MMClassification Documentation

Release 0.25.0

MMClassification Authors

Jul 19, 2023

GET STARTED

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

您可以在页面左下角切换中英文文档。

CHAPTER

ONE

PREREQUISITES

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

MMClassification works on Linux, Windows and macOS. It requires Python 3.6+, CUDA 9.2+ and PyTorch 1.5+.

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 matches 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 MMClassification. However, the whole process is highly customizable. See *Customize Installation* section for more information.

2.1 Best Practices

Step 0. Install MMCV using MIM.

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

Step 1. Install MMClassification.

According to your needs, we support two install modes:

- *Install from source (Recommended)*: You want to develop your own image classification task or new features based on MMClassification framework. For example, you want to add new dataset or new models. And you can use all tools we provided.
- *Install as a Python package*: You just want to call MMClassification's APIs or import MMClassification's modules in your project.

2.1.1 Install from source

In this case, install mmcls from source:

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

Optionally, if you want to contribute to MMClassification or experience experimental functions, please checkout to the dev branch:

```
git checkout dev
```

2.1.2 Install as a Python package

Just install with pip.

```
pip install mmcls
```

2.2 Verify the installation

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

Step 1. We need to download config and checkpoint files.

```
mim download mmcls --config resnet50_8xb32_in1k --dest .
```

Step 2. Verify the inference demo.

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

```
python demo/image_demo.py demo/demo.JPEG resnet50_8xb32_in1k.py resnet50_8xb32_in1k_
↪20210831-ea4938fc.pth --device cpu
```

You will see the output result dict including `pred_label`, `pred_score` and `pred_class` in your terminal. And if you have graphical interface (instead of remote terminal etc.), you can enable `--show` option to show the demo image with these predictions in a window.

Option (b). If you install mmcls as a python package, open you python interpreter and copy&paste the following codes.

```
from mmcls.apis import init_model, inference_model

config_file = 'resnet50_8xb32_in1k.py'
checkpoint_file = 'resnet50_8xb32_in1k_20210831-ea4938fc.pth'
model = init_model(config_file, checkpoint_file, device='cpu')  # or device='cuda:0'
inference_model(model, 'demo/demo.JPEG')
```

You will see a dict printed, including the predicted label, score and category name.

2.3 Customize Installation

2.3.1 CUDA versions

When installing PyTorch, you 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, thus depending 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-full built for PyTorch 1.10.x and CUDA 11.3.

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

2.3.3 Install on CPU-only platforms

MMClassification can be built for CPU only environment. In CPU mode you can train (requires MMCV version >= 1.4.4), test or inference a model.

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

2.3.4 Install on Google Colab

Google Colab usually has PyTorch installed, thus we only need to install MMCV and MMClassification with the following commands.

Step 1. Install MMCV using MIM.

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

Step 2. Install MMClassification from the source.

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

Step 3. Verification.

```
import mmcls
print(mmcls.__version__)
# Example output: 0.23.0 or newer
```

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

2.3.5 Using MMClassification with Docker

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

```
# build an image with PyTorch 1.8.1, CUDA 10.2
# If you prefer other versions, just modified the Dockerfile
docker build -t mmclassification docker/
```

Run it with

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

2.4 Trouble shooting

If you have some issues during the installation, please first view the *FAQ* page. You may open an issue on GitHub if no solution is found.

CHAPTER

THREE

GETTING STARTED

This page provides basic tutorials about the usage of MMClassification.

3.1 Prepare datasets

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

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

For ImageNet, it has multiple versions, but the most commonly used one is ILSVRC 2012. It can be accessed with the following steps.

1. Register an account and login to the download page.
2. Find download links for ILSVRC2012 and download the following two files
   - ILSVRC2012_img_train.tar (~138GB)
   - ILSVRC2012_img_val.tar (~6.3GB)
3. Untar the downloaded files
4. Download meta data using this script

For MNIST, CIFAR10 and CIFAR100, the datasets will be downloaded and unzipped automatically if they are not found.

For using custom datasets, please refer to *Tutorial 3: Customize Dataset*.

3.2 Inference with pretrained models

We provide scripts to inference a single image, inference a dataset and test a dataset (e.g., ImageNet).

3.2.1 Inference a single image

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

# Example
python demo/image_demo.py demo/demo.JPEG configs/resnet/resnet50_8xb32_in1k.py \
  https://download.openmmlab.com/mmclassification/v0/resnet/resnet50_8xb32_in1k_20210831-
↪ea4938fc.pth
```

3.2.2 Inference and test a dataset

- single GPU
- CPU
- single node multiple GPU
- multiple node

You can use the following commands to infer a dataset.

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

# CPU: disable GPUs and run single-gpu testing script
export CUDA_VISIBLE_DEVICES=-1
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--metrics ${METRICS}] [--out $
↪{RESULT_FILE}]

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

# multi-node in slurm environment
python tools/test.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--metrics ${METRICS}] [--out $
↪{RESULT_FILE}] --launcher slurm
```

Optional arguments:

- `RESULT_FILE`: Filename of the output results. If not specified, the results will not be saved to a file. Support formats include json, yaml and pickle.
- `METRICS`： Items to be evaluated on the results, like accuracy, precision, recall, etc.

Examples:

Infer ResNet-50 on CIFAR10 validation set to get predicted labels and their corresponding predicted scores.

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

3.3 Train a model

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

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

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

```
evaluation = dict(interval=12)  # Evaluate the model per 12 epochs.
```

3.3.1 Train with a single GPU

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

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

3.3.2 Train with CPU

The process of training on the CPU is consistent with single GPU training. We just need to disable GPUs before the training process.

```
export CUDA_VISIBLE_DEVICES=-1
```

And then run the script *above*.

Warning: The process of training on the CPU is consistent with single GPU training. We just need to disable GPUs before the training process.

3.3.3 Train with multiple GPUs in single machine

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

Optional arguments are:

- `--no-validate` (**not suggested**): By default, the codebase will perform evaluation at every k (default value is 1) epochs during the training. To disable this behavior, use `--no-validate`.
- `--work-dir ${WORK_DIR}`: Override the working directory specified in the config file.
- `--resume-from ${CHECKPOINT_FILE}`: Resume from a previous checkpoint file.

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

3.3.4 Train with multiple machines

If you launch with multiple machines simply connected with ethernet, you can simply run following commands:

On the first machine:

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

On the second machine:

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

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

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

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

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

If you have just multiple machines connected with ethernet, you can refer to PyTorch launch utility. Usually it is slow if you do not have high speed networking like InfiniBand.

3.3.5 Launch multiple jobs on a single machine

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

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

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

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

In `config1.py`,

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

In `config2.py`,

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

Then you can launch two jobs with `config1.py` ang `config2.py`.

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

3.3.6 Train with IPU

The process of training on the IPU is consistent with single GPU training. We just need to have IPU machine and environment and add an extra argument `--ipu-replicas ${IPU_NUM}`

3.4 Useful tools

We provide lots of useful tools under `tools/` directory.

3.4.1 Get the FLOPs and params (experimental)

We provide a script adapted from flops-counter.pytorch to compute the FLOPs and params of a given model.

```
python tools/analysis_tools/get_flops.py ${CONFIG_FILE} [--shape ${INPUT_SHAPE}]
```

You will get the result like this.

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

Warning: This tool is still experimental and we do not guarantee that the number is correct. You may well use the result for simple comparisons, but double check it before you adopt it in technical reports or papers.

- FLOPs are related to the input shape while parameters are not. The default input shape is (1, 3, 224, 224).
- Some operators are not counted into FLOPs like GN and custom operators. Refer to `mmcv.cnn.get_model_complexity_info()` for details.

3.4.2 Publish a model

Before you publish a model, 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/convert_models/publish_model.py ${INPUT_FILENAME} ${OUTPUT_FILENAME}
```

E.g.,

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

The final output filename will be `imagenet_resnet50_{date}-{hash id}.pth`.

3.5 Tutorials

Currently, we provide five tutorials for users.

- *learn about config*
- *finetune models*
- *add new dataset*
- *design data pipeline*
- *add new modules*
- *customize schedule*
- *customize runtime settings*.

CHAPTER

FOUR

TUTORIAL 1: LEARN ABOUT CONFIGS

MMClassification mainly uses python files as configs. The design of our configuration file system integrates modularity and inheritance, facilitating users to conduct various experiments. All configuration files are placed in the `configs` folder, which mainly contains the primitive configuration folder of `_base_` and many algorithm folders such as `resnet`, `swin_transformer`, `vision_transformer`, etc.

If you wish to inspect the config file, you may run `python tools/misc/print_config.py /PATH/TO/CONFIG` to see the complete config.

- *Config File and Checkpoint Naming Convention*
- *Config File Structure*
- *Inherit and Modify Config File*
  - *Use intermediate variables in configs*
  - *Ignore some fields in the base configs*
  - *Use some fields in the base configs*
- *Modify config through script arguments*
- *Import user-defined modules*
- *FAQ*

4.1 Config File and Checkpoint Naming Convention

We follow the below convention to name config files. Contributors are advised to follow the same style. The config file names are divided into four parts: algorithm info, module information, training information and data information. Logically, different parts are concatenated by underscores `'_'`, and words in the same part are concatenated by dashes `'-'`.

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

- `algorithm info`: algorithm information, model name and neural network architecture, such as resnet, etc.;
- `module info`: module information is used to represent some special neck, head and pretrain information;
- `training info`: Training information, some training schedule, including batch size, lr schedule, data augment and the like;
- `data info`: Data information, dataset name, input size and so on, such as imagenet, cifar, etc.;

4.1.1 Algorithm information

The main algorithm name and the corresponding branch architecture information. E.g：

- `resnet50`
- `mobilenet-v3-large`
- `vit-small-patch32` : `patch32` represents the size of the partition in `ViT` algorithm;
- `seresnext101-32x4d` : `SeResNet101` network structure, `32x4d` means that `groups` and `width_per_group` are 32 and 4 respectively in `Bottleneck`;

4.1.2 Module information

Some special `neck`, `head` and `pretrain` information. In classification tasks, `pretrain` information is the most commonly used:

- `in21k-pre` : pre-trained on ImageNet21k;
- `in21k-pre-3rd-party` : pre-trained on ImageNet21k and the checkpoint is converted from a third-party repository;

4.1.3 Training information

Training schedule, including training type, `batch size`, `lr schedule`, data augment, special loss functions and so on:

- format `{gpu x batch_per_gpu}`, such as `8xb32`

Training type (mainly seen in the transformer network, such as the `ViT` algorithm, which is usually divided into two training type: pre-training and fine-tuning):

- `ft` : configuration file for fine-tuning
- `pt` : configuration file for pretraining

Training recipe. Usually, only the part that is different from the original paper will be marked. These methods will be arranged in the order `{pipeline aug}-{train aug}-{loss trick}-{scheduler}-{epochs}`.

- `coslr-200e` : use cosine scheduler to train 200 epochs
- `autoaug-mixup-lbs-coslr-50e` : use `autoaug`, `mixup`, `label smooth`, `cosine scheduler` to train 50 epochs

4.1.4 Data information

- `in1k` : `ImageNet1k` dataset, default to use the input image size of 224x224;
- `in21k` : `ImageNet21k` dataset, also called `ImageNet22k` dataset, default to use the input image size of 224x224;
- `in1k-384px` : Indicates that the input image size is 384x384;
- `cifar100`

4.1.5 Config File Name Example

```
repvgg-D2se_deploy_4xb64-autoaug-lbs-mixup-coslr-200e_in1k.py
```

- `repvgg-D2se`: Algorithm information
  - `repvgg`: The main algorithm.
  - `D2se`: The architecture.
- `deploy`: Module information, means the backbone is in the deploy state.
- `4xb64-autoaug-lbs-mixup-coslr-200e`: Training information.
  - `4xb64`: Use 4 GPUs and the size of batches per GPU is 64.
  - `autoaug`: Use `AutoAugment` in training pipeline.
  - `lbs`: Use label smoothing loss.
  - `mixup`: Use `mixup` training augment method.
  - `coslr`: Use cosine learning rate scheduler.
  - `200e`: Train the model for 200 epochs.
- `in1k`: Dataset information. The config is for `ImageNet1k` dataset and the input size is `224x224`.

Note: Some configuration files currently do not follow this naming convention, and related files will be updated in the near future.

4.1.6 Checkpoint Naming Convention

The naming of the weight mainly includes the configuration file name, date and hash value.

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

4.2 Config File Structure

There are four kinds of basic component file in the `configs/_base_` folders, namely:

- models
- datasets
- schedules
- runtime

You can easily build your own training config file by inherit some base config files. And the configs that are composed by components from `_base_` are called *primitive*.

For easy understanding, we use ResNet50 primitive config as a example and comment the meaning of each line. For more detaile, please refer to the API documentation.

```
_base_ = [
    '../_base_/models/resnet50.py',           # model
    '../_base_/datasets/imagenet_bs32.py',    # data
    '../_base_/schedules/imagenet_bs256.py',  # training schedule
    '../_base_/default_runtime.py'            # runtime setting
]
```

The four parts are explained separately below, and the above-mentioned ResNet50 primitive config are also used as an example.

4.2.1 model

The parameter "`model`" is a python dictionary in the configuration file, which mainly includes information such as network structure and loss function:

- `type` : Classifier name, MMCls supports `ImageClassifier`, refer to API documentation.
- `backbone` : Backbone configs, refer to API documentation for available options.
- `neck` : Neck network name, MMCls supports `GlobalAveragePooling`, please refer to API documentation.
- `head`: Head network name, MMCls supports single-label and multi-label classification head networks, available options refer to API documentation.
  - `loss`: Loss function type, supports `CrossEntropyLoss`, `LabelSmoothLoss` etc., For available options, refer to API documentation.
- `train_cfg` : Training augment config, MMCls supports `mixup`, `cutmix` and other augments.

Note: The 'type' in the configuration file is not a constructed parameter, but a class name.

```
model = dict(
    type='ImageClassifier',     # Classifier name
    backbone=dict(
        type='ResNet',          # Backbones name
        depth=50,               # depth of backbone, ResNet has options of 18, 34, 50,␣
↪101, 152.
        num_stages=4,           # number of stages，The feature maps generated by these␣
↪states are used as the input for the subsequent neck and head.
        out_indices=(3, ),      # The output index of the output feature maps.
        frozen_stages=-1,       # the stage to be frozen, '-1' means not be forzen
        style='pytorch'),        # The style of backbone, 'pytorch' means that stride 2␣
↪layers are in 3x3 conv, 'caffe' means stride 2 layers are in 1x1 convs.
    neck=dict(type='GlobalAveragePooling'),    # neck network name
    head=dict(
        type='LinearClsHead',     # linear classification head，
        num_classes=1000,         # The number of output categories, consistent with the␣
↪number of categories in the dataset
        in_channels=2048,         # The number of input channels, consistent with the␣
↪output channel of the neck
        loss=dict(type='CrossEntropyLoss', loss_weight=1.0), # Loss function␣
↪configuration information
        topk=(1, 5),              # Evaluation index, Top-k accuracy rate, here is the␣
↪accuracy rate of top1 and top5
```

(continues on next page)

(continued from previous page)

```
    ))
```

4.2.2 data

The parameter "`data`" is a python dictionary in the configuration file, which mainly includes information to construct dataloader:

- `samples_per_gpu` : the BatchSize of each GPU when building the dataloader
- `workers_per_gpu` : the number of threads per GPU when building dataloader
- `train ｜ val ｜ test` : config to construct dataset
  - `type`: Dataset name, MMCls supports `ImageNet`, `Cifar` etc., refer to API documentation
  - `data_prefix` : Dataset root directory
  - `pipeline` : Data processing pipeline, refer to related tutorial CUSTOM DATA PIPELINES

The parameter `evaluation` is also a dictionary, which is the configuration information of `evaluation hook`, mainly including evaluation interval, evaluation index, etc..

```
# dataset settings
dataset_type = 'ImageNet'  # dataset name，
img_norm_cfg = dict(       # Image normalization config to normalize the input images
    mean=[123.675, 116.28, 103.53],  # Mean values used to pre-training the pre-trained␣
↪backbone models
    std=[58.395, 57.12, 57.375],     # Standard variance used to pre-training the pre-
↪trained backbone models
    to_rgb=True)                     # Whether to invert the color channel, rgb2bgr or␣
↪bgr2rgb.
# train data pipeline
train_pipeline = [
    dict(type='LoadImageFromFile'),                # First pipeline to load images from␣
↪file path
    dict(type='RandomResizedCrop', size=224),      # RandomResizedCrop
    dict(type='RandomFlip', flip_prob=0.5, direction='horizontal'),  # Randomly flip the␣
↪picture horizontally with a probability of 0.5
    dict(type='Normalize', **img_norm_cfg),        # normalization
    dict(type='ImageToTensor', keys=['img']),      # convert image from numpy into torch.
↪Tensor
    dict(type='ToTensor', keys=['gt_label']),      # convert gt_label into torch.Tensor
    dict(type='Collect', keys=['img', 'gt_label']) # Pipeline that decides which keys in␣
↪the data should be passed to the detector
]
# test data pipeline
test_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='Resize', size=(256, -1)),
    dict(type='CenterCrop', crop_size=224),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='ImageToTensor', keys=['img']),
    dict(type='Collect', keys=['img'])             # do not pass gt_label while testing
]
```

(continues on next page)

(continued from previous page)

```
data = dict(
    samples_per_gpu=32,     # Batch size of a single GPU
    workers_per_gpu=2,      # Worker to pre-fetch data for each single GPU
    train=dict(  # Train dataset config
    train=dict(            # train data config
        type=dataset_type,                  # dataset name
        data_prefix='data/imagenet/train',  # Dataset root, when ann_file does not exist,
↪ the category information is automatically obtained from the root folder
        pipeline=train_pipeline),           # train data pipeline
    val=dict(              # val data config
        type=dataset_type,
        data_prefix='data/imagenet/val',
        ann_file='data/imagenet/meta/val.txt',   #  ann_file existes, the category␣
↪information is obtained from file
        pipeline=test_pipeline),
    test=dict(             # test data config
        type=dataset_type,
        data_prefix='data/imagenet/val',
        ann_file='data/imagenet/meta/val.txt',
        pipeline=test_pipeline))
evaluation = dict(       # The config to build the evaluation hook, refer to https://
↪github.com/open-mmlab/mmdetection/blob/master/mmdet/core/evaluation/eval_hooks.py#L7␣
↪for more details.
    interval=1,          # Evaluation interval
    metric='accuracy')   # Metrics used during evaluation
```

4.2.3 training schedule

Mainly include optimizer settings, `optimizer hook` settings, learning rate schedule and `runner` settings:

- `optimizer`: optimizer setting , support all optimizers in `pytorch`, refer to related mmcv documentation.
- `optimizer_config`: `optimizer hook` configuration file, such as setting gradient limit, refer to related mmcv code.
- `lr_config`: Learning rate scheduler, supports "CosineAnnealing", "Step", "Cyclic", etc. refer to related mmcv documentation for more options.
- `runner`: For `runner`, please refer to `mmcv` for `runner` introduction document.

```
# he configuration file used to build the optimizer, support all optimizers in PyTorch.
optimizer = dict(type='SGD',         # Optimizer type
                lr=0.1,              # Learning rate of optimizers, see detail usages of␣
↪the parameters in the documentation of PyTorch
                momentum=0.9,        # Momentum
                weight_decay=0.0001) # Weight decay of SGD
# Config used to build the optimizer hook, refer to https://github.com/open-mmlab/mmcv/
↪blob/master/mmcv/runner/hooks/optimizer.py#L8 for implementation details.
optimizer_config = dict(grad_clip=None)  # Most of the methods do not use gradient clip
# Learning rate scheduler config used to register LrUpdater hook
lr_config = dict(policy='step',          # The policy of scheduler, also support␣
↪CosineAnnealing, Cyclic, etc. Refer to details of supported LrUpdater from https://
↪github.com/open-mmlab/mmcv/blob/master/mmcv/runner/hooks/lr_updater.py#L9.
```

(continues on next page)

(continued from previous page)

```
                 step=[30, 60, 90])      # Steps to decay the learning rate
runner = dict(type='EpochBasedRunner',   # Type of runner to use (i.e. IterBasedRunner␣
↪or EpochBasedRunner)
            max_epochs=100)    # Runner that runs the workflow in total max_epochs. For␣
↪IterBasedRunner use `max_iters`
```

4.2.4 runtime setting

This part mainly includes saving the checkpoint strategy, log configuration, training parameters, breakpoint weight path, working directory, etc..

```
# Config to set the checkpoint hook, Refer to https://github.com/open-mmlab/mmcv/blob/
↪master/mmcv/runner/hooks/checkpoint.py for implementation.
checkpoint_config = dict(interval=1)    # The save interval is 1
# config to register logger hook
log_config = dict(
    interval=100,                       # Interval to print the log
    hooks=[
        dict(type='TextLoggerHook'),           # The Tensorboard logger is also supported
        # dict(type='TensorboardLoggerHook')
    ])

dist_params = dict(backend='nccl')   # Parameters to setup distributed training, the␣
↪port can also be set.
log_level = 'INFO'              # The output level of the log.
resume_from = None              # Resume checkpoints from a given path, the training will␣
↪be resumed from the epoch when the checkpoint's is saved.
workflow = [('train', 1)]       # Workflow for runner. [('train', 1)] means there is only␣
↪one workflow and the workflow named 'train' is executed once.
work_dir = 'work_dir'           # Directory to save the model checkpoints and logs for␣
↪the current experiments.
```

4.3 Inherit and Modify Config File

For easy understanding, we recommend contributors to inherit from existing methods.

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 example, if your config file is based on ResNet with some other modification, you can first inherit the basic ResNet structure, dataset and other training setting by specifying `_base_ ='./resnet50_8xb32_in1k.py'` (The path relative to your config file), and then modify the necessary parameters in the config file. A more specific example, now we want to use almost all configs in `configs/resnet/resnet50_8xb32_in1k.py`, but change the number of training epochs from 100 to 300, modify when to decay the learning rate, and modify the dataset path, you can create a new config file `configs/resnet/resnet50_8xb32-300e_in1k.py` with content as below:

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

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

(continues on next page)

(continued from previous page)

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

4.3.1 Use intermediate variables in configs

Some intermediate variables are used in the configuration file. The intermediate variables make the configuration file clearer and easier to modify.

For example, `train_pipeline` / `test_pipeline` is the intermediate variable of the data pipeline. We first need to define `train_pipeline` / `test_pipeline`, and then pass them to `data`. If you want to modify the size of the input image during training and testing, you need to modify the intermediate variables of `train_pipeline` / `test_pipeline`.

```
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='RandomResizedCrop', size=384, backend='pillow',),
    dict(type='RandomFlip', flip_prob=0.5, direction='horizontal'),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='ImageToTensor', keys=['img']),
    dict(type='ToTensor', keys=['gt_label']),
    dict(type='Collect', keys=['img', 'gt_label'])
]
test_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='Resize', size=384, backend='pillow'),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='ImageToTensor', keys=['img']),
    dict(type='Collect', keys=['img'])
]
data = dict(
    train=dict(pipeline=train_pipeline),
    val=dict(pipeline=test_pipeline),
    test=dict(pipeline=test_pipeline))
```

4.3.2 Ignore some fields in the base configs

Sometimes, you need to set `_delete_=True` to ignore some domain content in the basic configuration file. You can refer to mmcv for more instructions.

The following is an example. If you want to use cosine schedule in the above ResNet50 case, just using inheritance and directly modify it will report `get unexcepected keyword'step'` error, because the `'step'` field of the basic config in `lr_config` domain information is reserved, and you need to add `_delete_ =True` to ignore the content of `lr_config` related fields in the basic configuration file:

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

```

(continues on next page)

(continued from previous page)

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

4.3.3 Use some fields in the base configs

Sometimes, you may refer to some fields in the `_base_` config, so as to avoid duplication of definitions. You can refer to mmcv for some more instructions.

