_cfg = None  # 训练 BMN 的超参配置
test_cfg = dict(average_clips='score')  # 测试 BMN 的超参配置

# 数据集设置
dataset_type = 'ActivityNetDataset'  # 训练，验证，测试的数据集类型
data_root = 'data/activitynet_feature_cuhk/csv_mean_100/'  # 训练集的根目录
data_root_val = 'data/activitynet_feature_cuhk/csv_mean_100/'  # 验证集和测试集的根目
录
ann_file_train = 'data/ActivityNet/anet_anno_train.json'  # 训练集的标注文件
ann_file_val = 'data/ActivityNet/anet_anno_val.json'  # 验证集的标注文件
ann_file_test = 'data/ActivityNet/anet_anno_test.json'  # 测试集的标注文件

train_pipeline = [  # 训练数据前处理流水线步骤组成的列表
    dict(type='LoadLocalizationFeature'),  # 加载时序动作检测特征
    dict(type='GenerateLocalizationLabels'),  # 生成时序动作检测标签
    dict(  # Collect 类的配置
        type='Collect',  # Collect 类决定哪些键会被传递到时序检测器中
        keys=['raw_feature', 'gt_bbox'],  # 输入的键
        meta_name='video_meta',  # 元名称
        meta_keys=['video_name']),  # 输入的元键
    dict(  # ToTensor 类的配置
        type='ToTensor',  # ToTensor 类将其他类型转化为 Tensor 类型
        keys=['raw_feature']),  # 将被从其他类型转化为 Tensor 类型的特征
    dict(  # ToDataContainer 类的配置
        type='ToDataContainer',  # 将一些信息转入到 ToDataContainer 中
        fields=[dict(key='gt_bbox', stack=False, cpu_only=True)])  # 携带额外键和属性
的信息域
]
val_pipeline = [  # 验证数据前处理流水线步骤组成的列表
    dict(type='LoadLocalizationFeature'),  # 加载时序动作检测特征
    dict(type='GenerateLocalizationLabels'),  # 生成时序动作检测标签
    dict(  # Collect 类的配置
        type='Collect',  # Collect 类决定哪些键会被传递到时序检测器中
        keys=['raw_feature', 'gt_bbox'],  # 输入的键
        meta_name='video_meta',  # 元名称
        meta_keys=[
            'video_name', 'duration_second', 'duration_frame', 'annotations',
            'feature_frame'
        ]),  # 输入的元键
    dict(  # ToTensor 类的配置
        type='ToTensor',  # ToTensor 类将其他类型转化为 Tensor 类型
        keys=['raw_feature']),  # 将被从其他类型转化为 Tensor 类型的特征
    dict(  # ToDataContainer 类的配置
```

(下页继续)

(续上页)

```
        type='ToDataContainer',  # 将一些信息转入到 ToDataContainer 中
        fields=[dict(key='gt_bbox', stack=False, cpu_only=True)])  # 携带额外键和属性
的信息域
]
test_pipeline = [  # 测试数据前处理流水线步骤组成的列表
    dict(type='LoadLocalizationFeature'),  # 加载时序动作检测特征
    dict(  # Collect 类的配置
        type='Collect',  # Collect 类决定哪些键会被传递到时序检测器中
        keys=['raw_feature'],  # 输入的键
        meta_name='video_meta',  # 元名称
        meta_keys=[
            'video_name', 'duration_second', 'duration_frame', 'annotations',
            'feature_frame'
        ]),  # 输入的元键
    dict(  # ToTensor 类的配置
        type='ToTensor',  # ToTensor 类将其他类型转化为 Tensor 类型
        keys=['raw_feature']),  # 将被从其他类型转化为 Tensor 类型的特征
]
data = dict(  # 数据的配置
    videos_per_gpu=8,  # 单个 GPU 的批大小
    workers_per_gpu=8,  # 单个 GPU 的 dataloader 的进程
    train_dataloader=dict(  # 训练过程 dataloader 的额外设置
        drop_last=True),  # 在训练过程中是否丢弃最后一个批次
    val_dataloader=dict(  # 验证过程 dataloader 的额外设置
        videos_per_gpu=1),  # 单个 GPU 的批大小
    test_dataloader=dict(  # 测试过程 dataloader 的额外设置
        videos_per_gpu=2),  # 单个 GPU 的批大小
    test=dict(  # 测试数据集的设置
        type=dataset_type,
        ann_file=ann_file_test,
        pipeline=test_pipeline,
        data_prefix=data_root_val),
    val=dict(  # 验证数据集的设置
        type=dataset_type,
        ann_file=ann_file_val,
        pipeline=val_pipeline,
        data_prefix=data_root_val),
    train=dict(  # 训练数据集的设置
        type=dataset_type,
        ann_file=ann_file_train,
        pipeline=train_pipeline,
        data_prefix=data_root))
```

(下页继续)

(续上页)

```
# 优化器设置
optimizer = dict(
    # 构建优化器的设置，支持:
    # (1) 所有 PyTorch 原生的优化器，这些优化器的参数和 PyTorch 对应的一致;
    # (2) 自定义的优化器，这些优化器在 `constructor` 的基础上构建。
    # 更多细节可参考 "tutorials/5_new_modules.md" 部分
    type='Adam',  # 优化器类型，参考 https://github.com/open-mmlab/mmcv/blob/master/
↪mmcv/runner/optimizer/default_constructor.py#L13 for more details
    lr=0.001,  # 学习率，参数的细节使用可参考 PyTorch 的对应文档
    weight_decay=0.0001)  # Adam 优化器的权重衰减
optimizer_config = dict(  # 用于构建优化器钩子的设置
    grad_clip=None)  # 大部分的方法不使用梯度裁剪
# 学习策略设置
lr_config = dict(  # 用于注册学习率调整钩子的设置
    policy='step',  # 调整器策略，支持 CosineAnnealing，Cyclic 等方法。更多细节可参考␣
↪https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/lr_updater.py
↪#L9
    step=7)  # 学习率衰减步长

total_epochs = 9  # 训练模型的总周期数
checkpoint_config = dict(  # 模型权重文件钩子设置，更多细节可参考 https://github.com/
↪open-mmlab/mmcv/blob/master/mmcv/runner/hooks/checkpoint.py
    interval=1)  # 模型权重文件保存间隔
evaluation = dict(  # 训练期间做验证的设置
    interval=1,  # 执行验证的间隔
    metrics=['AR@AN'])  # 验证方法
log_config = dict(  # 注册日志钩子的设置
    interval=50,  # 打印日志间隔
    hooks=[  # 训练期间执行的钩子
        dict(type='TextLoggerHook'),  # 记录训练过程信息的日志
        # dict(type='TensorboardLoggerHook'),  # 同时支持 Tensorboard 日志
    ])

# 运行设置
dist_params = dict(backend='nccl')  # 建立分布式训练的设置（端口号，多 GPU 通信框架等）
log_level = 'INFO'  # 日志等级
work_dir = './work_dirs/bmn_400x100_2x8_9e_activitynet_feature/'  # 记录当前实验日志
和模型权重文件的文件夹
load_from = None  # 从给定路径加载模型作为预训练模型. 这个选项不会用于断点恢复训练
resume_from = None  # 加载给定路径的模型权重文件作为断点续连的模型，训练将从该时间点保存的周期
点继续进行
workflow = [('train', 1)]  # runner 的执行流. [('train', 1)] 代表只有一个执行流，并且这
个名为 train 的执行流只执行一次
```

(下页继续)

(续上页)

```
output_config = dict(  # 时序检测器输出设置
    out=f'{work_dir}/results.json',  # 输出文件路径
    output_format='json')  # 输出文件格式
```

13.3.2 动作识别的配置文件系统

MMAction2 将模块化设计整合到配置文件系统中，以便执行各类不同实验。

- 以 TSN 为例

  为了帮助用户理解 MMAction2 的配置文件结构，以及动作识别系统中的一些模块，这里以 TSN 为例，给出其配置文件的注释。对于每个模块的详细用法以及对应参数的选择，请参照 API 文档。

```
# 模型设置
model = dict(  # 模型的配置
    type='Recognizer2D',  # 动作识别器的类型
    backbone=dict(  # Backbone 字典设置
        type='ResNet',  # Backbone 名
        pretrained='torchvision://resnet50',  # 预训练模型的 url 或文件位置
        depth=50,  # ResNet 模型深度
        norm_eval=False),  # 训练时是否设置 BN 层为验证模式
    cls_head=dict(  # 分类器字典设置
        type='TSNHead',  # 分类器名
        num_classes=400,  # 分类类别数量
        in_channels=2048,  # 分类器里输入通道数
        spatial_type='avg',  # 空间维度的池化种类
        consensus=dict(type='AvgConsensus', dim=1),  # consensus 模块设置
        dropout_ratio=0.4,  # dropout 层概率
        init_std=0.01), # 线性层初始化 std 值
        # 模型训练和测试的设置
    train_cfg=None,  # 训练 TSN 的超参配置
    test_cfg=dict(average_clips=None))  # 测试 TSN 的超参配置

# 数据集设置
dataset_type = 'RawframeDataset'  # 训练，验证，测试的数据集类型
data_root = 'data/kinetics400/rawframes_train/'  # 训练集的根目录
data_root_val = 'data/kinetics400/rawframes_val/'  # 验证集，测试集的根目录
ann_file_train = 'data/kinetics400/kinetics400_train_list_rawframes.txt'  # 训练集
的标注文件
ann_file_val = 'data/kinetics400/kinetics400_val_list_rawframes.txt'  # 验证集的标注
文件
ann_file_test = 'data/kinetics400/kinetics400_val_list_rawframes.txt'  # 测试集的标
注文件
```

(下页继续)

(续上页)

