MMAction2

Release 1.0.0rc3

MMAction2 Authors

Feb 16, 2023

GET STARTED

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

CHAPTER ONE

PREREQUISITES

In this section we demonstrate how to prepare an environment with PyTorch.

MMAction2 works on Linux, Windows and macOS. It requires Python 3.7+, CUDA 9.2+ and PyTorch 1.6+.

Note: If you are experienced with PyTorch and have already installed it, just skip this part and jump to the next section. Otherwise, you can follow these steps for the preparation.

Step 1. Download and install Miniconda from the official website.

Step 2. Create a conda environment and activate it.

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

Step 3. Install PyTorch following official instructions, e.g.

On GPU platforms:

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

Warning: This command will automatically install the latest version PyTorch and cudatoolkit, please check whether they match your environment.

On CPU platforms:

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

CHAPTER

TWO

INSTALLATION

We recommend that users follow our best practices to install MMAction2. However, the whole process is highly customizable. See Customize Installation section for more information.

2.1 Best Practices

Step 1. Install MMEngine and MMCV using MIM.

```
pip install -U openmim
mim install mmengine 'mmcv>=2.0.0rc1'
```

Note that some of the demo scripts in MMAction2 require MMDetection (mmdet) for human detection, and MMPose for pose estimation. If you want to run these demo scripts, you can easily install mmdet and mmpose as dependencies by running:

```
mim install "mmdet>=3.0.0rc5"
mim install "mmpose>=1.0.0rc0"
```

Step 2. Install MMAction2.

According to your needs, we support two install modes:

- Install from source (Recommended): You want to develop your own action recognition task or new features on MMAction2 framework. For example, adding new dataset or new models. Thus, you can use all tools we provided.
- Install as a Python package: You just want to call MMAction2's APIs or import MMAction2's modules in your project.

2.1.1 Install from source

In this case, install mmaction2 from source:

```
git clone https://github.com/open-mmlab/mmaction2.git
cd mmaction2
git checkout 1.x
pip install -v -e .
# "-v" means verbose, or more output
# "-e" means installing a project in editable mode,
# thus any local modifications made to the code will take effect without re-installation.
```

Optionally, if you want to contribute to MMAction2 or experience experimental functions, please checkout to the dev-1.x branch:

```
git checkout dev-1.x
```

2.1.2 Install as a Python package

Just install with pip.

```
pip install "mmaction2>=1.0rc0"
```

2.2 Verify the installation

To verify whether MMAction2 is installed correctly, we provide some sample codes to run an inference demo.

Step 1. Download the config and checkpoint files.

```
mim download mmaction2 --config tsn_imagenet-pretrained-r50_8xb32-1x1x8-100e_kinetics400-
↪rgb --dest .
```

Step 2. Verify the inference demo.

Option (a). If you install mmaction2 from source, you can run the following command:

```
# The demo.mp4 and label_map_k400.txt are both from Kinetics-400
python demo/demo.py tsn_imagenet-pretrained-r50_8xb32-1x1x8-100e_kinetics400-rgb.py \
    tsn_imagenet-pretrained-r50_8xb32-1x1x8-100e_kinetics400-rgb_20220906-2692d16c.pth \
    demo/demo.mp4 tools/data/kinetics/label_map_k400.txt
```

You will see the top-5 labels with corresponding scores in your terminal.

Option (b). If you install mmaction2 as a python package, you can run the following codes in your python interpreter, which will do the similar verification:

```
from operator import itemgetter
from mmaction.apis import init_recognizer, inference_recognizer

config_file = 'tsn_imagenet-pretrained-r50_8xb32-1x1x8-100e_kinetics400-rgb.py'
checkpoint_file = 'tsn_imagenet-pretrained-r50_8xb32-1x1x8-100e_kinetics400-rgb_20220906-
↪2692d16c.pth'
video_file = 'demo/demo.mp4'
label_file = 'tools/data/kinetics/label_map_k400.txt'
model = init_recognizer(config_file, checkpoint_file, device='cpu')  # or device='cuda:0'
pred_result = inference_recognizer(model, video_file)

pred_scores = pred_result.pred_scores.item.tolist()
score_tuples = tuple(zip(range(len(pred_scores)), pred_scores))
score_sorted = sorted(score_tuples, key=itemgetter(1), reverse=True)
top5_label = score_sorted[:5]

labels = open(label_file).readlines()
labels = [x.strip() for x in labels]
```

(continues on next page)

(continued from previous page)

```
results = [(labels[k[0]], k[1]) for k in top5_label]

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

2.3 Customize Installation

2.3.1 CUDA versions

When installing PyTorch, you may need to specify the version of CUDA. If you are not clear on which to choose, follow our recommendations:

- For Ampere-based NVIDIA GPUs, such as GeForce 30 series and NVIDIA A100, CUDA 11 is a must.
- For older NVIDIA GPUs, CUDA 11 is backward compatible, but CUDA 10.2 offers better compatibility and is more lightweight.

Please make sure the GPU driver satisfies the minimum version requirements. See this table for more information.

Note: Installing CUDA runtime libraries is enough if you follow our best practices, because no CUDA code will be compiled locally. However if you hope to compile MMCV from source or develop other CUDA operators, you need to install the complete CUDA toolkit from NVIDIA's website, and its version should match the CUDA version of PyTorch. i.e., the specified version of cudatoolkit in `conda install` command.

2.3.2 Install MMCV without MIM

MMCV contains C++ and CUDA extensions, so it depends on PyTorch in a complex way. MIM solves such dependencies automatically and makes the installation easier. However, it is not a must.

To install MMCV with pip instead of MIM, please follow MMCV installation guides. This requires manually specifying a find-url based on PyTorch version and its CUDA version.

For example, the following command install mmcv built for PyTorch 1.10.x and CUDA 11.3.

```
pip install 'mmcv>=2.0.0rc1' -f https://download.openmmlab.com/mmcv/dist/cu113/torch1.10/
↪index.html
```

2.3.3 Install on CPU-only platforms

MMAction2 can be built for CPU-only environment. In CPU mode you can train, test or inference a model.

Some functionalities are gone in this mode, usually GPU-compiled ops. But don't worry, almost all models in MMAction2 don't depend on these ops.

2.3.4 Using MMAction2 with Docker

We provide a Dockerfile to build an image. Ensure that your docker version >=19.03.

```
# build an image with PyTorch 1.6.0, CUDA 10.1, CUDNN 7.
# If you prefer other versions, just modified the Dockerfile
docker build -f ./docker/Dockerfile --rm -t mmaction2 .
```

Run it with

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

CHAPTER

THREE

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_tools/print_config.py /PATH/TO/CONFIG` to see the complete config.

- Modify config through script arguments
- Config File Structure
- Config File Naming Convention
  - Config System for Action Recognition
  - Config System for Spatio-Temporal Action Detection
  - Config System for Action localization

3.1 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 `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 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 `model.data_preprocessor.mean=[123.675, 116.28, 103.53]`. If you want to change this key, you may specify `--cfg-options model.data_preprocessor.mean="[128,128,128]"`. Note that the quotation mark " is necessary to support list/tuple data types.

3.2 Config File Structure

There are 3 basic component types under `configs/_base_`, models, schedules, 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 based on TSN, users may first inherit the basic TSN structure by specifying `_base_ = ../tsn/tsn_imagenet-pretrained-r50_8xb32-1x1x3-100e_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 mmengine for detailed documentation.

3.3 Config File Naming Convention

We follow the style below to name config files. Contributors are advised to follow the same style. The config file names are divided into several parts. Logically, different parts are concatenated by underscores `'_'`, and settings in the same part are concatenated by dashes `'-'`.

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

`{xxx}` is required field and `[yyy]` is optional.

- `{algorithm info}`:
  - `{model}`: model type, e.g. `tsn`, `i3d`, `swin`, `vit`, etc.
  - `[model setting]`: specific setting for some models, e.g. `base`, `p16`, `w877`, etc.
- `{module info}`:
  - `[pretained info]`: pretrained information, e.g. `kinetics400-pretrained`, `in1k-pre`, etc.
  - `{backbone}`: backbone type. e.g. `r50` (ResNet-50), etc.
  - `[backbone setting]`: specific setting for some backbones, e.g. `nl-dot-product`, `bnfrozen`, `nopool`, etc.
- `{training info}`:
  - `{gpu x batch_per_gpu]}`: GPUs and samples per GPU.
  - `{pipeline setting}`: frame sample setting, e.g. `dense`, `{clip_len}x{frame_interval}x{num_clips}`, `u48`, etc.
  - `{schedule}`: training schedule, e.g. `coslr-20e`.
- `{data info}`:
  - `{dataset}`: dataset name, e.g. `kinetics400`, `mmit`, etc.
  - `{modality}`: data modality, e.g. `rgb`, `flow`, `keypoint-2d`, etc.

3.3.1 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',  # Class name 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
        average_clips='prob'),  # Method to average multiple clip results
    data_preprocessor=dict(  # Dict for data preprocessor
        type='ActionDataPreprocessor',  # Name of data preprocessor
        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
        format_shape='NCHW'),  # Final image shape format
    # model training and testing settings
    train_cfg=None,  # Config of training hyperparameters for TSN
    test_cfg=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

train_pipeline = [  # Training data processing pipeline
    dict(  # Config of SampleFrames
        type='SampleFrames',  # Sample frames pipeline, sampling frames from video
```

(continues on next page)

(continued from previous page)

```
        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 FormatShape
        type='FormatShape',  # Format shape pipeline, Format final image shape to␣
↪the given input_format
        input_format='NCHW'),  # Final image shape format
    dict(type='PackActionInputs')  # Config of PackActionInputs
]
val_pipeline = [  # Validation data processing pipeline
    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
    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 FormatShape
        type='FormatShape',  # Format shape pipeline, Format final image shape to␣
↪the given input_format
```

(continues on next page)

(continued from previous page)

```
        input_format='NCHW'),  # Final image shape format
    dict(type='PackActionInputs')  # Config of PackActionInputs
]
test_pipeline = [  # Testing data processing pipeline
    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 FormatShape
        type='FormatShape',  # Format shape pipeline, Format final image shape to␣
↪the given input_format
        input_format='NCHW'),  # Final image shape format
    dict(type='PackActionInputs')  # Config of PackActionInputs
]

train_dataloader = dict(  # Config of train dataloader
    batch_size=32,  # Batch size of each single GPU during training
    num_workers=8,  # Workers to pre-fetch data for each single GPU during training
    persistent_workers=True,  # If `True`, the dataloader will not shut down the␣
↪worker processes after an epoch end, which can accelerate training speed
    sampler=dict(
        type='DefaultSampler',  # DefaultSampler which supports both distributed␣
↪and non-distributed training. Refer to https://github.com/open-mmlab/mmengine/
↪blob/main/mmengine/dataset/sampler.py
        shuffle=True),  # Randomly shuffle the training data in each epoch
    dataset=dict(  # Config of train dataset
        type=dataset_type,
        ann_file=ann_file_train,  # Path of annotation file
        data_prefix=dict(img=data_root),  # Prefix of frame path
        pipeline=train_pipeline))
val_dataloader = dict(  # Config of validation dataloader
    batch_size=1,  # Batch size of each single GPU during validation
    num_workers=8,  # Workers to pre-fetch data for each single GPU during␣
↪validation
    persistent_workers=True,  # If `True`, the dataloader will not shut down the␣
↪worker processes after an epoch end
    sampler=dict(
        type='DefaultSampler',
```

(continues on next page)

(continued from previous page)

```
        shuffle=False),  # Not shuffle during validation and testing
    dataset=dict(  # Config of validation dataset
        type=dataset_type,
        ann_file=ann_file_val,  # Path of annotation file
        data_prefix=dict(img=data_root_val),  # Prefix of frame path
        pipeline=val_pipeline,
        test_mode=True))
test_dataloader = dict(  # Config of test dataloader
    batch_size=32,  # Batch size of each single GPU during testing
    num_workers=8,  # Workers to pre-fetch data for each single GPU during testing
    persistent_workers=True,  # If `True`, the dataloader will not shut down the␣
↪worker processes after an epoch end
    sampler=dict(
        type='DefaultSampler',
        shuffle=False),  # Not shuffle during validation and testing
    dataset=dict(  # Config of test dataset
        type=dataset_type,
        ann_file=ann_file_val,  # Path of annotation file
        data_prefix=dict(img=data_root_val),  # Prefix of frame path
        pipeline=test_pipeline,
        test_mode=True))

# evaluation settings
val_evaluator = dict(type='AccMetric')  # Config of validation evaluator
test_evaluator = val_evaluator  # Config of testing evaluator

train_cfg = dict(  # Config of training loop
    type='EpochBasedTrainLoop',  # Name of training loop
    max_epochs=100,  # Total training epochs
    val_begin=1,  # The epoch that begins validating
    val_interval=1)  # Validation interval
val_cfg = dict(  # Config of validation loop
    type='ValLoop')  # Name of validation loop
test_cfg = dict( # Config of testing loop
    type='TestLoop')  # Name of testing loop

# learning policy
param_scheduler = [  # Parameter scheduler for updating optimizer parameters,␣
↪support dict or list
    dict(type='MultiStepLR',  # Decays the learning rate once the number of epoch␣
↪reaches one of the milestones
        begin=0,  # Step at which to start updating the learning rate
        end=100,  # Step at which to stop updating the learning rate
        by_epoch=True,  # Whether the scheduled learning rate is updated by epochs
        milestones=[40, 80],  # Steps to decay the learning rate
        gamma=0.1)]  # Multiplicative factor of learning rate decay

# optimizer
optim_wrapper = dict(  # Config of optimizer wrapper
    type='OptimWrapper',  # Name of optimizer wrapper, switch to AmpOptimWrapper to␣
↪enable mixed precision training
    optimizer=dict(  # Config of optimizer. Support all kinds of optimizers in␣
↪PyTorch. Refer to https://pytorch.org/docs/stable/optim.html#algorithms
```

(continues on next page)

(continued from previous page)

```
        type='SGD',  # Name of optimizer
        lr=0.01,  # Learning rate
        momentum=0.9,  # Momentum factor
        weight_decay=0.0001),  # Weight decay
    clip_grad=dict(max_norm=40, norm_type=2))  # Config of gradient clip

# runtime settings
default_scope = 'mmaction'  # The default registry scope to find modules. Refer to␣
↪https://mmengine.readthedocs.io/en/latest/tutorials/registry.html
default_hooks = dict(  # Hooks to execute default actions like updating model␣
↪parameters and saving checkpoints.
    runtime_info=dict(type='RuntimeInfoHook'),  # The hook to updates runtime␣
↪information into message hub
    timer=dict(type='IterTimerHook'),  # The logger used to record time spent␣
↪during iteration
    logger=dict(
        type='LoggerHook',  # The logger used to record logs during training/
↪validation/testing phase
        interval=20,  # Interval to print the log
        ignore_last=False), # Ignore the log of last iterations in each epoch
    param_scheduler=dict(type='ParamSchedulerHook'),  # The hook to update some␣
↪hyper-parameters in optimizer
    checkpoint=dict(
        type='CheckpointHook',  # The hook to save checkpoints periodically
        interval=3,  # The saving period
        save_best='auto',  # Specified metric to mearsure the best checkpoint␣
↪during evaluation
        max_keep_ckpts=3),  # The maximum checkpoints to keep
    sampler_seed=dict(type='DistSamplerSeedHook'),  # Data-loading sampler for␣
↪distributed training
    sync_buffers=dict(type='SyncBuffersHook'))  # Synchronize model buffers at the␣
↪end of each epoch
env_cfg = dict(  # Dict for setting environment
    cudnn_benchmark=False,  # Whether to enable cudnn benchmark
    mp_cfg=dict(mp_start_method='fork', opencv_num_threads=0), # Parameters to␣
↪setup multiprocessing
    dist_cfg=dict(backend='nccl')) # Parameters to setup distributed environment,␣
↪the port can also be set

log_processor = dict(
    type='LogProcessor',  # Log processor used to format log information
    window_size=20,  # Default smooth interval
    by_epoch=True)  # Whether to format logs with epoch type
vis_backends = [  # List of visualization backends
    dict(type='LocalVisBackend')]  # Local visualization backend
visualizer = dict(  # Config of visualizer
    type='ActionVisualizer',  # Name of visualizer
    vis_backends=vis_backends)
log_level = 'INFO'  # The level of logging
load_from = None  # Load model checkpoint as a pre-trained model from a given path.␣
↪This will not resume training.
resume = False  # Whether to resume from the checkpoint defined in `load_from`. If␣
↪`load_from` is None, it will resume the latest checkpoint in the `work_dir`.
```