The following is an example of using auto augment in the training data preprocessing pipeline, refer to `configs/_base_/datasets/imagenet_bs64_autoaug.py`. When defining `train_pipeline`, just add the definition file name of auto augment to `_base_`, and then use `{{_base_.auto_increasing_policies}}` to reference the variables:

```
_base_ = ['./pipelines/auto_aug.py']

# dataset settings
dataset_type = 'ImageNet'
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='RandomResizedCrop', size=224),
    dict(type='RandomFlip', flip_prob=0.5, direction='horizontal'),
    dict(type='AutoAugment', policies={{_base_.auto_increasing_policies}}),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='ImageToTensor', keys=['img']),
    dict(type='ToTensor', keys=['gt_label']),
    dict(type='Collect', keys=['img', 'gt_label'])
]
test_pipeline = [...]
data = dict(
    samples_per_gpu=64,
    workers_per_gpu=2,
    train=dict(..., pipeline=train_pipeline),
    val=dict(..., pipeline=test_pipeline))
evaluation = dict(interval=1, metric='accuracy')
```

4.4 Modify config through script arguments

When users use the script "tools/train.py" or "tools/test.py" to submit tasks or use some other tools, they can directly modify the content of the configuration file used by specifying the `--cfg-options` parameter.

- Update config keys of dict chains.

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

- Update keys inside a list of configs.

  Some config dicts are composed as a list in your config. For example, the training pipeline `data.train.pipeline` is normally a list e.g. `[dict(type='LoadImageFromFile'), dict(type='TopDownRandomFlip', flip_prob=0.5), ...]`. If you want to change `'flip_prob=0.5'` to `'flip_prob=0.0'` in the pipeline, you may specify `--cfg-options data.train.pipeline.1.flip_prob=0.0`.

- Update values of list/tuples.

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

4.5 Import user-defined modules

Note: This part may only be used when using MMClassification as a third party library to build your own project, and beginners can skip it.

After studying the follow-up tutorials ADDING NEW DATASET, CUSTOM DATA PIPELINES, ADDING NEW MODULES. You may use MMClassification to complete your project and create new classes of datasets, models, data enhancements, etc. in the project. In order to streamline the code, you can use MMClassification as a third-party library, you just need to keep your own extra code and import your own custom module in the configuration files. For examples, you may refer to OpenMMLab Algorithm Competition Project .

Add the following code to your own configuration files:

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

4.6 FAQ

- None

CHAPTER

FIVE

TUTORIAL 2: FINE-TUNE MODELS

Classification models pre-trained on the ImageNet dataset have been demonstrated to be effective for other datasets and other downstream tasks. This tutorial provides instructions for users to use the models provided in the *Model Zoo* for other datasets to obtain better performance.

There are two steps to fine-tune a model on a new dataset.

- Add support for the new dataset following *Tutorial 3: Customize Dataset*.
- Modify the configs as will be discussed in this tutorial.

Assume we have a ResNet-50 model pre-trained on the ImageNet-2012 dataset and want to take the fine-tuning on the CIFAR-10 dataset, we need to modify five parts in the config.

5.1 Inherit base configs

At first, create a new config file `configs/tutorial/resnet50_finetune_cifar.py` to store our configs. Of course, the path can be customized by yourself.

To reuse the common parts among different configs, we support inheriting configs from multiple existing configs. To fine-tune a ResNet-50 model, the new config needs to inherit `configs/_base_/models/resnet50.py` to build the basic structure of the model. To use the CIFAR-10 dataset, the new config can also simply inherit `configs/_base_/datasets/cifar10_bs16.py`. For runtime settings such as training schedules, the new config needs to inherit `configs/_base_/default_runtime.py`.

To inherit all above configs, put the following code at the config file.

```
_base_ = [
    '../_base_/models/resnet50.py',
    '../_base_/datasets/cifar10_bs16.py', '../_base_/default_runtime.py'
]
```

Besides, you can also choose to write the whole contents rather than use inheritance, like `configs/lenet/lenet5_mnist.py`.

5.2 Modify model

When fine-tuning a model, usually we want to load the pre-trained backbone weights and train a new classification head.

To load the pre-trained backbone, we need to change the initialization config of the backbone and use `Pretrained` initialization function. Besides, in the `init_cfg`, we use `prefix='backbone'` to tell the initialization function to remove the prefix of keys in the checkpoint, for example, it will change `backbone.conv1` to `conv1`. And here we use an online checkpoint, it will be downloaded during training, you can also download the model manually and use a local path.

And then we need to modify the head according to the class numbers of the new datasets by just changing `num_classes` in the head.

```
model = dict(
    backbone=dict(
        init_cfg=dict(
            type='Pretrained',
            checkpoint='https://download.openmmlab.com/mmclassification/v0/resnet/
↪resnet50_8xb32_in1k_20210831-ea4938fc.pth',
            prefix='backbone',
        )),
    head=dict(num_classes=10),
)
```

Tip: Here we only need to set the part of configs we want to modify, because the inherited configs will be merged and get the entire configs.

Sometimes, we want to freeze the first several layers' parameters of the backbone, that will help the network to keep ability to extract low-level information learnt from pre-trained model. In MMClassification, you can simply specify how many layers to freeze by `frozen_stages` argument. For example, to freeze the first two layers' parameters, just use the following config:

```
model = dict(
    backbone=dict(
        frozen_stages=2,
        init_cfg=dict(
            type='Pretrained',
            checkpoint='https://download.openmmlab.com/mmclassification/v0/resnet/
↪resnet50_8xb32_in1k_20210831-ea4938fc.pth',
            prefix='backbone',
        )),
    head=dict(num_classes=10),
)
```

Note: Not all backbones support the `frozen_stages` argument by now. Please check the docs to confirm if your backbone supports it.

5.3 Modify dataset

When fine-tuning on a new dataset, usually we need to modify some dataset configs. Here, we need to modify the pipeline to resize the image from 32 to 224 to fit the input size of the model pre-trained on ImageNet, and some other configs.

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

5.4 Modify training schedule

The fine-tuning hyper parameters vary from the default schedule. It usually requires smaller learning rate and less training epochs.

```
# lr is set for a batch size of 128
optimizer = dict(type='SGD', lr=0.01, momentum=0.9, weight_decay=0.0001)
optimizer_config = dict(grad_clip=None)
# learning policy
lr_config = dict(policy='step', step=[15])
runner = dict(type='EpochBasedRunner', max_epochs=200)
log_config = dict(interval=100)
```

5.5 Start Training

Now, we have finished the fine-tuning config file as following:

```
_base_ = [
    '../_base_/models/resnet50.py',
    '../_base_/datasets/cifar10_bs16.py', '../_base_/default_runtime.py'
]

# Model config
model = dict(
    backbone=dict(
        frozen_stages=2,
        init_cfg=dict(
            type='Pretrained',
            checkpoint='https://download.openmmlab.com/mmclassification/v0/resnet/
↪resnet50_8xb32_in1k_20210831-ea4938fc.pth',
            prefix='backbone',
        )),
    head=dict(num_classes=10),
)

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

# Training schedule config
# lr is set for a batch size of 128
optimizer = dict(type='SGD', lr=0.01, momentum=0.9, weight_decay=0.0001)
optimizer_config = dict(grad_clip=None)
```

(continues on next page)

(continued from previous page)

```
# learning policy
lr_config = dict(policy='step', step=[15])
runner = dict(type='EpochBasedRunner', max_epochs=200)
log_config = dict(interval=100)
```

Here we use 8 GPUs on your computer to train the model with the following command:

```
bash tools/dist_train.sh configs/tutorial/resnet50_finetune_cifar.py 8
```

Also, you can use only one GPU to train the model with the following command:

```
python tools/train.py configs/tutorial/resnet50_finetune_cifar.py
```

But wait, an important config need to be changed if using one GPU. We need to change the dataset config as following:

```
data = dict(
    samples_per_gpu=128,
    train=dict(pipeline=train_pipeline),
    val=dict(pipeline=test_pipeline),
    test=dict(pipeline=test_pipeline),
)
```

It's because our training schedule is for a batch size of 128. If using 8 GPUs, just use `samples_per_gpu=16` config in the base config file, and the total batch size will be 128. But if using one GPU, you need to change it to 128 manually to match the training schedule.

CHAPTER

SIX

TUTORIAL 3: CUSTOMIZE DATASET

We support many common public datasets for image classification task, you can find them in this page.

In this section, we demonstrate how to *use your own dataset* and *use dataset wrapper*.

6.1 Use your own dataset

6.1.1 Reorganize dataset to existing format

The simplest way to use your own dataset is to convert it to existing dataset formats.

For multi-class classification task, we recommend to use the format of `CustomDataset`.

The `CustomDataset` supports two kinds of format:

1. An annotation file is provided, and each line indicates a sample image.

   The sample images can be organized in any structure, like:

   ```
   train/
   ├── folder_1
   │   ├── xxx.png
   │   ├── xxy.png
   │   └── ...
   ├── 123.png
   ├── nsdf3.png
   └── ...
   ```

   And an annotation file records all paths of samples and corresponding category index. The first column is the image path relative to the folder (in this example, `train`) and the second column is the index of category:

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

   Note: The value of the category indices should fall in range `[0, num_classes - 1]`.

2. The sample images are arranged in the special structure:

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

In this case, you don't need provide annotation file, and all images in the directory `cat` will be recognized as samples of `cat`.

Usually, we will split the whole dataset to three sub datasets: `train`, `val` and `test` for training, validation and test. And **every** sub dataset should be organized as one of the above structures.

For example, the whole dataset is as below (using the first structure):

```
mmclassification
└── data
    └── my_dataset
        ├── meta
        │   ├── train.txt
        │   ├── val.txt
        │   └── test.txt
        ├── train
        ├── val
        └── test
```

And in your config file, you can modify the `data` field as below:

```
...
dataset_type = 'CustomDataset'
classes = ['cat', 'bird', 'dog']  # The category names of your dataset

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

(continues on next page)

(continued from previous page)

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

6.1.2 Create a new dataset class

You can write a new dataset class inherited from `BaseDataset`, and overwrite `load_annotations(self)`, like CIFAR10 and CustomDataset.

Typically, this function returns a list, where each sample is a dict, containing necessary data information, e.g., `img` and `gt_label`.

Assume we are going to implement a `Filelist` dataset, which takes filelists for both training and testing. The format of annotation list is as follows:

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

We can create a new dataset in `mmcls/datasets/filelist.py` to load the data.

```
import mmcv
import numpy as np

from .builder import DATASETS
from .base_dataset import BaseDataset


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

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

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

And add this dataset class in `mmcls/datasets/__init__.py`

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

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

Then in the config, to use `Filelist` you can modify the config as the following

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

6.2 Use dataset wrapper

The dataset wrapper is a kind of class to change the behavior of dataset class, such as repeat the dataset or re-balance the samples of different categories.

6.2.1 Repeat dataset

We use `RepeatDataset` as wrapper to repeat the dataset. For example, suppose the original dataset is `Dataset_A`, to repeat it, the config looks like the following

```
data = dict(
    train = dict(
        type='RepeatDataset',
        times=N,
        dataset=dict(  # This is the original config of Dataset_A
            type='Dataset_A',
            ...
            pipeline=train_pipeline
        )
    )
    ...
)
```

6.2.2 Class balanced dataset

We use `ClassBalancedDataset` as wrapper to repeat the dataset based on category frequency. The dataset to repeat needs to implement method `get_cat_ids(idx)` to support `ClassBalancedDataset`. For example, to repeat `Dataset_A` with `oversample_thr=1e-3`, the config looks like the following

```
data = dict(
    train = dict(
        type='ClassBalancedDataset',
        oversample_thr=1e-3,
```

(continues on next page)

(continued from previous page)

```
        dataset=dict(  # This is the original config of Dataset_A
            type='Dataset_A',
            ...
            pipeline=train_pipeline
        )
    )
    ...
)
```

You may refer to API reference for details.

CHAPTER

SEVEN

TUTORIAL 4: CUSTOM DATA PIPELINES

7.1 Design of Data pipelines

Following typical conventions, we use `Dataset` and `DataLoader` for data loading with multiple workers. Indexing `Dataset` returns a dict of data items corresponding to the arguments of models forward method.

The data preparation pipeline and the dataset is decomposed. Usually a dataset defines how to process the annotations and a data pipeline defines all the steps to prepare a data dict. A pipeline consists of a sequence of operations. Each operation takes a dict as input and also output a dict for the next transform.

The operations are categorized into data loading, pre-processing and formatting.

Here is an pipeline example for ResNet-50 training on ImageNet.

```
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='RandomResizedCrop', size=224),
    dict(type='RandomFlip', flip_prob=0.5, direction='horizontal'),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='ImageToTensor', keys=['img']),
    dict(type='ToTensor', keys=['gt_label']),
    dict(type='Collect', keys=['img', 'gt_label'])
]
test_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='Resize', size=256),
    dict(type='CenterCrop', crop_size=224),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='ImageToTensor', keys=['img']),
    dict(type='Collect', keys=['img'])
]
```

For each operation, we list the related dict fields that are added/updated/removed. At the end of the pipeline, we use `Collect` to only retain the necessary items for forward computation.

7.1.1 Data loading

`LoadImageFromFile`

- add: img, img_shape, ori_shape

By default, `LoadImageFromFile` loads images from disk but it may lead to IO bottleneck for efficient small models. Various backends are supported by mmcv to accelerate this process. For example, if the training machines have setup memcached, we can revise the config as follows.

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

More supported backends can be found in mmcv.fileio.FileClient.

7.1.2 Pre-processing

`Resize`

- add: scale, scale_idx, pad_shape, scale_factor, keep_ratio
- update: img, img_shape

`RandomFlip`

- add: flip, flip_direction
- update: img

`RandomCrop`

- update: img, pad_shape

`Normalize`

- add: img_norm_cfg
- update: img

7.1.3 Formatting

`ToTensor`

- update: specified by `keys`.

`ImageToTensor`

- update: specified by `keys`.

`Collect`

- remove: all other keys except for those specified by `keys`

For more information about other data transformation classes, please refer to *Data Transformations*

7.2 Extend and use custom pipelines

1. Write a new pipeline in any file, e.g., `my_pipeline.py`, and place it in the folder `mmcls/datasets/pipelines/`. The pipeline class needs to override the `__call__` method which takes a dict as input and returns a dict.

```
from mmcls.datasets import PIPELINES

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

    def __call__(self, results):
        # apply transforms on results['img']
        return results
```

2. Import the new class in `mmcls/datasets/pipelines/__init__.py`.

```
...
from .my_pipeline import MyTransform

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

3. Use it in config files.

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

7.3 Pipeline visualization

After designing data pipelines, you can use the *visualization tools* to view the performance.

CHAPTER

EIGHT

TUTORIAL 5: ADDING NEW MODULES

8.1 Develop new components

We basically categorize model components into 3 types.

- backbone: usually an feature extraction network, e.g., ResNet, MobileNet.
- neck: the component between backbones and heads, e.g., GlobalAveragePooling.
- head: the component for specific tasks, e.g., classification or regression.

8.1.1 Add new backbones

Here we show how to develop new components with an example of ResNet_CIFAR. As the input size of CIFAR is 32x32, this backbone replaces the `kernel_size=7, stride=2` to `kernel_size=3, stride=1` and remove the MaxPooling after stem, to avoid forwarding small feature maps to residual blocks. It inherits from ResNet and only modifies the stem layers.

1. Create a new file `mmcls/models/backbones/resnet_cifar.py`.

```
import torch.nn as nn

from ..builder import BACKBONES
from .resnet import ResNet


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

    """ResNet backbone for CIFAR.

    short description of the backbone

    Args:
        depth(int): Network depth, from {18, 34, 50, 101, 152}.
        ...
    """

    def __init__(self, depth, deep_stem, **kwargs):
        # call ResNet init
        super(ResNet_CIFAR, self).__init__(depth, deep_stem=deep_stem, **kwargs)
        # other specific initialization
```

(continues on next page)

(continued from previous page)

```
        assert not self.deep_stem, 'ResNet_CIFAR do not support deep_stem'

    def _make_stem_layer(self, in_channels, base_channels):
        # override ResNet method to modify the network structure
        self.conv1 = build_conv_layer(
            self.conv_cfg,
            in_channels,
            base_channels,
            kernel_size=3,
            stride=1,
            padding=1,
            bias=False)
        self.norm1_name, norm1 = build_norm_layer(
            self.norm_cfg, base_channels, postfix=1)
        self.add_module(self.norm1_name, norm1)
        self.relu = nn.ReLU(inplace=True)

    def forward(self, x):  # should return a tuple
        pass  # implementation is ignored

    def init_weights(self, pretrained=None):
        pass  # override ResNet init_weights if necessary

    def train(self, mode=True):
        pass  # override ResNet train if necessary
```

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

```
...
from .resnet_cifar import ResNet_CIFAR

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

3. Use it in your config file.

```
model = dict(
    ...
    backbone=dict(
        type='ResNet_CIFAR',
        depth=18,
        other_arg=xxx),
    ...
```

8.1.2 Add new necks

Here we take `GlobalAveragePooling` as an example. It is a very simple neck without any arguments. To add a new neck, we mainly implement the `forward` function, which applies some operation on the output from backbone and forward the results to head.

1. Create a new file in `mmcls/models/necks/gap.py`.

```python
import torch.nn as nn

from ..builder import NECKS

@NECKS.register_module()
class GlobalAveragePooling(nn.Module):

    def __init__(self):
        self.gap = nn.AdaptiveAvgPool2d((1, 1))

    def forward(self, inputs):
        # we regard inputs as tensor for simplicity
        outs = self.gap(inputs)
        outs = outs.view(inputs.size(0), -1)
        return outs
```

2. Import the module in `mmcls/models/necks/__init__.py`.

```python
...
from .gap import GlobalAveragePooling

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

3. Modify the config file.

```python
model = dict(
    neck=dict(type='GlobalAveragePooling'),
)
```

8.1.3 Add new heads

Here we show how to develop a new head with the example of `LinearClsHead` as the following. To implement a new head, basically we need to implement `forward_train`, which takes the feature maps from necks or backbones as input and compute loss based on ground-truth labels.

1. Create a new file in `mmcls/models/heads/linear_head.py`.

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


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

```

(continues on next page)

(continued from previous page)

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

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

        self._init_layers()

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

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

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

2. Import the module in `mmcls/models/heads/__init__.py`.

```
...
from .linear_head import LinearClsHead

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

3. Modify the config file.

Together with the added GlobalAveragePooling neck, an entire config for a model is as follows.

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

(continues on next page)

(continued from previous page)

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

8.1.4 Add new loss

To add a new loss function, we mainly implement the `forward` function in the loss module. In addition, it is helpful to leverage the decorator `weighted_loss` to weight the loss for each element. Assuming that we want to mimic a probabilistic distribution generated from another classification model, we implement a L1Loss to fulfil the purpose as below.

1. Create a new file in `mmcls/models/losses/l1_loss.py`.

```
import torch
import torch.nn as nn

from ..builder import LOSSES
from .utils import weighted_loss

@weighted_loss
def l1_loss(pred, target):
    assert pred.size() == target.size() and target.numel() > 0
    loss = torch.abs(pred - target)
    return loss

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

    def __init__(self, reduction='mean', loss_weight=1.0):
        super(L1Loss, self).__init__()
        self.reduction = reduction
        self.loss_weight = loss_weight

    def forward(self,
                pred,
                target,
                weight=None,
                avg_factor=None,
                reduction_override=None):
        assert reduction_override in (None, 'none', 'mean', 'sum')
        reduction = (
            reduction_override if reduction_override else self.reduction)
        loss = self.loss_weight * l1_loss(
            pred, target, weight, reduction=reduction, avg_factor=avg_factor)
        return loss
```

2. Import the module in `mmcls/models/losses/__init__.py`.

```
...
from .l1_loss import L1Loss, l1_loss
```

(continues on next page)

(continued from previous page)

```python

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

3. Modify loss field in the config.

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

CHAPTER

NINE

TUTORIAL 6: CUSTOMIZE SCHEDULE

In this tutorial, we will introduce some methods about how to construct optimizers, customize learning rate and momentum schedules, parameter-wise finely configuration, gradient clipping, gradient accumulation, and customize self-implemented methods for the project.

9.1 Customize optimizer supported by PyTorch

We already support to use all the optimizers implemented by PyTorch, and to use and modify them, please change the `optimizer` field of config files.

For example, if you want to use `SGD`, the modification could be as the following.

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

To modify the learning rate of the model, just modify the `lr` in the config of optimizer. You can also directly set other arguments according to the API doc of PyTorch.

For example, if you want to use `Adam` with the setting like `torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsgrad=False)` in PyTorch, the config should looks like.

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

9.2 Customize learning rate schedules

9.2.1 Learning rate decay

Learning rate decay is widely used to improve performance. And to use learning rate decay, please set the `lr_confg` field in config files.

For example, we use step policy as the default learning rate decay policy of ResNet, and the config is:

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

Then during training, the program will call `StepLRHook` periodically to update the learning rate.

We also support many other learning rate schedules here, such as `CosineAnnealing` and `Poly` schedule. Here are some examples

- ConsineAnnealing schedule:

```
lr_config = dict(
    policy='CosineAnnealing',
    warmup='linear',
    warmup_iters=1000,
    warmup_ratio=1.0 / 10,
    min_lr_ratio=1e-5)
```

- Poly schedule:

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

9.2.2 Warmup strategy

In the early stage, training is easy to be volatile, and warmup is a technique to reduce volatility. With warmup, the learning rate will increase gradually from a minor value to the expected value.

In MMClassification, we use `lr_config` to configure the warmup strategy, the main parameters are as follows:

- `warmup`: The warmup curve type. Please choose one from 'constant', 'linear', 'exp' and `None`, and `None` means disable warmup.
- `warmup_by_epoch` : if warmup by epoch or not, default to be True, if set to be False, warmup by iter.
- `warmup_iters` : the number of warm-up iterations, when `warmup_by_epoch=True`, the unit is epoch; when `warmup_by_epoch=False`, the unit is the number of iterations (iter).
- `warmup_ratio` : warm-up initial learning rate will calculate as `lr = lr * warmup_ratio`。

Here are some examples

1. linear & warmup by iter

```
lr_config = dict(
    policy='CosineAnnealing',
    by_epoch=False,
    min_lr_ratio=1e-2,
    warmup='linear',
    warmup_ratio=1e-3,
```

(continues on next page)

(continued from previous page)

```
    warmup_iters=20 * 1252,
    warmup_by_epoch=False)
```

2. exp & warmup by epoch

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

Tip: After completing your configuration file， you could use learning rate visualization tool to draw the corresponding learning rate adjustment curve.

9.3 Customize momentum schedules

We support the momentum scheduler to modify the model's momentum according to learning rate, which could make the model converge in a faster way.

Momentum scheduler is usually used with LR scheduler, for example, the following config is used to accelerate convergence. For more details, please refer to the implementation of CyclicLrUpdater and CyclicMomentumUpdater.

Here is an example

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

9.4 Parameter-wise finely configuration

Some models may have some parameter-specific settings for optimization, for example, no weight decay to the BatchNorm layer or using different learning rates for different network layers. To finely configuration them, we can use the `paramwise_cfg` option in `optimizer`.

We provide some examples here and more usages refer to DefaultOptimizerConstructor.

- Using specified options

The DefaultOptimizerConstructor provides options including bias_lr_mult, bias_decay_mult, norm_decay_mult, dwconv_decay_mult, dcn_offset_lr_mult and bypass_duplicate to configure special optimizer behaviors of bias, normalization, depth-wise convolution, deformable convolution and duplicated parameter. E.g:

1. No weight decay to the BatchNorm layer

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

- Using custom_keys dict

  MMClassification can use custom_keys to specify different parameters to use different learning rates or weight decays, for example:

1. No weight decay for specific parameters

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

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

2. Using a smaller learning rate and a weight decay for the backbone layers

```
optimizer = dict(
    type='SGD',
    lr=0.8,
    weight_decay=1e-4,
    # 'lr' for backbone and 'weight_decay' are 0.1 * lr and 0.9 * weight_decay
    paramwise_cfg=dict(
        custom_keys={'backbone': dict(lr_mult=0.1, decay_mult=0.9)}))
```

9.5 Gradient clipping and gradient accumulation

Besides the basic function of PyTorch optimizers, we also provide some enhancement functions, such as gradient clipping, gradient accumulation, etc., refer to MMCV.

9.5.1 Gradient clipping

During the training process, the loss function may get close to a cliffy region and cause gradient explosion. And gradient clipping is helpful to stabilize the training process. More introduction can be found in this page.

Currently we support `grad_clip` option in `optimizer_config`, and the arguments refer to PyTorch Documentation.