```
img_norm_cfg = dict(  # 图像正则化参数设置
    mean=[123.675, 116.28, 103.53],  # 图像正则化平均值
    std=[58.395, 57.12, 57.375],  # 图像正则化方差
    to_bgr=False)  # 是否将通道数从 RGB 转为 BGR

train_pipeline = [  # 训练数据前处理流水线步骤组成的列表
    dict(  # SampleFrames 类的配置
        type='SampleFrames',  # 选定采样哪些视频帧
        clip_len=1,  # 每个输出视频片段的帧
        frame_interval=1,  # 所采相邻帧的时序间隔
        num_clips=3),  # 所采帧片段的数量
    dict(  # RawFrameDecode 类的配置
        type='RawFrameDecode'),  # 给定帧序列，加载对应帧，解码对应帧
    dict(  # Resize 类的配置
        type='Resize',  # 调整图片尺寸
        scale=(-1, 256)),  # 调整比例
    dict(  # MultiScaleCrop 类的配置
        type='MultiScaleCrop',  # 多尺寸裁剪，随机从一系列给定尺寸中选择一个比例尺寸进行裁剪
        input_size=224,  # 网络输入
        scales=(1, 0.875, 0.75, 0.66),  # 长宽比例选择范围
        random_crop=False,  # 是否进行随机裁剪
        max_wh_scale_gap=1),  # 长宽最大比例间隔
    dict(  # Resize 类的配置
        type='Resize',  # 调整图片尺寸
        scale=(224, 224),  # 调整比例
        keep_ratio=False),  # 是否保持长宽比
    dict(  # Flip 类的配置
        type='Flip',  # 图片翻转
        flip_ratio=0.5),  # 执行翻转几率
    dict(  # Normalize 类的配置
        type='Normalize',  # 图片正则化
        **img_norm_cfg),  # 图片正则化参数
    dict(  # FormatShape 类的配置
        type='FormatShape',  # 将图片格式转变为给定的输入格式
        input_format='NCHW'),  # 最终的图片组成格式
    dict(  # Collect 类的配置
        type='Collect',  # Collect 类决定哪些键会被传递到行为识别器中
        keys=['imgs', 'label'],  # 输入的键
        meta_keys=[]),  # 输入的元键
    dict(  # ToTensor 类的配置
        type='ToTensor',  # ToTensor 类将其他类型转化为 Tensor 类型
        keys=['imgs', 'label'])  # 将被从其他类型转化为 Tensor 类型的特征
]
```

(下页继续)

(续上页)

```
val_pipeline = [  # 验证数据前处理流水线步骤组成的列表
    dict(  # SampleFrames 类的配置
        type='SampleFrames',  # 选定采样哪些视频帧
        clip_len=1,  # 每个输出视频片段的帧
        frame_interval=1,  # 所采相邻帧的时序间隔
        num_clips=3,  # 所采帧片段的数量
        test_mode=True),  # 是否设置为测试模式采帧
    dict(  # RawFrameDecode 类的配置
        type='RawFrameDecode'),  # 给定帧序列，加载对应帧，解码对应帧
    dict(  # Resize 类的配置
        type='Resize',  # 调整图片尺寸
        scale=(-1, 256)),  # 调整比例
    dict(  # CenterCrop 类的配置
        type='CenterCrop',  # 中心裁剪
        crop_size=224),  # 裁剪部分的尺寸
    dict(  # Flip 类的配置
        type='Flip',  # 图片翻转
        flip_ratio=0),  # 翻转几率
    dict(  # Normalize 类的配置
        type='Normalize',  # 图片正则化
        **img_norm_cfg),  # 图片正则化参数
    dict(  # FormatShape 类的配置
        type='FormatShape',  # 将图片格式转变为给定的输入格式
        input_format='NCHW'),  # 最终的图片组成格式
    dict(  # Collect 类的配置
        type='Collect',  # Collect 类决定哪些键会被传递到行为识别器中
        keys=['imgs', 'label'],  # 输入的键
        meta_keys=[]),  # 输入的元键
    dict(  # ToTensor 类的配置
        type='ToTensor',  # ToTensor 类将其他类型转化为 Tensor 类型
        keys=['imgs'])  # 将被从其他类型转化为 Tensor 类型的特征
]
test_pipeline = [  # 测试数据前处理流水线步骤组成的列表
    dict(  # SampleFrames 类的配置
        type='SampleFrames',  # 选定采样哪些视频帧
        clip_len=1,  # 每个输出视频片段的帧
        frame_interval=1,  # 所采相邻帧的时序间隔
        num_clips=25,  # 所采帧片段的数量
        test_mode=True),  # 是否设置为测试模式采帧
    dict(  # RawFrameDecode 类的配置
        type='RawFrameDecode'),  # 给定帧序列，加载对应帧，解码对应帧
    dict(  # Resize 类的配置
        type='Resize',  # 调整图片尺寸
```

(下页继续)

(续上页)

```
        scale=(-1, 256)),  # 调整比例
    dict(  # TenCrop 类的配置
        type='TenCrop',  # 裁剪 10 个区域
        crop_size=224),  # 裁剪部分的尺寸
    dict(  # Flip 类的配置
        type='Flip',  # 图片翻转
        flip_ratio=0),  # 执行翻转几率
    dict(  # Normalize 类的配置
        type='Normalize',  # 图片正则化
        **img_norm_cfg),  # 图片正则化参数
    dict(  # FormatShape 类的配置
        type='FormatShape',  # 将图片格式转变为给定的输入格式
        input_format='NCHW'),  # 最终的图片组成格式
    dict(  # Collect 类的配置
        type='Collect',  # Collect 类决定哪些键会被传递到行为识别器中
        keys=['imgs', 'label'],  # 输入的键
        meta_keys=[]),  # 输入的元键
    dict(  # ToTensor 类的配置
        type='ToTensor',  # ToTensor 类将其他类型转化为 Tensor 类型
        keys=['imgs'])  # 将被从其他类型转化为 Tensor 类型的特征
]
data = dict(  # 数据的配置
    videos_per_gpu=32,  # 单个 GPU 的批大小
    workers_per_gpu=2,  # 单个 GPU 的 dataloader 的进程
    train_dataloader=dict(  # 训练过程 dataloader 的额外设置
        drop_last=True),  # 在训练过程中是否丢弃最后一个批次
    val_dataloader=dict(  # 验证过程 dataloader 的额外设置
        videos_per_gpu=1),  # 单个 GPU 的批大小
    test_dataloader=dict(  # 测试过程 dataloader 的额外设置
        videos_per_gpu=2),  # 单个 GPU 的批大小
    train=dict(  # 训练数据集的设置
        type=dataset_type,
        ann_file=ann_file_train,
        data_prefix=data_root,
        pipeline=train_pipeline),
    val=dict(  # 验证数据集的设置
        type=dataset_type,
        ann_file=ann_file_val,
        data_prefix=data_root_val,
        pipeline=val_pipeline),
    test=dict(  # 测试数据集的设置
        type=dataset_type,
        ann_file=ann_file_test,
```

(下页继续)

(续上页)

```
        data_prefix=data_root_val,
        pipeline=test_pipeline))
# 优化器设置
optimizer = dict(
    # 构建优化器的设置，支持:
    # (1) 所有 PyTorch 原生的优化器，这些优化器的参数和 PyTorch 对应的一致;
    # (2) 自定义的优化器，这些优化器在 `constructor` 的基础上构建。
    # 更多细节可参考 "tutorials/5_new_modules.md" 部分
    type='SGD',  # 优化器类型，参考 https://github.com/open-mmlab/mmcv/blob/master/
↪mmcv/runner/optimizer/default_constructor.py#L13
    lr=0.01,  # 学习率，参数的细节使用可参考 PyTorch 的对应文档
    momentum=0.9,  # 动量大小
    weight_decay=0.0001)  # SGD 优化器权重衰减
optimizer_config = dict(  # 用于构建优化器钩子的设置
    grad_clip=dict(max_norm=40, norm_type=2))  # 使用梯度裁剪
# 学习策略设置
lr_config = dict(  # 用于注册学习率调整钩子的设置
    policy='step',  # 调整器策略，支持 CosineAnnealing，Cyclic 等方法。更多细节可参考␣
↪https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/lr_updater.py
↪#L9
    step=[40, 80])  # 学习率衰减步长
total_epochs = 100  # 训练模型的总周期数
checkpoint_config = dict(  # 模型权重钩子设置，更多细节可参考 https://github.com/open-
↪mmlab/mmcv/blob/master/mmcv/runner/hooks/checkpoint.py
    interval=5)  # 模型权重文件保存间隔
evaluation = dict(  # 训练期间做验证的设置
    interval=5,  # 执行验证的间隔
    metrics=['top_k_accuracy', 'mean_class_accuracy'],  # 验证方法
    save_best='top1_acc')  # 设置 `top1_acc` 作为指示器，用于存储最好的模型权重文件
log_config = dict(  # 注册日志钩子的设置
    interval=20,  # 打印日志间隔
    hooks=[  # 训练期间执行的钩子
        dict(type='TextLoggerHook'),  # 记录训练过程信息的日志
        # dict(type='TensorboardLoggerHook'),  # 同时支持 Tensorboard 日志
    ])

# 运行设置
dist_params = dict(backend='nccl')  # 建立分布式训练的设置，其中端口号也可以设置
log_level = 'INFO'  # 日志等级
work_dir = './work_dirs/tsn_r50_1x1x3_100e_kinetics400_rgb/'  # 记录当前实验日志和模型
权重文件的文件夹
load_from = None  # 从给定路径加载模型作为预训练模型. 这个选项不会用于断点恢复训练
resume_from = None  # 加载给定路径的模型权重文件作为断点续连的模型，训练将从该时间点保存的周期
点继续进行
```

(下页继续)

(续上页)

```
workflow = [('train', 1)]  # runner 的执行流. [('train', 1)] 代表只有一个执行流，并且这
个名为 train 的执行流只执行一次
```

13.3.3 时空动作检测的配置文件系统

MMAction2 将模块化设计整合到配置文件系统中，以便于执行各种不同的实验。

- 以 FastRCNN 为例

  为了帮助用户理解 MMAction2 的完整配置文件结构，以及时空检测系统中的一些模块，这里以 FastRCNN 为例，给出其配置文件的注释。对于每个模块的详细用法以及对应参数的选择，请参照 API 文档。

```
# 模型设置
model = dict(  # 模型的配置
    type='FastRCNN',  # 时空检测器类型
    backbone=dict(  # Backbone 字典设置
        type='ResNet3dSlowOnly',  # Backbone 名
        depth=50, # ResNet 模型深度
        pretrained=None,    # 预训练模型的 url 或文件位置
        pretrained2d=False, # 预训练模型是否为 2D 模型
        lateral=False,  # backbone 是否有侧连接
        num_stages=4, # ResNet 模型阶数
        conv1_kernel=(1, 7, 7), # Conv1 卷积核尺寸
        conv1_stride_t=1, # Conv1 时序步长
        pool1_stride_t=1, # Pool1 时序步长
        spatial_strides=(1, 2, 2, 1)),  # 每个 ResNet 阶的空间步长
    roi_head=dict(  # roi_head 字典设置
        type='AVARoIHead',  # roi_head 名
        bbox_roi_extractor=dict(  # bbox_roi_extractor 字典设置
            type='SingleRoIExtractor3D',  # bbox_roi_extractor 名
            roi_layer_type='RoIAlign',  # RoI op 类型
            output_size=8,  # RoI op 输出特征尺寸
            with_temporal_pool=True), # 时序维度是否要经过池化
        bbox_head=dict( # bbox_head 字典设置
            type='BBoxHeadAVA', # bbox_head 名
            in_channels=2048, # 输入特征通道数
            num_classes=81, # 动作类别数 + 1（背景）
            multilabel=True,  # 数据集是否多标签
            dropout_ratio=0.5)),  # dropout 比率
    # 模型训练和测试的设置
    train_cfg=dict(  # 训练 FastRCNN 的超参配置
```

(下页继续)

(续上页)