(continues on next page)

(continued from previous page)

```
```

3.3.2 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',  # Class name of the detector
    _scope_='mmdet',  # The scope of current config
    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
    data_preprocessor=dict(  # Dict for data preprocessor
        type='ActionDataPreprocessor',  # Name of data preprocessor
        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
        format_shape='NCHW')),  # Final image shape format
    # 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
```

(continues on next page)

(continued from previous page)

```
                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
    test_cfg=dict(rcnn=None))  # Testing config of FastRCNN

# 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

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

train_pipeline = [  # Training data processing pipeline
    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
```

(continues on next page)

(continued from previous page)

```
        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 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(type='PackActionInputs') # Pack input data
]

val_pipeline = [  # Validation data processing pipeline
    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 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(type='PackActionInputs') # Pack input data
]

train_dataloader = dict(  # Config of train dataloader
    batch_size=32,  # Batch size of each single GPU during training
    num_workers=8,  # Workers to pre-fetch data for each single GPU during training
    persistent_workers=True,  # If `True`, the dataloader will not shut down the␣
↪worker processes after an epoch end, which can accelerate training speed
    sampler=dict(
        type='DefaultSampler',  # DefaultSampler which supports both distributed␣
↪and non-distributed training. Refer to https://github.com/open-mmlab/mmengine/
↪blob/main/mmengine/dataset/sampler.py
        shuffle=True),  # Randomly shuffle the training data in each epoch
    dataset=dict(  # Config of train dataset
        type=dataset_type,
        ann_file=ann_file_train,  # Path of annotation file
        exclude_file=exclude_file_train,  # Path of exclude annotation file
        label_file=label_file,  # Path of label file
        data_prefix=dict(img=data_root),  # Prefix of frame path
        proposal_file=proposal_file_train,  # Path of human detection proposals
        pipeline=train_pipeline))
val_dataloader = dict(  # Config of validation dataloader
    batch_size=1,  # Batch size of each single GPU during evaluation
```

(continues on next page)

(continued from previous page)

```
    num_workers=8,  # Workers to pre-fetch data for each single GPU during␣
↪evaluation
    persistent_workers=True,  # If `True`, the dataloader will not shut down the␣
↪worker processes after an epoch end
    sampler=dict(
        type='DefaultSampler',
        shuffle=False),  # Not shuffle during validation and testing
    dataset=dict(  # Config of validation dataset
        type=dataset_type,
        ann_file=ann_file_val,  # Path of annotation file
        exclude_file=exclude_file_val,  # Path of exclude annotation file
        label_file=label_file,  # Path of label file
        data_prefix=dict(img=data_root_val),  # Prefix of frame path
        proposal_file=proposal_file_val,  # Path of human detection proposals
        pipeline=val_pipeline,
        test_mode=True))
test_dataloader = val_dataloader  # Config of testing dataloader

# evaluation settings
val_evaluator = dict(  # Config of validation evaluator
    type='AVAMetric',
    ann_file=ann_file_val,
    label_file=label_file,
    exclude_file=exclude_file_val)
test_evaluator = val_evaluator  # Config of testing evaluator

train_cfg = dict(  # Config of training loop
    type='EpochBasedTrainLoop',  # Name of training loop
    max_epochs=20,  # Total training epochs
    val_begin=1,  # The epoch that begins validating
    val_interval=1)  # Validation interval
val_cfg = dict(  # Config of validation loop
    type='ValLoop')  # Name of validation loop
test_cfg = dict( # Config of testing loop
    type='TestLoop')  # Name of testing loop

# learning policy
param_scheduler = [ # Parameter scheduler for updating optimizer parameters,␣
↪support dict or list
    dict(type='LinearLR',  # Decays the learning rate of each parameter group by␣
↪linearly changing small multiplicative factor
        start_factor=0.1,  # The number we multiply learning rate in the first epoch
        by_epoch=True,  # Whether the scheduled learning rate is updated by epochs
            begin=0,  # Step at which to start updating the learning rate
            end=5),  # Step at which to stop updating the learning rate
    dict(type='MultiStepLR',  # Decays the learning rate once the number of epoch␣
↪reaches one of the milestones
        begin=0,  # Step at which to start updating the learning rate
        end=20,  # Step at which to stop updating the learning rate
        by_epoch=True,  # Whether the scheduled learning rate is updated by epochs
        milestones=[10, 15],  # Steps to decay the learning rate
        gamma=0.1)]  # Multiplicative factor of learning rate decay
```

(continues on next page)

(continued from previous page)

```
# optimizer
optim_wrapper = dict(  # Config of optimizer wrapper
    type='OptimWrapper',  # Name of optimizer wrapper, switch to AmpOptimWrapper to␣
↪enable mixed precision training
    optimizer=dict(  # Config of optimizer. Support all kinds of optimizers in␣
↪PyTorch. Refer to https://pytorch.org/docs/stable/optim.html#algorithms
        type='SGD',  # Name of optimizer
        lr=0.2,  # Learning rate
        momentum=0.9,  # Momentum factor
        weight_decay=0.0001),  # Weight decay
    clip_grad=dict(max_norm=40, norm_type=2))  # Config of gradient clip

# runtime settings
default_scope = 'mmaction'  # The default registry scope to find modules. Refer to␣
↪https://mmengine.readthedocs.io/en/latest/tutorials/registry.html
default_hooks = dict(  # Hooks to execute default actions like updating model␣
↪parameters and saving checkpoints.
    runtime_info=dict(type='RuntimeInfoHook'),  # The hook to updates runtime␣
↪information into message hub
    timer=dict(type='IterTimerHook'),  # The logger used to record time spent␣
↪during iteration
    logger=dict(
        type='LoggerHook',  # The logger used to record logs during training/
↪validation/testing phase
        interval=20,  # Interval to print the log
        ignore_last=False), # Ignore the log of last iterations in each epoch
    param_scheduler=dict(type='ParamSchedulerHook'),  # The hook to update some␣
↪hyper-parameters in optimizer
    checkpoint=dict(
        type='CheckpointHook',  # The hook to save checkpoints periodically
        interval=3,  # The saving period
        save_best='auto',  # Specified metric to mearsure the best checkpoint␣
↪during evaluation
        max_keep_ckpts=3),  # The maximum checkpoints to keep
    sampler_seed=dict(type='DistSamplerSeedHook'),  # Data-loading sampler for␣
↪distributed training
    sync_buffers=dict(type='SyncBuffersHook'))  # Synchronize model buffers at the␣
↪end of each epoch
env_cfg = dict(  # Dict for setting environment
    cudnn_benchmark=False,  # Whether to enable cudnn benchmark
    mp_cfg=dict(mp_start_method='fork', opencv_num_threads=0), # Parameters to␣
↪setup multiprocessing
    dist_cfg=dict(backend='nccl')) # Parameters to setup distributed environment,␣
↪the port can also be set

log_processor = dict(
    type='LogProcessor',  # Log processor used to format log information
    window_size=20,  # Default smooth interval
    by_epoch=True)  # Whether to format logs with epoch type
vis_backends = [  # List of visualization backends
    dict(type='LocalVisBackend')]  # Local visualization backend
```

(continues on next page)

(continued from previous page)

```
visualizer = dict(  # Config of visualizer
    type='ActionVisualizer',  # Name of visualizer
    vis_backends=vis_backends)
log_level = 'INFO'  # The level of logging
load_from = ('https://download.openmmlab.com/mmaction/v1.0/recognition/slowonly/'
             'slowonly_imagenet-pretrained-r50_8xb16-4x16x1-steplr-150e_kinetics400-
↪rgb/'
             'slowonly_imagenet-pretrained-r50_8xb16-4x16x1-steplr-150e_kinetics400-
↪rgb_20220901-e7b65fad.pth')  # Load model checkpoint as a pre-trained model from␣
↪a given path. This will not resume training.
resume = False  # Whether to resume from the checkpoint defined in `load_from`. If␣
↪`load_from` is None, it will resume the latest checkpoint in the `work_dir`.
```

3.3.3 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',  # Class name 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

# 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 = [  # Training data processing pipeline
    dict(type='LoadLocalizationFeature'),  # Load localization feature pipeline
```

(continues on next page)

(continued from previous page)

```
    dict(type='GenerateLocalizationLabels'),  # Generate localization labels␣
↪pipeline
    dict(
        type='PackLocalizationInputs', # Pack localization data
        keys=('gt_bbox'), # Keys of input
        meta_keys=('video_name'))] # Meta keys of input
val_pipeline = [  # Validation data processing pipeline
    dict(type='LoadLocalizationFeature'),  # Load localization feature pipeline
    dict(type='GenerateLocalizationLabels'),  # Generate localization labels␣
↪pipeline
    dict(
        type='PackLocalizationInputs',  # Pack localization data
        keys=('gt_bbox'),   # Keys of input
        meta_keys=('video_name', 'duration_second', 'duration_frame',
                   'annotations', 'feature_frame'))]  # Meta keys of input
test_pipeline = [  # Testing data processing pipeline
    dict(type='LoadLocalizationFeature'),  # Load localization feature pipeline
    dict(
        type='PackLocalizationInputs',  # Pack localization data
        keys=('gt_bbox'),  # Keys of input
        meta_keys=('video_name', 'duration_second', 'duration_frame',
                   'annotations', 'feature_frame'))]  # Meta keys of input
train_dataloader = dict(  # Config of train dataloader
    batch_size=8,  # Batch size of each single GPU during training
    num_workers=8,  # Workers to pre-fetch data for each single GPU during training
    persistent_workers=True,  # If `True`, the dataloader will not shut down the␣
↪worker processes after an epoch end, which can accelerate training speed
    sampler=dict(
        type='DefaultSampler',  # DefaultSampler which supports both distributed␣
↪and non-distributed training. Refer to https://github.com/open-mmlab/mmengine/
↪blob/main/mmengine/dataset/sampler.py
        shuffle=True),  # Randomly shuffle the training data in each epoch
    dataset=dict(  # Config of train dataset
        type=dataset_type,
        ann_file=ann_file_train,  # Path of annotation file
        data_prefix=dict(video=data_root),  # Prefix of video path
        pipeline=train_pipeline))
val_dataloader = dict(  # Config of validation dataloader
    batch_size=1,  # Batch size of each single GPU during evaluation
    num_workers=8,  # Workers to pre-fetch data for each single GPU during␣
↪evaluation
    persistent_workers=True,  # If `True`, the dataloader will not shut down the␣
↪worker processes after an epoch end
    sampler=dict(
        type='DefaultSampler',
        shuffle=False),  # Not shuffle during validation and testing
    dataset=dict(  # Config of validation dataset
        type=dataset_type,
        ann_file=ann_file_val,  # Path of annotation file
        data_prefix=dict(video=data_root_val),  # Prefix of video path
        pipeline=val_pipeline,
        test_mode=True))
```

(continues on next page)

(continued from previous page)

```
test_dataloader = dict(  # Config of test dataloader
    batch_size=1,  # Batch size of each single GPU during testing
    num_workers=8,  # Workers to pre-fetch data for each single GPU during testing
    persistent_workers=True,  # If `True`, the dataloader will not shut down the␣
↪worker processes after an epoch end
    sampler=dict(
        type='DefaultSampler',
        shuffle=False),  # Not shuffle during validation and testing
    dataset=dict(  # Config of test dataset
        type=dataset_type,
        ann_file=ann_file_val,  # Path of annotation file
        data_prefix=dict(video=data_root_val),  # Prefix of video path
        pipeline=test_pipeline,
        test_mode=True))

# evaluation settings
work_dir = './work_dirs/bmn_400x100_2x8_9e_activitynet_feature/'  # Directory to␣
↪save the model checkpoints and logs for the current experiments
val_evaluator = dict(
  type='ANetMetric',
  metric_type='AR@AN',
  dump_config=dict(  # Config of localization output
      out=f'{work_dir}/results.json',  # Path to the output file
      output_format='json'))  # File format of the output file
test_evaluator = val_evaluator   # Set test_evaluator as val_evaluator

max_epochs = 9  # Total epochs to train the model
train_cfg = dict(  # Config of training loop
    type='EpochBasedTrainLoop',  # Name of training loop
    max_epochs=max_epochs,  # Total training epochs
    val_begin=1,  # The epoch that begins validating
    val_interval=1)  # Validation interval
val_cfg = dict(  # Config of validation loop
    type='ValLoop')  # Name of validating loop
test_cfg = dict( # Config of testing loop
    type='TestLoop')  # Name of testing loop

# learning policy
param_scheduler = [  # Parameter scheduler for updating optimizer parameters,␣
↪support dict or list
    dict(type='MultiStepLR',  # Decays the learning rate once the number of epoch␣
↪reaches one of the milestones
    begin=0,  # Step at which to start updating the learning rate
    end=max_epochs,  # Step at which to stop updating the learning rate
    by_epoch=True,  # Whether the scheduled learning rate is updated by epochs
    milestones=[7, ],  # Steps to decay the learning rate
    gamma=0.1)]  # Multiplicative factor of parameter value decay

# optimizer
optim_wrapper = dict(  # Config of optimizer wrapper
    type='OptimWrapper',  # Name of optimizer wrapper, switch to AmpOptimWrapper to␣
↪enable mixed precision training
```

(continues on next page)

(continued from previous page)

```
    optimizer=dict(  # Config of optimizer. Support all kinds of optimizers in␣
↪PyTorch. Refer to https://pytorch.org/docs/stable/optim.html#algorithms
        type='Adam',  # Name of optimizer
        lr=0.001,  # Learning rate
        weight_decay=0.0001),  # Weight decay
    clip_grad=dict(max_norm=40, norm_type=2))  # Config of gradient clip

# runtime settings
default_scope = 'mmaction'  # The default registry scope to find modules. Refer to␣
↪https://mmengine.readthedocs.io/en/latest/tutorials/registry.html
default_hooks = dict(  # Hooks to execute default actions like updating model␣
↪parameters and saving checkpoints.
    runtime_info=dict(type='RuntimeInfoHook'),  # The hook to updates runtime␣
↪information into message hub
    timer=dict(type='IterTimerHook'),  # The logger used to record time spent␣
↪during iteration
    logger=dict(
        type='LoggerHook',  # The logger used to record logs during training/
↪validation/testing phase
        interval=20,  # Interval to print the log
        ignore_last=False), # Ignore the log of last iterations in each epoch
    param_scheduler=dict(type='ParamSchedulerHook'),  # The hook to update some␣
↪hyper-parameters in optimizer
    checkpoint=dict(
        type='CheckpointHook',  # The hook to save checkpoints periodically
        interval=3,  # The saving period
        save_best='auto',  # Specified metric to mearsure the best checkpoint␣
↪during evaluation
        max_keep_ckpts=3),  # The maximum checkpoints to keep
    sampler_seed=dict(type='DistSamplerSeedHook'),  # Data-loading sampler for␣
↪distributed training
    sync_buffers=dict(type='SyncBuffersHook'))  # Synchronize model buffers at the␣
↪end of each epoch
env_cfg = dict(  # Dict for setting environment
    cudnn_benchmark=False,  # Whether to enable cudnn benchmark
    mp_cfg=dict(mp_start_method='fork', opencv_num_threads=0), # Parameters to␣
↪setup multiprocessing
    dist_cfg=dict(backend='nccl')) # Parameters to setup distributed environment,␣
↪the port can also be set

log_processor = dict(
    type='LogProcessor',  # Log processor used to format log information
    window_size=20,  # Default smooth interval
    by_epoch=True)  # Whether to format logs with epoch type
vis_backends = [  # List of visualization backends
    dict(type='LocalVisBackend')]  # Local visualization backend
visualizer = dict(  # Config of visualizer
    type='ActionVisualizer',  # Name of visualizer
    vis_backends=vis_backends)
log_level = 'INFO'  # The level of logging
load_from = None  # Load model checkpoint as a pre-trained model from a given path.␣
↪This will not resume training.
```

(continues on next page)

(continued from previous page)

```
resume = False  # Whether to resume from the checkpoint defined in `load_from`. If␣
↪`load_from` is None, it will resume the latest checkpoint in the `work_dir`.
```

CHAPTER

FOUR

TUTORIAL 2: PREPARE DATASETS

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

- Notes on Video Data Format
- Getting Data
  - Prepare videos
  - Extract frames
    - Alternative to denseflow
  - Generate file list
  - Prepare audio

4.1 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.

