MMAction2

Release 0.24.1

MMAction2 Authors

Jun 19, 2023

CONTENTS

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

CHAPTER

ONE

INSTALLATION

We provide some tips for MMAction2 installation in this file.

1.1 Requirements

- Linux, Windows (We can successfully install mmaction2 on Windows and run inference, but we haven't tried training yet)
- Python 3.7+
- PyTorch 1.3+
- CUDA 9.2+ (If you build PyTorch from source, CUDA 9.0 is also compatible)
- GCC 5+
- mmcv 1.1.1+
- Numpy
- ffmpeg (4.2 is preferred)
- decord (optional, 0.4.1+): Install CPU version by `pip install decord==0.4.1` and install GPU version from source
- PyAV (optional): `conda install av -c conda-forge -y`
- PyTurboJPEG (optional): `pip install PyTurboJPEG`
- denseflow (optional): See here for simple install scripts.
- moviepy (optional): `pip install moviepy`. See here for official installation. **Note**(according to this issue) that:

1. For Windows users, ImageMagick will not be automatically detected by MoviePy, there is a need to modify `moviepy/config_defaults.py` file by providing the path to the ImageMagick binary called `magick`, like `IMAGEMAGICK_BINARY = "C:\\Program Files\\ImageMagick_VERSION\\magick.exe"`
2. For Linux users, there is a need to modify the `/etc/ImageMagick-6/policy.xml` file by commenting out `<policy domain="path" rights="none" pattern="@*" />` to `<!-- <policy domain="path" rights="none" pattern="@*" /> -->`, if ImageMagick is not detected by `moviepy`.

- Pillow-SIMD (optional): Install it by the following scripts.

```
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
```

Note: You need to run `pip uninstall mmcv` first if you have mmcv installed. If mmcv and mmcv-full are both installed, there will be `ModuleNotFoundError`.

1.2 Prepare environment

a. Create a conda virtual environment and activate it.

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

b. Install PyTorch and torchvision following the official instructions, e.g.,

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

Note: Make sure that your compilation CUDA version and runtime CUDA version match. You can check the supported CUDA version for precompiled packages on the PyTorch website.

`E.g.1` If you have CUDA 10.1 installed under `/usr/local/cuda` and would like to install PyTorch 1.5, you need to install the prebuilt PyTorch with CUDA 10.1.

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

`E.g.2` If you have CUDA 9.2 installed under `/usr/local/cuda` and would like to install PyTorch 1.3.1., you need to install the prebuilt PyTorch with CUDA 9.2.

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

If you build PyTorch from source instead of installing the prebuilt package, you can use more CUDA versions such as 9.0.

1.3 Install MMAction2

We recommend you to install MMAction2 with MIM.

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

MIM can automatically install OpenMMLab projects and their requirements.

Or, you can install MMAction2 manually:

a. Install mmcv-full, we recommend you to install the pre-built package as below.

```
# 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
```

mmcv-full is only compiled on PyTorch 1.x.0 because the compatibility usually holds between 1.x.0 and 1.x.1. If your PyTorch version is 1.x.1, you can install mmcv-full compiled with PyTorch 1.x.0 and it usually works well.

```
# We can ignore the micro version of PyTorch
pip install mmcv-full -f https://download.openmmlab.com/mmcv/dist/cu102/torch1.10/index.
↪html
```

See here for different versions of MMCV compatible to different PyTorch and CUDA versions.

Optionally you can choose to compile mmcv from source by the following command

```
git clone https://github.com/open-mmlab/mmcv.git
cd mmcv
MMCV_WITH_OPS=1 pip install -e .  # package mmcv-full, which contains cuda ops, will be␣
↪installed after this step
# OR pip install -e .  # package mmcv, which contains no cuda ops, will be installed␣
↪after this step
cd ..
```

Or directly run

```
pip install mmcv-full
# alternative: pip install mmcv
```

Important: You need to run `pip uninstall mmcv` first if you have mmcv installed. If mmcv and mmcv-full are both installed, there will be `ModuleNotFoundError`.

b. Clone the MMAction2 repository.

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

c. Install build requirements and then install MMAction2.

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

If you build MMAction2 on macOS, replace the last command with

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

d. Install mmdetection for spatial temporal detection tasks.

This part is **optional** if you're not going to do spatial temporal detection.

See here to install mmdetection.

Note:

1. The git commit id will be written to the version number with step b, e.g. 0.6.0+2e7045c. The version will also be saved in trained models. It is recommended that you run step b each time you pull some updates from github. If C++/CUDA codes are modified, then this step is compulsory.
2. Following the above instructions, MMAction2 is installed on `dev` mode, any local modifications made to the code will take effect without the need to reinstall it (unless you submit some commits and want to update the version number).
3. If you would like to use `opencv-python-headless` instead of `opencv-python`, you can install it before installing MMCV.
4. If you would like to use `PyAV`, you can install it with `conda install av -c conda-forge -y`.
5. Some dependencies are optional. Running `python setup.py develop` will only install the minimum runtime requirements. To use optional dependencies like `decord`, either install them with `pip install -r requirements/optional.txt` or specify desired extras when calling `pip` (e.g. `pip install -v -e .[optional]`, valid keys for the `[optional]` field are `all`, `tests`, `build`, and `optional`) like `pip install -v -e .[tests,build]`.

1.4 Install with CPU only

The code can be built for CPU only environment (where CUDA isn't available).

In CPU mode you can run the demo/demo.py for example.

1.5 Another option: Docker Image

We provide a Dockerfile to build an image.

```
# build an image with PyTorch 1.6.0, CUDA 10.1, CUDNN 7.
docker build -f ./docker/Dockerfile --rm -t mmaction2 .
```

Important: Make sure you've installed the nvidia-container-toolkit.

Run it with command:

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

1.6 A from-scratch setup script

Here is a full script for setting up MMAction2 with conda and link the dataset path (supposing that your Kinetics-400 dataset path is $KINETICS400_ROOT).

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

# install latest pytorch prebuilt with the default prebuilt CUDA version (usually the␣
↪latest)
conda install -c pytorch pytorch torchvision -y

# install the latest mmcv or mmcv-full, here we take mmcv as example
pip install mmcv

# install 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 Developing with multiple MMAction2 versions

The train and test scripts already modify the PYTHONPATH to ensure the script use the MMAction2 in the current directory.

To use the default MMAction2 installed in the environment rather than that you are working with, you can remove the following line in those scripts.

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

1.8 Verification

To verify whether MMAction2 and the required environment are installed correctly, we can run sample python codes to initialize a recognizer and inference a demo video:

```
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' # or 'cpu'
device = torch.device(device)

model = init_recognizer(config_file, device=device)
# inference the demo video
inference_recognizer(model, 'demo/demo.mp4')
```

CHAPTER

TWO

GETTING STARTED

This page provides basic tutorials about the usage of MMAction2. For installation instructions, please see *install.md*.

2.1 Datasets

It is recommended to symlink the dataset root to `$MMACTION2/data`. If your folder structure is different, you may need to change the corresponding paths in config files.

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
│   ├── kinetics400
│   │   ├── rawframes_train
│   │   ├── rawframes_val
```

(continues on next page)

(continued from previous page)

```
│   │   ├── kinetics_train_list.txt
│   │   ├── kinetics_val_list.txt
│   ├── ucf101
│   │   ├── rawframes_train
│   │   ├── rawframes_val
│   │   ├── ucf101_train_list.txt
│   │   ├── ucf101_val_list.txt
│   ├── ...
```

For more information on data preparation, please see *data_preparation.md*

For using custom datasets, please refer to *Tutorial 3: Adding New Dataset*

2.2 Inference with Pre-Trained Models

We provide testing scripts to evaluate a whole dataset (Kinetics-400, Something-Something V1&V2, (Multi-)Moments in Time, etc.), and provide some high-level apis for easier integration to other projects.

MMAction2 also supports testing with CPU. However, it will be **very slow** and should only be used for debugging on a device without GPU. To test with CPU, one should first disable all GPUs (if exist) with `export CUDA_VISIBLE_DEVICES=-1`, and then call the testing scripts directly with `python tools/test.py {OTHER_ARGS}`.

2.2.1 Test a dataset

- [x] single GPU
- [x] single node multiple GPUs
- [x] multiple node

You can use the following commands to test a dataset.

```
# single-gpu testing
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]

# multi-gpu testing
./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}]
```

Optional arguments:

- `RESULT_FILE`: Filename of the output results. If not specified, the results will not be saved to a file.
- `EVAL_METRICS`: Items to be evaluated on the results. Allowed values depend on the dataset, e.g., `top_k_accuracy`, `mean_class_accuracy` are available for all datasets in recognition,

`mmit_mean_average_precision` for Multi-Moments in Time, `mean_average_precision` for Multi-Moments in Time and HVU single category. `AR@AN` for ActivityNet, etc.

- `--gpu-collect`: If specified, recognition results will be collected using gpu communication. Otherwise, it will save the results on different gpus to `TMPDIR` and collect them by the rank 0 worker.
- `TMPDIR`: Temporary directory used for collecting results from multiple workers, available when `--gpu-collect` is not specified.
- `OPTIONS`: Custom options used for evaluation. Allowed values depend on the arguments of the `evaluate` function in dataset.
- `AVG_TYPE`: Items to average the test clips. If set to `prob`, it will apply softmax before averaging the clip scores. Otherwise, it will directly average the clip scores.
- `JOB_LAUNCHER`: Items for distributed job initialization launcher. Allowed choices are `none`, `pytorch`, `slurm`, `mpi`. Especially, if set to none, it will test in a non-distributed mode.
- `LOCAL_RANK`: ID for local rank. If not specified, it will be set to 0.
- `--onnx`: If specified, recognition results will be generated by onnx model and `CHECKPOINT_FILE` should be onnx model file path. Onnx model files are generated by `/tools/deployment/pytorch2onnx.py`. For now, multi-gpu mode and dynamic input shape mode are not supported. Please note that the output tensors of dataset and the input tensors of onnx model should share the same shape. And it is recommended to remove all test-time augmentation methods in `test_pipeline`(`ThreeCrop`, `TenCrop`, `twice_sample`, etc.)
- `--tensorrt`: If specified, recognition results will be generated by TensorRT engine and `CHECKPOINT_FILE` should be TensorRT engine file path. TensorRT engines are generated by exported onnx models and TensorRT official conversion tools. For now, multi-gpu mode and dynamic input shape mode are not supported. Please note that the output tensors of dataset and the input tensors of TensorRT engine should share the same shape. And it is recommended to remove all test-time augmentation methods in `test_pipeline`(`ThreeCrop`, `TenCrop`, `twice_sample`, etc.)

Examples:

Assume that you have already downloaded the checkpoints to the directory `checkpoints/`.

1. Test TSN on Kinetics-400 (without saving the test results) and evaluate the top-k accuracy and mean class accuracy.

```
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. Test TSN on Something-Something V1 with 8 GPUS, and evaluate the top-k accuracy.

```
./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. Test TSN on Kinetics-400 in slurm environment and evaluate the top-k accuracy

```
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. Test TSN on Something-Something V1 with onnx model and evaluate the top-k accuracy

```
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 High-level APIs for testing a video and rawframes

Here is an example of building the model and testing a given video.

```
import torch

from mmaction.apis import init_recognizer, inference_recognizer

config_file = 'configs/recognition/tsn/tsn_r50_video_inference_1x1x3_100e_kinetics400_
↪rgb.py'
# download the checkpoint from model zoo and put it in `checkpoints/`
checkpoint_file = 'checkpoints/tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.pth'

# assign the desired device.
device = 'cuda:0' # or 'cpu'
device = torch.device(device)

 # build the model from a config file and a checkpoint file
model = init_recognizer(config_file, checkpoint_file, device=device)

# test a single video and show the result:
video = 'demo/demo.mp4'
labels = 'tools/data/kinetics/label_map_k400.txt'
results = inference_recognizer(model, video)

# show the results
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])
```

Here is an example of building the model and testing with a given rawframes directory.

```
import torch

from mmaction.apis import init_recognizer, inference_recognizer

config_file = 'configs/recognition/tsn/tsn_r50_inference_1x1x3_100e_kinetics400_rgb.py'
# download the checkpoint from model zoo and put it in `checkpoints/`
checkpoint_file = 'checkpoints/tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.pth'

# assign the desired device.
device = 'cuda:0' # or 'cpu'
device = torch.device(device)

```

(continues on next page)

(continued from previous page)

```
 # build the model from a config file and a checkpoint file
model = init_recognizer(config_file, checkpoint_file, device=device)

# test rawframe directory of a single video and show the result:
video = 'SOME_DIR_PATH/'
labels = 'tools/data/kinetics/label_map_k400.txt'
results = inference_recognizer(model, video)

# show the results
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])
```

Here is an example of building the model and testing with a given video url.

```
import torch

from mmaction.apis import init_recognizer, inference_recognizer

config_file = 'configs/recognition/tsn/tsn_r50_video_inference_1x1x3_100e_kinetics400_
↪rgb.py'
# download the checkpoint from model zoo and put it in `checkpoints/`
checkpoint_file = 'checkpoints/tsn_r50_1x1x3_100e_kinetics400_rgb_20200614-e508be42.pth'

# assign the desired device.
device = 'cuda:0' # or 'cpu'
device = torch.device(device)

 # build the model from a config file and a checkpoint file
model = init_recognizer(config_file, checkpoint_file, device=device)

# test url of a single video and show the result:
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)

# show the results
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])
```

Note: We define `data_prefix` in config files and set it None as default for our provided inference configs. If the `data_prefix` is not None, the path for the video file (or rawframe directory) to get will be `data_prefix/video`.

Here, the `video` is the param in the demo scripts above. This detail can be found in `rawframe_dataset.py` and `video_dataset.py`. For example,

- When video (rawframes) path is `SOME_DIR_PATH/VIDEO.mp4` (`SOME_DIR_PATH/VIDEO_NAME/img_xxxxx.jpg`), and `data_prefix` is None in the config file, the param `video` should be `SOME_DIR_PATH/VIDEO.mp4` (`SOME_DIR_PATH/VIDEO_NAME`).
- When video (rawframes) path is `SOME_DIR_PATH/VIDEO.mp4` (`SOME_DIR_PATH/VIDEO_NAME/img_xxxxx.jpg`), and `data_prefix` is `SOME_DIR_PATH` in the config file, the param `video` should be `VIDEO.mp4` (`VIDEO_NAME`).
- When rawframes path is `VIDEO_NAME/img_xxxxx.jpg`, and `data_prefix` is None in the config file, the param `video` should be `VIDEO_NAME`.
- When passing a url instead of a local video file, you need to use OpenCV as the video decoding backend.

A notebook demo can be found in demo/demo.ipynb

2.3 Build a Model

2.3.1 Build a model with basic components

In MMAction2, model components are basically categorized as 4 types.

- recognizer: the whole recognizer model pipeline, usually contains a backbone and cls_head.
- backbone: usually an FCN network to extract feature maps, e.g., ResNet, BNInception.
- cls_head: the component for classification task, usually contains an FC layer with some pooling layers.
- localizer: the model for localization task, currently available: BSN, BMN.

Following some basic pipelines (e.g., `Recognizer2D`), the model structure can be customized through config files with no pains.

If we want to implement some new components, e.g., the temporal shift backbone structure as in TSM: Temporal Shift Module for Efficient Video Understanding, there are several things to do.

1. create a new file in `mmaction/models/backbones/resnet_tsm.py`.

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

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

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

(continues on next page)

(continued from previous page)

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

2. Import the module in `mmaction/models/backbones/__init__.py`

```
from .resnet_tsm import ResNetTSM
```

3. modify the config file from

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

to

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

2.3.2 Write a new model

To write a new recognition pipeline, you need to inherit from `BaseRecognizer`, which defines the following abstract methods.

- `forward_train()`: forward method of the training mode.
- `forward_test()`: forward method of the testing mode.

Recognizer2D and Recognizer3D are good examples which show how to do that.

2.4 Train a Model

2.4.1 Iteration pipeline

MMAction2 implements distributed training and non-distributed training, which uses `MMDistributedDataParallel` and `MMDataParallel` respectively.

We adopt distributed training for both single machine and multiple machines. Supposing that the server has 8 GPUs, 8 processes will be started and each process runs on a single GPU.

Each process keeps an isolated model, data loader, and optimizer. Model parameters are only synchronized once at the beginning. After a forward and backward pass, gradients will be allreduced among all GPUs, and the optimizer will update model parameters. Since the gradients are allreduced, the model parameter stays the same for all processes after the iteration.

2.4.2 Training setting

All outputs (log files and checkpoints) will be saved to the working directory, which is specified by `work_dir` in the config file.

By default we evaluate the model on the validation set after each epoch, you can change the evaluation interval by modifying the interval argument in the training config

```
evaluation = dict(interval=5)  # This evaluate the model per 5 epoch.
```

According to the Linear Scaling Rule, you need to set the learning rate proportional to the batch size if you use different GPUs or videos per GPU, e.g., lr=0.01 for 4 GPUs x 2 video/gpu and lr=0.08 for 16 GPUs x 4 video/gpu.

MMAction2 also supports training with CPU. However, it will be **very slow** and should only be used for debugging on a device without GPU. To train with CPU, one should first disable all GPUs (if exist) with `export CUDA_VISIBLE_DEVICES=-1`, and then call the training scripts directly with `python tools/train.py {OTHER_ARGS}`.

2.4.3 Train with a single GPU

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

If you want to specify the working directory in the command, you can add an argument `--work-dir ${YOUR_WORK_DIR}`.

2.4.4 Train with multiple GPUs

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

Optional arguments are:

- `--validate` (**strongly recommended**): Perform evaluation at every k (default value is 5, which can be modified by changing the `interval` value in `evaluation` dict in each config file) epochs during the training.
- `--test-last`: Test the final checkpoint when training is over, save the prediction to `${WORK_DIR}/last_pred.pkl`.
- `--test-best`: Test the best checkpoint when training is over, save the prediction to `${WORK_DIR}/best_pred.pkl`.
- `--work-dir ${WORK_DIR}`: Override the working directory specified in the config file.
- `--resume-from ${CHECKPOINT_FILE}`: Resume from a previous checkpoint file.
- `--gpus ${GPU_NUM}`: Number of gpus to use, which is only applicable to non-distributed training.
- `--gpu-ids ${GPU_IDS}`: IDs of gpus to use, which is only applicable to non-distributed training.
- `--seed ${SEED}`: Seed id for random state in python, numpy and pytorch to generate random numbers.
- `--deterministic`: If specified, it will set deterministic options for CUDNN backend.
- `JOB_LAUNCHER`: Items for distributed job initialization launcher. Allowed choices are `none`, `pytorch`, `slurm`, `mpi`. Especially, if set to none, it will test in a non-distributed mode.
- `LOCAL_RANK`: ID for local rank. If not specified, it will be set to 0.

Difference between `resume-from` and `load-from`: `resume-from` loads both the model weights and optimizer status, and the epoch is also inherited from the specified checkpoint. It is usually used for resuming the training process that is interrupted accidentally. `load-from` only loads the model weights and the training epoch starts from 0. It is usually used for finetuning.

Here is an example of using 8 GPUs to load TSN checkpoint.

```
./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 Train with multiple machines

If you can run MMAction2 on a cluster managed with slurm, you can use the script `slurm_train.sh`. (This script also supports single machine training.)

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

Here is an example of using 16 GPUs to train TSN on the dev partition in a slurm cluster. (use `GPUS_PER_NODE=8` to specify a single slurm cluster node with 8 GPUs.)

```
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
```

You can check slurm_train.sh for full arguments and environment variables.

If you have just multiple machines connected with ethernet, you can simply run the following commands:

On the first machine:

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

On the second machine:

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

It can be extremely slow if you do not have high-speed networking like InfiniBand.

2.4.6 Launch multiple jobs on a single machine

If you launch multiple jobs on a single machine, e.g., 2 jobs of 4-GPU training on a machine with 8 GPUs, you need to specify different ports (29500 by default) for each job to avoid communication conflict.

If you use `dist_train.sh` to launch training jobs, you can set the port in commands.

```
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
```

If you use launch training jobs with slurm, you need to modify `dist_params` in the config files (usually the 6th line from the bottom in config files) to set different communication ports.

In `config1.py`,

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

In `config2.py`,

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

Then you can launch two jobs with `config1.py` ang `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 Tutorials

Currently, we provide some tutorials for users to *learn about configs*, *finetune model*, *add new dataset*, *customize data pipelines*, *add new modules*, *export a model to ONNX* and *customize runtime settings*.

CHAPTER

THREE

DEMO

3.1 Outline

- Modify configs through script arguments: Tricks to directly modify configs through script arguments.
- *Video demo*: A demo script to predict the recognition result using a single video.
- *SpatioTemporal Action Detection Video Demo*: A demo script to predict the SpatioTemporal Action Detection result using a single video.
- *Video GradCAM Demo*: A demo script to visualize GradCAM results using a single video.
- *Webcam demo*: A demo script to implement real-time action recognition from a web camera.
- *Long Video demo*: a demo script to predict different labels using a single long video.
- *SpatioTemporal Action Detection Webcam Demo*: A demo script to implement real-time spatio-temporal action detection from a web camera.
- *Skeleton-based Action Recognition Demo*: A demo script to predict the skeleton-based action recognition result using a single video.
- *Video Structuralize Demo*: A demo script to predict the skeleton-based and rgb-based action recognition and spatio-temporal action detection result using a single video.
- *Audio Demo*: A demo script to predict the recognition result using a single audio file.

3.2 Modify configs through script arguments

When running demos using our provided scripts, you may specify `--cfg-options` to in-place modify the config.

- Update config keys of dict.

  The config options can be specified following the order of the dict keys in the original config. For example, `--cfg-options model.backbone.norm_eval=False` changes the all BN modules in model backbones to `train` mode.

- Update keys inside a list of configs.

  Some config dicts are composed as a list in your config. For example, the training pipeline `data.train.pipeline` is normally a list e.g. `[dict(type='SampleFrames'), ...]`. If you want to change `'SampleFrames'` to `'DenseSampleFrames'` in the pipeline, you may specify `--cfg-options data.train.pipeline.0.type=DenseSampleFrames`.

- Update values of list/tuples.

  If the value to be updated is a list or a tuple. For example, the config file normally sets `workflow=[('train', 1)]`. If you want to change this key, you may specify `--cfg-options workflow="[(train,1),(val,1)]"`. Note that the quotation mark " is necessary to support list/tuple data types, and that **NO** white space is allowed inside the quotation marks in the specified value.

3.3 Video demo

We provide a demo script to predict the recognition result using a single video. In order to get predict results in range `[0, 1]`, make sure to set `model['test_cfg'] = dict(average_clips='prob')` in config file.

```
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}]
```

Optional arguments:

- `--use-frames`: If specified, the demo will take rawframes as input. Otherwise, it will take a video as input.
- `DEVICE_TYPE`: Type of device to run the demo. Allowed values are cuda device like `cuda:0` or `cpu`. If not specified, it will be set to `cuda:0`.
- `FPS`: FPS value of the output video when using rawframes as input. If not specified, it will be set to 30.
- `FONT_SCALE`: Font scale of the label added in the video. If not specified, it will be 0.5.
- `FONT_COLOR`: Font color of the label added in the video. If not specified, it will be `white`.
- `TARGET_RESOLUTION`: Resolution(desired_width, desired_height) for resizing the frames before output when using a video as input. If not specified, it will be None and the frames are resized by keeping the existing aspect ratio.
- `RESIZE_ALGORITHM`: Resize algorithm used for resizing. If not specified, it will be set to `bicubic`.
- `OUT_FILE`: Path to the output file which can be a video format or gif format. If not specified, it will be set to `None` and does not generate the output file.

Examples:

Assume that you are located at `$MMACTION2` and have already downloaded the checkpoints to the directory `checkpoints/`, or use checkpoint url from `configs/` to directly load corresponding checkpoint, which will be automatically saved in `$HOME/.cache/torch/checkpoints`.

1. Recognize a video file as input by using a TSN model on cuda by default.

   ```
   # The demo.mp4 and label_map_k400.txt are both from Kinetics-400
   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. Recognize a video file as input by using a TSN model on cuda by default, loading checkpoint from url.

```
# The demo.mp4 and label_map_k400.txt are both from Kinetics-400
python demo/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 \
    demo/demo.mp4 tools/data/kinetics/label_map_k400.txt
```

3. Recognize a list of rawframes as input by using a TSN model on cpu.

```
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. Recognize a video file as input by using a TSN model and then generate an mp4 file.

```
# The demo.mp4 and label_map_k400.txt are both from Kinetics-400
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. Recognize a list of rawframes as input by using a TSN model and then generate a gif file.

```
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. Recognize a video file as input by using a TSN model, then generate an mp4 file with a given resolution and resize algorithm.

```
# The demo.mp4 and label_map_k400.txt are both from Kinetics-400
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
```

```
# The demo.mp4 and label_map_k400.txt are both from Kinetics-400
# If either dimension is set to -1, the frames are resized by keeping the existing␣
↪aspect ratio
# For --target-resolution 170 -1, original resolution (340, 256) -> target␣
↪resolution (170, 128)
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 170 -1␣
↪--resize-algorithm bilinear \
    --out-filename demo/demo_out.mp4
```

7. Recognize a video file as input by using a TSN model, then generate an mp4 file with a label in a red color and

fontscale 1.

```
# The demo.mp4 and label_map_k400.txt are both from Kinetics-400
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-scale 1 --font-
↪color red \
    --out-filename demo/demo_out.mp4
```

8. Recognize a list of rawframes as input by using a TSN model and then generate an mp4 file with 24 fps.

```
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
```

3.4 SpatioTemporal Action Detection Video Demo

We provide a demo script to predict the SpatioTemporal Action Detection result using a single video.

```
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}]
```

Optional arguments:

- `SPATIOTEMPORAL_ACTION_DETECTION_CONFIG_FILE`: The spatiotemporal action detection config file path.
- `SPATIOTEMPORAL_ACTION_DETECTION_CHECKPOINT`: The spatiotemporal action detection checkpoint URL.
- `HUMAN_DETECTION_CONFIG_FILE`: The human detection config file path.
- `HUMAN_DETECTION_CHECKPOINT`: The human detection checkpoint URL.
- `HUMAN_DETECTION_SCORE_THRE`: The score threshold for human detection. Default: 0.9.
- `ACTION_DETECTION_SCORE_THRESHOLD`: The score threshold for action detection. Default: 0.5.
- `LABEL_MAP`: The label map used. Default: `tools/data/ava/label_map.txt`.
- `DEVICE`: Type of device to run the demo. Allowed values are cuda device like `cuda:0` or `cpu`. Default: `cuda:0`.
- `OUTPUT_FILENAME`: Path to the output file which is a video format. Default: `demo/stdet_demo.mp4`.
- `PREDICT_STEPSIZE`: Make a prediction per N frames. Default: 8.

- OUTPUT_STEPSIZE: Output 1 frame per N frames in the input video. Note that PREDICT_STEPSIZE % OUTPUT_STEPSIZE == 0. Default: 4.
- OUTPUT_FPS: The FPS of demo video output. Default: 6.

Examples:

Assume that you are located at $MMACTION2 .

1. Use the Faster RCNN as the human detector, SlowOnly-8x8-R101 as the action detector. Making predictions per 8 frames, and output 1 frame per 4 frames to the output video. The FPS of the output video is 4.

```
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
```

3.5 Video GradCAM Demo

We provide a demo script to visualize GradCAM results using a single video.

```
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}]
```

- --use-frames: If specified, the demo will take rawframes as input. Otherwise, it will take a video as input.
- DEVICE_TYPE: Type of device to run the demo. Allowed values are cuda device like cuda:0 or cpu. If not specified, it will be set to cuda:0.
- FPS: FPS value of the output video when using rawframes as input. If not specified, it will be set to 30.
- OUT_FILE: Path to the output file which can be a video format or gif format. If not specified, it will be set to None and does not generate the output file.
- TARGET_LAYER_NAME: Layer name to generate GradCAM localization map.
- TARGET_RESOLUTION: Resolution(desired_width, desired_height) for resizing the frames before output when using a video as input. If not specified, it will be None and the frames are resized by keeping the existing aspect ratio.
- RESIZE_ALGORITHM: Resize algorithm used for resizing. If not specified, it will be set to bilinear.

Examples:

Assume that you are located at `$MMACTION2` and have already downloaded the checkpoints to the directory `checkpoints/`, or use checkpoint url from `configs/` to directly load corresponding checkpoint, which will be automatically saved in `$HOME/.cache/torch/checkpoints`.

1. Get GradCAM results of a I3D model, using a video file as input and then generate an gif file with 10 fps.

```
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. Get GradCAM results of a TSM model, using a video file as input and then generate an gif file, loading checkpoint from url.

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

3.6 Webcam demo

We provide a demo script to implement real-time action recognition from web camera. In order to get predict results in range `[0, 1]`, make sure to set `model.['test_cfg'] = dict(average_clips='prob')` in config file.

```
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}]
```

Optional arguments:

- `DEVICE_TYPE`: Type of device to run the demo. Allowed values are cuda device like `cuda:0` or `cpu`. If not specified, it will be set to `cuda:0`.
- `CAMERA_ID`: ID of camera device If not specified, it will be set to 0.
- `THRESHOLD`: Threshold of prediction score for action recognition. Only label with score higher than the threshold will be shown. If not specified, it will be set to 0.
- `AVERAGE_SIZE`: Number of latest clips to be averaged for prediction. If not specified, it will be set to 1.
- `DRAWING_FPS`: Upper bound FPS value of the output drawing. If not specified, it will be set to 20.
- `INFERENCE_FPS`: Upper bound FPS value of the output drawing. If not specified, it will be set to 4.

Note: If your hardware is good enough, increasing the value of `DRAWING_FPS` and `INFERENCE_FPS` will get a better experience.

Examples:

Assume that you are located at `$MMACTION2` and have already downloaded the checkpoints to the directory `checkpoints/`, or use checkpoint url from `configs/` to directly load corresponding checkpoint, which will be automatically saved in `$HOME/.cache/torch/checkpoints`.

1. Recognize the action from web camera as input by using a TSN model on cpu, averaging the score per 5 times and outputting result labels with score higher than 0.2.

```
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. Recognize the action from web camera as input by using a TSN model on cpu, averaging the score per 5 times and outputting result labels with score higher than 0.2, loading checkpoint from url.

```
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. Recognize the action from web camera as input by using a I3D model on gpu by default, averaging the score per 5 times and outputting result labels with score higher than 0.2.

```
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
```

Note: Considering the efficiency difference for users' hardware, Some modifications might be done to suit the case. Users can change:

1). `SampleFrames` step (especially the number of `clip_len` and `num_clips`) of `test_pipeline` in the config file, like `--cfg-options data.test.pipeline.0.num_clips=3`. 2). Change to the suitable Crop methods like `TenCrop`, `ThreeCrop`, `CenterCrop`, etc. in `test_pipeline` of the config file, like `--cfg-options data.test.pipeline.4.type=CenterCrop`. 3). Change the number of `--average-size`. The smaller, the faster.

3.7 Long video demo

We provide a demo script to predict different labels using a single long video. In order to get predict results in range `[0, 1]`, make sure to set `test_cfg = dict(average_clips='prob')` in config file.

```
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}]
```

Optional arguments:

- OUT_FILE: Path to the output, either video or json file
- INPUT_STEP: Input step for sampling frames, which can help to get more spare input. If not specified , it will be set to 1.
- DEVICE_TYPE: Type of device to run the demo. Allowed values are cuda device like cuda:0 or cpu. If not specified, it will be set to cuda:0.
- THRESHOLD: Threshold of prediction score for action recognition. Only label with score higher than the threshold will be shown. If not specified, it will be set to 0.01.
- STRIDE: By default, the demo generates a prediction for each single frame, which might cost lots of time. To speed up, you can set the argument STRIDE and then the demo will generate a prediction every STRIDE x sample_length frames (sample_length indicates the size of temporal window from which you sample frames, which equals to clip_len x frame_interval). For example, if the sample_length is 64 frames and you set STRIDE to 0.5, predictions will be generated every 32 frames. If set as 0, predictions will be generated for each frame. The desired value of STRIDE is (0, 1], while it also works for STRIDE > 1 (the generated predictions will be too sparse). Default: 0.
- LABEL_COLOR: Font Color of the labels in (B, G, R). Default is white, that is (256, 256, 256).
- MSG_COLOR: Font Color of the messages in (B, G, R). Default is gray, that is (128, 128, 128).

Examples:

Assume that you are located at $MMACTION2 and have already downloaded the checkpoints to the directory checkpoints/, or use checkpoint url from configs/ to directly load corresponding checkpoint, which will be automatically saved in $HOME/.cache/torch/checkpoints.

1. Predict different labels in a long video by using a TSN model on cpu, with 3 frames for input steps (that is, random sample one from each 3 frames) and outputting result labels with score higher than 0.2.

```
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. Predict different labels in a long video by using a TSN model on cpu, with 3 frames for input steps (that is, random sample one from each 3 frames) and outputting result labels with score higher than 0.2, loading checkpoint from url.

```
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. Predict different labels in a long video from web by using a TSN model on cpu, with 3 frames for input steps (that is, random sample one from each 3 frames) and outputting result labels with score higher than 0.2, loading checkpoint from url.

```
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 \
```

(continues on next page)

(continued from previous page)

```
  tools/data/kinetics/label_map_k400.txt PATH_TO_SAVED_VIDEO --input-step 3 --
↪device cpu --threshold 0.2
```

4. Predict different labels in a long video by using a I3D model on gpu, with input_step=1, threshold=0.01 as default and print the labels in cyan.

```
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 \
  --label-color 255 255 0
```

5. Predict different labels in a long video by using a I3D model on gpu and save the results as a `json` file

```
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 ./results.json
```

3.8 SpatioTemporal Action Detection Webcam Demo

We provide a demo script to implement real-time spatio-temporal action detection from a web camera.

```
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}
```

Optional arguments:

- `SPATIOTEMPORAL_ACTION_DETECTION_CONFIG_FILE`: The spatiotemporal action detection config file path.
- `SPATIOTEMPORAL_ACTION_DETECTION_CHECKPOINT`: The spatiotemporal action detection checkpoint path or URL.
- `ACTION_DETECTION_SCORE_THRESHOLD`: The score threshold for action detection. Default: 0.4.
- `HUMAN_DETECTION_CONFIG_FILE`: The human detection config file path.
- `HUMAN_DETECTION_CHECKPOINT`: The human detection checkpoint URL.

- HUMAN_DETECTION_SCORE_THRE: The score threshold for human detection. Default: 0.9.
- INPUT_VIDEO: The webcam id or video path of the source. Default: 0.
- LABEL_MAP: The label map used. Default: tools/data/ava/label_map.txt.
- DEVICE: Type of device to run the demo. Allowed values are cuda device like cuda:0 or cpu. Default: cuda:0.
- OUTPUT_FPS: The FPS of demo video output. Default: 15.
- OUTPUT_FILENAME: Path to the output file which is a video format. Default: None.
- --show: Whether to show predictions with cv2.imshow.
- DISPLAY_HEIGHT: The height of the display frame. Default: 0.
- DISPLAY_WIDTH: The width of the display frame. Default: 0. If DISPLAY_HEIGHT <= 0 and DISPLAY_WIDTH <= 0, the display frame and input video share the same shape.
- PREDICT_STEPSIZE: Make a prediction per N frames. Default: 8.
- CLIP_VIS_LENGTH: The number of the draw frames for each clip. In other words, for each clip, there are at most CLIP_VIS_LENGTH frames to be draw around the keyframe. DEFAULT: 8.

Tips to get a better experience for webcam demo:

- How to choose --output-fps?
  - --output-fps should be almost equal to read thread fps.
  - Read thread fps is printed by logger in format DEBUG:__main__:Read Thread: {duration} ms, {fps} fps
- How to choose --predict-stepsize?
  - It's related to how to choose human detector and spatio-temporval model.
  - Overall, the duration of read thread for each task should be greater equal to that of model inference.
  - The durations for read/inference are both printed by logger.
  - Larger --predict-stepsize leads to larger duration for read thread.
  - In order to fully take the advantage of computation resources, decrease the value of --predict-stepsize.

Examples:

Assume that you are located at $MMACTION2 .

1. Use the Faster RCNN as the human detector, SlowOnly-8x8-R101 as the action detector. Making predictions per 40 frames, and FPS of the output is 20. Show predictions with cv2.imshow.

```
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 \
```

(continues on next page)

(continued from previous page)

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

3.9 Skeleton-based Action Recognition Demo

We provide a demo script to predict the skeleton-based action recognition result using a single video.

```
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}
```

Optional arguments:

- `SKELETON_BASED_ACTION_RECOGNITION_CONFIG_FILE`: The skeleton-based action recognition config file path.
- `SKELETON_BASED_ACTION_RECOGNITION_CHECKPOINT`: The skeleton-based action recognition checkpoint path or URL.
- `HUMAN_DETECTION_CONFIG_FILE`: The human detection config file path.
- `HUMAN_DETECTION_CHECKPOINT`: The human detection checkpoint URL.
- `HUMAN_DETECTION_SCORE_THRE`: The score threshold for human detection. Default: 0.9.
- `HUMAN_POSE_ESTIMATION_CONFIG_FILE`: The human pose estimation config file path (trained on COCO-Keypoint).
- `HUMAN_POSE_ESTIMATION_CHECKPOINT`: The human pose estimation checkpoint URL (trained on COCO-Keypoint).
- `LABEL_MAP`: The label map used. Default: `tools/data/ava/label_map.txt`.
- `DEVICE`: Type of device to run the demo. Allowed values are cuda device like `cuda:0` or `cpu`. Default: `cuda:0`.
- `SHORT_SIDE`: The short side used for frame extraction. Default: 480.

Examples:

Assume that you are located at `$MMACTION2` .

1. Use the Faster RCNN as the human detector, HRNetw32 as the pose estimator, PoseC3D-NTURGB+D-120-Xsub-keypoint as the skeleton-based action recognizer.

```
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 \
```

(continued from previous page)

(continued from previous page)

```
    --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. Use the Faster RCNN as the human detector, HRNetw32 as the pose estimator, STGCN-NTURGB+D-60-Xsub-keypoint as the skeleton-based action recognizer.

```
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
```

3.10 Video Structuralize Demo

We provide a demo script to predict the skeleton-based and rgb-based action recognition and spatio-temporal action detection result using a single video.

```
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}] \
```

(continues on next page)

(continued from previous page)

```
    [--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]
```

Optional arguments:

- `RGB_BASED_SPATIO_TEMPORAL_ACTION_DETECTION_CONFIG_FILE`: The rgb-based spatio temoral action detection config file path.
- `RGB_BASED_SPATIO_TEMPORAL_ACTION_DETECTION_CHECKPOINT`: The rgb-based spatio temoral action detection checkpoint path or URL.
- `SKELETON_BASED_SPATIO_TEMPORAL_ACTION_DETECTION_CHECKPOINT`: The skeleton-based spatio temoral action detection checkpoint path or URL.
- `HUMAN_DETECTION_CONFIG_FILE`: The human detection config file path.
- `HUMAN_DETECTION_CHECKPOINT`: The human detection checkpoint URL.
- `HUMAN_POSE_ESTIMATION_CONFIG_FILE`: The human pose estimation config file path (trained on COCO-Keypoint).
- `HUMAN_POSE_ESTIMATION_CHECKPOINT`: The human pose estimation checkpoint URL (trained on COCO-Keypoint).
- `SKELETON_BASED_ACTION_RECOGNITION_CONFIG_FILE`: The skeleton-based action recognition config file path.
- `SKELETON_BASED_ACTION_RECOGNITION_CHECKPOINT`: The skeleton-based action recognition checkpoint path or URL.
- `RGB_BASED_ACTION_RECOGNITION_CONFIG_FILE`: The rgb-based action recognition config file path.
- `RGB_BASED_ACTION_RECOGNITION_CHECKPOINT`: The rgb-based action recognition checkpoint path or URL.
- `USE_SKELETON_BASED_SPATIO_TEMPORAL_DETECTION_METHOD`: Use skeleton-based spatio temporal action detection method.
- `USE_SKELETON_BASED_ACTION_RECOGNITION_METHOD`: Use skeleton-based action recognition method.
- `HUMAN_DETECTION_SCORE_THRE`: The score threshold for human detection. Default: 0.9.
- `ACTION_DETECTION_SCORE_THRE`: The score threshold for action detection. Default: 0.4.
- `LABEL_MAP_FOR_SPATIO_TEMPORAL_ACTION_DETECTION`: The label map for spatio temporal action detection used. Default: `tools/data/ava/label_map.txt`.
- `LABEL_MAP`: The label map for action recognition. Default: `tools/data/kinetics/label_map_k400.txt`.
- `DEVICE`: Type of device to run the demo. Allowed values are cuda device like `cuda:0` or `cpu`. Default: `cuda:0`.
- `OUTPUT_FILENAME`: Path to the output file which is a video format. Default: `demo/test_stdet_recognition_output.mp4`.
- `PREDICT_STEPSIZE`: Make a prediction per N frames. Default: 8.
- `OUTPUT_STEPSIZE`: Output 1 frame per N frames in the input video. Note that `PREDICT_STEPSIZE % OUTPUT_STEPSIZE == 0`. Default: 1.
- `OUTPUT_FPS`: The FPS of demo video output. Default: 24.

Examples:

Assume that you are located at `$MMACTION2` .