```
        rcnn=dict(  # rcnn 训练字典设置
            assigner=dict(  # assigner 字典设置
                type='MaxIoUAssignerAVA', # assigner 名
                pos_iou_thr=0.9,  # 正样本 IoU 阈值, > pos_iou_thr -> positive
                neg_iou_thr=0.9,  # 负样本 IoU 阈值, < neg_iou_thr -> negative
                min_pos_iou=0.9), # 正样本最小可接受 IoU
            sampler=dict( # sample 字典设置
                type='RandomSampler', # sampler 名
                num=32, # sampler 批大小
                pos_fraction=1, # sampler 正样本边界框比率
                neg_pos_ub=-1,  # 负样本数转正样本数的比率上界
                add_gt_as_proposals=True), # 是否添加 ground truth 为候选
            pos_weight=1.0, # 正样本 loss 权重
            debug=False)), # 是否为 debug 模式
    test_cfg=dict( # 测试 FastRCNN 的超参设置
        rcnn=dict(  # rcnn 测试字典设置
            action_thr=0.002))) # 某行为的阈值

# 数据集设置
dataset_type = 'AVADataset' # 训练, 验证, 测试的数据集类型
data_root = 'data/ava/rawframes'  # 训练集的根目录
anno_root = 'data/ava/annotations'  # 标注文件目录

ann_file_train = f'{anno_root}/ava_train_v2.1.csv'  # 训练集的标注文件
ann_file_val = f'{anno_root}/ava_val_v2.1.csv'  # 验证集的标注文件

exclude_file_train = f'{anno_root}/ava_train_excluded_timestamps_v2.1.csv'  # 训练
除外数据集文件路径
exclude_file_val = f'{anno_root}/ava_val_excluded_timestamps_v2.1.csv'  # 验证除外数
据集文件路径

label_file = f'{anno_root}/ava_action_list_v2.1_for_activitynet_2018.pbtxt'  # 标签
文件路径

proposal_file_train = f'{anno_root}/ava_dense_proposals_train.FAIR.recall_93.9.pkl
↪'  # 训练样本检测候选框的文件路径
proposal_file_val = f'{anno_root}/ava_dense_proposals_val.FAIR.recall_93.9.pkl'
↪# 验证样本检测候选框的文件路径

img_norm_cfg = dict(  # 图像正则化参数设置
    mean=[123.675, 116.28, 103.53], # 图像正则化平均值
    std=[58.395, 57.12, 57.375],    # 图像正则化方差
    to_bgr=False) # 是否将通道数从 RGB 转为 BGR
```

(下页继续)

(续上页)

```
train_pipeline = [  # 训练数据前处理流水线步骤组成的列表
    dict(  # SampleFrames 类的配置
        type='AVASampleFrames',  # 选定采样哪些视频帧
        clip_len=4,  # 每个输出视频片段的帧
        frame_interval=16), # 所采相邻帧的时序间隔
    dict(  # RawFrameDecode 类的配置
        type='RawFrameDecode'),  # 给定帧序列，加载对应帧，解码对应帧
    dict(  # RandomRescale 类的配置
        type='RandomRescale',    # 给定一个范围，进行随机短边缩放
        scale_range=(256, 320)),    # RandomRescale 的短边缩放范围
    dict(  # RandomCrop 类的配置
        type='RandomCrop',    # 给定一个尺寸进行随机裁剪
        size=256),    # 裁剪尺寸
    dict(  # Flip 类的配置
        type='Flip',  # 图片翻转
        flip_ratio=0.5),  # 执行翻转几率
    dict(  # Normalize 类的配置
        type='Normalize',  # 图片正则化
        **img_norm_cfg),  # 图片正则化参数
    dict(  # FormatShape 类的配置
        type='FormatShape',  # 将图片格式转变为给定的输入格式
        input_format='NCTHW',  # 最终的图片组成格式
        collapse=True),    # 去掉 N 梯度当 N == 1
    dict(  # Rename 类的配置
        type='Rename',  # 重命名 key 名
        mapping=dict(imgs='img')),  # 改名映射字典
    dict(  # ToTensor 类的配置
        type='ToTensor',  # ToTensor 类将其他类型转化为 Tensor 类型
        keys=['img', 'proposals', 'gt_bboxes', 'gt_labels']),  # 将被从其他类型转化
为 Tensor 类型的特征
    dict(  # ToDataContainer 类的配置
        type='ToDataContainer',  # 将一些信息转入到 ToDataContainer 中
        fields=[   # 转化为 Datacontainer 的域
            dict(   # 域字典
                key=['proposals', 'gt_bboxes', 'gt_labels'],  # 将转化为␣
↪DataContainer 的键
                stack=False)]),  # 是否要堆列这些 tensor
    dict(  # Collect 类的配置
        type='Collect',  # Collect 类决定哪些键会被传递到时空检测器中
        keys=['img', 'proposals', 'gt_bboxes', 'gt_labels'],  # 输入的键
        meta_keys=['scores', 'entity_ids']),  # 输入的元键
]
```

(下页继续)

(续上页)

```
val_pipeline = [  # 验证数据前处理流水线步骤组成的列表
    dict(  # SampleFrames 类的配置
        type='AVASampleFrames',  # 选定采样哪些视频帧
        clip_len=4,  # 每个输出视频片段的帧
        frame_interval=16),  # 所采相邻帧的时序间隔
    dict(  # RawFrameDecode 类的配置
        type='RawFrameDecode'),  # 给定帧序列，加载对应帧，解码对应帧
    dict(  # Resize 类的配置
        type='Resize',  # 调整图片尺寸
        scale=(-1, 256)),  # 调整比例
    dict(  # Normalize 类的配置
        type='Normalize',  # 图片正则化
        **img_norm_cfg),  # 图片正则化参数
    dict(  # FormatShape 类的配置
        type='FormatShape',  # 将图片格式转变为给定的输入格式
        input_format='NCTHW',  # 最终的图片组成格式
        collapse=True),  # 去掉 N 梯度当 N == 1
    dict(  # Rename 类的配置
        type='Rename',  # 重命名 key 名
        mapping=dict(imgs='img')),  # 改名映射字典
    dict(  # ToTensor 类的配置
        type='ToTensor',  # ToTensor 类将其他类型转化为 Tensor 类型
        keys=['img', 'proposals']),  # 将被从其他类型转化为 Tensor 类型的特征
    dict(  # ToDataContainer 类的配置
        type='ToDataContainer',  # 将一些信息转入到 ToDataContainer 中
        fields=[  # 转化为 Datacontainer 的域
            dict(  # 域字典
                key=['proposals'],  # 将转化为 DataContainer 的键
                stack=False)]),  # 是否要堆列这些 tensor
    dict(  # Collect 类的配置
        type='Collect',  # Collect 类决定哪些键会被传递到时空检测器中
        keys=['img', 'proposals'],  # 输入的键
        meta_keys=['scores', 'entity_ids'],  # 输入的元键
        nested=True)  # 是否将数据包装为嵌套列表
]

data = dict(  # 数据的配置
    videos_per_gpu=16,  # 单个 GPU 的批大小
    workers_per_gpu=2,  # 单个 GPU 的 dataloader 的进程
    val_dataloader=dict(  # 验证过程 dataloader 的额外设置
        videos_per_gpu=1),  # 单个 GPU 的批大小
    train=dict(  # 训练数据集的设置
```

(下页继续)

(续上页)

```
        type=dataset_type,
        ann_file=ann_file_train,
        exclude_file=exclude_file_train,
        pipeline=train_pipeline,
        label_file=label_file,
        proposal_file=proposal_file_train,
        person_det_score_thr=0.9,
        data_prefix=data_root),
    val=dict(      # 验证数据集的设置
        type=dataset_type,
        ann_file=ann_file_val,
        exclude_file=exclude_file_val,
        pipeline=val_pipeline,
        label_file=label_file,
        proposal_file=proposal_file_val,
        person_det_score_thr=0.9,
        data_prefix=data_root))
data['test'] = data['val']     # 将验证数据集设置复制到测试数据集设置

# 优化器设置
optimizer = dict(
    # 构建优化器的设置，支持:
    # (1) 所有 PyTorch 原生的优化器，这些优化器的参数和 PyTorch 对应的一致;
    # (2) 自定义的优化器，这些优化器在 `constructor` 的基础上构建。
    # 更多细节可参考 "tutorials/5_new_modules.md" 部分
    type='SGD',  # 优化器类型，参考 https://github.com/open-mmlab/mmcv/blob/master/
↪mmcv/runner/optimizer/default_constructor.py#L13
    lr=0.2,  # 学习率，参数的细节使用可参考 PyTorch 的对应文档
    momentum=0.9,  # 动量大小
    weight_decay=0.00001)  # SGD 优化器权重衰减

optimizer_config = dict(  # 用于构建优化器钩子的设置
    grad_clip=dict(max_norm=40, norm_type=2))    # 使用梯度裁剪

lr_config = dict(  # 用于注册学习率调整钩子的设置
    policy='step',  # 调整器策略，支持 CosineAnnealing，Cyclic 等方法。更多细节可参考␣
↪https://github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/lr_updater.py
↪#L9
    step=[40, 80],  # 学习率衰减步长
    warmup='linear',  # Warmup 策略
    warmup_by_epoch=True,  # Warmup 单位为 epoch 还是 iteration
    warmup_iters=5,   # warmup 数
    warmup_ratio=0.1)   # 初始学习率为 warmup_ratio * lr
```

(下页继续)

(续上页)