4.2 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`.

4.2.1 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

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

4.2.2 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.

4.2.3 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.

4.2.4 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

FIVE

TUTORIAL 3: INFERENCE WITH EXISTING MODELS

MMAction2 provides pre-trained models for video understanding in *Model Zoo*. This note will show **how to use existing models to inference on given video**.

As for how to test existing models on standard datasets, please see this guide

5.1 Inference on a given video

MMAction2 provides high-level Python APIs for inference on a given video:

- init_recognizer: Initialize a recognizer with a config and checkpoint
- inference_recognizer: Inference on a given video

Here is an example of building the model and inference on a given video by using Kinitics-400 pre-trained checkpoint.

Note: If you use mmaction2 as a 3rd-party package, you need to download the conifg and the demo video in the example.

Run 'mim download mmaction2 –config tsn_imagenet-pretrained-r50_8xb32-1x1x8-100e_kinetics400-rgb –dest .' to download the required config.

Run 'wget https://github.com/open-mmlab/mmaction2/blob/dev-1.x/demo/demo.mp4' to download the desired demo video.

```
from mmaction.apis import inference_recognizer, init_recognizer

config_path = 'configs/recognition/tsn/tsn_imagenet-pretrained-r50_8xb32-1x1x8-100e_
↪kinetics400-rgb.py'
checkpoint_path = 'https://download.openmmlab.com/mmaction/v1.0/recognition/tsn/tsn_
↪imagenet-pretrained-r50_8xb32-1x1x8-100e_kinetics400-rgb/tsn_imagenet-pretrained-r50_
↪8xb32-1x1x8-100e_kinetics400-rgb_20220906-2692d16c.pth' # can be a local path
img_path = 'demo/demo.mp4'   # you can specify your own picture path

# build the model from a config file and a checkpoint file
model = init_recognizer(config_path, checkpoint_path, device="cpu")  # device can be
↪'cuda:0'
# test a single image
result = inference_recognizer(model, img_path)
```

`result` is a dictionary containing `pred_scores`.

An action recognition demo can be found in demo/demo.py.

CHAPTER

SIX

TUTORIAL 4: TRAINING AND TEST

6.1 Training

6.1.1 Training with your PC

You can use `tools/train.py` to train a model on a single machine with a CPU and optionally a GPU.

Here is the full usage of the script:

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

Note: By default, MMAction2 prefers GPU to CPU. If you want to train a model on CPU, please empty `CUDA_VISIBLE_DEVICES` or set it to -1 to make GPU invisible to the program.

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

6.1.2 Training with multiple GPUs

We provide a shell script to start a multi-GPUs task with `torch.distributed.launch`.

```
bash tools/dist_train.sh ${CONFIG} ${GPUS} [PY_ARGS]
```

You can also specify extra arguments of the launcher by environment variables. For example, change the communication port of the launcher to 29666 by the following command:

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

If you want to startup multiple training jobs and use different GPUs, you can launch them by specifying different port and visible devices.

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

6.1.3 Training with multiple machines

Multiple machines in the same network

If you launch a training job with multiple machines connected with ethernet, you can run the following commands:

On the first machine:

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

On the second machine:

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

The following extra environment variables need to be specified to train or test models with multiple machines:

Usually it is slow if you do not have high speed networking like InfiniBand.

Multiple machines managed with slurm

If you run MMAction2 on a cluster managed with slurm, you can use the script `slurm_train.sh`.

```
[ENV_VARS] bash tools/slurm_train.sh ${PARTITION} ${JOB_NAME} ${CONFIG} [PY_ARGS]
```

Here are the arguments description of the script.

Here are the environment variables can be used to configure the slurm job.

6.2 Test

6.2.1 Test with your PC

You can use `tools/test.py` to test a model on a single machine with a CPU and optionally a GPU.

Here is the full usage of the script:

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

Note: By default, MMAction2 prefers GPU to CPU. If you want to test a model on CPU, please empty `CUDA_VISIBLE_DEVICES` or set it to -1 to make GPU invisible to the program.

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

6.2.2 Test with multiple GPUs

We provide a shell script to start a multi-GPUs task with `torch.distributed.launch`.

```
bash tools/dist_test.sh ${CONFIG} ${CHECKPOINT} ${GPUS} [PY_ARGS]
```

You can also specify extra arguments of the launcher by environment variables. For example, change the communication port of the launcher to 29666 by the following command:

```
PORT=29666 bash tools/dist_test.sh ${CONFIG} ${CHECKPOINT} ${GPUS} [PY_ARGS]
```

If you want to startup multiple test jobs and use different GPUs, you can launch them by specifying different port and visible devices.

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

6.2.3 Test with multiple machines

Multiple machines in the same network

If you launch a test job with multiple machines connected with ethernet, you can run the following commands:

On the first machine:

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

On the second machine:

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

Compared with multi-GPUs in a single machine, you need to specify some extra environment variables:

Usually it is slow if you do not have high speed networking like InfiniBand.

Multiple machines managed with slurm

If you run MMAction2 on a cluster managed with slurm, you can use the script `slurm_test.sh`.

```
[ENV_VARS] bash tools/slurm_test.sh ${PARTITION} ${JOB_NAME} ${CONFIG} ${CHECKPOINT} [PY_
↪ARGS]
```

Here are the arguments description of the script.

Here are the environment variables can be used to configure the slurm job.

CHAPTER

SEVEN

OTHER USEFUL TOOLS

Apart from training/testing scripts, We provide lots of useful tools under the `tools/` directory.

7.1 Useful Tools Link

7.2 Model Conversion

7.2.1 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_8xb32-1x1x3-100e_kinetics400-
↪rgb/latest.pth tsn_r50_1x1x3_100e_kinetics400_rgb.pth
```

The final output filename will be `tsn_r50_8xb32-1x1x3-100e_kinetics400-rgb-{hash id}.pth`.

7.3 Miscellaneous

7.3.1 Evaluating a metric

`tools/analysis_tools/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}]
```

7.3.2 Print the entire config

`tools/analysis_tools/print_config.py` prints the whole config verbatim, expanding all its imports.

```
python tools/analysis_tools/print_config.py ${CONFIG} [-h] [--options ${OPTIONS [OPTIONS.
↪..]}]
```

7.3.3 Check videos

`tools/analysis_tools/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_tools/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]
```

7.3.4 Multi-Stream Fusion

`tools/analysis_tools/report_accuracy.py` uses the dumped results (by setting `--dump res.pkl` when testing) to fuse the multi-stream prediction scores, i.e., late fusion.

```
python tools/analysis_tools/report_accuracy.py [--preds ${RESULT_PKL_1 [RESULT_PKL_2 ...
↪]}] [--coefficients ${COEFFICIENT_1 [COEFFICIENT_2, ...]}] [--apply-softmax]
```

Take joint-bone fusion as an example, which is a general practice in the task of skeleton-based action recognition.

```
python tools/analysis_tools/report_accuracy.py --preds demo/fuse/joint.pkl demo/fuse/
↪bone.pkl --coefficients 1.0 1.0
```

Note: Mean Class Accuracy: 0.9180 Top 1 Accuracy: 0.9333 Top 5 Accuracy: 0.9833

CHAPTER

EIGHT

VISUALIZATION TOOLS

8.1 Visualize dataset

You can use tools/analysis_tools/browse_dataset.py to visualize video datasets:

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

CHAPTER

NINE

MIGRATION FROM MMACTION2 0.X

MMAction2 1.x introduced major refactorings and modifications including some BC-breaking changes. We provide this tutorial to help you migrate your projects from MMAction2 0.x smoothly.

9.1 New dependencies

MMAction2 1.x depends on the following packages. You are recommended to prepare a new clean environment and install them according to *install tutorial*

1. MMEngine: MMEngine is a foundational library for training deep learning model introduced in OpenMMLab 2.0 architecture.
2. MMCV: MMCV is a foundational library for computer vision. MMAction2 1.x requires `mmcv>=2.0.0rc0` which is more compact and efficient than `mmcv-full==1.x`.

9.2 Configuration files

In MMAction2 1.x, we refactored the structure of configuration files. The configuration files with the old style will be incompatible.

In this section, we will introduce all changes of the configuration files. And we assume you are already familiar with the config files.

9.2.1 Model settings

No changes in `model.backbone` and `model.neck`. For `model.cls_head`, we move the `average_clips` inside it, which is originally set in `model.test_cfg`.

9.2.2 Data settings

Changes in `data`

- The original `data` field is splited to `train_dataloader`, `val_dataloader` and `test_dataloader`. This allows us to configure them in fine-grained. For example, you can specify different sampler and batch size during training and test.
- The `videos_per_gpu` is renamed to `batch_size`.
- The `workers_per_gpu` is renamed to `num_workers`.

```
data = dict(
    videos_per_gpu=32,
    workers_per_gpu=2,
    train=dict(...),
    val=dict(...),
    test=dict(...),
)
```

```
train_dataloader = dict(
    batch_size=32,
    num_workers=2,
    dataset=dict(...),
    sampler=dict(type='DefaultSampler', shuffle=True)  # necessary
)

val_dataloader = dict(
    batch_size=32,
    num_workers=2,
    dataset=dict(...),
    sampler=dict(type='DefaultSampler', shuffle=False)  # necessary
)

test_dataloader = val_dataloader
```

Changes in `pipeline`

- The original formatting transforms **`ToTensor`**, **`Collect`** are combined as `PackActionInputs`.
- We don't recommend to do **`Normalize`** in the dataset pipeline. Please remove it from pipelines and set it in the `model.data_preprocessor` field.

```

train_pipeline = [
    dict(type='DecordInit'),
    dict(type='SampleFrames', clip_len=1, frame_interval=1, num_clips=8),
    dict(type='DecordDecode'),
    dict(type='Resize', scale=(-1, 256)),
    dict(
        type='MultiScaleCrop',
        input_size=224,
        scales=(1, 0.875, 0.75, 0.66),
        random_crop=False,
        max_wh_scale_gap=1),
    dict(type='Resize', scale=(224, 224), keep_ratio=False),
    dict(type='Flip', flip_ratio=0.5),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='FormatShape', input_format='NCHW'),
    dict(type='Collect', keys=['imgs', 'label'], meta_keys=[]),
    dict(type='ToTensor', keys=['imgs', 'label'])
]
```

```
model.data_preprocessor = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=False)

train_pipeline = [
    dict(type='DecordInit'),
    dict(type='SampleFrames', clip_len=1, frame_interval=1, num_clips=5),
    dict(type='DecordDecode'),
    dict(type='Resize', scale=(-1, 256)),
    dict(
        type='MultiScaleCrop',
        input_size=224,
        scales=(1, 0.875, 0.75, 0.66),
        random_crop=False,
        max_wh_scale_gap=1),
    dict(type='Resize', scale=(224, 224), keep_ratio=False),
    dict(type='Flip', flip_ratio=0.5),
    dict(type='FormatShape', input_format='NCHW'),
    dict(type='PackActionInputs')
]
```

Changes in `evaluation`

- The **`evaluation`** field is splited to `val_evaluator` and `test_evaluator`. And it won't support `interval` and `save_best` arguments.
- The `interval` is moved to `train_cfg.val_interval` and the `save_best` is moved to `default_hooks.checkpoint.save_best`.
- The 'mean_average_precision', 'mean_class_accuracy', 'mmit_mean_average_precision', 'top_k_accuracy' are combined as `AccMetric`, and you could use `metric_list` to specify which metric to calculate.
- The `AVAMetric` is used to evaluate AVA Dataset.
- The `ANetMetric` is used to evaluate ActivityNet Dataset.

```
evaluation = dict(
    interval=5,
    metrics=['top_k_accuracy', 'mean_class_accuracy'])
```

```
val_evaluator = dict(
    type='AccMetric',
    metric_list=('top_k_accuracy', 'mean_class_accuracy'))
test_evaluator = val_evaluator
```

9.2.3 Schedule settings

Changes in `optimizer` and `optimizer_config`

- Now we use `optim_wrapper` field to configure the optimization process. And the `optimizer` becomes a sub field of `optim_wrapper`.
- `paramwise_cfg` is also a sub field of `optim_wrapper` parallel to `optimizer`.
- `optimizer_config` is removed now, and all configurations of it are moved to `optim_wrapper`.
- `grad_clip` is renamed to `clip_grad`.

```
optimizer = dict(
    type='AdamW',
    lr=0.0015,
    weight_decay=0.3,
    paramwise_cfg = dict(
        norm_decay_mult=0.0,
        bias_decay_mult=0.0,
    ))

optimizer_config = dict(grad_clip=dict(max_norm=1.0))
```

```
optim_wrapper = dict(
    optimizer=dict(type='AdamW', lr=0.0015, weight_decay=0.3),
    paramwise_cfg = dict(
        norm_decay_mult=0.0,
        bias_decay_mult=0.0,
    ),
    clip_gard=dict(max_norm=1.0),
)
```

Changes in `lr_config`

- The `lr_config` field is removed and we use new `param_scheduler` to replace it.
- The `warmup` related arguments are removed, since we use schedulers combination to implement this functionality.

The new schedulers combination mechanism is very flexible, and you can use it to design many kinds of learning rate / momentum curves.

```
lr_config = dict(
    policy='CosineAnnealing',
    min_lr=0,
    warmup='linear',
    warmup_iters=5,
    warmup_ratio=0.01,
    warmup_by_epoch=True)
```

```
param_scheduler = [
    # warmup
    dict(
        type='LinearLR',
```

(continues on next page)

(continued from previous page)

```
        start_factor=0.01,
        by_epoch=True,
        end=5,
        # Update the learning rate after every iters.
        convert_to_iter_based=True),
    # main learning rate scheduler
    dict(type='CosineAnnealingLR', by_epoch=True, begin=5),
]
```

Changes in `runner`

Most configuration in the original `runner` field is moved to `train_cfg`, `val_cfg` and `test_cfg`, which configure the loop in training, validation and test.

```
runner = dict(type='EpochBasedRunner', max_epochs=100)
```

```
# The `val_interval` is the original `evaluation.interval`.
train_cfg = dict(type='EpochBasedTrainLoop', max_epochs=100, val_begin=1, val_interval=1)
val_cfg = dict(type='ValLoop')   # Use the default validation loop.
test_cfg = dict(type='TestLoop')  # Use the default test loop.
```

In fact, in OpenMMLab 2.0, we introduced `Loop` to control the behaviors in training, validation and test. And the functionalities of `Runner` are also changed. You can find more details in the MMEngine tutorials.

9.2.4 Runtime settings

Changes in `checkpoint_config` and `log_config`

The `checkpoint_config` are moved to `default_hooks.checkpoint` and the `log_config` are moved to `default_hooks.logger`. And we move many hooks settings from the script code to the `default_hooks` field in the runtime configuration.

```
default_hooks = dict(
    # update runtime information, e.g. current iter and lr.
    runtime_info=dict(type='RuntimeInfoHook'),

    # record the time of every iterations.
    timer=dict(type='IterTimerHook'),

    # print log every 100 iterations.
    logger=dict(type='LoggerHook', interval=100),

    # enable the parameter scheduler.
    param_scheduler=dict(type='ParamSchedulerHook'),

    # save checkpoint per epoch, and automatically save the best checkpoint.
    checkpoint=dict(type='CheckpointHook', interval=1, save_best='auto'),

    # set sampler seed in distributed environment.
    sampler_seed=dict(type='DistSamplerSeedHook'),
```

(continues on next page)

(continued from previous page)

```
    # synchronize model buffers at the end of each epoch.
    sync_buffers=dict(type='SyncBuffersHook')
)
```

In addition, we splited the original logger to logger and visualizer. The logger is used to record information and the visualizer is used to show the logger in different backends, like terminal, TensorBoard and Wandb.