Here is an example:

```
optimizer_config = dict(grad_clip=dict(max_norm=35, norm_type=2))
# norm_type: type of the used p-norm, here norm_type is 2.
```

When inheriting from base and modifying configs, if `grad_clip=None` in base, `_delete_=True` is needed. For more details about `_delete_` you can refer to TUTORIAL 1: LEARN ABOUT CONFIGS. For example,

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

optimizer_config = dict(grad_clip=dict(max_norm=35, norm_type=2), _delete_=True, type=
↪'OptimizerHook')
# you can ignore type if type is 'OptimizerHook', otherwise you must add "type=
↪'xxxxxOptimizerHook'" here
```

9.5.2 Gradient accumulation

When computing resources are lacking, the batch size can only be set to a small value, which may affect the performance of models. Gradient accumulation can be used to solve this problem.

Here is an example:

```
data = dict(samples_per_gpu=64)
optimizer_config = dict(type="GradientCumulativeOptimizerHook", cumulative_iters=4)
```

Indicates that during training, back-propagation is performed every 4 iters. And the above is equivalent to:

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

Note: When the optimizer hook type is not specified in `optimizer_config`, `OptimizerHook` is used by default.

9.6 Customize self-implemented methods

In academic research and industrial practice, it may be necessary to use optimization methods not implemented by MMClassification, and you can add them through the following methods.

Note: This part will modify the MMClassification source code or add code to the MMClassification framework, beginners can skip it.

9.6.1 Customize self-implemented optimizer

1. Define a new optimizer

A customized optimizer could be defined as below.

Assume you want to add an optimizer named `MyOptimizer`, which has arguments `a`, `b`, and `c`. You need to create a new directory named `mmcls/core/optimizer`. And then implement the new optimizer in a file, e.g., in `mmcls/core/optimizer/my_optimizer.py`:

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


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

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

2. Add the optimizer to registry

To find the above module defined above, this module should be imported into the main namespace at first. There are two ways to achieve it.

- Modify `mmcls/core/optimizer/__init__.py` to import it into `optimizer` package, and then modify `mmcls/core/__init__.py` to import the new `optimizer` package.

  Create the `mmcls/core/optimizer` folder and the `mmcls/core/optimizer/__init__.py` file if they don't exist. The newly defined module should be imported in `mmcls/core/optimizer/__init__.py` and `mmcls/core/__init__.py` so that the registry will find the new module and add it:

```
# In mmcls/core/optimizer/__init__.py
from .my_optimizer import MyOptimizer # MyOptimizer maybe other class name

__all__ = ['MyOptimizer']
```

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

- Use `custom_imports` in the config to manually import it

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

The module `mmcls.core.optimizer.my_optimizer` will be imported at the beginning of the program and the class `MyOptimizer` is then automatically registered. Note that only the package containing the class `MyOptimizer` should be imported. `mmcls.core.optimizer.my_optimizer.MyOptimizer` **cannot** be imported directly.

3. Specify the optimizer in the config file

Then you can use `MyOptimizer` in `optimizer` field of config files. In the configs, the optimizers are defined by the field `optimizer` like the following:

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

To use your own optimizer, the field can be changed to

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

9.6.2 Customize optimizer constructor

Some models may have some parameter-specific settings for optimization, e.g. weight decay for BatchNorm layers.

Although our `DefaultOptimizerConstructor` is powerful, it may still not cover your need. If that, you can do those fine-grained parameter tuning through customizing optimizer constructor.

```
from mmcv.runner.optimizer import OPTIMIZER_BUILDERS


@OPTIMIZER_BUILDERS.register_module()
class MyOptimizerConstructor:

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

    def __call__(self, model):
        ...       # Construct your optimzier here.
        return my_optimizer
```

The default optimizer constructor is implemented here, which could also serve as a template for new optimizer constructor.

CHAPTER

TEN

TUTORIAL 7: CUSTOMIZE RUNTIME SETTINGS

In this tutorial, we will introduce some methods about how to customize workflow and hooks when running your own settings for the project.

10.1 Customize Workflow

Workflow is a list of (phase, duration) to specify the running order and duration. The meaning of "duration" depends on the runner's type.

For example, we use epoch-based runner by default, and the "duration" means how many epochs the phase to be executed in a cycle. Usually, we only want to execute training phase, just use the following config.

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

Sometimes we may want to check some metrics (e.g. loss, accuracy) about the model on the validate set. In such case, we can set the workflow as

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

so that 1 epoch for training and 1 epoch for validation will be run iteratively.

By default, we recommend using **`EvalHook`** to do evaluation after the training epoch, but you can still use `val` workflow as an alternative.

Note:

1. The parameters of model will not be updated during the val epoch.
2. Keyword `max_epochs` in the config only controls the number of training epochs and will not affect the validation workflow.
3. Workflows `[('train', 1), ('val', 1)]` and `[('train', 1)]` will not change the behavior of `EvalHook` because `EvalHook` is called by `after_train_epoch` and validation workflow only affect hooks that are called

through `after_val_epoch`. Therefore, the only difference between `[('train', 1), ('val', 1)]` and `[('train', 1)]` is that the runner will calculate losses on the validation set after each training epoch.

10.2 Hooks

The hook mechanism is widely used in the OpenMMLab open-source algorithm library. Combined with the `Runner`, the entire life cycle of the training process can be managed easily. You can learn more about the hook through related article.

Hooks only work after being registered into the runner. At present, hooks are mainly divided into two categories:

- default training hooks

The default training hooks are registered by the runner by default. Generally, they are hooks for some basic functions, and have a certain priority, you don't need to modify the priority.

- custom hooks

The custom hooks are registered through `custom_hooks`. Generally, they are hooks with enhanced functions. The priority needs to be specified in the configuration file. If you do not specify the priority of the hook, it will be set to 'NORMAL' by default.

Priority list

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

The priority determines the execution order of the hooks. Before training, the log will print out the execution order of the hooks at each stage to facilitate debugging.

10.2.1 default training hooks

Some common hooks are not registered through `custom_hooks`, they are

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

`OptimizerHook`, `MomentumUpdaterHook` and `LrUpdaterHook` have been introduced in *sehedule strategy*. `IterTimerHook` is used to record elapsed time and does not support modification.

Here we reveal how to customize `CheckpointHook`, `LoggerHooks`, and `EvalHook`.

CheckpointHook

The MMCV runner will use `checkpoint_config` to initialize `CheckpointHook`.

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

We could set `max_keep_ckpts` to save only a small number of checkpoints or decide whether to store state dict of optimizer by `save_optimizer`. More details of the arguments are here

LoggerHooks

The `log_config` wraps multiple logger hooks and enables to set intervals. Now MMCV supports `TextLoggerHook`, `WandbLoggerHook`, `MlflowLoggerHook`, `NeptuneLoggerHook`, `DvcliveLoggerHook` and `TensorboardLoggerHook`. The detailed usages can be found in the doc.

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

EvalHook

The config of `evaluation` will be used to initialize the `EvalHook`.

The `EvalHook` has some reserved keys, such as `interval`, `save_best` and `start`, and the other arguments such as `metrics` will be passed to the `dataset.evaluate()`

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

You can save the model weight when the best verification result is obtained by modifying the parameter `save_best`:

```
# "auto" means automatically select the metrics to compare.
# You can also use a specific key like "accuracy_top-1".
evaluation = dict(interval=1, save_best="auto", metric='accuracy', metric_options={'topk
↪': (1, )})
```

When running some large experiments, you can skip the validation step at the beginning of training by modifying the parameter `start` as below:

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

This indicates that, before the 200th epoch, evaluations would not be executed. Since the 200th epoch, evaluations would be executed after the training process.

Note: In the default configuration files of MMClassification, the evaluation field is generally placed in the datasets configs.

10.2.2 Use other implemented hooks

Some hooks have been already implemented in MMCV and MMClassification, they are:

- EMAHook
- SyncBuffersHook
- EmptyCacheHook
- ProfilerHook
- ……

If the hook is already implemented in MMCV, you can directly modify the config to use the hook as below

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

such as using `EMAHook`, interval is 100 iters:

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

10.3 Customize self-implemented hooks

10.3.1 1. Implement a new hook

Here we give an example of creating a new hook in MMClassification and using it in training.

```
from mmcv.runner import HOOKS, Hook


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

    def __init__(self, a, b):
        pass

    def before_run(self, runner):
        pass

    def after_run(self, runner):
        pass

    def before_epoch(self, runner):
```

(continues on next page)

(continued from previous page)

```
        pass

    def after_epoch(self, runner):
        pass

    def before_iter(self, runner):
        pass

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

Depending on the functionality of the hook, the users need to specify what the hook will do at each stage of the training in `before_run`, `after_run`, `before_epoch`, `after_epoch`, `before_iter`, and `after_iter`.

10.3.2 2. Register the new hook

Then we need to make `MyHook` imported. Assuming the file is in `mmcls/core/utils/my_hook.py` there are two ways to do that:

- Modify `mmcls/core/utils/__init__.py` to import it.

  The newly defined module should be imported in `mmcls/core/utils/__init__.py` so that the registry will find the new module and add it:

```
from .my_hook import MyHook
```

- Use `custom_imports` in the config to manually import it

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

10.3.3 3. Modify the config

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

You can also set the priority of the hook as below:

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

By default, the hook's priority is set as `NORMAL` during registration.

10.4 FAQ

10.4.1 1. `resume_from` and `load_from` and `init_cfg.Pretrained`

- `load_from` : only imports model weights, which is mainly used to load pre-trained or trained models;
- `resume_from` : not only import model weights, but also optimizer information, current epoch information, mainly used to continue training from the checkpoint.
- `init_cfg.Pretrained` : Load weights during weight initialization, and you can specify which module to load. This is usually used when fine-tuning a model, refer to *Tutorial 2: Fine-tune Models*.

CHAPTER

ELEVEN

MODEL ZOO SUMMARY

- Number of papers: 34
  - ALGORITHM: 34
- Number of checkpoints: 224
  - [ALGORITHM] *Conformer: Local Features Coupling Global Representations for Visual Recognition* (4 ckpts)
  - [ALGORITHM] *Patches Are All You Need?* (3 ckpts)
  - [ALGORITHM] *A ConvNet for the 2020s* (13 ckpts)
  - [ALGORITHM] *CSPNet: A New Backbone that can Enhance Learning Capability of CNN* (3 ckpts)
  - [ALGORITHM] *Residual Attention: A Simple but Effective Method for Multi-Label Recognition* (1 ckpts)
  - [ALGORITHM] *Training data-efficient image transformers & distillation through attention* (9 ckpts)
  - [ALGORITHM] *Densely Connected Convolutional Networks* (4 ckpts)
  - [ALGORITHM] *EfficientFormer: Vision Transformers at MobileNet Speed* (3 ckpts)
  - [ALGORITHM] *Rethinking Model Scaling for Convolutional Neural Networks* (23 ckpts)
  - [ALGORITHM] *HorNet: Efficient High-Order Spatial Interactions with Recursive Gated Convolutions* (9 ckpts)
  - [ALGORITHM] *Deep High-Resolution Representation Learning for Visual Recognition* (9 ckpts)
  - [ALGORITHM] *MLP-Mixer: An all-MLP Architecture for Vision* (2 ckpts)
  - [ALGORITHM] *MobileNetV2: Inverted Residuals and Linear Bottlenecks* (1 ckpts)
  - [ALGORITHM] *Searching for MobileNetV3* (2 ckpts)
  - [ALGORITHM] *MViTv2: Improved Multiscale Vision Transformers for Classification and Detection* (4 ckpts)
  - [ALGORITHM] *MetaFormer is Actually What You Need for Vision* (5 ckpts)
  - [ALGORITHM] *Designing Network Design Spaces* (16 ckpts)
  - [ALGORITHM] *RepMLP: Re-parameterizing Convolutions into Fully-connected Layers forImage Recognition* (2 ckpts)
  - [ALGORITHM] *Repvgg: Making vgg-style convnets great again* (12 ckpts)
  - [ALGORITHM] *Res2Net: A New Multi-scale Backbone Architecture* (3 ckpts)
  - [ALGORITHM] *Deep Residual Learning for Image Recognition* (26 ckpts)
  - [ALGORITHM] *Aggregated Residual Transformations for Deep Neural Networks* (4 ckpts)

  - [ALGORITHM] *Squeeze-and-Excitation Networks* (2 ckpts)
  - [ALGORITHM] *ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices* (1 ckpts)
  - [ALGORITHM] *Shufflenet v2: Practical guidelines for efficient cnn architecture design* (1 ckpts)
  - [ALGORITHM] *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows* (14 ckpts)
  - [ALGORITHM] *Swin Transformer V2: Scaling Up Capacity and Resolution* (12 ckpts)
  - [ALGORITHM] *Tokens-to-Token ViT: Training Vision Transformers from Scratch on ImageNet* (3 ckpts)
  - [ALGORITHM] *Transformer in Transformer* (1 ckpts)
  - [ALGORITHM] *Twins: Revisiting the Design of Spatial Attention in Vision Transformers* (6 ckpts)
  - [ALGORITHM] *Visual Attention Network* (8 ckpts)
  - [ALGORITHM] *Very Deep Convolutional Networks for Large-Scale Image Recognition* (8 ckpts)
  - [ALGORITHM] *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale* (7 ckpts)
  - [ALGORITHM] *Wide Residual Networks* (3 ckpts)

CHAPTER

TWELVE

MODEL ZOO

12.1 ImageNet

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

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	C
VGG-11	132.86	7.63	68.75	88.87	c
VGG-13	133.05	11.34	70.02	89.46	c
VGG-16	138.36	15.5	71.62	90.49	c
VGG-19	143.67	19.67	72.41	90.80	c
VGG-11-BN	132.87	7.64	70.75	90.12	c
VGG-13-BN	133.05	11.36	72.15	90.71	c
VGG-16-BN	138.37	15.53	73.72	91.68	c
VGG-19-BN	143.68	19.7	74.70	92.24	c
RepVGG-A0*	9.11 （train) | 8.31 (deploy)	1.52 (train) | 1.36 (deploy)	72.41	90.50	c
RepVGG-A1*	14.09 (train) | 12.79 (deploy)	2.64 (train) | 2.37 (deploy)	74.47	91.85	c
RepVGG-A2*	28.21 (train) | 25.5 (deploy)	5.7 (train) | 5.12 (deploy)	76.48	93.01	c
RepVGG-B0*	15.82 (train) | 14.34 (deploy)	3.42 (train) | 3.06 (deploy)	75.14	92.42	c
RepVGG-B1*	57.42 (train) | 51.83 (deploy)	13.16 (train) | 11.82 (deploy)	78.37	94.11	c
RepVGG-B1g2*	45.78 (train) | 41.36 (deploy)	9.82 (train) | 8.82 (deploy)	77.79	93.88	c
RepVGG-B1g4*	39.97 (train) | 36.13 (deploy)	8.15 (train) | 7.32 (deploy)	77.58	93.84	c
RepVGG-B2*	89.02 (train) | 80.32 (deploy)	20.46 (train) | 18.39 (deploy)	78.78	94.42	c
RepVGG-B2g4*	61.76 (train) | 55.78 (deploy)	12.63 (train) | 11.34 (deploy)	79.38	94.68	c
RepVGG-B3*	123.09 (train) | 110.96 (deploy)	29.17 (train) | 26.22 (deploy)	80.52	95.26	c
RepVGG-B3g4*	83.83 (train) | 75.63 (deploy)	17.9 (train) | 16.08 (deploy)	80.22	95.10	c
RepVGG-D2se*	133.33 (train) | 120.39 (deploy)	36.56 (train) | 32.85 (deploy)	81.81	95.94	c
ResNet-18	11.69	1.82	70.07	89.44	c
ResNet-34	21.8	3.68	73.85	91.53	c
ResNet-50 (rsb-a1)	25.56	4.12	80.12	94.78	c
ResNet-101	44.55	7.85	78.18	94.03	c
ResNet-152	60.19	11.58	78.63	94.16	c
Res2Net-50-14w-8s*	25.06	4.22	78.14	93.85	c
Res2Net-50-26w-8s*	48.40	8.39	79.20	94.36	c
Res2Net-101-26w-4s*	45.21	8.12	79.19	94.44	c
ResNeSt-50*	27.48	5.41	81.13	95.59	c
ResNeSt-101*	48.28	10.27	82.32	96.24	c
ResNeSt-200*	70.2	17.53	82.41	96.22	c
ResNeSt-269*	110.93	22.58	82.70	96.28	c

Table 1 – continued from previous page

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	C
ResNetV1D-50	25.58	4.36	77.54	93.57	c
ResNetV1D-101	44.57	8.09	78.93	94.48	c
ResNetV1D-152	60.21	11.82	79.41	94.7	c
ResNeXt-32x4d-50	25.03	4.27	77.90	93.66	c
ResNeXt-32x4d-101	44.18	8.03	78.71	94.12	c
ResNeXt-32x8d-101	88.79	16.5	79.23	94.58	c
ResNeXt-32x4d-152	59.95	11.8	78.93	94.41	c
SE-ResNet-50	28.09	4.13	77.74	93.84	c
SE-ResNet-101	49.33	7.86	78.26	94.07	c
RegNetX-400MF	5.16	0.41	72.56	90.78	c
RegNetX-800MF	7.26	0.81	74.76	92.32	c
RegNetX-1.6GF	9.19	1.63	76.84	93.31	c
RegNetX-3.2GF	15.3	3.21	78.09	94.08	c
RegNetX-4.0GF	22.12	4.0	78.60	94.17	c
RegNetX-6.4GF	26.21	6.51	79.38	94.65	c
RegNetX-8.0GF	39.57	8.03	79.12	94.51	c
RegNetX-12GF	46.11	12.15	79.67	95.03	c
ShuffleNetV1 1.0x (group=3)	1.87	0.146	68.13	87.81	c
ShuffleNetV2 1.0x	2.28	0.149	69.55	88.92	c
MobileNet V2	3.5	0.319	71.86	90.42	c
ViT-B/16*	86.86	33.03	85.43	97.77	c
ViT-B/32*	88.3	8.56	84.01	97.08	c
ViT-L/16*	304.72	116.68	85.63	97.63	c
Swin-Transformer tiny	28.29	4.36	81.18	95.61	c
Swin-Transformer small	49.61	8.52	83.02	96.29	c
Swin-Transformer base	87.77	15.14	83.36	96.44	c
Transformer in Transformer small*	23.76	3.36	81.52	95.73	c
T2T-ViT_t-14	21.47	4.34	81.83	95.84	c
T2T-ViT_t-19	39.08	7.80	82.63	96.18	c
T2T-ViT_t-24	64.00	12.69	82.71	96.09	c
Mixer-B/16*	59.88	12.61	76.68	92.25	c
Mixer-L/16*	208.2	44.57	72.34	88.02	c
DeiT-tiny	5.72	1.08	74.50	92.24	c
DeiT-tiny distilled*	5.72	1.08	74.51	91.90	c
DeiT-small	22.05	4.24	80.69	95.06	c
DeiT-small distilled*	22.05	4.24	81.17	95.40	c
DeiT-base	86.57	16.86	81.76	95.81	c
DeiT-base distilled*	86.57	16.86	83.33	96.49	c
DeiT-base 384px*	86.86	49.37	83.04	96.31	c
DeiT-base distilled 384px*	86.86	49.37	85.55	97.35	c
Conformer-tiny-p16*	23.52	4.90	81.31	95.60	c
Conformer-small-p32*	38.85	7.09	81.96	96.02	c
Conformer-small-p16*	37.67	10.31	83.32	96.46	c
Conformer-base-p16*	83.29	22.89	83.82	96.59	c
PCPVT-small*	24.11	3.67	81.14	95.69	c
PCPVT-base*	43.83	6.45	82.66	96.26	c
PCPVT-large*	60.99	9.51	83.09	96.59	c
SVT-small*	24.06	2.82	81.77	95.57	c
SVT-base*	56.07	8.35	83.13	96.29	c

Table 1 – continued from previous page

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	C
SVT-large*	99.27	14.82	83.60	96.50	c
EfficientNet-B0*	5.29	0.02	76.74	93.17	c
EfficientNet-B0 (AA)*	5.29	0.02	77.26	93.41	c
EfficientNet-B0 (AA + AdvProp)*	5.29	0.02	77.53	93.61	c
EfficientNet-B1*	7.79	0.03	78.68	94.28	c
EfficientNet-B1 (AA)*	7.79	0.03	79.20	94.42	c
EfficientNet-B1 (AA + AdvProp)*	7.79	0.03	79.52	94.43	c
EfficientNet-B2*	9.11	0.03	79.64	94.80	c
EfficientNet-B2 (AA)*	9.11	0.03	80.21	94.96	c
EfficientNet-B2 (AA + AdvProp)*	9.11	0.03	80.45	95.07	c
EfficientNet-B3*	12.23	0.06	81.01	95.34	c
EfficientNet-B3 (AA)*	12.23	0.06	81.58	95.67	c
EfficientNet-B3 (AA + AdvProp)*	12.23	0.06	81.81	95.69	c
EfficientNet-B4*	19.34	0.12	82.57	96.09	c
EfficientNet-B4 (AA)*	19.34	0.12	82.95	96.26	c
EfficientNet-B4 (AA + AdvProp)*	19.34	0.12	83.25	96.44	c
EfficientNet-B5*	30.39	0.24	83.18	96.47	c
EfficientNet-B5 (AA)*	30.39	0.24	83.82	96.76	c
EfficientNet-B5 (AA + AdvProp)*	30.39	0.24	84.21	96.98	c
EfficientNet-B6 (AA)*	43.04	0.41	84.05	96.82	c
EfficientNet-B6 (AA + AdvProp)*	43.04	0.41	84.74	97.14	c
EfficientNet-B7 (AA)*	66.35	0.72	84.38	96.88	c
EfficientNet-B7 (AA + AdvProp)*	66.35	0.72	85.14	97.23	c
EfficientNet-B8 (AA + AdvProp)*	87.41	1.09	85.38	97.28	c
ConvNeXt-T*	28.59	4.46	82.05	95.86	c
ConvNeXt-S*	50.22	8.69	83.13	96.44	c
ConvNeXt-B*	88.59	15.36	83.85	96.74	c
ConvNeXt-B*	88.59	15.36	85.81	97.86	c
ConvNeXt-L*	197.77	34.37	84.30	96.89	c
ConvNeXt-L*	197.77	34.37	86.61	98.04	c
ConvNeXt-XL*	350.20	60.93	86.97	98.20	c
HRNet-W18*	21.30	4.33	76.75	93.44	c
HRNet-W30*	37.71	8.17	78.19	94.22	c
HRNet-W32*	41.23	8.99	78.44	94.19	c
HRNet-W40*	57.55	12.77	78.94	94.47	c
HRNet-W44*	67.06	14.96	78.88	94.37	c
HRNet-W48*	77.47	17.36	79.32	94.52	c
HRNet-W64*	128.06	29.00	79.46	94.65	c
HRNet-W18 (ssld)*	21.30	4.33	81.06	95.70	c
HRNet-W48 (ssld)*	77.47	17.36	83.63	96.79	c
WRN-50*	68.88	11.44	81.45	95.53	c
WRN-101*	126.89	22.81	78.84	94.28	c
CSPDarkNet50*	27.64	5.04	80.05	95.07	c
CSPResNet50*	21.62	3.48	79.55	94.68	c
CSPResNeXt50*	20.57	3.11	79.96	94.96	c
DenseNet121*	7.98	2.88	74.96	92.21	c
DenseNet169*	14.15	3.42	76.08	93.11	c
DenseNet201*	20.01	4.37	77.32	93.64	c
DenseNet161*	28.68	7.82	77.61	93.83	c

Table 1 – continued from previous page

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	C
VAN-T*	4.11	0.88	75.41	93.02	c
VAN-S*	13.86	2.52	81.01	95.63	c
VAN-B*	26.58	5.03	82.80	96.21	c
VAN-L*	44.77	8.99	83.86	96.73	c
MViTv2-tiny*	24.17	4.70	82.33	96.15	c
MViTv2-small*	34.87	7.00	83.63	96.51	c
MViTv2-base*	51.47	10.20	84.34	96.86	c
MViTv2-large*	217.99	42.10	85.25	97.14	c
EfficientFormer-l1*	12.19	1.30	80.46	94.99	c
EfficientFormer-l3*	31.41	3.93	82.45	96.18	c
EfficientFormer-l7*	82.23	10.16	83.40	96.60	c

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

12.2 CIFAR10

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

CHAPTER

THIRTEEN

CONFORMER

Conformer: Local Features Coupling Global Representations for Visual Recognition

13.1 Abstract

Within Convolutional Neural Network (CNN), the convolution operations are good at extracting local features but experience difficulty to capture global representations. Within visual transformer, the cascaded self-attention modules can capture long-distance feature dependencies but unfortunately deteriorate local feature details. In this paper, we propose a hybrid network structure, termed Conformer, to take advantage of convolutional operations and self-attention mechanisms for enhanced representation learning. Conformer roots in the Feature Coupling Unit (FCU), which fuses local features and global representations under different resolutions in an interactive fashion. Conformer adopts a concurrent structure so that local features and global representations are retained to the maximum extent. Experiments show that Conformer, under the comparable parameter complexity, outperforms the visual transformer (DeiT-B) by 2.3% on ImageNet. On MSCOCO, it outperforms ResNet-101 by 3.7% and 3.6% mAPs for object detection and instance segmentation, respectively, demonstrating the great potential to be a general backbone network.