```
total_epochs = 20  # 训练模型的总周期数
checkpoint_config = dict(  # 模型权重文件钩子设置，更多细节可参考 https://github.com/
↪open-mmlab/mmcv/blob/master/mmcv/runner/hooks/checkpoint.py
    interval=1)  # 模型权重文件保存间隔
workflow = [('train', 1)]  # runner 的执行流. [('train', 1)] 代表只有一个执行流，并且这
个名为 train 的执行流只执行一次
evaluation = dict(  # 训练期间做验证的设置
    interval=1, save_best='mAP@0.5IOU')  # 执行验证的间隔，以及设置 `mAP@0.5IOU` 作为指
示器，用于存储最好的模型权重文件
log_config = dict(  # 注册日志钩子的设置
    interval=20,  # 打印日志间隔
    hooks=[  # 训练期间执行的钩子
        dict(type='TextLoggerHook'),  # 记录训练过程信息的日志
    ])

# 运行设置
dist_params = dict(backend='nccl')  # 建立分布式训练的设置，其中端口号也可以设置
log_level = 'INFO'  # 日志等级
work_dir = ('./work_dirs/ava/'  # 记录当前实验日志和模型权重文件的文件夹
            'slowonly_kinetics_pretrained_r50_4x16x1_20e_ava_rgb')
load_from = ('https://download.openmmlab.com/mmaction/recognition/slowonly/'  # 从
给定路径加载模型作为预训练模型. 这个选项不会用于断点恢复训练
             'slowonly_r50_4x16x1_256e_kinetics400_rgb/'
             'slowonly_r50_4x16x1_256e_kinetics400_rgb_20200704-a69556c6.pth')
resume_from = None  # 加载给定路径的模型权重文件作为断点续连的模型，训练将从该时间点保存的周期
点继续进行
```

13.4 常见问题

13.4.1 配置文件中的中间变量

配置文件中会用到一些中间变量，如 train_pipeline/val_pipeline/test_pipeline, ann_file_train/ann_file_val/ann_file_test, img_norm_cfg 等。

例如，首先定义中间变量 train_pipeline/val_pipeline/test_pipeline，再将上述变量传递到 data。因此，train_pipeline/val_pipeline/test_pipeline 为中间变量

这里也定义了 ann_file_train/ann_file_val/ann_file_test 和 data_root/data_root_val 为数据处理流程提供一些基本信息。

此外，使用 img_norm_cfg 作为中间变量，构建一些数组增强组件。

```
...
dataset_type = 'RawframeDataset'
data_root = 'data/kinetics400/rawframes_train'
data_root_val = 'data/kinetics400/rawframes_val'
ann_file_train = 'data/kinetics400/kinetics400_train_list_rawframes.txt'
ann_file_val = 'data/kinetics400/kinetics400_val_list_rawframes.txt'
ann_file_test = 'data/kinetics400/kinetics400_val_list_rawframes.txt'

img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_bgr=False)

train_pipeline = [
    dict(type='SampleFrames', clip_len=32, frame_interval=2, num_clips=1),
    dict(type='RawFrameDecode'),
    dict(type='Resize', scale=(-1, 256)),
    dict(
        type='MultiScaleCrop',
        input_size=224,
        scales=(1, 0.8),
        random_crop=False,
        max_wh_scale_gap=0),
    dict(type='Resize', scale=(224, 224), keep_ratio=False),
    dict(type='Flip', flip_ratio=0.5),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='FormatShape', input_format='NCTHW'),
    dict(type='Collect', keys=['imgs', 'label'], meta_keys=[]),
    dict(type='ToTensor', keys=['imgs', 'label'])
]
val_pipeline = [
    dict(
        type='SampleFrames',
        clip_len=32,
        frame_interval=2,
        num_clips=1,
        test_mode=True),
    dict(type='RawFrameDecode'),
    dict(type='Resize', scale=(-1, 256)),
    dict(type='CenterCrop', crop_size=224),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='FormatShape', input_format='NCTHW'),
    dict(type='Collect', keys=['imgs', 'label'], meta_keys=[]),
    dict(type='ToTensor', keys=['imgs'])
]
test_pipeline = [
```

(下页继续)

(续上页)

```
    dict(
        type='SampleFrames',
        clip_len=32,
        frame_interval=2,
        num_clips=10,
        test_mode=True),
    dict(type='RawFrameDecode'),
    dict(type='Resize', scale=(-1, 256)),
    dict(type='ThreeCrop', crop_size=256),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='FormatShape', input_format='NCTHW'),
    dict(type='Collect', keys=['imgs', 'label'], meta_keys=[]),
    dict(type='ToTensor', keys=['imgs'])
]

data = dict(
    videos_per_gpu=8,
    workers_per_gpu=2,
    train=dict(
        type=dataset_type,
        ann_file=ann_file_train,
        data_prefix=data_root,
        pipeline=train_pipeline),
    val=dict(
        type=dataset_type,
        ann_file=ann_file_val,
        data_prefix=data_root_val,
        pipeline=val_pipeline),
    test=dict(
        type=dataset_type,
        ann_file=ann_file_val,
        data_prefix=data_root_val,
        pipeline=test_pipeline))
```

CHAPTER 14

教程 2：如何微调模型

本教程介绍如何使用预训练模型在其他数据集上进行微调。

14.1 概要

对新数据集上的模型进行微调需要进行两个步骤：

1. 增加对新数据集的支持。详情请见教程 3：如何增加新数据集
2. 修改配置文件。这部分将在本教程中做具体讨论。

例如，如果用户想要微调 Kinetics-400 数据集的预训练模型到另一个数据集上，如 UCF101，则需要注意配置文件 中 Head、数据集、训练策略、预训练模型四个部分，下面分别介绍。

14.2 修改 Head

`cls_head` 中的 `num_classes` 参数需改为新数据集中的类别数。预训练模型中，除了最后一层外的权重都会被重新利用，因此这个改动是安全的。例如，UCF101 拥有 101 类行为，因此需要把 400 (Kinetics-400 的类别数) 改为 101。

```
model = dict(
    type='Recognizer2D',
    backbone=dict(
        type='ResNet',
        pretrained='torchvision://resnet50',
        depth=50,
        norm_eval=False),
    cls_head=dict(
        type='TSNHead',
        num_classes=101,    # 从 400 改为 101
        in_channels=2048,
        spatial_type='avg',
        consensus=dict(type='AvgConsensus', dim=1),
        dropout_ratio=0.4,
        init_std=0.01),
    train_cfg=None,
    test_cfg=dict(average_clips=None))
```

其中，`pretrained='torchvision://resnet50'` 表示通过 ImageNet 预训练权重初始化 backbone。然而，模型微调时的预训练权重一般通过 `load_from`（而不是 `pretrained`）指定。

14.3 修改数据集

MMAction2 支持 UCF101, Kinetics-400, Moments in Time, Multi-Moments in Time, THUMOS14, Something-Something V1&V2, ActivityNet 等数据集。用户可将自建数据集转换已有数据集格式。对动作识别任务来讲，MMAction2 提供了 `RawframeDataset` 和 `VideoDataset` 等通用的数据集读取类，数据集格式相对简单。以 `UCF101` 和 `RawframeDataset` 为例，

```
# 数据集设置
dataset_type = 'RawframeDataset'
data_root = 'data/ucf101/rawframes_train/'
data_root_val = 'data/ucf101/rawframes_val/'
ann_file_train = 'data/ucf101/ucf101_train_list.txt'
ann_file_val = 'data/ucf101/ucf101_val_list.txt'
ann_file_test = 'data/ucf101/ucf101_val_list.txt'
```

14.4 修改训练策略

通常情况下，设置较小的学习率，微调模型少量训练批次，即可取得较好效果。

```
# 优化器
optimizer = dict(type='SGD', lr=0.005, momentum=0.9, weight_decay=0.0001)  # 从 0.01 改
为 0.005
optimizer_config = dict(grad_clip=dict(max_norm=40, norm_type=2))
# 学习策略
lr_config = dict(policy='step', step=[20, 40]) # step 与 total_epoch 相适应
total_epochs = 50 # 从 100 改为 50
checkpoint_config = dict(interval=5)
```

14.5 使用预训练模型

若要将预训练模型用于整个网络（主干网络设置中的 pretrained，仅会在主干网络模型上加载预训练参数），可通过 load_from 指定模型文件路径或模型链接，实现预训练权重导入。MMAction2 在 configs/_base_/default_runtime.py 文件中将 load_from=None 设为默认。由于配置文件的可继承性，用户可直接在下游配置文件中设置 load_from 的值来进行更改。

```
# 将预训练模型用于整个 TSN 网络
load_from = 'https://open-mmlab.s3.ap-northeast-2.amazonaws.com/mmaction/mmaction-v1/
↪recognition/tsn_r50_1x1x3_100e_kinetics400_rgb/tsn_r50_1x1x3_100e_kinetics400_rgb_
↪20200614-e508be42.pth'  # 模型路径可以在 model zoo 中找到
```

CHAPTER 15

教程 3：如何增加新数据集

在本教程中，我们将介绍一些有关如何按已支持的数据格式进行数据组织，和组合已有数据集来自定义数据集的方法。

15.1 通过重组数据来自定义数据集

15.1.1 将数据集重新组织为现有格式

最简单的方法是将数据集转换为现有的数据集格式（RawframeDataset 或 VideoDataset）。

有三种标注文件：

- 帧标注（rawframe annotation）

  帧数据集（rawframe dataset）标注文件由多行文本组成，每行代表一个样本，每个样本分为三个部分，分别是 帧（相对）文件夹（rawframe directory of relative path），总帧数（total frames）以及 标签（label），

通过空格进行划分

示例如下：

```
some/directory-1 163 1
some/directory-2 122 1
some/directory-3 258 2
some/directory-4 234 2
some/directory-5 295 3
some/directory-6 121 3
```

- 视频标注（video annotation）

  视频数据集（video dataset）标注文件由多行文本组成，每行代表一个样本，每个样本分为两个部分，分别是 文件（相对）路径（filepath of relative path）和 标签（label），通过空格进行划分

  示例如下：

```
some/path/000.mp4 1
some/path/001.mp4 1
some/path/002.mp4 2
some/path/003.mp4 2
some/path/004.mp4 3
some/path/005.mp4 3
```

- ActivityNet 标注

  ActivityNet 数据集的标注文件是一个 json 文件。每个键是一个视频名，其对应的值是这个视频的元数据和注释。

  示例如下：

```
{
  "video1": {
      "duration_second": 211.53,
      "duration_frame": 6337,
      "annotations": [
          {
              "segment": [
                  30.025882995319815,
                  205.2318595943838
              ],
              "label": "Rock climbing"
          }
      ],
      "feature_frame": 6336,
      "fps": 30.0,
```

(下页继续)

(续上页)