```
log_config = dict(
    interval=100,
    hooks=[
        dict(type='TextLoggerHook'),
        dict(type='TensorboardLoggerHook'),
    ])
```

```
default_hooks = dict(
    ...
    logger=dict(type='LoggerHook', interval=100),
)

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

Changes in `load_from` and `resume_from`

- The `resume_from` is removed. And we use `resume` and `load_from` to replace it.
  - If `resume=True` and `load_from` is not None, resume training from the checkpoint in `load_from`.
  - If `resume=True` and `load_from` is None, try to resume from the latest checkpoint in the work directory.
  - If `resume=False` and `load_from` is not None, only load the checkpoint, not resume training.
  - If `resume=False` and `load_from` is None, do not load nor resume.

Changes in `dist_params`

The `dist_params` field is a sub field of `env_cfg` now. And there are some new configurations in the `env_cfg`.

```
env_cfg = dict(
    # whether to enable cudnn benchmark
    cudnn_benchmark=False,

    # set multi process parameters
    mp_cfg=dict(mp_start_method='fork', opencv_num_threads=0),

    # set distributed parameters
    dist_cfg=dict(backend='nccl'),
)
```

Changes in `workflow`

`Workflow` related functionalities are removed.

New field `visualizer`

The visualizer is a new design in OpenMMLab 2.0 architecture. We use a visualizer instance in the runner to handle results & log visualization and save to different backends.

```
visualizer = dict(
    type='ActionVisualizer',
    vis_backends=[
        dict(type='LocalVisBackend'),
        # Uncomment the below line to save the log and visualization results to␣
↪TensorBoard.
        # dict(type='TensorboardVisBackend')
    ]
)
```

New field `default_scope`

The start point to search module for all registries. The `default_scope` in MMAction2 is `mmaction`. See the registry tutorial for more details.

9.3 Packages

9.3.1 `mmaction.apis`

The documentation can be found here.

9.3.2 `mmaction.core`

The `mmaction.core` package is renamed to `mmaction.engine`.

9.3.3 `mmaction.datasets`

The documentation can be found here

Changes in `BaseActionDataset`:

Now, you can write a new Dataset class inherited from `BaseActionDataset` and overwrite `load_data_list` only. To load more data information, you could overwrite `get_data_info` like `RawframeDataset` and `AVADataset`. The `mmaction.datasets.pipelines` is renamed to `mmaction.datasets.transforms` and the `mmaction.datasets.pipelines.augmentations` is renamed to `mmaction.datasets.pipelines.processing`.

9.3.4 `mmaction.models`

The documentation can be found here. The interface of all **backbones**, **necks** and **losses** didn't change.

Changes in `BaseRecognizer`:

Changes in BaseHead:

9.3.5 `mmaction.utils`

9.3.6 Other changes

- We moved the definition of all registries in different packages to the `mmaction.registry` package.

CHAPTER

TEN

OVERVIEW

- Number of checkpoints: 179
- Number of configs: 177
- Number of papers: 33
  - ALGORITHM: 28
  - BACKBONE: 1
  - DATASET: 3
  - OTHERS: 1

For supported datasets, see datasets overview.

10.1 Spatio Temporal Action Detection Models

- Number of checkpoints: 25
- Number of configs: 27
- Number of papers: 5
  - [ALGORITHM] Ava: A Video Dataset of Spatio-Temporally Localized Atomic Visual Actions (-> -> ->)
  - [ALGORITHM] Slowfast Networks for Video Recognition (->)
  - [ALGORITHM] The Ava-Kinetics Localized Human Actions Video Dataset (->)
  - [DATASET] Ava: A Video Dataset of Spatio-Temporally Localized Atomic Visual Actions (-> -> ->)
  - [DATASET] The Ava-Kinetics Localized Human Actions Video Dataset (->)

10.2 Action Localization Models

- Number of checkpoints: 2
- Number of configs: 2
- Number of papers: 3
  - [ALGORITHM] Bmn: Boundary-Matching Network for Temporal Action Proposal Generation (->)
  - [ALGORITHM] Bsn: Boundary Sensitive Network for Temporal Action Proposal Generation (->)
  - [DATASET] Cuhk & Ethz & Siat Submission to Activitynet Challenge 2017 (->)

10.3 Action Recognition Models

- Number of checkpoints: 114
- Number of configs: 111
- Number of papers: 22
  - [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] Mvitv2: Improved Multiscale Vision Transformers for Classification and Detection (->)
  - [ALGORITHM] Non-Local Neural Networks (-> -> ->)
  - [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] Uniformer: Unified Transformer for Efficient Spatial-Temporal Representation Learning (->)
  - [ALGORITHM] Uniformerv2: Spatiotemporal Learning by Arming Image Vits With Video Uniformer (->)
  - [ALGORITHM] Video Classification With Channel-Separated Convolutional Networks (->)
  - [ALGORITHM] Video Swin Transformer (->)
  - [ALGORITHM] Video{mae (->)
  - [ALGORITHM] X3d: Expanding Architectures for Efficient Video Recognition (->)
  - [BACKBONE] Non-Local Neural Networks (-> -> ->)
  - [OTHERS] Large-Scale Weakly-Supervised Pre-Training for Video Action Recognition (->)

10.4 Skeleton-based Action Recognition Models

- Number of checkpoints: 38
- Number of configs: 37
- Number of papers: 4
  - [ALGORITHM] Pyskl: Towards Good Practices for Skeleton Action Recognition (->)
  - [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

ELEVEN

ACTION RECOGNITION MODELS

11.1 C2D

Non-local Neural Networks

11.1.1 Abstract

Both convolutional and recurrent operations are building blocks that process one local neighborhood at a time. In this paper, we present non-local operations as a generic family of building blocks for capturing long-range dependencies. Inspired by the classical non-local means method in computer vision, our non-local operation computes the response at a position as a weighted sum of the features at all positions. This building block can be plugged into many computer vision architectures. On the task of video classification, even without any bells and whistles, our non-local models can compete or outperform current competition winners on both Kinetics and Charades datasets. In static image recognition, our non-local models improve object detection/segmentation and pose estimation on the COCO suite of tasks.

11.1.2 Results and Models

Kinetics-400

1. The values in columns named after "reference" are the results reported in the original repo, using the same model settings.
2. The **gpus** indicates the number of gpus we used to get the checkpoint. If you want to use a different number of gpus or videos per gpu, the best way is to set `--auto-scale-lr` when calling `tools/train.py`, this parameter will auto-scale the learning rate according to the actual batch size and the original batch size.
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.

11.1.3 Train

You can use the following command to train a model.

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

Example: train C2D model on Kinetics-400 dataset in a deterministic option with periodic validation.

```
python tools/train.py configs/recognition/c2d/c2d_r50-in1k-pre_8xb32-8x8x1-100e_
↪kinetics400-rgb.py  \
    --seed 0 --deterministic
```

For more details, you can refer to the **Training** part in the Training and Test Tutorial.

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 C2D model on Kinetics-400 dataset and dump the result to a pkl file.

```
python tools/test.py configs/recognition/c2d/c2d_r50-in1k-pre_8xb32-8x8x1-100e_
↪kinetics400-rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

11.1.5 Citation

```
@article{XiaolongWang2017NonlocalNN,
  title={Non-local Neural Networks},
  author={Xiaolong Wang and Ross Girshick and Abhinav Gupta and Kaiming He},
  journal={arXiv: Computer Vision and Pattern Recognition},
  year={2017}
}
```

11.2 C3D

Learning Spatiotemporal Features with 3D Convolutional Networks

11.2.1 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.

11.2.2 Results and Models

UCF-101

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 gpus we used to get the checkpoint. If you want to use a different number of gpus or videos per gpu, the best way is to set `--auto-scale-lr` when calling `tools/train.py`, this parameter will auto-scale the learning rate according to the actual batch size and the original batch size.

For more details on data preparation, you can refer to UCF101.

11.2.3 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-pretrained_8xb30-16x1x1-45e_
↪ucf101-rgb.py \
    --seed=0 --deterministic
```

For more details, you can refer to the **Training** part in the Training and Test Tutorial.

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 C3D model on UCF-101 dataset and dump the result to a pkl file.

```
python tools/test.py configs/recognition/c3d_sports1m-pretrained_8xb30-16x1x1-45e_ucf101-
↪rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

11.2.5 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}
}
```

11.3 CSN

Video Classification With Channel-Separated Convolutional Networks

11.3.1 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.

11.3.2 Results and Models

Kinetics-400

1. The **gpus** indicates the number of gpus we used to get the checkpoint. If you want to use a different number of gpus or videos per gpu, the best way is to set `--auto-scale-lr` when calling `tools/train.py`, this parameter will auto-scale the learning rate according to the actual batch size and the original batch size.
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.
3. The **infer_ckpt** means those checkpoints are ported from VMZ.

For more details on data preparation, you can refer to Kinetics400.

11.3.3 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_8xb12-32x2x1-
↪58e_kinetics400-rgb.py  \
    --seed=0 --deterministic
```

For more details, you can refer to the **Training** part in the Training and Test Tutorial.

11.3.4 Test

You can use the following command to test a model.

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

Example: test CSN model on Kinetics-400 dataset and dump the result to a pkl file.

```
python tools/test.py configs/recognition/csn/ircsn_ig65m-pretrained-r152_8xb12-32x2x1-
↪58e_kinetics400-rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

11.3.5 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}
}
```

11.4 I3D

Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset

Non-local Neural Networks

11.4.1 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.

11.4.2 Results and Models

Kinetics-400

1. The **gpus** indicates the number of gpus we used to get the checkpoint. If you want to use a different number of gpus or videos per gpu, the best way is to set `--auto-scale-lr` when calling `tools/train.py`, this parameter will auto-scale the learning rate according to the actual batch size and the original batch size.
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.

11.4.3 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_imagenet-pretrained-r50_8xb8-32x2x1-
↪100e_kinetics400-rgb.py \
    --seed=0 --deterministic
```

For more details, you can refer to the **Training** part in the Training and Test Tutorial.

11.4.4 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 pkl file.

```
python tools/test.py configs/recognition/i3d/i3d_imagenet-pretrained-r50_8xb8-32x2x1-
↪100e_kinetics400-rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

11.4.5 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}
}
```

11.5 MViT V2

MViTv2: Improved Multiscale Vision Transformers for Classification and Detection

11.5.1 Abstract

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

11.5.2 Results and Models

1. Models with * in `Inference results` are ported from the repo SlowFast and tested on our data, and models in `Training results` are trained in MMAction2 on our data.
2. The values in columns named after `reference` are copied from paper, and `reference*` are results using SlowFast repo and trained on our data.
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. MaskFeat fine-tuning experiment is based on pretrain model from MMSelfSup, and the corresponding reference result is based on pretrain model from SlowFast.
5. Due to the different versions of Kinetics-400, our training results are different from paper.
6. Due to the training efficiency, we currently only provide MViT-small training results, we don't ensure other config files' training accuracy and welcome you to contribute your reproduction results.
7. We use `repeat augment` in MViT training configs following SlowFast. Repeat augment takes multiple times of data augment for one video, this way can improve the generalization of the model and relieve the IO stress of loading videos. And please note that the actual batch size is `num_repeats` times of `batch_size` in `train_dataloader`.

Inference results

Kinetics-400

Something-Something V2

Training results

Kinetics-400

the corresponding result without repeat augment is as follows:

Something-Something V2

For more details on data preparation, you can refer to

- Kinetics
- Something-something V2

11.5.3 Test

You can use the following command to test a model.

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

Example: test MViT model on Kinetics-400 dataset and dump the result to a pkl file.

```
python tools/test.py configs/recognition/mvit/mvit-small-p244_16x4x1_kinetics400-rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

11.5.4 Citation

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

11.6 Omnisource

11.6.1 Abstract

We propose to train a recognizer that can classify images and videos. The recognizer is jointly trained on image and video datasets. Compared with pre-training on the same image dataset, this method can significantly improve the video recognition performance.

11.6.2 Results and Models

Kinetics-400

1. The **gpus** indicates the number of gpus we used to get the checkpoint. If you want to use a different number of gpus or videos per gpu, the best way is to set `--auto-scale-lr` when calling `tools/train.py`, this parameter will auto-scale the learning rate according to the actual batch size and the original batch size.
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.

11.6.3 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/omnisource/slowonly_r50_8xb16-8x8x1-256e_
↪imagenet-kinetics400-rgb.py \
    --seed=0 --deterministic
```

We found that the training of this Omnisource model could crash for unknown reasons. If this happens, you can resume training by adding the `--cfg-options resume=True` to the training script.

For more details, you can refer to the **Training** part in the Training and Test Tutorial.

11.6.4 Test

You can use the following command to test a model.

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

Example: test SlowOnly model on Kinetics-400 dataset and dump the result to a pkl file.

```
python tools/test.py configs/recognition/omnisource/slowonly_r50_8xb16-8x8x1-256e_
↪imagenet-kinetics400-rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

11.6.5 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}
}
```

11.7 R2plus1D

A closer look at spatiotemporal convolutions for action recognition

11.7.1 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.

11.7.2 Results and Models

Kinetics-400

1. The **gpus** indicates the number of gpus we used to get the checkpoint. If you want to use a different number of gpus or videos per gpu, the best way is to set `--auto-scale-lr` when calling `tools/train.py`, this parameter will auto-scale the learning rate according to the actual batch size and the original batch size.
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.

11.7.3 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.

```
python tools/train.py configs/recognition/r2plus1d/r2plus1d_r34_8xb8-8x8x1-180e_
↪kinetics400-rgb.py \
    --seed=0 --deterministic
```

For more details, you can refer to the **Training** part in the Training and Test Tutorial.

11.7.4 Test

You can use the following command to test a model.

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

Example: test R(2+1)D model on Kinetics-400 dataset and dump the result to a pkl file.

```
python tools/test.py configs/recognition/r2plus1d/r2plus1d_r34_8xb8-8x8x1-180e_
↪kinetics400-rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

11.7.5 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}
}
```

11.8 SlowFast

SlowFast Networks for Video Recognition

11.8.1 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.

11.8.2 Results and Models

Kinetics-400

1. The **gpus** indicates the number of gpus we used to get the checkpoint. If you want to use a different number of gpus or videos per gpu, the best way is to set `--auto-scale-lr` when calling `tools/train.py`, this parameter will auto-scale the learning rate according to the actual batch size and the original batch size.
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.

11.8.3 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_8xb8-4x16x1-256e_
↪kinetics400-rgb.py \
    --seed=0 --deterministic
```

For more details, you can refer to the **Training** part in the Training and Test Tutorial.

11.8.4 Test

You can use the following command to test a model.

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

Example: test SlowFast model on Kinetics-400 dataset and dump the result to a pkl file.

```
python tools/test.py configs/recognition/slowfast/slowfast_r50_8xb8-4x16x1-256e_
↪kinetics400-rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

11.8.5 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}
}
```

11.9 SlowOnly

Slowfast networks for video recognition

11.9.1 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.

11.9.2 Results and Models

Kinetics-400

Kinetics-700

1. The **gpus** indicates the number of gpus we used to get the checkpoint. If you want to use a different number of gpus or videos per gpu, the best way is to set `--auto-scale-lr` when calling `tools/train.py`, this parameter will auto-scale the learning rate according to the actual batch size and the original batch size.

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.

11.9.3 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_8xb16-4x16x1-256e_
↪kinetics400-rgb.py \
    --seed=0 --deterministic
```

For more details, you can refer to the **Training** part in the Training and Test Tutorial.

11.9.4 Test

You can use the following command to test a model.

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

Example: test SlowOnly model on Kinetics-400 dataset and dump the result to a pkl file.

```
python tools/test.py configs/recognition/slowonly/slowonly_r50_8xb16-4x16x1-256e_
↪kinetics400-rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

11.9.5 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}
}
```

11.10 VideoSwin

Video Swin Transformer

11.10.1 Abstract

The vision community is witnessing a modeling shift from CNNs to Transformers, where pure Transformer architectures have attained top accuracy on the major video recognition benchmarks. These video models are all built on Transformer layers that globally connect patches across the spatial and temporal dimensions. In this paper, we instead advocate an inductive bias of locality in video Transformers, which leads to a better speed-accuracy trade-off compared to previous approaches which compute self-attention globally even with spatial-temporal factorization. The locality of the proposed video architecture is realized by adapting the Swin Transformer designed for the image domain, while continuing to leverage the power of pre-trained image models. Our approach achieves state-of-the-art accuracy on a broad range of video recognition benchmarks, including on action recognition (84.9 top-1 accuracy on Kinetics-400 and 85.9 top-1 accuracy on Kinetics-600 with ~20xless pre-training data and ~3xsmaller model size) and temporal modeling (69.6 top-1 accuracy on Something-Something v2).