13.2 Results and models

13.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
Conformer-tiny-p16*	23.52	4.90	81.31	95.60	config	model
Conformer-small-p32*	38.85	7.09	81.96	96.02	config	model
Conformer-small-p16*	37.67	10.31	83.32	96.46	config	model
Conformer-base-p16*	83.29	22.89	83.82	96.59	config	model

*Models with * are converted from the official repo. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

13.3 Citation

```
@article{peng2021conformer,
      title={Conformer: Local Features Coupling Global Representations for Visual␣
↪Recognition},
      author={Zhiliang Peng and Wei Huang and Shanzhi Gu and Lingxi Xie and Yaowei Wang␣
↪and Jianbin Jiao and Qixiang Ye},
      journal={arXiv preprint arXiv:2105.03889},
      year={2021},
}
```

CHAPTER

FOURTEEN

CONVMIXER

Patches Are All You Need?

14.1 Abstract

Although convolutional networks have been the dominant architecture for vision tasks for many years, recent experiments have shown that Transformer-based models, most notably the Vision Transformer (ViT), may exceed their performance in some settings. However, due to the quadratic runtime of the self-attention layers in Transformers, ViTs require the use of patch embeddings, which group together small regions of the image into single input features, in order to be applied to larger image sizes. This raises a question: Is the performance of ViTs due to the inherently-more-powerful Transformer architecture, or is it at least partly due to using patches as the input representation? In this paper, we present some evidence for the latter: specifically, we propose the ConvMixer, an extremely simple model that is similar in spirit to the ViT and the even-more-basic MLP-Mixer in that it operates directly on patches as input, separates the mixing of spatial and channel dimensions, and maintains equal size and resolution throughout the network. In contrast, however, the ConvMixer uses only standard convolutions to achieve the mixing steps. Despite its simplicity, we show that the ConvMixer outperforms the ViT, MLP-Mixer, and some of their variants for similar parameter counts and data set sizes, in addition to outperforming classical vision models such as the ResNet.

14.2 Results and models

14.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
ConvMixer-768/32*	21.11	19.62	80.16	95.08	config	model
ConvMixer-1024/20*	24.38	5.55	76.94	93.36	config	model
ConvMixer-1536/20*	51.63	48.71	81.37	95.61	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

14.3 Citation

```
@misc{trockman2022patches,
      title={Patches Are All You Need?},
      author={Asher Trockman and J. Zico Kolter},
      year={2022},
      eprint={2201.09792},
      archivePrefix={arXiv},
      primaryClass={cs.CV}
}
```

CHAPTER

FIFTEEN

CONVNEXT

A ConvNet for the 2020s

15.1 Abstract

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

15.2 Results and models

15.2.1 ImageNet-1k

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

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

15.2.2 Pre-trained Models

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

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

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

15.3 Citation

```
@Article{liu2022convnet,
  author  = {Zhuang Liu and Hanzi Mao and Chao-Yuan Wu and Christoph Feichtenhofer and␣
↪Trevor Darrell and Saining Xie},
  title   = {A ConvNet for the 2020s},
  journal = {arXiv preprint arXiv:2201.03545},
  year    = {2022},
}
```

CHAPTER

SIXTEEN

CSPNET

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

16.1 Abstract

Neural networks have enabled state-of-the-art approaches to achieve incredible results on computer vision tasks such as object detection. However, such success greatly relies on costly computation resources, which hinders people with cheap devices from appreciating the advanced technology. In this paper, we propose Cross Stage Partial Network (CSPNet) to mitigate the problem that previous works require heavy inference computations from the network architecture perspective. We attribute the problem to the duplicate gradient information within network optimization. The proposed networks respect the variability of the gradients by integrating feature maps from the beginning and the end of a network stage, which, in our experiments, reduces computations by 20% with equivalent or even superior accuracy on the ImageNet dataset, and significantly outperforms state-of-the-art approaches in terms of AP50 on the MS COCO object detection dataset. The CSPNet is easy to implement and general enough to cope with architectures based on ResNet, ResNeXt, and DenseNet. Source code is at this https URL.

16.2 Results and models

16.2.1 ImageNet-1k

Model	Pretrain	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
CSPDarkNet50*	From scratch	27.64	5.04	80.05	95.07	config	model
CSPResNet50*	From scratch	21.62	3.48	79.55	94.68	config	model
CSPResNeXt50*	From scratch	20.57	3.11	79.96	94.96	config	model

*Models with * are converted from the timm repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

16.3 Citation

```
@inproceedings{wang2020cspnet,
  title={CSPNet: A new backbone that can enhance learning capability of CNN},
  author={Wang, Chien-Yao and Liao, Hong-Yuan Mark and Wu, Yueh-Hua and Chen, Ping-Yang␣
↪and Hsieh, Jun-Wei and Yeh, I-Hau},
  booktitle={Proceedings of the IEEE/CVF conference on computer vision and pattern␣
↪recognition workshops},
  pages={390--391},
  year={2020}
}
```

CHAPTER

SEVENTEEN

CSRA

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

17.1 Abstract

Multi-label image recognition is a challenging computer vision task of practical use. Progresses in this area, however, are often characterized by complicated methods, heavy computations, and lack of intuitive explanations. To effectively capture different spatial regions occupied by objects from different categories, we propose an embarrassingly simple module, named class-specific residual attention (CSRA). CSRA generates class-specific features for every category by proposing a simple spatial attention score, and then combines it with the class-agnostic average pooling feature. CSRA achieves state-of-the-art results on multilabel recognition, and at the same time is much simpler than them. Furthermore, with only 4 lines of code, CSRA also leads to consistent improvement across many diverse pretrained models and datasets without any extra training. CSRA is both easy to implement and light in computations, which also enjoys intuitive explanations and visualizations.

17.2 Results and models

17.2.1 VOC2007

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

17.3 Citation

```
@misc{https://doi.org/10.48550/arxiv.2108.02456,
  doi = {10.48550/ARXIV.2108.02456},
  url = {https://arxiv.org/abs/2108.02456},
  author = {Zhu, Ke and Wu, Jianxin},
  keywords = {Computer Vision and Pattern Recognition (cs.CV), FOS: Computer and␣
↪information sciences, FOS: Computer and information sciences},
  title = {Residual Attention: A Simple but Effective Method for Multi-Label Recognition}
↪,
  publisher = {arXiv},
```

(continues on next page)

(continued from previous page)

```
  year = {2021},
  copyright = {arXiv.org perpetual, non-exclusive license}
}
```

CHAPTER

EIGHTEEN

DEIT

Training data-efficient image transformers & distillation through attention

18.1 Abstract

Recently, neural networks purely based on attention were shown to address image understanding tasks such as image classification. However, these visual transformers are pre-trained with hundreds of millions of images using an expensive infrastructure, thereby limiting their adoption. In this work, we produce a competitive convolution-free transformer by training on Imagenet only. We train them on a single computer in less than 3 days. Our reference vision transformer (86M parameters) achieves top-1 accuracy of 83.1% (single-crop evaluation) on ImageNet with no external data. More importantly, we introduce a teacher-student strategy specific to transformers. It relies on a distillation token ensuring that the student learns from the teacher through attention. We show the interest of this token-based distillation, especially when using a convnet as a teacher. This leads us to report results competitive with convnets for both Imagenet (where we obtain up to 85.2% accuracy) and when transferring to other tasks. We share our code and models.

18.2 Results and models

18.2.1 ImageNet-1k

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

Model	Pretrain	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
DeiT-tiny	From scratch	5.72	1.08	74.50	92.24	config	model | log
DeiT-tiny distilled*	From scratch	5.72	1.08	74.51	91.90	config	model
DeiT-small	From scratch	22.05	4.24	80.69	95.06	config	model | log
DeiT-small distilled*	From scratch	22.05	4.24	81.17	95.40	config	model
DeiT-base	From scratch	86.57	16.86	81.76	95.81	config	model | log
DeiT-base*	From scratch	86.57	16.86	81.79	95.59	config	model
DeiT-base distilled*	From scratch	86.57	16.86	83.33	96.49	config	model
DeiT-base 384px*	ImageNet-1k	86.86	49.37	83.04	96.31	config	model
DeiT-base distilled 384px*	ImageNet-1k	86.86	49.37	85.55	97.35	config	model

*Models with * are converted from the official repo. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

Warning: MMClassification doesn't support training the distilled version DeiT. And we provide distilled version checkpoints for inference only.

18.3 Citation

```
@InProceedings{pmlr-v139-touvron21a,
  title =     {Training data-efficient image transformers &amp; distillation through␣
↪attention},
  author =    {Touvron, Hugo and Cord, Matthieu and Douze, Matthijs and Massa, Francisco␣
↪and Sablayrolles, Alexandre and Jegou, Herve},
  booktitle = {International Conference on Machine Learning},
  pages =     {10347--10357},
  year =      {2021},
  volume =    {139},
  month =     {July}
}
```

CHAPTER

NINETEEN

DENSENET

Densely Connected Convolutional Networks

19.1 Abstract

Recent work has shown that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output. In this paper, we embrace this observation and introduce the Dense Convolutional Network (DenseNet), which connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections - one between each layer and its subsequent layer - our network has L(L+1)/2 direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters. We evaluate our proposed architecture on four highly competitive object recognition benchmark tasks (CIFAR-10, CIFAR-100, SVHN, and ImageNet). DenseNets obtain significant improvements over the state-of-the-art on most of them, whilst requiring less computation to achieve high performance.

19.2 Results and models

19.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
DenseNet121*	7.98	2.88	74.96	92.21	config	model
DenseNet169*	14.15	3.42	76.08	93.11	config	model
DenseNet201*	20.01	4.37	77.32	93.64	config	model
DenseNet161*	28.68	7.82	77.61	93.83	config	model

*Models with * are converted from pytorch, guided by original repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

19.3 Citation

```
@misc{https://doi.org/10.48550/arxiv.1608.06993,
      doi = {10.48550/ARXIV.1608.06993},
      url = {https://arxiv.org/abs/1608.06993},
      author = {Huang, Gao and Liu, Zhuang and van der Maaten, Laurens and Weinberger,␣
↪Kilian Q.},
      keywords = {Computer Vision and Pattern Recognition (cs.CV), Machine Learning (cs.
↪LG), FOS: Computer and information sciences, FOS: Computer and information sciences},
      title = {Densely Connected Convolutional Networks},
      publisher = {arXiv},
      year = {2016},
      copyright = {arXiv.org perpetual, non-exclusive license}
}
```

CHAPTER

TWENTY

EFFICIENTFORMER

EfficientFormer: Vision Transformers at MobileNet Speed

20.1 Abstract

Vision Transformers (ViT) have shown rapid progress in computer vision tasks, achieving promising results on various benchmarks. However, due to the massive number of parameters and model design, e.g., attention mechanism, ViT-based models are generally times slower than lightweight convolutional networks. Therefore, the deployment of ViT for real-time applications is particularly challenging, especially on resource-constrained hardware such as mobile devices. Recent efforts try to reduce the computation complexity of ViT through network architecture search or hybrid design with MobileNet block, yet the inference speed is still unsatisfactory. This leads to an important question: can transformers run as fast as MobileNet while obtaining high performance? To answer this, we first revisit the network architecture and operators used in ViT-based models and identify inefficient designs. Then we introduce a dimension-consistent pure transformer (without MobileNet blocks) as a design paradigm. Finally, we perform latency-driven slimming to get a series of final models dubbed EfficientFormer. Extensive experiments show the superiority of EfficientFormer in performance and speed on mobile devices. Our fastest model, EfficientFormer-L1, achieves 79.2% top-1 accuracy on ImageNet-1K with only 1.6 ms inference latency on iPhone 12 (compiled with CoreML), which runs as fast as MobileNetV2×1.4 (1.6 ms, 74.7% top-1), and our largest model, EfficientFormer-L7, obtains 83.3% accuracy with only 7.0 ms latency. Our work proves that properly designed transformers can reach extremely low latency on mobile devices while maintaining high performance.

20.2 Results and models

20.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
EfficientFormer-l1*	12.19	1.30	80.46	94.99	config	model
EfficientFormer-l3*	31.41	3.93	82.45	96.18	config	model
EfficientFormer-l7*	82.23	10.16	83.40	96.60	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

20.3 Citation

```
@misc{https://doi.org/10.48550/arxiv.2206.01191,
  doi = {10.48550/ARXIV.2206.01191},

  url = {https://arxiv.org/abs/2206.01191},

  author = {Li, Yanyu and Yuan, Geng and Wen, Yang and Hu, Eric and Evangelidis,␣
↪Georgios and Tulyakov, Sergey and Wang, Yanzhi and Ren, Jian},

  keywords = {Computer Vision and Pattern Recognition (cs.CV), FOS: Computer and␣
↪information sciences, FOS: Computer and information sciences},

  title = {EfficientFormer: Vision Transformers at MobileNet Speed},

  publisher = {arXiv},

  year = {2022},

  copyright = {Creative Commons Attribution 4.0 International}
}
```

CHAPTER

TWENTYONE

EFFICIENTNET

Rethinking Model Scaling for Convolutional Neural Networks

21.1 Abstract

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

21.2 Results and models

21.2.1 ImageNet-1k

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

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

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

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

21.3 Citation

```
@inproceedings{tan2019efficientnet,
  title={Efficientnet: Rethinking model scaling for convolutional neural networks},
  author={Tan, Mingxing and Le, Quoc},
  booktitle={International Conference on Machine Learning},
  pages={6105--6114},
  year={2019},
  organization={PMLR}
}
```

CHAPTER TWENTYTWO

HORNET

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

22.1 Abstract

Recent progress in vision Transformers exhibits great success in various tasks driven by the new spatial modeling mechanism based on dot-product self-attention. In this paper, we show that the key ingredients behind the vision Transformers, namely input-adaptive, long-range and high-order spatial interactions, can also be efficiently implemented with a convolution-based framework. We present the Recursive Gated Convolution (g nConv) that performs high-order spatial interactions with gated convolutions and recursive designs. The new operation is highly flexible and customizable, which is compatible with various variants of convolution and extends the two-order interactions in self-attention to arbitrary orders without introducing significant extra computation. g nConv can serve as a plug-and-play module to improve various vision Transformers and convolution-based models. Based on the operation, we construct a new family of generic vision backbones named HorNet. Extensive experiments on ImageNet classification, COCO object detection and ADE20K semantic segmentation show HorNet outperform Swin Transformers and ConvNeXt by a significant margin with similar overall architecture and training configurations. HorNet also shows favorable scalability to more training data and a larger model size. Apart from the effectiveness in visual encoders, we also show g nConv can be applied to task-specific decoders and consistently improve dense prediction performance with less computation. Our results demonstrate that g nConv can be a new basic module for visual modeling that effectively combines the merits of both vision Transformers and CNNs. Code is available at https://github.com/raoyongming/HorNet.

22.2 Results and models

22.2.1 ImageNet-1k

Model	Pretrain	resolution	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
HorNet-T*	From scratch	224x224	22.41	3.98	82.84	96.24	config	model
HorNet-T-GF*	From scratch	224x224	22.99	3.9	82.98	96.38	config	model
HorNet-S*	From scratch	224x224	49.53	8.83	83.79	96.75	config	model
HorNet-S-GF*	From scratch	224x224	50.4	8.71	83.98	96.77	config	model
HorNet-B*	From scratch	224x224	87.26	15.59	84.24	96.94	config	model
HorNet-B-GF*	From scratch	224x224	88.42	15.42	84.32	96.95	config	model

*Models with * are converted from the official repo. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.

22.2.2 Pre-trained Models

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

Model	Pretrain	resolution	Params(M)	Flops(G)	Download
HorNet-L*	ImageNet-21k	224x224	194.54	34.83	model
HorNet-L-GF*	ImageNet-21k	224x224	196.29	34.58	model
HorNet-L-GF384*	ImageNet-21k	384x384	201.23	101.63	model

*Models with * are converted from the official repo.

22.3 Citation

```
@article{rao2022hornet,
  title={HorNet: Efficient High-Order Spatial Interactions with Recursive Gated␣
↪Convolutions},
  author={Rao, Yongming and Zhao, Wenliang and Tang, Yansong and Zhou, Jie and Lim, Ser-
↪Lam and Lu, Jiwen},
  journal={arXiv preprint arXiv:2207.14284},
  year={2022}
}
```

CHAPTER

TWENTYTHREE

HRNET

Deep High-Resolution Representation Learning for Visual Recognition

23.1 Abstract

High-resolution representations are essential for position-sensitive vision problems, such as human pose estimation, semantic segmentation, and object detection. Existing state-of-the-art frameworks first encode the input image as a low-resolution representation through a subnetwork that is formed by connecting high-to-low resolution convolutions *in series* (e.g., ResNet, VGGNet), and then recover the high-resolution representation from the encoded low-resolution representation. Instead, our proposed network, named as High-Resolution Network (HRNet), maintains high-resolution representations through the whole process. There are two key characteristics: (i) Connect the high-to-low resolution convolution streams *in parallel*; (ii) Repeatedly exchange the information across resolutions. The benefit is that the resulting representation is semantically richer and spatially more precise. We show the superiority of the proposed HRNet in a wide range of applications, including human pose estimation, semantic segmentation, and object detection, suggesting that the HRNet is a stronger backbone for computer vision problems.

23.2 Results and models

23.3 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
HRNet-W18*	21.30	4.33	76.75	93.44	config	model
HRNet-W30*	37.71	8.17	78.19	94.22	config	model
HRNet-W32*	41.23	8.99	78.44	94.19	config	model
HRNet-W40*	57.55	12.77	78.94	94.47	config	model
HRNet-W44*	67.06	14.96	78.88	94.37	config	model
HRNet-W48*	77.47	17.36	79.32	94.52	config	model
HRNet-W64*	128.06	29.00	79.46	94.65	config	model
HRNet-W18 (ssld)*	21.30	4.33	81.06	95.70	config	model
HRNet-W48 (ssld)*	77.47	17.36	83.63	96.79	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

23.4 Citation

```
@article{WangSCJDZLMTWLX19,
  title={Deep High-Resolution Representation Learning for Visual Recognition},
  author={Jingdong Wang and Ke Sun and Tianheng Cheng and
          Borui Jiang and Chaorui Deng and Yang Zhao and Dong Liu and Yadong Mu and
          Mingkui Tan and Xinggang Wang and Wenyu Liu and Bin Xiao},
  journal   = {TPAMI}
  year={2019}
}
```

CHAPTER

TWENTYFOUR

MLP-MIXER

MLP-Mixer: An all-MLP Architecture for Vision

24.1 Abstract

Convolutional Neural Networks (CNNs) are the go-to model for computer vision. Recently, attention-based networks, such as the Vision Transformer, have also become popular. In this paper we show that while convolutions and attention are both sufficient for good performance, neither of them are necessary. We present MLP-Mixer, an architecture based exclusively on multi-layer perceptrons (MLPs). MLP-Mixer contains two types of layers: one with MLPs applied independently to image patches (i.e. "mixing" the per-location features), and one with MLPs applied across patches (i.e. "mixing" spatial information). When trained on large datasets, or with modern regularization schemes, MLP-Mixer attains competitive scores on image classification benchmarks, with pre-training and inference cost comparable to state-of-the-art models. We hope that these results spark further research beyond the realms of well established CNNs and Transformers.

24.2 Results and models

24.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
Mixer-B/16*	59.88	12.61	76.68	92.25	config	model
Mixer-L/16*	208.2	44.57	72.34	88.02	config	model

*Models with * are converted from timm. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

24.3 Citation

```
@misc{tolstikhin2021mlpmixer,
      title={MLP-Mixer: An all-MLP Architecture for Vision},
      author={Ilya Tolstikhin and Neil Houlsby and Alexander Kolesnikov and Lucas Beyer␣
↪and Xiaohua Zhai and Thomas Unterthiner and Jessica Yung and Andreas Steiner and␣
↪Daniel Keysers and Jakob Uszkoreit and Mario Lucic and Alexey Dosovitskiy},
      year={2021},
      eprint={2105.01601},
```

(continues on next page)

(continued from previous page)

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

CHAPTER TWENTYFIVE

MOBILENET V2

MobileNetV2: Inverted Residuals and Linear Bottlenecks

25.1 Abstract

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

The MobileNetV2 architecture is based on an inverted residual structure where the input and output of the residual block are thin bottleneck layers opposite to traditional residual models which use expanded representations in the input an MobileNetV2 uses lightweight depthwise convolutions to filter features in the intermediate expansion layer. Additionally, we find that it is important to remove non-linearities in the narrow layers in order to maintain representational power. We demonstrate that this improves performance and provide an intuition that led to this design. Finally, our approach allows decoupling of the input/output domains from the expressiveness of the transformation, which provides a convenient framework for further analysis. We measure our performance on Imagenet classification, COCO object detection, VOC image segmentation. We evaluate the trade-offs between accuracy, and number of operations measured by multiply-adds (MAdd), as well as the number of parameters

25.2 Results and models

25.2.1 ImageNet-1k

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

25.3 Citation

```
@INPROCEEDINGS{8578572,
  author={M. {Sandler} and A. {Howard} and M. {Zhu} and A. {Zhmoginov} and L. {Chen}},
  booktitle={2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition},
  title={MobileNetV2: Inverted Residuals and Linear Bottlenecks},
  year={2018},
  volume={},
  number={},
  pages={4510-4520},
  doi={10.1109/CVPR.2018.00474}}
}
```

CHAPTER

TWENTYSIX

MOBILENET V3

Searching for MobileNetV3

26.1 Abstract

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

26.2 Results and models

26.2.1 ImageNet-1k

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

*Models with * are converted from torchvision. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

26.3 Citation

```
@inproceedings{Howard_2019_ICCV,
    author = {Howard, Andrew and Sandler, Mark and Chu, Grace and Chen, Liang-Chieh and␣
↪Chen, Bo and Tan, Mingxing and Wang, Weijun and Zhu, Yukun and Pang, Ruoming and␣
↪Vasudevan, Vijay and Le, Quoc V. and Adam, Hartwig},
    title = {Searching for MobileNetV3},
    booktitle = {Proceedings of the IEEE/CVF International Conference on Computer Vision␣
↪(ICCV)},
    month = {October},
    year = {2019}
}
```

CHAPTER

TWENTYSEVEN

MVIT V2

MViTv2: Improved Multiscale Vision Transformers for Classification and Detection

27.1 Abstract

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

27.2 Results and models

27.2.1 ImageNet-1k

Model	Pretrain	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
MViTv2-tiny*	From scratch	24.17	4.70	82.33	96.15	config	model
MViTv2-small*	From scratch	34.87	7.00	83.63	96.51	config	model
MViTv2-base*	From scratch	51.47	10.20	84.34	96.86	config	model
MViTv2-large*	From scratch	217.99	42.10	85.25	97.14	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

27.3 Citation

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

(continues on next page)

(continued from previous page)

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

CHAPTER

TWENTYEIGHT

POOLFORMER

MetaFormer is Actually What You Need for Vision

28.1 Abstract

Transformers have shown great potential in computer vision tasks. A common belief is their attention-based token mixer module contributes most to their competence. However, recent works show the attention-based module in transformers can be replaced by spatial MLPs and the resulted models still perform quite well. Based on this observation, we hypothesize that the general architecture of the transformers, instead of the specific token mixer module, is more essential to the model's performance. To verify this, we deliberately replace the attention module in transformers with an embarrassingly simple spatial pooling operator to conduct only basic token mixing. Surprisingly, we observe that the derived model, termed as PoolFormer, achieves competitive performance on multiple computer vision tasks. For example, on ImageNet-1K, PoolFormer achieves 82.1% top-1 accuracy, surpassing well-tuned vision transformer/MLP-like baselines DeiT-B/ResMLP-B24 by 0.3%/1.1% accuracy with 35%/52% fewer parameters and 49%/61% fewer MACs. The effectiveness of PoolFormer verifies our hypothesis and urges us to initiate the concept of "MetaFormer", a general architecture abstracted from transformers without specifying the token mixer. Based on the extensive experiments, we argue that MetaFormer is the key player in achieving superior results for recent transformer and MLP-like models on vision tasks. This work calls for more future research dedicated to improving MetaFormer instead of focusing on the token mixer modules. Additionally, our proposed PoolFormer could serve as a starting baseline for future MetaFormer architecture design.