```
        "rfps": 29.9579255898
    },
    "video2": {
        "duration_second": 26.75,
        "duration_frame": 647,
        "annotations": [
            {
                "segment": [
                    2.578755070202808,
                    24.914101404056165
                ],
                "label": "Drinking beer"
            }
        ],
        "feature_frame": 624,
        "fps": 24.0,
        "rfps": 24.1869158879
    }
}
```

有两种使用自定义数据集的方法:

- 在线转换

  用户可以通过继承 BaseDataset 基类编写一个新的数据集类，并重写三个抽象类方法: `load_annotations(self)`, `evaluate(self, results, metrics, logger)` 和 `dump_results(self, results, out)`，如 RawframeDataset，VideoDataset 或 ActivityNetDataset。

- 本地转换

  用户可以转换标注文件格式为上述期望的格式，并将其存储为 pickle 或 json 文件，然后便可以应用于 `RawframeDataset`，`VideoDataset` 或 `ActivityNetDataset` 中。

数据预处理后，用户需要进一步修改配置文件以使用数据集。这里展示了以帧形式使用自定义数据集的例子:

在 `configs/task/method/my_custom_config.py` 下:

```
...
# 数据集设定
dataset_type = 'RawframeDataset'
data_root = 'path/to/your/root'
data_root_val = 'path/to/your/root_val'
ann_file_train = 'data/custom/custom_train_list.txt'
ann_file_val = 'data/custom/custom_val_list.txt'
ann_file_test = 'data/custom/custom_val_list.txt'
...
```

(下页继续)

(续上页)

```
data = dict(
    videos_per_gpu=32,
    workers_per_gpu=2,
    train=dict(
        type=dataset_type,
        ann_file=ann_file_train,
        ...),
    val=dict(
        type=dataset_type,
        ann_file=ann_file_val,
        ...),
    test=dict(
        type=dataset_type,
        ann_file=ann_file_test,
        ...))
...
```

15.1.2 自定义数据集的示例

假设注释在文本文件中以新格式显示，并且图像文件名具有类似“img_00005.jpg”的模板。那么视频注释将以以下形式存储在文本文件 `annotation.txt` 中。

```
# 文件夹, 总帧数, 类别
D32_1gwq35E,299,66
-G-5CJ0JkKY,249,254
T4h1bvOd9DA,299,33
4uZ27ivBl00,299,341
0LfESFkfBSw,249,186
-YIsNpBEx6c,299,169
```

在 `mmaction/datasets/my_dataset.py` 中创建新数据集加载数据

```
import copy
import os.path as osp

import mmcv

from .base import BaseDataset
from .builder import DATASETS


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

(下页继续)

(续上页)

```
    def __init__(self,
                 ann_file,
                 pipeline,
                 data_prefix=None,
                 test_mode=False,
                 filename_tmpl='img_{:05}.jpg'):
        super(MyDataset, self).__init__(ann_file, pipeline, test_mode)

        self.filename_tmpl = filename_tmpl

    def load_annotations(self):
        video_infos = []
        with open(self.ann_file, 'r') as fin:
            for line in fin:
                if line.startswith("directory"):
                    continue
                frame_dir, total_frames, label = line.split(',')
                if self.data_prefix is not None:
                    frame_dir = osp.join(self.data_prefix, frame_dir)
                video_infos.append(
                    dict(
                        frame_dir=frame_dir,
                        total_frames=int(total_frames),
                        label=int(label)))
        return video_infos

    def prepare_train_frames(self, idx):
        results = copy.deepcopy(self.video_infos[idx])
        results['filename_tmpl'] = self.filename_tmpl
        return self.pipeline(results)

    def prepare_test_frames(self, idx):
        results = copy.deepcopy(self.video_infos[idx])
        results['filename_tmpl'] = self.filename_tmpl
        return self.pipeline(results)

    def evaluate(self,
                 results,
                 metrics='top_k_accuracy',
                 topk=(1, 5),
                 logger=None):
        pass
```

然后在配置文件中，用户可通过如下修改来使用 MyDataset：

```
dataset_A_train = dict(
    type='MyDataset',
    ann_file=ann_file_train,
    pipeline=train_pipeline
)
```

15.2 通过组合已有数据集来自定义数据集

MMAction2 还支持组合已有数据集以进行训练。目前，它支持重复数据集（repeat dataset）。

15.2.1 重复数据集

MMAction2 使用 “RepeatDataset” 作为包装器来重复数据集。例如，假设原始数据集为 “Dataset_A”，为了重复此数据集，可设置配置如下：

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

CHAPTER 16

教程 4：如何设计数据处理流程

在本教程中，我们将介绍一些有关数据前处理流水线设计的方法，以及如何为项目自定义和扩展自己的数据流水线。

16.1 数据前处理流水线设计

按照惯例，MMAction2 使用 `Dataset` 和 `DataLoader` 实现多进程数据加载。`Dataset` 返回一个字典，作为模型的输入。由于动作识别和时序动作检测的数据大小不一定相同（图片大小，边界框大小等），MMAction2 使用 MMCV 中的 `DataContainer` 收集和分配不同大小的数据，详情可见 这里。

“数据前处理流水线”和“数据集构建”是相互解耦的。通常，“数据集构建”定义如何处理标注文件，“数据前处理流水线”定义数据加载、预处理、格式化等功能（后文将详细介绍）。数据前处理流水线由一系列相互解耦的操作组成。每个操作都输入一个字典（dict），新增/更新/删除相关字段，最终输出该字典，作为下一个操作的输入。

我们在下图中展示了一个典型的流水线。蓝色块是流水线操作。随着流水线的深入，每个操作都可以向结果字典添加新键（标记为绿色）或更新现有键（标记为橙色）。

这些操作分为数据加载，数据预处理和数据格式化。

这里以 TSN 的数据前处理流水线为例：

```
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_bgr=False)
train_pipeline = [
    dict(type='SampleFrames', clip_len=1, frame_interval=1, num_clips=3),
    dict(type='RawFrameDecode', io_backend='disk'),
    dict(type='Resize', scale=(-1, 256)),
    dict(
        type='MultiScaleCrop',
        input_size=224,
        scales=(1, 0.875, 0.75, 0.66),
        random_crop=False,
        max_wh_scale_gap=1),
    dict(type='Resize', scale=(224, 224), keep_ratio=False),
    dict(type='Flip', flip_ratio=0.5),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='FormatShape', input_format='NCHW'),
    dict(type='Collect', keys=['imgs', 'label'], meta_keys=[]),
    dict(type='ToTensor', keys=['imgs', 'label'])
]
val_pipeline = [
    dict(
        type='SampleFrames',
        clip_len=1,
        frame_interval=1,
        num_clips=3,
        test_mode=True),
    dict(type='RawFrameDecode', io_backend='disk'),
    dict(type='Resize', scale=(-1, 256)),
    dict(type='CenterCrop', crop_size=224),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='FormatShape', input_format='NCHW'),
    dict(type='Collect', keys=['imgs', 'label'], meta_keys=[]),
    dict(type='ToTensor', keys=['imgs'])
]
test_pipeline = [
    dict(
        type='SampleFrames',
```

(下页继续)

(续上页)

```
        clip_len=1,
        frame_interval=1,
        num_clips=25,
        test_mode=True),
    dict(type='RawFrameDecode', io_backend='disk'),
    dict(type='Resize', scale=(-1, 256)),
    dict(type='TenCrop', crop_size=224),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='FormatShape', input_format='NCHW'),
    dict(type='Collect', keys=['imgs', 'label'], meta_keys=[]),
    dict(type='ToTensor', keys=['imgs'])
]
```

MMAction2 也支持一些 lazy 操作符。Lazy 操作记录如何处理数据，但是它会推迟对原始数据的处理，直到进入 Fuse 阶段。具体而言，lazy 操作符避免了对原始数据的频繁读取和修改操作，只在最后的 Fuse 阶段中对原始数据进行了一次处理，从而加快了数据预处理速度，因此，推荐用户使用本功能。

这是使用 lazy 运算符的数据前处理流水线的例子：

```
train_pipeline = [
    dict(type='SampleFrames', clip_len=32, frame_interval=2, num_clips=1),
    dict(type='RawFrameDecode', decoding_backend='turbojpeg'),
    # 以下三个 lazy 操作符仅处理帧的 bbox 而不修改原始数据。
    dict(type='Resize', scale=(-1, 256), lazy=True),
    dict(
        type='MultiScaleCrop',
        input_size=224,
        scales=(1, 0.8),
        random_crop=False,
        max_wh_scale_gap=0,
        lazy=True),
    dict(type='Resize', scale=(224, 224), keep_ratio=False, lazy=True),
    # lazy 操作符 "Flip" 仅记录是否应该翻转框架和翻转方向。
    dict(type='Flip', flip_ratio=0.5, lazy=True),
    # 在 Fuse 阶段处理一次原始数据
    dict(type='Fuse'),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='FormatShape', input_format='NCTHW'),
    dict(type='Collect', keys=['imgs', 'label'], meta_keys=[]),
    dict(type='ToTensor', keys=['imgs', 'label'])
]
```

本节将所有操作分为数据加载、数据预处理、数据格式化三类，列出每个操作新增/更新/删除的相关字典字段，其中 * 代表所对应的键值不一定会被影响。

16.1.1 数据加载

`SampleFrames`

- 新增: frame_inds, clip_len, frame_interval, num_clips, *total_frames

`DenseSampleFrames`

- 新增: frame_inds, clip_len, frame_interval, num_clips, *total_frames

`PyAVDecode`

- 新增: imgs, original_shape
- 更新: *frame_inds

`DecordDecode`

- 新增: imgs, original_shape
- 更新: *frame_inds

`OpenCVDecode`

- 新增: imgs, original_shape
- 更新: *frame_inds

`RawFrameDecode`

- 新增: imgs, original_shape
- 更新: *frame_inds

16.1.2 数据预处理

`RandomCrop`

- 新增: crop_bbox, img_shape
- 更新: imgs

`RandomResizedCrop`

- 新增: crop_bbox, img_shape
- 更新: imgs

`MultiScaleCrop`

- 新增: crop_bbox, img_shape, scales
- 更新: imgs

`Resize`

- 新增: img_shape, keep_ratio, scale_factor

- 更新: imgs

`Flip`

- 新增: flip, flip_direction
- 更新: imgs, label

`Normalize`

- 新增: img_norm_cfg
- 更新: imgs

`CenterCrop`

- 新增: crop_bbox, img_shape
- 更新: imgs

`ThreeCrop`

- 新增: crop_bbox, img_shape
- 更新: imgs

`TenCrop`

- 新增: crop_bbox, img_shape
- 更新: imgs

16.1.3 数据格式化

`ToTensor`

- 更新: specified by `keys`.