11.10.2 Results and Models

Kinetics-400

Kinetics-700

1. The **gpus** indicates the number of gpus we used to get the checkpoint. If you want to use a different number of gpus or videos per gpu, the best way is to set `--auto-scale-lr` when calling `tools/train.py`, this parameter will auto-scale the learning rate according to the actual batch size and the original batch size.
2. 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. * means that the numbers are copied from the paper.
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. Pre-trained image models can be downloaded from Swin Transformer for ImageNet Classification.

For more details on data preparation, you can refer to Kinetics.

11.10.3 Train

You can use the following command to train a model.

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

Example: train VideoSwin model on Kinetics-400 dataset in a deterministic option with periodic validation.

```
python tools/train.py configs/recognition/swin/swin-tiny-p244-w877_in1k-pre_8xb8-amp-
↪32x2x1-30e_kinetics400-rgb.py \
    --seed=0 --deterministic
```

For more details, you can refer to the **Training** part in the Training and Test Tutorial.

11.10.4 Test

You can use the following command to test a model.

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

Example: test VideoSwin model on Kinetics-400 dataset and dump the result to a pkl file.

```
python tools/test.py configs/recognition/swin/swin-tiny-p244-w877_in1k-pre_8xb8-amp-
↪32x2x1-30e_kinetics400-rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

11.10.5 Citation

```
@inproceedings{liu2022video,
  title={Video swin transformer},
  author={Liu, Ze and Ning, Jia and Cao, Yue and Wei, Yixuan and Zhang, Zheng and Lin,␣
↪Stephen and Hu, Han},
  booktitle={Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern␣
↪Recognition},
  pages={3202--3211},
  year={2022}
}
```

11.11 TANet

TAM: Temporal Adaptive Module for Video Recognition

11.11.1 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.

11.11.2 Results and Models

Kinetics-400

Something-Something V1

1. The **gpus** indicates the number of gpus we used to get the checkpoint. If you want to use a different number of gpus or videos per gpu, the best way is to set `--auto-scale-lr` when calling `tools/train.py`, this parameter will auto-scale the learning rate according to the actual batch size and the original batch size.
2. 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.
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
- Something-something V1

11.11.3 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_imagenet-pretrained-r50_8xb8-dense-
↪1x1x8-100e_kinetics400-rgb.py \
    --seed=0 --deterministic
```

For more details, you can refer to the **Training** part in the Training and Test Tutorial.

11.11.4 Test

You can use the following command to test a model.

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

Example: test TANet model on Kinetics-400 dataset and dump the result to a pkl file.

```
python tools/test.py configs/recognition/tanet/tanet_imagenet-pretrained-r50_8xb8-dense-
↪1x1x8-100e_kinetics400-rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

11.11.5 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}
}
```

11.12 TimeSformer

Is Space-Time Attention All You Need for Video Understanding?

11.12.1 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).

11.12.2 Results and Models

Kinetics-400

1. The **gpus** indicates the number of gpus we used to get the checkpoint. If you want to use a different number of gpus or videos per gpu, the best way is to set `--auto-scale-lr` when calling `tools/train.py`, this parameter will auto-scale the learning rate according to the actual batch size and the original batch size.
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.

11.12.3 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_8xb8-8x32x1-15e_
↪kinetics400-rgb.py \
    --seed=0 --deterministic
```

For more details, you can refer to the **Training** part in the Training and Test Tutorial.

11.12.4 Test

You can use the following command to test a model.

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

Example: test TimeSformer model on Kinetics-400 dataset and dump the result to a pkl file.

```
python tools/test.py configs/recognition/timesformer/timesformer_divST_8xb8-8x32x1-15e_
↪kinetics400-rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

11.12.5 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}
}
```

11.13 TIN

Temporal Interlacing Network

11.13.1 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.

11.13.2 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 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 Kinetics400, Something-Something V1 and Something-Something V2 in Prepare Datasets.

11.13.3 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_imagenet-pretrained-r50_8xb6-1x1x8-40e_
↪sthv1-rgb.py \
    --work-dir work_dirs/tin_imagenet-pretrained-r50_8xb6-1x1x8-40e_sthv1-rgb randomness.
↪seed=0 randomness.deterministic=True
```

For more details, you can refer to the **Training** part in the Training and Test Tutorial.

11.13.4 Test

You can use the following command to test a model.

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

Example: test TIN model on Something-Something V1 dataset and dump the result to a json file.

```
python tools/test.py configs/recognition/tin/tin_imagenet-pretrained-r50_8xb6-1x1x8-40e_
↪sthv1-rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.json
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

11.13.5 Citation

```
@article{shao2020temporal,
    title={Temporal Interlacing Network},
    author={Hao Shao and Shengju Qian and Yu Liu},
    year={2020},
    journal={AAAI},
}
```

11.14 TPN

Temporal Pyramid Network for Action Recognition

11.14.1 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.

11.14.2 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 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.
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, Something-Something V1 and Something-Something V2 in Data Preparation.

11.14.3 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_8xb8-8x8x1-150e_
↪kinetics400-rgb.py \
    --work-dir work_dirs/tpn-slowonly_r50_8xb8-8x8x1-150e_kinetics400-rgb [--validate --
↪seed 0 --deterministic]
```

For more details, you can refer to the **Training** part in the Training and Test Tutorial.

11.14.4 Test

You can use the following command to test a model.

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

Example: test TPN model on Kinetics-400 dataset and dump the result to a json file.

```
python tools/test.py configs/recognition/tpn/tpn-slowonly_r50_8xb8-8x8x1-150e_
↪kinetics400-rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

11.14.5 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},
}
```

11.15 TRN

Temporal Relational Reasoning in Videos

11.15.1 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.

11.15.2 Results and Models

Something-Something V1

Something-Something V2

1. The **gpus** indicates the number of gpus we used to get the checkpoint. If you want to use a different number of gpus or videos per gpu, the best way is to set `--auto-scale-lr` when calling `tools/train.py`, this parameter will auto-scale the learning rate according to the actual batch size and the original batch size.
2. There are two kinds of test settings for Something-Something dataset, efficient setting (center crop only) and accurate setting (three crop and `twice_sample`).
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 31.81 (60.47).

For more details on data preparation, you can refer to Something-something V1 and Something-something V2.

11.15.3 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_imagenet-pretrained-r50_8xb16-1x1x8-
↪50e_sthv1-rgb.py \
    --seed=0 --deterministic
```

For more details, you can refer to the **Training** part in the Training and Test Tutorial.

11.15.4 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 pkl file.

```
python tools/test.py configs/recognition/trn/trn_imagenet-pretrained-r50_8xb16-1x1x8-50e_
↪sthv1-rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

11.15.5 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}
}
```

11.16 TSM

TSM: Temporal Shift Module for Efficient Video Understanding

11.16.1 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.

11.16.2 Results and Models

Kinetics-400

Something-something V2

1. The **gpus** indicates the number of gpus we used to get the checkpoint. If you want to use a different number of gpus or videos per gpu, the best way is to set `--auto-scale-lr` when calling `tools/train.py`, this parameter will auto-scale the learning rate according to the actual batch size and the original batch size.
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.

11.16.3 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.

```
python tools/train.py configs/recognition/tsm/tsm_imagenet-pretrained-r50_8xb16-1x1x8-
↪50e_kinetics400-rgb.py \
    --seed=0 --deterministic
```

For more details, you can refer to the **Training** part in the Training and Test Tutorial.

11.16.4 Test

You can use the following command to test a model.

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

Example: test TSM model on Kinetics-400 dataset and dump the result to a pkl file.

```
python tools/test.py configs/recognition/tsm/tsm_imagenet-pretrained-r50_8xb16-1x1x8-50e_
↪kinetics400-rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

11.16.5 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}
}
```

11.17 TSN

Temporal segment networks: Towards good practices for deep action recognition

11.17.1 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.

11.17.2 Results and Models

Kinetics-400

Something-Something V2

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.
2. The **gpus** indicates the number of gpus we used to get the checkpoint. If you want to use a different number of gpus or videos per gpu, the best way is to set `--auto-scale-lr` when calling `tools/train.py`, this parameter will auto-scale the learning rate according to the actual batch size and the original batch size.
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

- Kinetics
- Something-something V2

11.17.3 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.

```
python tools/train.py configs/recognition/tsn/tsn_imagenet-pretrained-r50_8xb32-1x1x3-
↪100e_kinetics400-rgb.py  \
    --seed=0 --deterministic
```

For more details, you can refer to the **Training** part in the Training and Test Tutorial.

11.17.4 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 pkl file.

```
python tools/test.py configs/recognition/tsn/tsn_imagenet-pretrained-r50_8xb32-1x1x3-
↪100e_kinetics400-rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

11.17.5 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}
}
```

11.18 UniFormer

UniFormer: Unified Transformer for Efficient Spatiotemporal Representation Learning

11.18.1 Abstract

It is a challenging task to learn rich and multi-scale spatiotemporal semantics from high-dimensional videos, due to large local redundancy and complex global dependency between video frames. The recent advances in this research have been mainly driven by 3D convolutional neural networks and vision transformers. Although 3D convolution can efficiently aggregate local context to suppress local redundancy from a small 3D neighborhood, it lacks the capability to capture global dependency because of the limited receptive field. Alternatively, vision transformers can effectively capture long-range dependency by self-attention mechanism, while having the limitation on reducing local redundancy with blind similarity comparison among all the tokens in each layer. Based on these observations, we propose a novel Unified transFormer (UniFormer) which seamlessly integrates merits of 3D convolution and spatiotemporal self-attention in a concise transformer format, and achieves a preferable balance between computation and accuracy. Different from traditional transformers, our relation aggregator can tackle both spatiotemporal redundancy and dependency, by learning local and global token affinity respectively in shallow and deep layers. We conduct extensive experiments on the popular video benchmarks, e.g., Kinetics-400, Kinetics-600, and Something-Something V1&V2. With only ImageNet-1K pretraining, our UniFormer achieves 82.9%/84.8% top-1 accuracy on Kinetics-400/Kinetics-600, while requiring 10x fewer GFLOPs than other state-of-the-art methods. For Something-Something V1 and V2, our UniFormer achieves new state-of-the-art performances of 60.9% and 71.2% top-1 accuracy respectively.

11.18.2 Results and Models

Kinetics-400

The models are ported from the repo UniFormer and tested on our data. Currently, we only support the testing of UniFormer models, training will be available soon.

1. The values in columns named after "reference" are the results of the original repo.
2. The values in `top1/5 acc` is tested on the same data list as the original repo, and the label map is provided by UniFormer. The total videos are available at Kinetics400 (BaiduYun password: g5kp), which consists of 19787 videos.
3. The values in columns named after "mm-Kinetics" are the testing results on the Kinetics dataset held by MMAction2, which is also used by other models in MMAction2. Due to the differences between various versions of Kinetics dataset, there is a little gap between `top1/5 acc` and `mm-Kinetics top1/5 acc`. For a fair comparison with other models, we report both results here. Note that we simply report the inference results, since the training set is different between UniFormer and other models, the results are lower than that tested on the author's version.
4. Since the original models for Kinetics-400/600/700 adopt different label file, we simply map the weight according to the label name. New label map for Kinetics-400/600/700 can be found here.
5. Due to some difference between SlowFast and MMAction, there are some gaps between their performances.

For more details on data preparation, you can refer to preparing_kinetics.

11.18.3 Test

You can use the following command to test a model.

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

Example: test UniFormer-S model on Kinetics-400 dataset and dump the result to a pkl file.

```
python tools/test.py configs/recognition/uniformer/uniformer-small_imagenet1k-pre_16x4x1_
↪kinetics400-rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

11.18.4 Citation

```
@inproceedings{
  li2022uniformer,
  title={UniFormer: Unified Transformer for Efficient Spatial-Temporal Representation␣
↪Learning},
  author={Kunchang Li and Yali Wang and Gao Peng and Guanglu Song and Yu Liu and␣
↪Hongsheng Li and Yu Qiao},
  booktitle={International Conference on Learning Representations},
  year={2022},
  url={https://openreview.net/forum?id=nBU_u6DLvoK}
}
```

11.19 UniFormerV2

UniFormerV2: Spatiotemporal Learning by Arming Image ViTs with Video UniFormer

11.19.1 Abstract

Learning discriminative spatiotemporal representation is the key problem of video understanding. Recently, Vision Transformers (ViTs) have shown their power in learning long-term video dependency with self-attention. Unfortunately, they exhibit limitations in tackling local video redundancy, due to the blind global comparison among tokens. UniFormer has successfully alleviated this issue, by unifying convolution and self-attention as a relation aggregator in the transformer format. However, this model has to require a tiresome and complicated image-pretraining phrase, before being finetuned on videos. This blocks its wide usage in practice. On the contrary, open-sourced ViTs are readily available and well-pretrained with rich image supervision. Based on these observations, we propose a generic paradigm to build a powerful family of video networks, by arming the pretrained ViTs with efficient UniFormer designs. We call this family UniFormerV2, since it inherits the concise style of the UniFormer block. But it contains brand-new local and global relation aggregators, which allow for preferable accuracy-computation balance by seamlessly integrating advantages from both ViTs and UniFormer. Without any bells and whistles, our UniFormerV2 gets the state-of-the-art recognition performance on 8 popular video benchmarks, including scene-related Kinetics-400/600/700 and Moments in Time, temporal-related Something-Something V1/V2, untrimmed ActivityNet and HACS. In particular, it is the first model to achieve 90% top-1 accuracy on Kinetics-400, to our best knowledge.

11.19.2 Results and Models

Kinetics-400

Kinetics-600

Kinetics-700

MiTv1

Kinetics-710

The models are ported from the repo UniFormerV2 and tested on our data. Currently, we only support the testing of UniFormerV2 models, training will be available soon.

1. The values in columns named after "reference" are the results of the original repo.
2. The values in `top1/5 acc` is tested on the same data list as the original repo, and the label map is provided by UniFormerV2.
3. The values in columns named after "mm-Kinetics" are the testing results on the Kinetics dataset held by MMAction2, which is also used by other models in MMAction2. Due to the differences between various versions of Kinetics dataset, there is a little gap between `top1/5 acc` and `mm-Kinetics top1/5 acc`. For a fair comparison with other models, we report both results here. Note that we simply report the inference results, since the training set is different between UniFormer and other models, the results are lower than that tested on the author's version.
4. Since the original models for Kinetics-400/600/700 adopt different label file, we simply map the weight according to the label name. New label map for Kinetics-400/600/700 can be found here.
5. Due to some differences between SlowFast and MMAction2, there are some gaps between their performances.

6. Kinetics-710 is used for pretraining, which helps improve the performance on other datasets efficiently. You can find more details in the paper.

For more details on data preparation, you can refer to

- preparing_kinetics
- preparing_mit

11.19.3 Test

You can use the following command to test a model.

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

Example: test UniFormerV2-B/16 model on Kinetics-400 dataset and dump the result to a pkl file.

```
python tools/test.py configs/recognition/uniformerv2/uniformerv2-base-p16-res224_clip-
↪kinetics710-pre_u8_kinetics400-rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

11.19.4 Citation

```
@article{Li2022UniFormerV2SL,
  title={UniFormerV2: Spatiotemporal Learning by Arming Image ViTs with Video UniFormer},
  author={Kunchang Li and Yali Wang and Yinan He and Yizhuo Li and Yi Wang and Limin␣
↪Wang and Y. Qiao},
  journal={ArXiv},
  year={2022},
  volume={abs/2211.09552}
}
```

11.20 VideoMAE

VideoMAE: Masked Autoencoders are Data-Efficient Learners for Self-Supervised Video Pre-Training

11.20.1 Abstract

Pre-training video transformers on extra large-scale datasets is generally required to achieve premier performance on relatively small datasets. In this paper, we show that video masked autoencoders (VideoMAE) are data-efficient learners for self-supervised video pre-training (SSVP). We are inspired by the recent ImageMAE and propose customized video tube masking with an extremely high ratio. This simple design makes video reconstruction a more challenging self-supervision task, thus encouraging extracting more effective video representations during this pre-training process. We obtain three important findings on SSVP: (1) An extremely high proportion of masking ratio (i.e., 90% to 95%) still yields favorable performance of VideoMAE. The temporally redundant video content enables a higher masking ratio than that of images. (2) VideoMAE achieves impressive results on very small datasets (i.e., around 3k-4k videos) without using any extra data. (3) VideoMAE shows that data quality is more important than data quantity for SSVP. Domain shift between pre-training and target datasets is an important issue. Notably, our VideoMAE with the vanilla

ViT can achieve 87.4% on Kinetics-400, 75.4% on Something-Something V2, 91.3% on UCF101, and 62.6% on HMDB51, without using any extra data.

11.20.2 Results and Models

Kinetics-400

[1] The models are ported from the repo VideoMAE and tested on our data. Currently, we only support the testing of VideoMAE models, training will be available soon.

1. The values in columns named after "reference" are the results of the original repo.
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 preparing_kinetics.

11.20.3 Test

You can use the following command to test a model.

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

Example: test ViT-base model on Kinetics-400 dataset and dump the result to a pkl file.