28.2 Results and models

28.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
PoolFormer-S12*	11.92	1.87	77.24	93.51	config	model
PoolFormer-S24*	21.39	3.51	80.33	95.05	config	model
PoolFormer-S36*	30.86	5.15	81.43	95.45	config	model
PoolFormer-M36*	56.17	8.96	82.14	95.71	config	model
PoolFormer-M48*	73.47	11.80	82.51	95.95	config	model

*Models with * are converted from the official repo. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

28.3 Citation

```
@article{yu2021metaformer,
  title={MetaFormer is Actually What You Need for Vision},
  author={Yu, Weihao and Luo, Mi and Zhou, Pan and Si, Chenyang and Zhou, Yichen and␣
↪Wang, Xinchao and Feng, Jiashi and Yan, Shuicheng},
  journal={arXiv preprint arXiv:2111.11418},
  year={2021}
}
```

CHAPTER TWENTYNINE

REGNET

Designing Network Design Spaces

29.1 Abstract

In this work, we present a new network design paradigm. Our goal is to help advance the understanding of network design and discover design principles that generalize across settings. Instead of focusing on designing individual network instances, we design network design spaces that parametrize populations of networks. The overall process is analogous to classic manual design of networks, but elevated to the design space level. Using our methodology we explore the structure aspect of network design and arrive at a low-dimensional design space consisting of simple, regular networks that we call RegNet. The core insight of the RegNet parametrization is surprisingly simple: widths and depths of good networks can be explained by a quantized linear function. We analyze the RegNet design space and arrive at interesting findings that do not match the current practice of network design. The RegNet design space provides simple and fast networks that work well across a wide range of flop regimes. Under comparable training settings and flops, the RegNet models outperform the popular EfficientNet models while being up to 5x faster on GPUs.

29.2 Results and models

29.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
RegNetX-400MF	5.16	0.41	72.56	90.78	config	model | log
RegNetX-800MF	7.26	0.81	74.76	92.32	config	model | log
RegNetX-1.6GF	9.19	1.63	76.84	93.31	config	model | log
RegNetX-3.2GF	15.3	3.21	78.09	94.08	config	model | log
RegNetX-4.0GF	22.12	4.0	78.60	94.17	config	model | log
RegNetX-6.4GF	26.21	6.51	79.38	94.65	config	model | log
RegNetX-8.0GF	39.57	8.03	79.12	94.51	config	model | log
RegNetX-12GF	46.11	12.15	79.67	95.03	config	model | log
RegNetX-400MF*	5.16	0.41	72.55	90.91	config	model
RegNetX-800MF*	7.26	0.81	75.21	92.37	config	model
RegNetX-1.6GF*	9.19	1.63	77.04	93.51	config	model
RegNetX-3.2GF*	15.3	3.21	78.26	94.20	config	model
RegNetX-4.0GF*	22.12	4.0	78.72	94.22	config	model
RegNetX-6.4GF*	26.21	6.51	79.22	94.61	config	model
RegNetX-8.0GF*	39.57	8.03	79.31	94.57	config	model
RegNetX-12GF*	46.11	12.15	79.91	94.78	config	model

*Models with * are converted from pycls. The config files of these models are only for validation.*

29.3 Citation

```
@article{radosavovic2020designing,
    title={Designing Network Design Spaces},
    author={Ilija Radosavovic and Raj Prateek Kosaraju and Ross Girshick and Kaiming He␣
↪and Piotr Dollár},
    year={2020},
    eprint={2003.13678},
    archivePrefix={arXiv},
    primaryClass={cs.CV}
}
```

CHAPTER

THIRTY

REPMLP

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

30.1 Abstract

We propose RepMLP, a multi-layer-perceptron-style neural network building block for image recognition, which is composed of a series of fully-connected (FC) layers. Compared to convolutional layers, FC layers are more efficient, better at modeling the long-range dependencies and positional patterns, but worse at capturing the local structures, hence usually less favored for image recognition. We propose a structural re-parameterization technique that adds local prior into an FC to make it powerful for image recognition. Specifically, we construct convolutional layers inside a RepMLP during training and merge them into the FC for inference. On CIFAR, a simple pure-MLP model shows performance very close to CNN. By inserting RepMLP in traditional CNN, we improve ResNets by 1.8% accuracy on ImageNet, 2.9% for face recognition, and 2.3% mIoU on Cityscapes with lower FLOPs. Our intriguing findings highlight that combining the global representational capacity and positional perception of FC with the local prior of convolution can improve the performance of neural network with faster speed on both the tasks with translation invariance (e.g., semantic segmentation) and those with aligned images and positional patterns (e.g., face recognition).

30.2 Results and models

30.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
RepMLP-B224*	68.24	6.71	80.41	95.12	train_cfg | deploy_cfg	model
RepMLP-B256*	96.45	9.69	81.11	95.5	train_cfg | deploy_cfg	model

*Models with * are converted from the official repo.. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

30.3 How to use

The checkpoints provided are all `training-time` models. Use the reparameterize tool to switch them to more efficient `inference-time` architecture, which not only has fewer parameters but also less calculations.

30.3.1 Use tool

Use provided tool to reparameterize the given model and save the checkpoint:

```
python tools/convert_models/reparameterize_model.py ${CFG_PATH} ${SRC_CKPT_PATH} $
↪{TARGET_CKPT_PATH}
```

`${CFG_PATH}` is the config file, `${SRC_CKPT_PATH}` is the source chenpoint file, `${TARGET_CKPT_PATH}` is the target deploy weight file path.

To use reparameterized weights, the config file must switch to the deploy config files.

```
python tools/test.py ${Deploy_CFG} ${Deploy_Checkpoint} --metrics accuracy
```

30.3.2 In the code

Use `backbone.switch_to_deploy()` or `classificer.backbone.switch_to_deploy()` to switch to the deploy mode. For example:

```
from mmcls.models import build_backbone

backbone_cfg=dict(type='RepMLPNet', arch='B', img_size=224, reparam_conv_kernels=(1, 3),␣
↪deploy=False)
backbone = build_backbone(backbone_cfg)
backbone.switch_to_deploy()
```

or

```
from mmcls.models import build_classifier

cfg = dict(
    type='ImageClassifier',
    backbone=dict(
        type='RepMLPNet',
        arch='B',
        img_size=224,
        reparam_conv_kernels=(1, 3),
        deploy=False),
    neck=dict(type='GlobalAveragePooling'),
    head=dict(
        type='LinearClsHead',
        num_classes=1000,
        in_channels=768,
        loss=dict(type='CrossEntropyLoss', loss_weight=1.0),
        topk=(1, 5),
    ))
```

(continues on next page)

(continued from previous page)

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

30.4 Citation

```
@article{ding2021repmlp,
  title={Repmlp: Re-parameterizing convolutions into fully-connected layers for image␣
↪recognition},
  author={Ding, Xiaohan and Xia, Chunlong and Zhang, Xiangyu and Chu, Xiaojie and Han,␣
↪Jungong and Ding, Guiguang},
  journal={arXiv preprint arXiv:2105.01883},
  year={2021}
}
```

CHAPTER

THIRTYONE

REPVGG

Repvgg: Making vgg-style convnets great again

31.1 Abstract

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

31.2 Results and models

31.2.1 ImageNet-1k

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

*Models with * are converted from the official repo. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

31.3 How to use

The checkpoints provided are all `training-time` models. Use the reparameterize tool to switch them to more efficient `inference-time` architecture, which not only has fewer parameters but also less calculations.

31.3.1 Use tool

Use provided tool to reparameterize the given model and save the checkpoint:

```
python tools/convert_models/reparameterize_model.py ${CFG_PATH} ${SRC_CKPT_PATH} $
↪{TARGET_CKPT_PATH}
```

`${CFG_PATH}` is the config file, `${SRC_CKPT_PATH}` is the source chenpoint file, `${TARGET_CKPT_PATH}` is the target deploy weight file path.

To use reparameterized weights, the config file must switch to the deploy config files.

```
python tools/test.py ${Deploy_CFG} ${Deploy_Checkpoint} --metrics accuracy
```

31.3.2 In the code

Use `backbone.switch_to_deploy()` or `classificer.backbone.switch_to_deploy()` to switch to the deploy mode. For example:

```
from mmcls.models import build_backbone

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

or

```
from mmcls.models import build_classifier

cfg = dict(
    type='ImageClassifier',
    backbone=dict(
        type='RepVGG',
        arch='A0'),
    neck=dict(type='GlobalAveragePooling'),
    head=dict(
        type='LinearClsHead',
        num_classes=1000,
        in_channels=1280,
        loss=dict(type='CrossEntropyLoss', loss_weight=1.0),
        topk=(1, 5),
    ))

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

31.4 Citation

```
@inproceedings{ding2021repvgg,
  title={Repvgg: Making vgg-style convnets great again},
  author={Ding, Xiaohan and Zhang, Xiangyu and Ma, Ningning and Han, Jungong and Ding,␣
↪Guiguang and Sun, Jian},
  booktitle={Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern␣
↪Recognition},
  pages={13733--13742},
  year={2021}
}
```

CHAPTER THIRTYTWO

RES2NET

Res2Net: A New Multi-scale Backbone Architecture

32.1 Abstract

Representing features at multiple scales is of great importance for numerous vision tasks. Recent advances in backbone convolutional neural networks (CNNs) continually demonstrate stronger multi-scale representation ability, leading to consistent performance gains on a wide range of applications. However, most existing methods represent the multi-scale features in a layer-wise manner. In this paper, we propose a novel building block for CNNs, namely Res2Net, by constructing hierarchical residual-like connections within one single residual block. The Res2Net represents multi-scale features at a granular level and increases the range of receptive fields for each network layer. The proposed Res2Net block can be plugged into the state-of-the-art backbone CNN models, e.g., ResNet, ResNeXt, and DLA. We evaluate the Res2Net block on all these models and demonstrate consistent performance gains over baseline models on widely-used datasets, e.g., CIFAR-100 and ImageNet. Further ablation studies and experimental results on representative computer vision tasks, i.e., object detection, class activation mapping, and salient object detection, further verify the superiority of the Res2Net over the state-of-the-art baseline methods.

32.2 Results and models

32.2.1 ImageNet-1k

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

*Models with * are converted from the official repo. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

32.3 Citation

```
@article{gao2019res2net,
  title={Res2Net: A New Multi-scale Backbone Architecture},
  author={Gao, Shang-Hua and Cheng, Ming-Ming and Zhao, Kai and Zhang, Xin-Yu and Yang,␣
↪Ming-Hsuan and Torr, Philip},
  journal={IEEE TPAMI},
  year={2021},
  doi={10.1109/TPAMI.2019.2938758},
}
```

CHAPTER

THIRTYTHREE

RESNET

Deep Residual Learning for Image Recognition

33.1 Abstract

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

The depth of representations is of central importance for many visual recognition tasks. Solely due to our extremely deep representations, we obtain a 28% relative improvement on the COCO object detection dataset. Deep residual nets are foundations of our submissions to ILSVRC & COCO 2015 competitions, where we also won the 1st places on the tasks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation.

33.2 Results and models

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

Model	resolution	Params(M)	Flops(G)	Download
ResNet-50-mill	224x224	86.74	15.14	model

The "mill" means using the mutil-label pretrain weight from ImageNet-21K Pretraining for the Masses.

33.2.1 Cifar10

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
ResNet-18	11.17	0.56	94.82	99.87	config	model | log
ResNet-34	21.28	1.16	95.34	99.87	config	model | log
ResNet-50	23.52	1.31	95.55	99.91	config	model | log
ResNet-101	42.51	2.52	95.58	99.87	config	model | log
ResNet-152	58.16	3.74	95.76	99.89	config	model | log

33.2.2 Cifar100

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

33.2.3 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
ResNet-18	11.69	1.82	69.90	89.43	config	model | log
ResNet-34	21.8	3.68	73.62	91.59	config	model | log
ResNet-50	25.56	4.12	76.55	93.06	config	model | log
ResNet-101	44.55	7.85	77.97	94.06	config	model | log
ResNet-152	60.19	11.58	78.48	94.13	config	model | log
ResNetV1C-50	25.58	4.36	77.01	93.58	config	model | log
ResNetV1C-101	44.57	8.09	78.30	94.27	config	model | log
ResNetV1C-152	60.21	11.82	78.76	94.41	config	model | log
ResNetV1D-50	25.58	4.36	77.54	93.57	config	model | log
ResNetV1D-101	44.57	8.09	78.93	94.48	config	model | log
ResNetV1D-152	60.21	11.82	79.41	94.70	config	model | log
ResNet-50 (fp16)	25.56	4.12	76.30	93.07	config	model | log
Wide-ResNet-50*	68.88	11.44	78.48	94.08	config	model
Wide-ResNet-101*	126.89	22.81	78.84	94.28	config	model
ResNet-50 (rsb-a1)	25.56	4.12	80.12	94.78	config	model | log
ResNet-50 (rsb-a2)	25.56	4.12	79.55	94.37	config	model | log
ResNet-50 (rsb-a3)	25.56	4.12	78.30	93.80	config	model | log

The "rsb" means using the training settings from ResNet strikes back: An improved training procedure in timm.

*Models with * are converted from the official repo. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

33.2.4 CUB-200-2011

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

33.2.5 Stanford-Cars

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

33.3 Citation

```
@inproceedings{he2016deep,
  title={Deep residual learning for image recognition},
  author={He, Kaiming and Zhang, Xiangyu and Ren, Shaoqing and Sun, Jian},
  booktitle={Proceedings of the IEEE conference on computer vision and pattern␣
↪recognition},
  pages={770--778},
  year={2016}
}
```

CHAPTER

THIRTYFOUR

RESNEXT

Aggregated Residual Transformations for Deep Neural Networks

34.1 Abstract

We present a simple, highly modularized network architecture for image classification. Our network is constructed by repeating a building block that aggregates a set of transformations with the same topology. Our simple design results in a homogeneous, multi-branch architecture that has only a few hyper-parameters to set. This strategy exposes a new dimension, which we call "cardinality" (the size of the set of transformations), as an essential factor in addition to the dimensions of depth and width. On the ImageNet-1K dataset, we empirically show that even under the restricted condition of maintaining complexity, increasing cardinality is able to improve classification accuracy. Moreover, increasing cardinality is more effective than going deeper or wider when we increase the capacity. Our models, named ResNeXt, are the foundations of our entry to the ILSVRC 2016 classification task in which we secured 2nd place. We further investigate ResNeXt on an ImageNet-5K set and the COCO detection set, also showing better results than its ResNet counterpart. The code and models are publicly available online.

34.2 Results and models

34.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
ResNeXt-32x4d-50	25.03	4.27	77.90	93.66	config	model | log
ResNeXt-32x4d-101	44.18	8.03	78.61	94.17	config	model | log
ResNeXt-32x8d-101	88.79	16.5	79.27	94.58	config	model | log
ResNeXt-32x4d-152	59.95	11.8	78.88	94.33	config	model | log

34.3 Citation

```
@inproceedings{xie2017aggregated,
  title={Aggregated residual transformations for deep neural networks},
  author={Xie, Saining and Girshick, Ross and Doll{\'a}r, Piotr and Tu, Zhuowen and He,␣
↪Kaiming},
  booktitle={Proceedings of the IEEE conference on computer vision and pattern␣
↪recognition},
  pages={1492--1500},
```

(continues on next page)

(continued from previous page)

```
  year={2017}
}
```

CHAPTER THIRTYFIVE

SE-RESNET

Squeeze-and-Excitation Networks

35.1 Abstract

The central building block of convolutional neural networks (CNNs) is the convolution operator, which enables networks to construct informative features by fusing both spatial and channel-wise information within local receptive fields at each layer. A broad range of prior research has investigated the spatial component of this relationship, seeking to strengthen the representational power of a CNN by enhancing the quality of spatial encodings throughout its feature hierarchy. In this work, we focus instead on the channel relationship and propose a novel architectural unit, which we term the "Squeeze-and-Excitation" (SE) block, that adaptively recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels. We show that these blocks can be stacked together to form SENet architectures that generalise extremely effectively across different datasets. We further demonstrate that SE blocks bring significant improvements in performance for existing state-of-the-art CNNs at slight additional computational cost. Squeeze-and-Excitation Networks formed the foundation of our ILSVRC 2017 classification submission which won first place and reduced the top-5 error to 2.251%, surpassing the winning entry of 2016 by a relative improvement of ~25%.

35.2 Results and models

35.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
SE-ResNet-50	28.09	4.13	77.74	93.84	config	model | log
SE-ResNet-101	49.33	7.86	78.26	94.07	config	model | log

35.3 Citation

```
@inproceedings{hu2018squeeze,
  title={Squeeze-and-excitation networks},
  author={Hu, Jie and Shen, Li and Sun, Gang},
  booktitle={Proceedings of the IEEE conference on computer vision and pattern␣
↪recognition},
  pages={7132--7141},
```

(continues on next page)

(continued from previous page)

```
  year={2018}
}
```

CHAPTER THIRTYSIX

SHUFFLENET V1

ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices

36.1 Abstract

We introduce an extremely computation-efficient CNN architecture named ShuffleNet, which is designed specially for mobile devices with very limited computing power (e.g., 10-150 MFLOPs). The new architecture utilizes two new operations, pointwise group convolution and channel shuffle, to greatly reduce computation cost while maintaining accuracy. Experiments on ImageNet classification and MS COCO object detection demonstrate the superior performance of ShuffleNet over other structures, e.g. lower top-1 error (absolute 7.8%) than recent MobileNet on ImageNet classification task, under the computation budget of 40 MFLOPs. On an ARM-based mobile device, ShuffleNet achieves ~13x actual speedup over AlexNet while maintaining comparable accuracy.

36.2 Results and models

36.2.1 ImageNet-1k

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

36.3 Citation

```
@inproceedings{zhang2018shufflenet,
  title={Shufflenet: An extremely efficient convolutional neural network for mobile␣
↪devices},
  author={Zhang, Xiangyu and Zhou, Xinyu and Lin, Mengxiao and Sun, Jian},
  booktitle={Proceedings of the IEEE conference on computer vision and pattern␣
↪recognition},
  pages={6848--6856},
  year={2018}
}
```

CHAPTER

THIRTYSEVEN

SHUFFLENET V2

Shufflenet v2: Practical guidelines for efficient cnn architecture design

37.1 Abstract

Currently, the neural network architecture design is mostly guided by the *indirect* metric of computation complexity, i.e., FLOPs. However, the *direct* metric, e.g., speed, also depends on the other factors such as memory access cost and platform characterics. Thus, this work proposes to evaluate the direct metric on the target platform, beyond only considering FLOPs. Based on a series of controlled experiments, this work derives several practical *guidelines* for efficient network design. Accordingly, a new architecture is presented, called *ShuffleNet V2*. Comprehensive ablation experiments verify that our model is the state-of-the-art in terms of speed and accuracy tradeoff.

37.2 Results and models

37.2.1 ImageNet-1k

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

37.3 Citation

```
@inproceedings{ma2018shufflenet,
  title={Shufflenet v2: Practical guidelines for efficient cnn architecture design},
  author={Ma, Ningning and Zhang, Xiangyu and Zheng, Hai-Tao and Sun, Jian},
  booktitle={Proceedings of the European conference on computer vision (ECCV)},
  pages={116--131},
  year={2018}
}
```

CHAPTER

THIRTYEIGHT

SWIN TRANSFORMER

Swin Transformer: Hierarchical Vision Transformer using Shifted Windows

38.1 Abstract

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

38.2 Results and models

38.2.1 ImageNet-21k

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

Model	resolution	Params(M)	Flops(G)	Download
Swin-B	224x224	86.74	15.14	model
Swin-B	384x384	86.88	44.49	model
Swin-L	224x224	195.00	34.04	model
Swin-L	384x384	195.20	100.04	model

38.2.2 ImageNet-1k

Model	Pretrain	resolution	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
Swin-T	From scratch	224x224	28.29	4.36	81.18	95.61	config	model | log
Swin-S	From scratch	224x224	49.61	8.52	83.02	96.29	config	model | log
Swin-B	From scratch	224x224	87.77	15.14	83.36	96.44	config	model | log
Swin-S*	From scratch	224x224	49.61	8.52	83.21	96.25	config	model
Swin-B*	From scratch	224x224	87.77	15.14	83.42	96.44	config	model
Swin-B*	From scratch	384x384	87.90	44.49	84.49	96.95	config	model
Swin-B*	ImageNet-21k	224x224	87.77	15.14	85.16	97.50	config	model
Swin-B*	ImageNet-21k	384x384	87.90	44.49	86.44	98.05	config	model
Swin-L*	ImageNet-21k	224x224	196.53	34.04	86.24	97.88	config	model
Swin-L*	ImageNet-21k	384x384	196.74	100.04	87.25	98.25	config	model

*Models with * are converted from the official repo. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

38.2.3 CUB-200-2011

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

38.3 Citation

```
@article{liu2021Swin,
  title={Swin Transformer: Hierarchical Vision Transformer using Shifted Windows},
  author={Liu, Ze and Lin, Yutong and Cao, Yue and Hu, Han and Wei, Yixuan and Zhang,␣
↪Zheng and Lin, Stephen and Guo, Baining},
  journal={arXiv preprint arXiv:2103.14030},
  year={2021}
}
```

CHAPTER

THIRTYNINE

SWIN TRANSFORMER V2

Swin Transformer V2: Scaling Up Capacity and Resolution

39.1 Abstract

Large-scale NLP models have been shown to significantly improve the performance on language tasks with no signs of saturation. They also demonstrate amazing few-shot capabilities like that of human beings. This paper aims to explore large-scale models in computer vision. We tackle three major issues in training and application of large vision models, including training instability, resolution gaps between pre-training and fine-tuning, and hunger on labelled data. Three main techniques are proposed: 1) a residual-post-norm method combined with cosine attention to improve training stability; 2) A log-spaced continuous position bias method to effectively transfer models pre-trained using low-resolution images to downstream tasks with high-resolution inputs; 3) A self-supervised pre-training method, SimMIM, to reduce the needs of vast labeled images. Through these techniques, this paper successfully trained a 3 billion-parameter Swin Transformer V2 model, which is the largest dense vision model to date, and makes it capable of training with images of up to 1,536×1,536 resolution. It set new performance records on 4 representative vision tasks, including ImageNet-V2 image classification, COCO object detection, ADE20K semantic segmentation, and Kinetics-400 video action classification. Also note our training is much more efficient than that in Google's billion-level visual models, which consumes 40 times less labelled data and 40 times less training time.

39.2 Results and models

39.2.1 ImageNet-21k

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

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

39.2.2 ImageNet-1k

Model	Pretrain	resolution	window	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
Swin-T*	From scratch	256x256	8x8	28.35	4.35	81.76	95.87	config	model
Swin-T*	From scratch	256x256	16x16	28.35	4.4	82.81	96.23	config	model
Swin-S*	From scratch	256x256	8x8	49.73	8.45	83.74	96.6	config	model
Swin-S*	From scratch	256x256	16x16	49.73	8.57	84.13	96.83	config	model
Swin-B*	From scratch	256x256	8x8	87.92	14.99	84.2	96.86	config	model
Swin-B*	From scratch	256x256	16x16	87.92	15.14	84.6	97.05	config	model
Swin-B*	ImageNet-21k	256x256	16x16	87.92	15.14	86.17	97.88	config	model
Swin-B*	ImageNet-21k	384x384	24x24	87.92	34.07	87.14	98.23	config	model
Swin-L*	ImageNet-21k	256X256	16x16	196.75	33.86	86.93	98.06	config	model
Swin-L*	ImageNet-21k	384x384	24x24	196.75	76.2	87.59	98.27	config	model

*Models with * are converted from the official repo. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

ImageNet-21k pretrained models with input resolution of 256x256 and 384x384 both fine-tuned from the same pre-training model using a smaller input resolution of 192x192.