`ImageToTensor`

- 更新: specified by `keys`.

`Transpose`

- 更新: specified by `keys`.

`Collect`

- 新增: img_metas (所有需要的图像元数据，会被在此阶段整合进 `meta_keys` 键值中)
- 删除: 所有没有被整合进 `keys` 的键值

值得注意的是，第一个键，通常是 `imgs`，会作为主键用来计算批大小。

`FormatShape`

- 新增: input_shape

- 更新: imgs

16.2 扩展和使用自定义流水线

1. 在任何文件写入一个新的处理流水线，如 my_pipeline.py。它以一个字典作为输入并返回一个字典

```
from mmaction.datasets import PIPELINES

@PIPELINES.register_module()
class MyTransform:

    def __call__(self, results):
        results['key'] = value
        return results
```

2. 导入新类

```
from .my_pipeline import MyTransform
```

3. 在配置文件使用它

```
img_norm_cfg = dict(
     mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
train_pipeline = [
    dict(type='DenseSampleFrames', clip_len=8, frame_interval=8, num_clips=1),
    dict(type='RawFrameDecode', io_backend='disk'),
    dict(type='MyTransform'),        # 使用自定义流水线操作
    dict(type='Normalize', **img_norm_cfg),
    dict(type='FormatShape', input_format='NCTHW'),
    dict(type='Collect', keys=['imgs', 'label'], meta_keys=[]),
    dict(type='ToTensor', keys=['imgs', 'label'])
]
```

CHAPTER 17

教程 5：如何添加新模块

在本教程中，我们将介绍一些有关如何为该项目定制优化器，开发新组件，以及添加新的学习率调整器（更新器）的方法。

17.1 自定义优化器

CopyOfSGD 是自定义优化器的一个例子，写在 `mmaction/core/optimizer/copy_of_sgd.py` 文件中。更一般地，可以根据如下方法自定义优化器。

假设添加的优化器名为 `MyOptimizer`，它有 `a`，`b` 和 `c` 三个参数。用户需要首先实现一个新的优化器文件，如 `mmaction/core/optimizer/my_optimizer.py`：

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

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

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

然后添加这个模块到 mmaction/core/optimizer/__init__.py 中，从而让注册器可以找到这个新的模块并添加它：

```
from .my_optimizer import MyOptimizer
```

之后，用户便可以在配置文件的 optimizer 字段中使用 MyOptimizer。在配置中，优化器由 optimizer 字段所定义，如下所示：

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

用户可以直接根据 PyTorch API 文档 对参数进行直接设置。

17.2 自定义优化器构造器

某些模型可能对不同层的参数有特定的优化设置，例如 BatchNorm 层的梯度衰减。用户可以通过自定义优化器构造函数来进行那些细粒度的参数调整。

用户可以编写一个基于 DefaultOptimizerConstructor 的新的优化器构造器，并且重写 add_params(self, params, module) 方法。

一个自定义优化器构造器的例子是 TSMOptimizerConstructor。更具体地，可以如下定义定制的优化器构造器。

在 mmaction/core/optimizer/my_optimizer_constructor.py：

```
from mmcv.runner import OPTIMIZER_BUILDERS, DefaultOptimizerConstructor

@OPTIMIZER_BUILDERS.register_module()
class MyOptimizerConstructor(DefaultOptimizerConstructor):

```

在 mmaction/core/optimizer/__init__.py：

```
from .my_optimizer_constructor import MyOptimizerConstructor
```

之后便可在配置文件的 optimizer 域中使用 MyOptimizerConstructor。

```
# 优化器
optimizer = dict(
    type='SGD',
    constructor='MyOptimizerConstructor',
    paramwise_cfg=dict(fc_lr5=True),
    lr=0.02,
    momentum=0.9,
    weight_decay=0.0001)
```

17.3 开发新组件

MMAction2 将模型组件分为 4 种基础模型：

- 识别器（recognizer）：整个识别器模型流水线，通常包含一个主干网络（backbone）和分类头（cls_head）。
- 主干网络（backbone）：通常为一个用于提取特征的 FCN 网络，例如 ResNet，BNInception。
- 分类头（cls_head）：用于分类任务的组件，通常包括一个带有池化层的 FC 层。
- 时序检测器（localizer）：用于时序检测的模型，目前有的检测器包含 BSN，BMN，SSN。

17.3.1 添加新的 backbones

这里以 TSN 为例，说明如何开发新的组件。

1. 创建新文件 `mmaction/models/backbones/resnet.py`

```
import torch.nn as nn

from ..builder import BACKBONES

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

    def __init__(self, arg1, arg2):
        pass

    def forward(self, x):  # 应该返回一个元组
        pass

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

2. 在 `mmaction/models/backbones/__init__.py` 中导入模型

```
from .resnet import ResNet
```

3. 在配置文件中使用它

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

17.3.2 添加新的 heads

这里以 TSNHead 为例，说明如何开发新的 head

1. 创建新文件 mmaction/models/heads/tsn_head.py

   可以通过继承 BaseHead 编写一个新的分类头，并重写 init_weights(self) 和 forward(self, x) 方法

```
from ..builder import HEADS
from .base import BaseHead


@HEADS.register_module()
class TSNHead(BaseHead):

    def __init__(self, arg1, arg2):
        pass

    def forward(self, x):
        pass

    def init_weights(self):
        pass
```

2. 在 mmaction/models/heads/__init__.py 中导入模型

```
from .tsn_head import TSNHead
```

3. 在配置文件中使用它

```
model = dict(
    ...
    cls_head=dict(
        type='TSNHead',
        num_classes=400,
        in_channels=2048,
        arg1=xxx,
        arg2=xxx),
```

17.3.3 添加新的 loss function

假设用户想添加新的 loss 为 MyLoss。为了添加一个新的损失函数，需要在 mmaction/models/losses/my_loss.py 下进行实现。

```
import torch
import torch.nn as nn

from ..builder import LOSSES

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


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

    def forward(self, pred, target):
        loss = my_loss(pred, target)
        return loss
```

之后，用户需要把它添加进 mmaction/models/losses/__init__.py

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

为了使用它，需要修改 loss_xxx 域。由于 MyLoss 用户识别任务，可以把它作为边界框损失 loss_bbox

```
loss_bbox=dict(type='MyLoss'))
```

17.3.4 添加新的学习率调节器（更新器）

构造学习率更新器（即 PyTorch 中的“scheduler”）的默认方法是修改配置，例如：

```
...
lr_config = dict(policy='step', step=[20, 40])
...
```

在 `train.py` 的 api 中，它会在以下位置注册用于学习率更新的钩子：

```
...
    runner.register_training_hooks(
        cfg.lr_config,
        optimizer_config,
        cfg.checkpoint_config,
        cfg.log_config,
        cfg.get('momentum_config', None))
...
```

到目前位置，所有支持的更新器可参考 mmcv，但如果用户想自定义学习率更新器，则需要遵循以下步骤：

1. 首先，在 `$MMAction2/mmaction/core/scheduler` 编写自定义的学习率更新钩子（LrUpdaterHook）。以下片段是自定义学习率更新器的例子，它使用基于特定比率的学习率 `lrs`，并在每个 `steps` 处进行学习率衰减。以下代码段是自定义学习率更新器的例子：

```
# 在此注册
@HOOKS.register_module()
class RelativeStepLrUpdaterHook(LrUpdaterHook):
    # 该类应当继承于 mmcv.LrUpdaterHook
    def __init__(self, steps, lrs, **kwargs):
        super().__init__(**kwargs)
        assert len(steps) == (len(lrs))
        self.steps = steps
        self.lrs = lrs

    def get_lr(self, runner, base_lr):
        # 仅需要重写该函数
        # 该函数在每个训练周期之前被调用，并返回特定的学习率.
        progress = runner.epoch if self.by_epoch else runner.iter
        for i in range(len(self.steps)):
            if progress < self.steps[i]:
                return self.lrs[i]
```

2. 修改配置

在配置文件下替换原先的 `lr_config` 变量

```
lr_config = dict(policy='RelativeStep', steps=[20, 40, 60], lrs=[0.1, 0.01, 0.001])
```

更多例子可参考 mmcv

CHAPTER 18

教程 6：如何导出模型为 onnx 格式

开放式神经网络交换格式（Open Neural Network Exchange，即 ONNX）是一个开放的生态系统，使 AI 开发人员能够随着项目的发展选择正确的工具。

18.1 支持的模型

到目前为止，MMAction2 支持将训练的 pytorch 模型中进行 onnx 导出。支持的模型有：

- I3D
- TSN
- TIN
- TSM
- R(2+1)D

- SLOWFAST
- SLOWONLY
- BMN
- BSN(tem, pem)

18.2 如何使用

对于简单的模型导出，用户可以使用这里的 脚本。注意，需要安装 onnx 和 onnxruntime 包以进行导出后的验证。

18.2.1 准备工作

首先，安装 onnx

```
pip install onnx onnxruntime
```

MMAction2 提供了一个 python 脚本，用于将 MMAction2 训练的 pytorch 模型导出到 ONNX。

```
python tools/deployment/pytorch2onnx.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--shape $
↪{SHAPE}] \
    [--verify] [--show] [--output-file ${OUTPUT_FILE}]  [--is-localizer] [--opset-
↪version ${VERSION}]
```

可选参数：

- `--shape`: 模型输入张量的形状。对于 2D 模型（如 TSN），输入形状应当为 `$batch $clip $channel $height $width`（例如，`1 1 3 224 224`）；对于 3D 模型（如 I3D），输入形状应当为 `$batch $clip $channel $time $height $width`（如，`1 1 3 32 224 224`）；对于时序检测器如 BSN，每个模块的数据都不相同，请查看对应的 `forward` 函数。如果没有被指定，它将被置为 `1 1 3 224 224`。
- `--verify`: 决定是否对导出模型进行验证，验证项包括是否可运行，数值是否正确等。如果没有被指定，它将被置为 `False`。
- `--show`: 决定是否打印导出模型的结构。如果没有被指定，它将被置为 `False`。
- `--output-file`: 导出的 onnx 模型名。如果没有被指定，它将被置为 `tmp.onnx`。
- `--is-localizer`：决定导出的模型是否为时序检测器。如果没有被指定，它将被置为 `False`。
- `--opset-version`：决定 onnx 的执行版本，MMAction2 推荐用户使用高版本（例如 11 版本）的 onnx 以确保稳定性。如果没有被指定，它将被置为 `11`。
- `--softmax`: 是否在行为识别器末尾添加 Softmax。如果没有指定，将被置为 `False`。目前仅支持行为识别器，不支持时序动作检测器。

18.2.2 行为识别器

对于行为识别器，可运行：