```
python tools/test.py configs/recognition/videomae/vit-base-p16_videomae-k400-pre_16x4x1_
↪kinetics-400.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

11.20.4 Citation

```
@inproceedings{tong2022videomae,
  title={Video{MAE}: Masked Autoencoders are Data-Efficient Learners for Self-Supervised␣
↪Video Pre-Training},
  author={Zhan Tong and Yibing Song and Jue Wang and Limin Wang},
  booktitle={Advances in Neural Information Processing Systems},
  year={2022}
}
```

11.21 X3D

X3D: Expanding Architectures for Efficient Video Recognition

11.21.1 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.

11.21.2 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.

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 is same as the repo SlowFast, which is available here.

For more details on data preparation, you can refer to Kinetics400.

11.21.3 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 pkl file.

```
python tools/test.py configs/recognition/x3d/x3d_s_13x6x1_facebook-kinetics400-rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

11.21.4 Citation

```
@misc{feichtenhofer2020x3d,
      title={X3D: Expanding Architectures for Efficient Video Recognition},
      author={Christoph Feichtenhofer},
      year={2020},
      eprint={2004.04730},
      archivePrefix={arXiv},
      primaryClass={cs.CV}
}
```

11.22 ResNet for Audio

Audiovisual SlowFast Networks for Video Recognition

11.22.1 Abstract

We present Audiovisual SlowFast Networks, an architecture 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.

11.22.2 Results and Models

Kinetics-400

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 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 Tutorial.

11.22.3 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/recognition_audio/resnet/tsn_r18_8xb320-64x1x1-100e_
↪kinetics400-audio-feature.py \
    --cfg-options randomness.seed=0 randomness.deterministic=True
```

For more details, you can refer to the **Training** part in the Training and Test Tutorial.

11.22.4 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 pkl file.

```
python tools/test.py configs/recognition_audio/resnet/tsn_r18_8xb320-64x1x1-100e_
↪kinetics400-audio-feature.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

11.22.5 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

TWELVE

SPATIO TEMPORAL ACTION DETECTION MODELS

12.1 ACRN

Actor-centric relation network

12.1.1 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.

12.1.2 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 to AVA Data Preparation.

12.1.3 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 in a deterministic option.

```
python tools/train.py configs/detection/acrn/slowfast-acrn_kinetics400-pretrained-r50_
↪8xb8-8x8x1-cosine-10e_ava21-rgb.py \
    --cfg-options randomness.seed=0 randomness.deterministic=True
```

For more details and optional arguments infos, you can refer to the **Training** part in the Training and Test Tutorial.

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 ACRN with SlowFast backbone on AVA and dump the result to a pkl file.

```
python tools/test.py configs/detection/acrn/slowfast-acrn_kinetics400-pretrained-r50_
↪8xb8-8x8x1-cosine-10e_ava21-rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details and optional arguments infos, you can refer to the **Test** part in the Training and Test Tutorial.

12.1.5 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}
}
```

12.2 AVA

Ava: A video dataset of spatio-temporally localized atomic visual actions

12.2.1 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}
}
```

12.2.2 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. **With 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 Data Preparation](https://
↪github.com/open-mmlab/mmaction2/tree/master/tools/data/ava/README.md).

### Train

You can use the following command to train a model.

```

(continues on next page)

(continued from previous page)

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

Example: train the SlowOnly model on AVA in a deterministic option.

```shell
python tools/train.py configs/detection/ava/slowonly_kinetics400-pretrained-r50_8xb16-
↪4x16x1-20e_ava21-rgb.py \
    --cfg-options randomness.seed=0 randomness.deterministic=True
```

For more details, you can refer to the **Training** part in the [Training and Test␣
↪Tutorial](en/user_guides/4_train_test.md).

### Test

You can use the following command to test a model.

```shell
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [optional arguments]
```

Example: test the SlowOnly model on AVA and dump the result to a pkl file.

```shell
python tools/test.py configs/detection/ava/slowonly_kinetics400-pretrained-r50_8xb16-
↪4x16x1-20e_ava21-rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the [Training and Test␣
↪Tutorial](en/user_guides/4_train_test.md).

### Citation

<!-- [DATASET] -->

```BibTeX
@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}
}
```

```BibTeX
~~~

(continues on next page)

(continued from previous page)

~~~
@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}
}
```

## AVA

[The AVA-Kinetics Localized Human Actions Video Dataset](https://arxiv.org/abs/2005.
↪00214)

<!-- [ALGORITHM] -->

<div align="center">
  <img src="https://user-images.githubusercontent.com/35267818/205511687-8cafd48c-7f4a-
↪4a4c-a8e6-8182635b0411.png" width="800px"/>
</div>

### Abstract

<!-- [ABSTRACT] -->

This paper describes the AVA-Kinetics localized human actions video dataset. The dataset␣
↪is collected by annotating videos from the Kinetics-700 dataset using the AVA␣
↪annotation protocol, and extending the original AVA dataset with these new AVA␣
↪annotated Kinetics clips. The dataset contains over 230k clips annotated with the 80␣
↪AVA action classes for each of the humans in key-frames. We describe the annotation␣
↪process and provide statistics about the new dataset. We also include a baseline␣
↪evaluation using the Video Action Transformer Network on the AVA-Kinetics dataset,␣
↪demonstrating improved performance for action classification on the AVA test set.

```BibTeX
@article{li2020ava,
  title={The ava-kinetics localized human actions video dataset},
  author={Li, Ang and Thotakuri, Meghana and Ross, David A and Carreira, Jo{\~a}o and␣
↪Vostrikov, Alexander and Zisserman, Andrew},
  journal={arXiv preprint arXiv:2005.00214},
  year={2020}
}
```


### Results and Models

#### AVA2.2

Currently, we only use the training set of AVA-Kinetics and evaluate on the AVA2.2␣
↪validation dataset. The AVA-Kinetics validation dataset will be supported soon.

<table border="1" class="docutils">
<thead>
<tr>
~~~

(continues on next page)

(continued from previous page)

```
<th style="text-align: center;">frame sampling strategy</th>
<th style="text-align: center;">resolution</th>
<th style="text-align: center;">gpus</th>
<th style="text-align: center;">backbone</th>
<th style="text-align: center;">pretrain</th>
<th style="text-align: center;">mAP</th>
<th style="text-align: center;">config</th>
<th style="text-align: center;">ckpt</th>
<th style="text-align: center;">log</th>
</tr>
</thead>
<tbody>
<tr>
<td style="text-align: center;">4x16x1</td>
<td style="text-align: center;">raw</td>
<td style="text-align: center;">8</td>
<td style="text-align: center;">SlowOnly ResNet50</td>
<td style="text-align: center;">Kinetics-400</td>
<td style="text-align: center;">24.53</td>
<td style="text-align: center;"><a href="https://github.com/open-mmlab/mmaction2/tree/
↪master/configs/detection/ava_kinetics/slowonly_k400-pre-r50_8xb8-4x16x1-10e_ava-
↪kinetics-rgb.py">config</a></td>
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/ava_kinetics/slowonly_k400-pre-r50_8xb8-4x16x1-10e_ava-kinetics-rgb/slowonly_
↪k400-pre-r50_8xb8-4x16x1-10e_ava-kinetics-rgb_20221205-33e3ca7c.pth">ckpt</a></td>
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/ava_kinetics/slowonly_k400-pre-r50_8xb8-4x16x1-10e_ava-kinetics-rgb/slowonly_
↪k400-pre-r50_8xb8-4x16x1-10e_ava-kinetics-rgb.log">log</a></td>
</tr>
<tr>
<td style="text-align: center;">4x16x1</td>
<td style="text-align: center;">raw</td>
<td style="text-align: center;">8</td>
<td style="text-align: center;">SlowOnly ResNet50</td>
<td style="text-align: center;">Kinetics-700</td>
<td style="text-align: center;">25.87</td>
<td style="text-align: center;"><a href="https://github.com/open-mmlab/mmaction2/tree/
↪master/configs/detection/ava_kinetics/slowonly_k700-pre-r50_8xb8-4x16x1-10e_ava-
↪kinetics-rgb.py">config</a></td>
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/ava_kinetics/slowonly_k700-pre-r50_8xb8-4x16x1-10e_ava-kinetics-rgb/slowonly_
↪k700-pre-r50_8xb8-4x16x1-10e_ava-kinetics-rgb_20221205-a07e8c15.pth">ckpt</a></td>
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/ava_kinetics/slowonly_k700-pre-r50_8xb8-4x16x1-10e_ava-kinetics-rgb/slowonly_
↪k700-pre-r50_8xb8-4x16x1-10e_ava-kinetics-rgb.log">log</a></td>
</tr>
<tr>
<td style="text-align: center;">8x8x1</td>
<td style="text-align: center;">raw</td>
<td style="text-align: center;">8</td>
<td style="text-align: center;">SlowOnly ResNet50</td>
<td style="text-align: center;">Kinetics-400</td>
```

(continues on next page)

(continued from previous page)

```
<td style="text-align: center;">26.10</td>
<td style="text-align: center;"><a href="https://github.com/open-mmlab/mmaction2/tree/
↪master/configs/detection/ava_kinetics/slowonly_k400-pre-r50_8xb8-8x8x1-10e_ava-
↪kinetics-rgb.py">config</a></td>
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/ava_kinetics/slowonly_k400-pre-r50_8xb8-8x8x1-10e_ava-kinetics-rgb/slowonly_
↪k400-pre-r50_8xb8-8x8x1-10e_ava-kinetics-rgb_20221205-8f8dff3b.pth">ckpt</a></td>
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/ava_kinetics/slowonly_k400-pre-r50_8xb8-8x8x1-10e_ava-kinetics-rgb/slowonly_
↪k400-pre-r50_8xb8-8x8x1-10e_ava-kinetics-rgb.log">log</a></td>
</tr>
<tr>
<td style="text-align: center;">8x8x1</td>
<td style="text-align: center;">raw</td>
<td style="text-align: center;">8</td>
<td style="text-align: center;">SlowOnly ResNet50</td>
<td style="text-align: center;">Kinetics-700</td>
<td style="text-align: center;">27.82</td>
<td style="text-align: center;"><a href="https://github.com/open-mmlab/mmaction2/tree/
↪master/configs/detection/ava_kinetics/slowonly_k700-pre-r50_8xb8-8x8x1-10e_ava-
↪kinetics-rgb.py">config</a></td>
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/ava_kinetics/slowonly_k700-pre-r50_8xb8-8x8x1-10e_ava-kinetics-rgb/slowonly_
↪k700-pre-r50_8xb8-8x8x1-10e_ava-kinetics-rgb_20221205-16a01c37.pth">ckpt</a></td>
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/ava_kinetics/slowonly_k700-pre-r50_8xb8-8x8x1-10e_ava-kinetics-rgb/slowonly_
↪k700-pre-r50_8xb8-8x8x1-10e_ava-kinetics-rgb.log">log</a></td>
</tr>
</tbody>
</table>

#### Training with tricks

We conduct ablation studies to show the improvements of training tricks using␣
↪SlowOnly8x8 pretrained on the Kinetics700 dataset. The baseline is the last raw in␣
↪[AVA2.2](https://github.com/hukkai/mmaction2/tree/ava-kinetics-exp/configs/detection/
↪ava_kinetics##ava22).

<table border="1" class="docutils">
<thead>
<tr>
<th style="text-align: center;">method</th>
<th style="text-align: center;">frame sampling strategy</th>
<th style="text-align: center;">resolution</th>
<th style="text-align: center;">gpus</th>
<th style="text-align: center;">backbone</th>
<th style="text-align: center;">pretrain</th>
<th style="text-align: center;">mAP</th>
<th style="text-align: center;">config</th>
<th style="text-align: center;">ckpt</th>
<th style="text-align: center;">log</th>
</tr>
```

(continues on next page)

(continued from previous page)

```
</thead>
<tbody>
<tr>
<td style="text-align: center;">baseline</td>
<td style="text-align: center;">8x8x1</td>
<td style="text-align: center;">raw</td>
<td style="text-align: center;">8</td>
<td style="text-align: center;">SlowOnly ResNet50</td>
<td style="text-align: center;">Kinetics-700</td>
<td style="text-align: center;">27.82</td>
<td style="text-align: center;"><a href="https://github.com/open-mmlab/mmaction2/tree/
↪master/configs/detection/ava_kinetics/slowonly_k700-pre-r50_8xb8-8x8x1-10e_ava-
↪kinetics-rgb.py">config</a></td>
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/ava_kinetics/slowonly_k700-pre-r50_8xb8-8x8x1-10e_ava-kinetics-rgb/slowonly_
↪k700-pre-r50_8xb8-8x8x1-10e_ava-kinetics-rgb_20221205-16a01c37.pth">ckpt</a></td>
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/ava_kinetics/slowonly_k700-pre-r50_8xb8-8x8x1-10e_ava-kinetics-rgb/slowonly_
↪k700-pre-r50_8xb8-8x8x1-10e_ava-kinetics-rgb.log">log</a></td>
</tr>
<tr>
<td style="text-align: center;">+ context</td>
<td style="text-align: center;">8x8x1</td>
<td style="text-align: center;">raw</td>
<td style="text-align: center;">8</td>
<td style="text-align: center;">SlowOnly ResNet50</td>
<td style="text-align: center;">Kinetics-700</td>
<td style="text-align: center;">28.31</td>
<td style="text-align: center;"><a href="https://github.com/open-mmlab/mmaction2/tree/
↪master/configs/detection/ava_kinetics/slowonly_k700-pre-r50-context_8xb8-8x8x1-10e_ava-
↪kinetics-rgb.py">config</a></td>
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/ava_kinetics/slowonly_k700-pre-r50-context_8xb8-8x8x1-10e_ava-kinetics-rgb/
↪slowonly_k700-pre-r50-context_8xb8-8x8x1-10e_ava-kinetics-rgb_20221205-5d514f8c.pth">
↪ckpt</a></td>
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/ava_kinetics/slowonly_k700-pre-r50-context_8xb8-8x8x1-10e_ava-kinetics-rgb/
↪slowonly_k700-pre-r50-context_8xb8-8x8x1-10e_ava-kinetics-rgb.log">log</a></td>
</tr>
<tr>
<td style="text-align: center;">+ temporal max pooling</td>
<td style="text-align: center;">8x8x1</td>
<td style="text-align: center;">raw</td>
<td style="text-align: center;">8</td>
<td style="text-align: center;">SlowOnly ResNet50</td>
<td style="text-align: center;">Kinetics-700</td>
<td style="text-align: center;">28.48</td>
<td style="text-align: center;"><a href="https://github.com/open-mmlab/mmaction2/tree/
↪master/configs/detection/ava_kinetics/slowonly_k700-pre-r50-context-temporal-max_8xb8-
↪8x8x1-10e_ava-kinetics-rgb.py">config</a></td>
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/ava_kinetics/slowonly_k700-pre-r50-context-temporal-max_8xb8-8x8x1-10e_ava-
↪kinetics-rgb/slowonly_k700-pre-r50-context-temporal-max_8xb8-8x8x1-10e_ava-kinetics-
↪rgb_20221205-5b5e71eb.pth">ckpt</a></td>
```

(continues on next page)

(continued from previous page)