1. Use the Faster RCNN as the human detector, HRNetw32 as the pose estimator, PoseC3D as the skeleton-based action recognizer and the skeleton-based spatio temporal action detector. Making action detection predictions per 8 frames, and output 1 frame per 1 frame to the output video. The FPS of the output video is 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 \
    --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. Use the Faster RCNN as the human detector, TSN-R50-1x1x3 as the rgb-based action recognizer, SlowOnly-8x8-R101 as the rgb-based spatio temporal action detector. Making action detection predictions per 8 frames, and output 1 frame per 1 frame to the output video. The FPS of the output video is 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. Use the Faster RCNN as the human detector, HRNetw32 as the pose estimator, PoseC3D as the skeleton-based action recognizer, SlowOnly-8x8-R101 as the rgb-based spatio temporal action detector. Making action detection predictions per 8 frames, and output 1 frame per 1 frame to the output video. The FPS of the output video is 24.

```
python demo/demo_video_structuralize.py \
    --rgb-stdet-config configs/detection/ava/slowonly_omnisource_pretrained_r101_8x8x1_
↪20e_ava_rgb.py \
```

(continues on next page)

(continued from previous page)

```
    --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 \
    --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. Use the Faster RCNN as the human detector, HRNetw32 as the pose estimator, TSN-R50-1x1x3 as the rgb-based action recognizer, PoseC3D as the skeleton-based spatio temporal action detector. Making action detection predictions per 8 frames, and output 1 frame per 1 frame to the output video. The FPS of the output video is 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
```

3.11 Audio Demo

Demo script to predict the audio-based action recognition using a single audio feature.

The script `extract_audio.py` can be used to extract audios from videos and the script `build_audio_features.py` can be used to extract the audio features.

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

Optional arguments:

- `DEVICE`: Type of device to run the demo. Allowed values are cuda devices like `cuda:0` or `cpu`. If not specified, it will be set to `cuda:0`.

Examples:

Assume that you are located at `$MMACTION2` and have already downloaded the checkpoints to the directory `checkpoints/`, or use checkpoint url from `configs/` to directly load the corresponding checkpoint, which will be automatically saved in `$HOME/.cache/torch/checkpoints`.

1. Recognize an audio file as input by using a tsn model on cuda by default.

```
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

FOUR

BENCHMARK

We compare our results with some popular frameworks and official releases in terms of speed.

4.1 Settings

4.1.1 Hardware

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

4.1.2 Software Environment

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

4.1.3 Metrics

The time we measured is the average training time for an iteration, including data processing and model training. The training speed is measure with s/iter. The lower, the better. Note that we skip the first 50 iter times as they may contain the device warmup time.

4.1.4 Comparison Rules

Here we compare our MMAction2 repo with other video understanding toolboxes in the same data and model settings by the training time per iteration. Here, we use

- commit id 7f3490d(1/5/2020) of MMAction
- commit id 8d53d6f(5/5/2020) of Temporal-Shift-Module
- commit id 8299c98(7/7/2020) of PySlowFast
- commit id f13707f(12/12/2018) of BSN(boundary sensitive network)

- commit id 45d0514(17/10/2019) of BMN(boundary matching network)

To ensure the fairness of the comparison, the comparison experiments were conducted under the same hardware environment and using the same dataset. The rawframe dataset we used is generated by the data preparation tools, the video dataset we used is a special version of resized video cache called '256p dense-encoded video', featuring a faster decoding speed which is generated by the scripts here. Significant improvement can be observed when comparing with normal 256p videos as shown in the table below, especially when the sampling is sparse(like TSN).

For each model setting, we kept the same data preprocessing methods to make sure the same feature input. In addition, we also used Memcached, a distributed cached system, to load the data for the same IO time except for fair comparisons with Pyslowfast which uses raw videos directly from disk by default.

We provide the training log based on which we calculate the average iter time, with the actual setting logged inside, feel free to verify it and fire an issue if something does not make sense.

4.2 Main Results

4.2.1 Recognizers

4.2.2 Localizers

4.3 Details of Comparison

4.3.1 TSN

- **MMAction2**

```
# rawframes
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

# videos
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**

```
# rawframes
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

# videos
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
```

You may reproduce the result by writing a simple script to parse out the value of the field 'time_diff'.

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
```

- **PySlowFast**

```
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
```

You may reproduce the result by writing a simple script to parse out the value of the field 'time_diff'.

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
```

(continues on next page)

(continued from previous page)

You may reproduce the result by writing a simple script to parse out the value of the field 'time_diff'.

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

FIVE

OVERVIEW

- Number of papers: 16
  - DATASET: 16

For supported action algorithms, see *modelzoo overview*.

5.1 Supported Datasets

- Number of papers: 16
  - [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] Revisiting Skeleton-Based Action Recognition (Skeleton Dataset ->, Skeleton Dataset ->, Skeleton Dataset ->)
  - [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] Ucf101: A Dataset of 101 Human Actions Classes From Videos in the Wild (UCF101-24 ->, UCF-101 ->, UCF101-24 ->, UCF-101 ->, UCF101-24 ->, UCF-101 ->)
- [DATASET] {Thumos (THUMOS'14 ->, THUMOS'14 ->, THUMOS'14 ->)

CHAPTER

SIX

DATA PREPARATION

We provide some tips for MMAction2 data preparation in this file.

6.1 Notes on Video Data Format

MMAction2 supports two types of data format: raw frames and video. The former is widely used in previous projects such as TSN. This is fast when SSD is available but fails to scale to the fast-growing datasets. (For example, the newest edition of Kinetics has 650K videos and the total frames will take up several TBs.) The latter saves much space but has to do the computation intensive video decoding at execution time. To make video decoding faster, we support several efficient video loading libraries, such as decord, PyAV, etc.

6.2 Getting Data

The following guide is helpful when you want to experiment with custom dataset. Similar to the datasets stated above, it is recommended organizing in `$MMACTION2/data/$DATASET`.

6.2.1 Prepare videos

Please refer to the official website and/or the official script to prepare the videos. Note that the videos should be arranged in either

(1). A two-level directory organized by `${CLASS_NAME}/${VIDEO_ID}`, which is recommended to be used for action recognition datasets (such as UCF101 and Kinetics)

(2). A single-level directory, which is recommended to be used for action detection datasets or those with multiple annotations per video (such as THUMOS14).

6.2.2 Extract frames

To extract both frames and optical flow, you can use the tool denseflow we wrote. Since different frame extraction tools produce different number of frames, it is beneficial to use the same tool to do both frame extraction and the flow computation, to avoid mismatching of frame counts.

```
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: Folder of the original video.
- OUT_FOLDER: Root folder where the extracted frames and optical flow store.
- TASK: Extraction task indicating which kind of frames to extract. Allowed choices are rgb, flow, both.
- LEVEL: Directory level. 1 for the single-level directory or 2 for the two-level directory.
- NUM_WORKER: Number of workers to build rawframes.
- FLOW_TYPE: Flow type to extract, e.g., None, tvl1, warp_tvl1, farn, brox.
- OUT_FORMAT: Output format for extracted frames, e.g., jpg, h5, png.
- EXT: Video file extension, e.g., avi, mp4.
- NEW_WIDTH: Resized image width of output.
- NEW_HEIGHT: Resized image height of output.
- NEW_SHORT: Resized image short side length keeping ratio.
- --resume: Whether to resume optical flow extraction instead of overwriting.
- --use-opencv: Whether to use OpenCV to extract rgb frames.
- --mixed-ext: Indicate whether process video files with mixed extensions.

The recommended practice is

1. set $OUT_FOLDER to be a folder located in SSD.
2. symlink the link $OUT_FOLDER to $MMACTION2/data/$DATASET/rawframes.
3. set new-short instead of using new-width and new-height.

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

Alternative to denseflow

In case your device doesn't fulfill the installation requirement of denseflow(like Nvidia driver version), or you just want to see some quick demos about flow extraction, we provide a python script `tools/misc/flow_extraction.py` as an alternative to denseflow. You can use it for rgb frames and optical flow extraction from one or several videos. Note that the speed of the script is much slower than denseflow, since it runs optical flow algorithms on CPU.

```
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`: Videos for frame extraction, can be single video or a video list, the video list should be a txt file and just consists of filenames without directories.
- `PREFIX`: The prefix of input videos, used when input is a video list.
- `DEST`: The destination to save extracted frames.
- `RGB_TMPL`: The template filename of rgb frames.
- `FLOW_TMPL`: The template filename of flow frames.
- `START_IDX`: The start index of extracted frames.
- `METHOD`: The method used to generate flow.
- `BOUND`: The maximum of optical flow.
- `SAVE_RGB`: Also save extracted rgb frames.

6.2.3 Generate file list

We provide a convenient script to generate annotation file list. You can use the following command to generate file lists given extracted frames / downloaded videos.

```
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`: Dataset to be prepared, e.g., `ucf101`, `kinetics400`, `thumos14`, `sthv1`, `sthv2`, etc.
- `SRC_FOLDER`: Folder of the corresponding data format:
  - "$MMACTION2/data/$DATASET/rawframes" if `--format rawframes`.
  - "$MMACTION2/data/$DATASET/videos" if `--format videos`.
- `RGB_PREFIX`: Name prefix of rgb frames.
- `FLOW_X_PREFIX`: Name prefix of x flow frames.
- `FLOW_Y_PREFIX`: Name prefix of y flow frames.
- `NUM_SPLIT`: Number of split to file list.

- SUBSET: Subset to generate file list. Allowed choice are train, val, test.
- LEVEL: Directory level. 1 for the single-level directory or 2 for the two-level directory.
- FORMAT: Source data format to generate file list. Allowed choices are rawframes, videos.
- OUT_ROOT_PATH: Root path for output
- SEED: Random seed.
- --shuffle: Whether to shuffle the file list.

Now, you can go to *getting_started.md* to train and test the model.

6.2.4 Prepare audio

We also provide a simple script for audio waveform extraction and mel-spectrogram generation.

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

- ROOT: The root directory of the videos.
- DST_ROOT: The destination root directory of the audios.
- EXT: Extension of the video files. e.g., mp4.
- N_WORKERS: Number of processes to be used.

After extracting audios, you are free to decode and generate the spectrogram on-the-fly such as this. As for the annotations, you can directly use those of the rawframes as long as you keep the relative position of audio files same as the rawframes directory. However, extracting spectrogram on-the-fly is slow and bad for prototype iteration. Therefore, we also provide a script (and many useful tools to play with) for you to generation spectrogram off-line.

```
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: The root directory of the audio files.
- SPECTROGRAM_SAVE_PATH: The destination root directory of the audio features.
- EXT: Extension of the audio files. e.g., m4a.
- N_WORKERS: Number of processes to be used.
- PART: Determines how many parts to be splited and which part to run. e.g., 2/5 means splitting all files into 5-fold and executing the 2nd part. This is useful if you have several machines.

The annotations for audio spectrogram features are identical to those of rawframes. You can simply make a copy of dataset_[train/val]_list_rawframes.txt and rename it as dataset_[train/val]_list_audio_feature.txt

CHAPTER

SEVEN

SUPPORTED DATASETS

- Action Recognition
  - *UCF101* [Homepage].
  - *HMDB51* [Homepage].
  - *Kinetics-[400/600/700]* [Homepage]
  - *Something-Something V1* [Homepage]
  - *Something-Something V2* [Homepage]
  - *Moments in Time* [Homepage]
  - *Multi-Moments in Time* [Homepage]
  - *HVU* [Homepage]
  - *Jester* [Homepage]
  - *GYM* [Homepage]
  - *ActivityNet* [Homepage]
  - *Diving48* [Homepage]
  - *OmniSource* [Homepage]
- Temporal Action Detection
  - *ActivityNet* [Homepage]
  - *THUMOS14* [Homepage]
- Spatial Temporal Action Detection
  - *AVA* [Homepage]
  - *UCF101-24* [Homepage]
  - *JHMDB* [Homepage]
- Skeleton-based Action Recognition
  - *PoseC3D Skeleton Dataset* [Homepage]

The supported datasets are listed above. We provide shell scripts for data preparation under the path `$MMACTION2/tools/data/`. Below is the detailed tutorials of data deployment for each dataset.

7.1 ActivityNet

7.1.1 Introduction

```
@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}
}
```

For basic dataset information, please refer to the official website. For action detection, you can either use the ActivityNet rescaled feature provided in this repo or extract feature with mmaction2 (which has better performance). We release both pipeline. Before we start, please make sure that current working directory is `$MMACTION2/tools/data/activitynet/`.

7.1.2 Option 1: Use the ActivityNet rescaled feature provided in this repo

Step 1. Download Annotations

First of all, you can run the following script to download annotation files.

```
bash download_feature_annotations.sh
```

Step 2. Prepare Videos Features

Then, you can run the following script to download activitynet features.

```
bash download_features.sh
```

Step 3. Process Annotation Files

Next, you can run the following script to process the downloaded annotation files for training and testing. It first merges the two annotation files together and then separates the annoations by `train`, `val` and `test`.

```
python process_annotations.py
```

7.1.3 Option 2: Extract ActivityNet feature using MMAction2 with all videos provided in official website

Step 1. Download Annotations

First of all, you can run the following script to download annotation files.

```
bash download_annotations.sh
```

Step 2. Prepare Videos

Then, you can run the following script to prepare videos. The codes are adapted from the official crawler. Note that this might take a long time.

```
bash download_videos.sh
```

Since some videos in the ActivityNet dataset might be no longer available on YouTube, official website has made the full dataset available on Google and Baidu drives. To accommodate missing data requests, you can fill in this request form provided in official download page to have a 7-day-access to download the videos from the drive folders.

We also provide download steps for annotations from BSN repo

```
bash download_bsn_videos.sh
```

For this case, the downloading scripts update the annotation file after downloading to make sure every video in it exists.

Step 3. Extract RGB and Flow

Before extracting, please refer to *install.md* for installing denseflow.

Use following scripts to extract both RGB and Flow.

```
bash extract_frames.sh
```

The command above can generate images with new short edge 256. If you want to generate images with short edge 320 (320p), or with fix size 340x256, you can change the args `--new-short 256` to `--new-short 320` or `--new-width 340 --new-height 256`. More details can be found in [data_preparation](data_preparation.md)

Step 4. Generate File List for ActivityNet Finetuning

With extracted frames, you can generate video-level or clip-level lists of rawframes, which can be used for ActivityNet Finetuning.

```
python generate_rawframes_filelist.py
```

Step 5. Finetune TSN models on ActivityNet

You can use ActivityNet configs in `configs/recognition/tsn` to finetune TSN models on ActivityNet. You need to use Kinetics models for pretraining. Both RGB models and Flow models are supported.

Step 6. Extract ActivityNet Feature with finetuned ckpts

After finetuning TSN on ActivityNet, you can use it to extract both RGB and Flow feature.

```
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
```

(continues on next page)

(continued from previous page)

```
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
```

After feature extraction, you can use our post processing scripts to concat RGB and Flow feature, generate the `100-t X 400-d` feature for Action Detection.

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

7.1.4 Final Step. Check Directory Structure

After the whole data pipeline for ActivityNet preparation, you will get the features, videos, frames and annotation files.

In the context of the whole project (for ActivityNet only), the folder structure will look like:

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

(if Option 1 used)
│   │   ├── 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
│   │   │   │   ├── ..

(if Option 2 used)
│   │   ├── 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
```

(continues on next page)

(continued from previous page)

```
│   │   │   │   ├── flow_x_00000.jpg
│   │   │   │   ├── flow_y_00000.jpg
│   │   │   │   ├── ..
│   │   │   └── ..
```

For training and evaluating on ActivityNet, please refer to *getting_started.md*.

7.2 AVA

7.2.1 Introduction

```
@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}
}
```

For basic dataset information, please refer to the official website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/ava/`.

7.2.2 Step 1. Prepare Annotations

First of all, you can run the following script to prepare annotations.

```
bash download_annotations.sh
```

This command will download `ava_v2.1.zip` for AVA `v2.1` annotation. If you need the AVA `v2.2` annotation, you can try the following script.

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

7.2.3 Step 2. Prepare Videos

Then, use the following script to prepare videos. The codes are adapted from the official crawler. Note that this might take a long time.

```
bash download_videos.sh
```

Or you can use the following command to downloading AVA videos in parallel using a python script.

```
bash download_videos_parallel.sh
```

Note that if you happen to have sudoer or have GNU parallel on your machine, you can speed up the procedure by downloading in parallel.

```
## sudo apt-get install parallel
bash download_videos_gnu_parallel.sh
```

7.2.4 Step 3. Cut Videos

Cut each video from its 15th to 30th minute and make them at 30 fps.

```
bash cut_videos.sh
```

7.2.5 Step 4. Extract RGB and Flow

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance. And you can run the following script to soft link the extracted frames.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/ava_extracted/
ln -s /mnt/SSD/ava_extracted/ ../data/ava/rawframes/
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

```
bash extract_rgb_frames.sh
```

If you didn't install denseflow, you can still extract RGB frames using ffmpeg by the following script.

```
bash extract_rgb_frames_ffmpeg.sh
```

If both are required, run the following script to extract frames.

```
bash extract_frames.sh
```

7.2.6 Step 5. Fetch Proposal Files

The scripts are adapted from FAIR's Long-Term Feature Banks.

Run the following scripts to fetch the pre-computed proposal list.

```
bash fetch_ava_proposals.sh
```

7.2.7 Step 6. Folder Structure

After the whole data pipeline for AVA preparation. you can get the rawframes (RGB + Flow), videos and annotation files for AVA.

In the context of the whole project (for AVA only), the *minimal* folder structure will look like: (*minimal* means that some data are not necessary: for example, you may want to evaluate AVA using the original video format.)

```
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
│   │   │   │   ├── ...
```

For training and evaluating on AVA, please refer to [getting_started](getting_started.md).

7.2.8 Reference

1. O. Tange (2018): GNU Parallel 2018, March 2018, https://doi.org/10.5281/zenodo.1146014

7.3 Diving48

7.3.1 Introduction

```
@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)},
```

(continues on next page)

(continued from previous page)

```
  pages={513--528},
  year={2018}
}
```

For basic dataset information, you can refer to the official dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/diving48/`.

7.3.2 Step 1. Prepare Annotations

You can run the following script to download annotations (considering the correctness of annotation files, we only download V2 version here).

```
bash download_annotations.sh
```

7.3.3 Step 2. Prepare Videos

You can run the following script to download videos.

```
bash download_videos.sh
```

7.3.4 Step 3. Prepare RGB and Flow

This part is **optional** if you only want to use the video loader.

The frames provided in official compressed file are not complete. You may need to go through the following extraction steps to get the complete frames.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/diving48_extracted/
ln -s /mnt/SSD/diving48_extracted/ ../../../data/diving48/rawframes
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

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

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

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

If both are required, run the following script to extract frames.

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

7.3.5 Step 4. Generate File List

you can run the follow script to generate file list in the format of rawframes and videos.

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

7.3.6 Step 5. Check Directory Structure

After the whole data process for Diving48 preparation, you will get the rawframes (RGB + Flow), videos and annotation files for Diving48.

In the context of the whole project (for Diving48 only), the folder structure will look like:

```
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
    |   |   ├── _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
    |   |   ├── ...
```

For training and evaluating on Diving48, please refer to *getting_started.md*.

7.4 GYM

7.4.1 Introduction

```
@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}
}
```

For basic dataset information, please refer to the official project and the paper. We currently provide the data preprocessing pipeline for GYM99. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/gym/`.

7.4.2 Step 1. Prepare Annotations

First of all, you can run the following script to prepare annotations.

```
bash download_annotations.sh
```

7.4.3 Step 2. Prepare Videos

Then, you can run the following script to prepare videos. The codes are adapted from the official crawler. Note that this might take a long time.

```
bash download_videos.sh
```

7.4.4 Step 3. Trim Videos into Events

First, you need to trim long videos into events based on the annotation of GYM with the following scripts.

```
python trim_event.py
```

7.4.5 Step 4. Trim Events into Subactions

Then, you need to trim events into subactions based on the annotation of GYM with the following scripts. We use the two stage trimming for better efficiency (trimming multiple short clips from a long video can be extremely inefficient, since you need to go over the video many times).

```
python trim_subaction.py
```

7.4.6 Step 5. Extract RGB and Flow

This part is **optional** if you only want to use the video loader for RGB model training.

Before extracting, please refer to *install.md* for installing denseflow.

Run the following script to extract both rgb and flow using "tvl1" algorithm.

```
bash extract_frames.sh
```

7.4.7 Step 6. Generate file list for GYM99 based on extracted subactions

You can use the following script to generate train / val lists for GYM99.

```
python generate_file_list.py
```

7.4.8 Step 7. Folder Structure

After the whole data pipeline for GYM preparation. You can get the subaction clips, event clips, raw videos and GYM99 train/val lists.

In the context of the whole project (for GYM only), the full folder structure will look like:

```
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
```

For training and evaluating on GYM, please refer to [getting_started](getting_started.md).

7.5 HMDB51

7.5.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/hmdb51/`.

To run the bash scripts below, you need to install `unrar`. you can install it by `sudo apt-get install unrar`, or refer to this repo by following the usage and taking `zzunrar.sh` script for easy installation without sudo.

7.5.2 Step 1. Prepare Annotations

First of all, you can run the following script to prepare annotations.

```
bash download_annotations.sh
```

7.5.3 Step 2. Prepare Videos

Then, you can run the following script to prepare videos.

```
bash download_videos.sh
```

7.5.4 Step 3. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/hmdb51_extracted/
ln -s /mnt/SSD/hmdb51_extracted/ ../../../data/hmdb51/rawframes
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

```
bash extract_rgb_frames.sh
```

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

```
bash extract_rgb_frames_opencv.sh
```

If both are required, run the following script to extract frames using "tvl1" algorithm.

```
bash extract_frames.sh
```

7.5.5 Step 4. Generate File List

you can run the follow script to generate file list in the format of rawframes and videos.

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

7.5.6 Step 5. Check Directory Structure

After the whole data process for HMDB51 preparation, you will get the rawframes (RGB + Flow), videos and annotation files for HMDB51.

In the context of the whole project (for HMDB51 only), the folder structure will look like:

```
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
                ├── 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
                ├── ...
```

(continues on next page)

(continued from previous page)

```
│   │   │   │   ├── winKen_wave_u_cm_np1_ri_bad_1
```

For training and evaluating on HMDB51, please refer to getting_started.md.

7.6 HVU

7.6.1 Introduction

```
@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}
}
```

For basic dataset information, please refer to the official project and the paper. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/hvu/`.

7.6.2 Step 1. Prepare Annotations

First of all, you can run the following script to prepare annotations.

```
bash download_annotations.sh
```

Besides, you need to run the following command to parse the tag list of HVU.

```
python parse_tag_list.py
```

7.6.3 Step 2. Prepare Videos

Then, you can run the following script to prepare videos. The codes are adapted from the official crawler. Note that this might take a long time.

```
bash download_videos.sh
```

7.6.4 Step 3. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

You can use the following script to extract both RGB and Flow frames.

```
bash extract_frames.sh
```

By default, we generate frames with short edge resized to 256. More details can be found in [data_preparation](data_preparation.md)

7.6.5 Step 4. Generate File List

You can run the follow scripts to generate file list in the format of videos and rawframes, respectively.

```
bash generate_videos_filelist.sh
## execute the command below when rawframes are ready
bash generate_rawframes_filelist.sh
```

7.6.6 Step 5. Generate File List for Each Individual Tag Categories

This part is **optional** if you don't want to train models on HVU for a specific tag category.

The file list generated in step 4 contains labels of different categories. These file lists can only be handled with HVUDataset and used for multi-task learning of different tag categories. The component `LoadHVULabel` is needed to load the multi-category tags, and the `HVULoss` should be used to train the model.

If you only want to train video recognition models for a specific tag category, i.e. you want to train a recognition model on HVU which only handles tags in the category `action`, we recommend you to use the following command to generate file lists for the specific tag category. The new list, which only contains tags of a specific category, can be handled with `VideoDataset` or `RawframeDataset`. The recognition models can be trained with `BCELossWithLogits`.

The following command generates file list for the tag category ${category}, note that the tag category you specified should be in the 6 tag categories available in HVU: ['action', 'attribute', 'concept', 'event', 'object', 'scene'].

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

The filename of the generated file list for ${category} is generated by replacing `hvu` in the original filename with `hvu_${category}`. For example, if the original filename is `hvu_train.json`, the filename of the file list for action is `hvu_action_train.json`.

7.6.7 Step 6. Folder Structure

After the whole data pipeline for HVU preparation. you can get the rawframes (RGB + Flow), videos and annotation files for HVU.

In the context of the whole project (for HVU only), the full folder structure will look like:

```
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
```

(continues on next page)

(continued from previous page)

```
│   │   ├── rawframes_train
│   │   ├── rawframes_val
```

For training and evaluating on HVU, please refer to [getting_started](getting_started.md).

7.7 Jester

7.7.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/jester/`.

7.7.2 Step 1. Prepare Annotations

First of all, you have to sign in and download annotations to `$MMACTION2/data/jester/annotations` on the official website.

7.7.3 Step 2. Prepare RGB Frames

Since the jester website doesn't provide the original video data and only extracted RGB frames are available, you have to directly download RGB frames from jester website.

You can download all RGB frame parts on jester website to `$MMACTION2/data/jester/` and use the following command to extract.

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

For users who only want to use RGB frames, you can skip to step 5 to generate file lists in the format of rawframes. Since the prefix of official JPGs is "%05d.jpg" (e.g., "00001.jpg"), we add `"filename_tmpl='{:05}.jpg'"` to the dict of `data.train`, `data.val` and `data.test` in the config files related with jester like this:

```
data = dict(
    videos_per_gpu=16,
    workers_per_gpu=2,
    train=dict(
        type=dataset_type,
```

(continues on next page)

(continued from previous page)

```
        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 Step 3. Extract Flow

This part is **optional** if you only want to use RGB frames.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/jester_extracted/
ln -s /mnt/SSD/jester_extracted/ ../../../data/jester/rawframes
```

Then, you can run the following script to extract optical flow based on RGB frames.

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

7.7.5 Step 4. Encode Videos

This part is **optional** if you only want to use RGB frames.

You can run the following script to encode videos.

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

7.7.6 Step 5. Generate File List

You can run the follow script to generate file list in the format of rawframes and videos.

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

7.7.7 Step 5. Check Directory Structure

After the whole data process for Jester preparation, you will get the rawframes (RGB + Flow), and annotation files for Jester.

In the context of the whole project (for Jester only), the folder structure will look like:

```
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
│   │   │   │   ├── flow_y_00002.jpg
│   │   │   │   ├── ...
│   │   │   ├── 2
│   │   │   ├── ...
```

For training and evaluating on Jester, please refer to *getting_started.md*.

7.8 JHMDB

7.8.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/jhmdb/`.

7.8.2 Download and Extract

You can download the RGB frames, optical flow and ground truth annotations from google drive. The data are provided from MOC, which is adapted from act-detector.

After downloading the `JHMDB.tar.gz` file and put it in `$MMACTION2/tools/data/jhmdb/`, you can run the following command to extract.

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

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/JHMDB/
ln -s /mnt/SSD/JHMDB/ ../../../data/jhmdb
```

7.8.3 Check Directory Structure

After extracting, you will get the `FlowBrox04` directory, `Frames` directory and `JHMDB-GT.pkl` for JHMDB.

In the context of the whole project (for JHMDB only), the folder structure will look like:

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
│   ├── jhmdb
│   │   ├── FlowBrox04
│   │   │   ├── brush_hair
│   │   │   │   ├── April_09_brush_hair_u_nm_np1_ba_goo_0
│   │   │   │   │   ├── 00001.jpg
│   │   │   │   │   ├── 00002.jpg
│   │   │   │   │   ├── ...
```

(continues on next page)

(continued from previous page)

```
│   │   │   │   ├── 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
```

Note: The `JHMDB-GT.pkl` exists as a cache, it contains 6 items as follows:

1. `labels` (list): List of the 21 labels.
2. `gttubes` (dict): Dictionary that contains the ground truth tubes for each video. A **gttube** is dictionary that associates with each index of label and a list of tubes. A **tube** is a numpy array with `nframes` rows and 5 columns, each col is in format like `<frame index> <x1> <y1> <x2> <y2>`.
3. `nframes` (dict): Dictionary that contains the number of frames for each video, like `'walk/Panic_in_the_Streets_walk_u_cm_np1_ba_med_5': 16`.
4. `train_videos` (list): A list with `nsplits=1` elements, each one containing the list of training videos.
5. `test_videos` (list): A list with `nsplits=1` elements, each one containing the list of testing videos.
6. `resolution` (dict): Dictionary that outputs a tuple (h,w) of the resolution for each video, like `'pour/Bartender_School_Students_Practice_pour_u_cm_np1_fr_med_1': (240, 320)`.

7.9 Kinetics-[400/600/700]

7.9.1 Introduction

```
@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}
}
```

For basic dataset information, please refer to the official website. The scripts can be used for preparing kinetics400, kinetics600, kinetics700. To prepare different version of kinetics, you need to replace `${DATASET}` in the following examples with the specific dataset name. The choices of dataset names are `kinetics400`, `kinetics600` and `kinetics700`. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/${DATASET}/`.

Note: Because of the expirations of some YouTube links, the sizes of kinetics dataset copies may be different. Here are the sizes of our kinetics dataset copies that used to train all checkpoints.

7.9.2 Step 1. Prepare Annotations

First of all, you can run the following script to prepare annotations by downloading from the official website.

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

Since some video urls are invalid, the number of video items in current official annotations are less than the original official ones. So we provide an alternative way to download the older one as a reference. Among these, the annotation files of Kinetics400 and Kinetics600 are from official crawler, the annotation files of Kinetics700 are from website downloaded in 05/02/2021.

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

7.9.3 Step 2. Prepare Videos

Then, you can run the following script to prepare videos. The codes are adapted from the official crawler. Note that this might take a long time.

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

Important: If you have already downloaded video dataset using the download script above, you must replace all whitespaces in the class name for ease of processing by running

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

For better decoding speed, you can resize the original videos into smaller sized, densely encoded version by:

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

You can also download from Academic Torrents (kinetics400 & kinetics700 with short edge 256 pixels are available) and cvdfoundation/kinetics-dataset (Host by Common Visual Data Foundation and Kinetics400/Kinetics600/Kinetics-700-2020 are available)

7.9.4 Step 3. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance. And you can run the following script to soft link the extracted frames.

```
## execute these two line (Assume the SSD is mounted at "/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/
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

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

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

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

If both are required, run the following script to extract frames.

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

The commands above can generate images with new short edge 256. If you want to generate images with short edge 320 (320p), or with fix size 340x256, you can change the args `--new-short 256` to `--new-short 320` or `--new-width 340 --new-height 256`. More details can be found in *data_preparation*

7.9.5 Step 4. Generate File List

you can run the follow scripts to generate file list in the format of videos and rawframes, respectively.

```
bash generate_videos_filelist.sh ${DATASET}
## execute the command below when rawframes are ready
bash generate_rawframes_filelist.sh ${DATASET}
```

7.9.6 Step 5. Folder Structure

After the whole data pipeline for Kinetics preparation. you can get the rawframes (RGB + Flow), videos and annotation files for Kinetics.

In the context of the whole project (for Kinetics only), the *minimal* folder structure will look like: (*minimal* means that some data are not necessary: for example, you may want to evaluate kinetics using the original video format.)

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

For training and evaluating on Kinetics, please refer to *getting_started*.

7.10 Moments in Time

7.10.1 Introduction

```
@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},
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/mit/`.

7.10.2 Step 1. Prepare Annotations and Videos

First of all, you have to visit the official website, fill in an application form for downloading the dataset. Then you will get the download link. You can use `bash preprocess_data.sh` to prepare annotations and videos. However, the download command is missing in that script. Remember to download the dataset to the proper place follow the comment in this script.

For better decoding speed, you can resize the original videos into smaller sized, densely encoded version by:

```
python ../resize_videos.py ../../../data/mit/videos/ ../../../data/mit/videos_256p_dense_
↪cache --dense --level 2
```

7.10.3 Step 2. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance. And you can run the following script to soft link the extracted frames.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/mit_extracted/
ln -s /mnt/SSD/mit_extracted/ ../../../data/mit/rawframes
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

```
bash extract_rgb_frames.sh
```

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

```
bash extract_rgb_frames_opencv.sh
```

If both are required, run the following script to extract frames.

```
bash extract_frames.sh
```

7.10.4 Step 4. Generate File List

you can run the follow script to generate file list in the format of rawframes and videos.

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

7.10.5 Step 5. Check Directory Structure

After the whole data process for Moments in Time preparation, you will get the rawframes (RGB + Flow), videos and annotation files for Moments in Time.

In the context of the whole project (for Moments in Time only), the folder structure will look like:

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

For training and evaluating on Moments in Time, please refer to *getting_started.md*.

7.11 Multi-Moments in Time

7.11.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/mmit/`.

7.11.2 Step 1. Prepare Annotations and Videos

First of all, you have to visit the official website, fill in an application form for downloading the dataset. Then you will get the download link. You can use `bash preprocess_data.sh` to prepare annotations and videos. However, the download command is missing in that script. Remember to download the dataset to the proper place follow the comment in this script.

For better decoding speed, you can resize the original videos into smaller sized, densely encoded version by:

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

7.11.3 Step 2. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

First, you can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/mmit_extracted/
ln -s /mnt/SSD/mmit_extracted/ ../../../data/mmit/rawframes
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

```
bash extract_rgb_frames.sh
```

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

```
bash extract_rgb_frames_opencv.sh
```

If both are required, run the following script to extract frames using "tvl1" algorithm.

```
bash extract_frames.sh
```

7.11.4 Step 3. Generate File List

you can run the follow script to generate file list in the format of rawframes or videos.

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

7.11.5 Step 4. Check Directory Structure

After the whole data process for Multi-Moments in Time preparation, you will get the rawframes (RGB + Flow), videos and annotation files for Multi-Moments in Time.

In the context of the whole project (for Multi-Moments in Time only), the folder structure will look like:

```
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
        │   │   ├── ...
        │   │   ├── 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
            └── ...
```

For training and evaluating on Multi-Moments in Time, please refer to *getting_started.md*.

7.12 OmniSource

7.12.1 Introduction

```
@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}
}
```

We release a subset of the OmniSource web dataset used in the paper Omni-sourced Webly-supervised Learning for Video Recognition. Since all web dataset in OmniSource are built based on the Kinetics-400 taxonomy, we select those web data related to the 200 classes in Mini-Kinetics subset (which is proposed in Rethinking Spatiotemporal Feature Learning: Speed-Accuracy Trade-offs in Video Classification).

We provide data from all sources that are related to the 200 classes in Mini-Kinetics (including Kinetics trimmed clips, Kinetics untrimmed videos, images from Google and Instagram, video clips from Instagram). To obtain this dataset, please first fill in the request form. We will share the download link to you after your request is received. Since we release all data crawled from the web without any filtering, the dataset is large and it may take some time to download them. We describe the size of the datasets in the following table:

The file structure of our uploaded OmniSource dataset looks like:

```
OmniSource/
├── annotations
│   ├── googleimage_200
│   │   ├── googleimage_200.txt                        File list of all valid images␣
↪crawled from Google.
│   │   ├── tsn_8seg_googleimage_200_duplicate.txt     Positive file list of images␣
↪crawled from Google, which is similar to a validation example.
│   │   ├── tsn_8seg_googleimage_200.txt               Positive file list of images␣
↪crawled from Google, filtered by the teacher model.
│   │   └── tsn_8seg_googleimage_200_wodup.txt         Positive file list of images␣
↪crawled from Google, filtered by the teacher model, after de-duplication.
│   ├── 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                               The list of action names of the␣
↪200 classes in MiniKinetics.
│   ├── K400_to_MiniKinetics_classidx_mapping.json     The index mapping from Kinetics-
↪400 to MiniKinetics.
│   ├── kinetics_200
│   │   ├── k200_train.txt
│   │   └── k200_val.txt
│   ├── kinetics_raw_200
```

(continues on next page)

(continued from previous page)

```
│   │   └── 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                                   (10 volumes)
│   ├── vol_0.tar
│   ├── ...
│   └── vol_9.tar
├── insimage_200                                      (10 volumes)
│   ├── vol_0.tar
│   ├── ...
│   └── vol_9.tar
├── insvideo_200                                      (20 volumes)
│   ├── vol_00.tar
│   ├── ...
│   └── vol_19.tar
├── kinetics_200_train
│   └── kinetics_200_train.tar
├── kinetics_200_val
│   └── kinetics_200_val.tar
└── kinetics_raw_200_train                            (16 volumes)
    ├── vol_0.tar
    ├── ...
    └── vol_15.tar
```

7.12.2 Data Preparation

For data preparation, you need to first download those data. For `kinetics_200` and 3 web datasets: `googleimage_200`, `insimage_200` and `insvideo_200`, you just need to extract each volume and merge their contents.

For Kinetics raw videos, since loading long videos is very heavy, you need to first trim it into clips. Here we provide a script named `trim_raw_video.py`. It trims a long video into 10-second clips and remove the original raw video. You can use it to trim the Kinetics raw video.

The data should be placed in `data/OmniSource/`. When data preparation finished, the folder structure of `data/OmniSource` looks like (We omit the files not needed in training & testing for simplicity):

```
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
```

(continues on next page)

(continued from previous page)

```
│   │   └── 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
│   │   ├── ...
│   │   ├── 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 Skeleton Dataset

```
@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 Introduction

We release the skeleton annotations used in Revisiting Skeleton-based Action Recognition. By default, we use Faster-RCNN with ResNet50 backbone for human detection and HRNet-w32 for single person pose estimation. For FineGYM, we use Ground-Truth bounding boxes for the athlete instead of detection bounding boxes. Currently, we release the skeleton annotations for FineGYM and NTURGB-D Xsub split. Other annotations will be soo released.

7.13.2 Prepare Annotations

Currently, we support HMDB51, UCF101, FineGYM and NTURGB+D. For FineGYM, you can execute following scripts to prepare the annotations.

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

Due to Conditions of Use of the NTURGB+D dataset, we can not directly release the annotations used in our experiments. So that we provide a script to generate pose annotations for videos in NTURGB+D datasets, which generate a dictionary and save it as a single pickle file. You can create a list which contain all annotation dictionaries of corresponding videos and save them as a pickle file. Then you can get the `ntu60_xsub_train.pkl`, `ntu60_xsub_val.pkl`, `ntu120_xsub_train.pkl`, `ntu120_xsub_val.pkl` that we used in training.

For those who have not enough computations for pose extraction, we provide the outputs of the above pipeline here, corresponding to 4 different splits of NTURGB+D datasets:

- 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

To generate 2D pose annotations for a single video, first, you need to install mmdetection and mmpose from src code. After that, you need to replace the placeholder `mmdet_root` and `mmpose_root` in `ntu_pose_extraction.py` with your installation path. Then you can use following scripts for NTURGB+D video pose extraction:

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

After you get pose annotations for all videos in a dataset split, like `ntu60_xsub_val`. You can gather them into a single list and save the list as `ntu60_xsub_val.pkl`. You can use those larger pickle files for training and testing.

7.13.3 The Format of PoseC3D Annotations

Here we briefly introduce the format of PoseC3D Annotations, we will take `gym_train.pkl` as an example: the content of `gym_train.pkl` is a list of length 20484, each item is a dictionary that is the skeleton annotation of one video. Each dictionary has following fields:

- keypoint: The keypoint coordinates, which is a numpy array of the shape N (##person) x T (temporal length) x K (#keypoints, 17 in our case) x 2 (x, y coordinate).
- keypoint_score: The keypoint confidence scores, which is a numpy array of the shape N (##person) x T (temporal length) x K (#keypoints, 17 in our case).
- frame_dir: The corresponding video name.
- label: The action category.
- img_shape: The image shape of each frame.
- original_shape: Same as above.
- total_frames: The temporal length of the video.

For training with your custom dataset, you can refer to Custom Dataset Training.

7.13.4 Visualization

For skeleton data visualization, you need also to prepare the RGB videos. Please refer to visualize_heatmap_volume for detailed process. Here we provide some visualization examples from NTU-60 and FineGYM.

7.13.5 Convert the NTU RGB+D raw skeleton data to our format (only applicable to GCN backbones)

Here we also provide the script for converting the NTU RGB+D raw skeleton data to our format. First, download the raw skeleton data of NTU-RGBD 60 and NTU-RGBD 120 from https://github.com/shahroudy/NTURGB-D.

For NTU-RGBD 60, preprocess data and convert the data format with

```
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
```

For NTU-RGBD 120, preprocess data and convert the data format with

```
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 Convert annotations from third-party projects

We provide scripts to convert skeleton annotations from third-party projects to MMAction2 formats:

- BABEL: `babel2mma2.py`

TODO:

- [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 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset paper. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/sthv1/`.

7.14.2 Step 1. Prepare Annotations

Since the official website of Something-Something V1 is currently unavailable, you can download the annotations from third-part source to `$MMACTION2/data/sthv1/` .

7.14.3 Step 2. Prepare RGB Frames

Since the official dataset doesn't provide the original video data and only extracted RGB frames are available, you have to directly download RGB frames.

You can download all compressed file parts from third-part source to `$MMACTION2/data/sthv1/` and use the following command to uncompress.

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

For users who only want to use RGB frames, you can skip to step 5 to generate file lists in the format of rawframes. Since the prefix of official JPGs is "%05d.jpg" (e.g., "00001.jpg"), users need to add `"filename_tmpl='{:05}.jpg'"` to the dict of `data.train`, `data.val` and `data.test` in the config files related with sthv1 like this:

```
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',
        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 Step 3. Extract Flow

This part is **optional** if you only want to use RGB frames.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/sthv1_extracted/
ln -s /mnt/SSD/sthv1_extracted/ ../../../data/sthv1/rawframes
```

Then, you can run the following script to extract optical flow based on RGB frames.

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

7.14.5 Step 4. Encode Videos

This part is **optional** if you only want to use RGB frames.

You can run the following script to encode videos.

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

7.14.6 Step 5. Generate File List

You can run the follow script to generate file list in the format of rawframes and videos.