39.3 Citation

```
@article{https://doi.org/10.48550/arxiv.2111.09883,
  doi = {10.48550/ARXIV.2111.09883},
  url = {https://arxiv.org/abs/2111.09883},
  author = {Liu, Ze and Hu, Han and Lin, Yutong and Yao, Zhuliang and Xie, Zhenda and␣
↪Wei, Yixuan and Ning, Jia and Cao, Yue and Zhang, Zheng and Dong, Li and Wei, Furu and␣
↪Guo, Baining},
  keywords = {Computer Vision and Pattern Recognition (cs.CV), FOS: Computer and␣
↪information sciences, FOS: Computer and information sciences},
  title = {Swin Transformer V2: Scaling Up Capacity and Resolution},
  publisher = {arXiv},
  year = {2021},
  copyright = {Creative Commons Attribution 4.0 International}
}
```

CHAPTER

FORTY

TOKENS-TO-TOKEN VIT

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

40.1 Abstract

Transformers, which are popular for language modeling, have been explored for solving vision tasks recently, e.g., the Vision Transformer (ViT) for image classification. The ViT model splits each image into a sequence of tokens with fixed length and then applies multiple Transformer layers to model their global relation for classification. However, ViT achieves inferior performance to CNNs when trained from scratch on a midsize dataset like ImageNet. We find it is because: 1) the simple tokenization of input images fails to model the important local structure such as edges and lines among neighboring pixels, leading to low training sample efficiency; 2) the redundant attention backbone design of ViT leads to limited feature richness for fixed computation budgets and limited training samples. To overcome such limitations, we propose a new Tokens-To-Token Vision Transformer (T2T-ViT), which incorporates 1) a layer-wise Tokens-to-Token (T2T) transformation to progressively structurize the image to tokens by recursively aggregating neighboring Tokens into one Token (Tokens-to-Token), such that local structure represented by surrounding tokens can be modeled and tokens length can be reduced; 2) an efficient backbone with a deep-narrow structure for vision transformer motivated by CNN architecture design after empirical study. Notably, T2T-ViT reduces the parameter count and MACs of vanilla ViT by half, while achieving more than 3.0% improvement when trained from scratch on ImageNet. It also outperforms ResNets and achieves comparable performance with MobileNets by directly training on ImageNet. For example, T2T-ViT with comparable size to ResNet50 (21.5M parameters) can achieve 83.3% top1 accuracy in image resolution 384×384 on ImageNet.

40.2 Results and models

40.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
T2T-ViT_t-14	21.47	4.34	81.83	95.84	config	model | log
T2T-ViT_t-19	39.08	7.80	82.63	96.18	config	model | log
T2T-ViT_t-24	64.00	12.69	82.71	96.09	config	model | log

In consistent with the official repo, we adopt the best checkpoints during training.

40.3 Citation

```
@article{yuan2021tokens,
  title={Tokens-to-token vit: Training vision transformers from scratch on imagenet},
  author={Yuan, Li and Chen, Yunpeng and Wang, Tao and Yu, Weihao and Shi, Yujun and Tay,
↪ Francis EH and Feng, Jiashi and Yan, Shuicheng},
  journal={arXiv preprint arXiv:2101.11986},
  year={2021}
}
```

CHAPTER

FORTYONE

TNT

Transformer in Transformer

41.1 Abstract

Transformer is a new kind of neural architecture which encodes the input data as powerful features via the attention mechanism. Basically, the visual transformers first divide the input images into several local patches and then calculate both representations and their relationship. Since natural images are of high complexity with abundant detail and color information, the granularity of the patch dividing is not fine enough for excavating features of objects in different scales and locations. In this paper, we point out that the attention inside these local patches are also essential for building visual transformers with high performance and we explore a new architecture, namely, Transformer iN Transformer (TNT). Specifically, we regard the local patches (e.g., 16×16) as "visual sentences" and present to further divide them into smaller patches (e.g., 4×4) as "visual words". The attention of each word will be calculated with other words in the given visual sentence with negligible computational costs. Features of both words and sentences will be aggregated to enhance the representation ability. Experiments on several benchmarks demonstrate the effectiveness of the proposed TNT architecture, e.g., we achieve an 81.5% top-1 accuracy on the ImageNet, which is about 1.7% higher than that of the state-of-the-art visual transformer with similar computational cost.

41.2 Results and models

41.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
TNT-small*	23.76	3.36	81.52	95.73	config	model

*Models with * are converted from timm. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

41.3 Citation

```
@misc{han2021transformer,
      title={Transformer in Transformer},
      author={Kai Han and An Xiao and Enhua Wu and Jianyuan Guo and Chunjing Xu and␣
↪Yunhe Wang},
      year={2021},
      eprint={2103.00112},
      archivePrefix={arXiv},
      primaryClass={cs.CV}
}
```

CHAPTER

FORTYTWO

TWINS

Twins: Revisiting the Design of Spatial Attention in Vision Transformers

42.1 Abstract

Very recently, a variety of vision transformer architectures for dense prediction tasks have been proposed and they show that the design of spatial attention is critical to their success in these tasks. In this work, we revisit the design of the spatial attention and demonstrate that a carefully-devised yet simple spatial attention mechanism performs favourably against the state-of-the-art schemes. As a result, we propose two vision transformer architectures, namely, Twins-PCPVT and Twins-SVT. Our proposed architectures are highly-efficient and easy to implement, only involving matrix multiplications that are highly optimized in modern deep learning frameworks. More importantly, the proposed architectures achieve excellent performance on a wide range of visual tasks, including image level classification as well as dense detection and segmentation. The simplicity and strong performance suggest that our proposed architectures may serve as stronger backbones for many vision tasks. Our code is released at this https URL.

42.2 Results and models

42.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
PCPVT-small*	24.11	3.67	81.14	95.69	config	model
PCPVT-base*	43.83	6.45	82.66	96.26	config	model
PCPVT-large*	60.99	9.51	83.09	96.59	config	model
SVT-small*	24.06	2.82	81.77	95.57	config	model
SVT-base*	56.07	8.35	83.13	96.29	config	model
SVT-large*	99.27	14.82	83.60	96.50	config	model

*Models with * are converted from the official repo. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results. The validation accuracy is a little different from the official paper because of the PyTorch version. This result is get in PyTorch=1.9 while the official result is get in PyTorch=1.7*

42.3 Citation

```
@article{chu2021twins,
  title={Twins: Revisiting spatial attention design in vision transformers},
  author={Chu, Xiangxiang and Tian, Zhi and Wang, Yuqing and Zhang, Bo and Ren, Haibing␣
↪and Wei, Xiaolin and Xia, Huaxia and Shen, Chunhua},
  journal={arXiv preprint arXiv:2104.13840},
  year={2021}altgvt
}
```

CHAPTER

FORTYTHREE

VISUAL ATTENTION NETWORK

Visual Attention Network

43.1 Abstract

While originally designed for natural language processing (NLP) tasks, the self-attention mechanism has recently taken various computer vision areas by storm. However, the 2D nature of images brings three challenges for applying self-attention in computer vision. (1) Treating images as 1D sequences neglects their 2D structures. (2) The quadratic complexity is too expensive for high-resolution images. (3) It only captures spatial adaptability but ignores channel adaptability. In this paper, we propose a novel large kernel attention (LKA) module to enable self-adaptive and long-range correlations in self-attention while avoiding the above issues. We further introduce a novel neural network based on LKA, namely Visual Attention Network (VAN). While extremely simple and efficient, VAN outperforms the state-of-the-art vision transformers and convolutional neural networks with a large margin in extensive experiments, including image classification, object detection, semantic segmentation, instance segmentation, etc.

43.2 Results and models

43.2.1 ImageNet-1k

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

*Models with * are converted from the official repo. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.

43.2.2 Pre-trained Models

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

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

*Models with * are converted from the official repo.

43.3 Citation

```
@article{guo2022visual,
  title={Visual Attention Network},
  author={Guo, Meng-Hao and Lu, Cheng-Ze and Liu, Zheng-Ning and Cheng, Ming-Ming and Hu,
↪ Shi-Min},
  journal={arXiv preprint arXiv:2202.09741},
  year={2022}
}
```

CHAPTER

FORTYFOUR

VGG

Very Deep Convolutional Networks for Large-Scale Image Recognition

44.1 Abstract

In this work we investigate the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. Our main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small (3x3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16-19 weight layers. These findings were the basis of our ImageNet Challenge 2014 submission, where our team secured the first and the second places in the localisation and classification tracks respectively. We also show that our representations generalise well to other datasets, where they achieve state-of-the-art results. We have made our two best-performing ConvNet models publicly available to facilitate further research on the use of deep visual representations in computer vision.

44.2 Results and models

44.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
VGG-11	132.86	7.63	68.75	88.87	config	model | log
VGG-13	133.05	11.34	70.02	89.46	config	model | log
VGG-16	138.36	15.5	71.62	90.49	config	model | log
VGG-19	143.67	19.67	72.41	90.80	config	model | log
VGG-11-BN	132.87	7.64	70.67	90.16	config	model | log
VGG-13-BN	133.05	11.36	72.12	90.66	config	model | log
VGG-16-BN	138.37	15.53	73.74	91.66	config	model | log
VGG-19-BN	143.68	19.7	74.68	92.27	config	model | log

44.3 Citation

```
@article{simonyan2014very,
  title={Very deep convolutional networks for large-scale image recognition},
  author={Simonyan, Karen and Zisserman, Andrew},
  journal={arXiv preprint arXiv:1409.1556},
  year={2014}
}
```

CHAPTER

FORTYFIVE

VISION TRANSFORMER

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

45.1 Abstract

While the Transformer architecture has become the de-facto standard for natural language processing tasks, its applications to computer vision remain limited. In vision, attention is either applied in conjunction with convolutional networks, or used to replace certain components of convolutional networks while keeping their overall structure in place. We show that this reliance on CNNs is not necessary and a pure transformer applied directly to sequences of image patches can perform very well on image classification tasks. When pre-trained on large amounts of data and transferred to multiple mid-sized or small image recognition benchmarks (ImageNet, CIFAR-100, VTAB, etc.), Vision Transformer (ViT) attains excellent results compared to state-of-the-art convolutional networks while requiring substantially fewer computational resources to train.

45.2 Results and models

The training step of Vision Transformers is divided into two steps. The first step is training the model on a large dataset, like ImageNet-21k, and get the pre-trained model. And the second step is training the model on the target dataset, like ImageNet-1k, and get the fine-tuned model. Here, we provide both pre-trained models and fine-tuned models.

45.2.1 ImageNet-21k

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

Model	resolution	Params(M)	Flops(G)	Download
ViT-B16*	224x224	86.86	33.03	model
ViT-B32*	224x224	88.30	8.56	model
ViT-L16*	224x224	304.72	116.68	model

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

45.2.2 ImageNet-1k

Model	Pretrain	resolution	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
ViT-B16*	ImageNet-21k	384x384	86.86	33.03	85.43	97.77	config	model
ViT-B32*	ImageNet-21k	384x384	88.30	8.56	84.01	97.08	config	model
ViT-L16*	ImageNet-21k	384x384	304.72	116.68	85.63	97.63	config	model
ViT-B16 (IPU)	ImageNet-21k	224x224	86.86	33.03	81.22	95.56	config	model | log

*Models with * are converted from the official repo. The config files of these models are only for validation. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

45.3 Citation

```
@inproceedings{
  dosovitskiy2021an,
  title={An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale},
  author={Alexey Dosovitskiy and Lucas Beyer and Alexander Kolesnikov and Dirk␣
↪Weissenborn and Xiaohua Zhai and Thomas Unterthiner and Mostafa Dehghani and Matthias␣
↪Minderer and Georg Heigold and Sylvain Gelly and Jakob Uszkoreit and Neil Houlsby},
  booktitle={International Conference on Learning Representations},
  year={2021},
  url={https://openreview.net/forum?id=YicbFdNTTy}
}
```

CHAPTER

FORTYSIX

WIDE-RESNET

Wide Residual Networks

46.1 Abstract

Deep residual networks were shown to be able to scale up to thousands of layers and still have improving performance. However, each fraction of a percent of improved accuracy costs nearly doubling the number of layers, and so training very deep residual networks has a problem of diminishing feature reuse, which makes these networks very slow to train. To tackle these problems, in this paper we conduct a detailed experimental study on the architecture of ResNet blocks, based on which we propose a novel architecture where we decrease depth and increase width of residual networks. We call the resulting network structures wide residual networks (WRNs) and show that these are far superior over their commonly used thin and very deep counterparts. For example, we demonstrate that even a simple 16-layer-deep wide residual network outperforms in accuracy and efficiency all previous deep residual networks, including thousand-layer-deep networks, achieving new state-of-the-art results on CIFAR, SVHN, COCO, and significant improvements on ImageNet.

46.2 Results and models

46.2.1 ImageNet-1k

Model	Params(M)	Flops(G)	Top-1 (%)	Top-5 (%)	Config	Download
WRN-50*	68.88	11.44	78.48	94.08	config	model
WRN-101*	126.89	22.81	78.84	94.28	config	model
WRN-50 (timm)*	68.88	11.44	81.45	95.53	config	model

*Models with * are converted from the TorchVision and TIMM. The config files of these models are only for inference. We don't ensure these config files' training accuracy and welcome you to contribute your reproduction results.*

46.3 Citation

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

CHAPTER

FORTYSEVEN

PYTORCH TO ONNX (EXPERIMENTAL)

47.1 How to convert models from Pytorch to ONNX

47.1.1 Prerequisite

1. Please refer to install for installation of MMClassification.
2. Install onnx and onnxruntime

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

47.1.2 Usage

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

47.1.3 Description of all arguments:

- `config` : The path of a model config file.
- `--checkpoint` : The path of a model checkpoint file.
- `--output-file`: The path of output ONNX model. If not specified, it will be set to `tmp.onnx`.
- `--shape`: The height and width of input tensor to the model. If not specified, it will be set to `224 224`.
- `--opset-version` : The opset version of ONNX. If not specified, it will be set to `11`.
- `--dynamic-export` : Determines whether to export ONNX with dynamic input shape and output shapes. If not specified, it will be set to `False`.
- `--show`: Determines whether to print the architecture of the exported model. If not specified, it will be set to `False`.
- `--simplify`: Determines whether to simplify the exported ONNX model. If not specified, it will be set to `False`.
- `--verify`: Determines whether to verify the correctness of an exported model. If not specified, it will be set to `False`.

Example:

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

47.2 How to evaluate ONNX models with ONNX Runtime

We prepare a tool `tools/deployment/test.py` to evaluate ONNX models with ONNXRuntime or TensorRT.

47.2.1 Prerequisite

- Install onnx and onnxruntime-gpu

  ```
  pip install onnx onnxruntime-gpu
  ```

47.2.2 Usage

```
python tools/deployment/test.py \
    ${CONFIG_FILE} \
    ${ONNX_FILE} \
    --backend ${BACKEND} \
    --out ${OUTPUT_FILE} \
    --metrics ${EVALUATION_METRICS} \
    --metric-options ${EVALUATION_OPTIONS} \
    --show
    --show-dir ${SHOW_DIRECTORY} \
    --cfg-options ${CFG_OPTIONS} \
```

47.2.3 Description of all arguments

- `config`: The path of a model config file.
- `model`: The path of a ONNX model file.
- `--backend`: Backend for input model to run and should be `onnxruntime` or `tensorrt`.
- `--out`: The path of output result file in pickle format.
- `--metrics`: Evaluation metrics, which depends on the dataset, e.g., "accuracy", "precision", "recall", "f1_score", "support" for single label dataset, and "mAP", "CP", "CR", "CF1", "OP", "OR", "OF1" for multi-label dataset.
- `--show`: Determines whether to show classifier outputs. If not specified, it will be set to `False`.
- `--show-dir`: Directory where painted images will be saved
- `--metrics-options`: Custom options for evaluation, the key-value pair in `xxx=yyy` format will be kwargs for `dataset.evaluate()` function
- `--cfg-options`: Override some settings in the used config file, the key-value pair in `xxx=yyy` format will be merged into config file.

47.2.4 Results and Models

This part selects ImageNet for onnxruntime verification. ImageNet has multiple versions, but the most commonly used one is ILSVRC 2012.

47.3 List of supported models exportable to ONNX

The table below lists the models that are guaranteed to be exportable to ONNX and runnable in ONNX Runtime.

Model	Config	Batch Inference	Dynamic Shape	Note
MobileNetV2	mobilenet-v2_8xb32_in1k.py	Y	Y	
ResNet	resnet18_8xb16_cifar10.py	Y	Y	
ResNeXt	resnext50-32x4d_8xb32_in1k.py	Y	Y	
SE-ResNet	seresnet50_8xb32_in1k.py	Y	Y	
ShuffleNetV1	shufflenet-v1-1x_16xb64_in1k.py	Y	Y	
ShuffleNetV2	shufflenet-v2-1x_16xb64_in1k.py	Y	Y	

Notes:

- *All models above are tested with Pytorch==1.6.0*

47.4 Reminders

- If you meet any problem with the listed models above, please create an issue and it would be taken care of soon. For models not included in the list, please try to dig a little deeper and debug a little bit more and hopefully solve them by yourself.

47.5 FAQs

- None

CHAPTER

FORTYEIGHT

ONNX TO TENSORRT (EXPERIMENTAL)

48.1 How to convert models from ONNX to TensorRT

48.1.1 Prerequisite

1. Please refer to install.md for installation of MMClassification from source.
2. Use our tool *pytorch2onnx.md* to convert the model from PyTorch to ONNX.

48.1.2 Usage

```
python tools/deployment/onnx2tensorrt.py \
    ${MODEL} \
    --trt-file ${TRT_FILE} \
    --shape ${IMAGE_SHAPE} \
    --max-batch-size ${MAX_BATCH_SIZE} \
    --workspace-size ${WORKSPACE_SIZE} \
    --fp16 \
    --show \
    --verify \
```

Description of all arguments:

- `model` : The path of an ONNX model file.
- `--trt-file`: The Path of output TensorRT engine file. If not specified, it will be set to `tmp.trt`.
- `--shape`: The height and width of model input. If not specified, it will be set to `224 224`.
- `--max-batch-size`: The max batch size of TensorRT model, should not be less than 1.

- `--fp16`: Enable fp16 mode.
- `--workspace-size` : The required GPU workspace size in GiB to build TensorRT engine. If not specified, it will be set to 1 GiB.
- `--show`: Determines whether to show the outputs of the model. If not specified, it will be set to `False`.
- `--verify`: Determines whether to verify the correctness of models between ONNXRuntime and TensorRT. If not specified, it will be set to `False`.

Example:

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

48.2 List of supported models convertible to TensorRT

The table below lists the models that are guaranteed to be convertible to TensorRT.

Model	Config	Status
MobileNetV2	`configs/mobilenet_v2/mobilenet-v2_8xb32_in1k.py`	Y
ResNet	`configs/resnet/resnet18_8xb16_cifar10.py`	Y
ResNeXt	`configs/resnext/resnext50-32x4d_8xb32_in1k.py`	Y
ShuffleNetV1	`configs/shufflenet_v1/shufflenet-v1-1x_16xb64_in1k.py`	Y
ShuffleNetV2	`configs/shufflenet_v2/shufflenet-v2-1x_16xb64_in1k.py`	Y

Notes:

- *All models above are tested with Pytorch==1.6.0 and TensorRT-7.2.1.6.Ubuntu-16.04.x86_64-gnu.cuda-10.2.cudnn8.0*

48.3 Reminders

- If you meet any problem with the listed models above, please create an issue and it would be taken care of soon. For models not included in the list, we may not provide much help here due to the limited resources. Please try to dig a little deeper and debug by yourself.

48.4 FAQs

- None

CHAPTER

FORTYNINE

PYTORCH TO TORCHSCRIPT (EXPERIMENTAL)

49.1 How to convert models from Pytorch to TorchScript

49.1.1 Usage

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

49.1.2 Description of all arguments

- `config` : The path of a model config file.
- `--checkpoint` : The path of a model checkpoint file.
- `--output-file`: The path of output TorchScript model. If not specified, it will be set to `tmp.pt`.
- `--shape`: The height and width of input tensor to the model. If not specified, it will be set to `224 224`.
- `--verify`: Determines whether to verify the correctness of an exported model. If not specified, it will be set to `False`.

Example:

```
python tools/deployment/pytorch2torchscript.py \
    configs/resnet/resnet18_8xb16_cifar10.py \
    --checkpoint checkpoints/resnet/resnet18_8xb16_cifar10.pth \
```

(continues on next page)

(continued from previous page)

```
    --output-file checkpoints/resnet/resnet18_8xb16_cifar10.pt \
    --verify \
```

Notes:

- *All models are tested with Pytorch==1.8.1*

49.2 Reminders

- For torch.jit.is_tracing() is only supported after v1.6. For users with pytorch v1.3-v1.5, we suggest early returning tensors manually.
- If you meet any problem with the models in this repo, please create an issue and it would be taken care of soon.

49.3 FAQs

- None

CHAPTER

FIFTY

MODEL SERVING

In order to serve an `MMClassification` model with `TorchServe`, you can follow the steps:

50.1 1. Convert model from MMClassification to TorchServe

```
python tools/deployment/mmcls2torchserve.py ${CONFIG_FILE} ${CHECKPOINT_FILE} \
--output-folder ${MODEL_STORE} \
--model-name ${MODEL_NAME}
```

Note: ${MODEL_STORE} needs to be an absolute path to a folder.

Example:

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

50.2 2. Build `mmcls-serve` docker image

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

50.3 3. Run `mmcls-serve`

Check the official docs for running TorchServe with docker.

In order to run in GPU, you need to install nvidia-docker. You can omit the `--gpus` argument in order to run in GPU.

Example:

```
docker run --rm \
--cpus 8 \
--gpus device=0 \
-p8080:8080 -p8081:8081 -p8082:8082 \
```

(continues on next page)

(continued from previous page)

```
--mount type=bind,source=`realpath ./checkpoints`,target=/home/model-server/model-store \
mmcls-serve:latest
```

Note: `realpath ./checkpoints` points to the absolute path of "./checkpoints", and you can replace it with the absolute path where you store torchserve models.

Read the docs about the Inference (8080), Management (8081) and Metrics (8082) APis

50.4 4. Test deployment

```
curl http://127.0.0.1:8080/predictions/${MODEL_NAME} -T demo/demo.JPEG
```

You should obtain a response similar to:

```
{
  "pred_label": 58,
  "pred_score": 0.38102269172668457,
  "pred_class": "water snake"
}
```

And you can use `test_torchserver.py` to compare result of TorchServe and PyTorch, and visualize them.

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

Example:

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

CHAPTER

FIFTYONE

VISUALIZATION

51.1 Pipeline Visualization

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

Description of all arguments:

- `config` : The path of a model config file.
- `--output-dir`: The output path for visualized images. If not specified, it will be set to `''`, which means not to save.
- `--phase`: Phase of visualizing dataset，must be one of `[train, val, test]`. If not specified, it will be set to `train`.
- `--number`: The number of samples to visualized. If not specified, display all images in the dataset.
- `--skip-type`: The pipelines to be skipped. If not specified, it will be set to `['ToTensor', 'Normalize', 'ImageToTensor', 'Collect']`.
- `--mode`: The display mode, can be one of `[original, pipeline, concat]`. If not specified, it will be set to `concat`.
- `--show`: If set, display pictures in pop-up windows.

- `--adaptive`: If set, adaptively resize images for better visualization.
- `--min-edge-length`: The minimum edge length, used when `--adaptive` is set. When any side of the picture is smaller than `${MIN_EDGE_LENGTH}`, the picture will be enlarged while keeping the aspect ratio unchanged, and the short side will be aligned to `${MIN_EDGE_LENGTH}`. If not specified, it will be set to 200.
- `--max-edge-length`: The maximum edge length, used when `--adaptive` is set. When any side of the picture is larger than `${MAX_EDGE_LENGTH}`, the picture will be reduced while keeping the aspect ratio unchanged, and the long side will be aligned to `${MAX_EDGE_LENGTH}`. If not specified, it will be set to 1000.
- `--bgr2rgb`: If set, flip the color channel order of images.
- `--window-size`: The shape of the display window. If not specified, it will be set to `12*7`. If used, it must be in the format `'W*H'`.
- `--cfg-options` : Modifications to the configuration file, refer to Tutorial 1: Learn about Configs.

Note:

1. If the `--mode` is not specified, it will be set to `concat` as default, get the pictures stitched together by original pictures and transformed pictures; if the `--mode` is set to `original`, get the original pictures; if the `--mode` is set to `transformed`, get the transformed pictures; if the `--mode` is set to `pipeline`, get all the intermediate images through the pipeline.
2. When `--adaptive` option is set, images that are too large or too small will be automatically adjusted, you can use `--min-edge-length` and `--max-edge-length` to set the adjust size.