```
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/ava_kinetics/slowonly_k700-pre-r50-context-temporal-max_8xb8-8x8x1-10e_ava-
↪kinetics-rgb/slowonly_k700-pre-r50-context-temporal-max_8xb8-8x8x1-10e_ava-kinetics-
↪rgb.log">log</a></td>
</tr>
<tr>
<td style="text-align: center;">+ nonlinear head</td>
<td style="text-align: center;">8x8x1</td>
<td style="text-align: center;">raw</td>
<td style="text-align: center;">8</td>
<td style="text-align: center;">SlowOnly ResNet50</td>
<td style="text-align: center;">Kinetics-700</td>
<td style="text-align: center;">29.83</td>
<td style="text-align: center;"><a href="https://github.com/open-mmlab/mmaction2/tree/
↪master/configs/detection/ava_kinetics/slowonly_k700-pre-r50-context-temporal-max-nl-
↪head_8xb8-8x8x1-10e_ava-kinetics-rgb.py">config</a></td>
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/ava_kinetics/slowonly_k700-pre-r50-context-temporal-max-nl-head_8xb8-8x8x1-
↪10e_ava-kinetics-rgb/slowonly_k700-pre-r50-context-temporal-max-nl-head_8xb8-8x8x1-10e_
↪ava-kinetics-rgb_20221205-87624265.pth">ckpt</a></td>
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/ava_kinetics/slowonly_k700-pre-r50-context-temporal-max-nl-head_8xb8-8x8x1-
↪10e_ava-kinetics-rgb/slowonly_k700-pre-r50-context-temporal-max-nl-head_8xb8-8x8x1-10e_
↪ava-kinetics-rgb.log">log</a></td>
</tr>
<tr>
<td style="text-align: center;">+ focal loss</td>
<td style="text-align: center;">8x8x1</td>
<td style="text-align: center;">raw</td>
<td style="text-align: center;">8</td>
<td style="text-align: center;">SlowOnly ResNet50</td>
<td style="text-align: center;">Kinetics-700</td>
<td style="text-align: center;">30.33</td>
<td style="text-align: center;"><a href="https://github.com/open-mmlab/mmaction2/tree/
↪master/configs/detection/ava_kinetics/slowonly_k700-pre-r50-context-temporal-max-nl-
↪head_8xb8-8x8x1-focal-10e_ava-kinetics-rgb.py">config</a></td>
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/ava_kinetics/slowonly_k700-pre-r50-context-temporal-max-nl-head_8xb8-8x8x1-
↪focal-10e_ava-kinetics-rgb/slowonly_k700-pre-r50-context-temporal-max-nl-head_8xb8-
↪8x8x1-focal-10e_ava-kinetics-rgb_20221205-37aa8395.pth">ckpt</a></td>
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/ava_kinetics/slowonly_k700-pre-r50-context-temporal-max-nl-head_8xb8-8x8x1-
↪focal-10e_ava-kinetics-rgb/slowonly_k700-pre-r50-context-temporal-max-nl-head_8xb8-
↪8x8x1-focal-10e_ava-kinetics-rgb.log">log</a></td>
</tr>
<tr>
<td style="text-align: center;">+ more frames</td>
<td style="text-align: center;">16x4x1</td>
<td style="text-align: center;">raw</td>
<td style="text-align: center;">8</td>
<td style="text-align: center;">SlowOnly ResNet50</td>
<td style="text-align: center;">Kinetics-700</td>
```

(continues on next page)

(continued from previous page)

~~~
<td style="text-align: center;">31.29</td>
<td style="text-align: center;"><a href="https://github.com/open-mmlab/mmaction2/tree/
↪master/configs/detection/ava_kinetics/slowonly_k700-pre-r50_8xb8-16x4x1-10e-tricks_ava-
↪kinetics-rgb.py">config</a></td>
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/ava_kinetics/slowonly_k700-pre-r50_8xb8-16x4x1-10e-tricks_ava-kinetics-rgb/
↪slowonly_k700-pre-r50_8xb8-16x4x1-10e-tricks_ava-kinetics-rgb_20221205-dd652f81.pth">
↪ckpt</a></td>
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/ava_kinetics/slowonly_k700-pre-r50_8xb8-16x4x1-10e-tricks_ava-kinetics-rgb/
↪slowonly_k700-pre-r50_8xb8-16x4x1-10e-tricks_ava-kinetics-rgb.log">log</a></td>
</tr>
</tbody>
</table>

Note:

The **gpus** indicates the number of gpu we used to get the checkpoint; **+ context**␣
↪indicates that using both RoI feature and global pooled feature for classification;␣
↪**+ temporal max pooling** indicates that using max pooling in the temporal dimension␣
↪for the feature; **nonlinear head** indicates that using a 2-layer mlp instead of a␣
↪linear classifier.

For more details on data preparation, you can refer to [AVA-Kinetics Data␣
↪Preparation](https://github.com/open-mmlab/mmaction2/tree/master/tools/data/ava_
↪kinetics/README.md).

### Train

You can use the following command to train a model.

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

Example: train the SlowOnly model on AVA-Kinetics in a deterministic option.

```shell
python tools/train.py configs/detection/ava_kinetics/slowonly_k400-pre-r50_8xb8-4x16x1-
↪10e_ava-kinetics-rgb.py \
    --cfg-options randomness.seed=0 randomness.deterministic=True
```

For more details, you can refer to the **Training** part in the [Training and Test␣
↪Tutorial](en/user_guides/4_train_test.md).

### Test

You can use the following command to test a model.

```shell
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [optional arguments]
```

(continues on next page)

(continued from previous page)

~~~
```

Example: test the SlowOnly model on AVA-Kinetics and dump the result to a pkl file.

```shell
python tools/test.py configs/detection/ava_kinetics/slowonly_k400-pre-r50_8xb8-4x16x1-
↪10e_ava-kinetics-rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the [Training and Test␣
↪Tutorial](en/user_guides/4_train_test.md).

### Citation

<!-- [DATASET] -->

```BibTeX
@article{li2020ava,
  title={The ava-kinetics localized human actions video dataset},
  author={Li, Ang and Thotakuri, Meghana and Ross, David A and Carreira, Jo{\~a}o and␣
↪Vostrikov, Alexander and Zisserman, Andrew},
  journal={arXiv preprint arXiv:2005.00214},
  year={2020}
}
```
## LFB

[Long-term feature banks for detailed video understanding](https://openaccess.thecvf.com/
↪content_CVPR_2019/html/Wu_Long-Term_Feature_Banks_for_Detailed_Video_Understanding_
↪CVPR_2019_paper.html)

<!-- [ALGORITHM] -->

### Abstract

<!-- [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.

<!-- [IMAGE] -->

<div align=center>
<img src="https://user-images.githubusercontent.com/34324155/143016220-21d90fb3-fd9f-
↪499c-820f-f6c421bda7aa.png" width="800"/>
</div>
~~~

(continues on next page)

(continued from previous page)

```
### Results and Models

#### AVA2.1

<table border="1" class="docutils">
<thead>
<tr>
<th style="text-align: center;">frame sampling strategy</th>
<th style="text-align: center;">resolution</th>
<th style="text-align: center;">gpus</th>
<th style="text-align: center;">backbone</th>
<th style="text-align: center;">pretrain</th>
<th style="text-align: center;">mAP</th>
<th style="text-align: center;">gpu_mem(M)</th>
<th style="text-align: center;">config</th>
<th style="text-align: center;">ckpt</th>
<th style="text-align: center;">log</th>
</tr>
</thead>
<tbody>
<tr>
<td style="text-align: center;">4x16x1</td>
<td style="text-align: center;">raw</td>
<td style="text-align: center;">8</td>
<td style="text-align: center;">SlowOnly ResNet50 (with Nonlocal LFB)</td>
<td style="text-align: center;">Kinetics-400</td>
<td style="text-align: center;">24.05</td>
<td style="text-align: center;">8620</td>
<td style="text-align: center;"><a href="https://github.com/open-mmlab/mmaction2/tree/
↪master/configs/detection/lfb/slowonly-lfb-nl_kinetics400-pretrained-r50_8xb12-4x16x1-
↪20e_ava21-rgb.py">config</a></td>
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/lfb/slowonly-lfb-nl_kinetics400-pretrained-r50_8xb12-4x16x1-20e_ava21-rgb/
↪slowonly-lfb-nl_kinetics400-pretrained-r50_8xb12-4x16x1-20e_ava21-rgb_20220906-
↪4c5b9f25.pth">ckpt</a></td>
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/lfb/slowonly-lfb-nl_kinetics400-pretrained-r50_8xb12-4x16x1-20e_ava21-rgb/
↪slowonly-lfb-nl_kinetics400-pretrained-r50_8xb12-4x16x1-20e_ava21-rgb.log">log</a></td>
</tr>
<tr>
<td style="text-align: center;">4x16x1</td>
<td style="text-align: center;">raw</td>
<td style="text-align: center;">8</td>
<td style="text-align: center;">SlowOnly ResNet50 (with Max LFB)</td>
<td style="text-align: center;">Kinetics-400</td>
<td style="text-align: center;">22.15</td>
<td style="text-align: center;">8425</td>
<td style="text-align: center;"><a href="https://github.com/open-mmlab/mmaction2/tree/
↪master/configs/detection/lfb/slowonly-lfb-max_kinetics400-pretrained-r50_8xb12-4x16x1-
↪20e_ava21-rgb.py">config</a></td>
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/lfb/slowonly-lfb-max_kinetics400-pretrained-r50_8xb12-4x16x1-20e_ava21-rgb/
↪slowonly-lfb-max_kinetics400-pretrained-r50_8xb12-4x16x1-20e_ava21-rgb_20220906-
↪4963135b.pth">ckpt</a></td>
```

(continues on next page)

(continued from previous page)

```
<td style="text-align: center;"><a href="https://download.openmmlab.com/mmaction/v1.0/
↪detection/lfb/slowonly-lfb-max_kinetics400-pretrained-r50_8xb12-4x16x1-20e_ava21-rgb/
↪slowonly-lfb-max_kinetics400-pretrained-r50_8xb12-4x16x1-20e_ava21-rgb.log">log</a></
↪td>
</tr>
</tbody>
</table>

Note:

1. The **gpus** indicates the number of gpu we used to get the checkpoint.
   According to the [Linear Scaling Rule](https://arxiv.org/abs/1706.02677), 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 -> 24.05 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 [slowonly-lfb_ava-
↪pretrained-r50_infer-4x16x1_ava21-rgb.py](https://github.com/open-mmlab/mmaction2/tree/
↪master/configs/detection/lfb/slowonly-lfb_ava-pretrained-r50_infer-4x16x1_ava21-rgb.
↪py). For more details on infer feature bank, you can refer to [Train](##Train) part.
5. You can also dowonload long-term feature bank from [AVA_train_val_float32_lfb](https:/
↪/download.openmmlab.com/mmaction/detection/lfb/AVA_train_val_float32_lfb.rar) or [AVA_
↪train_val_float16_lfb](https://download.openmmlab.com/mmaction/detection/lfb/AVA_train_
↪val_float16_lfb.rar), 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.

### 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 [slowonly-lfb_ava-pretrained-r50_infer-4x16x1_ava21-rgb.py](https://github.com/
↪open-mmlab/mmaction2/tree/master/configs/detection/lfb/slowonly-lfb_ava-pretrained-r50_
↪infer-4x16x1_ava21-rgb.py) (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](https://
↪github.com/open-mmlab/mmaction2/tree/master/mmaction/models/roi_heads/shared_heads/lfb_
↪infer_head.py) 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.
```

(continues on next page)

(continued from previous page)

~~~
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`.

```shell
## set `dataset_mode = 'train'` in lfb_slowonly_r50_ava_infer.py
python tools/test.py slowonly-lfb_ava-pretrained-r50_infer-4x16x1_ava21-rgb.py \
    checkpoints/YOUR_BASELINE_CHECKPOINT.pth --eval mAP

## set `dataset_mode = 'val'` in lfb_slowonly_r50_ava_infer.py
python tools/test.py slowonly-lfb_ava-pretrained-r50_infer-4x16x1_ava21-rgb.py \
    checkpoints/YOUR_BASELINE_CHECKPOINT.pth --eval mAP
~~~

We use [slowonly_r50_4x16x1 checkpoint](https://download.openmmlab.com/mmaction/
↪detection/ava/slowonly_kinetics_pretrained_r50_4x16x1_20e_ava_rgb/slowonly_kinetics_
↪pretrained_r50_4x16x1_20e_ava_rgb_20201217-40061d5f.pth) from [slowonly_kinetics400-
↪pretrained-r50_8xb16-4x16x1-20e_ava21-rgb](https://github.com/open-mmlab/mmaction2/
↪tree/master/configs/detection/ava/slowonly_kinetics400-pretrained-r50_8xb16-4x16x1-20e_
↪ava21-rgb.py) to infer feature bank.

#### b. Train LFB

You can use the following command to train a model.

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

Example: train LFB model on AVA with half-precision long-term feature bank.

```shell
python tools/train.py configs/detection/lfb/slowonly-lfb-nl_kinetics400-pretrained-r50_
↪8xb12-4x16x1-20e_ava21-rgb.py \
  --validate --seed 0 --deterministic
```

For more details and optional arguments infos, you can refer to the **Training** part in␣
↪the [Training and Test Tutorial](en/user_guides/4_train_test.md).

### 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](#
↪#Train).

#### b. Test LFB
```

(continues on next page)
```

(continued from previous page)

~~~
You can use the following command to test a model.

```shell
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 pkl file.

```shell
python tools/test.py configs/detection/lfb/slowonly-lfb-nl_kinetics400-pretrained-r50_
↪8xb12-4x16x1-20e_ava21-rgb.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the [Training and Test␣
↪Tutorial](en/user_guides/4_train_test.md).

Citation

<!-- [DATASET] -->

```BibTeX
@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}
}
```

```BibTeX
@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

THIRTEEN

SKELETON-BASED ACTION RECOGNITION MODELS

13.1 AGCN

Two-Stream Adaptive Graph Convolutional Networks for Skeleton-Based Action Recognition

13.1.1 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.

13.1.2 Results and Models

NTU60_XSub_2D

NTU60_XSub_3D

1. The **gpus** indicates the number of gpus we used to get the checkpoint. If you want to use a different number of gpus or videos per gpu, the best way is to set `--auto-scale-lr` when calling `tools/train.py`, this parameter will auto-scale the learning rate according to the actual batch size, and the original batch size.
2. For two-stream fusion, we use **joint : bone = 1 : 1**. For four-stream fusion, we use **joint : joint-motion : bone : bone-motion = 2 : 1 : 2 : 1**. For more details about multi-stream fusion, please refer to this tutorial.

13.1.3 Train

You can use the following command to train a model.

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

Example: train STGCN model on NTU60-2D dataset in a deterministic option with periodic validation.

```
python tools/train.py configs/skeleton/2s-agcn/2s-agcn_8xb16-joint-u100-80e_ntu60-xsub-
↪keypoint-2d.py \
    --seed 0 --deterministic
```

For more details, you can refer to the **Training** part in the Training and Test Tutorial.

13.1.4 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 NTU60-2D dataset and dump the result to a pickle file.

```
python tools/test.py configs/skeleton/2s-agcn/2s-agcn_8xb16-joint-u100-80e_ntu60-xsub-
↪keypoint-2d.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

13.1.5 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}
}
```

13.2 PoseC3D

Revisiting Skeleton-based Action Recognition

13.2.1 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.

13.2.2 Results and Models

FineGYM

NTU60_XSub

UCF101

HMDB51

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.

13.2.3 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.

```
python tools/train.py configs/skeleton/posec3d/slowonly_r50_8xb16-u48-240e_gym-keypoint.
↪py \
    --cfg-options randomness.seed=0 randomness.deterministic=True
```

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

For more details, you can refer to the **Training** part in the Training and Test Tutorial.

13.2.4 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.