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

7.14.7 Step 6. Check Directory Structure

After the whole data process for Something-Something V1 preparation, you will get the rawframes (RGB + Flow), and annotation files for Something-Something V1.

In the context of the whole project (for Something-Something V1 only), the folder structure will look like:

```
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
│   │   │   │   ├── ...
```

(continues on next page)

(continued from previous page)

```
│   |   |   ├── 2
│   |   |   ├── ...
```

For training and evaluating on Something-Something V1, please refer to *getting_started.md*.

7.15 Something-Something V2

7.15.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/sthv2/`.

7.15.2 Step 1. Prepare Annotations

First of all, you have to sign in and download annotations to `$MMACTION2/data/sthv2/annotations` on the official website.

```
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 Step 2. Prepare Videos

Then, you can download all data parts to `$MMACTION2/data/sthv2/` and use the following command to uncompress.

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

7.15.4 Step 3. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/sthv2_extracted/
ln -s /mnt/SSD/sthv2_extracted/ ../../../data/sthv2/rawframes
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

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

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

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

If both are required, run the following script to extract frames.

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

7.15.5 Step 4. Generate File List

you can run the follow script to generate file list in the format of rawframes and videos.

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

7.15.6 Step 5. Check Directory Structure

After the whole data process for Something-Something V2 preparation, you will get the rawframes (RGB + Flow), videos and annotation files for Something-Something V2.

In the context of the whole project (for Something-Something V2 only), the folder structure will look like:

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
│   ├── sthv2
│   │   ├── sthv2_{train,val}_list_rawframes.txt
│   │   ├── sthv2_{train,val}_list_videos.txt
│   │   ├── annotations
```

(continues on next page)

(continued from previous page)

```
│   │   ├── 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
│   │   │   ├── ...
```

For training and evaluating on Something-Something V2, please refer to *getting_started.md*.

7.16 THUMOS'14

7.16.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/thumos14/`.

7.16.2 Step 1. Prepare Annotations

First of all, run the following script to prepare annotations.

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

7.16.3 Step 2. Prepare Videos

Then, you can run the following script to prepare videos.

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

7.16.4 Step 3. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/thumos14_extracted/
ln -s /mnt/SSD/thumos14_extracted/ ../data/thumos14/rawframes/
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

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

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

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

If both are required, run the following script to extract frames.

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

7.16.5 Step 4. Fetch File List

This part is **optional** if you do not use SSN model.

You can run the follow script to fetch pre-computed tag proposals.

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

7.16.6 Step 5. Denormalize Proposal File

This part is **optional** if you do not use SSN model.

You can run the follow script to denormalize pre-computed tag proposals according to actual number of local rawframes.

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

7.16.7 Step 6. Check Directory Structure

After the whole data process for THUMOS'14 preparation, you will get the rawframes (RGB + Flow), videos and annotation files for THUMOS'14.

In the context of the whole project (for THUMOS'14 only), the folder structure will look like:

```
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
```

For training and evaluating on THUMOS'14, please refer to *getting_started.md*.

7.17 UCF-101

7.17.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/ucf101/`.

7.17.2 Step 1. Prepare Annotations

First of all, you can run the following script to prepare annotations.

```
bash download_annotations.sh
```

7.17.3 Step 2. Prepare Videos

Then, you can run the following script to prepare videos.

```
bash download_videos.sh
```

For better decoding speed, you can resize the original videos into smaller sized, densely encoded version by:

```
python ../resize_videos.py ../../../data/ucf101/videos/ ../../../data/ucf101/videos_256p_
↪dense_cache --dense --level 2 --ext avi
```

7.17.4 Step 3. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance. The extracted frames (RGB + Flow) will take up about 100GB.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/ucf101_extracted/
ln -s /mnt/SSD/ucf101_extracted/ ../../../data/ucf101/rawframes
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

```
bash extract_rgb_frames.sh
```

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

```
bash extract_rgb_frames_opencv.sh
```

If Optical Flow is also required, run the following script to extract flow using "tvl1" algorithm.

```
bash extract_frames.sh
```

7.17.5 Step 4. Generate File List

you can run the follow script to generate file list in the format of rawframes and videos.

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

7.17.6 Step 5. Check Directory Structure

After the whole data process for UCF-101 preparation, you will get the rawframes (RGB + Flow), videos and annotation files for UCF-101.

In the context of the whole project (for UCF-101 only), the folder structure will look like:

```
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

            ├── 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
```

(continues on next page)

(continued from previous page)

```
│   │   │   ├── ...
│   │   │   ├── YoYo
│   │   │   │   ├── v_YoYo_g01_c01
│   │   │   │   ├── ...
│   │   │   │   ├── v_YoYo_g25_c05
```

For training and evaluating on UCF-101, please refer to *getting_started.md*.

7.18 UCF101-24

7.18.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/ucf101_24/`.

7.18.2 Download and Extract

You can download the RGB frames, optical flow and ground truth annotations from google drive. The data are provided from MOC, which is adapted from act-detector and corrected-UCF101-Annots.

Note: The annotation of this UCF101-24 is from here, which is more correct.

After downloading the `UCF101_v2.tar.gz` file and put it in `$MMACTION2/tools/data/ucf101_24/`, you can run the following command to uncompress.

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

7.18.3 Check Directory Structure

After uncompressing, you will get the `rgb-images` directory, `brox-images` directory and `UCF101v2-GT.pkl` for UCF101-24.

In the context of the whole project (for UCF101-24 only), the folder structure will look like:

```
mmaction2
├── mmaction
├── tools
├── configs
```

(continues on next page)

(continued from previous page)

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

Note: The `UCF101v2-GT.pkl` exists as a cache, it contains 6 items as follows:

1. `labels` (list): List of the 24 labels.
2. `gttubes` (dict): Dictionary that contains the ground truth tubes for each video. A **gttube** is dictionary that associates with each index of label and a list of tubes. A **tube** is a numpy array with `nframes` rows and 5 columns, each col is in format like `<frame index> <x1> <y1> <x2> <y2>`.
3. `nframes` (dict): Dictionary that contains the number of frames for each video, like `'HorseRiding/v_HorseRiding_g05_c02': 151`.
4. `train_videos` (list): A list with `nsplits=1` elements, each one containing the list of training videos.
5. `test_videos` (list): A list with `nsplits=1` elements, each one containing the list of testing videos.
6. `resolution` (dict): Dictionary that outputs a tuple (h,w) of the resolution for each video, like `'FloorGymnastics/v_FloorGymnastics_g09_c03': (240, 320)`.

7.19 ActivityNet

7.19.1 Introduction

```
@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}
}
```

For basic dataset information, please refer to the official website. For action detection, you can either use the ActivityNet rescaled feature provided in this repo or extract feature with mmaction2 (which has better performance). We release both pipeline. Before we start, please make sure that current working directory is `$MMACTION2/tools/data/activitynet/`.

7.19.2 Option 1: Use the ActivityNet rescaled feature provided in this repo

Step 1. Download Annotations

First of all, you can run the following script to download annotation files.

```
bash download_feature_annotations.sh
```

Step 2. Prepare Videos Features

Then, you can run the following script to download activitynet features.

```
bash download_features.sh
```

Step 3. Process Annotation Files

Next, you can run the following script to process the downloaded annotation files for training and testing. It first merges the two annotation files together and then separates the annoations by `train`, `val` and `test`.

```
python process_annotations.py
```

7.19.3 Option 2: Extract ActivityNet feature using MMAction2 with all videos provided in official website

Step 1. Download Annotations

First of all, you can run the following script to download annotation files.

```
bash download_annotations.sh
```

Step 2. Prepare Videos

Then, you can run the following script to prepare videos. The codes are adapted from the official crawler. Note that this might take a long time.

```
bash download_videos.sh
```

Since some videos in the ActivityNet dataset might be no longer available on YouTube, official website has made the full dataset available on Google and Baidu drives. To accommodate missing data requests, you can fill in this request form provided in official download page to have a 7-day-access to download the videos from the drive folders.

We also provide download steps for annotations from BSN repo

```
bash download_bsn_videos.sh
```

For this case, the downloading scripts update the annotation file after downloading to make sure every video in it exists.

Step 3. Extract RGB and Flow

Before extracting, please refer to *install.md* for installing denseflow.

Use following scripts to extract both RGB and Flow.

```
bash extract_frames.sh
```

The command above can generate images with new short edge 256. If you want to generate images with short edge 320 (320p), or with fix size 340x256, you can change the args `--new-short 256` to `--new-short 320` or `--new-width 340 --new-height 256`. More details can be found in [data_preparation](data_preparation.md)

Step 4. Generate File List for ActivityNet Finetuning

With extracted frames, you can generate video-level or clip-level lists of rawframes, which can be used for ActivityNet Finetuning.

```
python generate_rawframes_filelist.py
```

Step 5. Finetune TSN models on ActivityNet

You can use ActivityNet configs in `configs/recognition/tsn` to finetune TSN models on ActivityNet. You need to use Kinetics models for pretraining. Both RGB models and Flow models are supported.

Step 6. Extract ActivityNet Feature with finetuned ckpts

After finetuning TSN on ActivityNet, you can use it to extract both RGB and Flow feature.

```
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
```

(continues on next page)

(continued from previous page)

```
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
```

After feature extraction, you can use our post processing scripts to concat RGB and Flow feature, generate the `100-t X 400-d` feature for Action Detection.

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

7.19.4 Final Step. Check Directory Structure

After the whole data pipeline for ActivityNet preparation, you will get the features, videos, frames and annotation files.

In the context of the whole project (for ActivityNet only), the folder structure will look like:

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

(if Option 1 used)
│   │   ├── 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
│   │   │   │   ├── ..

(if Option 2 used)
│   │   ├── 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
```

(continues on next page)

(continued from previous page)

```
│   │   │   │   ├── flow_x_00000.jpg
│   │   │   │   ├── flow_y_00000.jpg
│   │   │   │   ├── ..
│   │   │   ├── ..
```

For training and evaluating on ActivityNet, please refer to *getting_started.md*.

7.20 AVA

7.20.1 Introduction

```
@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}
}
```

For basic dataset information, please refer to the official website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/ava/`.

7.20.2 Step 1. Prepare Annotations

First of all, you can run the following script to prepare annotations.

```
bash download_annotations.sh
```

This command will download `ava_v2.1.zip` for AVA `v2.1` annotation. If you need the AVA `v2.2` annotation, you can try the following script.

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

7.20.3 Step 2. Prepare Videos

Then, use the following script to prepare videos. The codes are adapted from the official crawler. Note that this might take a long time.

```
bash download_videos.sh
```

Or you can use the following command to downloading AVA videos in parallel using a python script.

```
bash download_videos_parallel.sh
```

Note that if you happen to have sudoer or have GNU parallel on your machine, you can speed up the procedure by downloading in parallel.

```
## sudo apt-get install parallel
bash download_videos_gnu_parallel.sh
```

7.20.4 Step 3. Cut Videos

Cut each video from its 15th to 30th minute and make them at 30 fps.

```
bash cut_videos.sh
```

7.20.5 Step 4. Extract RGB and Flow

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance. And you can run the following script to soft link the extracted frames.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/ava_extracted/
ln -s /mnt/SSD/ava_extracted/ ../data/ava/rawframes/
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

```
bash extract_rgb_frames.sh
```

If you didn't install denseflow, you can still extract RGB frames using ffmpeg by the following script.

```
bash extract_rgb_frames_ffmpeg.sh
```

If both are required, run the following script to extract frames.

```
bash extract_frames.sh
```

7.20.6 Step 5. Fetch Proposal Files

The scripts are adapted from FAIR's Long-Term Feature Banks.

Run the following scripts to fetch the pre-computed proposal list.

```
bash fetch_ava_proposals.sh
```

7.20.7 Step 6. Folder Structure

After the whole data pipeline for AVA preparation. you can get the rawframes (RGB + Flow), videos and annotation files for AVA.

In the context of the whole project (for AVA only), the *minimal* folder structure will look like: (*minimal* means that some data are not necessary: for example, you may want to evaluate AVA using the original video format.)

```
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
|   |   |   |   ├── ...
```

For training and evaluating on AVA, please refer to [getting_started](getting_started.md).

7.20.8 Reference

1. O. Tange (2018): GNU Parallel 2018, March 2018, https://doi.org/10.5281/zenodo.1146014

7.21 Diving48

7.21.1 Introduction

```
@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)},
```

(continues on next page)

(continued from previous page)

```
  pages={513--528},
  year={2018}
}
```

For basic dataset information, you can refer to the official dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/diving48/`.

7.21.2 Step 1. Prepare Annotations

You can run the following script to download annotations (considering the correctness of annotation files, we only download V2 version here).

```
bash download_annotations.sh
```

7.21.3 Step 2. Prepare Videos

You can run the following script to download videos.

```
bash download_videos.sh
```

7.21.4 Step 3. Prepare RGB and Flow

This part is **optional** if you only want to use the video loader.

The frames provided in official compressed file are not complete. You may need to go through the following extraction steps to get the complete frames.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/diving48_extracted/
ln -s /mnt/SSD/diving48_extracted/ ../../../data/diving48/rawframes
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

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

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

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

If both are required, run the following script to extract frames.

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

7.21.5 Step 4. Generate File List

you can run the follow script to generate file list in the format of rawframes and videos.

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

7.21.6 Step 5. Check Directory Structure

After the whole data process for Diving48 preparation, you will get the rawframes (RGB + Flow), videos and annotation files for Diving48.

In the context of the whole project (for Diving48 only), the folder structure will look like:

```
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
    |   |   ├── _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
    |   |   ├── ...
```

For training and evaluating on Diving48, please refer to *getting_started.md*.

7.22 GYM

7.22.1 Introduction

```
@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}
}
```

For basic dataset information, please refer to the official project and the paper. We currently provide the data preprocessing pipeline for GYM99. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/gym/`.

7.22.2 Step 1. Prepare Annotations

First of all, you can run the following script to prepare annotations.

```
bash download_annotations.sh
```

7.22.3 Step 2. Prepare Videos

Then, you can run the following script to prepare videos. The codes are adapted from the official crawler. Note that this might take a long time.

```
bash download_videos.sh
```

7.22.4 Step 3. Trim Videos into Events

First, you need to trim long videos into events based on the annotation of GYM with the following scripts.

```
python trim_event.py
```

7.22.5 Step 4. Trim Events into Subactions

Then, you need to trim events into subactions based on the annotation of GYM with the following scripts. We use the two stage trimming for better efficiency (trimming multiple short clips from a long video can be extremely inefficient, since you need to go over the video many times).

```
python trim_subaction.py
```

7.22.6 Step 5. Extract RGB and Flow

This part is **optional** if you only want to use the video loader for RGB model training.

Before extracting, please refer to *install.md* for installing denseflow.

Run the following script to extract both rgb and flow using "tvl1" algorithm.

```
bash extract_frames.sh
```

7.22.7 Step 6. Generate file list for GYM99 based on extracted subactions

You can use the following script to generate train / val lists for GYM99.

```
python generate_file_list.py
```

7.22.8 Step 7. Folder Structure

After the whole data pipeline for GYM preparation. You can get the subaction clips, event clips, raw videos and GYM99 train/val lists.

In the context of the whole project (for GYM only), the full folder structure will look like:

```
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
```

For training and evaluating on GYM, please refer to [getting_started](getting_started.md).

7.23 HMDB51

7.23.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/hmdb51/`.

To run the bash scripts below, you need to install `unrar`. you can install it by `sudo apt-get install unrar`, or refer to this repo by following the usage and taking `zzunrar.sh` script for easy installation without sudo.

7.23.2 Step 1. Prepare Annotations

First of all, you can run the following script to prepare annotations.

```
bash download_annotations.sh
```

7.23.3 Step 2. Prepare Videos

Then, you can run the following script to prepare videos.

```
bash download_videos.sh
```

7.23.4 Step 3. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/hmdb51_extracted/
ln -s /mnt/SSD/hmdb51_extracted/ ../../../data/hmdb51/rawframes
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

```
bash extract_rgb_frames.sh
```

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

```
bash extract_rgb_frames_opencv.sh
```

If both are required, run the following script to extract frames using "tvl1" algorithm.

```
bash extract_frames.sh
```

7.23.5 Step 4. Generate File List

you can run the follow script to generate file list in the format of rawframes and videos.

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

7.23.6 Step 5. Check Directory Structure

After the whole data process for HMDB51 preparation, you will get the rawframes (RGB + Flow), videos and annotation files for HMDB51.

In the context of the whole project (for HMDB51 only), the folder structure will look like:

```
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
                ├── 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
                ├── ...
```

(continues on next page)

(continued from previous page)

```
│   │   │   │   ├── winKen_wave_u_cm_np1_ri_bad_1
```

For training and evaluating on HMDB51, please refer to getting_started.md.

7.24 HVU

7.24.1 Introduction

```
@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}
}
```

For basic dataset information, please refer to the official project and the paper. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/hvu/`.

7.24.2 Step 1. Prepare Annotations

First of all, you can run the following script to prepare annotations.

```
bash download_annotations.sh
```

Besides, you need to run the following command to parse the tag list of HVU.

```
python parse_tag_list.py
```

7.24.3 Step 2. Prepare Videos

Then, you can run the following script to prepare videos. The codes are adapted from the official crawler. Note that this might take a long time.

```
bash download_videos.sh
```

7.24.4 Step 3. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

You can use the following script to extract both RGB and Flow frames.

```
bash extract_frames.sh
```

By default, we generate frames with short edge resized to 256. More details can be found in [data_preparation](data_preparation.md)

7.24.5 Step 4. Generate File List

You can run the follow scripts to generate file list in the format of videos and rawframes, respectively.

```
bash generate_videos_filelist.sh
## execute the command below when rawframes are ready
bash generate_rawframes_filelist.sh
```

7.24.6 Step 5. Generate File List for Each Individual Tag Categories

This part is **optional** if you don't want to train models on HVU for a specific tag category.

The file list generated in step 4 contains labels of different categories. These file lists can only be handled with HVUDataset and used for multi-task learning of different tag categories. The component `LoadHVULabel` is needed to load the multi-category tags, and the `HVULoss` should be used to train the model.

If you only want to train video recognition models for a specific tag category, i.e. you want to train a recognition model on HVU which only handles tags in the category `action`, we recommend you to use the following command to generate file lists for the specific tag category. The new list, which only contains tags of a specific category, can be handled with `VideoDataset` or `RawframeDataset`. The recognition models can be trained with `BCELossWithLogits`.

The following command generates file list for the tag category ${category}, note that the tag category you specified should be in the 6 tag categories available in HVU: ['action', 'attribute', 'concept', 'event', 'object', 'scene'].

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

The filename of the generated file list for ${category} is generated by replacing `hvu` in the original filename with `hvu_${category}`. For example, if the original filename is `hvu_train.json`, the filename of the file list for action is `hvu_action_train.json`.

7.24.7 Step 6. Folder Structure

After the whole data pipeline for HVU preparation. you can get the rawframes (RGB + Flow), videos and annotation files for HVU.

In the context of the whole project (for HVU only), the full folder structure will look like:

```
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
```

(continues on next page)

(continued from previous page)

```
│   │   ├── rawframes_train
│   │   ├── rawframes_val
```

For training and evaluating on HVU, please refer to [getting_started](getting_started.md).

7.25 Jester

7.25.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/jester/`.

7.25.2 Step 1. Prepare Annotations

First of all, you have to sign in and download annotations to `$MMACTION2/data/jester/annotations` on the official website.

7.25.3 Step 2. Prepare RGB Frames

Since the jester website doesn't provide the original video data and only extracted RGB frames are available, you have to directly download RGB frames from jester website.

You can download all RGB frame parts on jester website to `$MMACTION2/data/jester/` and use the following command to extract.

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

For users who only want to use RGB frames, you can skip to step 5 to generate file lists in the format of rawframes. Since the prefix of official JPGs is "%05d.jpg" (e.g., "00001.jpg"), we add `"filename_tmpl='{:05}.jpg'"` to the dict of `data.train`, `data.val` and `data.test` in the config files related with jester like this:

```
data = dict(
    videos_per_gpu=16,
    workers_per_gpu=2,
    train=dict(
        type=dataset_type,
```

(continues on next page)

(continued from previous page)

```
        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 Step 3. Extract Flow

This part is **optional** if you only want to use RGB frames.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/jester_extracted/
ln -s /mnt/SSD/jester_extracted/ ../../../data/jester/rawframes
```

Then, you can run the following script to extract optical flow based on RGB frames.

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

7.25.5 Step 4. Encode Videos

This part is **optional** if you only want to use RGB frames.

You can run the following script to encode videos.

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

7.25.6 Step 5. Generate File List

You can run the follow script to generate file list in the format of rawframes and videos.

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

7.25.7 Step 5. Check Directory Structure

After the whole data process for Jester preparation, you will get the rawframes (RGB + Flow), and annotation files for Jester.

In the context of the whole project (for Jester only), the folder structure will look like:

```
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
│   │   │   │   ├── flow_y_00002.jpg
│   │   │   │   ├── ...
│   │   │   ├── 2
│   │   │   ├── ...
```

For training and evaluating on Jester, please refer to *getting_started.md*.

7.26 JHMDB

7.26.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/jhmdb/`.

7.26.2 Download and Extract

You can download the RGB frames, optical flow and ground truth annotations from google drive. The data are provided from MOC, which is adapted from act-detector.

After downloading the `JHMDB.tar.gz` file and put it in `$MMACTION2/tools/data/jhmdb/`, you can run the following command to extract.

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

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/JHMDB/
ln -s /mnt/SSD/JHMDB/ ../../../data/jhmdb
```

7.26.3 Check Directory Structure

After extracting, you will get the `FlowBrox04` directory, `Frames` directory and `JHMDB-GT.pkl` for JHMDB.

In the context of the whole project (for JHMDB only), the folder structure will look like:

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
│   ├── jhmdb
│   │   ├── FlowBrox04
│   │   │   ├── brush_hair
│   │   │   │   ├── April_09_brush_hair_u_nm_np1_ba_goo_0
│   │   │   │   │   ├── 00001.jpg
│   │   │   │   │   ├── 00002.jpg
│   │   │   │   │   ├── ...
```

(continues on next page)

(continued from previous page)

```
│   │   │   │   │   ├── 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
```

Note: The `JHMDB-GT.pkl` exists as a cache, it contains 6 items as follows:

1. `labels` (list): List of the 21 labels.
2. `gttubes` (dict): Dictionary that contains the ground truth tubes for each video. A **gttube** is dictionary that associates with each index of label and a list of tubes. A **tube** is a numpy array with `nframes` rows and 5 columns, each col is in format like `<frame index> <x1> <y1> <x2> <y2>`.
3. `nframes` (dict): Dictionary that contains the number of frames for each video, like `'walk/Panic_in_the_Streets_walk_u_cm_np1_ba_med_5': 16`.
4. `train_videos` (list): A list with `nsplits=1` elements, each one containing the list of training videos.
5. `test_videos` (list): A list with `nsplits=1` elements, each one containing the list of testing videos.
6. `resolution` (dict): Dictionary that outputs a tuple (h,w) of the resolution for each video, like `'pour/Bartender_School_Students_Practice_pour_u_cm_np1_fr_med_1': (240, 320)`.

7.27 Kinetics-[400/600/700]

7.27.1 Introduction

```
@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}
}
```

For basic dataset information, please refer to the official website. The scripts can be used for preparing kinetics400, kinetics600, kinetics700. To prepare different version of kinetics, you need to replace `${DATASET}` in the following examples with the specific dataset name. The choices of dataset names are `kinetics400`, `kinetics600` and `kinetics700`. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/${DATASET}/`.

Note: Because of the expirations of some YouTube links, the sizes of kinetics dataset copies may be different. Here are the sizes of our kinetics dataset copies that used to train all checkpoints.

7.27.2 Step 1. Prepare Annotations

First of all, you can run the following script to prepare annotations by downloading from the official website.

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

Since some video urls are invalid, the number of video items in current official annotations are less than the original official ones. So we provide an alternative way to download the older one as a reference. Among these, the annotation files of Kinetics400 and Kinetics600 are from official crawler, the annotation files of Kinetics700 are from website downloaded in 05/02/2021.

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

7.27.3 Step 2. Prepare Videos

Then, you can run the following script to prepare videos. The codes are adapted from the official crawler. Note that this might take a long time.

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

Important: If you have already downloaded video dataset using the download script above, you must replace all whitespaces in the class name for ease of processing by running

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

For better decoding speed, you can resize the original videos into smaller sized, densely encoded version by:

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

You can also download from Academic Torrents (kinetics400 & kinetics700 with short edge 256 pixels are available) and cvdfoundation/kinetics-dataset (Host by Common Visual Data Foundation and Kinetics400/Kinetics600/Kinetics-700-2020 are available)

7.27.4 Step 3. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance. And you can run the following script to soft link the extracted frames.

```
## execute these two line (Assume the SSD is mounted at "/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/
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

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

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

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

If both are required, run the following script to extract frames.

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

The commands above can generate images with new short edge 256. If you want to generate images with short edge 320 (320p), or with fix size 340x256, you can change the args `--new-short 256` to `--new-short 320` or `--new-width 340 --new-height 256`. More details can be found in *data_preparation*

7.27.5 Step 4. Generate File List

you can run the follow scripts to generate file list in the format of videos and rawframes, respectively.

```
bash generate_videos_filelist.sh ${DATASET}
## execute the command below when rawframes are ready
bash generate_rawframes_filelist.sh ${DATASET}
```

7.27.6 Step 5. Folder Structure

After the whole data pipeline for Kinetics preparation. you can get the rawframes (RGB + Flow), videos and annotation files for Kinetics.

In the context of the whole project (for Kinetics only), the *minimal* folder structure will look like: (*minimal* means that some data are not necessary: for example, you may want to evaluate kinetics using the original video format.)

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

For training and evaluating on Kinetics, please refer to *getting_started*.

7.28 Moments in Time

7.28.1 Introduction

```
@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},
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/mit/`.

7.28.2 Step 1. Prepare Annotations and Videos

First of all, you have to visit the official website, fill in an application form for downloading the dataset. Then you will get the download link. You can use `bash preprocess_data.sh` to prepare annotations and videos. However, the download command is missing in that script. Remember to download the dataset to the proper place follow the comment in this script.

For better decoding speed, you can resize the original videos into smaller sized, densely encoded version by:

```
python ../resize_videos.py ../../../data/mit/videos/ ../../../data/mit/videos_256p_dense_
↪cache --dense --level 2
```

7.28.3 Step 2. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance. And you can run the following script to soft link the extracted frames.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/mit_extracted/
ln -s /mnt/SSD/mit_extracted/ ../../../data/mit/rawframes
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

```
bash extract_rgb_frames.sh
```

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

```
bash extract_rgb_frames_opencv.sh
```

If both are required, run the following script to extract frames.

```
bash extract_frames.sh
```

7.28.4 Step 4. Generate File List

you can run the follow script to generate file list in the format of rawframes and videos.

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

7.28.5 Step 5. Check Directory Structure

After the whole data process for Moments in Time preparation, you will get the rawframes (RGB + Flow), videos and annotation files for Moments in Time.

In the context of the whole project (for Moments in Time only), the folder structure will look like:

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

For training and evaluating on Moments in Time, please refer to *getting_started.md*.

7.29 Multi-Moments in Time

7.29.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/mmit/`.

7.29.2 Step 1. Prepare Annotations and Videos

First of all, you have to visit the official website, fill in an application form for downloading the dataset. Then you will get the download link. You can use `bash preprocess_data.sh` to prepare annotations and videos. However, the download command is missing in that script. Remember to download the dataset to the proper place follow the comment in this script.

For better decoding speed, you can resize the original videos into smaller sized, densely encoded version by:

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

7.29.3 Step 2. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

First, you can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/mmit_extracted/
ln -s /mnt/SSD/mmit_extracted/ ../../../data/mmit/rawframes
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

```
bash extract_rgb_frames.sh
```

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

```
bash extract_rgb_frames_opencv.sh
```

If both are required, run the following script to extract frames using "tvl1" algorithm.

```
bash extract_frames.sh
```

7.29.4 Step 3. Generate File List

you can run the follow script to generate file list in the format of rawframes or videos.

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

7.29.5 Step 4. Check Directory Structure

After the whole data process for Multi-Moments in Time preparation, you will get the rawframes (RGB + Flow), videos and annotation files for Multi-Moments in Time.

In the context of the whole project (for Multi-Moments in Time only), the folder structure will look like:

```
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
        │   │   ├── ...
        │   │   ├── 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
            └── ...
```

For training and evaluating on Multi-Moments in Time, please refer to *getting_started.md*.

7.30 OmniSource

7.30.1 Introduction

```
@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}
}
```

We release a subset of the OmniSource web dataset used in the paper Omni-sourced Webly-supervised Learning for Video Recognition. Since all web dataset in OmniSource are built based on the Kinetics-400 taxonomy, we select those web data related to the 200 classes in Mini-Kinetics subset (which is proposed in Rethinking Spatiotemporal Feature Learning: Speed-Accuracy Trade-offs in Video Classification).

We provide data from all sources that are related to the 200 classes in Mini-Kinetics (including Kinetics trimmed clips, Kinetics untrimmed videos, images from Google and Instagram, video clips from Instagram). To obtain this dataset, please first fill in the request form. We will share the download link to you after your request is received. Since we release all data crawled from the web without any filtering, the dataset is large and it may take some time to download them. We describe the size of the datasets in the following table:

The file structure of our uploaded OmniSource dataset looks like:

```
OmniSource/
├── annotations
│   ├── googleimage_200
│   │   ├── googleimage_200.txt                       File list of all valid images␣
↪crawled from Google.
│   │   ├── tsn_8seg_googleimage_200_duplicate.txt    Positive file list of images␣
↪crawled from Google, which is similar to a validation example.
│   │   ├── tsn_8seg_googleimage_200.txt              Positive file list of images␣
↪crawled from Google, filtered by the teacher model.
│   │   └── tsn_8seg_googleimage_200_wodup.txt        Positive file list of images␣
↪crawled from Google, filtered by the teacher model, after de-duplication.
│   ├── 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                              The list of action names of the␣
↪200 classes in MiniKinetics.
│   ├── K400_to_MiniKinetics_classidx_mapping.json    The index mapping from Kinetics-
↪400 to MiniKinetics.
│   ├── kinetics_200
│   │   ├── k200_train.txt
│   │   └── k200_val.txt
│   ├── kinetics_raw_200
```

(continues on next page)

(continued from previous page)

```
│   │   └── 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                                   (10 volumes)
│   ├── vol_0.tar
│   ├── ...
│   └── vol_9.tar
├── insimage_200                                      (10 volumes)
│   ├── vol_0.tar
│   ├── ...
│   └── vol_9.tar
├── insvideo_200                                      (20 volumes)
│   ├── vol_00.tar
│   ├── ...
│   └── vol_19.tar
├── kinetics_200_train
│   └── kinetics_200_train.tar
├── kinetics_200_val
│   └── kinetics_200_val.tar
└── kinetics_raw_200_train                            (16 volumes)
    ├── vol_0.tar
    ├── ...
    └── vol_15.tar
```

7.30.2 Data Preparation

For data preparation, you need to first download those data. For `kinetics_200` and 3 web datasets: `googleimage_200`, `insimage_200` and `insvideo_200`, you just need to extract each volume and merge their contents.

For Kinetics raw videos, since loading long videos is very heavy, you need to first trim it into clips. Here we provide a script named `trim_raw_video.py`. It trims a long video into 10-second clips and remove the original raw video. You can use it to trim the Kinetics raw video.

The data should be placed in `data/OmniSource/`. When data preparation finished, the folder structure of `data/OmniSource` looks like (We omit the files not needed in training & testing for simplicity):

```
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
```

(continues on next page)

(continued from previous page)

```
│   │   └── 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
│   │   ├── ...
│   │   ├── 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 Skeleton Dataset

```
@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 Introduction

We release the skeleton annotations used in Revisiting Skeleton-based Action Recognition. By default, we use Faster-RCNN with ResNet50 backbone for human detection and HRNet-w32 for single person pose estimation. For FineGYM, we use Ground-Truth bounding boxes for the athlete instead of detection bounding boxes. Currently, we release the skeleton annotations for FineGYM and NTURGB-D Xsub split. Other annotations will be soo released.

7.31.2 Prepare Annotations

Currently, we support HMDB51, UCF101, FineGYM and NTURGB+D. For FineGYM, you can execute following scripts to prepare the annotations.

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

Due to Conditions of Use of the NTURGB+D dataset, we can not directly release the annotations used in our experiments. So that we provide a script to generate pose annotations for videos in NTURGB+D datasets, which generate a dictionary and save it as a single pickle file. You can create a list which contain all annotation dictionaries of corresponding videos and save them as a pickle file. Then you can get the `ntu60_xsub_train.pkl`, `ntu60_xsub_val.pkl`, `ntu120_xsub_train.pkl`, `ntu120_xsub_val.pkl` that we used in training.

For those who have not enough computations for pose extraction, we provide the outputs of the above pipeline here, corresponding to 4 different splits of NTURGB+D datasets:

- 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

To generate 2D pose annotations for a single video, first, you need to install mmdetection and mmpose from src code. After that, you need to replace the placeholder `mmdet_root` and `mmpose_root` in `ntu_pose_extraction.py` with your installation path. Then you can use following scripts for NTURGB+D video pose extraction:

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

After you get pose annotations for all videos in a dataset split, like `ntu60_xsub_val`. You can gather them into a single list and save the list as `ntu60_xsub_val.pkl`. You can use those larger pickle files for training and testing.

7.31.3 The Format of PoseC3D Annotations

Here we briefly introduce the format of PoseC3D Annotations, we will take `gym_train.pkl` as an example: the content of `gym_train.pkl` is a list of length 20484, each item is a dictionary that is the skeleton annotation of one video. Each dictionary has following fields:

- keypoint: The keypoint coordinates, which is a numpy array of the shape N (##person) x T (temporal length) x K (#keypoints, 17 in our case) x 2 (x, y coordinate).
- keypoint_score: The keypoint confidence scores, which is a numpy array of the shape N (##person) x T (temporal length) x K (#keypoints, 17 in our case).
- frame_dir: The corresponding video name.
- label: The action category.
- img_shape: The image shape of each frame.
- original_shape: Same as above.
- total_frames: The temporal length of the video.

For training with your custom dataset, you can refer to Custom Dataset Training.

7.31.4 Visualization

For skeleton data visualization, you need also to prepare the RGB videos. Please refer to visualize_heatmap_volume for detailed process. Here we provide some visualization examples from NTU-60 and FineGYM.

7.31.5 Convert the NTU RGB+D raw skeleton data to our format (only applicable to GCN backbones)

Here we also provide the script for converting the NTU RGB+D raw skeleton data to our format. First, download the raw skeleton data of NTU-RGBD 60 and NTU-RGBD 120 from https://github.com/shahroudy/NTURGB-D.

For NTU-RGBD 60, preprocess data and convert the data format with

```
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
```

For NTU-RGBD 120, preprocess data and convert the data format with

```
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 Convert annotations from third-party projects

We provide scripts to convert skeleton annotations from third-party projects to MMAction2 formats:

- BABEL: `babel2mma2.py`

TODO:

- [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 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset paper. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/sthv1/`.

7.32.2 Step 1. Prepare Annotations

Since the official website of Something-Something V1 is currently unavailable, you can download the annotations from third-part source to `$MMACTION2/data/sthv1/` .

7.32.3 Step 2. Prepare RGB Frames

Since the official dataset doesn't provide the original video data and only extracted RGB frames are available, you have to directly download RGB frames.

You can download all compressed file parts from third-part source to `$MMACTION2/data/sthv1/` and use the following command to uncompress.

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

For users who only want to use RGB frames, you can skip to step 5 to generate file lists in the format of rawframes. Since the prefix of official JPGs is "%05d.jpg" (e.g., "00001.jpg"), users need to add `"filename_tmpl='{:05}.jpg'"` to the dict of `data.train`, `data.val` and `data.test` in the config files related with sthv1 like this:

```
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',
        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 Step 3. Extract Flow

This part is **optional** if you only want to use RGB frames.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/sthv1_extracted/
ln -s /mnt/SSD/sthv1_extracted/ ../../../data/sthv1/rawframes
```

Then, you can run the following script to extract optical flow based on RGB frames.

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

7.32.5 Step 4. Encode Videos

This part is **optional** if you only want to use RGB frames.

You can run the following script to encode videos.

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

7.32.6 Step 5. Generate File List

You can run the follow script to generate file list in the format of rawframes and videos.

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

7.32.7 Step 6. Check Directory Structure

After the whole data process for Something-Something V1 preparation, you will get the rawframes (RGB + Flow), and annotation files for Something-Something V1.

In the context of the whole project (for Something-Something V1 only), the folder structure will look like:

```
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
│   │   │   │   ├── ...
```

(continues on next page)

(continued from previous page)

```
|   |   |   ├── 2
|   |   |   ├── ...
```

For training and evaluating on Something-Something V1, please refer to *getting_started.md*.

7.33 Something-Something V2

7.33.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/sthv2/`.

7.33.2 Step 1. Prepare Annotations

First of all, you have to sign in and download annotations to `$MMACTION2/data/sthv2/annotations` on the official website.

```
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 Step 2. Prepare Videos

Then, you can download all data parts to `$MMACTION2/data/sthv2/` and use the following command to uncompress.

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

7.33.4 Step 3. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/sthv2_extracted/
ln -s /mnt/SSD/sthv2_extracted/ ../../../data/sthv2/rawframes
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

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

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

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

If both are required, run the following script to extract frames.

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

7.33.5 Step 4. Generate File List

you can run the follow script to generate file list in the format of rawframes and videos.

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

7.33.6 Step 5. Check Directory Structure

After the whole data process for Something-Something V2 preparation, you will get the rawframes (RGB + Flow), videos and annotation files for Something-Something V2.

In the context of the whole project (for Something-Something V2 only), the folder structure will look like:

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
│   ├── sthv2
│   │   ├── sthv2_{train,val}_list_rawframes.txt
│   │   ├── sthv2_{train,val}_list_videos.txt
│   │   ├── annotations
```

(continues on next page)

(continued from previous page)

```
│   │   ├── 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
│   │   │   ├── ...
```

For training and evaluating on Something-Something V2, please refer to *getting_started.md*.

7.34 THUMOS'14

7.34.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/thumos14/`.

7.34.2 Step 1. Prepare Annotations

First of all, run the following script to prepare annotations.

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

7.34.3 Step 2. Prepare Videos

Then, you can run the following script to prepare videos.

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

7.34.4 Step 3. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/thumos14_extracted/
ln -s /mnt/SSD/thumos14_extracted/ ../data/thumos14/rawframes/
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

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

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

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

If both are required, run the following script to extract frames.

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

7.34.5 Step 4. Fetch File List

This part is **optional** if you do not use SSN model.

You can run the follow script to fetch pre-computed tag proposals.

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

7.34.6 Step 5. Denormalize Proposal File

This part is **optional** if you do not use SSN model.

You can run the follow script to denormalize pre-computed tag proposals according to actual number of local rawframes.

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

7.34.7 Step 6. Check Directory Structure

After the whole data process for THUMOS'14 preparation, you will get the rawframes (RGB + Flow), videos and annotation files for THUMOS'14.

In the context of the whole project (for THUMOS'14 only), the folder structure will look like:

```
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
```

For training and evaluating on THUMOS'14, please refer to *getting_started.md*.

7.35 UCF-101

7.35.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/ucf101/`.

7.35.2 Step 1. Prepare Annotations

First of all, you can run the following script to prepare annotations.

```
bash download_annotations.sh
```

7.35.3 Step 2. Prepare Videos

Then, you can run the following script to prepare videos.

```
bash download_videos.sh
```

For better decoding speed, you can resize the original videos into smaller sized, densely encoded version by:

```
python ../resize_videos.py ../../../data/ucf101/videos/ ../../../data/ucf101/videos_256p_
↪dense_cache --dense --level 2 --ext avi
```

7.35.4 Step 3. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance. The extracted frames (RGB + Flow) will take up about 100GB.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/ucf101_extracted/
ln -s /mnt/SSD/ucf101_extracted/ ../../../data/ucf101/rawframes
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

```
bash extract_rgb_frames.sh
```

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

```
bash extract_rgb_frames_opencv.sh
```

If Optical Flow is also required, run the following script to extract flow using "tvl1" algorithm.

```
bash extract_frames.sh
```

7.35.5 Step 4. Generate File List

you can run the follow script to generate file list in the format of rawframes and videos.

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

7.35.6 Step 5. Check Directory Structure

After the whole data process for UCF-101 preparation, you will get the rawframes (RGB + Flow), videos and annotation files for UCF-101.

In the context of the whole project (for UCF-101 only), the folder structure will look like:

```
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

│   │   │   ├── 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
```

(continues on next page)

(continued from previous page)

```
│   │   │   ├── ...
│   │   │   ├── YoYo
│   │   │   │   ├── v_YoYo_g01_c01
│   │   │   │   ├── ...
│   │   │   │   ├── v_YoYo_g25_c05
```

For training and evaluating on UCF-101, please refer to *getting_started.md*.

7.36 UCF101-24

7.36.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/ucf101_24/`.

7.36.2 Download and Extract

You can download the RGB frames, optical flow and ground truth annotations from google drive. The data are provided from MOC, which is adapted from act-detector and corrected-UCF101-Annots.

Note: The annotation of this UCF101-24 is from here, which is more correct.

After downloading the `UCF101_v2.tar.gz` file and put it in `$MMACTION2/tools/data/ucf101_24/`, you can run the following command to uncompress.

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

7.36.3 Check Directory Structure

After uncompressing, you will get the `rgb-images` directory, `brox-images` directory and `UCF101v2-GT.pkl` for UCF101-24.