```
python tools/deployment/pytorch2onnx.py $CONFIG_PATH $CHECKPOINT_PATH --shape $SHAPE -
↪-verify
```

18.2.3 时序动作检测器

对于时序动作检测器，可运行：

```
python tools/deployment/pytorch2onnx.py $CONFIG_PATH $CHECKPOINT_PATH --is-localizer -
↪-shape $SHAPE --verify
```

如果发现提供的模型权重文件没有被成功导出，或者存在精度损失，可以在本 repo 下提出问题（issue）。

CHAPTER 19

教程 7：如何自定义模型运行参数

在本教程中，我们将介绍如何在运行自定义模型时，进行自定义参数优化方法，学习率调整策略，工作流和钩子的方法。

19.1 定制优化方法

19.1.1 使用 PyTorch 内置的优化器

MMAction2 支持 PyTorch 实现的所有优化器，仅需在配置文件中，指定“optimizer”字段例如，如果要使用“Adam”，则修改如下。

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

要修改模型的学习率，用户只需要在优化程序的配置中修改“lr”即可。用户可根据 PyTorch API 文档 进行参数设置

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

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

19.1.2 定制用户自定义的优化器

1. 定义一个新的优化器

一个自定义的优化器可根据如下规则进行定制

假设用户想添加一个名为 `MyOptimzer` 的优化器，其拥有参数 `a`，`b` 和 `c`，可以创建一个名为 `mmaction/core/optimizer` 的文件夹，并在目录下的文件进行构建，如 `mmaction/core/optimizer/my_optimizer.py`：

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


@OPTIMIZERS.register_module()
```

(下页继续)

(续上页)

```
class MyOptimizer(Optimizer):

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

2. 注册优化器

要找到上面定义的上述模块，首先应将此模块导入到主命名空间中。有两种方法可以实现它。

- 修改 mmaction/core/optimizer/__init__.py 来进行调用

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

```
from .my_optimizer import MyOptimizer
```

- 在配置中使用 custom_imports 手动导入

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

mmaction.core.optimizer.my_optimizer 模块将会在程序开始阶段被导入，MyOptimizer 类会随之自动被注册。注意，只有包含 MyOptmizer 类的包会被导入。mmaction.core.optimizer.my_optimizer.MyOptimizer **不会**被直接导入。

3. 在配置文件中指定优化器

之后，用户便可在配置文件的 optimizer 域中使用 MyOptimizer。在配置中，优化器由 “optimizer” 字段定义，如下所示：

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

要使用自定义的优化器，可以将该字段更改为

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

19.1.3 定制优化器构造器

某些模型可能具有一些特定于参数的设置以进行优化，例如 BatchNorm 层的权重衰减。用户可以通过自定义优化器构造函数来进行那些细粒度的参数调整。

```
from mmcv.runner.optimizer import OPTIMIZER_BUILDERS


@OPTIMIZER_BUILDERS.register_module()
class MyOptimizerConstructor:

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

    def __call__(self, model):

        return my_optimizer
```

默认的优化器构造器被创建于此，可被视为新优化器构造器的模板。

19.1.4 额外设定

优化器没有实现的优化技巧（trick）可通过优化器构造函数（例如，设置按参数的学习率）或钩子来实现。下面列出了一些可以稳定训练或加快训练速度的常用设置。用户亦可通过为 MMAction2 创建 PR，发布更多设置。

- **使用梯度裁剪来稳定训练**一些模型需要使用梯度裁剪来剪辑渐变以稳定训练过程。一个例子如下：

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

- **使用动量调整来加速模型收敛** MMAction2 支持动量调整器根据学习率修改模型的动量，从而使模型收敛更快。动量调整程序通常与学习率调整器一起使用，例如，以下配置用于 3D 检测以加速收敛。更多细节可参考 CyclicLrUpdater 和 CyclicMomentumUpdater。

  ```
  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,
  ```

(下页继续)

(续上页)

```
    step_ratio_up=0.4,
)
```

19.2 定制学习率调整策略

在配置文件中使用默认值的逐步学习率调整，它调用 MMCV 中的 `StepLRHook`。此外，也支持其他学习率调整方法，如 `CosineAnnealing` 和 `Poly`。详情可见 这里

- Poly:

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

- ConsineAnnealing:

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

19.3 定制工作流

默认情况下，MMAction2 推荐用户在训练周期中使用“EvalHook”进行模型验证，也可以选择“val”工作流模型进行模型验证。

工作流是一个形如 (工作流名, 周期数) 的列表，用于指定运行顺序和周期。其默认设置为：

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

其代表要进行一轮周期的训练。有时，用户可能希望检查有关验证集中模型的某些指标（例如，损失，准确性）。在这种情况下，可以将工作流程设置为

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

从而将迭代运行 1 个训练时间和 1 个验证时间。

值得注意的是：

1. 在验证周期时不会更新模型参数。
2. 配置文件内的关键词 `total_epochs` 控制训练时期数，并且不会影响验证工作流程。

3. 工作流 `[('train', 1), ('val', 1)]` 和 `[('train', 1)]` 不会改变 `EvalHook` 的行为。因为 `EvalHook` 由 `after_train_epoch` 调用，而验证工作流只会影响 `after_val_epoch` 调用的钩子。因此，`[('train', 1), ('val', 1)]` 和 `[('train', 1)]` 的区别在于，runner 在完成每一轮训练后，会计算验证集上的损失。

19.4 定制钩子

19.4.1 定制用户自定义钩子

1. 创建一个新钩子

这里举一个在 MMAction2 中创建一个新钩子，并在训练中使用它的示例：

```
from mmcv.runner import HOOKS, Hook


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

    def __init__(self, a, b):
        pass

    def before_run(self, runner):
        pass

    def after_run(self, runner):
        pass

    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`。

2. 注册新钩子

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

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

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

```
from .my_hook import MyHook
```

- 使用配置文件中的 custom_imports 变量手动导入

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

3. 修改配置

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

还可通过 priority 参数（可选参数值包括 'NORMAL' 或 'HIGHEST'）设置钩子优先级，如下所示：

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

默认情况下，在注册过程中，钩子的优先级设置为“NORMAL”。

19.4.2 使用 MMCV 内置钩子

如果该钩子已在 MMCV 中实现，则可以直接修改配置以使用该钩子，如下所示

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

19.4.3 修改默认运行的钩子

有一些常见的钩子未通过 `custom_hooks` 注册，但在导入 MMCV 时已默认注册，它们是：

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

在这些钩子中，只有 log_config 具有“VERY_LOW”优先级，其他钩子具有“NORMAL”优先级。上述教程已经介绍了如何修改“optimizer_config”，“momentum_config”和“lr_config”。下面介绍如何使用 log_config，checkpoint_config，以及 evaluation 能做什么。

模型权重文件配置

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。

日志配置

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

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

验证配置

评估的配置将用于初始化 EvalHook。除了键 interval 外，其他参数，如 “metrics” 也将传递给 dataset.evaluate()。

```
evaluation = dict(interval=1, metrics='bbox')
```

除了训练/测试脚本外，MMAction2 还在 tools/ 目录下提供了许多有用的工具。

CHAPTER 20

目录

CHAPTER 21

日志分析

输入变量指定一个训练日志文件，可通过 `tools/analysis/analyze_logs.py` 脚本绘制 loss/top-k 曲线。本功能依赖于 `seaborn`，使用前请先通过 `pip install seaborn` 安装依赖包。

```
python tools/analysis/analyze_logs.py plot_curve ${JSON_LOGS} [--keys ${KEYS}] [--
↪title ${TITLE}] [--legend ${LEGEND}] [--backend ${BACKEND}] [--style ${STYLE}] [--
↪out ${OUT_FILE}]
```

例如:

- 绘制某日志文件对应的分类损失曲线图。

  ```
  python tools/analysis/analyze_logs.py plot_curve log.json --keys loss_cls --
  ↪legend loss_cls
  ```

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

  ```
  python tools/analysis/analyze_logs.py plot_curve log.json --keys top1_acc top5_
  ↪acc --out results.pdf
  ```

- 在同一图像内绘制两份日志文件对应的 top-1 准确率曲线图。

  ```
  python tools/analysis/analyze_logs.py plot_curve log1.json log2.json --keys top1_
  ↪acc --legend run1 run2
  ```

  用户还可以通过本工具计算平均训练速度。

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

- 计算某日志文件对应的平均训练速度。

```
python tools/analysis/analyze_logs.py cal_train_time work_dirs/some_exp/20200422_
↪153324.log.json
```

预计输出结果如下所示：

```
-----Analyze train time of work_dirs/some_exp/20200422_153324.log.json-----
slowest epoch 60, average time is 0.9736
fastest epoch 18, average time is 0.9001
time std over epochs is 0.0177
average iter time: 0.9330 s/iter
```

CHAPTER 22

模型复杂度分析

/tools/analysis/get_flops.py 是根据 flops-counter.pytorch 库改编的脚本，用于计算输入变量指定模型的 FLOPs 和参数量。

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

预计输出结果如下所示：

```
==============================
Input shape: (1, 3, 32, 340, 256)
Flops: 37.1 GMac
Params: 28.04 M
==============================
```

注意：该工具仍处于试验阶段，不保证该数字绝对正确。用户可以将结果用于简单比较，但若要在技术报告或论文中采用该结果，请仔细检查。

(1) FLOPs 与输入变量形状有关，但是模型的参数量与输入变量形状无关。2D 行为识别器的默认形状为 (1, 3, 340, 256)，3D 行为识别器的默认形状为 (1, 3, 32, 340, 256)。(2) 部分算子不参与 FLOPs 以及参数量的计算，如 GN 和一些自定义算子。更多详细信息请参考 `mmcv.cnn.get_model_complexity_info()`