```
python tools/test.py configs/skeleton/posec3d/slowonly_r50_8xb16-u48-240e_gym-keypoint.
↪py \
    checkpoints/SOME_CHECKPOINT.pth
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

13.2.5 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}
}
```

13.3 STGCN

Spatial temporal graph convolutional networks for skeleton-based action recognition

13.3.1 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.

13.3.2 Results and Models

NTU60_XSub_2D

NTU60_XSub_3D

NTU120_XSub_2D

NTU120_XSub_3D

1. The **gpus** indicates the number of gpus we used to get the checkpoint. If you want to use a different number of gpus or videos per gpu, the best way is to set `--auto-scale-lr` when calling `tools/train.py`, this parameter will auto-scale the learning rate according to the actual batch size, and the original batch size.
2. For two-stream fusion, we use **joint : bone = 1 : 1**. For four-stream fusion, we use **joint : joint-motion : bone : bone-motion = 2 : 1 : 2 : 1**. For more details about multi-stream fusion, please refer to this tutorial.

13.3.3 Train

You can use the following command to train a model.

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

Example: train STGCN model on NTU60-2D dataset in a deterministic option with periodic validation.

```
python tools/train.py configs/skeleton/stgcn/stgcn_8xb16-joint-u100-80e_ntu60-xsub-
↪keypoint-2d.py \
    --seed 0 --deterministic
```

For more details, you can refer to the **Training** part in the Training and Test Tutorial.

13.3.4 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 NTU60-2D dataset and dump the result to a pickle file.

```
python tools/test.py configs/skeleton/stgcn/stgcn_8xb16-joint-u100-80e_ntu60-xsub-
↪keypoint-2d.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

13.3.5 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}
}
```

13.4 STGCN++

PYSKL: Towards Good Practices for Skeleton Action Recognition

13.4.1 Abstract

We present PYSKL: an open-source toolbox for skeleton-based action recognition based on PyTorch. The toolbox supports a wide variety of skeleton action recognition algorithms, including approaches based on GCN and CNN. In contrast to existing open-source skeleton action recognition projects that include only one or two algorithms, PYSKL implements six different algorithms under a unified framework with both the latest and original good practices to ease the comparison of efficacy and efficiency. We also provide an original GCN-based skeleton action recognition model named ST-GCN++, which achieves competitive recognition performance without any complicated attention schemes, serving as a strong baseline. Meanwhile, PYSKL supports the training and testing of nine skeleton-based action recognition benchmarks and achieves state-of-the-art recognition performance on eight of them. To facilitate future research on skeleton action recognition, we also provide a large number of trained models and detailed benchmark results to give some insights. PYSKL is released at this https URL and is actively maintained. We will update this report when we add new features or benchmarks. The current version corresponds to PYSKL v0.2.

13.4.2 Results and Models

NTU60_XSub_2D

NTU60_XSub_3D

1. The **gpus** indicates the number of gpus we used to get the checkpoint. If you want to use a different number of gpus or videos per gpu, the best way is to set `--auto-scale-lr` when calling `tools/train.py`, this parameter will auto-scale the learning rate according to the actual batch size, and the original batch size.
2. For two-stream fusion, we use **joint : bone = 1 : 1**. For four-stream fusion, we use **joint : joint-motion : bone : bone-motion = 2 : 1 : 2 : 1**. For more details about multi-stream fusion, please refer to this tutorial.

13.4.3 Train

You can use the following command to train a model.

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

Example: train STGCN++ model on NTU60-2D dataset in a deterministic option with periodic validation.

```
python tools/train.py configs/skeleton/stgcnpp/stgcnpp_8xb16-joint-u100-80e_ntu60-xsub-
↪keypoint-2d.py \
    --seed 0 --deterministic
```

For more details, you can refer to the **Training** part in the Training and Test Tutorial.

13.4.4 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 NTU60-2D dataset and dump the result to a pickle file.

```
python tools/test.py configs/skeleton/stgcnpp/stgcnpp_8xb16-joint-u100-80e_ntu60-xsub-
↪keypoint-2d.py \
    checkpoints/SOME_CHECKPOINT.pth --dump result.pkl
```

For more details, you can refer to the **Test** part in the Training and Test Tutorial.

13.4.5 Citation

```
@misc{duan2022PYSKL,
  url = {https://arxiv.org/abs/2205.09443},
  author = {Duan, Haodong and Wang, Jiaqi and Chen, Kai and Lin, Dahua},
  title = {PYSKL: Towards Good Practices for Skeleton Action Recognition},
  publisher = {arXiv},
  year = {2022}
}
```

CHAPTER

FOURTEEN

ACTION LOCALIZATION MODELS

14.1 BMN

Bmn: Boundary-matching network for temporal action proposal generation

14.1.1 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.

14.1.2 Results and Models

ActivityNet feature

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.
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 Data Preparation.

14.1.3 Train

Train BMN model on ActivityNet features dataset.

```
bash tools/dist_train.sh configs/localization/bmn/bmn_400x100_2x8_9e_activitynet_feature.
↪py 2
```

For more details, you can refer to the **Training** part in the Training and Test Tutorial.

14.1.4 Test

Test BMN on ActivityNet feature dataset.

```
python3 tools/test.py  configs/localization/bmn/bmn_400x100_2x8_9e_activitynet_feature.
↪py CHECKPOINT.PTH
```

For more details, you can refer to the **Testing** part in the Training and Test Tutorial.

14.1.5 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}
}
```

14.2 BSN

Bsn: Boundary sensitive network for temporal action proposal generation

14.2.1 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.

14.2.2 Results and Models

ActivityNet feature

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.

For more details on data preparation, you can refer to ActivityNet Data Preparation.

14.2.3 Training and Test

The traing of the BSN model is three-stages. Firstly train the Temporal evaluation module (TEM):

```
python3 tools/train.py configs/localization/bsn/bsn_tem_1xb16-400x100-20e_activitynet-
↪feature.py
```

After training use the TEM module to generate the probabilities sequence (actionness, starting, and ending) for the training and validation dataset:

```
python tools/test.py configs/localization/bsn/bsn_tem_400x100_1xb16_20e_activitynet_
↪feature.py \
    work_dirs/bsn_400x100_20e_1xb16_activitynet_feature/tem_epoch_20.pth
```

The second step is to run the Proposal generation module (PGM) to generate Boundary-Sensitive Proposal (BSP) feature for the training and validation dataset:

```
python tools/misc/bsn_proposal_generation.py configs/localization/bsn/bsn_pgm_400x100_
↪activitynet-feature.py --mode train
python tools/misc/bsn_proposal_generation.py configs/localization/bsn/bsn_pgm_400x100_
↪activitynet-feature.py --mode test
```

The last step is to train (and validate) the Proposal evaluation module (PEM):

```
python python tools/train.py configs/localization/bsn/bsn_pem_1xb16-400x100-20e_
↪activitynet-feature.py
```

(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
```

14.2.4 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}
}
```

CHAPTER FIFTEEN

CONTRIBUTING TO MMACTION2

All kinds of contributions are welcome, including but not limited to the following.

- Fixes (typo, bugs)
- New features and components
- Add documentation or translate the documentation into other languages

15.1 Workflow

1. Fork and pull the latest mmaction2
2. Checkout a new branch with a meaningful name (do not use master branch for PRs)
3. Commit your changes
4. Create a PR

Note:

- If you plan to add some new features that involve large changes, it is encouraged to open an issue for discussion first.
- If you are the author of some papers and would like to include your method to mmaction2, please contact us. We will much appreciate your contribution.

15.2 Code style

15.2.1 Python

We adopt PEP8 as the preferred code style.

We use the following tools for linting and formatting:

- flake8: linter
- yapf: formatter
- isort: sort imports
- codespell: A Python utility to fix common misspellings in text files.

- mdformat: Mdformat is an opinionated Markdown formatter that can be used to enforce a consistent style in Markdown files.
- docformatter: A formatter to format docstring.

Style configurations of yapf and isort can be found in setup.cfg.

We use pre-commit hook that checks and formats for `flake8`, `yapf`, `isort`, `trailing whitespaces`, `markdown files`, fixes `end-of-files`, sorts `requirments.txt` automatically on every commit. The config for a pre-commit hook is stored in .pre-commit-config.

After you clone the repository, you will need to install initialize pre-commit hook.

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

From the repository folder

```
pre-commit install
```

After this on every commit check code linters and formatter will be enforced.

> Before you create a PR, make sure that your code lints and is formatted by yapf.

15.2.2 C++ and CUDA

We follow the Google C++ Style Guide.

CHAPTER

SIXTEEN

PROJECTS BASED ON MMACTION2

There are many research works and projects built on MMAction2. We list some of them as examples of how to extend MMAction2 for your own projects. As the page might not be completed, please feel free to create a PR to update this page.

16.1 Projects as an extension

- OTEAction2: OpenVINO Training Extensions for Action Recognition.
- PYSKL: A Toolbox Focusing on Skeleton-Based Action Recognition.

16.2 Projects of papers

There are also projects released with papers. Some of the papers are published in top-tier conferences (CVPR, ICCV, and ECCV), the others are also highly influential. To make this list also a reference for the community to develop and compare new video understanding algorithms, we list them following the time order of top-tier conferences. Methods already supported and maintained by MMAction2 are not listed.

- Video Swin Transformer, CVPR 2022. [paper][github]
- Evidential Deep Learning for Open Set Action Recognition, ICCV 2021 Oral. [paper][github]
- Rethinking Self-supervised Correspondence Learning: A Video Frame-level Similarity Perspective, ICCV 2021 Oral. [paper][github]
- MGSampler: An Explainable Sampling Strategy for Video Action Recognition, ICCV 2021. [paper][github]
- MultiSports: A Multi-Person Video Dataset of Spatio-Temporally Localized Sports Actions, ICCV 2021. [paper]
- Long Short-Term Transformer for Online Action Detection, NeurIPS 2021 [paper][github]

CHAPTER

SEVENTEEN

CHANGELOG

17.1 1.0.0rc3 (2/10/2023)

Highlights

- Support Action Recognition model UniFormer V1(ICLR'2022), UniFormer V2(Arxiv'2022).
- Support training MViT V2(CVPR'2022), and MaskFeat(CVPR'2022) fine-tuning.

New Features

- Support UniFormer V1/V2 (#2153)
- Support training MViT, and MaskFeat fine-tuning (#2186)
- Support a unified inference interface: Inferencer (#2164)

Improvements

- Support load data list from multi-backends (#2176)

Bug Fixes

- Upgrade isort to fix CI (#2198)
- Fix bug in skeleton demo (#2214)

Documentation

- Add Chinese documentation for config.md (#2188)
- Add readme for Omnisource (#2205)

17.2 1.0.0rc2 (1/10/2023)

Highlights

- Support Action Recognition model VideoMAE(NeurIPS'2022), MViT V2(CVPR'2022), C2D and skeleton-based action recognition model STGCN++
- Support Omni-Source training on ImageNet and Kinetics datasets
- Support exporting spatial-temporal detection models to ONNX

New Features

- Support VideoMAE (#1942)
- Support MViT V2 (#2007)

- Support C2D (#2022)
- Support AVA-Kinetics dataset (#2080)
- Support STGCN++ (#2156)
- Support exporting spatial-temporal detection models to ONNX (#2148)
- Support Omni-Source training on ImageNet and Kinetics datasets (#2143)

Improvements

- Support repeat batch data augmentation (#2170)
- Support calculating FLOPs tool powered by fvcore (#1997)
- Support Spatial-temporal detection demo (#2019)
- Add SyncBufferHook and add randomness config in train.py (#2044)
- Refactor gradcam (#2049)
- Support init_cfg in Swin and ViTMAE (#2055)
- Refactor STGCN and related pipelines (#2087)
- Refactor visualization tools (#2092)
- Update `SampleFrames` transform and improve most models' performance (#1942)
- Support real-time webcam demo (#2152)
- Refactor and enhance 2s-AGCN (#2130)
- Support adjusting fps in `SampleFrame` (#2157)

Bug Fixes

- Fix CI upstream library dependency (#2000)
- Fix SlowOnly readme typos and results (#2006)
- Fix VideoSwin readme (#2010)
- Fix tools and mim error (#2028)
- Fix Imgaug wrapper (#2024)
- Remove useless scripts (#2032)
- Fix multi-view inference (#2045)
- Update mmcv maximum version to 1.8.0 (#2047)
- Fix torchserver dependency (#2053)
- Fix `gen_ntu_rgbd_raw` script (#2076)
- Update AVA-Kinetics experiment configs and results (#2099)
- Add `joint.pkl` and `bone.pkl` used in multi-stream fusion tool (#2106)
- Fix lint CI config (#2110)
- Update testing accuracy for modified `SampleFrames` (#2117), (#2121), (#2122), (#2124), (#2125), (#2126), (#2129), (#2128)
- Fix timm related bug (#1976)
- Fix `check_videos.py` script (#2134)

- Update CI maximum torch version to 1.13.0 (#2118)

Documentation

- Add MMYOLO description in README (#2011)
- Add v1.x introduction in README (#2023)
- Fix link in README (#2035)
- Refine some docs (#2038), (#2040), (#2058)
- Update TSN/TSM Readme (#2082)
- Add chinese document (#2083)
- Adjust document structure (#2088)
- Fix Sth-Sth and Jester dataset links (#2103)
- Fix doc link (#2131)

17.3 1.0.0rc1 (10/14/2022)

Highlights

- Support Video Swin Transformer

New Features

- Support Video Swin Transformer (#1939)

Improvements

- Add colab tutorial for 1.x (#1956)
- Support skeleton-based action recognition demo (#1920)

Bug Fixes

- Fix link in doc (#1986, #1967, #1951, #1926,#1944, #1944, #1927, #1925)
- Fix CI (#1987, #1930, #1923)
- Fix pre-commit hook config (#1971)
- Fix TIN config (#1912)
- Fix UT for BMN and BSN (#1966)
- Fix UT for Recognizer2D (#1937)
- Fix BSN and BMN configs for localization (#1913)
- Modeify ST-GCN configs (#1913)
- Fix typo in migration doc (#1931)
- Remove Onnx related tools (#1928)
- Update TANet readme (#1916, #1890)
- Update 2S-AGCN readme (#1915)
- Fix TSN configs (#1905)
- Fix configs for detection (#1903)

- Fix typo in TIN config (#1904)
- Fix PoseC3D readme (#1899)
- Fix ST-GCN configs (#1891)
- Fix audio recognition readme (#1898)
- Fix TSM readme (#1887)
- Fix SlowOnly readme (#1889)
- Fix TRN readme (#1888)
- Fix typo in get_started doc (#1895)

17.4 1.0.0rc0 (09/01/2022)

We are excited to announce the release of MMAction2 v1.0.0rc0. MMAction2 1.0.0beta is the first version of MMAction2 1.x, a part of the OpenMMLab 2.0 projects. Built upon the new training engine.

Highlights

- **New engines**. MMAction2 1.x is based on MMEngine](https://github.com/open-mmlab/mmengine), which provides a general and powerful runner that allows more flexible customizations and significantly simplifies the entrypoints of high-level interfaces.
- **Unified interfaces**. As a part of the OpenMMLab 2.0 projects, MMAction2 1.x unifies and refactors the interfaces and internal logics of train, testing, datasets, models, evaluation, and visualization. All the OpenMMLab 2.0 projects share the same design in those interfaces and logics to allow the emergence of multi-task/modality algorithms.
- **More documentation and tutorials**. We add a bunch of documentation and tutorials to help users get started more smoothly. Read it here.

Breaking Changes

In this release, we made lots of major refactoring and modifications. Please refer to the *migration guide* for details and migration instructions.

17.5 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)

17.6 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

17.7 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)

17.8 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)

17.9 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)

17.10 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)

17.11 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)

17.12 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)

17.13 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)

17.14 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)

17.15 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)

17.16 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)

17.17 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)

17.18 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

17.19 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)

17.20 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)

17.21 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)

17.22 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)

17.23 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)

17.24 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

EIGHTEEN

FAQ

18.1 Outline

We list some common issues faced by many users and their corresponding solutions here.

- FAQ
  - Outline
  - Installation
  - Data
  - Training
  - Testing

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 to fill in all required information in the template.

18.2 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 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 and mmengine before you install them. You could follow this tutorial to install them.

18.3 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) (2)**

  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_frames` beforehand.

18.4 Training

- **How to just use trained recognizer models for backbone pre-training?**

  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 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 almost 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 MMEngine 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.

18.5 Testing

- **How to make predicted score normalized by softmax within [0, 1]?**

  change this in the config, make `model.cls_head.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.

CHAPTER NINETEEN

ENGLISH

CHAPTER TWENTY

CHAPTER

TWENTYONE

INDICES AND TABLES

- genindex
- search