In the context of the whole project (for UCF101-24 only), the folder structure will look like:

```
mmaction2
├── mmaction
├── tools
├── configs
```

(continues on next page)

(continued from previous page)

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

Note: The `UCF101v2-GT.pkl` exists as a cache, it contains 6 items as follows:

1. `labels` (list): List of the 24 labels.
2. `gttubes` (dict): Dictionary that contains the ground truth tubes for each video. A **gttube** is dictionary that associates with each index of label and a list of tubes. A **tube** is a numpy array with `nframes` rows and 5 columns, each col is in format like `<frame index> <x1> <y1> <x2> <y2>`.
3. `nframes` (dict): Dictionary that contains the number of frames for each video, like `'HorseRiding/v_HorseRiding_g05_c02': 151`.
4. `train_videos` (list): A list with `nsplits=1` elements, each one containing the list of training videos.
5. `test_videos` (list): A list with `nsplits=1` elements, each one containing the list of testing videos.
6. `resolution` (dict): Dictionary that outputs a tuple (h,w) of the resolution for each video, like `'FloorGymnastics/v_FloorGymnastics_g09_c03': (240, 320)`.

7.37 ActivityNet

7.37.1 Introduction

```
@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}
}
```

For basic dataset information, please refer to the official website. For action detection, you can either use the ActivityNet rescaled feature provided in this repo or extract feature with mmaction2 (which has better performance). We release both pipeline. Before we start, please make sure that current working directory is `$MMACTION2/tools/data/activitynet/`.

7.37.2 Option 1: Use the ActivityNet rescaled feature provided in this repo

Step 1. Download Annotations

First of all, you can run the following script to download annotation files.

```
bash download_feature_annotations.sh
```

Step 2. Prepare Videos Features

Then, you can run the following script to download activitynet features.

```
bash download_features.sh
```

Step 3. Process Annotation Files

Next, you can run the following script to process the downloaded annotation files for training and testing. It first merges the two annotation files together and then separates the annoations by `train`, `val` and `test`.

```
python process_annotations.py
```

7.37.3 Option 2: Extract ActivityNet feature using MMAction2 with all videos provided in official website

Step 1. Download Annotations

First of all, you can run the following script to download annotation files.

```
bash download_annotations.sh
```

Step 2. Prepare Videos

Then, you can run the following script to prepare videos. The codes are adapted from the official crawler. Note that this might take a long time.

```
bash download_videos.sh
```

Since some videos in the ActivityNet dataset might be no longer available on YouTube, official website has made the full dataset available on Google and Baidu drives. To accommodate missing data requests, you can fill in this request form provided in official download page to have a 7-day-access to download the videos from the drive folders.

We also provide download steps for annotations from BSN repo

```
bash download_bsn_videos.sh
```

For this case, the downloading scripts update the annotation file after downloading to make sure every video in it exists.

Step 3. Extract RGB and Flow

Before extracting, please refer to *install.md* for installing denseflow.

Use following scripts to extract both RGB and Flow.

```
bash extract_frames.sh
```

The command above can generate images with new short edge 256. If you want to generate images with short edge 320 (320p), or with fix size 340x256, you can change the args `--new-short 256` to `--new-short 320` or `--new-width 340 --new-height 256`. More details can be found in [data_preparation](data_preparation.md)

Step 4. Generate File List for ActivityNet Finetuning

With extracted frames, you can generate video-level or clip-level lists of rawframes, which can be used for ActivityNet Finetuning.

```
python generate_rawframes_filelist.py
```

Step 5. Finetune TSN models on ActivityNet

You can use ActivityNet configs in `configs/recognition/tsn` to finetune TSN models on ActivityNet. You need to use Kinetics models for pretraining. Both RGB models and Flow models are supported.

Step 6. Extract ActivityNet Feature with finetuned ckpts

After finetuning TSN on ActivityNet, you can use it to extract both RGB and Flow feature.

```
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
```

(continues on next page)

(continued from previous page)

```
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
```

After feature extraction, you can use our post processing scripts to concat RGB and Flow feature, generate the `100-t X 400-d` feature for Action Detection.

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

7.37.4 Final Step. Check Directory Structure

After the whole data pipeline for ActivityNet preparation, you will get the features, videos, frames and annotation files.

In the context of the whole project (for ActivityNet only), the folder structure will look like:

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

(if Option 1 used)
│   │   ├── 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
│   │   │   │   ├── ..

(if Option 2 used)
│   │   ├── 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
```

(continues on next page)

(continued from previous page)

```
│   │   │   │   ├── flow_x_00000.jpg
│   │   │   │   ├── flow_y_00000.jpg
│   │   │   │   ├── ..
│   │   │   └── ..
```

For training and evaluating on ActivityNet, please refer to *getting_started.md*.

7.38 AVA

7.38.1 Introduction

```
@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}
}
```

For basic dataset information, please refer to the official website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/ava/`.

7.38.2 Step 1. Prepare Annotations

First of all, you can run the following script to prepare annotations.

```
bash download_annotations.sh
```

This command will download `ava_v2.1.zip` for AVA `v2.1` annotation. If you need the AVA `v2.2` annotation, you can try the following script.

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

7.38.3 Step 2. Prepare Videos

Then, use the following script to prepare videos. The codes are adapted from the official crawler. Note that this might take a long time.

```
bash download_videos.sh
```

Or you can use the following command to downloading AVA videos in parallel using a python script.

```
bash download_videos_parallel.sh
```

Note that if you happen to have sudoer or have GNU parallel on your machine, you can speed up the procedure by downloading in parallel.

```
## sudo apt-get install parallel
bash download_videos_gnu_parallel.sh
```

7.38.4 Step 3. Cut Videos

Cut each video from its 15th to 30th minute and make them at 30 fps.

```
bash cut_videos.sh
```

7.38.5 Step 4. Extract RGB and Flow

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance. And you can run the following script to soft link the extracted frames.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/ava_extracted/
ln -s /mnt/SSD/ava_extracted/ ../data/ava/rawframes/
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

```
bash extract_rgb_frames.sh
```

If you didn't install denseflow, you can still extract RGB frames using ffmpeg by the following script.

```
bash extract_rgb_frames_ffmpeg.sh
```

If both are required, run the following script to extract frames.

```
bash extract_frames.sh
```

7.38.6 Step 5. Fetch Proposal Files

The scripts are adapted from FAIR's Long-Term Feature Banks.

Run the following scripts to fetch the pre-computed proposal list.

```
bash fetch_ava_proposals.sh
```

7.38.7 Step 6. Folder Structure

After the whole data pipeline for AVA preparation. you can get the rawframes (RGB + Flow), videos and annotation files for AVA.

In the context of the whole project (for AVA only), the *minimal* folder structure will look like: (*minimal* means that some data are not necessary: for example, you may want to evaluate AVA using the original video format.)

```
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
│   │   │   │   ├── ...
```

For training and evaluating on AVA, please refer to [getting_started](getting_started.md).

7.38.8 Reference

1. O. Tange (2018): GNU Parallel 2018, March 2018, https://doi.org/10.5281/zenodo.1146014

7.39 Diving48

7.39.1 Introduction

```
@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)},
```

(continues on next page)

(continued from previous page)

```
  pages={513--528},
  year={2018}
}
```

For basic dataset information, you can refer to the official dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/diving48/`.

7.39.2 Step 1. Prepare Annotations

You can run the following script to download annotations (considering the correctness of annotation files, we only download V2 version here).

```
bash download_annotations.sh
```

7.39.3 Step 2. Prepare Videos

You can run the following script to download videos.

```
bash download_videos.sh
```

7.39.4 Step 3. Prepare RGB and Flow

This part is **optional** if you only want to use the video loader.

The frames provided in official compressed file are not complete. You may need to go through the following extraction steps to get the complete frames.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/diving48_extracted/
ln -s /mnt/SSD/diving48_extracted/ ../../../data/diving48/rawframes
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

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

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

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

If both are required, run the following script to extract frames.

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

7.39.5 Step 4. Generate File List

you can run the follow script to generate file list in the format of rawframes and videos.

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

7.39.6 Step 5. Check Directory Structure

After the whole data process for Diving48 preparation, you will get the rawframes (RGB + Flow), videos and annotation files for Diving48.

In the context of the whole project (for Diving48 only), the folder structure will look like:

```
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
        |   ├── _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
        |   ├── ...
```

For training and evaluating on Diving48, please refer to *getting_started.md*.

7.40 GYM

7.40.1 Introduction

```
@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}
}
```

For basic dataset information, please refer to the official project and the paper. We currently provide the data preprocessing pipeline for GYM99. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/gym/`.

7.40.2 Step 1. Prepare Annotations

First of all, you can run the following script to prepare annotations.

```
bash download_annotations.sh
```

7.40.3 Step 2. Prepare Videos

Then, you can run the following script to prepare videos. The codes are adapted from the official crawler. Note that this might take a long time.

```
bash download_videos.sh
```

7.40.4 Step 3. Trim Videos into Events

First, you need to trim long videos into events based on the annotation of GYM with the following scripts.

```
python trim_event.py
```

7.40.5 Step 4. Trim Events into Subactions

Then, you need to trim events into subactions based on the annotation of GYM with the following scripts. We use the two stage trimming for better efficiency (trimming multiple short clips from a long video can be extremely inefficient, since you need to go over the video many times).

```
python trim_subaction.py
```

7.40.6 Step 5. Extract RGB and Flow

This part is **optional** if you only want to use the video loader for RGB model training.

Before extracting, please refer to *install.md* for installing denseflow.

Run the following script to extract both rgb and flow using "tvl1" algorithm.

```
bash extract_frames.sh
```

7.40.7 Step 6. Generate file list for GYM99 based on extracted subactions

You can use the following script to generate train / val lists for GYM99.

```
python generate_file_list.py
```

7.40.8 Step 7. Folder Structure

After the whole data pipeline for GYM preparation. You can get the subaction clips, event clips, raw videos and GYM99 train/val lists.

In the context of the whole project (for GYM only), the full folder structure will look like:

```
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
```

For training and evaluating on GYM, please refer to [getting_started](getting_started.md).

7.41 HMDB51

7.41.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/hmdb51/`.

To run the bash scripts below, you need to install `unrar`. you can install it by `sudo apt-get install unrar`, or refer to this repo by following the usage and taking `zzunrar.sh` script for easy installation without sudo.

7.41.2 Step 1. Prepare Annotations

First of all, you can run the following script to prepare annotations.

```
bash download_annotations.sh
```

7.41.3 Step 2. Prepare Videos

Then, you can run the following script to prepare videos.

```
bash download_videos.sh
```

7.41.4 Step 3. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/hmdb51_extracted/
ln -s /mnt/SSD/hmdb51_extracted/ ../../../data/hmdb51/rawframes
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

```
bash extract_rgb_frames.sh
```

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

```
bash extract_rgb_frames_opencv.sh
```

If both are required, run the following script to extract frames using "tvl1" algorithm.

```
bash extract_frames.sh
```

7.41.5 Step 4. Generate File List

you can run the follow script to generate file list in the format of rawframes and videos.

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

7.41.6 Step 5. Check Directory Structure

After the whole data process for HMDB51 preparation, you will get the rawframes (RGB + Flow), videos and annotation files for HMDB51.

In the context of the whole project (for HMDB51 only), the folder structure will look like:

```
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
                ├── 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
                ├── ...
```

(continues on next page)

(continued from previous page)

```
│   │   │   │   ├── winKen_wave_u_cm_np1_ri_bad_1
```

For training and evaluating on HMDB51, please refer to getting_started.md.

7.42 HVU

7.42.1 Introduction

```
@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}
}
```

For basic dataset information, please refer to the official project and the paper. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/hvu/`.

7.42.2 Step 1. Prepare Annotations

First of all, you can run the following script to prepare annotations.

```
bash download_annotations.sh
```

Besides, you need to run the following command to parse the tag list of HVU.

```
python parse_tag_list.py
```

7.42.3 Step 2. Prepare Videos

Then, you can run the following script to prepare videos. The codes are adapted from the official crawler. Note that this might take a long time.

```
bash download_videos.sh
```

7.42.4 Step 3. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

You can use the following script to extract both RGB and Flow frames.

```
bash extract_frames.sh
```

By default, we generate frames with short edge resized to 256. More details can be found in [data_preparation](data_preparation.md)

7.42.5 Step 4. Generate File List

You can run the follow scripts to generate file list in the format of videos and rawframes, respectively.

```
bash generate_videos_filelist.sh
## execute the command below when rawframes are ready
bash generate_rawframes_filelist.sh
```

7.42.6 Step 5. Generate File List for Each Individual Tag Categories

This part is **optional** if you don't want to train models on HVU for a specific tag category.

The file list generated in step 4 contains labels of different categories. These file lists can only be handled with HVUDataset and used for multi-task learning of different tag categories. The component `LoadHVULabel` is needed to load the multi-category tags, and the `HVULoss` should be used to train the model.

If you only want to train video recognition models for a specific tag category, i.e. you want to train a recognition model on HVU which only handles tags in the category `action`, we recommend you to use the following command to generate file lists for the specific tag category. The new list, which only contains tags of a specific category, can be handled with `VideoDataset` or `RawframeDataset`. The recognition models can be trained with `BCELossWithLogits`.

The following command generates file list for the tag category ${category}, note that the tag category you specified should be in the 6 tag categories available in HVU: ['action', 'attribute', 'concept', 'event', 'object', 'scene'].

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

The filename of the generated file list for ${category} is generated by replacing `hvu` in the original filename with `hvu_${category}`. For example, if the original filename is `hvu_train.json`, the filename of the file list for action is `hvu_action_train.json`.

7.42.7 Step 6. Folder Structure

After the whole data pipeline for HVU preparation. you can get the rawframes (RGB + Flow), videos and annotation files for HVU.

In the context of the whole project (for HVU only), the full folder structure will look like:

```
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
```

(continues on next page)

(continued from previous page)

```
│   │   ├── rawframes_train
│   │   ├── rawframes_val
```

For training and evaluating on HVU, please refer to [getting_started](getting_started.md).

7.43 Jester

7.43.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/jester/`.

7.43.2 Step 1. Prepare Annotations

First of all, you have to sign in and download annotations to `$MMACTION2/data/jester/annotations` on the official website.

7.43.3 Step 2. Prepare RGB Frames

Since the jester website doesn't provide the original video data and only extracted RGB frames are available, you have to directly download RGB frames from jester website.

You can download all RGB frame parts on jester website to `$MMACTION2/data/jester/` and use the following command to extract.

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

For users who only want to use RGB frames, you can skip to step 5 to generate file lists in the format of rawframes. Since the prefix of official JPGs is "%05d.jpg" (e.g., "00001.jpg"), we add `"filename_tmpl='{:05}.jpg'"` to the dict of `data.train`, `data.val` and `data.test` in the config files related with jester like this:

```
data = dict(
    videos_per_gpu=16,
    workers_per_gpu=2,
    train=dict(
        type=dataset_type,
```

(continues on next page)

(continued from previous page)

```
        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 Step 3. Extract Flow

This part is **optional** if you only want to use RGB frames.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/jester_extracted/
ln -s /mnt/SSD/jester_extracted/ ../../../data/jester/rawframes
```

Then, you can run the following script to extract optical flow based on RGB frames.

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

7.43.5 Step 4. Encode Videos

This part is **optional** if you only want to use RGB frames.

You can run the following script to encode videos.

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

7.43.6 Step 5. Generate File List

You can run the follow script to generate file list in the format of rawframes and videos.

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

7.43.7 Step 5. Check Directory Structure

After the whole data process for Jester preparation, you will get the rawframes (RGB + Flow), and annotation files for Jester.

In the context of the whole project (for Jester only), the folder structure will look like:

```
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
│   │   │   │   ├── flow_y_00002.jpg
│   │   │   │   ├── ...
│   │   │   ├── 2
│   │   │   ├── ...
```

For training and evaluating on Jester, please refer to *getting_started.md*.

7.44 JHMDB

7.44.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/jhmdb/`.

7.44.2 Download and Extract

You can download the RGB frames, optical flow and ground truth annotations from google drive. The data are provided from MOC, which is adapted from act-detector.

After downloading the `JHMDB.tar.gz` file and put it in `$MMACTION2/tools/data/jhmdb/`, you can run the following command to extract.

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

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/JHMDB/
ln -s /mnt/SSD/JHMDB/ ../../../data/jhmdb
```

7.44.3 Check Directory Structure

After extracting, you will get the `FlowBrox04` directory, `Frames` directory and `JHMDB-GT.pkl` for JHMDB.

In the context of the whole project (for JHMDB only), the folder structure will look like:

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
    ├── jhmdb
    |   ├── FlowBrox04
    |   |   ├── brush_hair
    |   |   |   ├── April_09_brush_hair_u_nm_np1_ba_goo_0
    |   |   |   |   ├── 00001.jpg
    |   |   |   |   ├── 00002.jpg
    |   |   |   |   ├── ...
```

(continues on next page)

(continued from previous page)

```
│   │   │   │   │   ├── 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
```

Note: The `JHMDB-GT.pkl` exists as a cache, it contains 6 items as follows:

1. `labels` (list): List of the 21 labels.
2. `gttubes` (dict): Dictionary that contains the ground truth tubes for each video. A **gttube** is dictionary that associates with each index of label and a list of tubes. A **tube** is a numpy array with `nframes` rows and 5 columns, each col is in format like `<frame index> <x1> <y1> <x2> <y2>`.
3. `nframes` (dict): Dictionary that contains the number of frames for each video, like `'walk/Panic_in_the_Streets_walk_u_cm_np1_ba_med_5': 16`.
4. `train_videos` (list): A list with `nsplits=1` elements, each one containing the list of training videos.
5. `test_videos` (list): A list with `nsplits=1` elements, each one containing the list of testing videos.
6. `resolution` (dict): Dictionary that outputs a tuple (h,w) of the resolution for each video, like `'pour/Bartender_School_Students_Practice_pour_u_cm_np1_fr_med_1': (240, 320)`.

7.45 Kinetics-[400/600/700]

7.45.1 Introduction

```
@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}
}
```

For basic dataset information, please refer to the official website. The scripts can be used for preparing kinetics400, kinetics600, kinetics700. To prepare different version of kinetics, you need to replace `${DATASET}` in the following examples with the specific dataset name. The choices of dataset names are `kinetics400`, `kinetics600` and `kinetics700`. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/${DATASET}/`.

Note: Because of the expirations of some YouTube links, the sizes of kinetics dataset copies may be different. Here are the sizes of our kinetics dataset copies that used to train all checkpoints.

7.45.2 Step 1. Prepare Annotations

First of all, you can run the following script to prepare annotations by downloading from the official website.

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

Since some video urls are invalid, the number of video items in current official annotations are less than the original official ones. So we provide an alternative way to download the older one as a reference. Among these, the annotation files of Kinetics400 and Kinetics600 are from official crawler, the annotation files of Kinetics700 are from website downloaded in 05/02/2021.

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

7.45.3 Step 2. Prepare Videos

Then, you can run the following script to prepare videos. The codes are adapted from the official crawler. Note that this might take a long time.

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

Important: If you have already downloaded video dataset using the download script above, you must replace all whitespaces in the class name for ease of processing by running

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

For better decoding speed, you can resize the original videos into smaller sized, densely encoded version by:

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

You can also download from Academic Torrents (kinetics400 & kinetics700 with short edge 256 pixels are available) and cvdfoundation/kinetics-dataset (Host by Common Visual Data Foundation and Kinetics400/Kinetics600/Kinetics-700-2020 are available)

7.45.4 Step 3. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance. And you can run the following script to soft link the extracted frames.

```
## execute these two line (Assume the SSD is mounted at "/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/
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

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

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

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

If both are required, run the following script to extract frames.

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

The commands above can generate images with new short edge 256. If you want to generate images with short edge 320 (320p), or with fix size 340x256, you can change the args `--new-short 256` to `--new-short 320` or `--new-width 340 --new-height 256`. More details can be found in *data_preparation*

7.45.5 Step 4. Generate File List

you can run the follow scripts to generate file list in the format of videos and rawframes, respectively.

```
bash generate_videos_filelist.sh ${DATASET}
## execute the command below when rawframes are ready
bash generate_rawframes_filelist.sh ${DATASET}
```

7.45.6 Step 5. Folder Structure

After the whole data pipeline for Kinetics preparation. you can get the rawframes (RGB + Flow), videos and annotation files for Kinetics.

In the context of the whole project (for Kinetics only), the *minimal* folder structure will look like: (*minimal* means that some data are not necessary: for example, you may want to evaluate kinetics using the original video format.)

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

For training and evaluating on Kinetics, please refer to *getting_started*.

7.46 Moments in Time

7.46.1 Introduction

```
@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},
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/mit/`.

7.46.2 Step 1. Prepare Annotations and Videos

First of all, you have to visit the official website, fill in an application form for downloading the dataset. Then you will get the download link. You can use `bash preprocess_data.sh` to prepare annotations and videos. However, the download command is missing in that script. Remember to download the dataset to the proper place follow the comment in this script.

For better decoding speed, you can resize the original videos into smaller sized, densely encoded version by:

```
python ../resize_videos.py ../../../data/mit/videos/ ../../../data/mit/videos_256p_dense_
↪cache --dense --level 2
```

7.46.3 Step 2. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance. And you can run the following script to soft link the extracted frames.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/mit_extracted/
ln -s /mnt/SSD/mit_extracted/ ../../../data/mit/rawframes
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

```
bash extract_rgb_frames.sh
```

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

```
bash extract_rgb_frames_opencv.sh
```

If both are required, run the following script to extract frames.

```
bash extract_frames.sh
```

7.46.4 Step 4. Generate File List

you can run the follow script to generate file list in the format of rawframes and videos.

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

7.46.5 Step 5. Check Directory Structure

After the whole data process for Moments in Time preparation, you will get the rawframes (RGB + Flow), videos and annotation files for Moments in Time.

In the context of the whole project (for Moments in Time only), the folder structure will look like:

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

For training and evaluating on Moments in Time, please refer to *getting_started.md*.

7.47 Multi-Moments in Time

7.47.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/mmit/`.

7.47.2 Step 1. Prepare Annotations and Videos

First of all, you have to visit the official website, fill in an application form for downloading the dataset. Then you will get the download link. You can use `bash preprocess_data.sh` to prepare annotations and videos. However, the download command is missing in that script. Remember to download the dataset to the proper place follow the comment in this script.

For better decoding speed, you can resize the original videos into smaller sized, densely encoded version by:

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

7.47.3 Step 2. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

First, you can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/mmit_extracted/
ln -s /mnt/SSD/mmit_extracted/ ../../../data/mmit/rawframes
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

```
bash extract_rgb_frames.sh
```

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

```
bash extract_rgb_frames_opencv.sh
```

If both are required, run the following script to extract frames using "tvl1" algorithm.

```
bash extract_frames.sh
```

7.47.4 Step 3. Generate File List

you can run the follow script to generate file list in the format of rawframes or videos.

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

7.47.5 Step 4. Check Directory Structure

After the whole data process for Multi-Moments in Time preparation, you will get the rawframes (RGB + Flow), videos and annotation files for Multi-Moments in Time.

In the context of the whole project (for Multi-Moments in Time only), the folder structure will look like:

```
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
        │   │   ├── ...
        │   │   ├── 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
            └── ...
```

For training and evaluating on Multi-Moments in Time, please refer to *getting_started.md*.

7.48 OmniSource

7.48.1 Introduction

```
@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}
}
```

We release a subset of the OmniSource web dataset used in the paper Omni-sourced Webly-supervised Learning for Video Recognition. Since all web dataset in OmniSource are built based on the Kinetics-400 taxonomy, we select those web data related to the 200 classes in Mini-Kinetics subset (which is proposed in Rethinking Spatiotemporal Feature Learning: Speed-Accuracy Trade-offs in Video Classification).

We provide data from all sources that are related to the 200 classes in Mini-Kinetics (including Kinetics trimmed clips, Kinetics untrimmed videos, images from Google and Instagram, video clips from Instagram). To obtain this dataset, please first fill in the request form. We will share the download link to you after your request is received. Since we release all data crawled from the web without any filtering, the dataset is large and it may take some time to download them. We describe the size of the datasets in the following table:

The file structure of our uploaded OmniSource dataset looks like:

```
OmniSource/
├── annotations
│   ├── googleimage_200
│   │   ├── googleimage_200.txt                        File list of all valid images␣
↪crawled from Google.
│   │   ├── tsn_8seg_googleimage_200_duplicate.txt     Positive file list of images␣
↪crawled from Google, which is similar to a validation example.
│   │   ├── tsn_8seg_googleimage_200.txt               Positive file list of images␣
↪crawled from Google, filtered by the teacher model.
│   │   └── tsn_8seg_googleimage_200_wodup.txt         Positive file list of images␣
↪crawled from Google, filtered by the teacher model, after de-duplication.
│   ├── 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                               The list of action names of the␣
↪200 classes in MiniKinetics.
│   ├── K400_to_MiniKinetics_classidx_mapping.json     The index mapping from Kinetics-
↪400 to MiniKinetics.
│   ├── kinetics_200
│   │   ├── k200_train.txt
│   │   └── k200_val.txt
│   ├── kinetics_raw_200
```

(continues on next page)

(continued from previous page)

```
│   │   └── 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                                   (10 volumes)
│   ├── vol_0.tar
│   ├── ...
│   └── vol_9.tar
├── insimage_200                                      (10 volumes)
│   ├── vol_0.tar
│   ├── ...
│   └── vol_9.tar
├── insvideo_200                                      (20 volumes)
│   ├── vol_00.tar
│   ├── ...
│   └── vol_19.tar
├── kinetics_200_train
│   └── kinetics_200_train.tar
├── kinetics_200_val
│   └── kinetics_200_val.tar
└── kinetics_raw_200_train                            (16 volumes)
    ├── vol_0.tar
    ├── ...
    └── vol_15.tar
```

7.48.2 Data Preparation

For data preparation, you need to first download those data. For `kinetics_200` and 3 web datasets: `googleimage_200`, `insimage_200` and `insvideo_200`, you just need to extract each volume and merge their contents.

For Kinetics raw videos, since loading long videos is very heavy, you need to first trim it into clips. Here we provide a script named `trim_raw_video.py`. It trims a long video into 10-second clips and remove the original raw video. You can use it to trim the Kinetics raw video.

The data should be placed in `data/OmniSource/`. When data preparation finished, the folder structure of `data/OmniSource` looks like (We omit the files not needed in training & testing for simplicity):

```
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
```

(continues on next page)

(continued from previous page)

```
│   │   └── 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
│   │   ├── ...
│   │   ├── 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 Skeleton Dataset

```
@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 Introduction

We release the skeleton annotations used in Revisiting Skeleton-based Action Recognition. By default, we use Faster-RCNN with ResNet50 backbone for human detection and HRNet-w32 for single person pose estimation. For FineGYM, we use Ground-Truth bounding boxes for the athlete instead of detection bounding boxes. Currently, we release the skeleton annotations for FineGYM and NTURGB-D Xsub split. Other annotations will be soo released.

7.49.2 Prepare Annotations

Currently, we support HMDB51, UCF101, FineGYM and NTURGB+D. For FineGYM, you can execute following scripts to prepare the annotations.

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

Due to Conditions of Use of the NTURGB+D dataset, we can not directly release the annotations used in our experiments. So that we provide a script to generate pose annotations for videos in NTURGB+D datasets, which generate a dictionary and save it as a single pickle file. You can create a list which contain all annotation dictionaries of corresponding videos and save them as a pickle file. Then you can get the `ntu60_xsub_train.pkl`, `ntu60_xsub_val.pkl`, `ntu120_xsub_train.pkl`, `ntu120_xsub_val.pkl` that we used in training.

For those who have not enough computations for pose extraction, we provide the outputs of the above pipeline here, corresponding to 4 different splits of NTURGB+D datasets:

- 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

To generate 2D pose annotations for a single video, first, you need to install mmdetection and mmpose from src code. After that, you need to replace the placeholder `mmdet_root` and `mmpose_root` in `ntu_pose_extraction.py` with your installation path. Then you can use following scripts for NTURGB+D video pose extraction:

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

After you get pose annotations for all videos in a dataset split, like `ntu60_xsub_val`. You can gather them into a single list and save the list as `ntu60_xsub_val.pkl`. You can use those larger pickle files for training and testing.

7.49.3 The Format of PoseC3D Annotations

Here we briefly introduce the format of PoseC3D Annotations, we will take `gym_train.pkl` as an example: the content of `gym_train.pkl` is a list of length 20484, each item is a dictionary that is the skeleton annotation of one video. Each dictionary has following fields:

- keypoint: The keypoint coordinates, which is a numpy array of the shape N (##person) x T (temporal length) x K (#keypoints, 17 in our case) x 2 (x, y coordinate).
- keypoint_score: The keypoint confidence scores, which is a numpy array of the shape N (##person) x T (temporal length) x K (#keypoints, 17 in our case).
- frame_dir: The corresponding video name.
- label: The action category.
- img_shape: The image shape of each frame.
- original_shape: Same as above.
- total_frames: The temporal length of the video.

For training with your custom dataset, you can refer to Custom Dataset Training.

7.49.4 Visualization

For skeleton data visualization, you need also to prepare the RGB videos. Please refer to visualize_heatmap_volume for detailed process. Here we provide some visualization examples from NTU-60 and FineGYM.

7.49.5 Convert the NTU RGB+D raw skeleton data to our format (only applicable to GCN backbones)

Here we also provide the script for converting the NTU RGB+D raw skeleton data to our format. First, download the raw skeleton data of NTU-RGBD 60 and NTU-RGBD 120 from https://github.com/shahroudy/NTURGB-D.

For NTU-RGBD 60, preprocess data and convert the data format with

```
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
```

For NTU-RGBD 120, preprocess data and convert the data format with

```
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 Convert annotations from third-party projects

We provide scripts to convert skeleton annotations from third-party projects to MMAction2 formats:

- BABEL: `babel2mma2.py`

TODO:

- [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 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset paper. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/sthv1/`.

7.50.2 Step 1. Prepare Annotations

Since the official website of Something-Something V1 is currently unavailable, you can download the annotations from third-part source to `$MMACTION2/data/sthv1/` .

7.50.3 Step 2. Prepare RGB Frames

Since the official dataset doesn't provide the original video data and only extracted RGB frames are available, you have to directly download RGB frames.

You can download all compressed file parts from third-part source to `$MMACTION2/data/sthv1/` and use the following command to uncompress.

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

For users who only want to use RGB frames, you can skip to step 5 to generate file lists in the format of rawframes. Since the prefix of official JPGs is "%05d.jpg" (e.g., "00001.jpg"), users need to add `"filename_tmpl='{:05}.jpg'"` to the dict of `data.train`, `data.val` and `data.test` in the config files related with sthv1 like this:

```
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',
        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 Step 3. Extract Flow

This part is **optional** if you only want to use RGB frames.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/sthv1_extracted/
ln -s /mnt/SSD/sthv1_extracted/ ../../../data/sthv1/rawframes
```

Then, you can run the following script to extract optical flow based on RGB frames.

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

7.50.5 Step 4. Encode Videos

This part is **optional** if you only want to use RGB frames.

You can run the following script to encode videos.

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

7.50.6 Step 5. Generate File List

You can run the follow script to generate file list in the format of rawframes and videos.

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

7.50.7 Step 6. Check Directory Structure

After the whole data process for Something-Something V1 preparation, you will get the rawframes (RGB + Flow), and annotation files for Something-Something V1.

In the context of the whole project (for Something-Something V1 only), the folder structure will look like:

```
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
│   │   │   │   ├── ...
```

(continues on next page)

(continued from previous page)

```
│   |   |   ├── 2
│   |   |   ├── ...
```

For training and evaluating on Something-Something V1, please refer to *getting_started.md*.

7.51 Something-Something V2

7.51.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/sthv2/`.

7.51.2 Step 1. Prepare Annotations

First of all, you have to sign in and download annotations to `$MMACTION2/data/sthv2/annotations` on the official website.

```
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 Step 2. Prepare Videos

Then, you can download all data parts to `$MMACTION2/data/sthv2/` and use the following command to uncompress.

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

7.51.4 Step 3. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/sthv2_extracted/
ln -s /mnt/SSD/sthv2_extracted/ ../../../data/sthv2/rawframes
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

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

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

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

If both are required, run the following script to extract frames.

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

7.51.5 Step 4. Generate File List

you can run the follow script to generate file list in the format of rawframes and videos.

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

7.51.6 Step 5. Check Directory Structure

After the whole data process for Something-Something V2 preparation, you will get the rawframes (RGB + Flow), videos and annotation files for Something-Something V2.

In the context of the whole project (for Something-Something V2 only), the folder structure will look like:

```
mmaction2
├── mmaction
├── tools
├── configs
├── data
│   ├── sthv2
│   │   ├── sthv2_{train,val}_list_rawframes.txt
│   │   ├── sthv2_{train,val}_list_videos.txt
│   │   ├── annotations
```

(continues on next page)

(continued from previous page)

```
│   │   ├── 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
│   │   │   ├── ...
```

For training and evaluating on Something-Something V2, please refer to *getting_started.md*.

7.52 THUMOS'14

7.52.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/thumos14/`.

7.52.2 Step 1. Prepare Annotations

First of all, run the following script to prepare annotations.

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

7.52.3 Step 2. Prepare Videos

Then, you can run the following script to prepare videos.

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

7.52.4 Step 3. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/thumos14_extracted/
ln -s /mnt/SSD/thumos14_extracted/ ../data/thumos14/rawframes/
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

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

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

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

If both are required, run the following script to extract frames.

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

7.52.5 Step 4. Fetch File List

This part is **optional** if you do not use SSN model.

You can run the follow script to fetch pre-computed tag proposals.

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

7.52.6 Step 5. Denormalize Proposal File

This part is **optional** if you do not use SSN model.

You can run the follow script to denormalize pre-computed tag proposals according to actual number of local rawframes.

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

7.52.7 Step 6. Check Directory Structure

After the whole data process for THUMOS'14 preparation, you will get the rawframes (RGB + Flow), videos and annotation files for THUMOS'14.

In the context of the whole project (for THUMOS'14 only), the folder structure will look like:

```
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
```

For training and evaluating on THUMOS'14, please refer to *getting_started.md*.

7.53 UCF-101

7.53.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/ucf101/`.

7.53.2 Step 1. Prepare Annotations

First of all, you can run the following script to prepare annotations.

```
bash download_annotations.sh
```

7.53.3 Step 2. Prepare Videos

Then, you can run the following script to prepare videos.

```
bash download_videos.sh
```

For better decoding speed, you can resize the original videos into smaller sized, densely encoded version by:

```
python ../resize_videos.py ../../../data/ucf101/videos/ ../../../data/ucf101/videos_256p_
↪dense_cache --dense --level 2 --ext avi
```

7.53.4 Step 3. Extract RGB and Flow

This part is **optional** if you only want to use the video loader.

Before extracting, please refer to *install.md* for installing denseflow.

If you have plenty of SSD space, then we recommend extracting frames there for better I/O performance. The extracted frames (RGB + Flow) will take up about 100GB.

You can run the following script to soft link SSD.

```
## execute these two line (Assume the SSD is mounted at "/mnt/SSD/")
mkdir /mnt/SSD/ucf101_extracted/
ln -s /mnt/SSD/ucf101_extracted/ ../../../data/ucf101/rawframes
```

If you only want to play with RGB frames (since extracting optical flow can be time-consuming), consider running the following script to extract **RGB-only** frames using denseflow.

```
bash extract_rgb_frames.sh
```

If you didn't install denseflow, you can still extract RGB frames using OpenCV by the following script, but it will keep the original size of the images.

```
bash extract_rgb_frames_opencv.sh
```

If Optical Flow is also required, run the following script to extract flow using "tvl1" algorithm.

```
bash extract_frames.sh
```

7.53.5 Step 4. Generate File List

you can run the follow script to generate file list in the format of rawframes and videos.

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

7.53.6 Step 5. Check Directory Structure

After the whole data process for UCF-101 preparation, you will get the rawframes (RGB + Flow), videos and annotation files for UCF-101.

In the context of the whole project (for UCF-101 only), the folder structure will look like:

```
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

│   │   │   ├── 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
```

(continues on next page)

(continued from previous page)

```
│   │   │   ├── ...
│   │   │   ├── YoYo
│   │   │   │   ├── v_YoYo_g01_c01
│   │   │   │   ├── ...
│   │   │   │   ├── v_YoYo_g25_c05
```

For training and evaluating on UCF-101, please refer to *getting_started.md*.

7.54 UCF101-24

7.54.1 Introduction

```
@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}
}
```

For basic dataset information, you can refer to the dataset website. Before we start, please make sure that the directory is located at `$MMACTION2/tools/data/ucf101_24/`.

7.54.2 Download and Extract

You can download the RGB frames, optical flow and ground truth annotations from google drive. The data are provided from MOC, which is adapted from act-detector and corrected-UCF101-Annots.

Note: The annotation of this UCF101-24 is from here, which is more correct.

After downloading the `UCF101_v2.tar.gz` file and put it in `$MMACTION2/tools/data/ucf101_24/`, you can run the following command to uncompress.

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

7.54.3 Check Directory Structure

After uncompressing, you will get the `rgb-images` directory, `brox-images` directory and `UCF101v2-GT.pkl` for UCF101-24.

In the context of the whole project (for UCF101-24 only), the folder structure will look like:

```
mmaction2
├── mmaction
├── tools
├── configs
```

(continues on next page)

(continued from previous page)

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

Note: The `UCF101v2-GT.pkl` exists as a cache, it contains 6 items as follows:

1. `labels` (list): List of the 24 labels.
2. `gttubes` (dict): Dictionary that contains the ground truth tubes for each video. A **gttube** is dictionary that associates with each index of label and a list of tubes. A **tube** is a numpy array with `nframes` rows and 5 columns, each col is in format like `<frame index> <x1> <y1> <x2> <y2>`.
3. `nframes` (dict): Dictionary that contains the number of frames for each video, like `'HorseRiding/v_HorseRiding_g05_c02': 151`.
4. `train_videos` (list): A list with `nsplits=1` elements, each one containing the list of training videos.
5. `test_videos` (list): A list with `nsplits=1` elements, each one containing the list of testing videos.
6. `resolution` (dict): Dictionary that outputs a tuple (h,w) of the resolution for each video, like `'FloorGymnastics/v_FloorGymnastics_g09_c03': (240, 320)`.

CHAPTER

EIGHT

OVERVIEW

- Number of checkpoints: 220
- Number of configs: 199
- Number of papers: 26
  - ALGORITHM: 22
  - BACKBONE: 1
  - DATASET: 2
  - OTHERS: 1

For supported datasets, see *datasets overview*.

8.1 Spatio Temporal Action Detection Models

- Number of checkpoints: 22
- Number of configs: 22
- Number of papers: 3
  - [ALGORITHM] Ava: A Video Dataset of Spatio-Temporally Localized Atomic Visual Actions (-> -> ->)
  - [ALGORITHM] Slowfast Networks for Video Recognition (->)
  - [DATASET] Ava: A Video Dataset of Spatio-Temporally Localized Atomic Visual Actions (-> -> ->)

8.2 Action Localization Models

- Number of checkpoints: 7
- Number of configs: 3
- Number of papers: 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 Action Recognition Models

- Number of checkpoints: 175
- Number of configs: 158
- Number of papers: 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 Skeleton-based Action Recognition Models

- Number of checkpoints: 16
- Number of configs: 16
- Number of papers: 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

NINE

ACTION RECOGNITION MODELS

9.1 C3D

Learning Spatiotemporal Features with 3D Convolutional Networks

9.1.1 Abstract

We propose a simple, yet effective approach for spatiotemporal feature learning using deep 3-dimensional convolutional networks (3D ConvNets) trained on a large scale supervised video dataset. Our findings are three-fold: 1) 3D ConvNets are more suitable for spatiotemporal feature learning compared to 2D ConvNets; 2) A homogeneous architecture with small 3x3x3 convolution kernels in all layers is among the best performing architectures for 3D ConvNets; and 3) Our learned features, namely C3D (Convolutional 3D), with a simple linear classifier outperform state-of-the-art methods on 4 different benchmarks and are comparable with current best methods on the other 2 benchmarks. In addition, the features are compact: achieving 52.8% accuracy on UCF101 dataset with only 10 dimensions and also very efficient to compute due to the fast inference of ConvNets. Finally, they are conceptually very simple and easy to train and use.

9.1.2 Results and Models

UCF-101

Note:

1. The author of C3D normalized UCF-101 with volume mean and used SVM to classify videos, while we normalized the dataset with RGB mean value and used a linear classifier.
2. The **gpus** indicates the number of gpu (32G V100) we used to get the checkpoint. It is noteworthy that the configs we provide are used for 8 gpus as default. According to the Linear Scaling Rule, you may set the learning rate proportional to the batch size if you use different GPUs or videos per GPU, e.g., lr=0.01 for 4 GPUs x 2 video/gpu and lr=0.08 for 16 GPUs x 4 video/gpu.
3. The **inference_time** is got by this benchmark script, where we use the sampling frames strategy of the test setting and only care about the model inference time, not including the IO time and pre-processing time. For each setting, we use 1 gpu and set batch size (videos per gpu) to 1 to calculate the inference time.

For more details on data preparation, you can refer to UCF-101 in *Data Preparation*.

9.1.3 Train

You can use the following command to train a model.

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

Example: train C3D model on UCF-101 dataset in a deterministic option with periodic validation.

```
python tools/train.py configs/recognition/c3d/c3d_sports1m_16x1x1_45e_ucf101_rgb.py \
    --validate --seed 0 --deterministic
```

For more details, you can refer to **Training setting** part in getting_started.

9.1.4 Test

You can use the following command to test a model.

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

Example: test C3D model on UCF-101 dataset and dump the result to a json file.