CHAPTER 23

模型转换

23.1 导出 MMAction2 模型为 ONNX 格式（实验特性）

`/tools/deployment/pytorch2onnx.py` 脚本用于将模型转换为 ONNX 格式。同时，该脚本支持比较 PyTorch 模型和 ONNX 模型的输出结果，验证输出结果是否相同。本功能依赖于 `onnx` 以及 `onnxruntime`，使用前请先通过 `pip install onnx onnxruntime` 安装依赖包。请注意，可通过 `--softmax` 选项在行为识别器末尾添加 Softmax 层，从而获取 `[0, 1]` 范围内的预测结果。

- 对于行为识别模型，请运行：

```
python tools/deployment/pytorch2onnx.py $CONFIG_PATH $CHECKPOINT_PATH --shape
↪$SHAPE --verify
```

- 对于时序动作检测模型，请运行：

```
python tools/deployment/pytorch2onnx.py $CONFIG_PATH $CHECKPOINT_PATH --is-
↪localizer --shape $SHAPE --verify
```

23.2 发布模型

tools/deployment/publish_model.py 脚本用于进行模型发布前的准备工作，主要包括：

(1) 将模型的权重张量转化为 CPU 张量。(2) 删除优化器状态信息。(3) 计算模型权重文件的哈希值，并将哈希值添加到文件名后。

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

例如，

```
python tools/deployment/publish_model.py work_dirs/tsn_r50_1x1x3_100e_kinetics400_rgb/
↪latest.pth tsn_r50_1x1x3_100e_kinetics400_rgb.pth
```

最终，输出文件名为 tsn_r50_1x1x3_100e_kinetics400_rgb-{hash id}.pth。

CHAPTER 24

其他脚本

24.1 指标评价

tools/analysis/eval_metric.py 脚本通过输入变量指定配置文件，以及对应的结果存储文件，计算某一评价指标。

结果存储文件通过 tools/test.py 脚本（通过参数 --out ${RESULT_FILE} 指定）生成，保存了指定模型在指定数据集中的预测结果。

```
python tools/analysis/eval_metric.py ${CONFIG_FILE} ${RESULT_FILE} [--eval ${EVAL_
↪METRICS}] [--cfg-options ${CFG_OPTIONS}] [--eval-options ${EVAL_OPTIONS}]
```

24.2 打印完整配置

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

```
python tools/analysis/print_config.py ${CONFIG} [-h] [--options ${OPTIONS [OPTIONS...
↪]}]
```

24.3 检查视频

tools/analysis/check_videos.py 脚本利用指定视频编码器，遍历指定配置文件视频数据集中所有样本，寻找无效视频文件（文件破损或者文件不存在），并将无效文件路径保存到输出文件中。请注意，删除无效视频文件后，需要重新生成视频文件列表。

```
python tools/analysis/check_videos.py ${CONFIG} [-h] [--options OPTIONS [OPTIONS ...
↪]] [--cfg-options CFG_OPTIONS [CFG_OPTIONS ...]] [--output-file OUTPUT_FILE] [--
↪split SPLIT] [--decoder DECODER] [--num-processes NUM_PROCESSES] [--remove-
↪corrupted-videos]
```

CHAPTER 25

常见问题解答

本文这里列出了用户们遇到的一些常见问题，及相应的解决方案。如果您发现了任何社区中经常出现的问题，也有了相应的解决方案，欢迎充实本文档来帮助他人。如果本文档不包括您的问题，欢迎使用提供的 模板创建问题，还请确保您在模板中填写了所有必需的信息。

25.1 安装

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

  1. 使用 `pip uninstall mmcv` 卸载环境中已安装的 `mmcv`。
  2. 遵循 MMCV 安装文档 来安装 `mmcv-full`。

- **“OSError: MoviePy Error: creation of None failed because of the following error”**

  参照 MMAction2 安装文档

  1. 对于 Windows 用户，ImageMagick 不再被 MoviePy 自动检测，需要获取名为 `magick` 的 ImageMagick 二进制包的路径，来修改 `moviepy/config_defaults.py` 文件中的 `IMAGEMAGICK_BINARY`，如 `IMAGEMAGICK_BINARY = "C:\\Program Files\\ImageMagick_VERSION\\magick.exe"`
  2. 对于 Linux 用户，如果 ImageMagick 没有被 moviepy 检测，需要注释掉 `/etc/ImageMagick-6/policy.xml` 文件中的 `<policy domain="path" rights="none" pattern="@*" />`，即改为 `<!-- <policy domain="path" rights="none" pattern="@*" /> -->`。

- **“Please install XXCODEBASE to use XXX”**

如得到报错消息 “Please install XXCODEBASE to use XXX”，代表 MMAction2 无法从 XXCODEBASE 中 import XXX。用户可以执行对应 import 语句定位原因。一个可能的原因是，对于部分 OpenMMLAB 中的代码库，需先安装 mmcv-full 后再进行安装。

25.2 数据

- **FileNotFound 如 `No such file or directory: xxx/xxx/img_00300.jpg`**

  在 MMAction2 中，对于帧数据集，`start_index` 的默认值为 1，而对于视频数据集，`start_index` 的默认值为 0。如果 FileNotFound 错误发生于视频的第一帧或最后一帧，则需根据视频首帧（即 `xxx_00000.jpg` 或 `xxx_00001.jpg`）的偏移量，修改配置文件中数据处理流水线的 `start_index` 值。

- **如何处理数据集中传入视频的尺寸？是把所有视频调整为固定尺寸，如 “340x256”，还是把所有视频的短边调整成相同的长度（256 像素或 320 像素）？**

  从基准测试来看，总体来说，后者（把所有视频的短边调整成相同的长度）效果更好，所以 “调整尺寸为短边 256 像素” 被设置为默认的数据处理方式。用户可以在 TSN 数据基准测试 和 SlowOnly 数据基准测试 中查看相关的基准测试结果。

- **输入数据格式（视频或帧）与数据流水线不匹配，导致异常，如 `KeyError: 'total_frames'`**

  对于视频和帧，我们都有相应的流水线来处理。

  对于视频，应该在处理时首先对其进行解码。可选的解码方式，有 `DecordInit & DecordDecode`, `OpenCVInit & OpenCVDecode`, `PyAVInit & PyAVDecode` 等等。可以参照 这个例子。

  对于帧，已经事先在本地对其解码，所以使用 `RawFrameDecode` 对帧处理即可。可以参照 这个例子。

  `KeyError: 'total_frames'` 是因为错误地使用了 `RawFrameDecode` 来处理视频。当输入是视频的时候，程序是无法事先得到 `total_frame` 的。

25.3 训练

- **如何使用训练过的识别器作为主干网络的预训练模型？**

  参照 使用预训练模型，如果想对整个网络使用预训练模型，可以在配置文件中，将 `load_from` 设置为预训练模型的链接。

  如果只想对主干网络使用预训练模型，可以在配置文件中，将主干网络 `backbone` 中的 `pretrained` 设置为预训练模型的地址或链接。在训练时，预训练模型中无法与主干网络对应的参数会被忽略。

- **如何实时绘制训练集和验证集的准确率/损失函数曲线图？**

  使用 `log_config` 中的 `TensorboardLoggerHook`，如：

```
log_config=dict(
    interval=20,
    hooks=[
        dict(type='TensorboardLoggerHook')
    ]
)
```

可以参照*教程 1：如何编写配置文件*，教程 7：如何自定义模型运行参数，和 这个例子 了解更多相关内容。

- **在 batchnorm.py 中抛出错误: Expected more than 1 value per channel when training**

  BatchNorm 层要求批大小（batch size）大于 1。构建数据集时，若 `drop_last` 被设为 `False`，有时每个轮次的最后一个批次的批大小可能为 1，进而在训练时抛出错误，可以设置 `drop_last=True` 来避免该错误，如：

```
train_dataloader=dict(drop_last=True)
```

- **微调模型参数时，如何冻结主干网络中的部分参数？**

  可以参照 `def _freeze_stages()` 和 `frozen_stages`。在分布式训练和测试时，还须设置 `find_unused_parameters = True`。

  实际上，除了少数模型，如 C3D 等，用户都能通过设置 `frozen_stages` 来冻结模型参数，因为大多数主干网络继承自 `ResNet` 和 `ResNet3D`，而这两个模型都支持 `_freeze_stages()` 方法。

- **如何在配置文件中设置 `load_from` 参数以进行模型微调？**

  MMAction2 在 `configs/_base_/default_runtime.py` 文件中将 `load_from=None` 设为默认。由于配置文件的可继承性，用户可直接在下游配置文件中设置 `load_from` 的值来进行更改。

25.4 测试

- **如何将预测分值用 softmax 归一化到 [0, 1] 区间内？**

  可以通过设置 `model['test_cfg'] = dict(average_clips='prob')` 来实现。

- **如果模型太大，连一个测试样例都没法放进显存，怎么办？**

  默认情况下，3D 模型是以 `10 clips x 3 crops` 的设置进行测试的，也即采样 10 个帧片段，每帧裁剪出 3 个图像块，总计有 30 个视图。对于特别大的模型，GPU 显存可能连一个视频都放不下。对于这种情况，您可以在配置文件的 `model['test_cfg']` 中设置 `max_testing_views=n`。如此设置，在模型推理过程中，一个批只会使用 n 个视图，以节省显存。

- **如何保存测试结果？**

  测试时，用户可在运行指令中设置可选项 `--out xxx.json/pkl/yaml` 来输出结果文件，以供后续

检查。输出的测试结果顺序和测试集顺序保持一致。除此之外，MMAction2 也在 `tools/analysis/eval_metric.py` 中提供了分析工具，用于结果文件的模型评估。

25.5 部署

- **为什么由 MMAction2 转换的 ONNX 模型在转换到其他框架（如 TensorRT）时会抛出错误？**

  目前只能确保 MMAction2 中的模型与 ONNX 兼容。但是，ONNX 中的某些算子可能不受其他框架支持，例如 这个问题 中的 TensorRT。当这种情况发生时，如果 `pytorch2onnx.py` 没有出现问题，转换过去的 ONNX 模型也通过了数值检验，可以提 issue 让社区提供帮助。

CHAPTER 26

mmaction.apis

CHAPTER 27

mmaction.core

27.1 optimizer

27.2 evaluation

scheduler ^^ .. automodule:: mmaction.core.scheduler

members

CHAPTER 28

mmaction.localization

28.1 localization

CHAPTER 29

mmaction.models

CHAPTER 30

mmaction.datasets

30.1 datasets

30.2 pipelines

30.3 samplers

CHAPTER 31

mmaction.utils

CHAPTER 32

mmaction.localization

CHAPTER 33

English

CHAPTER 34

简体中文

CHAPTER 35

索引和表格

- genindex
- search