```
python tools/test.py configs/recognition/c3d/c3d_sports1m_16x1x1_45e_ucf101_rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --eval top_k_accuracy
```

For more details, you can refer to **Test a dataset** part in getting_started.

9.1.5 Citation

```
@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}
}
```

9.2 CSN

Video Classification With Channel-Separated Convolutional Networks

9.2.1 Abstract

Group convolution has been shown to offer great computational savings in various 2D convolutional architectures for image classification. It is natural to ask: 1) if group convolution can help to alleviate the high computational cost of video classification networks; 2) what factors matter the most in 3D group convolutional networks; and 3) what are good computation/accuracy trade-offs with 3D group convolutional networks. This paper studies the effects of different design choices in 3D group convolutional networks for video classification. We empirically demonstrate that the amount of channel interactions plays an important role in the accuracy of 3D group convolutional networks. Our experiments suggest two main findings. First, it is a good practice to factorize 3D convolutions by separating channel interactions and spatiotemporal interactions as this leads to improved accuracy and lower computational cost. Second, 3D channel-separated convolutions provide a form of regularization, yielding lower training accuracy but higher test accuracy compared to 3D convolutions. These two empirical findings lead us to design an architecture – Channel-Separated Convolutional Network (CSN) – which is simple, efficient, yet accurate. On Sports1M, Kinetics, and Something-Something, our CSNs are comparable with or better than the state-of-the-art while being 2-3 times more efficient.

9.2.2 Results and Models

Kinetics-400

Note:

1. The **gpus** indicates the number of gpu (32G V100) we used to get the checkpoint. It is noteworthy that the configs we provide are used for 8 gpus as default. According to the Linear Scaling Rule, you may set the learning rate proportional to the batch size if you use different GPUs or videos per GPU, e.g., lr=0.01 for 4 GPUs x 2 video/gpu and lr=0.08 for 16 GPUs x 4 video/gpu.
2. The **inference_time** is got by this benchmark script, where we use the sampling frames strategy of the test setting and only care about the model inference time, not including the IO time and pre-processing time. For each setting, we use 1 gpu and set batch size (videos per gpu) to 1 to calculate the inference time.
3. The validation set of Kinetics400 we used consists of 19796 videos. These videos are available at Kinetics400-Validation. The corresponding data list (each line is of the format 'video_id, num_frames, label_index') and the label map are also available.
4. The **infer_ckpt** means those checkpoints are ported from VMZ.

For more details on data preparation, you can refer to Kinetics400 in *Data Preparation*.

9.2.3 Train

You can use the following command to train a model.

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

Example: train CSN model on Kinetics-400 dataset in a deterministic option with periodic validation.

```
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
```

For more details, you can refer to **Training setting** part in getting_started.

9.2.4 Test

You can use the following command to test a model.

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

Example: test CSN model on Kinetics-400 dataset and dump the result to a json file.

```
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
```

For more details, you can refer to **Test a dataset** part in getting_started.

9.2.5 Citation

```
@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}
}
```

9.3 I3D

Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset

Non-local Neural Networks

9.3.1 Abstract

The paucity of videos in current action classification datasets (UCF-101 and HMDB-51) has made it difficult to identify good video architectures, as most methods obtain similar performance on existing small-scale benchmarks. This paper re-evaluates state-of-the-art architectures in light of the new Kinetics Human Action Video dataset. Kinetics has two orders of magnitude more data, with 400 human action classes and over 400 clips per class, and is collected from realistic, challenging YouTube videos. We provide an analysis on how current architectures fare on the task of action classification on this dataset and how much performance improves on the smaller benchmark datasets after pre-training on Kinetics. We also introduce a new Two-Stream Inflated 3D ConvNet (I3D) that is based on 2D ConvNet inflation:

filters and pooling kernels of very deep image classification ConvNets are expanded into 3D, making it possible to learn seamless spatio-temporal feature extractors from video while leveraging successful ImageNet architecture designs and even their parameters. We show that, after pre-training on Kinetics, I3D models considerably improve upon the state-of-the-art in action classification, reaching 80.9% on HMDB-51 and 98.0% on UCF-101.

9.3.2 Results and Models

Kinetics-400

Note:

1. The **gpus** indicates the number of gpu we used to get the checkpoint. It is noteworthy that the configs we provide are used for 8 gpus as default. According to the Linear Scaling Rule, you may set the learning rate proportional to the batch size if you use different GPUs or videos per GPU, e.g., lr=0.01 for 4 GPUs x 2 video/gpu and lr=0.08 for 16 GPUs x 4 video/gpu.
2. The **inference_time** is got by this benchmark script, where we use the sampling frames strategy of the test setting and only care about the model inference time, not including the IO time and pre-processing time. For each setting, we use 1 gpu and set batch size (videos per gpu) to 1 to calculate the inference time.
3. The validation set of Kinetics400 we used consists of 19796 videos. These videos are available at Kinetics400-Validation. The corresponding data list (each line is of the format 'video_id, num_frames, label_index') and the label map are also available.

For more details on data preparation, you can refer to Kinetics400 in *Data Preparation*.

9.3.3 Train

You can use the following command to train a model.

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

Example: train I3D model on Kinetics-400 dataset in a deterministic option with periodic validation.

```
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
```

For more details, you can refer to **Training setting** part in getting_started.

9.3.4 Test

You can use the following command to test a model.

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

Example: test I3D model on Kinetics-400 dataset and dump the result to a json file.

```
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
```

For more details, you can refer to **Test a dataset** part in getting_started.

9.3.5 Citation

```
@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}
}
```

9.4 Omni-sourced Webly-supervised Learning for Video Recognition

Omni-sourced Webly-supervised Learning for Video Recognition

Dataset

9.4.1 Abstract

We introduce OmniSource, a novel framework for leveraging web data to train video recognition models. OmniSource overcomes the barriers between data formats, such as images, short videos, and long untrimmed videos for webly-supervised learning. First, data samples with multiple formats, curated by task-specific data collection and automatically filtered by a teacher model, are transformed into a unified form. Then a joint-training strategy is proposed to deal with the domain gaps between multiple data sources and formats in webly-supervised learning. Several good practices, including data balancing, resampling, and cross-dataset mixup are adopted in joint training. Experiments show that by utilizing data from multiple sources and formats, OmniSource is more data-efficient in training. With only 3.5M images and 800K minutes videos crawled from the internet without human labeling (less than 2% of prior works), our models learned with OmniSource improve Top-1 accuracy of 2D- and 3D-ConvNet baseline models by 3.0% and 3.9%, respectively, on the Kinetics-400 benchmark. With OmniSource, we establish new records with different pretraining strategies for video recognition. Our best models achieve 80.4%, 80.5%, and 83.6 Top-1 accuracies on the Kinetics-400 benchmark respectively for training-from-scratch, ImageNet pre-training and IG-65M pre-training.

9.4.2 Results and Models

Kinetics-400 Model Release

We currently released 4 models trained with OmniSource framework, including both 2D and 3D architectures. We compare the performance of models trained with or without OmniSource in the following table.

1. The validation set of Kinetics400 we used consists of 19796 videos. These videos are available at Kinetics400-Validation. The corresponding data list (each line is of the format 'video_id, num_frames, label_index') and the label map are also available.

9.4.3 Benchmark on Mini-Kinetics

We release a subset of web dataset used in the OmniSource paper. Specifically, we release the web data in the 200 classes of Mini-Kinetics. The statistics of those datasets is detailed in preparing_omnisource. To obtain those data, you need to fill in a data request form. After we received your request, the download link of these data will be send to you. For more details on the released OmniSource web dataset, please refer to preparing_omnisource.

We benchmark the OmniSource framework on the released subset, results are listed in the following table (we report the Top-1 and Top-5 accuracy on Mini-Kinetics validation). The benchmark can be used as a baseline for video recognition with web data.

TSN-8seg-ResNet50

SlowOnly-8x8-ResNet50

We also list the benchmark in the original paper which run on Kinetics-400 for comparison:

9.4.4 Citation

```
@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

A closer look at spatiotemporal convolutions for action recognition

9.5.1 Abstract

In this paper we discuss several forms of spatiotemporal convolutions for video analysis and study their effects on action recognition. Our motivation stems from the observation that 2D CNNs applied to individual frames of the video have remained solid performers in action recognition. In this work we empirically demonstrate the accuracy advantages of 3D CNNs over 2D CNNs within the framework of residual learning. Furthermore, we show that factorizing the 3D convolutional filters into separate spatial and temporal components yields significantly advantages in accuracy. Our empirical study leads to the design of a new spatiotemporal convolutional block "R(2+1)D" which gives rise to CNNs that achieve results comparable or superior to the state-of-the-art on Sports-1M, Kinetics, UCF101 and HMDB51.

9.5.2 Results and Models

Kinetics-400

Note:

1. The **gpus** indicates the number of gpu we used to get the checkpoint. It is noteworthy that the configs we provide are used for 8 gpus as default. According to the Linear Scaling Rule, you may set the learning rate proportional to the batch size if you use different GPUs or videos per GPU, e.g., lr=0.01 for 4 GPUs x 2 video/gpu and lr=0.08 for 16 GPUs x 4 video/gpu.
2. The **inference_time** is got by this benchmark script, where we use the sampling frames strategy of the test setting and only care about the model inference time, not including the IO time and pre-processing time. For each setting, we use 1 gpu and set batch size (videos per gpu) to 1 to calculate the inference time.
3. The validation set of Kinetics400 we used consists of 19796 videos. These videos are available at Kinetics400-Validation. The corresponding data list (each line is of the format 'video_id, num_frames, label_index') and the label map are also available.

For more details on data preparation, you can refer to Kinetics400 in *Data Preparation*.

9.5.3 Train

You can use the following command to train a model.

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

Example: train R(2+1)D model on Kinetics-400 dataset in a deterministic option with periodic validation.

```
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
```

For more details, you can refer to **Training setting** part in getting_started.

9.5.4 Test

You can use the following command to test a model.

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

Example: test R(2+1)D model on Kinetics-400 dataset and dump the result to a json file.

```
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
```

For more details, you can refer to **Test a dataset** part in getting_started.

9.5.5 Citation

```
@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}
}
```

9.6 SlowFast

SlowFast Networks for Video Recognition

9.6.1 Abstract

We present SlowFast networks for video recognition. Our model involves (i) a Slow pathway, operating at low frame rate, to capture spatial semantics, and (ii) a Fast pathway, operating at high frame rate, to capture motion at fine temporal resolution. The Fast pathway can be made very lightweight by reducing its channel capacity, yet can learn useful temporal information for video recognition. Our models achieve strong performance for both action classification and detection in video, and large improvements are pin-pointed as contributions by our SlowFast concept. We report state-of-the-art accuracy on major video recognition benchmarks, Kinetics, Charades and AVA.

9.6.2 Results and Models

Kinetics-400

Something-Something V1

Note:

1. The **gpus** indicates the number of gpu we used to get the checkpoint. It is noteworthy that the configs we provide are used for 8 gpus as default. According to the Linear Scaling Rule, you may set the learning rate proportional to the batch size if you use different GPUs or videos per GPU, e.g., lr=0.01 for 4 GPUs x 2 video/gpu and lr=0.08 for 16 GPUs x 4 video/gpu.
2. The **inference_time** is got by this benchmark script, where we use the sampling frames strategy of the test setting and only care about the model inference time, not including the IO time and pre-processing time. For each setting, we use 1 gpu and set batch size (videos per gpu) to 1 to calculate the inference time.
3. The validation set of Kinetics400 we used consists of 19796 videos. These videos are available at Kinetics400-Validation. The corresponding data list (each line is of the format 'video_id, num_frames, label_index') and the label map are also available.

For more details on data preparation, you can refer to Kinetics400 in *Data Preparation*.

9.6.3 Train

You can use the following command to train a model.

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

Example: train SlowFast model on Kinetics-400 dataset in a deterministic option with periodic validation.

```
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
```

For more details, you can refer to **Training setting** part in getting_started.

9.6.4 Test

You can use the following command to test a model.

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

Example: test SlowFast model on Kinetics-400 dataset and dump the result to a json file.

```
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
```

For more details, you can refer to **Test a dataset** part in getting_started.

9.6.5 Citation

```
@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 SlowOnly

Slowfast networks for video recognition

9.7.1 Abstract

We present SlowFast networks for video recognition. Our model involves (i) a Slow pathway, operating at low frame rate, to capture spatial semantics, and (ii) a Fast pathway, operating at high frame rate, to capture motion at fine temporal resolution. The Fast pathway can be made very lightweight by reducing its channel capacity, yet can learn useful temporal information for video recognition. Our models achieve strong performance for both action classification and detection in video, and large improvements are pin-pointed as contributions by our SlowFast concept. We report state-of-the-art accuracy on major video recognition benchmarks, Kinetics, Charades and AVA.

9.7.2 Results and Models

Kinetics-400

Kinetics-400 Data Benchmark

In data benchmark, we compare two different data preprocessing methods: (1) Resize video to 340x256, (2) Resize the short edge of video to 320px, (3) Resize the short edge of video to 256px.

Kinetics-400 OmniSource Experiments

Kinetics-600

Kinetics-700

GYM99

Jester

HMDB51

UCF101

Something-Something V1

Note:

1. The **gpus** indicates the number of gpu we used to get the checkpoint. It is noteworthy that the configs we provide are used for 8 gpus as default. According to the Linear Scaling Rule, you may set the learning rate proportional to the batch size if you use different GPUs or videos per GPU, e.g., lr=0.01 for 4 GPUs x 2 video/gpu and lr=0.08 for 16 GPUs x 4 video/gpu.
2. The **inference_time** is got by this benchmark script, where we use the sampling frames strategy of the test setting and only care about the model inference time, not including the IO time and pre-processing time. For each setting, we use 1 gpu and set batch size (videos per gpu) to 1 to calculate the inference time.
3. The validation set of Kinetics400 we used consists of 19796 videos. These videos are available at Kinetics400-Validation. The corresponding data list (each line is of the format 'video_id, num_frames, label_index') and the label map are also available.

For more details on data preparation, you can refer to corresponding parts in *Data Preparation*.

9.7.3 Train

You can use the following command to train a model.

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

Example: train SlowOnly model on Kinetics-400 dataset in a deterministic option with periodic validation.

```
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
```

For more details, you can refer to **Training setting** part in getting_started.

9.7.4 Test

You can use the following command to test a model.

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

Example: test SlowOnly model on Kinetics-400 dataset and dump the result to a json file.

```
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
```

For more details, you can refer to **Test a dataset** part in getting_started.

9.7.5 Citation

```
@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}
}
```

9.8 TANet

TAM: Temporal Adaptive Module for Video Recognition

9.8.1 Abstract

Video data is with complex temporal dynamics due to various factors such as camera motion, speed variation, and different activities. To effectively capture this diverse motion pattern, this paper presents a new temporal adaptive module ({\bf TAM}) to generate video-specific temporal kernels based on its own feature map. TAM proposes a unique two-level adaptive modeling scheme by decoupling the dynamic kernel into a location sensitive importance map and a location invariant aggregation weight. The importance map is learned in a local temporal window to capture short-term information, while the aggregation weight is generated from a global view with a focus on long-term structure. TAM is a modular block and could be integrated into 2D CNNs to yield a powerful video architecture (TANet) with a very small extra computational cost. The extensive experiments on Kinetics-400 and Something-Something datasets demonstrate that our TAM outperforms other temporal modeling methods consistently, and achieves the state-of-the-art performance under the similar complexity.

9.8.2 Results and Models

Kinetics-400

Something-Something V1

Note:

1. The **gpus** indicates the number of gpu we used to get the checkpoint. It is noteworthy that the configs we provide are used for 8 gpus as default. According to the Linear Scaling Rule, you may set the learning rate proportional to the batch size if you use different GPUs or videos per GPU, e.g., lr=0.01 for 8 GPUs x 8 videos/gpu and lr=0.04 for 16 GPUs x 16 videos/gpu.
2. The **inference_time** is got by this benchmark script, where we use the sampling frames strategy of the test setting and only care about the model inference time, not including the IO time and pre-processing time. For each setting, we use 1 gpu and set batch size (videos per gpu) to 1 to calculate the inference time.
3. The values in columns named after "reference" are the results got by testing on our dataset, using the checkpoints provided by the author with same model settings. The checkpoints for reference repo can be downloaded here.
4. The validation set of Kinetics400 we used consists of 19796 videos. These videos are available at Kinetics400-Validation. The corresponding data list (each line is of the format 'video_id, num_frames, label_index') and the label map are also available.

For more details on data preparation, you can refer to corresponding parts in *Data Preparation*.

9.8.3 Train

You can use the following command to train a model.

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

Example: train TANet model on Kinetics-400 dataset in a deterministic option with periodic validation.

```
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
```

For more details, you can refer to **Training setting** part in getting_started.

9.8.4 Test

You can use the following command to test a model.

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

Example: test TANet model on Kinetics-400 dataset and dump the result to a json file.

```
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
```

For more details, you can refer to **Test a dataset** part in getting_started.

9.8.5 Citation

```
@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}
}
```

9.9 TimeSformer

Is Space-Time Attention All You Need for Video Understanding?

9.9.1 Abstract

We present a convolution-free approach to video classification built exclusively on self-attention over space and time. Our method, named "TimeSformer," adapts the standard Transformer architecture to video by enabling spatiotemporal feature learning directly from a sequence of frame-level patches. Our experimental study compares different self-attention schemes and suggests that "divided attention," where temporal attention and spatial attention are separately applied within each block, leads to the best video classification accuracy among the design choices considered. Despite the radically new design, TimeSformer achieves state-of-the-art results on several action recognition benchmarks, including the best reported accuracy on Kinetics-400 and Kinetics-600. Finally, compared to 3D convolutional networks, our model is faster to train, it can achieve dramatically higher test efficiency (at a small drop in accuracy), and it can also be applied to much longer video clips (over one minute long).

9.9.2 Results and Models

Kinetics-400

Note:

1. The **gpus** indicates the number of gpu (32G V100) we used to get the checkpoint. It is noteworthy that the configs we provide are used for 8 gpus as default. According to the Linear Scaling Rule, you may set the learning rate proportional to the batch size if you use different GPUs or videos per GPU, e.g., lr=0.005 for 8 GPUs x 8 videos/gpu and lr=0.00375 for 8 GPUs x 6 videos/gpu.
2. We keep the test setting with the original repo (three crop x 1 clip).
3. The pretrained model `vit_base_patch16_224.pth` used by TimeSformer was converted from vision_transformer.

For more details on data preparation, you can refer to Kinetics400 in *Data Preparation*.

9.9.3 Train

You can use the following command to train a model.

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

Example: train TimeSformer model on Kinetics-400 dataset in a deterministic option with periodic validation.

```
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
```

For more details, you can refer to **Training setting** part in getting_started.

9.9.4 Test

You can use the following command to test a model.

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

Example: test TimeSformer model on Kinetics-400 dataset and dump the result to a json file.

```
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
```

For more details, you can refer to **Test a dataset** part in getting_started.

9.9.5 Citation

```
@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}
}
```

9.10 TIN

Temporal Interlacing Network

9.10.1 Abstract

For a long time, the vision community tries to learn the spatio-temporal representation by combining convolutional neural network together with various temporal models, such as the families of Markov chain, optical flow, RNN and temporal convolution. However, these pipelines consume enormous computing resources due to the alternately learning process for spatial and temporal information. One natural question is whether we can embed the temporal information into the spatial one so the information in the two domains can be jointly learned once-only. In this work, we answer this question by presenting a simple yet powerful operator – temporal interlacing network (TIN). Instead of learning the temporal features, TIN fuses the two kinds of information by interlacing spatial representations from the past to the future, and vice versa. A differentiable interlacing target can be learned to control the interlacing process. In this way, a heavy temporal model is replaced by a simple interlacing operator. We theoretically prove that with a learnable interlacing target, TIN performs equivalently to the regularized temporal convolution network (r-TCN), but gains 4% more accuracy with 6x less latency on 6 challenging benchmarks. These results push the state-of-the-art performances of video understanding by a considerable margin. Not surprising, the ensemble model of the proposed TIN won the 1st place in the ICCV19 - Multi Moments in Time challenge.

9.10.2 Results and Models

Something-Something V1

Something-Something V2

Kinetics-400

Here, we use `finetune` to indicate that we use TSM model trained on Kinetics-400 to finetune the TIN model on Kinetics-400.

Note:

1. The **reference topk acc** are got by training the original repo ##1aacd0c with no AverageMeter issue. The AverageMeter issue will lead to incorrect performance, so we fix it before running.
2. The **gpus** indicates the number of gpu we used to get the checkpoint. It is noteworthy that the configs we provide are used for 8 gpus as default. According to the Linear Scaling Rule, you may set the learning rate proportional to the batch size if you use different GPUs or videos per GPU, e.g., lr=0.01 for 4 GPUs x 2 video/gpu and lr=0.08 for 16 GPUs x 4 video/gpu.
3. The **inference_time** is got by this benchmark script, where we use the sampling frames strategy of the test setting and only care about the model inference time, not including the IO time and pre-processing time. For each setting, we use 1 gpu and set batch size (videos per gpu) to 1 to calculate the inference time.
4. The values in columns named after "reference" are the results got by training on the original repo, using the same model settings.
5. The validation set of Kinetics400 we used consists of 19796 videos. These videos are available at Kinetics400-Validation. The corresponding data list (each line is of the format 'video_id, num_frames, label_index') and the label map are also available.

For more details on data preparation, you can refer to Kinetics400, Something-Something V1 and Something-Something V2 in *Data Preparation*.

9.10.3 Train

You can use the following command to train a model.

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

Example: train TIN model on Something-Something V1 dataset in a deterministic option with periodic validation.

```
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
```

For more details, you can refer to **Training setting** part in getting_started.

9.10.4 Test

You can use the following command to test a model.

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

Example: test TIN model on Something-Something V1 dataset and dump the result to a json file.

```
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
```

For more details, you can refer to **Test a dataset** part in getting_started.

9.10.5 Citation

```
@article{shao2020temporal,
    title={Temporal Interlacing Network},
    author={Hao Shao and Shengju Qian and Yu Liu},
    year={2020},
    journal={AAAI},
}
```

9.11 TPN

Temporal Pyramid Network for Action Recognition

9.11.1 Abstract

Visual tempo characterizes the dynamics and the temporal scale of an action. Modeling such visual tempos of different actions facilitates their recognition. Previous works often capture the visual tempo through sampling raw videos at multiple rates and constructing an input-level frame pyramid, which usually requires a costly multi-branch network to handle. In this work we propose a generic Temporal Pyramid Network (TPN) at the feature-level, which can be flexibly integrated into 2D or 3D backbone networks in a plug-and-play manner. Two essential components of TPN, the source of features and the fusion of features, form a feature hierarchy for the backbone so that it can capture action instances at various tempos. TPN also shows consistent improvements over other challenging baselines on several action recognition datasets. Specifically, when equipped with TPN, the 3D ResNet-50 with dense sampling obtains a 2% gain on the validation set of Kinetics-400. A further analysis also reveals that TPN gains most of its improvements on action classes that have large variances in their visual tempos, validating the effectiveness of TPN.

9.11.2 Results and Models

Kinetics-400

Something-Something V1

Note:

1. The **gpus** indicates the number of gpu we used to get the checkpoint. It is noteworthy that the configs we provide are used for 8 gpus as default. According to the Linear Scaling Rule, you may set the learning rate proportional to the batch size if you use different GPUs or videos per GPU, e.g., lr=0.01 for 4 GPUs x 2 video/gpu and lr=0.08 for 16 GPUs x 4 video/gpu.
2. The **inference_time** is got by this benchmark script, where we use the sampling frames strategy of the test setting and only care about the model inference time, not including the IO time and pre-processing time. For each setting, we use 1 gpu and set batch size (videos per gpu) to 1 to calculate the inference time.
3. The values in columns named after "reference" are the results got by testing the checkpoint released on the original repo and codes, using the same dataset with ours.
4. The validation set of Kinetics400 we used consists of 19796 videos. These videos are available at Kinetics400-Validation. The corresponding data list (each line is of the format 'video_id, num_frames, label_index') and the label map are also available.

For more details on data preparation, you can refer to Kinetics400, Something-Something V1 and Something-Something V2 in *Data Preparation*.

9.11.3 Train

You can use the following command to train a model.

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

Example: train TPN model on Kinetics-400 dataset in a deterministic option with periodic validation.

```
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]
```

For more details, you can refer to **Training setting** part in getting_started.

9.11.4 Test

You can use the following command to test a model.

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

Example: test TPN model on Kinetics-400 dataset and dump the result to a json file.

```
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
```

For more details, you can refer to **Test a dataset** part in getting_started.

9.11.5 Citation

```
@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},
}
```

9.12 TRN

Temporal Relational Reasoning in Videos

9.12.1 Abstract

Temporal relational reasoning, the ability to link meaningful transformations of objects or entities over time, is a fundamental property of intelligent species. In this paper, we introduce an effective and interpretable network module, the Temporal Relation Network (TRN), designed to learn and reason about temporal dependencies between video frames at multiple time scales. We evaluate TRN-equipped networks on activity recognition tasks using three recent video datasets - Something-Something, Jester, and Charades - which fundamentally depend on temporal relational reasoning. Our results demonstrate that the proposed TRN gives convolutional neural networks a remarkable capacity to discover temporal relations in videos. Through only sparsely sampled video frames, TRN-equipped networks can accurately predict human-object interactions in the Something-Something dataset and identify various human gestures on the Jester dataset with very competitive performance. TRN-equipped networks also outperform two-stream networks and 3D convolution networks in recognizing daily activities in the Charades dataset. Further analyses show that the models learn intuitive and interpretable visual common sense knowledge in videos.

9.12.2 Results and Models

Something-Something V1

Something-Something V2

Note:

1. The **gpus** indicates the number of gpu we used to get the checkpoint. It is noteworthy that the configs we provide are used for 8 gpus as default. According to the Linear Scaling Rule, you may set the learning rate proportional to the batch size if you use different GPUs or videos per GPU, e.g., lr=0.01 for 4 GPUs x 2 video/gpu and lr=0.08 for 16 GPUs x 4 video/gpu.

2. There are two kinds of test settings for Something-Something dataset, efficient setting (center crop x 1 clip) and accurate setting (Three crop x 2 clip).
3. In the original repository, the author augments data with random flipping on something-something dataset, but the augmentation method may be wrong due to the direct actions, such as `push left to right`. So, we replaced `flip` with `flip with label mapping`, and change the testing method `TenCrop`, which has five flipped crops, to `Twice Sample & ThreeCrop`.
4. We use `ResNet50` instead of `BNInception` as the backbone of TRN. When Training `TRN-ResNet50` on sthv1 dataset in the original repository, we get top1 (top5) accuracy 30.542 (58.627) vs. ours 31.62 (60.01).

For more details on data preparation, you can refer to

- preparing_sthv1
- preparing_sthv2

9.12.3 Train

You can use the following command to train a model.

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

Example: train TRN model on sthv1 dataset in a deterministic option with periodic validation.

```
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
```

For more details, you can refer to **Training setting** part in getting_started.

9.12.4 Test

You can use the following command to test a model.

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

Example: test TRN model on sthv1 dataset and dump the result to a json file.

```
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
```

For more details, you can refer to **Test a dataset** part in getting_started.

9.12.5 Citation

```
@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}
}
```

9.13 TSM

TSM: Temporal Shift Module for Efficient Video Understanding

9.13.1 Abstract

The explosive growth in video streaming gives rise to challenges on performing video understanding at high accuracy and low computation cost. Conventional 2D CNNs are computationally cheap but cannot capture temporal relationships; 3D CNN based methods can achieve good performance but are computationally intensive, making it expensive to deploy. In this paper, we propose a generic and effective Temporal Shift Module (TSM) that enjoys both high efficiency and high performance. Specifically, it can achieve the performance of 3D CNN but maintain 2D CNN's complexity. TSM shifts part of the channels along the temporal dimension; thus facilitate information exchanged among neighboring frames. It can be inserted into 2D CNNs to achieve temporal modeling at zero computation and zero parameters. We also extended TSM to online setting, which enables real-time low-latency online video recognition and video object detection. TSM is accurate and efficient: it ranks the first place on the Something-Something leaderboard upon publication; on Jetson Nano and Galaxy Note8, it achieves a low latency of 13ms and 35ms for online video recognition.

9.13.2 Results and Models

Kinetics-400

Diving48

Something-Something V1

Something-Something V2

MixUp & CutMix on Something-Something V1

Jester

HMDB51

UCF101

Note:

1. The **gpus** indicates the number of gpu we used to get the checkpoint. It is noteworthy that the configs we provide are used for 8 gpus as default. According to the Linear Scaling Rule, you may set the learning rate proportional

to the batch size if you use different GPUs or videos per GPU, e.g., lr=0.01 for 4 GPUs x 2 video/gpu and lr=0.08 for 16 GPUs x 4 video/gpu.

2. The **inference_time** is got by this benchmark script, where we use the sampling frames strategy of the test setting and only care about the model inference time, not including the IO time and pre-processing time. For each setting, we use 1 gpu and set batch size (videos per gpu) to 1 to calculate the inference time.
3. The values in columns named after "reference" are the results got by training on the original repo, using the same model settings. The checkpoints for reference repo can be downloaded here.
4. There are two kinds of test settings for Something-Something dataset, efficient setting (center crop x 1 clip) and accurate setting (Three crop x 2 clip), which is referred from the original repo. We use efficient setting as default provided in config files, and it can be changed to accurate setting by

```
...
test_pipeline = [
    dict(
        type='SampleFrames',
        clip_len=1,
        frame_interval=1,
        num_clips=16,   ## `num_clips = 8` when using 8 segments
        twice_sample=True,    ## set `twice_sample=True` for twice sample in accurate␣
↪setting
        test_mode=True),
    dict(type='RawFrameDecode'),
    dict(type='Resize', scale=(-1, 256)),
    ## dict(type='CenterCrop', crop_size=224), it is used for efficient setting
    dict(type='ThreeCrop', crop_size=256),  ## it is used for accurate setting
    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. When applying Mixup and CutMix, we use the hyper parameter `alpha=0.2`.
6. The validation set of Kinetics400 we used consists of 19796 videos. These videos are available at Kinetics400-Validation. The corresponding data list (each line is of the format 'video_id, num_frames, label_index') and the label map are also available.
7. The **infer_ckpt** means those checkpoints are ported from TSM.

For more details on data preparation, you can refer to corresponding parts in *Data Preparation*.

9.13.3 Train

You can use the following command to train a model.

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

Example: train TSM model on Kinetics-400 dataset in a deterministic option with periodic validation.

```
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
```

For more details, you can refer to **Training setting** part in getting_started.

9.13.4 Test

You can use the following command to test a model.

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

Example: test TSM model on Kinetics-400 dataset and dump the result to a json file.

```
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
```

For more details, you can refer to **Test a dataset** part in getting_started.

9.13.5 Citation

```
@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},
  journal =  {CVPR},
  year =     {2018}
}
```

9.14 TSN

Temporal segment networks: Towards good practices for deep action recognition

9.14.1 Abstract

Deep convolutional networks have achieved great success for visual recognition in still images. However, for action recognition in videos, the advantage over traditional methods is not so evident. This paper aims to discover the principles to design effective ConvNet architectures for action recognition in videos and learn these models given limited training samples. Our first contribution is temporal segment network (TSN), a novel framework for video-based action recognition. which is based on the idea of long-range temporal structure modeling. It combines a sparse temporal sampling strategy and video-level supervision to enable efficient and effective learning using the whole action video. The other contribution is our study on a series of good practices in learning ConvNets on video data with the help of temporal segment network. Our approach obtains the state-the-of-art performance on the datasets of HMDB51 (69.4%) and UCF101 (94.2%). We also visualize the learned ConvNet models, which qualitatively demonstrates the effectiveness of temporal segment network and the proposed good practices.

9.14.2 Results and Models

UCF-101

[1] We report the performance on UCF-101 split1.

Diving48

HMDB51

Kinetics-400

Here, We use [1: 1] to indicate that we combine rgb and flow score with coefficients 1: 1 to get the two-stream prediction (without applying softmax).

Using backbones from 3rd-party in TSN

It's possible and convenient to use a 3rd-party backbone for TSN under the framework of MMAction2, here we provide some examples for:

- [x] Backbones from MMClassification
- [x] Backbones from TorchVision
- [x] Backbones from TIMM (pytorch-image-models)

1. Note that some backbones in TIMM are not supported due to multiple reasons. Please refer to to PR ##880 for details.

Kinetics-400 Data Benchmark (8-gpus, ResNet50, ImageNet pretrain; 3 segments)

In data benchmark, we compare:

1. Different data preprocessing methods: (1) Resize video to 340x256, (2) Resize the short edge of video to 320px, (3) Resize the short edge of video to 256px;
2. Different data augmentation methods: (1) MultiScaleCrop, (2) RandomResizedCrop;
3. Different testing protocols: (1) 25 frames x 10 crops, (2) 25 frames x 3 crops.

Kinetics-400 OmniSource Experiments

[1] We obtain the pre-trained model from torch-hub, the pretrain model we used is `resnet50_swsl`

Kinetics-600

Kinetics-700

Something-Something V1

Something-Something V2

Moments in Time

Multi-Moments in Time

ActivityNet v1.3

HVU

[1] For simplicity, we train a specific model for each tag category as the baselines for HVU.

[2] The performance of HATNet and HATNet-multi are from the paper Large Scale Holistic Video Understanding. The proposed HATNet is a 2 branch Convolution Network (one 2D branch, one 3D branch) and share the same backbone(ResNet18) with us. The inputs of HATNet are 16 or 32 frames long video clips (which is much larger than us), while the input resolution is coarser (112 instead of 224). HATNet is trained on each individual task (each tag category) while HATNet-multi is trained on multiple tasks. Since there is no released codes or models for the HATNet, we just include the performance reported by the original paper.

Note:

1. The **gpus** indicates the number of gpu we used to get the checkpoint. It is noteworthy that the configs we provide are used for 8 gpus as default. According to the Linear Scaling Rule, you may set the learning rate proportional to the batch size if you use different GPUs or videos per GPU, e.g., lr=0.01 for 4 GPUs x 2 video/gpu and lr=0.08 for 16 GPUs x 4 video/gpu.
2. The **inference_time** is got by this benchmark script, where we use the sampling frames strategy of the test setting and only care about the model inference time, not including the IO time and pre-processing time. For each setting, we use 1 gpu and set batch size (videos per gpu) to 1 to calculate the inference time.
3. The values in columns named after "reference" are the results got by training on the original repo, using the same model settings.
4. The validation set of Kinetics400 we used consists of 19796 videos. These videos are available at Kinetics400-Validation. The corresponding data list (each line is of the format 'video_id, num_frames, label_index') and the label map are also available.

For more details on data preparation, you can refer to

- preparing_ucf101
- preparing_kinetics
- preparing_sthv1
- preparing_sthv2
- preparing_mit
- preparing_mmit
- preparing_hvu

- preparing_hmdb51

9.14.3 Train

You can use the following command to train a model.

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

Example: train TSN model on Kinetics-400 dataset in a deterministic option with periodic validation.

```
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
```

For more details, you can refer to **Training setting** part in getting_started.

9.14.4 Test

You can use the following command to test a model.

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

Example: test TSN model on Kinetics-400 dataset and dump the result to a json file.

```
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
```

For more details, you can refer to **Test a dataset** part in getting_started.

9.14.5 Citation

```
@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}
}
```

9.15 X3D

X3D: Expanding Architectures for Efficient Video Recognition

9.15.1 Abstract

This paper presents X3D, a family of efficient video networks that progressively expand a tiny 2D image classification architecture along multiple network axes, in space, time, width and depth. Inspired by feature selection methods in machine learning, a simple stepwise network expansion approach is employed that expands a single axis in each step, such that good accuracy to complexity trade-off is achieved. To expand X3D to a specific target complexity, we perform progressive forward expansion followed by backward contraction. X3D achieves state-of-the-art performance while requiring 4.8x and 5.5x fewer multiply-adds and parameters for similar accuracy as previous work. Our most surprising finding is that networks with high spatiotemporal resolution can perform well, while being extremely light in terms of network width and parameters. We report competitive accuracy at unprecedented efficiency on video classification and detection benchmarks.

9.15.2 Results and Models

Kinetics-400

[1] The models are ported from the repo SlowFast and tested on our data. Currently, we only support the testing of X3D models, training will be available soon.

Note:

1. The values in columns named after "reference" are the results got by testing the checkpoint released on the original repo and codes, using the same dataset with ours.
2. The validation set of Kinetics400 we used consists of 19796 videos. These videos are available at Kinetics400-Validation. The corresponding data list (each line is of the format 'video_id, num_frames, label_index') and the label map are also available.

For more details on data preparation, you can refer to Kinetics400 in *Data Preparation*.

9.15.3 Test

You can use the following command to test a model.

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

Example: test X3D model on Kinetics-400 dataset and dump the result to a json file.

```
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
```

For more details, you can refer to **Test a dataset** part in getting_started.

9.15.4 Citation

```
@misc{feichtenhofer2020x3d,
      title={X3D: Expanding Architectures for Efficient Video Recognition},
      author={Christoph Feichtenhofer},
      year={2020},
      eprint={2004.04730},
      archivePrefix={arXiv},
      primaryClass={cs.CV}
}
```

9.16 ResNet for Audio

Audiovisual SlowFast Networks for Video Recognition

9.16.1 Abstract

We present Audiovisual SlowFast Networks, an archi- tecture for integrated audiovisual perception. AVSlowFast has Slow and Fast visual pathways that are deeply inte- grated with a Faster Audio pathway to model vision and sound in a unified representation. We fuse audio and vi- sual features at multiple layers, enabling audio to con- tribute to the formation of hierarchical audiovisual con- cepts. To overcome training difficulties that arise from dif- ferent learning dynamics for audio and visual modalities, we introduce DropPathway, which randomly drops the Au- dio pathway during training as an effective regularization technique. Inspired by prior studies in neuroscience, we perform hierarchical audiovisual synchronization to learn joint audiovisual features. We report state-of-the-art results on six video action classification and detection datasets, perform detailed ablation studies, and show the gener- alization of AVSlowFast to learn self-supervised audiovi- sual features. Code will be made available at: https: //github.com/facebookresearch/SlowFast.

9.16.2 Results and Models

Kinetics-400

Note:

1. The **gpus** indicates the number of gpus we used to get the checkpoint. It is noteworthy that the configs we provide are used for 8 gpus as default. According to the Linear Scaling Rule, you may set the learning rate proportional to the batch size if you use different GPUs or videos per GPU, e.g., lr=0.01 for 4 GPUs x 2 video/gpu and lr=0.08 for 16 GPUs x 4 video/gpu.
2. The **inference_time** is got by this benchmark script, where we use the sampling frames strategy of the test setting and only care about the model inference time, not including the IO time and pre-processing time. For each setting, we use 1 gpu and set batch size (videos per gpu) to 1 to calculate the inference time.
3. The validation set of Kinetics400 we used consists of 19796 videos. These videos are available at Kinetics400-Validation. The corresponding data list (each line is of the format 'video_id, num_frames, label_index') and the label map are also available.

For more details on data preparation, you can refer to `Prepare audio` in [Data Preparation](data_preparation.md).

9.16.3 Train

You can use the following command to train a model.

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

Example: train ResNet model on Kinetics-400 audio dataset in a deterministic option with periodic validation.

```
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
```

For more details, you can refer to **Training setting** part in getting_started.

9.16.4 Test

You can use the following command to test a model.

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

Example: test ResNet model on Kinetics-400 audio dataset and dump the result to a json file.

```
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
```

For more details, you can refer to **Test a dataset** part in getting_started.

9.16.5 Fusion

For multi-modality fusion, you can use the simple script, the standard usage is:

```
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: The saved output file of `tools/test.py` by the argument `--out`.
- VISUAL_RESULT_PKL: The saved output file of `tools/test.py` by the argument `--out`.

9.16.6 Citation

```
@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}
}
```

CHAPTER

TEN

ACTION LOCALIZATION MODELS

10.1 BMN

Bmn: Boundary-matching network for temporal action proposal generation

10.1.1 Abstract

Temporal action proposal generation is an challenging and promising task which aims to locate temporal regions in real-world videos where action or event may occur. Current bottom-up proposal generation methods can generate proposals with precise boundary, but cannot efficiently generate adequately reliable confidence scores for retrieving proposals. To address these difficulties, we introduce the Boundary-Matching (BM) mechanism to evaluate confidence scores of densely distributed proposals, which denote a proposal as a matching pair of starting and ending boundaries and combine all densely distributed BM pairs into the BM confidence map. Based on BM mechanism, we propose an effective, efficient and end-to-end proposal generation method, named Boundary-Matching Network (BMN), which generates proposals with precise temporal boundaries as well as reliable confidence scores simultaneously. The two-branches of BMN are jointly trained in an unified framework. We conduct experiments on two challenging datasets: THUMOS-14 and ActivityNet-1.3, where BMN shows significant performance improvement with remarkable efficiency and generalizability. Further, combining with existing action classifier, BMN can achieve state-of-the-art temporal action detection performance.

10.1.2 Results and Models

ActivityNet feature

Note:

1. The **gpus** indicates the number of gpu we used to get the checkpoint. According to the Linear Scaling Rule, you may set the learning rate proportional to the batch size if you use different GPUs or videos per GPU, e.g., lr=0.01 for 4 GPUs x 2 video/gpu and lr=0.08 for 16 GPUs x 4 video/gpu.
2. For feature column, cuhk_mean_100 denotes the widely used cuhk activitynet feature extracted by anet2016-cuhk, mmaction_video and mmaction_clip denote feature extracted by mmaction, with video-level activitynet finetuned model or clip-level activitynet finetuned model respectively.
3. We evaluate the action detection performance of BMN, using anet_cuhk_2017 submission for ActivityNet2017 Untrimmed Video Classification Track to assign label for each action proposal.

*We train BMN with the official repo, evaluate its proposal generation and action detection performance with anet_cuhk_2017 for label assigning.

For more details on data preparation, you can refer to ActivityNet feature in *Data Preparation*.

10.1.3 Train

You can use the following command to train a model.

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

Example: train BMN model on ActivityNet features dataset.

```
python tools/train.py configs/localization/bmn/bmn_400x100_2x8_9e_activitynet_feature.py
```

For more details and optional arguments infos, you can refer to **Training setting** part in getting_started .

10.1.4 Test

You can use the following command to test a model.

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

Example: test BMN on ActivityNet feature dataset.

```
## Note: If evaluated, then please make sure the annotation file for test data contains␣
↪groundtruth.
python tools/test.py configs/localization/bmn/bmn_400x100_2x8_9e_activitynet_feature.py␣
↪checkpoints/SOME_CHECKPOINT.pth --eval AR@AN --out results.json
```

You can also test the action detection performance of the model, with anet_cuhk_2017 prediction file and generated proposal file (`results.json` in last command).

```
python tools/analysis/report_map.py --proposal path/to/proposal_file
```

Note:

1. (Optional) You can use the following command to generate a formatted proposal file, which will be fed into the action classifier (Currently supports SSN and P-GCN, not including TSN, I3D etc.) to get the classification result of proposals.

   ```
   python tools/data/activitynet/convert_proposal_format.py
   ```

For more details and optional arguments infos, you can refer to **Test a dataset** part in getting_started .

10.1.5 Citation

```
@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}
}
```

10.2 BSN

Bsn: Boundary sensitive network for temporal action proposal generation

10.2.1 Abstract

Temporal action proposal generation is an important yet challenging problem, since temporal proposals with rich action content are indispensable for analysing real-world videos with long duration and high proportion irrelevant content. This problem requires methods not only generating proposals with precise temporal boundaries, but also retrieving proposals to cover truth action instances with high recall and high overlap using relatively fewer proposals. To address these difficulties, we introduce an effective proposal generation method, named Boundary-Sensitive Network (BSN), which adopts "local to global" fashion. Locally, BSN first locates temporal boundaries with high probabilities, then directly combines these boundaries as proposals. Globally, with Boundary-Sensitive Proposal feature, BSN retrieves proposals by evaluating the confidence of whether a proposal contains an action within its region. We conduct experiments on two challenging datasets: ActivityNet-1.3 and THUMOS14, where BSN outperforms other state-of-the-art temporal action proposal generation methods with high recall and high temporal precision. Finally, further experiments demonstrate that by combining existing action classifiers, our method significantly improves the state-of-the-art temporal action detection performance.

10.2.2 Results and Models

ActivityNet feature

Note:

1. The **gpus** indicates the number of gpu we used to get the checkpoint. According to the Linear Scaling Rule, you may set the learning rate proportional to the batch size if you use different GPUs or videos per GPU, e.g., lr=0.01 for 4 GPUs x 2 video/gpu and lr=0.08 for 16 GPUs x 4 video/gpu.

2. For feature column, cuhk_mean_100 denotes the widely used cuhk activitynet feature extracted by anet2016-cuhk, mmaction_video and mmaction_clip denote feature extracted by mmaction, with video-level activitynet finetuned model or clip-level activitynet finetuned model respectively.

For more details on data preparation, you can refer to ActivityNet feature in *Data Preparation*.

10.2.3 Train

You can use the following commands to train a model.

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

Examples:

1. train BSN(TEM) on ActivityNet features dataset.

```
python tools/train.py configs/localization/bsn/bsn_tem_400x100_1x16_20e_activitynet_
↪feature.py
```

2. train BSN(PEM) on PGM results.

```
python tools/train.py configs/localization/bsn/bsn_pem_400x100_1x16_20e_activitynet_
↪feature.py
```

For more details and optional arguments infos, you can refer to **Training setting** part in getting_started.

10.2.4 Inference

You can use the following commands to inference a model.

1. For TEM Inference

```
## Note: This could not be evaluated.
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [optional arguments]
```

2. For PGM Inference

```
python tools/misc/bsn_proposal_generation.py ${CONFIG_FILE} [--mode ${MODE}]
```

3. For PEM Inference

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

Examples:

1. Inference BSN(TEM) with pretrained model.

```
python tools/test.py configs/localization/bsn/bsn_tem_400x100_1x16_20e_activitynet_
↪feature.py checkpoints/SOME_CHECKPOINT.pth
```

2. Inference BSN(PGM) with pretrained model.

```
python tools/misc/bsn_proposal_generation.py configs/localization/bsn/bsn_pgm_
↪400x100_activitynet_feature.py --mode train
```

3. Inference BSN(PEM) with evaluation metric 'AR@AN' and output the results.

```
## Note: If evaluated, then please make sure the annotation file for test data␣
↪contains groundtruth.
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 Test

You can use the following commands to test a model.

1. TEM

```
## Note: This could not be evaluated.
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]
```

Examples:

1. Test a TEM model on ActivityNet dataset.

```
python tools/test.py configs/localization/bsn/bsn_tem_400x100_1x16_20e_activitynet_
↪feature.py checkpoints/SOME_CHECKPOINT.pth
```

2. Test a PGM model on ActivityNet dataset.

```
python tools/misc/bsn_proposal_generation.py configs/localization/bsn/bsn_pgm_
↪400x100_activitynet_feature.py --mode test
```

3. Test a PEM model with with evaluation metric 'AR@AN' and output the results.

```
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
```

Note:

1. (Optional) You can use the following command to generate a formatted proposal file, which will be fed into the action classifier (Currently supports only SSN and P-GCN, not including TSN, I3D etc.) to get the classification result of proposals.

```
python tools/data/activitynet/convert_proposal_format.py
```

For more details and optional arguments infos, you can refer to **Test a dataset** part in getting_started.

10.2.6 Citation

```
@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}
}
```

10.3 SSN

Temporal Action Detection With Structured Segment Networks

10.3.1 Abstract

Detecting actions in untrimmed videos is an important yet challenging task. In this paper, we present the structured segment network (SSN), a novel framework which models the temporal structure of each action instance via a structured temporal pyramid. On top of the pyramid, we further introduce a decomposed discriminative model comprising two classifiers, respectively for classifying actions and determining completeness. This allows the framework to effectively distinguish positive proposals from background or incomplete ones, thus leading to both accurate recognition and localization. These components are integrated into a unified network that can be efficiently trained in an end-to-end fashion. Additionally, a simple yet effective temporal action proposal scheme, dubbed temporal actionness grouping (TAG) is devised to generate high quality action proposals. On two challenging benchmarks, THUMOS14 and ActivityNet, our method remarkably outperforms previous state-of-the-art methods, demonstrating superior accuracy and strong adaptivity in handling actions with various temporal structures.

10.3.2 Results and Models

Note:

1. The **gpus** indicates the number of gpu we used to get the checkpoint. According to the Linear Scaling Rule, you may set the learning rate proportional to the batch size if you use different GPUs or videos per GPU, e.g., lr=0.01 for 4 GPUs x 2 video/gpu and lr=0.08 for 16 GPUs x 4 video/gpu.
2. Since SSN utilizes different structured temporal pyramid pooling methods at training and testing, please refer to ssn_r50_450e_thumos14_rgb_train at training and ssn_r50_450e_thumos14_rgb_test at testing.
3. We evaluate the action detection performance of SSN, using action proposals of TAG. For more details on data preparation, you can refer to thumos14 TAG proposals in *Data Preparation*.
4. The reference SSN in is evaluated with `ResNet50` backbone in MMAction, which is the same backbone with ours. Note that the original setting of MMAction SSN uses the `BNInception` backbone.

10.3.3 Train

You can use the following command to train a model.

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

Example: train SSN model on thumos14 dataset.

```
python tools/train.py configs/localization/ssn/ssn_r50_450e_thumos14_rgb_train.py
```

For more details and optional arguments infos, you can refer to **Training setting** part in getting_started.

10.3.4 Test

You can use the following command to test a model.

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

Example: test BMN on ActivityNet feature dataset.

```
## Note: If evaluated, then please make sure the annotation file for test data contains␣
↪groundtruth.
python tools/test.py configs/localization/ssn/ssn_r50_450e_thumos14_rgb_test.py␣
↪checkpoints/SOME_CHECKPOINT.pth --eval mAP
```

For more details and optional arguments infos, you can refer to **Test a dataset** part in getting_started.

10.3.5 Citation

```
@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}
}
```

CHAPTER

ELEVEN

SPATIO TEMPORAL ACTION DETECTION MODELS

11.1 ACRN

Actor-centric relation network

11.1.1 Abstract

Current state-of-the-art approaches for spatio-temporal action localization rely on detections at the frame level and model temporal context with 3D ConvNets. Here, we go one step further and model spatio-temporal relations to capture the interactions between human actors, relevant objects and scene elements essential to differentiate similar human actions. Our approach is weakly supervised and mines the relevant elements automatically with an actor-centric relational network (ACRN). ACRN computes and accumulates pair-wise relation information from actor and global scene features, and generates relation features for action classification. It is implemented as neural networks and can be trained jointly with an existing action detection system. We show that ACRN outperforms alternative approaches which capture relation information, and that the proposed framework improves upon the state-of-the-art performance on JHMDB and AVA. A visualization of the learned relation features confirms that our approach is able to attend to the relevant relations for each action.

11.1.2 Results and Models

AVA2.1

AVA2.2

Note:

1. The **gpus** indicates the number of gpu we used to get the checkpoint. According to the Linear Scaling Rule, you may set the learning rate proportional to the batch size if you use different GPUs or videos per GPU, e.g., lr=0.01 for 4 GPUs x 2 video/gpu and lr=0.08 for 16 GPUs x 4 video/gpu.

For more details on data preparation, you can refer to AVA in *Data Preparation*.

11.1.3 Train

You can use the following command to train a model.

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

Example: train ACRN with SlowFast backbone on AVA with periodic validation.

```
python tools/train.py configs/detection/acrn/slowfast_acrn_kinetics_pretrained_r50_8x8x1_
↪cosine_10e_ava22_rgb.py --validate
```

For more details and optional arguments infos, you can refer to **Training setting** part in getting_started.

11.1.4 Test

You can use the following command to test a model.

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

Example: test ACRN with SlowFast backbone on AVA and dump the result to a csv file.

```
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
```

For more details and optional arguments infos, you can refer to **Test a dataset** part in getting_started .

11.1.5 Citation

```
@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}
}
```

11.2 AVA

Ava: A video dataset of spatio-temporally localized atomic visual actions

11.2.1 Abstract

This paper introduces a video dataset of spatio-temporally localized Atomic Visual Actions (AVA). The AVA dataset densely annotates 80 atomic visual actions in 430 15-minute video clips, where actions are localized in space and time, resulting in 1.58M action labels with multiple labels per person occurring frequently. The key characteristics of our dataset are: (1) the definition of atomic visual actions, rather than composite actions; (2) precise spatio-temporal annotations with possibly multiple annotations for each person; (3) exhaustive annotation of these atomic actions over 15-minute video clips; (4) people temporally linked across consecutive segments; and (5) using movies to gather a varied set of action representations. This departs from existing datasets for spatio-temporal action recognition, which typically provide sparse annotations for composite actions in short video clips. We will release the dataset publicly. AVA, with its realistic scene and action complexity, exposes the intrinsic difficulty of action recognition. To benchmark this, we present a novel approach for action localization that builds upon the current state-of-the-art methods, and demonstrates better performance on JHMDB and UCF101-24 categories. While setting a new state of the art on existing datasets, the overall results on AVA are low at 15.6% mAP, underscoring the need for developing new approaches for video understanding.

```
@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 Results and Models

AVA2.1

AVA2.2

Note:

1. The **gpus** indicates the number of gpu we used to get the checkpoint. According to the Linear Scaling Rule, you may set the learning rate proportional to the batch size if you use different GPUs or videos per GPU, e.g., lr=0.01 for 4 GPUs x 2 video/gpu and lr=0.08 for 16 GPUs x 4 video/gpu.
2. **Context** indicates that using both RoI feature and global pooled feature for classification, which leads to around 1% mAP improvement in general.

For more details on data preparation, you can refer to AVA in *Data Preparation*.

11.2.3 Train

You can use the following command to train a model.

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

Example: train SlowOnly model on AVA with periodic validation.

```
python tools/train.py configs/detection/ava/slowonly_kinetics_pretrained_r50_8x8x1_20e_
↪ava_rgb.py --validate
```

For more details and optional arguments infos, you can refer to **Training setting** part in getting_started .

Train Custom Classes From Ava Dataset

You can train custom classes from ava. Ava suffers from class imbalance. There are more then 100,000 samples for classes like `stand/listen to (a person)/talk to (e.g., self, a person, a group)/watch (a person)`, whereas half of all classes has less than 500 samples. In most cases, training custom classes with fewer samples only will lead to better results.

Three steps to train custom classes:

- Step 1: Select custom classes from original classes, named `custom_classes`. Class `0` should not be selected since it is reserved for further usage (to identify whether a proposal is positive or negative, not implemented yet) and will be added automatically.
- Step 2: Set `num_classes`. In order to be compatible with current codes, Please make sure `num_classes == len(custom_classes) + 1`.
  - The new class `0` corresponds to original class `0`. The new class `i`(i > 0) corresponds to original class `custom_classes[i-1]`.
  - There are three `num_classes` in ava config, `model -> roi_head -> bbox_head -> num_classes`, `data -> train -> num_classes` and `data -> val -> num_classes`.
  - If `num_classes <= 5`, input arg `topk` of `BBoxHeadAVA` should be modified. The default value of `topk` is `(3, 5)`, and all elements of `topk` must be smaller than `num_classes`.
- Step 3: Make sure all custom classes are in `label_file`. It is worth mentioning that there are two label files, `ava_action_list_v2.1_for_activitynet_2018.pbtxt`(contains 60 classes, 20 classes are missing) and `ava_action_list_v2.1.pbtxt`(contains all 80 classes).

Take `slowonly_kinetics_pretrained_r50_4x16x1_20e_ava_rgb` as an example, training custom classes with AP in range `(0.1, 0.3)`, aka `[3, 6, 10, 27, 29, 38, 41, 48, 51, 53, 54, 59, 61, 64, 70, 72]`. Please note that, the previously mentioned AP is calculated by original ckpt, which is trained by all 80 classes. The results are listed as follows.

11.2.4 Test

You can use the following command to test a model.

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

Example: test SlowOnly model on AVA and dump the result to a csv file.

```
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
```

For more details and optional arguments infos, you can refer to **Test a dataset** part in getting_started .

11.2.5 Citation

```
@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}
}
```

11.3 LFB

Long-term feature banks for detailed video understanding

11.3.1 Abstract

To understand the world, we humans constantly need to relate the present to the past, and put events in context. In this paper, we enable existing video models to do the same. We propose a long-term feature bank—supportive information extracted over the entire span of a video—to augment state-of-the-art video models that otherwise would only view short clips of 2-5 seconds. Our experiments demonstrate that augmenting 3D convolutional networks with a long-term feature bank yields state-of-the-art results on three challenging video datasets: AVA, EPIC-Kitchens, and Charades.

11.3.2 Results and Models

AVA2.1

Note:

1. The **gpus** indicates the number of gpu we used to get the checkpoint. According to the Linear Scaling Rule, you may set the learning rate proportional to the batch size if you use different GPUs or videos per GPU, e.g., lr=0.01 for 4 GPUs x 2 video/gpu and lr=0.08 for 16 GPUs x 4 video/gpu.
2. We use `slowonly_r50_4x16x1` instead of `I3D-R50-NL` in the original paper as the backbone of LFB, but we have achieved the similar improvement: (ours: 20.1 -> 24.11 vs. author: 22.1 -> 25.8).
3. Because the long-term features are randomly sampled in testing, the test accuracy may have some differences.

4. Before train or test lfb, you need to infer feature bank with the lfb_slowonly_r50_ava_infer.py. For more details on infer feature bank, you can refer to Train part.
5. You can also dowonload long-term feature bank from AVA_train_val_float32_lfb or AVA_train_val_float16_lfb, and then put them on `lfb_prefix_path`.
6. The ROIHead now supports single-label classification (i.e. the network outputs at most one-label per actor). This can be done by (a) setting multilabel=False during training and the test_cfg.rcnn.action_thr for testing.

11.3.3 Train

a. Infer long-term feature bank for training

Before train or test lfb, you need to infer long-term feature bank first.

Specifically, run the test on the training, validation, testing dataset with the config file lfb_slowonly_r50_ava_infer (The config file will only infer the feature bank of training dataset and you need set `dataset_mode = 'val'` to infer the feature bank of validation dataset in the config file.), and the shared head LFBInferHead will generate the feature bank.

A long-term feature bank file of AVA training and validation datasets with float32 precision occupies 3.3 GB. If store the features with float16 precision, the feature bank occupies 1.65 GB.

You can use the following command to infer feature bank of AVA training and validation dataset and the feature bank will be stored in `lfb_prefix_path/lfb_train.pkl` and `lfb_prefix_path/lfb_val.pkl`.

```
## set `dataset_mode = 'train'` in lfb_slowonly_r50_ava_infer.py
python tools/test.py configs/detection/lfb/lfb_slowonly_r50_ava_infer.py \
    checkpoints/YOUR_BASELINE_CHECKPOINT.pth --eval mAP

## set `dataset_mode = 'val'` in lfb_slowonly_r50_ava_infer.py
python tools/test.py configs/detection/lfb/lfb_slowonly_r50_ava_infer.py \
    checkpoints/YOUR_BASELINE_CHECKPOINT.pth --eval mAP
```

We use slowonly_r50_4x16x1 checkpoint from slowonly_kinetics_pretrained_r50_4x16x1_20e_ava_rgb to infer feature bank.

b. Train LFB

You can use the following command to train a model.

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

Example: train LFB model on AVA with half-precision long-term feature bank.

```
python tools/train.py configs/detection/lfb/lfb_nl_kinetics_pretrained_slowonly_r50_
↪4x16x1_20e_ava_rgb.py \
  --validate --seed 0 --deterministic
```

For more details and optional arguments infos, you can refer to **Training setting** part in getting_started.

11.3.4 Test

a. Infer long-term feature bank for testing

Before train or test lfb, you also need to infer long-term feature bank first. If you have generated the feature bank file, you can skip it.

The step is the same with **Infer long-term feature bank for training** part in Train.

b. Test LFB

You can use the following command to test a model.

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

Example: test LFB model on AVA with half-precision long-term feature bank and dump the result to a csv file.

```
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
```

For more details, you can refer to **Test a dataset** part in getting_started.

11.3.5 Citation

```
@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{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}
}
```

CHAPTER

TWELVE

SKELETON-BASED ACTION RECOGNITION MODELS

12.1 AGCN

Two-Stream Adaptive Graph Convolutional Networks for Skeleton-Based Action Recognition

12.1.1 Abstract

In skeleton-based action recognition, graph convolutional networks (GCNs), which model the human body skeletons as spatiotemporal graphs, have achieved remarkable performance. However, in existing GCN-based methods, the topology of the graph is set manually, and it is fixed over all layers and input samples. This may not be optimal for the hierarchical GCN and diverse samples in action recognition tasks. In addition, the second-order information (the lengths and directions of bones) of the skeleton data, which is naturally more informative and discriminative for action recognition, is rarely investigated in existing methods. In this work, we propose a novel two-stream adaptive graph convolutional network (2s-AGCN) for skeleton-based action recognition. The topology of the graph in our model can be either uniformly or individually learned by the BP algorithm in an end-to-end manner. This data-driven method increases the flexibility of the model for graph construction and brings more generality to adapt to various data samples. Moreover, a two-stream framework is proposed to model both the first-order and the second-order information simultaneously, which shows notable improvement for the recognition accuracy. Extensive experiments on the two large-scale datasets, NTU-RGBD and Kinetics-Skeleton, demonstrate that the performance of our model exceeds the state-of-the-art with a significant margin.

12.1.2 Results and Models

NTU60_XSub

12.1.3 Train

You can use the following command to train a model.

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

Example: train AGCN model on joint data of NTU60 dataset in a deterministic option with periodic validation.

```
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
```

Example: train AGCN model on bone data of NTU60 dataset in a deterministic option with periodic validation.

```
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
```

For more details, you can refer to **Training setting** part in getting_started.

12.1.4 Test

You can use the following command to test a model.

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

Example: test AGCN model on joint data of NTU60 dataset and dump the result to a pickle file.

```
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
```

Example: test AGCN model on bone data of NTU60 dataset and dump the result to a pickle file.

```
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
```

For more details, you can refer to **Test a dataset** part in getting_started.

12.1.5 Citation

```
@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}
}
```

12.2 PoseC3D

Revisiting Skeleton-based Action Recognition

12.2.1 Abstract

Human skeleton, as a compact representation of human action, has received increasing attention in recent years. Many skeleton-based action recognition methods adopt graph convolutional networks (GCN) to extract features on top of human skeletons. Despite the positive results shown in previous works, GCN-based methods are subject to limitations in robustness, interoperability, and scalability. In this work, we propose PoseC3D, a new approach to skeleton-based action recognition, which relies on a 3D heatmap stack instead of a graph sequence as the base representation of human skeletons. Compared to GCN-based methods, PoseC3D is more effective in learning spatiotemporal features, more robust against pose estimation noises, and generalizes better in cross-dataset settings. Also, PoseC3D can handle multiple-person scenarios without additional computation cost, and its features can be easily integrated with other modalities at early fusion stages, which provides a great design space to further boost the performance. On four challenging datasets, PoseC3D consistently obtains superior performance, when used alone on skeletons and in combination with the RGB modality.

12.2.2 Results and Models

FineGYM

NTU60_XSub

NTU120_XSub

UCF101

HMDB51

Note:

1. The **gpus** indicates the number of gpu we used to get the checkpoint. It is noteworthy that the configs we provide are used for 8 gpus as default. According to the Linear Scaling Rule, you may set the learning rate proportional to the batch size if you use different GPUs or videos per GPU, e.g., lr=0.01 for 8 GPUs x 8 videos/gpu and lr=0.04 for 16 GPUs x 16 videos/gpu.
2. You can follow the guide in Preparing Skeleton Dataset to obtain skeleton annotations used in the above configs.

12.2.3 Train

You can use the following command to train a model.

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

Example: train PoseC3D model on FineGYM dataset in a deterministic option with periodic validation.

```
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
```

For training with your custom dataset, you can refer to Custom Dataset Training.

For more details, you can refer to **Training setting** part in getting_started.

12.2.4 Test

You can use the following command to test a model.

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

Example: test PoseC3D model on FineGYM dataset and dump the result to a pickle file.

```
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
```

For more details, you can refer to **Test a dataset** part in getting_started.

12.2.5 Citation

```
@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}
}
```

12.3 STGCN

Spatial temporal graph convolutional networks for skeleton-based action recognition

12.3.1 Abstract

Dynamics of human body skeletons convey significant information for human action recognition. Conventional approaches for modeling skeletons usually rely on hand-crafted parts or traversal rules, thus resulting in limited expressive power and difficulties of generalization. In this work, we propose a novel model of dynamic skeletons called Spatial-Temporal Graph Convolutional Networks (ST-GCN), which moves beyond the limitations of previous methods by automatically learning both the spatial and temporal patterns from data. This formulation not only leads to greater expressive power but also stronger generalization capability. On two large datasets, Kinetics and NTU-RGBD, it achieves substantial improvements over mainstream methods.

12.3.2 Results and Models

NTU60_XSub

BABEL

* The number is copied from the paper, the performance of the released checkpoints for BABEL-120 is inferior.

12.3.3 Train

You can use the following command to train a model.

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

Example: train STGCN model on NTU60 dataset in a deterministic option with periodic validation.

```
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
```

For more details, you can refer to **Training setting** part in getting_started.

12.3.4 Test

You can use the following command to test a model.

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

Example: test STGCN model on NTU60 dataset and dump the result to a pickle file.

```
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
```

For more details, you can refer to **Test a dataset** part in getting_started.

12.3.5 Citation

```
@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}
}
```

CHAPTER

THIRTEEN

TUTORIAL 1: LEARN ABOUT CONFIGS

We use python files as configs, incorporate modular and inheritance design into our config system, which is convenient to conduct various experiments. You can find all the provided configs under `$MMAction2/configs`. If you wish to inspect the config file, you may run `python tools/analysis/print_config.py /PATH/TO/CONFIG` to see the complete config.

13.1 Modify config through script arguments

When submitting jobs using "tools/train.py" or "tools/test.py", you may specify `--cfg-options` to in-place modify the config.

- Update config keys of dict.

  The config options can be specified following the order of the dict keys in the original config. For example, `--cfg-options model.backbone.norm_eval=False` changes the all BN modules in model backbones to `train` mode.

- Update keys inside a list of configs.

  Some config dicts are composed as a list in your config. For example, the training pipeline `data.train.pipeline` is normally a list e.g. `[dict(type='SampleFrames'), ...]`. If you want to change `'SampleFrames'` to `'DenseSampleFrames'` in the pipeline, you may specify `--cfg-options data.train.pipeline.0.type=DenseSampleFrames`.

- Update values of list/tuples.

  If the value to be updated is a list or a tuple. For example, the config file normally sets `workflow=[('train', 1)]`. If you want to change this key, you may specify `--cfg-options workflow="[(train,1),(val,1)]"`. Note that the quotation mark " is necessary to support list/tuple data types, and that **NO** white space is allowed inside the quotation marks in the specified value.

13.2 Config File Structure

There are 3 basic component types under `config/_base_`, model, schedule, default_runtime. Many methods could be easily constructed with one of each like TSN, I3D, SlowOnly, etc. The configs that are composed by components from `_base_` are called *primitive*.

For all configs under the same folder, it is recommended to have only **one** *primitive* config. All other configs should inherit from the *primitive* config. In this way, the maximum of inheritance level is 3.

For easy understanding, we recommend contributors to inherit from exiting methods. For example, if some modification is made base on TSN, users may first inherit the basic TSN structure by specifying `_base_ = ../tsn/tsn_r50_1x1x3_100e_kinetics400_rgb.py`, then modify the necessary fields in the config files.

If you are building an entirely new method that does not share the structure with any of the existing methods, you may create a folder under `configs/TASK`.

Please refer to mmcv for detailed documentation.

13.3 Config File Naming Convention

We follow the style below to name config files. Contributors are advised to follow the same style.

```
{model}_[model setting]_{backbone}_[misc]_{data setting}_[gpu x batch_per_gpu]_{schedule}
↪_{dataset}_{modality}
```

`{xxx}` is required field and `[yyy]` is optional.

- `{model}`: model type, e.g. `tsn`, `i3d`, etc.
- `[model setting]`: specific setting for some models.
- `{backbone}`: backbone type, e.g. `r50` (ResNet-50), etc.
- `[misc]`: miscellaneous setting/plugins of model, e.g. `dense`, `320p`, `video`, etc.
- `{data setting}`: frame sample setting in `{clip_len}x{frame_interval}x{num_clips}` format.
- `[gpu x batch_per_gpu]`: GPUs and samples per GPU.
- `{schedule}`: training schedule, e.g. `20e` means 20 epochs.
- `{dataset}`: dataset name, e.g. `kinetics400`, `mmit`, etc.
- `{modality}`: frame modality, e.g. `rgb`, `flow`, etc.

13.3.1 Config System for Action localization

We incorporate modular design into our config system, which is convenient to conduct various experiments.

- An Example of BMN

  To help the users have a basic idea of a complete config structure and the modules in an action localization system, we make brief comments on the config of BMN as the following. For more detailed usage and alternative for per parameter in each module, please refer to the API documentation.

```
# model settings
model = dict(  # Config of the model
    type='BMN',  # Type of the localizer
    temporal_dim=100,  # Total frames selected for each video
    boundary_ratio=0.5,  # Ratio for determining video boundaries
    num_samples=32,  # Number of samples for each proposal
    num_samples_per_bin=3,  # Number of bin samples for each sample
    feat_dim=400,  # Dimension of feature
    soft_nms_alpha=0.4,  # Soft NMS alpha
    soft_nms_low_threshold=0.5,  # Soft NMS low threshold
    soft_nms_high_threshold=0.9,  # Soft NMS high threshold
    post_process_top_k=100)  # Top k proposals in post process
# model training and testing settings
train_cfg = None  # Config of training hyperparameters for BMN
test_cfg = dict(average_clips='score')  # Config for testing hyperparameters for BMN

# dataset settings
dataset_type = 'ActivityNetDataset'  # Type of dataset for training, validation and␣
↪testing
data_root = 'data/activitynet_feature_cuhk/csv_mean_100/'  # Root path to data for␣
↪training
data_root_val = 'data/activitynet_feature_cuhk/csv_mean_100/'  # Root path to data␣
↪for validation and testing
ann_file_train = 'data/ActivityNet/anet_anno_train.json'  # Path to the annotation␣
↪file for training
ann_file_val = 'data/ActivityNet/anet_anno_val.json'  # Path to the annotation file␣
↪for validation
ann_file_test = 'data/ActivityNet/anet_anno_test.json'  # Path to the annotation␣
↪file for testing

train_pipeline = [  # List of training pipeline steps
    dict(type='LoadLocalizationFeature'),  # Load localization feature pipeline
    dict(type='GenerateLocalizationLabels'),  # Generate localization labels␣
↪pipeline
    dict(  # Config of Collect
        type='Collect',  # Collect pipeline that decides which keys in the data␣
↪should be passed to the localizer
        keys=['raw_feature', 'gt_bbox'],  # Keys of input
        meta_name='video_meta',  # Meta name
        meta_keys=['video_name']),  # Meta keys of input
    dict(  # Config of ToTensor
        type='ToTensor',  # Convert other types to tensor type pipeline
        keys=['raw_feature']),  # Keys to be converted from image to tensor
    dict(  # Config of ToDataContainer
        type='ToDataContainer',  # Pipeline to convert the data to DataContainer
        fields=[dict(key='gt_bbox', stack=False, cpu_only=True)])  # Required␣
↪fields to be converted with keys and attributes
]
val_pipeline = [  # List of validation pipeline steps
    dict(type='LoadLocalizationFeature'),  # Load localization feature pipeline
    dict(type='GenerateLocalizationLabels'),  # Generate localization labels␣
↪pipeline
    dict(  # Config of Collect
```

(continues on next page)

(continued from previous page)

```
        type='Collect',  # Collect pipeline that decides which keys in the data␣
↪should be passed to the localizer
        keys=['raw_feature', 'gt_bbox'],  # Keys of input
        meta_name='video_meta',  # Meta name
        meta_keys=[
            'video_name', 'duration_second', 'duration_frame', 'annotations',
            'feature_frame'
        ]),  # Meta keys of input
    dict(  # Config of ToTensor
        type='ToTensor',  # Convert other types to tensor type pipeline
        keys=['raw_feature']),  # Keys to be converted from image to tensor
    dict(  # Config of ToDataContainer
        type='ToDataContainer',  # Pipeline to convert the data to DataContainer
        fields=[dict(key='gt_bbox', stack=False, cpu_only=True)])  # Required␣
↪fields to be converted with keys and attributes
]
test_pipeline = [  # List of testing pipeline steps
    dict(type='LoadLocalizationFeature'),  # Load localization feature pipeline
    dict(  # Config of Collect
        type='Collect',  # Collect pipeline that decides which keys in the data␣
↪should be passed to the localizer
        keys=['raw_feature'],  # Keys of input
        meta_name='video_meta',  # Meta name
        meta_keys=[
            'video_name', 'duration_second', 'duration_frame', 'annotations',
            'feature_frame'
        ]),  # Meta keys of input
    dict(  # Config of ToTensor
        type='ToTensor',  # Convert other types to tensor type pipeline
        keys=['raw_feature']),  # Keys to be converted from image to tensor
]
data = dict(  # Config of data
    videos_per_gpu=8,  # Batch size of each single GPU
    workers_per_gpu=8,  # Workers to pre-fetch data for each single GPU
    train_dataloader=dict(  # Additional config of train dataloader
        drop_last=True),  # Whether to drop out the last batch of data in training
    val_dataloader=dict(  # Additional config of validation dataloader
        videos_per_gpu=1),  # Batch size of each single GPU during evaluation
    test_dataloader=dict(  # Additional config of test dataloader
        videos_per_gpu=2),  # Batch size of each single GPU during testing
    test=dict(  # Testing dataset config
        type=dataset_type,
        ann_file=ann_file_test,
        pipeline=test_pipeline,
        data_prefix=data_root_val),
    val=dict(  # Validation dataset config
        type=dataset_type,
        ann_file=ann_file_val,
        pipeline=val_pipeline,
        data_prefix=data_root_val),
    train=dict(  # Training dataset config
        type=dataset_type,
```

(continues on next page)

(continued from previous page)

```
        ann_file=ann_file_train,
        pipeline=train_pipeline,
        data_prefix=data_root))

# optimizer
optimizer = dict(
    # Config used to build optimizer, support (1). All the optimizers in PyTorch
    # whose arguments are also the same as those in PyTorch. (2). Custom optimizers
    # which are built on `constructor`, referring to "tutorials/5_new_modules.md"
    # for implementation.
    type='Adam',  # Type of optimizer, refer to https://github.com/open-mmlab/mmcv/
↪blob/master/mmcv/runner/optimizer/default_constructor.py#L13 for more details
    lr=0.001,  # Learning rate, see detail usages of the parameters in the␣
↪documentation of PyTorch
    weight_decay=0.0001)  # Weight decay of Adam
optimizer_config = dict(  # Config used to build the optimizer hook
    grad_clip=None)  # Most of the methods do not use gradient clip
# learning policy
lr_config = dict(  # Learning rate scheduler config used to register LrUpdater hook
    policy='step',  # Policy of scheduler, also support CosineAnnealing, Cyclic,␣
↪etc. Refer to details of supported LrUpdater from https://github.com/open-mmlab/
↪mmcv/blob/master/mmcv/runner/hooks/lr_updater.py#L9
    step=7)  # Steps to decay the learning rate

total_epochs = 9  # Total epochs to train the model
checkpoint_config = dict(  # Config to set the checkpoint hook, Refer to https://
↪github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/checkpoint.py for␣
↪implementation
    interval=1)  # Interval to save checkpoint
evaluation = dict(  # Config of evaluation during training
    interval=1,  # Interval to perform evaluation
    metrics=['AR@AN'])  # Metrics to be performed
log_config = dict(  # Config to register logger hook
    interval=50,  # Interval to print the log
    hooks=[  # Hooks to be implemented during training
        dict(type='TextLoggerHook'),  # The logger used to record the training␣
↪process
        # dict(type='TensorboardLoggerHook'),  # The Tensorboard logger is also␣
↪supported
    ])

# runtime settings
dist_params = dict(backend='nccl')  # Parameters to setup distributed training, the␣
↪port can also be set
log_level = 'INFO'  # The level of logging
work_dir = './work_dirs/bmn_400x100_2x8_9e_activitynet_feature/'  # Directory to␣
↪save the model checkpoints and logs for the current experiments
load_from = None  # load models as a pre-trained model from a given path. This will␣
↪not resume training
resume_from = None  # Resume checkpoints from a given path, the training will be␣
↪resumed from the epoch when the checkpoint's is saved
workflow = [('train', 1)]  # Workflow for runner. [('train', 1)] means there is only␣
↪one workflow and the workflow named 'train' is executed once
```

(continues on next page)

(continued from previous page)

```
output_config = dict(  # Config of localization output
    out=f'{work_dir}/results.json',  # Path to output file
    output_format='json')  # File format of output file
```

13.3.2 Config System for Action Recognition

We incorporate modular design into our config system, which is convenient to conduct various experiments.

- An Example of TSN

  To help the users have a basic idea of a complete config structure and the modules in an action recognition system, we make brief comments on the config of TSN as the following. For more detailed usage and alternative for per parameter in each module, please refer to the API documentation.

```
# model settings
model = dict(  # Config of the model
    type='Recognizer2D',  # Type of the recognizer
    backbone=dict(  # Dict for backbone
        type='ResNet',  # Name of the backbone
        pretrained='torchvision://resnet50',  # The url/site of the pretrained model
        depth=50,  # Depth of ResNet model
        norm_eval=False),  # Whether to set BN layers to eval mode when training
    cls_head=dict(  # Dict for classification head
        type='TSNHead',  # Name of classification head
        num_classes=400,  # Number of classes to be classified.
        in_channels=2048,  # The input channels of classification head.
        spatial_type='avg',  # Type of pooling in spatial dimension
        consensus=dict(type='AvgConsensus', dim=1),  # Config of consensus module
        dropout_ratio=0.4,  # Probability in dropout layer
        init_std=0.01), # Std value for linear layer initiation
        # model training and testing settings
        train_cfg=None,  # Config of training hyperparameters for TSN
        test_cfg=dict(average_clips=None))  # Config for testing hyperparameters␣
↪for TSN.

# dataset settings
dataset_type = 'RawframeDataset'  # Type of dataset for training, validation and␣
↪testing
data_root = 'data/kinetics400/rawframes_train/'  # Root path to data for training
data_root_val = 'data/kinetics400/rawframes_val/'  # Root path to data for␣
↪validation and testing
ann_file_train = 'data/kinetics400/kinetics400_train_list_rawframes.txt'  # Path to␣
↪the annotation file for training
ann_file_val = 'data/kinetics400/kinetics400_val_list_rawframes.txt'  # Path to the␣
↪annotation file for validation
ann_file_test = 'data/kinetics400/kinetics400_val_list_rawframes.txt'  # Path to␣
↪the annotation file for testing
img_norm_cfg = dict(  # Config of image normalization used in data pipeline
    mean=[123.675, 116.28, 103.53],  # Mean values of different channels to␣
↪normalize
    std=[58.395, 57.12, 57.375],  # Std values of different channels to normalize
    to_bgr=False)  # Whether to convert channels from RGB to BGR
```

(continues on next page)

(continued from previous page)

```
train_pipeline = [  # List of training pipeline steps
    dict(  # Config of SampleFrames
        type='SampleFrames',  # Sample frames pipeline, sampling frames from video
        clip_len=1,  # Frames of each sampled output clip
        frame_interval=1,  # Temporal interval of adjacent sampled frames
        num_clips=3),  # Number of clips to be sampled
    dict(  # Config of RawFrameDecode
        type='RawFrameDecode'),  # Load and decode Frames pipeline, picking raw␣
↪frames with given indices
    dict(  # Config of Resize
        type='Resize',  # Resize pipeline
        scale=(-1, 256)),  # The scale to resize images
    dict(  # Config of MultiScaleCrop
        type='MultiScaleCrop',  # Multi scale crop pipeline, cropping images with a␣
↪list of randomly selected scales
        input_size=224,  # Input size of the network
        scales=(1, 0.875, 0.75, 0.66),  # Scales of width and height to be selected
        random_crop=False,  # Whether to randomly sample cropping bbox
        max_wh_scale_gap=1),  # Maximum gap of w and h scale levels
    dict(  # Config of Resize
        type='Resize',  # Resize pipeline
        scale=(224, 224),  # The scale to resize images
        keep_ratio=False),  # Whether to resize with changing the aspect ratio
    dict(  # Config of Flip
        type='Flip',  # Flip Pipeline
        flip_ratio=0.5),  # Probability of implementing flip
    dict(  # Config of Normalize
        type='Normalize',  # Normalize pipeline
        **img_norm_cfg),  # Config of image normalization
    dict(  # Config of FormatShape
        type='FormatShape',  # Format shape pipeline, Format final image shape to␣
↪the given input_format
        input_format='NCHW'),  # Final image shape format
    dict(  # Config of Collect
        type='Collect',  # Collect pipeline that decides which keys in the data␣
↪should be passed to the recognizer
        keys=['imgs', 'label'],  # Keys of input
        meta_keys=[]),  # Meta keys of input
    dict(  # Config of ToTensor
        type='ToTensor',  # Convert other types to tensor type pipeline
        keys=['imgs', 'label'])  # Keys to be converted from image to tensor
]
val_pipeline = [  # List of validation pipeline steps
    dict(  # Config of SampleFrames
        type='SampleFrames',  # Sample frames pipeline, sampling frames from video
        clip_len=1,  # Frames of each sampled output clip
        frame_interval=1,  # Temporal interval of adjacent sampled frames
        num_clips=3,  # Number of clips to be sampled
        test_mode=True),  # Whether to set test mode in sampling
    dict(  # Config of RawFrameDecode
        type='RawFrameDecode'),  # Load and decode Frames pipeline, picking raw␣
↪frames with given indices
```

(continues on next page)

(continued from previous page)

```
    dict(  # Config of Resize
        type='Resize',  # Resize pipeline
        scale=(-1, 256)),  # The scale to resize images
    dict(  # Config of CenterCrop
        type='CenterCrop',  # Center crop pipeline, cropping the center area from␣
↪images
        crop_size=224),  # The size to crop images
    dict(  # Config of Flip
        type='Flip',  # Flip pipeline
        flip_ratio=0),  # Probability of implementing flip
    dict(  # Config of Normalize
        type='Normalize',  # Normalize pipeline
        **img_norm_cfg),  # Config of image normalization
    dict(  # Config of FormatShape
        type='FormatShape',  # Format shape pipeline, Format final image shape to␣
↪the given input_format
        input_format='NCHW'),  # Final image shape format
    dict(  # Config of Collect
        type='Collect',  # Collect pipeline that decides which keys in the data␣
↪should be passed to the recognizer
        keys=['imgs', 'label'],  # Keys of input
        meta_keys=[]),  # Meta keys of input
    dict(  # Config of ToTensor
        type='ToTensor',  # Convert other types to tensor type pipeline
        keys=['imgs'])  # Keys to be converted from image to tensor
]
test_pipeline = [  # List of testing pipeline steps
    dict(  # Config of SampleFrames
        type='SampleFrames',  # Sample frames pipeline, sampling frames from video
        clip_len=1,  # Frames of each sampled output clip
        frame_interval=1,  # Temporal interval of adjacent sampled frames
        num_clips=25,  # Number of clips to be sampled
        test_mode=True),  # Whether to set test mode in sampling
    dict(  # Config of RawFrameDecode
        type='RawFrameDecode'),  # Load and decode Frames pipeline, picking raw␣
↪frames with given indices
    dict(  # Config of Resize
        type='Resize',  # Resize pipeline
        scale=(-1, 256)),  # The scale to resize images
    dict(  # Config of TenCrop
        type='TenCrop',  # Ten crop pipeline, cropping ten area from images
        crop_size=224),  # The size to crop images
    dict(  # Config of Flip
        type='Flip',  # Flip pipeline
        flip_ratio=0),  # Probability of implementing flip
    dict(  # Config of Normalize
        type='Normalize',  # Normalize pipeline
        **img_norm_cfg),  # Config of image normalization
    dict(  # Config of FormatShape
        type='FormatShape',  # Format shape pipeline, Format final image shape to␣
↪the given input_format
        input_format='NCHW'),  # Final image shape format
```

(continues on next page)

(continued from previous page)

```
    dict(  # Config of Collect
        type='Collect',  # Collect pipeline that decides which keys in the data␣
↪should be passed to the recognizer
        keys=['imgs', 'label'],  # Keys of input
        meta_keys=[]),  # Meta keys of input
    dict(  # Config of ToTensor
        type='ToTensor',  # Convert other types to tensor type pipeline
        keys=['imgs'])  # Keys to be converted from image to tensor
]
data = dict(  # Config of data
    videos_per_gpu=32,  # Batch size of each single GPU
    workers_per_gpu=2,  # Workers to pre-fetch data for each single GPU
    train_dataloader=dict(  # Additional config of train dataloader
        drop_last=True),  # Whether to drop out the last batch of data in training
    val_dataloader=dict(  # Additional config of validation dataloader
        videos_per_gpu=1),  # Batch size of each single GPU during evaluation
    test_dataloader=dict(  # Additional config of test dataloader
        videos_per_gpu=2),  # Batch size of each single GPU during testing
    train=dict(  # Training dataset config
        type=dataset_type,
        ann_file=ann_file_train,
        data_prefix=data_root,
        pipeline=train_pipeline),
    val=dict(  # Validation dataset config
        type=dataset_type,
        ann_file=ann_file_val,
        data_prefix=data_root_val,
        pipeline=val_pipeline),
    test=dict(  # Testing dataset config
        type=dataset_type,
        ann_file=ann_file_test,
        data_prefix=data_root_val,
        pipeline=test_pipeline))
# optimizer
optimizer = dict(
    # Config used to build optimizer, support (1). All the optimizers in PyTorch
    # whose arguments are also the same as those in PyTorch. (2). Custom optimizers
    # which are built on `constructor`, referring to "tutorials/5_new_modules.md"
    # for implementation.
    type='SGD',  # Type of optimizer, refer to https://github.com/open-mmlab/mmcv/
↪blob/master/mmcv/runner/optimizer/default_constructor.py#L13 for more details
    lr=0.01,  # Learning rate, see detail usages of the parameters in the␣
↪documentation of PyTorch
    momentum=0.9,  # Momentum,
    weight_decay=0.0001)  # Weight decay of SGD
optimizer_config = dict(  # Config used to build the optimizer hook
    grad_clip=dict(max_norm=40, norm_type=2))  # Use gradient clip
# learning policy
lr_config = dict(  # Learning rate scheduler config used to register LrUpdater hook
    policy='step',  # Policy of scheduler, also support CosineAnnealing, Cyclic,␣
↪etc. Refer to details of supported LrUpdater from https://github.com/open-mmlab/
↪mmcv/blob/master/mmcv/runner/hooks/lr_updater.py#L9
```

(continues on next page)

(continued from previous page)

```
    step=[40, 80])  # Steps to decay the learning rate
total_epochs = 100  # Total epochs to train the model
checkpoint_config = dict(  # Config to set the checkpoint hook, Refer to https://
↪github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/checkpoint.py for␣
↪implementation
    interval=5)  # Interval to save checkpoint
evaluation = dict(  # Config of evaluation during training
    interval=5,  # Interval to perform evaluation
    metrics=['top_k_accuracy', 'mean_class_accuracy'],  # Metrics to be performed
    metric_options=dict(top_k_accuracy=dict(topk=(1, 3))), # Set top-k accuracy to␣
↪1 and 3 during validation
    save_best='top1_acc')  # set `top1_acc` as key indicator to save best checkpoint
eval_config = dict(
    metric_options=dict(top_k_accuracy=dict(topk=(1, 3)))) # Set top-k accuracy to␣
↪1 and 3 during testing. You can also use `--eval top_k_accuracy` to assign␣
↪evaluation metrics
log_config = dict(  # Config to register logger hook
    interval=20,  # Interval to print the log
    hooks=[  # Hooks to be implemented during training
        dict(type='TextLoggerHook'),  # The logger used to record the training␣
↪process
        # dict(type='TensorboardLoggerHook'),  # The Tensorboard logger is also␣
↪supported
    ])

# runtime settings
dist_params = dict(backend='nccl')  # Parameters to setup distributed training, the␣
↪port can also be set
log_level = 'INFO'  # The level of logging
work_dir = './work_dirs/tsn_r50_1x1x3_100e_kinetics400_rgb/'  # Directory to save␣
↪the model checkpoints and logs for the current experiments
load_from = None  # load models as a pre-trained model from a given path. This will␣
↪not resume training
resume_from = None  # Resume checkpoints from a given path, the training will be␣
↪resumed from the epoch when the checkpoint's is saved
workflow = [('train', 1)]  # Workflow for runner. [('train', 1)] means there is only␣
↪one workflow and the workflow named 'train' is executed once
```

13.3.3 Config System for Spatio-Temporal Action Detection

We incorporate modular design into our config system, which is convenient to conduct various experiments.

- An Example of FastRCNN

  To help the users have a basic idea of a complete config structure and the modules in a spatio-temporal action detection system, we make brief comments on the config of FastRCNN as the following. For more detailed usage and alternative for per parameter in each module, please refer to the API documentation.

```
# model setting
model = dict(  # Config of the model
    type='FastRCNN',  # Type of the detector
```

(continues on next page)

(continued from previous page)

```
    backbone=dict(  # Dict for backbone
        type='ResNet3dSlowOnly',  # Name of the backbone
        depth=50, # Depth of ResNet model
        pretrained=None,   # The url/site of the pretrained model
        pretrained2d=False, # If the pretrained model is 2D
        lateral=False,  # If the backbone is with lateral connections
        num_stages=4, # Stages of ResNet model
        conv1_kernel=(1, 7, 7), # Conv1 kernel size
        conv1_stride_t=1, # Conv1 temporal stride
        pool1_stride_t=1, # Pool1 temporal stride
        spatial_strides=(1, 2, 2, 1)),  # The spatial stride for each ResNet stage
    roi_head=dict(  # Dict for roi_head
        type='AVARoIHead',  # Name of the roi_head
        bbox_roi_extractor=dict(  # Dict for bbox_roi_extractor
            type='SingleRoIExtractor3D',  # Name of the bbox_roi_extractor
            roi_layer_type='RoIAlign',  # Type of the RoI op
            output_size=8,  # Output feature size of the RoI op
            with_temporal_pool=True), # If temporal dim is pooled
        bbox_head=dict( # Dict for bbox_head
            type='BBoxHeadAVA', # Name of the bbox_head
            in_channels=2048, # Number of channels of the input feature
            num_classes=81, # Number of action classes + 1
            multilabel=True,  # If the dataset is multilabel
            dropout_ratio=0.5)),  # The dropout ratio used
    # model training and testing settings
    train_cfg=dict(  # Training config of FastRCNN
        rcnn=dict(  # Dict for rcnn training config
            assigner=dict(  # Dict for assigner
                type='MaxIoUAssignerAVA', # Name of the assigner
                pos_iou_thr=0.9,  # IoU threshold for positive examples, > pos_iou_
↪thr -> positive
                neg_iou_thr=0.9,  # IoU threshold for negative examples, < neg_iou_
↪thr -> negative
                min_pos_iou=0.9), # Minimum acceptable IoU for positive examples
            sampler=dict( # Dict for sample
                type='RandomSampler', # Name of the sampler
                num=32, # Batch Size of the sampler
                pos_fraction=1, # Positive bbox fraction of the sampler
                neg_pos_ub=-1,  # Upper bound of the ratio of num negative to num␣
↪positive
                add_gt_as_proposals=True), # Add gt bboxes as proposals
            pos_weight=1.0, # Loss weight of positive examples
            debug=False)), # Debug mode
    test_cfg=dict( # Testing config of FastRCNN
        rcnn=dict(  # Dict for rcnn testing config
            action_thr=0.002))) # The threshold of an action

# dataset settings
dataset_type = 'AVADataset' # Type of dataset for training, validation and testing
data_root = 'data/ava/rawframes'  # Root path to data
anno_root = 'data/ava/annotations'  # Root path to annotations
```

(continues on next page)

(continued from previous page)

```
ann_file_train = f'{anno_root}/ava_train_v2.1.csv'  # Path to the annotation file␣
↪for training
ann_file_val = f'{anno_root}/ava_val_v2.1.csv'  # Path to the annotation file for␣
↪validation

exclude_file_train = f'{anno_root}/ava_train_excluded_timestamps_v2.1.csv'  # Path␣
↪to the exclude annotation file for training
exclude_file_val = f'{anno_root}/ava_val_excluded_timestamps_v2.1.csv'  # Path to␣
↪the exclude annotation file for validation

label_file = f'{anno_root}/ava_action_list_v2.1_for_activitynet_2018.pbtxt'  # Path␣
↪to the label file

proposal_file_train = f'{anno_root}/ava_dense_proposals_train.FAIR.recall_93.9.pkl'␣
↪ # Path to the human detection proposals for training examples
proposal_file_val = f'{anno_root}/ava_dense_proposals_val.FAIR.recall_93.9.pkl'  #␣
↪Path to the human detection proposals for validation examples

img_norm_cfg = dict(  # Config of image normalization used in data pipeline
    mean=[123.675, 116.28, 103.53], # Mean values of different channels to normalize
    std=[58.395, 57.12, 57.375],   # Std values of different channels to normalize
    to_bgr=False) # Whether to convert channels from RGB to BGR

train_pipeline = [  # List of training pipeline steps
    dict(  # Config of SampleFrames
        type='AVASampleFrames',  # Sample frames pipeline, sampling frames from␣
↪video
        clip_len=4,  # Frames of each sampled output clip
        frame_interval=16),  # Temporal interval of adjacent sampled frames
    dict(  # Config of RawFrameDecode
        type='RawFrameDecode'),  # Load and decode Frames pipeline, picking raw␣
↪frames with given indices
    dict(  # Config of RandomRescale
        type='RandomRescale',   # Randomly rescale the shortedge by a given range
        scale_range=(256, 320)),   # The shortedge size range of RandomRescale
    dict(  # Config of RandomCrop
        type='RandomCrop',   # Randomly crop a patch with the given size
        size=256),   # The size of the cropped patch
    dict(  # Config of Flip
        type='Flip',  # Flip Pipeline
        flip_ratio=0.5),  # Probability of implementing flip
    dict(  # Config of Normalize
        type='Normalize',  # Normalize pipeline
        **img_norm_cfg),  # Config of image normalization
    dict(  # Config of FormatShape
        type='FormatShape',  # Format shape pipeline, Format final image shape to␣
↪the given input_format
        input_format='NCTHW',  # Final image shape format
        collapse=True),   # Collapse the dim N if N == 1
    dict(  # Config of Rename
        type='Rename',  # Rename keys
        mapping=dict(imgs='img')),  # The old name to new name mapping
```

(continues on next page)

(continued from previous page)

```
    dict(  # Config of ToTensor
        type='ToTensor',  # Convert other types to tensor type pipeline
        keys=['img', 'proposals', 'gt_bboxes', 'gt_labels']),  # Keys to be␣
↪converted from image to tensor
    dict(  # Config of ToDataContainer
        type='ToDataContainer',  # Convert other types to DataContainer type␣
↪pipeline
        fields=[   # Fields to convert to DataContainer
            dict(   # Dict of fields
                key=['proposals', 'gt_bboxes', 'gt_labels'],  # Keys to Convert to␣
↪DataContainer
                stack=False)]),  # Whether to stack these tensor
    dict(  # Config of Collect
        type='Collect',  # Collect pipeline that decides which keys in the data␣
↪should be passed to the detector
        keys=['img', 'proposals', 'gt_bboxes', 'gt_labels'],  # Keys of input
        meta_keys=['scores', 'entity_ids']),  # Meta keys of input
]

val_pipeline = [  # List of validation pipeline steps
    dict(  # Config of SampleFrames
        type='AVASampleFrames',  # Sample frames pipeline, sampling frames from␣
↪video
        clip_len=4,  # Frames of each sampled output clip
        frame_interval=16)  # Temporal interval of adjacent sampled frames
    dict(  # Config of RawFrameDecode
        type='RawFrameDecode'),  # Load and decode Frames pipeline, picking raw␣
↪frames with given indices
    dict(  # Config of Resize
        type='Resize',  # Resize pipeline
        scale=(-1, 256)),  # The scale to resize images
    dict(  # Config of Normalize
        type='Normalize',  # Normalize pipeline
        **img_norm_cfg),  # Config of image normalization
    dict(  # Config of FormatShape
        type='FormatShape',  # Format shape pipeline, Format final image shape to␣
↪the given input_format
        input_format='NCTHW',  # Final image shape format
        collapse=True),   # Collapse the dim N if N == 1
    dict(  # Config of Rename
        type='Rename',  # Rename keys
        mapping=dict(imgs='img')),  # The old name to new name mapping
    dict(  # Config of ToTensor
        type='ToTensor',  # Convert other types to tensor type pipeline
        keys=['img', 'proposals']),  # Keys to be converted from image to tensor
    dict(  # Config of ToDataContainer
        type='ToDataContainer',  # Convert other types to DataContainer type␣
↪pipeline
        fields=[   # Fields to convert to DataContainer
            dict(   # Dict of fields
                key=['proposals'],  # Keys to Convert to DataContainer
                stack=False)]),  # Whether to stack these tensor
```

(continues on next page)

(continued from previous page)

```
    dict(  # Config of Collect
        type='Collect',  # Collect pipeline that decides which keys in the data␣
↪should be passed to the detector
        keys=['img', 'proposals'],  # Keys of input
        meta_keys=['scores', 'entity_ids'],  # Meta keys of input
        nested=True)  # Whether to wrap the data in a nested list
]

data = dict(  # Config of data
    videos_per_gpu=16,  # Batch size of each single GPU
    workers_per_gpu=2,  # Workers to pre-fetch data for each single GPU
    val_dataloader=dict(   # Additional config of validation dataloader
        videos_per_gpu=1),  # Batch size of each single GPU during evaluation
    train=dict(   # Training dataset config
        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(     # Validation dataset config
        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']    # Set test_dataset as val_dataset

# optimizer
optimizer = dict(
    # Config used to build optimizer, support (1). All the optimizers in PyTorch
    # whose arguments are also the same as those in PyTorch. (2). Custom optimizers
    # which are built on `constructor`, referring to "tutorials/5_new_modules.md"
    # for implementation.
    type='SGD',  # Type of optimizer, refer to https://github.com/open-mmlab/mmcv/
↪blob/master/mmcv/runner/optimizer/default_constructor.py#L13 for more details
    lr=0.2,  # Learning rate, see detail usages of the parameters in the␣
↪documentation of PyTorch (for 8gpu)
    momentum=0.9,  # Momentum,
    weight_decay=0.00001)  # Weight decay of SGD

optimizer_config = dict(  # Config used to build the optimizer hook
    grad_clip=dict(max_norm=40, norm_type=2))   # Use gradient clip

lr_config = dict(  # Learning rate scheduler config used to register LrUpdater hook
    policy='step',  # Policy of scheduler, also support CosineAnnealing, Cyclic,␣
↪etc. Refer to details of supported LrUpdater from https://github.com/open-mmlab/
↪mmcv/blob/master/mmcv/runner/hooks/lr_updater.py#L9
```

(continues on next page)

(continued from previous page)

```
    step=[40, 80],  # Steps to decay the learning rate
    warmup='linear',  # Warmup strategy
    warmup_by_epoch=True,  # Warmup_iters indicates iter num or epoch num
    warmup_iters=5,  # Number of iters or epochs for warmup
    warmup_ratio=0.1)  # The initial learning rate is warmup_ratio * lr

total_epochs = 20  # Total epochs to train the model
checkpoint_config = dict(  # Config to set the checkpoint hook, Refer to https://
↪github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/checkpoint.py for␣
↪implementation
    interval=1)  # Interval to save checkpoint
workflow = [('train', 1)]  # Workflow for runner. [('train', 1)] means there is only␣
↪one workflow and the workflow named 'train' is executed once
evaluation = dict(  # Config of evaluation during training
    interval=1, save_best='mAP@0.5IOU')  # Interval to perform evaluation and the␣
↪key for saving best checkpoint
log_config = dict(  # Config to register logger hook
    interval=20,  # Interval to print the log
    hooks=[  # Hooks to be implemented during training
        dict(type='TextLoggerHook'),  # The logger used to record the training␣
↪process
    ])

# runtime settings
dist_params = dict(backend='nccl')  # Parameters to setup distributed training, the␣
↪port can also be set
log_level = 'INFO'  # The level of logging
work_dir = ('./work_dirs/ava/'  # Directory to save the model checkpoints and logs␣
↪for the current experiments
            'slowonly_kinetics_pretrained_r50_4x16x1_20e_ava_rgb')
load_from = ('https://download.openmmlab.com/mmaction/recognition/slowonly/'  #␣
↪load models as a pre-trained model from a given path. This will not resume␣
↪training
             'slowonly_r50_4x16x1_256e_kinetics400_rgb/'
             'slowonly_r50_4x16x1_256e_kinetics400_rgb_20200704-a69556c6.pth')
resume_from = None  # Resume checkpoints from a given path, the training will be␣
↪resumed from the epoch when the checkpoint's is saved
```

13.4 FAQ

13.4.1 Use intermediate variables in configs

Some intermediate variables are used in the config files, like `train_pipeline/val_pipeline/test_pipeline`, `ann_file_train/ann_file_val/ann_file_test`, `img_norm_cfg` etc.

For Example, we would like to first define `train_pipeline/val_pipeline/test_pipeline` and pass them into `data`. Thus, `train_pipeline/val_pipeline/test_pipeline` are intermediate variable.

we also define `ann_file_train/ann_file_val/ann_file_test` and `data_root/data_root_val` to provide data pipeline some basic information.

In addition, we use `img_norm_cfg` as intermediate variables to construct data augmentation components.

```
...
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 = [
    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),
```

(continues on next page)

(continued from previous page)

```
    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

FOURTEEN

TUTORIAL 2: FINETUNING MODELS

This tutorial provides instructions for users to use the pre-trained models to finetune them on other datasets, so that better performance can be achieved.

14.1 Outline

There are two steps to finetune a model on a new dataset.

1. Add support for the new dataset. See *Tutorial 3: Adding New Dataset*.
2. Modify the configs. This will be discussed in this tutorial.

For example, if the users want to finetune models pre-trained on Kinetics-400 Dataset to another dataset, say UCF101, then four parts in the config (see *here*) needs attention.

14.2 Modify Head

The `num_classes` in the `cls_head` need to be changed to the class number of the new dataset. The weights of the pre-trained models are reused except for the final prediction layer. So it is safe to change the class number. In our case, UCF101 has 101 classes. So we change it from 400 (class number of Kinetics-400) to 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,   # change from 400 to 101
```

(continues on next page)

(continued from previous page)

```
        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))
```

Note that the `pretrained='torchvision://resnet50'` setting is used for initializing backbone. If you are training a new model from ImageNet-pretrained weights, this is for you. However, this setting is not related to our task at hand. What we need is `load_from`, which will be discussed later.

14.3 Modify Dataset

MMAction2 supports UCF101, Kinetics-400, Moments in Time, Multi-Moments in Time, THUMOS14, Something-Something V1&V2, ActivityNet Dataset. The users may need to adapt one of the above dataset to fit for their special datasets. In our case, UCF101 is already supported by various dataset types, like `RawframeDataset`, so we change the config as follows.

```
# dataset settings
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 Modify Training Schedule

Finetuning usually requires smaller learning rate and less training epochs.

```
# optimizer
optimizer = dict(type='SGD', lr=0.005, momentum=0.9, weight_decay=0.0001)  # change from␣
↪0.01 to 0.005
optimizer_config = dict(grad_clip=dict(max_norm=40, norm_type=2))
# learning policy
lr_config = dict(policy='step', step=[20, 40])
total_epochs = 50 # change from 100 to 50
checkpoint_config = dict(interval=5)
```

14.5 Use Pre-Trained Model

To use the pre-trained model for the whole network, the new config adds the link of pre-trained models in the `load_from`. We set `load_from=None` as default in `configs/_base_/default_runtime.py` and owing to [inheritance design](tutorials/1_config.md), users can directly change it by setting `load_from` in their configs.

```
# use the pre-trained model for the whole TSN network
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'  # model path can be found in model zoo
```

CHAPTER

FIFTEEN

TUTORIAL 3: ADDING NEW DATASET

In this tutorial, we will introduce some methods about how to customize your own dataset by reorganizing data and mixing dataset for the project.

15.1 Customize Datasets by Reorganizing Data

15.1.1 Reorganize datasets to existing format

The simplest way is to convert your dataset to existing dataset formats (RawframeDataset or VideoDataset).

There are three kinds of annotation files.

- rawframe annotation

  The annotation of a rawframe dataset is a text file with multiple lines, and each line indicates `frame_directory` (relative path) of a video, `total_frames` of a video and the `label` of a video, which are split by a whitespace.

  Here is an example.

```
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
```

- video annotation

  The annotation of a video dataset is a text file with multiple lines, and each line indicates a sample video with the `filepath` (relative path) and `label`, which are split by a whitespace.

  Here is an example.

```
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 annotation

  The annotation of ActivityNet dataset is a json file. Each key is a video name and the corresponding value is the meta data and annotation for the video.

  Here is an example.

```
{
  "video1": {
      "duration_second": 211.53,
      "duration_frame": 6337,
      "annotations": [
          {
              "segment": [
                  30.025882995319815,
                  205.2318595943838
              ],
              "label": "Rock climbing"
          }
      ],
      "feature_frame": 6336,
      "fps": 30.0,
      "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
  }
}
```

There are two ways to work with custom datasets.

- online conversion

  You can write a new Dataset class inherited from BaseDataset, and overwrite three methods `load_annotations(self)`, `evaluate(self, results, metrics, logger)` and `dump_results(self, results, out)`, like RawframeDataset, VideoDataset or ActivityNetDataset.

- offline conversion

  You can convert the annotation format to the expected format above and save it to a pickle or json file, then you can simply use `RawframeDataset`, `VideoDataset` or `ActivityNetDataset`.

After the data pre-processing, the users need to further modify the config files to use the dataset. Here is an example of using a custom dataset in rawframe format.

In `configs/task/method/my_custom_config.py`:

```
...
# dataset settings
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'
...
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,
        ...))
...
```

We use this way to support Rawframe dataset.

15.1.2 An example of a custom dataset

Assume the annotation is in a new format in text files, and the image file name is of template like `img_00005.jpg` The video annotations are stored in text file `annotation.txt` as following

```
directory,total frames,class
D32_1gwq35E,299,66
-G-5CJ0JkKY,249,254
T4h1bvOd9DA,299,33
4uZ27ivBl00,299,341
0LfESFkfBSw,249,186
-YIsNpBEx6c,299,169
```

We can create a new dataset in `mmaction/datasets/my_dataset.py` to load the data.

```
import copy
import os.path as osp

import mmcv

from .base import BaseDataset
from .builder import DATASETS


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

    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):
```

(continues on next page)

(continued from previous page)

```
        pass
```

Then in the config, to use `MyDataset` you can modify the config as the following

```
dataset_A_train = dict(
    type='MyDataset',
    ann_file=ann_file_train,
    pipeline=train_pipeline
)
```

15.2 Customize Dataset by Mixing Dataset

MMAction2 also supports to mix dataset for training. Currently it supports to repeat dataset.

15.2.1 Repeat dataset

We use `RepeatDataset` as wrapper to repeat the dataset. For example, suppose the original dataset as `Dataset_A`, to repeat it, the config looks like the following

```
dataset_A_train = dict(
        type='RepeatDataset',
        times=N,
        dataset=dict(  # This is the original config of Dataset_A
            type='Dataset_A',
            ...
            pipeline=train_pipeline
        )
    )
```

CHAPTER

SIXTEEN

TUTORIAL 4: CUSTOMIZE DATA PIPELINES

In this tutorial, we will introduce some methods about the design of data pipelines, and how to customize and extend your own data pipelines for the project.

16.1 Design of Data Pipelines

Following typical conventions, we use `Dataset` and `DataLoader` for data loading with multiple workers. `Dataset` returns a dict of data items corresponding the arguments of models' forward method. Since the data in action recognition & localization may not be the same size (image size, gt bbox size, etc.), The `DataContainer` in MMCV is used to help collect and distribute data of different sizes. See here for more details.

The data preparation pipeline and the dataset is decomposed. Usually a dataset defines how to process the annotations and a data pipeline defines all the steps to prepare a data dict. A pipeline consists of a sequence of operations. Each operation takes a dict as input and also output a dict for the next operation.

We present a typical pipeline in the following figure. The blue blocks are pipeline operations. With the pipeline going on, each operator can add new keys (marked as green) to the result dict or update the existing keys (marked as orange).

The operations are categorized into data loading, pre-processing and formatting.

Here is a pipeline example for TSN.

```
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,
```

(continues on next page)

(continued from previous page)

```
        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',
        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'])
]
```

We have supported some lazy operators and encourage users to apply them. Lazy ops record how the data should be processed, but it will postpone the processing on the raw data until the raw data forward `Fuse` stage. Specifically, lazy ops avoid frequent reading and modification operation on the raw data, but process the raw data once in the final Fuse stage, thus accelerating data preprocessing.

Here is a pipeline example applying lazy ops.

```
train_pipeline = [
    dict(type='SampleFrames', clip_len=32, frame_interval=2, num_clips=1),
    dict(type='RawFrameDecode', decoding_backend='turbojpeg'),
    # The following three lazy ops only process the bbox of frames without
    # modifying the raw data.
    dict(type='Resize', scale=(-1, 256), lazy=True),
    dict(
```

(continues on next page)

(continued from previous page)

```
        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 operator `Flip` only record whether a frame should be fliped and the
    # flip direction.
    dict(type='Flip', flip_ratio=0.5, lazy=True),
    # Processing the raw data once in Fuse stage.
    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'])
]
```

For each operation, we list the related dict fields that are added/updated/removed below, where * means the key may not be affected.

16.1.1 Data loading

`SampleFrames`

- add: frame_inds, clip_len, frame_interval, num_clips, *total_frames

`DenseSampleFrames`

- add: frame_inds, clip_len, frame_interval, num_clips, *total_frames

`PyAVDecode`

- add: imgs, original_shape
- update: *frame_inds

`DecordDecode`

- add: imgs, original_shape
- update: *frame_inds

`OpenCVDecode`

- add: imgs, original_shape
- update: *frame_inds

`RawFrameDecode`

- add: imgs, original_shape
- update: *frame_inds

16.1.2 Pre-processing

`RandomCrop`

- add: crop_bbox, img_shape
- update: imgs

`RandomResizedCrop`

- add: crop_bbox, img_shape
- update: imgs

`MultiScaleCrop`

- add: crop_bbox, img_shape, scales
- update: imgs

`Resize`

- add: img_shape, keep_ratio, scale_factor
- update: imgs

`Flip`

- add: flip, flip_direction
- update: imgs, label

`Normalize`

- add: img_norm_cfg
- update: imgs

`CenterCrop`

- add: crop_bbox, img_shape
- update: imgs

`ThreeCrop`

- add: crop_bbox, img_shape
- update: imgs

`TenCrop`

- add: crop_bbox, img_shape
- update: imgs

16.1.3 Formatting

`ToTensor`

- update: specified by `keys`.

`ImageToTensor`

- update: specified by `keys`.

`Transpose`

- update: specified by `keys`.

`Collect`

- add: img_metas (the keys of img_metas is specified by `meta_keys`)
- remove: all other keys except for those specified by `keys`

It is **noteworthy** that the first key, commonly `imgs`, will be used as the main key to calculate the batch size.

`FormatShape`

- add: input_shape
- update: imgs

16.2 Extend and Use Custom Pipelines

1. Write a new pipeline in any file, e.g., `my_pipeline.py`. It takes a dict as input and return a dict.

```
from mmaction.datasets import PIPELINES

@PIPELINES.register_module()
class MyTransform:

    def __call__(self, results):
        results['key'] = value
        return results
```

2. Import the new class.

```
from .my_pipeline import MyTransform
```

3. Use it in config files.

```
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'),       # use a custom pipeline
    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

SEVENTEEN

TUTORIAL 5: ADDING NEW MODULES

In this tutorial, we will introduce some methods about how to customize optimizer, develop new components and new a learning rate scheduler for this project.

17.1 Customize Optimizer

An example of customized optimizer is CopyOfSGD is defined in `mmaction/core/optimizer/copy_of_sgd.py`. More generally, a customized optimizer could be defined as following.

Assume you want to add an optimizer named as `MyOptimizer`, which has arguments `a`, `b` and `c`. You need to first implement the new optimizer in a file, e.g., in `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):
```

Then add this module in `mmaction/core/optimizer/__init__.py`, thus the registry will find the new module and add it:

```
from .my_optimizer import MyOptimizer
```

Then you can use `MyOptimizer` in `optimizer` field of config files. In the configs, the optimizers are defined by the field `optimizer` like the following:

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

To use your own optimizer, the field can be changed as

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

We already support to use all the optimizers implemented by PyTorch, and the only modification is to change the `optimizer` field of config files. For example, if you want to use `ADAM`, though the performance will drop a lot, the modification could be as the following.

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

The users can directly set arguments following the API doc of PyTorch.

17.2 Customize Optimizer Constructor

Some models may have some parameter-specific settings for optimization, e.g. weight decay for BatchNorm layers. The users can do those fine-grained parameter tuning through customizing optimizer constructor.

You can write a new optimizer constructor inherit from DefaultOptimizerConstructor and overwrite the `add_params(self, params, module)` method.

An example of customized optimizer constructor is TSMOptimizerConstructor. More generally, a customized optimizer constructor could be defined as following.

In `mmaction/core/optimizer/my_optimizer_constructor.py`:

```
from mmcv.runner import OPTIMIZER_BUILDERS, DefaultOptimizerConstructor

@OPTIMIZER_BUILDERS.register_module()
class MyOptimizerConstructor(DefaultOptimizerConstructor):

```

In `mmaction/core/optimizer/__init__.py`:

```
from .my_optimizer_constructor import MyOptimizerConstructor
```

Then you can use `MyOptimizerConstructor` in `optimizer` field of config files.

```
# optimizer
optimizer = dict(
    type='SGD',
    constructor='MyOptimizerConstructor',
    paramwise_cfg=dict(fc_lr5=True),
    lr=0.02,
    momentum=0.9,
    weight_decay=0.0001)
```

17.3 Develop New Components

We basically categorize model components into 4 types.

- recognizer: the whole recognizer model pipeline, usually contains a backbone and cls_head.
- backbone: usually an FCN network to extract feature maps, e.g., ResNet, BNInception.
- cls_head: the component for classification task, usually contains an FC layer with some pooling layers.
- localizer: the model for temporal localization task, currently available: BSN, BMN, SSN.

17.3.1 Add new backbones

Here we show how to develop new components with an example of TSN.

1. Create a new file `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):  # should return a tuple
        pass

    def init_weights(self, pretrained=None):
        pass
```

2. Import the module in `mmaction/models/backbones/__init__.py`.

```
from .resnet import ResNet
```

3. Use it in your config file.

```
model = dict(
    ...
    backbone=dict(
        type='ResNet',
        arg1=xxx,
        arg2=xxx),
)
```

17.3.2 Add new heads

Here we show how to develop a new head with the example of TSNHead as the following.

1. Create a new file `mmaction/models/heads/tsn_head.py`.

   You can write a new classification head inheriting from BaseHead, and overwrite `init_weights(self)` and `forward(self, x)` method.

```
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. Import the module in `mmaction/models/heads/__init__.py`

```
from .tsn_head import TSNHead
```

3. Use it in your config file

```
model = dict(
    ...
    cls_head=dict(
        type='TSNHead',
        num_classes=400,
        in_channels=2048,
        arg1=xxx,
        arg2=xxx),
```

17.3.3 Add new loss

Assume you want to add a new loss as `MyLoss`. To add a new loss function, the users need implement it in `mmaction/models/losses/my_loss.py`.

```
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
```

(continues on next page)

(continued from previous page)

```
@LOSSES.register_module()
class MyLoss(nn.Module):

    def forward(self, pred, target):
        loss = my_loss(pred, target)
        return loss
```

Then the users need to add it in the `mmaction/models/losses/__init__.py`

```
from .my_loss import MyLoss, my_loss
```

To use it, modify the `loss_xxx` field. Since MyLoss is for regression, we can use it for the bbox loss `loss_bbox`.

```
loss_bbox=dict(type='MyLoss'))
```

17.4 Add new learning rate scheduler (updater)

The default manner of constructing a lr updater(namely, 'scheduler' by pytorch convention), is to modify the config such as:

```
...
lr_config = dict(policy='step', step=[20, 40])
...
```

In the api for `train.py`, it will register the learning rate updater hook based on the config at:

```
...
    runner.register_training_hooks(
        cfg.lr_config,
        optimizer_config,
        cfg.checkpoint_config,
        cfg.log_config,
        cfg.get('momentum_config', None))
...
```

So far, the supported updaters can be find in mmcv, but if you want to customize a new learning rate updater, you may follow the steps below:

1. First, write your own LrUpdaterHook in `$MMAction2/mmaction/core/scheduler`. The snippet followed is an example of customized lr updater that uses learning rate based on a specific learning rate ratio: `lrs`, by which the learning rate decreases at each `steps`:

```
@HOOKS.register_module()
# Register it here
class RelativeStepLrUpdaterHook(LrUpdaterHook):
    # You should inheritate it from mmcv.LrUpdaterHook
    def __init__(self, steps, lrs, **kwargs):
        super().__init__(**kwargs)
        assert len(steps) == (len(lrs))
```

(continues on next page)

(continued from previous page)

```
        self.steps = steps
        self.lrs = lrs

    def get_lr(self, runner, base_lr):
        # Only this function is required to override
        # This function is called before each training epoch, return the specific␣
↪learning rate here.
        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. Modify your config:

In your config file, swap the original `lr_config` by:

```
lr_config = dict(policy='RelativeStep', steps=[20, 40, 60], lrs=[0.1, 0.01, 0.001])
```

More examples can be found in mmcv.

CHAPTER

EIGHTEEN

TUTORIAL 6: EXPORTING A MODEL TO ONNX

Open Neural Network Exchange (ONNX) is an open ecosystem that empowers AI developers to choose the right tools as their project evolves.

18.1 Supported Models

So far, our codebase supports onnx exporting from pytorch models trained with MMAction2. The supported models are:

- I3D
- TSN
- TIN
- TSM
- R(2+1)D
- SLOWFAST
- SLOWONLY
- BMN
- BSN(tem, pem)

18.2 Usage

For simple exporting, you can use the script here. Note that the package `onnx` and `onnxruntime` are required for verification after exporting.

18.2.1 Prerequisite

First, install onnx.

```
pip install onnx onnxruntime
```

We provide a python script to export the pytorch model trained by MMAction2 to ONNX.

```
python tools/deployment/pytorch2onnx.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--shape $
↪{SHAPE}] \
    [--verify] [--show] [--output-file ${OUTPUT_FILE}]  [--is-localizer] [--opset-
↪version ${VERSION}]
```

Optional arguments:

- `--shape`: The shape of input tensor to the model. For 2D recognizer(e.g. TSN), the input should be `$batch $clip $channel $height $width`(e.g. `1 1 3 224 224`); For 3D recognizer(e.g. I3D), the input should be `$batch $clip $channel $time $height $width`(e.g. `1 1 3 32 224 224`); For localizer such as BSN, the input for each module is different, please check the `forward` function for it. If not specified, it will be set to `1 1 3 224 224`.
- `--verify`: Determines whether to verify the exported model, runnably and numerically. If not specified, it will be set to `False`.
- `--show`: Determines whether to print the architecture of the exported model. If not specified, it will be set to `False`.
- `--output-file`: The output onnx model name. If not specified, it will be set to `tmp.onnx`.
- `--is-localizer`: Determines whether the model to be exported is a localizer. If not specified, it will be set to `False`.
- `--opset-version`: Determines the operation set version of onnx, we recommend you to use a higher version such as 11 for compatibility. If not specified, it will be set to 11.
- `--softmax`: Determines whether to add a softmax layer at the end of recognizers. If not specified, it will be set to `False`. For now, localizers are not supported.

18.2.2 Recognizers

For recognizers, please run:

```
python tools/deployment/pytorch2onnx.py $CONFIG_PATH $CHECKPOINT_PATH --shape $SHAPE --
↪verify
```

18.2.3 Localizers

For localizers, please run:

```
python tools/deployment/pytorch2onnx.py $CONFIG_PATH $CHECKPOINT_PATH --is-localizer --
↪shape $SHAPE --verify
```

Please fire an issue if you discover any checkpoints that are not perfectly exported or suffer some loss in accuracy.

CHAPTER

NINETEEN

TUTORIAL 7: CUSTOMIZE RUNTIME SETTINGS

In this tutorial, we will introduce some methods about how to customize optimization methods, training schedules, workflow and hooks when running your own settings for the project.

19.1 Customize Optimization Methods

19.1.1 Customize optimizer supported by PyTorch

We already support to use all the optimizers implemented by PyTorch, and the only modification is to change the `optimizer` field of config files. For example, if you want to use `Adam`, the modification could be as the following.

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

To modify the learning rate of the model, the users only need to modify the `lr` in the config of optimizer. The users can directly set arguments following the API doc of PyTorch.

For example, if you want to use `Adam` with the setting like `torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsgrad=False)` in PyTorch, the modification could be set as the following.

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

19.1.2 Customize self-implemented optimizer

1. Define a new optimizer

A customized optimizer could be defined as following.

Assume you want to add an optimizer named `MyOptimizer`, which has arguments `a`, `b`, and `c`. You need to create a new directory named `mmaction/core/optimizer`. And then implement the new optimizer in a file, e.g., in `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):
```

2. Add the optimizer to registry

To find the above module defined above, this module should be imported into the main namespace at first. There are two ways to achieve it.

- Modify `mmaction/core/optimizer/__init__.py` to import it.

  The newly defined module should be imported in `mmaction/core/optimizer/__init__.py` so that the registry will find the new module and add it:

```
from .my_optimizer import MyOptimizer
```

- Use `custom_imports` in the config to manually import it

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

The module `mmaction.core.optimizer.my_optimizer` will be imported at the beginning of the program and the class `MyOptimizer` is then automatically registered. Note that only the package containing the class `MyOptimizer` should be imported. `mmaction.core.optimizer.my_optimizer.MyOptimizer` **cannot** be imported directly.

3. Specify the optimizer in the config file

Then you can use `MyOptimizer` in `optimizer` field of config files. In the configs, the optimizers are defined by the field `optimizer` like the following:

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

To use your own optimizer, the field can be changed to

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

19.1.3 Customize optimizer constructor

Some models may have some parameter-specific settings for optimization, e.g. weight decay for BatchNorm layers. The users can do those fine-grained parameter tuning through customizing optimizer constructor.

```
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
```

The default optimizer constructor is implemented here, which could also serve as a template for new optimizer constructor.

19.1.4 Additional settings

Tricks not implemented by the optimizer should be implemented through optimizer constructor (e.g., set parameter-wise learning rates) or hooks. We list some common settings that could stabilize the training or accelerate the training. Feel free to create PR, issue for more settings.

- **Use gradient clip to stabilize training**: Some models need gradient clip to clip the gradients to stabilize the training process. An example is as below:

  ```
  optimizer_config = dict(grad_clip=dict(max_norm=35, norm_type=2))
  ```

- **Use momentum schedule to accelerate model convergence**: We support momentum scheduler to modify model's momentum according to learning rate, which could make the model converge in a faster way. Momentum scheduler is usually used with LR scheduler, for example, the following config is used in 3D detection to accelerate convergence. For more details, please refer to the implementation of CyclicLrUpdater and 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,
    step_ratio_up=0.4,
)
```

19.2 Customize Training Schedules

we use step learning rate with default value in config files, this calls StepLRHook in MMCV. We support many other learning rate schedule here, such as `CosineAnnealing` and `Poly` schedule. Here are some examples

- Poly schedule:

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

- ConsineAnnealing schedule:

```
lr_config = dict(
    policy='CosineAnnealing',
    warmup='linear',
    warmup_iters=1000,
    warmup_ratio=1.0 / 10,
    min_lr_ratio=1e-5)
```

19.3 Customize Workflow

By default, we recommend users to use `EvalHook` to do evaluation after training epoch, but they can still use `val` workflow as an alternative.

Workflow is a list of (phase, epochs) to specify the running order and epochs. By default it is set to be

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

which means running 1 epoch for training. Sometimes user may want to check some metrics (e.g. loss, accuracy) about the model on the validate set. In such case, we can set the workflow as

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

so that 1 epoch for training and 1 epoch for validation will be run iteratively.

Note:

1. The parameters of model will not be updated during val epoch.
2. Keyword `total_epochs` in the config only controls the number of training epochs and will not affect the validation workflow.
3. Workflows `[('train', 1), ('val', 1)]` and `[('train', 1)]` will not change the behavior of `EvalHook` because `EvalHook` is called by `after_train_epoch` and validation workflow only affect hooks that are called through `after_val_epoch`. Therefore, the only difference between `[('train', 1), ('val', 1)]` and `[('train', 1)]` is that the runner will calculate losses on validation set after each training epoch.

19.4 Customize Hooks

19.4.1 Customize self-implemented hooks

1. Implement a new hook

Here we give an example of creating a new hook in MMAction2 and using it in training.

```python
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
```

Depending on the functionality of the hook, the users need to specify what the hook will do at each stage of the training in `before_run`, `after_run`, `before_epoch`, `after_epoch`, `before_iter`, and `after_iter`.

2. Register the new hook

Then we need to make `MyHook` imported. Assuming the file is in `mmaction/core/utils/my_hook.py` there are two ways to do that:

- Modify `mmaction/core/utils/__init__.py` to import it.

  The newly defined module should be imported in `mmaction/core/utils/__init__.py` so that the registry will find the new module and add it:

```
from .my_hook import MyHook
```

- Use `custom_imports` in the config to manually import it

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

3. Modify the config

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

You can also set the priority of the hook by adding key `priority` to `'NORMAL'` or `'HIGHEST'` as below

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

By default the hook's priority is set as `NORMAL` during registration.

19.4.2 Use hooks implemented in MMCV

If the hook is already implemented in MMCV, you can directly modify the config to use the hook as below

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

19.4.3 Modify default runtime hooks

There are some common hooks that are not registered through `custom_hooks` but has been registered by default when importing MMCV, they are

- log_config
- checkpoint_config
- evaluation
- lr_config
- optimizer_config
- momentum_config

In those hooks, only the logger hook has the `VERY_LOW` priority, others' priority are `NORMAL`. The above-mentioned tutorials already cover how to modify `optimizer_config`, `momentum_config`, and `lr_config`. Here we reveals how what we can do with `log_config`, `checkpoint_config`, and `evaluation`.

Checkpoint config

The MMCV runner will use `checkpoint_config` to initialize `CheckpointHook`.

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

The users could set `max_keep_ckpts` to only save only small number of checkpoints or decide whether to store state dict of optimizer by `save_optimizer`. More details of the arguments are here

Log config

The `log_config` wraps multiple logger hooks and enables to set intervals. Now MMCV supports `WandbLoggerHook`, `MlflowLoggerHook`, and `TensorboardLoggerHook`. The detail usages can be found in the doc.

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

Evaluation config

The config of `evaluation` will be used to initialize the `EvalHook`. Except the key `interval`, other arguments such as `metrics` will be passed to the `dataset.evaluate()`

```
evaluation = dict(interval=1, metrics='bbox')
```

Apart from training/testing scripts, We provide lots of useful tools under the `tools/` directory.

CHAPTER

TWENTY

USEFUL TOOLS LINK

CHAPTER

TWENTYONE

LOG ANALYSIS

`tools/analysis/analyze_logs.py` plots loss/top-k acc curves given a training log file. Run `pip install seaborn` first to install the dependency.

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

Examples:

- Plot the classification loss of some run.

```
python tools/analysis/analyze_logs.py plot_curve log.json --keys loss_cls --legend loss_cls
```

- Plot the top-1 acc and top-5 acc of some run, and save the figure to a pdf.

```
python tools/analysis/analyze_logs.py plot_curve log.json --keys top1_acc top5_acc --out results.pdf
```

- Compare the top-1 acc of two runs in the same figure.

```
python tools/analysis/analyze_logs.py plot_curve log1.json log2.json --keys top1_acc --legend run1 run2
```

  You can also compute the average training speed.

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

- Compute the average training speed for a config file.

```
python tools/analysis/analyze_logs.py cal_train_time work_dirs/some_exp/20200422_153324.log.json
```

  The output is expected to be like the following.

```
-----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

TWENTYTWO

MODEL COMPLEXITY

`/tools/analysis/get_flops.py` is a script adapted from flops-counter.pytorch to compute the FLOPs and params of a given model.

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

We will get the result like this

```
==============================
Input shape: (1, 3, 32, 340, 256)
Flops: 37.1 GMac
Params: 28.04 M
==============================
```

Note: This tool is still experimental and we do not guarantee that the number is absolutely correct. You may use the result for simple comparisons, but double check it before you adopt it in technical reports or papers.

(1) FLOPs are related to the input shape while parameters are not. The default input shape is (1, 3, 340, 256) for 2D recognizer, (1, 3, 32, 340, 256) for 3D recognizer. (2) Some operators are not counted into FLOPs like GN and custom operators. Refer to `mmcv.cnn.get_model_complexity_info()` for details.

CHAPTER

TWENTYTHREE

MODEL CONVERSION

23.1 MMAction2 model to ONNX (experimental)

`/tools/deployment/pytorch2onnx.py` is a script to convert model to ONNX format. It also supports comparing the output results between Pytorch and ONNX model for verification. Run `pip install onnx onnxruntime` first to install the dependency. Please note that a softmax layer could be added for recognizers by `--softmax` option, in order to get predictions in range `[0, 1]`.

- For recognizers, please run:

```
python tools/deployment/pytorch2onnx.py $CONFIG_PATH $CHECKPOINT_PATH --shape
↪$SHAPE --verify
```

- For localizers, please run:

```
python tools/deployment/pytorch2onnx.py $CONFIG_PATH $CHECKPOINT_PATH --is-
↪localizer --shape $SHAPE --verify
```

23.2 Prepare a model for publishing

`tools/deployment/publish_model.py` helps users to prepare their model for publishing.

Before you upload a model to AWS, you may want to:

(1) convert model weights to CPU tensors. (2) delete the optimizer states. (3) compute the hash of the checkpoint file and append the hash id to the filename.

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

E.g.,

```
python tools/deployment/publish_model.py work_dirs/tsn_r50_1x1x3_100e_kinetics400_rgb/
↪latest.pth tsn_r50_1x1x3_100e_kinetics400_rgb.pth
```

The final output filename will be `tsn_r50_1x1x3_100e_kinetics400_rgb-{hash id}.pth`.

CHAPTER

TWENTYFOUR

MODEL SERVING

In order to serve an `MMAction2` model with `TorchServe`, you can follow the steps:

24.1 1. Convert model from MMAction2 to TorchServe

```
python tools/deployment/mmaction2torchserve.py ${CONFIG_FILE} ${CHECKPOINT_FILE} \
--output_folder ${MODEL_STORE} \
--model-name ${MODEL_NAME} \
--label-file ${LABLE_FILE}
```

24.2 2. Build `mmaction-serve` docker image

```
DOCKER_BUILDKIT=1 docker build -t mmaction-serve:latest docker/serve/
```

24.3 3. Launch `mmaction-serve`

Check the official docs for running TorchServe with docker.

Example:

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

Note: ${MODEL_STORE} needs to be an absolute path. Read the docs about the Inference (8080), Management (8081) and Metrics (8082) APis

24.4 4. Test deployment

```
# Assume you are under the directory `mmaction2`
curl http://127.0.0.1:8080/predictions/${MODEL_NAME} -T demo/demo.mp4
```

You should obtain a response similar to:

```
{
  "arm wrestling": 1.0,
  "rock scissors paper": 4.962051880497143e-10,
  "shaking hands": 3.9761663406245873e-10,
  "massaging feet": 1.1924419784925533e-10,
  "stretching leg": 1.0601879096849842e-10
}
```

CHAPTER

TWENTYFIVE

MISCELLANEOUS

25.1 Evaluating a metric

`tools/analysis/eval_metric.py` evaluates certain metrics of the results saved in a file according to a config file.

The saved result file is created on `tools/test.py` by setting the arguments `--out ${RESULT_FILE}` to indicate the result file, which stores the final output of the whole model.

```
python tools/analysis/eval_metric.py ${CONFIG_FILE} ${RESULT_FILE} [--eval ${EVAL_
↪METRICS}] [--cfg-options ${CFG_OPTIONS}] [--eval-options ${EVAL_OPTIONS}]
```

25.2 Print the entire config

`tools/analysis/print_config.py` prints the whole config verbatim, expanding all its imports.

```
python tools/analysis/print_config.py ${CONFIG} [-h] [--options ${OPTIONS [OPTIONS...]}]
```

25.3 Check videos

`tools/analysis/check_videos.py` uses specified video encoder to iterate all samples that are specified by the input configuration file, looks for invalid videos (corrupted or missing), and saves the corresponding file path to the output file. Please note that after deleting invalid videos, users need to regenerate the video file list.

```
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

TWENTYSIX

CHANGELOG

26.1 0.24.0 (05/05/2022)

Highlights

- Support different seeds

New Features

- Add lateral norm in multigrid config (#1567)
- Add openpose 25 joints in graph config (#1578)
- Support MLU Backend (#1608)

Bug and Typo Fixes

- Fix local_rank (#1558)
- Fix install typo (#1571)
- Fix the inference API doc (#1580)
- Fix zh-CN demo.md and getting_started.md (#1587)
- Remove Recommonmark (#1595)
- Fix inference with ndarray (#1603)
- Fix the log error when `IterBasedRunner` is used (#1606)

26.2 0.23.0 (04/01/2022)

Highlights

- Support different seeds
- Provide multi-node training & testing script
- Update error log

New Features

- Support different seeds(#1502)
- Provide multi-node training & testing script(#1521)
- Update error log(#1546)

Documentations

- Update gpus in Slowfast readme(#1497)
- Fix work_dir in multigrid config(#1498)
- Add sub bn docs(#1503)
- Add shortcycle sampler docs(#1513)
- Update Windows Declaration(#1520)
- Update the link for ST-GCN(#1544)
- Update install commands(#1549)

Bug and Typo Fixes

- Update colab tutorial install cmds(#1522)
- Fix num_iters_per_epoch in analyze_logs.py(#1530)
- Fix distributed_sampler(#1532)
- Fix cd dir error(#1545)
- Update arg names(#1548)

ModelZoo

26.3 0.22.0 (03/05/2022)

Highlights

- Support Multigrid training strategy
- Support CPU training
- Support audio demo
- Support topk customizing in models/heads/base.py

New Features

- Support Multigrid training strategy(#1378)
- Support STGCN in demo_skeleton.py(#1391)
- Support CPU training(#1407)
- Support audio demo(#1425)
- Support topk customizing in models/heads/base.py(#1452)

Documentations

- Add OpenMMLab platform(#1393)
- Update links(#1394)
- Update readme in configs(#1404)
- Update instructions to install mmcv-full(#1426)
- Add shortcut(#1433)
- Update modelzoo(#1439)
- add video_structuralize in readme(#1455)

- Update OpenMMLab repo information(#1482)

Bug and Typo Fixes

- Update train.py(#1375)
- Fix printout bug(#1382)
- Update multi processing setting(#1395)
- Setup multi processing both in train and test(#1405)
- Fix bug in nondistributed multi-gpu training(#1406)
- Add variable fps in ava_dataset.py(#1409)
- Only support distributed training(#1414)
- Set test_mode for AVA configs(#1432)
- Support single label(#1434)
- Add check copyright(#1447)
- Support Windows CI(#1448)
- Fix wrong device of class_weight in models/losses/cross_entropy_loss.py(#1457)
- Fix bug caused by distributed(#1459)
- Update readme(#1460)
- Fix lint caused by colab automatic upload(#1461)
- Refine CI(#1471)
- Update pre-commit(#1474)
- Add deprecation message for deploy tool(#1483)

ModelZoo

- Support slowfast_steplr(#1421)

26.4 0.21.0 (31/12/2021)

Highlights

- Support 2s-AGCN
- Support publish models in Windows
- Improve some sthv1 related models
- Support BABEL

New Features

- Support 2s-AGCN(#1248)
- Support skip postproc in ntu_pose_extraction(#1295)
- Support publish models in Windows(#1325)
- Add copyright checkhook in pre-commit-config(#1344)

Documentations

- Add MMFlow (#1273)
- Revise README.md and add projects.md (#1286)
- Add 2s-AGCN in Updates(#1289)
- Add MMFewShot(#1300)
- Add MMHuman3d(#1304)
- Update pre-commit(#1313)
- Use share menu from the theme instead(#1328)
- Update installation command(#1340)

Bug and Typo Fixes

- Update the inference part in notebooks(#1256)
- Update the map_location(#1262)
- Fix bug that start_index is not used in RawFrameDecode(#1278)
- Fix bug in init_random_seed(#1282)
- Fix bug in setup.py(#1303)
- Fix interrogate error in workflows(#1305)
- Fix typo in slowfast config(#1309)
- Cancel previous runs that are not completed(#1327)
- Fix missing skip_postproc parameter(#1347)
- Update ssn.py(#1355)
- Use latest youtube-dl(#1357)
- Fix test-best(#1362)

ModelZoo

- Improve some sthv1 related models(#1306)
- Support BABEL(#1332)

26.5 0.20.0 (07/10/2021)

Highlights

- Support TorchServe
- Add video structuralize demo
- Support using 3D skeletons for skeleton-based action recognition
- Benchmark PoseC3D on UCF and HMDB

New Features

- Support TorchServe (#1212)
- Support 3D skeletons pre-processing (#1218)
- Support video structuralize demo (#1197)

Documentations

- Revise README.md and add projects.md (#1214)
- Add CN docs for Skeleton dataset, PoseC3D and ST-GCN (#1228, #1237, #1236)
- Add tutorial for custom dataset training for skeleton-based action recognition (#1234)

Bug and Typo Fixes

- Fix tutorial link (#1219)
- Fix GYM links (#1224)

ModelZoo

- Benchmark PoseC3D on UCF and HMDB (#1223)
- Add ST-GCN + 3D skeleton model for NTU60-XSub (#1236)

26.6 0.19.0 (07/10/2021)

Highlights

- Support ST-GCN
- Refactor the inference API
- Add code spell check hook

New Features

- Support ST-GCN (#1123)

Improvement

- Add label maps for every dataset (#1127)
- Remove useless code MultiGroupCrop (#1180)
- Refactor Inference API (#1191)
- Add code spell check hook (#1208)
- Use docker in CI (#1159)

Documentations

- Update metafiles to new OpenMMLAB protocols (#1134)
- Switch to new doc style (#1160)
- Improve the ERROR message (#1203)
- Fix invalid URL in getting_started (#1169)

Bug and Typo Fixes

- Compatible with new MMClassification (#1139)
- Add missing runtime dependencies (#1144)
- Fix THUMOS tag proposals path (#1156)
- Fix LoadHVULabel (#1194)
- Switch the default value of `persistent_workers` to False (#1202)

- Fix `_freeze_stages` for MobileNetV2 (#1193)
- Fix resume when building rawframes (#1150)
- Fix device bug for class weight (#1188)
- Correct Arg names in extract_audio.py (#1148)

ModelZoo

- Add TSM-MobileNetV2 ported from TSM (#1163)
- Add ST-GCN for NTURGB+D-XSub-60 (#1123)

26.7 0.18.0 (02/09/2021)

Improvement

- Add CopyRight (#1099)
- Support NTU Pose Extraction (#1076)
- Support Caching in RawFrameDecode (#1078)
- Add citations & Support python3.9 CI & Use fixed-version sphinx (#1125)

Documentations

- Add Descriptions of PoseC3D dataset (#1053)

Bug and Typo Fixes

- Fix SSV2 checkpoints (#1101)
- Fix CSN normalization (#1116)
- Fix typo (#1121)
- Fix new_crop_quadruple bug (#1108)

26.8 0.17.0 (03/08/2021)

Highlights

- Support PyTorch 1.9
- Support Pytorchvideo Transforms
- Support PreciseBN

New Features

- Support Pytorchvideo Transforms (#1008)
- Support PreciseBN (#1038)

Improvements

- Remove redundant augmentations in config files (#996)
- Make resource directory to hold common resource pictures (#1011)
- Remove deprecated FrameSelector (#1010)
- Support Concat Dataset (#1000)

- Add `to-mp4` option to resize_videos.py (#1021)
- Add option to keep tail frames (#1050)
- Update MIM support (#1061)
- Calculate Top-K accurate and inaccurate classes (#1047)

Bug and Typo Fixes

- Fix bug in PoseC3D demo (#1009)
- Fix some problems in resize_videos.py (#1012)
- Support torch1.9 (#1015)
- Remove redundant code in CI (#1046)
- Fix bug about persistent_workers (#1044)
- Support TimeSformer feature extraction (#1035)
- Fix ColorJitter (#1025)

ModelZoo

- Add TSM-R50 sthv1 models trained by PytorchVideo RandAugment and AugMix (#1008)
- Update SlowOnly SthV1 checkpoints (#1034)
- Add SlowOnly Kinetics400 checkpoints trained with Precise-BN (#1038)
- Add CSN-R50 from scratch checkpoints (#1045)
- TPN Kinetics-400 Checkpoints trained with the new ColorJitter (#1025)

Documentation

- Add Chinese translation of feature_extraction.md (#1020)
- Fix the code snippet in getting_started.md (#1023)
- Fix TANet config table (#1028)
- Add description to PoseC3D dataset (#1053)

26.9 0.16.0 (01/07/2021)

Highlights

- Support using backbone from pytorch-image-models(timm)
- Support PIMS Decoder
- Demo for skeleton-based action recognition
- Support Timesformer

New Features

- Support using backbones from pytorch-image-models(timm) for TSN (#880)
- Support torchvision transformations in preprocessing pipelines (#972)
- Demo for skeleton-based action recognition (#972)
- Support Timesformer (#839)

Improvements

- Add a tool to find invalid videos (#907, #950)
- Add an option to specify spectrogram_type (#909)
- Add json output to video demo (#906)
- Add MIM related docs (#918)
- Rename lr to scheduler (#916)
- Support `--cfg-options` for demos (#911)
- Support number counting for flow-wise filename template (#922)
- Add Chinese tutorial (#941)
- Change ResNet3D default values (#939)
- Adjust script structure (#935)
- Add font color to args in long_video_demo (#947)
- Polish code style with Pylint (#908)
- Support PIMS Decoder (#946)
- Improve Metafiles (#956, #979, #966)
- Add links to download Kinetics400 validation (#920)
- Audit the usage of shutil.rmtree (#943)
- Polish localizer related codes(#913)

Bug and Typo Fixes

- Fix spatiotemporal detection demo (#899)
- Fix docstring for 3D inflate (#925)
- Fix bug of writing text to video with TextClip (#952)
- Fix mmcv install in CI (#977)

ModelZoo

- Add TSN with Swin Transformer backbone as an example for using pytorch-image-models(timm) backbones (#880)
- Port CSN checkpoints from VMZ (#945)
- Release various checkpoints for UCF101, HMDB51 and Sthv1 (#938)
- Support Timesformer (#839)
- Update TSM modelzoo (#981)

26.10 0.15.0 (31/05/2021)

Highlights

- Support PoseC3D
- Support ACRN
- Support MIM

New Features

- Support PoseC3D (#786, #890)
- Support MIM (#870)
- Support ACRN and Focal Loss (#891)
- Support Jester dataset (#864)

Improvements

- Add `metric_options` for evaluation to docs (#873)
- Support creating a new label map based on custom classes for demos about spatio temporal demo (#879)
- Improve document about AVA dataset preparation (#878)
- Provide a script to extract clip-level feature (#856)

Bug and Typo Fixes

- Fix issues about resume (#877, #878)
- Correct the key name of `eval_results` dictionary for metric 'mmit_mean_average_precision' (#885)

ModelZoo

- Support Jester dataset (#864)
- Support ACRN and Focal Loss (#891)

26.11 0.14.0 (30/04/2021)

Highlights

- Support TRN
- Support Diving48

New Features

- Support TRN (#755)
- Support Diving48 (#835)
- Support Webcam Demo for Spatio-temporal Action Detection Models (#795)

Improvements

- Add softmax option for pytorch2onnx tool (#781)
- Support TRN (#755)
- Test with onnx models and TensorRT engines (#758)
- Speed up AVA Testing (#784)

- Add `self.with_neck` attribute (#796)
- Update installation document (#798)
- Use a random master port (#809)
- Update AVA processing data document (#801)
- Refactor spatio-temporal augmentation (#782)
- Add QR code in CN README (#812)
- Add Alternative way to download Kinetics (#817, #822)
- Refactor Sampler (#790)
- Use EvalHook in MMCV with backward compatibility (#793)
- Use MMCV Model Registry (#843)

Bug and Typo Fixes

- Fix a bug in pytorch2onnx.py when `num_classes <= 4` (#800, #824)
- Fix `demo_spatiotemporal_det.py` error (#803, #805)
- Fix loading config bugs when resume (#820)
- Make HMDB51 annotation generation more robust (#811)

ModelZoo

- Update checkpoint for 256 height in something-V2 (#789)
- Support Diving48 (#835)

26.12 0.13.0 (31/03/2021)

Highlights

- Support LFB
- Support using backbone from MMCls/TorchVision
- Add Chinese documentation

New Features

- Support LFB (#553)
- Support using backbones from MMCls for TSN (#679)
- Support using backbones from TorchVision for TSN (#720)
- Support Mixup and Cutmix for recognizers (#681)
- Support Chinese documentation (#665, #680, #689, #701, #702, #703, #706, #716, #717, #731, #733, #735, #736, #737, #738, #739, #740, #742, #752, #759, #761, #772, #775)

Improvements

- Add slowfast config/json/log/ckpt for training custom classes of AVA (#678)
- Set RandAugment as Imgaug default transforms (#585)
- Add `--test-last` & `--test-best` for `tools/train.py` to test checkpoints after training (#608)
- Add fcn_testing in TPN (#684)

- Remove redundant recall functions (#741)
- Recursively remove pretrained step for testing (#695)
- Improve demo by limiting inference fps (#668)

Bug and Typo Fixes

- Fix a bug about multi-class in VideoDataset (#723)
- Reverse key-value in anet filelist generation (#686)
- Fix flow norm cfg typo (#693)

ModelZoo

- Add LFB for AVA2.1 (#553)
- Add TSN with ResNeXt-101-32x4d backbone as an example for using MMCls backbones (#679)
- Add TSN with Densenet161 backbone as an example for using TorchVision backbones (#720)
- Add slowonly_nl_embedded_gaussian_r50_4x16x1_150e_kinetics400_rgb (#690)
- Add slowonly_nl_embedded_gaussian_r50_8x8x1_150e_kinetics400_rgb (#704)
- Add slowonly_nl_kinetics_pretrained_r50_4x16x1(8x8x1)_20e_ava_rgb (#730)

26.13 0.12.0 (28/02/2021)

Highlights

- Support TSM-MobileNetV2
- Support TANet
- Support GPU Normalize

New Features

- Support TSM-MobileNetV2 (#415)
- Support flip with label mapping (#591)
- Add seed option for sampler (#642)
- Support GPU Normalize (#586)
- Support TANet (#595)

Improvements

- Training custom classes of ava dataset (#555)
- Add CN README in homepage (#592, #594)
- Support soft label for CrossEntropyLoss (#625)
- Refactor config: Specify `train_cfg` and `test_cfg` in `model` (#629)
- Provide an alternative way to download older kinetics annotations (#597)
- Update FAQ for
  - 1). data pipeline about video and frames (#598)
  - 2). how to show results (#598)

  - 3). batch size setting for batchnorm (#657)
  - 4). how to fix stages of backbone when finetuning models (#658)
- Modify default value of `save_best` (#600)
- Use BibTex rather than latex in markdown (#607)
- Add warnings of uninstalling mmdet and supplementary documents (#624)
- Support soft label for CrossEntropyLoss (#625)

Bug and Typo Fixes

- Fix value of `pem_low_temporal_iou_threshold` in BSN (#556)
- Fix ActivityNet download script (#601)

ModelZoo

- Add TSM-MobileNetV2 for Kinetics400 (#415)
- Add deeper SlowFast models (#605)

26.14 0.11.0 (31/01/2021)

Highlights

- Support imgaug
- Support spatial temporal demo
- Refactor EvalHook, config structure, unittest structure

New Features

- Support imgaug for augmentations in the data pipeline (#492)
- Support setting `max_testing_views` for extremely large models to save GPU memory used (#511)
- Add spatial temporal demo (#547, #566)

Improvements

- Refactor EvalHook (#395)
- Refactor AVA hook (#567)
- Add repo citation (#545)
- Add dataset size of Kinetics400 (#503)
- Add lazy operation docs (#504)
- Add class_weight for CrossEntropyLoss and BCELossWithLogits (#509)
- add some explanation about the resampling in slowfast (#502)
- Modify paper title in README.md (#512)
- Add alternative ways to download Kinetics (#521)
- Add OpenMMLab projects link in README (#530)
- Change default preprocessing to shortedge to 256 (#538)
- Add config tag in dataset README (#540)

- Add solution for markdownlint installation issue (#497)
- Add dataset overview in readthedocs (#548)
- Modify the trigger mode of the warnings of missing mmdet (#583)
- Refactor config structure (#488, #572)
- Refactor unittest structure (#433)

Bug and Typo Fixes

- Fix a bug about ava dataset validation (#527)
- Fix a bug about ResNet pretrain weight initialization (#582)
- Fix a bug in CI due to MMCV index (#495)
- Remove invalid links of MiT and MMiT (#516)
- Fix frame rate bug for AVA preparation (#576)

ModelZoo

26.15 0.10.0 (31/12/2020)

Highlights

- Support Spatio-Temporal Action Detection (AVA)
- Support precise BN

New Features

- Support precise BN (#501)
- Support Spatio-Temporal Action Detection (AVA) (#351)
- Support to return feature maps in `inference_recognizer` (#458)

Improvements

- Add arg `stride` to long_video_demo.py, to make inference faster (#468)
- Support training and testing for Spatio-Temporal Action Detection (#351)
- Fix CI due to pip upgrade (#454)
- Add markdown lint in pre-commit hook (#255)
- Speed up confusion matrix calculation (#465)
- Use title case in modelzoo statistics (#456)
- Add FAQ documents for easy troubleshooting. (#413, #420, #439)
- Support Spatio-Temporal Action Detection with context (#471)
- Add class weight for CrossEntropyLoss and BCELossWithLogits (#509)
- Add Lazy OPs docs (#504)

Bug and Typo Fixes

- Fix typo in default argument of BaseHead (#446)
- Fix potential bug about `output_config` overwrite (#463)

ModelZoo

- Add SlowOnly, SlowFast for AVA2.1 (#351)

26.16 0.9.0 (30/11/2020)

Highlights

- Support GradCAM utils for recognizers
- Support ResNet Audio model

New Features

- Automatically add modelzoo statistics to readthedocs (#327)
- Support GYM99 (#331, #336)
- Add AudioOnly Pathway from AVSlowFast. (#355)
- Add GradCAM utils for recognizer (#324)
- Add print config script (#345)
- Add online motion vector decoder (#291)

Improvements

- Support PyTorch 1.7 in CI (#312)
- Support to predict different labels in a long video (#274)
- Update docs bout test crops (#359)
- Polish code format using pylint manually (#338)
- Update unittest coverage (#358, #322, #325)
- Add random seed for building filelists (#323)
- Update colab tutorial (#367)
- set default batch_size of evaluation and testing to 1 (#250)
- Rename the preparation docs to `README.md` (#388)
- Move docs about demo to `demo/README.md` (#329)
- Remove redundant code in `tools/test.py` (#310)
- Automatically calculate number of test clips for Recognizer2D (#359)

Bug and Typo Fixes

- Fix rename Kinetics classnames bug (#384)
- Fix a bug in BaseDataset when `data_prefix` is None (#314)
- Fix a bug about `tmp_folder` in `OpenCVInit` (#357)
- Fix `get_thread_id` when not using disk as backend (#354, #357)
- Fix the bug of HVU object `num_classes` from 1679 to 1678 (#307)
- Fix typo in `export_model.md` (#399)
- Fix OmniSource training configs (#321)

- Fix Issue #306: Bug of SampleAVAFrames (#317)

ModelZoo

- Add SlowOnly model for GYM99, both RGB and Flow (#336)
- Add auto modelzoo statistics in readthedocs (#327)
- Add TSN for HMDB51 pretrained on Kinetics400, Moments in Time and ImageNet. (#372)

26.17 v0.8.0 (31/10/2020)

Highlights

- Support OmniSource
- Support C3D
- Support video recognition with audio modality
- Support HVU
- Support X3D

New Features

- Support AVA dataset preparation (#266)
- Support the training of video recognition dataset with multiple tag categories (#235)
- Support joint training with multiple training datasets of multiple formats, including images, untrimmed videos, etc. (#242)
- Support to specify a start epoch to conduct evaluation (#216)
- Implement X3D models, support testing with model weights converted from SlowFast (#288)
- Support specify a start epoch to conduct evaluation (#216)

Improvements

- Set default values of 'average_clips' in each config file so that there is no need to set it explicitly during testing in most cases (#232)
- Extend HVU datatools to generate individual file list for each tag category (#258)
- Support data preparation for Kinetics-600 and Kinetics-700 (#254)
- Use `metric_dict` to replace hardcoded arguments in `evaluate` function (#286)
- Add `cfg-options` in arguments to override some settings in the used config for convenience (#212)
- Rename the old evaluating protocol `mean_average_precision` as `mmit_mean_average_precision` since it is only used on MMIT and is not the `mAP` we usually talk about. Add `mean_average_precision`, which is the real `mAP` (#235)
- Add accurate setting (Three crop * 2 clip) and report corresponding performance for TSM model (#241)
- Add citations in each preparing_dataset.md in `tools/data/dataset` (#289)
- Update the performance of audio-visual fusion on Kinetics-400 (#281)
- Support data preparation of OmniSource web datasets, including GoogleImage, InsImage, InsVideo and KineticsRawVideo (#294)
- Use `metric_options` dict to provide metric args in `evaluate` (#286)

Bug Fixes

- Register `FrameSelector` in `PIPELINES` (#268)
- Fix the potential bug for default value in dataset_setting (#245)
- Fix multi-node dist test (#292)
- Fix the data preparation bug for `something-something` dataset (#278)
- Fix the invalid config url in slowonly README data benchmark (#249)
- Validate that the performance of models trained with videos have no significant difference comparing to the performance of models trained with rawframes (#256)
- Correct the `img_norm_cfg` used by TSN-3seg-R50 UCF-101 model, improve the Top-1 accuracy by 3% (#273)

ModelZoo

- Add Baselines for Kinetics-600 and Kinetics-700, including TSN-R50-8seg and SlowOnly-R50-8x8 (#259)
- Add OmniSource benchmark on MiniKineitcs (#296)
- Add Baselines for HVU, including TSN-R18-8seg on 6 tag categories of HVU (#287)
- Add X3D models ported from SlowFast (#288)

26.18 v0.7.0 (30/9/2020)

Highlights

- Support TPN
- Support JHMDB, UCF101-24, HVU dataset preparation
- support onnx model conversion

New Features

- Support the data pre-processing pipeline for the HVU Dataset (#277)
- Support real-time action recognition from web camera (#171)
- Support onnx (#160)
- Support UCF101-24 preparation (#219)
- Support evaluating mAP for ActivityNet with CUHK17_activitynet_pred (#176)
- Add the data pipeline for ActivityNet, including downloading videos, extracting RGB and Flow frames, finetuning TSN and extracting feature (#190)
- Support JHMDB preparation (#220)

ModelZoo

- Add finetuning setting for SlowOnly (#173)
- Add TSN and SlowOnly models trained with OmniSource, which achieve 75.7% Top-1 with TSN-R50-3seg and 80.4% Top-1 with SlowOnly-R101-8x8 (#215)

Improvements

- Support demo with video url (#165)
- Support multi-batch when testing (#184)

- Add tutorial for adding a new learning rate updater (#181)
- Add config name in meta info (#183)
- Remove git hash in `__version__` (#189)
- Check mmcv version (#189)
- Update url with 'https://download.openmmlab.com' (#208)
- Update Docker file to support PyTorch 1.6 and update `install.md` (#209)
- Polish readsthedocs display (#217, #229)

Bug Fixes

- Fix the bug when using OpenCV to extract only RGB frames with original shape (#184)
- Fix the bug of sthv2 `num_classes` from 339 to 174 (#174, #207)

26.19 v0.6.0 (2/9/2020)

Highlights

- Support TIN, CSN, SSN, NonLocal
- Support FP16 training

New Features

- Support NonLocal module and provide ckpt in TSM and I3D (#41)
- Support SSN (#33, #37, #52, #55)
- Support CSN (#87)
- Support TIN (#53)
- Support HMDB51 dataset preparation (#60)
- Support encoding videos from frames (#84)
- Support FP16 training (#25)
- Enhance demo by supporting rawframe inference (#59), output video/gif (#72)

ModelZoo

- Update Slowfast modelzoo (#51)
- Update TSN, TSM video checkpoints (#50)
- Add data benchmark for TSN (#57)
- Add data benchmark for SlowOnly (#77)
- Add BSN/BMN performance results with feature extracted by our codebase (#99)

Improvements

- Polish data preparation codes (#70)
- Improve data preparation scripts (#58)
- Improve unittest coverage and minor fix (#62)
- Support PyTorch 1.6 in CI (#117)

- Support `with_offset` for rawframe dataset (#48)
- Support json annotation files (#119)
- Support `multi-class` in TSMHead (#104)
- Support using `val_step()` to validate data for each `val` workflow (#123)
- Use `xxInit()` method to get `total_frames` and make `total_frames` a required key (#90)
- Add paper introduction in model readme (#140)
- Adjust the directory structure of `tools/` and rename some scripts files (#142)

Bug Fixes

- Fix configs for localization test (#67)
- Fix configs of SlowOnly by fixing lr to 8 gpus (#136)
- Fix the bug in analyze_log (#54)
- Fix the bug of generating HMDB51 class index file (#69)
- Fix the bug of using `load_checkpoint()` in ResNet (#93)
- Fix the bug of `--work-dir` when using slurm training script (#110)
- Correct the sthv1/sthv2 rawframes filelist generate command (#71)
- `CosineAnnealing` typo (#47)

26.20 v0.5.0 (9/7/2020)

Highlights

- MMAction2 is released

New Features

- Support various datasets: UCF101, Kinetics-400, Something-Something V1&V2, Moments in Time, Multi-Moments in Time, THUMOS14
- Support various action recognition methods: TSN, TSM, R(2+1)D, I3D, SlowOnly, SlowFast, Non-local
- Support various action localization methods: BSN, BMN
- Colab demo for action recognition

CHAPTER

TWENTYSEVEN

FAQ

27.1 Outline

We list some common issues faced by many users and their corresponding solutions here.

- *FAQ*
  - *Outline*
  - *Installation*
  - *Data*
  - *Training*
  - *Testing*
  - *Deploying*

Feel free to enrich the list if you find any frequent issues and have ways to help others to solve them. If the contents here do not cover your issue, please create an issue using the provided templates and make sure you fill in all required information in the template.

27.2 Installation

- **"No module named 'mmcv.ops'"; "No module named 'mmcv._ext'"**
  1. Uninstall existing mmcv in the environment using `pip uninstall mmcv`
  2. Install mmcv-full following the installation instruction
- **"OSError: MoviePy Error: creation of None failed because of the following error"**

  Refer to install.md
  1. For Windows users, ImageMagick will not be automatically detected by MoviePy, there is a need to modify `moviepy/config_defaults.py` file by providing the path to the ImageMagick binary called `magick`, like `IMAGEMAGICK_BINARY = "C:\\Program Files\\ImageMagick_VERSION\\magick.exe"`
  2. For Linux users, there is a need to modify the `/etc/ImageMagick-6/policy.xml` file by commenting out `<policy domain="path" rights="none" pattern="@*" />` to `<!-- <policy domain="path" rights="none" pattern="@*" /> -->`, if ImageMagick is not detected by moviepy.
- **"Why I got the error message 'Please install XXCODEBASE to use XXX' even if I have already installed XXCODEBASE?"**

You got that error message because our project failed to import a function or a class from XXCODEBASE. You can try to run the corresponding line to see what happens. One possible reason is, for some codebases in OpenMMLAB, you need to install mmcv-full before you install them.

27.3 Data

- **FileNotFound like `No such file or directory: xxx/xxx/img_00300.jpg`**

  In our repo, we set `start_index=1` as default value for rawframe dataset, and `start_index=0` as default value for video dataset. If users encounter FileNotFound error for the first or last frame of the data, there is a need to check the files begin with offset 0 or 1, that is `xxx_00000.jpg` or `xxx_00001.jpg`, and then change the `start_index` value of data pipeline in configs.

- **How should we preprocess the videos in the dataset? Resizing them to a fix size(all videos with the same height-width ratio) like `340x256`(1) or resizing them so that the short edges of all videos are of the same length (256px or 320px)**

  We have tried both preprocessing approaches and found (2) is a better solution in general, so we use (2) with short edge length 256px as the default preprocessing setting. We benchmarked these preprocessing approaches and you may find the results in TSN Data Benchmark and SlowOnly Data Benchmark.

- **Mismatched data pipeline items lead to errors like `KeyError: 'total_frames'`**

  We have both pipeline for processing videos and frames.

  For videos, We should decode them on the fly in the pipeline, so pairs like `DecordInit & DecordDecode`, `OpenCVInit & OpenCVDecode`, `PyAVInit & PyAVDecode` should be used for this case like this example.

  For Frames, the image has been decoded offline, so pipeline item `RawFrameDecode` should be used for this case like this example.

  `KeyError: 'total_frames'` is caused by incorrectly using `RawFrameDecode` step for videos, since when the input is a video, it can not get the `total_frame` beforehand.

27.4 Training

- **How to just use trained recognizer models for backbone pre-training?**

  Refer to Use Pre-Trained Model, in order to use the pre-trained model for the whole network, the new config adds the link of pre-trained models in the `load_from`.

  And to use backbone for pre-training, you can change `pretrained` value in the backbone dict of config files to the checkpoint path / url. When training, the unexpected keys will be ignored.

- **How to visualize the training accuracy/loss curves in real-time?**

  Use `TensorboardLoggerHook` in `log_config` like

```
log_config=dict(interval=20, hooks=[dict(type='TensorboardLoggerHook')])
```

  You can refer to *tutorials/1_config.md*, tutorials/7_customize_runtime.md, and this.

- **In batchnorm.py: Expected more than 1 value per channel when training**

  To use batchnorm, the batch_size should be larger than 1. If `drop_last` is set as False when building dataloaders, sometimes the last batch of an epoch will have `batch_size==1` (what a coincidence ...) and training will throw out this error. You can set `drop_last` as True to avoid this error:

```
train_dataloader=dict(drop_last=True)
```

- **How to fix stages of backbone when finetuning a model?**

  You can refer to `def _freeze_stages()` and `frozen_stages`, reminding to set `find_unused_parameters = True` in config files for distributed training or testing.

  Actually, users can set `frozen_stages` to freeze stages in backbones except C3D model, since all backbones inheriting from `ResNet` and `ResNet3D` support the inner function `_freeze_stages()`.

- **How to set memcached setting in config files?**

  In MMAction2, you can pass memcached kwargs to `class DecordInit` for video dataset or `RawFrameDecode` for rawframes dataset. For more details, you can refer to `class FileClient` in MMCV for more details.

  Here is an example to use memcached for rawframes dataset:

```
mc_cfg = dict(server_list_cfg='server_list_cfg', client_cfg='client_cfg', sys_path=
↪'sys_path')

train_pipeline = [
  ...
  dict(type='RawFrameDecode', io_backend='memcached', **mc_cfg),
  ...
]
```

- **How to set `load_from` value in config files to finetune models?**

  In MMAction2, We set `load_from=None` as default in `configs/_base_/default_runtime.py` and owing to inheritance design, users can directly change it by setting `load_from` in their configs.

27.5 Testing

- **How to make predicted score normalized by softmax within [0, 1]?**

  change this in the config, make `model['test_cfg'] = dict(average_clips='prob')`.

- **What if the model is too large and the GPU memory can not fit even only one testing sample?**

  By default, the 3d models are tested with 10clips x 3crops, which are 30 views in total. For extremely large models, the GPU memory can not fit even only one testing sample (cuz there are 30 views). To handle this, you can set `max_testing_views=n` in `model['test_cfg']` of the config file. If so, n views will be used as a batch during forwarding to save GPU memory used.

- **How to show test results?**

  During testing, we can use the command `--out xxx.json/pkl/yaml` to output result files for checking. The testing output has exactly the same order as the test dataset. Besides, we provide an analysis tool for evaluating a model using the output result files in `tools/analysis/eval_metric.py`

27.6 Deploying

- **Why is the onnx model converted by mmaction2 throwing error when converting to other frameworks such as TensorRT?**

  For now, we can only make sure that models in mmaction2 are onnx-compatible. However, some operations in onnx may be unsupported by your target framework for deployment, e.g. TensorRT in this issue. When such situation occurs, we suggest you raise an issue and ask the community to help as long as `pytorch2onnx.py` works well and is verified numerically.

CHAPTER

TWENTYEIGHT

MMACTION.APIS

CHAPTER

TWENTYNINE

MMACTION.CORE

29.1 optimizer

29.2 evaluation

scheduler ^^ .. automodule:: mmaction.core.scheduler

members

CHAPTER

THIRTY

MMACTION.LOCALIZATION

30.1 localization

CHAPTER

THIRTYONE

MMACTION.MODELS

CHAPTER

THIRTYTWO

MMACTION.DATASETS

32.1 datasets

32.2 pipelines

32.3 samplers

CHAPTER

THIRTYTHREE

MMACTION.UTILS

CHAPTER

THIRTYFOUR

MMACTION.LOCALIZATION

CHAPTER

THIRTYFIVE

ENGLISH

CHAPTER

THIRTYSIX

CHAPTER

THIRTYSEVEN

INDICES AND TABLES

- genindex
- search