Examples:

1. In **'original'** mode, visualize 100 original pictures in the `CIFAR100` validation set, then display and save them in the `./tmp` folder:

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

2. In **'transformed'** mode, visualize all the transformed pictures of the `ImageNet` training set and display them in pop-up windows:

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

3. In **'concat'** mode, visualize 10 pairs of origin and transformed images for comparison in the `ImageNet` train set and save them in the `./tmp` folder:

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

4. In **'pipeline'** mode, visualize all the intermediate pictures in the `ImageNet` train set through the pipeline:

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

51.2 Learning Rate Schedule Visualization

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

Description of all arguments:

- `config` : The path of a model config file.
- `dataset-size` : The size of the datasets. If set， `build_dataset` will be skipped and `${DATASET_SIZE}` will be used as the size. Default to use the function `build_dataset`.
- `ngpus` : The number of GPUs used in training, default to be 1.
- `save-path` : The learning rate curve plot save path, default not to save.
- `title` : Title of figure. If not set, default to be config file name.
- `style` : Style of plt. If not set, default to be `whitegrid`.
- `window-size`: The shape of the display window. If not specified, it will be set to `12*7`. If used, it must be in the format `'W*H'`.
- `cfg-options` : Modifications to the configuration file, refer to Tutorial 1: Learn about Configs.

Note: Loading annotations maybe consume much time, you can directly specify the size of the dataset with `dataset-size` to save time.

Examples:

```
python tools/visualizations/vis_lr.py configs/resnet/resnet50_b16x8_cifar100.py
```

When using ImageNet, directly specify the size of ImageNet, as below:

```
python tools/visualizations/vis_lr.py configs/repvgg/repvgg-B3g4_4xb64-autoaug-lbs-mixup-
↪coslr-200e_in1k.py --dataset-size 1281167 --ngpus 4 --save-path ./repvgg-B3g4_4xb64-lr.
↪jpg
```

51.3 Class Activation Map Visualization

MMClassification provides `tools\visualizations\vis_cam.py` tool to visualize class activation map. Please use `pip install "grad-cam>=1.3.6"` command to install pytorch-grad-cam.

The supported methods are as follows:

Method	What it does
GradCAM	Weight the 2D activations by the average gradient
Grad-CAM++	Like GradCAM but uses second order gradients
XGradCAM	Like GradCAM but scale the gradients by the normalized activations
EigenCAM	Takes the first principle component of the 2D Activations (no class discrimination, but seems to give great results)
EigenGrad-CAM	Like EigenCAM but with class discrimination: First principle component of Activations*Grad. Looks like GradCAM, but cleaner
LayerCAM	Spatially weight the activations by positive gradients. Works better especially in lower layers

Command:

```
python tools/visualizations/vis_cam.py \
    ${IMG} \
    ${CONFIG_FILE} \
    ${CHECKPOINT} \
    [--target-layers ${TARGET-LAYERS}] \
    [--preview-model] \
    [--method ${METHOD}] \
    [--target-category ${TARGET-CATEGORY}] \
    [--save-path ${SAVE_PATH}] \
    [--vit-like] \
    [--num-extra-tokens ${NUM-EXTRA-TOKENS}]
    [--aug_smooth] \
    [--eigen_smooth] \
    [--device ${DEVICE}] \
    [--cfg-options ${CFG-OPTIONS}]
```

Description of all arguments:

- `img` : The target picture path.
- `config` : The path of the model config file.
- `checkpoint` : The path of the checkpoint.
- `--target-layers` : The target layers to get activation maps, one or more network layers can be specified. If not set, use the norm layer of the last block.
- `--preview-model` : Whether to print all network layer names in the model.
- `--method` : Visualization method, supports `GradCAM`, `GradCAM++`, `XGradCAM`, `EigenCAM`, `EigenGradCAM`, `LayerCAM`, which is case insensitive. Defaults to `GradCAM`.
- `--target-category` : Target category, if not set, use the category detected by the given model.
- `--save-path` : The path to save the CAM visualization image. If not set, the CAM image will not be saved.
- `--vit-like` : Whether the network is ViT-like network.
- `--num-extra-tokens` : The number of extra tokens in ViT-like backbones. If not set, use num_extra_tokens the backbone.
- `--aug_smooth` : Whether to use TTA(Test Time Augment) to get CAM.
- `--eigen_smooth` : Whether to use the principal component to reduce noise.
- `--device` : The computing device used. Default to 'cpu'.

- `--cfg-options` : Modifications to the configuration file, refer to Tutorial 1: Learn about Configs.

Note: The argument `--preview-model` can view all network layers names in the given model. It will be helpful if you know nothing about the model layers when setting `--target-layers`.

Examples(CNN):

Here are some examples of `target-layers` in ResNet-50, which can be any module or layer:

- `'backbone.layer4'` means the output of the forth ResLayer.
- `'backbone.layer4.2'` means the output of the third BottleNeck block in the forth ResLayer.
- `'backbone.layer4.2.conv1'` means the output of the `conv1` layer in above BottleNeck block.

Note: For `ModuleList` or `Sequential`, you can also use the index to specify which sub-module is the target layer.

For example, the `backbone.layer4[-1]` is the same as `backbone.layer4.2` since `layer4` is a `Sequential` with three sub-modules.

1. Use different methods to visualize CAM for `ResNet50`, the `target-category` is the predicted result by the given checkpoint, using the default `target-layers`.

```
python tools/visualizations/vis_cam.py \
    demo/bird.JPEG \
    configs/resnet/resnet50_8xb32_in1k.py \
    https://download.openmmlab.com/mmclassification/v0/resnet/resnet50_batch256_
↪imagenet_20200708-cfb998bf.pth \
    --method GradCAM
    # GradCAM++, XGradCAM, EigenCAM, EigenGradCAM, LayerCAM
```

Image	GradCAM	GradCAM++	EigenGradCAM	LayerCAM

2. Use different `target-category` to get CAM from the same picture. In `ImageNet` dataset, the category 238 is 'Greater Swiss Mountain dog', the category 281 is 'tabby, tabby cat'.

```
python tools/visualizations/vis_cam.py \
    demo/cat-dog.png configs/resnet/resnet50_8xb32_in1k.py \
    https://download.openmmlab.com/mmclassification/v0/resnet/resnet50_batch256_
↪imagenet_20200708-cfb998bf.pth \
    --target-layers 'backbone.layer4.2' \
    --method GradCAM \
    --target-category 238
    # --target-category 281
```

Category	Image	GradCAM	XGradCAM	LayerCAM
Dog				
Cat				

3. Use `--eigen-smooth` and `--aug-smooth` to improve visual effects.

```
python tools/visualizations/vis_cam.py \
    demo/dog.jpg  \
    configs/mobilenet_v3/mobilenet-v3-large_8xb32_in1k.py \
    https://download.openmmlab.com/mmclassification/v0/mobilenet_v3/convert/
↪mobilenet_v3_large-3ea3c186.pth \
    --target-layers 'backbone.layer16' \
    --method LayerCAM \
    --eigen-smooth --aug-smooth
```

Image	LayerCAM	eigen-smooth	aug-smooth	eigen&aug

Examples(Transformer):

Here are some examples:

- `'backbone.norm3'` for Swin-Transformer;
- `'backbone.layers[-1].ln1'` for ViT;

For ViT-like networks, such as ViT, T2T-ViT and Swin-Transformer, the features are flattened. And for drawing the CAM, we need to specify the `--vit-like` argument to reshape the features into square feature maps.

Besides the flattened features, some ViT-like networks also add extra tokens like the class token in ViT and T2T-ViT, and the distillation token in DeiT. In these networks, the final classification is done on the tokens computed in the last attention block, and therefore, the classification score will not be affected by other features and the gradient of the classification score with respect to them, will be zero. Therefore, you shouldn't use the output of the last attention block as the target layer in these networks.

To exclude these extra tokens, we need know the number of extra tokens. Almost all transformer-based backbones in MMClassification have the `num_extra_tokens` attribute. If you want to use this tool in a new or third-party network that don't have the `num_extra_tokens` attribute, please specify it the `--num-extra-tokens` argument.

1. Visualize CAM for `Swin Transformer`, using default `target-layers`:

```
python tools/visualizations/vis_cam.py \
    demo/bird.JPEG  \
    configs/swin_transformer/swin-tiny_16xb64_in1k.py \
    https://download.openmmlab.com/mmclassification/v0/swin-transformer/swin_tiny_
↪224_b16x64_300e_imagenet_20210616_090925-66df6be6.pth \
    --vit-like
```

2. Visualize CAM for `Vision Transformer(ViT)`:

```
python tools/visualizations/vis_cam.py \
    demo/bird.JPEG  \
    configs/vision_transformer/vit-base-p16_ft-64xb64_in1k-384.py \
    https://download.openmmlab.com/mmclassification/v0/vit/finetune/vit-base-p16_
↪in21k-pre-3rdparty_ft-64xb64_in1k-384_20210928-98e8652b.pth \
    --vit-like \
    --target-layers 'backbone.layers[-1].ln1'
```

3. Visualize CAM for `T2T-ViT`:

```
python tools/visualizations/vis_cam.py \
    demo/bird.JPEG  \
    configs/t2t_vit/t2t-vit-t-14_8xb64_in1k.py \
    https://download.openmmlab.com/mmclassification/v0/t2t-vit/t2t-vit-t-14_
↪3rdparty_8xb64_in1k_20210928-b7c09b62.pth \
    --vit-like \
    --target-layers 'backbone.encoder[-1].ln1'
```

Image	ResNet50	ViT	Swin	T2T-ViT

51.4 FAQs

- None

CHAPTER

FIFTYTWO

ANALYSIS

52.1 Log Analysis

52.1.1 Plot Curves

`tools/analysis_tools/analyze_logs.py` plots curves of given keys according to the log files.

```
python tools/analysis_tools/analyze_logs.py plot_curve  \
    ${JSON_LOGS}  \
    [--keys ${KEYS}]  \
    [--title ${TITLE}]  \
    [--legend ${LEGEND}]  \
    [--backend ${BACKEND}]  \
    [--style ${STYLE}]  \
    [--out ${OUT_FILE}]  \
    [--window-size ${WINDOW_SIZE}]
```

Description of all arguments:

- `json_logs` : The paths of the log files, separate multiple files by spaces.
- `--keys` : The fields of the logs to analyze, separate multiple keys by spaces. Defaults to 'loss'.
- `--title` : The title of the figure. Defaults to use the filename.
- `--legend` : The names of legend, the number of which must be equal to `len(${JSON_LOGS}) * len(${KEYS})`. Defaults to use `"${JSON_LOG}-${KEYS}"`.
- `--backend` : The backend of matplotlib. Defaults to auto selected by matplotlib.
- `--style` : The style of the figure. Default to `whitegrid`.

- `--out` : The path of the output picture. If not set, the figure won't be saved.
- `--window-size`: The shape of the display window. The format should be `'W*H'`. Defaults to `'12*7'`.

Note: The `--style` option depends on `seaborn` package, please install it before setting it.

Examples:

- Plot the loss curve in training.

```
python tools/analysis_tools/analyze_logs.py plot_curve your_log_json --keys loss --
↪legend loss
```

- Plot the top-1 accuracy and top-5 accuracy curves, and save the figure to results.jpg.

```
python tools/analysis_tools/analyze_logs.py plot_curve your_log_json --keys␣
↪accuracy_top-1 accuracy_top-5  --legend top1 top5 --out results.jpg
```

- Compare the top-1 accuracy of two log files in the same figure.

```
python tools/analysis_tools/analyze_logs.py plot_curve log1.json log2.json --keys␣
↪accuracy_top-1 --legend exp1 exp2
```

Note: The tool will automatically select to find keys in training logs or validation logs according to the keys. Therefore, if you add a custom evaluation metric, please also add the key to `TEST_METRICS` in this tool.

52.1.2 Calculate Training Time

`tools/analysis_tools/analyze_logs.py` can also calculate the training time according to the log files.

```
python tools/analysis_tools/analyze_logs.py cal_train_time \
    ${JSON_LOGS}
    [--include-outliers]
```

Description of all arguments:

- `json_logs` : The paths of the log files, separate multiple files by spaces.
- `--include-outliers` : If set, include the first iteration in each epoch (Sometimes the time of first iterations is longer).

Example:

```
python tools/analysis_tools/analyze_logs.py cal_train_time work_dirs/some_exp/20200422_
↪153324.log.json
```

The output is expected to be like the below.

```
-----Analyze train time of work_dirs/some_exp/20200422_153324.log.json-----
slowest epoch 68, average time is 0.3818
fastest epoch 1, average time is 0.3694
time std over epochs is 0.0020
average iter time: 0.3777 s/iter
```

52.2 Result Analysis

With the `--out` argument in `tools/test.py`, we can save the inference results of all samples as a file. And with this result file, we can do further analysis.

52.2.1 Evaluate Results

`tools/analysis_tools/eval_metric.py` can evaluate metrics again.

```
python tools/analysis_tools/eval_metric.py \
      ${CONFIG} \
      ${RESULT} \
      [--metrics ${METRICS}]  \
      [--cfg-options ${CFG_OPTIONS}] \
      [--metric-options ${METRIC_OPTIONS}]
```

Description of all arguments:

- `config` : The path of the model config file.
- `result`: The Output result file in json/pickle format from `tools/test.py`.
- `--metrics` : Evaluation metrics, the acceptable values depend on the dataset.
- `--cfg-options`: If specified, the key-value pair config will be merged into the config file, for more details please refer to *Tutorial 1: Learn about Configs*
- `--metric-options`: If specified, the key-value pair arguments will be passed to the `metric_options` argument of dataset's `evaluate` function.

Note: In `tools/test.py`, we support using `--out-items` option to select which kind of results will be saved. Please ensure the result file includes "class_scores" to use this tool.

Examples:

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

52.2.2 View Typical Results

`tools/analysis_tools/analyze_results.py` can save the images with the highest scores in successful or failed prediction.

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

Description of all arguments:

- `config` : The path of the model config file.

- `result`: Output result file in json/pickle format from `tools/test.py`.
- `--out-dir`: Directory to store output files.
- `--topk`: The number of images in successful or failed prediction with the highest `topk` scores to save. If not specified, it will be set to 20.
- `--cfg-options`: If specified, the key-value pair config will be merged into the config file, for more details please refer to *Tutorial 1: Learn about Configs*

Note: In `tools/test.py`, we support using `--out-items` option to select which kind of results will be saved. Please ensure the result file includes "pred_score", "pred_label" and "pred_class" to use this tool.

Examples:

```
python tools/analysis_tools/analyze_results.py \
       configs/resnet/resnet50_b32x8_imagenet.py \
       result.pkl \
       --out-dir results \
       --topk 50
```

52.3 Model Complexity

52.3.1 Get the FLOPs and params (experimental)

We provide a script adapted from flops-counter.pytorch to compute the FLOPs and params of a given model.

```
python tools/analysis_tools/get_flops.py ${CONFIG_FILE} [--shape ${INPUT_SHAPE}]
```

Description of all arguments:

- `config` : The path of the model config file.
- `--shape`: Input size, support single value or double value parameter, such as `--shape 256` or `--shape 224 256`. If not set, default to be `224 224`.

You will get a result like this.

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

Warning: This tool is still experimental and we do not guarantee that the number is correct. You may well use the result for simple comparisons, but double-check it before you adopt it in technical reports or papers.

- FLOPs are related to the input shape while parameters are not. The default input shape is (1, 3, 224, 224).
- Some operators are not counted into FLOPs like GN and custom operators. Refer to `mmcv.cnn.get_model_complexity_info()` for details.

52.4 FAQs

- None

CHAPTER FIFTYTHREE

MISCELLANEOUS

53.1 Print the entire config

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

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

Description of all arguments:

- `config` : The path of the model config file.
- `--cfg-options`: If specified, the key-value pair config will be merged into the config file, for more details please refer to *Tutorial 1: Learn about Configs*

Examples:

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

53.2 Verify Dataset

`tools/misc/verify_dataset.py` can verify dataset, check whether there are broken pictures in the given dataset.

```
python tools/misc/verify_dataset.py \
    ${CONFIG} \
    [--out-path ${OUT-PATH}] \
    [--phase ${PHASE}] \
    [--num-process ${NUM-PROCESS}]
    [--cfg-options ${CFG_OPTIONS}]
```

Description of all arguments:

- `config` : The path of the model config file.
- `--out-path` : The path to save the verification result, if not set, defaults to 'brokenfiles.log'.
- `--phase` : Phase of dataset to verify, accept "train" "test" and "val", if not set, defaults to "train".

- `--num-process` : number of process to use, if not set, defaults to 1.
- `--cfg-options`: If specified, the key-value pair config will be merged into the config file, for more details please refer to *Tutorial 1: Learn about Configs*

Examples:

```
python tools/misc/verify_dataset.py configs/t2t_vit/t2t-vit-t-14_8xb64_in1k.py --out-
↪path broken_imgs.log --phase val --num-process 8
```

53.3 FAQs

- None

CHAPTER

FIFTYFOUR

CONTRIBUTING TO OPENMMLAB

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

- Fix typo or bugs
- Add documentation or translate the documentation into other languages
- Add new features and components

54.1 Workflow

1. fork and pull the latest OpenMMLab repository (MMClassification)
2. checkout a new branch (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.

54.2 Code style

54.2.1 Python

We adopt PEP8 as the preferred code style.

We use the following tools for linting and formatting:

- flake8: A wrapper around some linter tools.
- isort: A Python utility to sort imports.
- yapf: A formatter for Python files.
- 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 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`, `double-quoted-strings`, `python-encoding-pragma`, `mixed-line-ending`, 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.

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

54.2.2 C++ and CUDA

We follow the Google C++ Style Guide.

CHAPTER FIFTYFIVE

MMCLS.APIS

These are some high-level APIs for classification tasks.

mmcls.apis

55.1 Train

55.2 Test

55.3 Inference

CHAPTER

FIFTYSIX

MMCLS.CORE

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

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

mmcls.core

56.1 Evaluation

Evaluation metrics calculation functions

56.2 Hook

56.3 Optimizers

CHAPTER

FIFTYSEVEN

MMCLS.MODELS

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

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

57.1 Classifier

57.2 Backbones

57.3 Necks

57.4 Heads

57.5 Losses

CHAPTER

FIFTYEIGHT

MMCLS.MODELS.UTILS

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

mmcls.models.utils

58.1 Common Components

58.2 Helper Functions

58.2.1 channel_shuffle

58.2.2 make_divisible

58.2.3 to_ntuple

58.2.4 is_tracing

CHAPTER

FIFTYNINE

MMCLS.DATASETS

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

59.1 Custom Dataset

59.2 ImageNet

59.3 CIFAR

59.4 MNIST

59.5 VOC

59.6 StanfordCars Cars

59.7 Base classes

59.8 Dataset Wrappers

CHAPTER

SIXTY

DATA TRANSFORMATIONS

In MMClassification, the data preparation and the dataset is decomposed. The datasets only define how to get samples' basic information from the file system. These basic information includes the ground-truth label and raw images data / the paths of images.

To prepare the inputs data, we need to do some transformations on these basic information. These transformations includes loading, preprocessing and formatting. And a series of data transformations makes up a data pipeline. Therefore, you can find the a `pipeline` argument in the configs of dataset, for example:

```
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395, 57.12, 57.375], to_rgb=True)
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='RandomResizedCrop', size=224),
    dict(type='RandomFlip', flip_prob=0.5, direction='horizontal'),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='ImageToTensor', keys=['img']),
    dict(type='ToTensor', keys=['gt_label']),
    dict(type='Collect', keys=['img', 'gt_label'])
]
test_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='Resize', size=256),
    dict(type='CenterCrop', crop_size=224),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='ImageToTensor', keys=['img']),
    dict(type='Collect', keys=['img'])
]

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

Every item of a pipeline list is one of the following data transformations class. And if you want to add a custom data transformation class, the tutorial *Custom Data Pipelines* will help you.

mmcls.datasets.pipelines

60.1 Loading

60.1.1 LoadImageFromFile

60.2 Preprocessing and Augmentation

60.2.1 CenterCrop

60.2.2 Lighting

60.2.3 Normalize

60.2.4 Pad

60.2.5 Resize

60.2.6 RandomCrop

60.2.7 RandomErasing

60.2.8 RandomFlip

60.2.9 RandomGrayscale

60.2.10 RandomResizedCrop

60.2.11 ColorJitter

60.2.12 Composed Augmentation

Composed augmentation is a kind of methods which compose a series of data augmentation transformations, such as `AutoAugment` and `RandAugment`.

In composed augmentation, we need to specify several data transformations or several groups of data transformations (The `policies` argument) as the random sampling space. These data transformations are chosen from the below table. In addition, we provide some preset policies in this folder.

60.3 Formatting

60.3.1 Collect

60.3.2 ImageToTensor

60.3.3 ToNumpy

60.3.4 ToPIL

60.3.5 ToTensor

60.3.6 Transpose

CHAPTER

SIXTYONE

BATCH AUGMENTATION

Batch augmentation is the augmentation which involve multiple samples, such as Mixup and CutMix.

In MMClassification, these batch augmentation is used as a part of *Classifier*. A typical usage is as below:

```
model = dict(
    backbone = ...,
    neck = ...,
    head = ...,
    train_cfg=dict(augments=[
        dict(type='BatchMixup', alpha=0.8, prob=0.5, num_classes=num_classes),
        dict(type='BatchCutMix', alpha=1.0, prob=0.5, num_classes=num_classes),
    ]))
)
```

61.1 Mixup

61.2 CutMix

61.3 ResizeMix

CHAPTER

SIXTYTWO

MMCLS.UTILS

These are some useful help function in the `utils` package.

CHAPTER

SIXTYTHREE

CHANGELOG

63.1 v0.25.0(06/12/2022)

63.1.1 Highlights

- Support MLU backend.

63.1.2 New Features

- Support MLU backend. (#1159)
- Support Activation Checkpointing for ConvNeXt. (#1152)

63.1.3 Improvements

- Add `dist_train_arm.sh` for ARM device and update NPU results. (#1218)

63.1.4 Bug Fixes

- Fix a bug caused `MMClsWandbHook` stuck. (#1242)
- Fix the redundant `device_ids` in `tools/test.py`. (#1215)

63.1.5 Docs Update

- Add version banner and version warning in master docs. (#1216)
- Update NPU support doc. (#1198)
- Fixed typo in `pytorch2torchscript.md`. (#1173)
- Fix typo in `miscellaneous.md`. (#1137)
- further detail for the doc for `ClassBalancedDataset`. (#901)

63.2 v0.24.1(31/10/2022)

63.2.1 New Features

- Support mmcls with NPU backend. (#1072)

63.2.2 Bug Fixes

- Fix performance issue in convnext DDP train. (#1098)

63.3 v0.24.0(30/9/2022)

63.3.1 Highlights

- Support HorNet, EfficientFormerm, SwinTransformer V2 and MViT backbones.
- Support Standford Cars dataset.

63.3.2 New Features

- Support HorNet Backbone. (#1013)
- Support EfficientFormer. (#954)
- Support Stanford Cars dataset. (#893)
- Support CSRA head. (#881)
- Support Swin Transform V2. (#799)
- Support MViT and add checkpoints. (#924)

63.3.3 Improvements

- [Improve] replace loop of progressbar in api/test. (#878)
- [Enhance] RepVGG for YOLOX-PAI. (#1025)
- [Enhancement] Update VAN. (#1017)
- [Refactor] Re-write `get_sinusoid_encoding` from third-party implementation. (#965)
- [Improve] Upgrade onnxsim to v0.4.0. (#915)
- [Improve] Fixed typo in `RepVGG`. (#985)
- [Improve] Using `train_step` instead of `forward` in PreciseBNHook (#964)
- [Improve] Use `forward_dummy` to calculate FLOPS. (#953)

63.3.4 Bug Fixes

- Fix warning with `torch.meshgrid`. (#860)
- Add matplotlib minimum version requriments. (#909)
- val loader should not drop last by default. (#857)
- Fix config.device bug in toturial. (#1059)
- Fix attenstion clamp max params (#1034)
- Fix device mismatch in Swin-v2. (#976)
- Fix the output position of Swin-Transformer. (#947)

63.3.5 Docs Update

- Fix typo in config.md. (#827)
- Add version for torchvision to avoide error. (#903)
- Fixed typo for `--out-dir` option of analyze_results.py. (#898)
- Refine the docstring of RegNet (#935)

63.4 v0.23.2(28/7/2022)

63.4.1 New Features

- Support MPS device. (#894)

63.4.2 Bug Fixes

- Fix a bug in Albu which caused crashing. (#918)

63.5 v0.23.1(2/6/2022)

63.5.1 New Features

- Dedicated MMClsWandbHook for MMClassification (Weights and Biases Integration) (#764)

63.5.2 Improvements

- Use mdformat instead of markdownlint to format markdown. (#844)

63.5.3 Bug Fixes

- Fix wrong `--local_rank`.

63.5.4 Docs Update

- Update install tutorials. (#854)
- Fix wrong link in README. (#835)

63.6 v0.23.0(1/5/2022)

63.6.1 New Features

- Support DenseNet. (#750)
- Support VAN. (#739)

63.6.2 Improvements

- Support training on IPU and add fine-tuning configs of ViT. (#723)

63.6.3 Docs Update

- New style API reference, and easier to use! Welcome view it. (#774)

63.7 v0.22.1(15/4/2022)

63.7.1 New Features

- [Feature] Support resize relative position embedding in `SwinTransformer`. (#749)
- [Feature] Add PoolFormer backbone and checkpoints. (#746)

63.7.2 Improvements

- [Enhance] Improve CPE performance by reduce memory copy. (#762)
- [Enhance] Add extra dataloader settings in configs. (#752)

63.8 v0.22.0(30/3/2022)

63.8.1 Highlights

- Support a series of CSP Network, such as CSP-ResNet, CSP-ResNeXt and CSP-DarkNet.
- A new `CustomDataset` class to help you build dataset of yourself!
- Support ConvMixer, RepMLP and new dataset - CUB dataset.

63.8.2 New Features

- [Feature] Add CSPNet and backbone and checkpoints (#735)
- [Feature] Add `CustomDataset`. (#738)
- [Feature] Add diff seeds to diff ranks. (#744)
- [Feature] Support ConvMixer. (#716)
- [Feature] Our `dist_train` & `dist_test` tools support distributed training on multiple machines. (#734)
- [Feature] Add RepMLP backbone and checkpoints. (#709)
- [Feature] Support CUB dataset. (#703)
- [Feature] Support ResizeMix. (#676)

63.8.3 Improvements

- [Enhance] Use `--a-b` instead of `--a_b` in arguments. (#754)
- [Enhance] Add `get_cat_ids` and `get_gt_labels` to KFoldDataset. (#721)
- [Enhance] Set torch seed in `worker_init_fn`. (#733)

63.8.4 Bug Fixes

- [Fix] Fix the discontiguous output feature map of ConvNeXt. (#743)

63.8.5 Docs Update

- [Docs] Add brief installation steps in README for copy&paste. (#755)
- [Docs] fix logo url link from mmocr to mmcls. (#732)

63.9 v0.21.0(04/03/2022)

63.9.1 Highlights

- Support ResNetV1c and Wide-ResNet, and provide pre-trained models.
- Support dynamic input shape for ViT-based algorithms. Now our ViT, DeiT, Swin-Transformer and T2T-ViT support forwarding with any input shape.
- Reproduce training results of DeiT. And our DeiT-T and DeiT-S have higher accuracy comparing with the official weights.

63.9.2 New Features

- Add ResNetV1c. (#692)
- Support Wide-ResNet. (#715)
- Support gem pooling (#677)

63.9.3 Improvements

- Reproduce training results of DeiT. (#711)
- Add ConvNeXt pretrain models on ImageNet-1k. (#707)
- Support dynamic input shape for ViT-based algorithms. (#706)
- Add `evaluate` function for ConcatDataset. (#650)
- Enhance vis-pipeline tool. (#604)
- Return code 1 if scripts runs failed. (#694)
- Use PyTorch official `one_hot` to implement `convert_to_one_hot`. (#696)
- Add a new pre-commit-hook to automatically add a copyright. (#710)
- Add deprecation message for deploy tools. (#697)
- Upgrade isort pre-commit hooks. (#687)
- Use `--gpu-id` instead of `--gpu-ids` in non-distributed multi-gpu training/testing. (#688)
- Remove deprecation. (#633)

63.9.4 Bug Fixes

- Fix Conformer forward with irregular input size. (#686)
- Add `dist.barrier` to fix a bug in directory checking. (#666)

63.10 v0.20.1(07/02/2022)

63.10.1 Bug Fixes

- Fix the MMCV dependency version.

63.11 v0.20.0(30/01/2022)

63.11.1 Highlights

- Support K-fold cross-validation. The tutorial will be released later.
- Support HRNet, ConvNeXt, Twins and EfficientNet.
- Support model conversion from PyTorch to Core-ML by a tool.

63.11.2 New Features

- Support K-fold cross-validation. (#563)
- Support HRNet and add pre-trained models. (#660)
- Support ConvNeXt and add pre-trained models. (#670)
- Support Twins and add pre-trained models. (#642)
- Support EfficientNet and add pre-trained models.(#649)
- Support `features_only` option in `TIMMBackbone`. (#668)
- Add conversion script from pytorch to Core-ML model. (#597)

63.11.3 Improvements

- New-style CPU training and inference. (#674)
- Add setup multi-processing both in train and test. (#671)
- Rewrite channel split operation in ShufflenetV2. (#632)
- Deprecate the support for “python setup.py test”. (#646)
- Support single-label, softmax, custom eps by asymmetric loss. (#609)
- Save class names in best checkpoint created by evaluation hook. (#641)

63.11.4 Bug Fixes

- Fix potential unexcepted behaviors if `metric_options` is not specified in multi-label evaluation. (#647)
- Fix API changes in `pytorch-grad-cam&gt;=1.3.7`. (#656)
- Fix bug which breaks `cal_train_time` in `analyze_logs.py`. (#662)

63.11.5 Docs Update

- Update README in configs according to OpenMMLab standard. (#672)
- Update installation guide and README. (#624)

63.12 v0.19.0(31/12/2021)

63.12.1 Highlights

- The feature extraction function has been enhanced. See #593 for more details.
- Provide the high-acc ResNet-50 training settings from *ResNet strikes back*.
- Reproduce the training accuracy of T2T-ViT & RegNetX, and provide self-training checkpoints.
- Support DeiT & Conformer backbone and checkpoints.
- Provide a CAM visualization tool based on pytorch-grad-cam, and detailed user guide!

63.12.2 New Features

- Support Precise BN. (#401)
- Add CAM visualization tool. (#577)
- Repeated Aug and Sampler Registry. (#588)
- Add DeiT backbone and checkpoints. (#576)
- Support LAMB optimizer. (#591)
- Implement the conformer backbone. (#494)
- Add the frozen function for Swin Transformer model. (#574)
- Support using checkpoint in Swin Transformer to save memory. (#557)

63.12.3 Improvements

- [Reproduction] Reproduce RegNetX training accuracy. (#587)
- [Reproduction] Reproduce training results of T2T-ViT. (#610)
- [Enhance] Provide high-acc training settings of ResNet. (#572)
- [Enhance] Set a random seed when the user does not set a seed. (#554)
- [Enhance] Added `NumClassCheckHook` and unit tests. (#559)
- [Enhance] Enhance feature extraction function. (#593)

- [Enhance] Improve efficiency of precision, recall, f1_score and support. (#595)
- [Enhance] Improve accuracy calculation performance. (#592)
- [Refactor] Refactor `analysis_log.py`. (#529)
- [Refactor] Use new API of matplotlib to handle blocking input in visualization. (#568)
- [CI] Cancel previous runs that are not completed. (#583)
- [CI] Skip build CI if only configs or docs modification. (#575)

63.12.4 Bug Fixes

- Fix test sampler bug. (#611)
- Try to create a symbolic link, otherwise copy. (#580)
- Fix a bug for multiple output in swin transformer. (#571)

63.12.5 Docs Update

- Update mmcv, torch, cuda version in Dockerfile and docs. (#594)
- Add analysis&misc docs. (#525)
- Fix docs build dependency. (#584)

63.13 v0.18.0(30/11/2021)

63.13.1 Highlights

- Support MLP-Mixer backbone and provide pre-trained checkpoints.
- Add a tool to visualize the learning rate curve of the training phase. Welcome to use with the tutorial!

63.13.2 New Features

- Add MLP Mixer Backbone. (#528, #539)
- Support positive weights in BCE. (#516)
- Add a tool to visualize learning rate in each iterations. (#498)

63.13.3 Improvements

- Use CircleCI to do unit tests. (#567)
- Focal loss for single label tasks. (#548)
- Remove useless `import_modules_from_string`. (#544)
- Rename config files according to the config name standard. (#508)
- Use `reset_classifier` to remove head of timm backbones. (#534)
- Support passing arguments to loss from head. (#523)

- Refactor `Resize` transform and add `Pad` transform. (#506)
- Update mmcv dependency version. (#509)

63.13.4 Bug Fixes

- Fix bug when using `ClassBalancedDataset`. (#555)
- Fix a bug when using iter-based runner with 'val' workflow. (#542)
- Fix interpolation method checking in `Resize`. (#547)
- Fix a bug when load checkpoints in mulit-GPUs environment. (#527)
- Fix an error on indexing scalar metrics in `analyze_result.py`. (#518)
- Fix wrong condition judgment in `analyze_logs.py` and prevent empty curve. (#510)

63.13.5 Docs Update

- Fix vit config and model broken links. (#564)
- Add abstract and image for every paper. (#546)
- Add mmflow and mim in banner and readme. (#543)
- Add schedule and runtime tutorial docs. (#499)
- Add the top-5 acc in ResNet-CIFAR README. (#531)
- Fix TOC of `visualization.md` and add example images. (#513)
- Use docs link of other projects and add MMCV docs. (#511)

63.14 v0.17.0(29/10/2021)

63.14.1 Highlights

- Support Tokens-to-Token ViT backbone and Res2Net backbone. Welcome to use!
- Support ImageNet21k dataset.
- Add a pipeline visualization tool. Try it with the tutorials!

63.14.2 New Features

- Add Tokens-to-Token ViT backbone and converted checkpoints. (#467)
- Add Res2Net backbone and converted weights. (#465)
- Support ImageNet21k dataset. (#461)
- Support seesaw loss. (#500)
- Add a pipeline visualization tool. (#406)
- Add a tool to find broken files. (#482)
- Add a tool to test TorchServe. (#468)

63.14.3 Improvements

- Refator Vision Transformer. (#395)
- Use context manager to reuse matplotlib figures. (#432)

63.14.4 Bug Fixes

- Remove `DistSamplerSeedHook` if use `IterBasedRunner`. (#501)
- Set the priority of `EvalHook` to "LOW" to avoid a bug when using `IterBasedRunner`. (#488)
- Fix a wrong parameter of `get_root_logger` in `apis/train.py`. (#486)
- Fix version check in dataset builder. (#474)

63.14.5 Docs Update

- Add English Colab tutorials and update Chinese Colab tutorials. (#483, #497)
- Add tutuorial for config files. (#487)
- Add model-pages in Model Zoo. (#480)
- Add code-spell pre-commit hook and fix a large mount of typos. (#470)

63.15 v0.16.0(30/9/2021)

63.15.1 Highlights

- We have improved compatibility with downstream repositories like MMDetection and MMSegmentation. We will add some examples about how to use our backbones in MMDetection.
- Add RepVGG backbone and checkpoints. Welcome to use it!
- Add timm backbones wrapper, now you can simply use backbones of pytorch-image-models in MMClassification!

63.15.2 New Features

- Add RepVGG backbone and checkpoints. (#414)
- Add timm backbones wrapper. (#427)

63.15.3 Improvements

- Fix TnT compatibility and verbose warning. (#436)
- Support setting `--out-items` in `tools/test.py`. (#437)
- Add datetime info and saving model using torch<1.6 format. (#439)
- Improve downstream repositories compatibility. (#421)
- Rename the option `--options` to `--cfg-options` in some tools. (#425)
- Add PyTorch 1.9 and Python 3.9 build workflow, and remove some CI. (#422)

63.15.4 Bug Fixes

- Fix format error in `test.py` when metric returns `np.ndarray`. (#441)
- Fix `publish_model` bug if no parent of `out_file`. (#463)
- Fix num_classes bug in pytorch2onnx.py. (#458)
- Fix missing runtime requirement `packaging`. (#459)
- Fix saving simplified model bug in ONNX export tool. (#438)

63.15.5 Docs Update

- Update `getting_started.md` and `install.md`. And rewrite `finetune.md`. (#466)
- Use PyTorch style docs theme. (#457)
- Update metafile and Readme. (#435)
- Add `CITATION.cff`. (#428)

63.16 v0.15.0(31/8/2021)

63.16.1 Highlights

- Support `hparams` argument in `AutoAugment` and `RandAugment` to provide hyperparameters for sub-policies.
- Support custom squeeze channels in `SELayer`.
- Support classwise weight in losses.

63.16.2 New Features

- Add `hparams` argument in `AutoAugment` and `RandAugment` and some other improvement. (#398)
- Support classwise weight in losses. (#388)
- Enhance `SELayer` to support custom squeeze channels. (#417)

63.16.3 Code Refactor

- Better result visualization. (#419)
- Use `post_process` function to handle pred result processing. (#390)
- Update `digit_version` function. (#402)
- Avoid albumentations to install both opencv and opencv-headless. (#397)
- Avoid unnecessary listdir when building ImageNet. (#396)
- Use dynamic mmcv download link in TorchServe dockerfile. (#387)

63.16.4 Docs Improvement

- Add readme of some algorithms and update meta yml. (#418)
- Add Copyright information. (#413)
- Fix typo 'metirc'. (#411)
- Update QQ group QR code. (#393)
- Add PR template and modify issue template. (#380)

63.17 v0.14.0(4/8/2021)

63.17.1 Highlights

- Add transformer-in-transformer backbone and pretrain checkpoints, refers to the paper.
- Add Chinese colab tutorial.
- Provide dockerfile to build mmcls dev docker image.

63.17.2 New Features

- Add transformer in transformer backbone and pretrain checkpoints. (#339)
- Support mim, welcome to use mim to manage your mmcls project. (#376)
- Add Dockerfile. (#365)
- Add ResNeSt configs. (#332)

63.17.3 Improvements

- Use the `presistent_works` option if available, to accelerate training. (#349)
- Add Chinese ipynb tutorial. (#306)
- Refactor unit tests. (#321)
- Support to test mmdet inference with mmcls backbone. (#343)
- Use zero as default value of `thrs` in metrics. (#341)

63.17.4 Bug Fixes

- Fix ImageNet dataset annotation file parse bug. (#370)
- Fix docstring typo and init bug in ShuffleNetV1. (#374)
- Use local ATTENTION registry to avoid conflict with other repositories. (#376)
- Fix swin transformer config bug. (#355)
- Fix `patch_cfg` argument bug in SwinTransformer. (#368)
- Fix duplicate `init_weights` call in ViT init function. (#373)
- Fix broken `_base_` link in a resnet config. (#361)
- Fix vgg-19 model link missing. (#363)

63.18 v0.13.0(3/7/2021)

- Support Swin-Transformer backbone and add training configs for Swin-Transformer on ImageNet.

63.18.1 New Features

- Support Swin-Transformer backbone and add training configs for Swin-Transformer on ImageNet. (#271)
- Add pretained model of RegNetX. (#269)
- Support adding custom hooks in config file. (#305)
- Improve and add Chinese translation of `CONTRIBUTING.md` and all tools tutorials. (#320)
- Dump config before training. (#282)
- Add torchscript and torchserve deployment tools. (#279, #284)

63.18.2 Improvements

- Improve test tools and add some new tools. (#322)
- Correct MobilenetV3 backbone structure and add pretained models. (#291)
- Refactor `PatchEmbed` and `HybridEmbed` as independent components. (#330)
- Refactor mixup and cutmix as `Augments` to support more functions. (#278)
- Refactor weights initialization method. (#270, #318, #319)
- Refactor `LabelSmoothLoss` to support multiple calculation formulas. (#285)

63.18.3 Bug Fixes

- Fix bug for CPU training. (#286)
- Fix missing test data when `num_imgs` can not be evenly divided by `num_gpus`. (#299)
- Fix build compatible with pytorch v1.3-1.5. (#301)
- Fix `magnitude_std` bug in `RandAugment`. (#309)
- Fix bug when `samples_per_gpu` is 1. (#311)

63.19 v0.12.0(3/6/2021)

- Finish adding Chinese tutorials and build Chinese documentation on readthedocs.
- Update ResNeXt checkpoints and ResNet checkpoints on CIFAR.

63.19.1 New Features

- Improve and add Chinese translation of `data_pipeline.md` and `new_modules.md`. (#265)
- Build Chinese translation on readthedocs. (#267)
- Add an argument efficientnet_style to `RandomResizedCrop` and `CenterCrop`. (#268)

63.19.2 Improvements

- Only allow directory operation when rank==0 when testing. (#258)
- Fix typo in `base_head`. (#274)
- Update ResNeXt checkpoints. (#283)

63.19.3 Bug Fixes

- Add attribute `data.test` in MNIST configs. (#264)
- Download CIFAR/MNIST dataset only on rank 0. (#273)
- Fix MMCV version compatibility. (#276)
- Fix CIFAR color channels bug and update checkpoints in model zoo. (#280)

63.20 v0.11.1(21/5/2021)

- Refine `new_dataset.md` and add Chinese translation of `finture.md`, `new_dataset.md`.

63.20.1 New Features

- Add `dim` argument for `GlobalAveragePooling`. (#236)
- Add random noise to `RandAugment` magnitude. (#240)
- Refine `new_dataset.md` and add Chinese translation of `finture.md`, `new_dataset.md`. (#243)

63.20.2 Improvements

- Refactor arguments passing for Heads. (#239)
- Allow more flexible `magnitude_range` in `RandAugment`. (#249)
- Inherits MMCV registry so that in the future OpenMMLab repos like MMDet and MMSeg could directly use the backbones supported in MMCls. (#252)

63.20.3 Bug Fixes

- Fix typo in `analyze_results.py`. (#237)
- Fix typo in unittests. (#238)
- Check if specified tmpdir exists when testing to avoid deleting existing data. (#242 & #258)
- Add missing config files in `MANIFEST.in`. (#250 & #255)
- Use temporary directory under shared directory to collect results to avoid unavailability of temporary directory for multi-node testing. (#251)

63.21 v0.11.0(1/5/2021)

- Support cutmix trick.
- Support random augmentation.
- Add `tools/deployment/test.py` as a ONNX runtime test tool.
- Support ViT backbone and add training configs for ViT on ImageNet.
- Add Chinese `README.md` and some Chinese tutorials.

63.21.1 New Features

- Support cutmix trick. (#198)
- Add `simplify` option in `pytorch2onnx.py`. (#200)
- Support random augmentation. (#201)
- Add config and checkpoint for training ResNet on CIFAR-100. (#208)
- Add `tools/deployment/test.py` as a ONNX runtime test tool. (#212)
- Support ViT backbone and add training configs for ViT on ImageNet. (#214)
- Add finetuning configs for ViT on ImageNet. (#217)
- Add `device` option to support training on CPU. (#219)

- Add Chinese `README.md` and some Chinese tutorials. (#221)
- Add `metafile.yml` in configs to support interaction with paper with code(PWC) and MMCLI. (#225)
- Upload configs and converted checkpoints for ViT fintuning on ImageNet. (#230)

63.21.2 Improvements

- Fix `LabelSmoothLoss` so that label smoothing and mixup could be enabled at the same time. (#203)
- Add `cal_acc` option in `ClsHead`. (#206)
- Check `CLASSES` in checkpoint to avoid unexpected key error. (#207)
- Check mmcv version when importing mmcls to ensure compatibility. (#209)
- Update `CONTRIBUTING.md` to align with that in MMCV. (#210)
- Change tags to html comments in configs README.md. (#226)
- Clean codes in ViT backbone. (#227)
- Reformat `pytorch2onnx.md` tutorial. (#229)
- Update `setup.py` to support MMCLI. (#232)

63.21.3 Bug Fixes

- Fix missing `cutmix_prob` in ViT configs. (#220)
- Fix backend for resize in ResNeXt configs. (#222)

63.22 v0.10.0(1/4/2021)

- Support AutoAugmentation
- Add tutorials for installation and usage.

63.22.1 New Features

- Add `Rotate` pipeline for data augmentation. (#167)
- Add `Invert` pipeline for data augmentation. (#168)
- Add `Color` pipeline for data augmentation. (#171)
- Add `Solarize` and `Posterize` pipeline for data augmentation. (#172)
- Support fp16 training. (#178)
- Add tutorials for installation and basic usage of MMClassification.(#176)
- Support `AutoAugmentation`, `AutoContrast`, `Equalize`, `Contrast`, `Brightness` and `Sharpness` pipelines for data augmentation. (#179)

63.22.2 Improvements

- Support dynamic shape export to onnx. (#175)
- Release training configs and update model zoo for fp16 (#184)
- Use MMCV's EvalHook in MMClassification (#182)

63.22.3 Bug Fixes

- Fix wrong naming in vgg config (#181)

63.23 v0.9.0(1/3/2021)

- Implement mixup trick.
- Add a new tool to create TensorRT engine from ONNX, run inference and verify outputs in Python.

63.23.1 New Features

- Implement mixup and provide configs of training ResNet50 using mixup. (#160)
- Add `Shear` pipeline for data augmentation. (#163)
- Add `Translate` pipeline for data augmentation. (#165)
- Add `tools/onnx2tensorrt.py` as a tool to create TensorRT engine from ONNX, run inference and verify outputs in Python. (#153)

63.23.2 Improvements

- Add `--eval-options` in `tools/test.py` to support eval options override, matching the behavior of other open-mmlab projects. (#158)
- Support showing and saving painted results in `mmcls.apis.test` and `tools/test.py`, matching the behavior of other open-mmlab projects. (#162)

63.23.3 Bug Fixes

- Fix configs for VGG, replace checkpoints converted from other repos with the ones trained by ourselves and upload the missing logs in the model zoo. (#161)

63.24 v0.8.0(31/1/2021)

- Support multi-label task.
- Support more flexible metrics settings.
- Fix bugs.

63.24.1 New Features

- Add evaluation metrics: mAP, CP, CR, CF1, OP, OR, OF1 for multi-label task. (#123)
- Add BCE loss for multi-label task. (#130)
- Add focal loss for multi-label task. (#131)
- Support PASCAL VOC 2007 dataset for multi-label task. (#134)
- Add asymmetric loss for multi-label task. (#132)
- Add analyze_results.py to select images for success/fail demonstration. (#142)
- Support new metric that calculates the total number of occurrences of each label. (#143)
- Support class-wise evaluation results. (#143)
- Add thresholds in eval_metrics. (#146)
- Add heads and a baseline config for multilabel task. (#145)

63.24.2 Improvements

- Remove the models with 0 checkpoint and ignore the repeated papers when counting papers to gain more accurate model statistics. (#135)
- Add tags in README.md. (#137)
- Fix optional issues in docstring. (#138)
- Update stat.py to classify papers. (#139)
- Fix mismatched columns in README.md. (#150)
- Fix test.py to support more evaluation metrics. (#155)

63.24.3 Bug Fixes

- Fix bug in VGG weight_init. (#140)
- Fix bug in 2 ResNet configs in which outdated heads were used. (#147)
- Fix bug of misordered height and width in `RandomCrop` and `RandomResizedCrop`. (#151)
- Fix missing `meta_keys` in `Collect`. (#149 & #152)

63.25 v0.7.0(31/12/2020)

- Add more evaluation metrics.
- Fix bugs.

63.25.1 New Features

- Remove installation of MMCV from requirements. (#90)
- Add 3 evaluation metrics: precision, recall and F-1 score. (#93)
- Allow config override during testing and inference with `--options`. (#91 & #96)

63.25.2 Improvements

- Use `build_runner` to make runners more flexible. (#54)
- Support to get category ids in `BaseDataset`. (#72)
- Allow `CLASSES` override during `BaseDateset` initialization. (#85)
- Allow input image as ndarray during inference. (#87)
- Optimize MNIST config. (#98)
- Add config links in model zoo documentation. (#99)
- Use functions from MMCV to collect environment. (#103)
- Refactor config files so that they are now categorized by methods. (#116)
- Add README in config directory. (#117)
- Add model statistics. (#119)
- Refactor documentation in consistency with other MM repositories. (#126)

63.25.3 Bug Fixes

- Add missing `CLASSES` argument to dataset wrappers. (#66)
- Fix slurm evaluation error during training. (#69)
- Resolve error caused by shape in `Accuracy`. (#104)
- Fix bug caused by extremely insufficient data in distributed sampler.(#108)
- Fix bug in `gpu_ids` in distributed training. (#107)
- Fix bug caused by extremely insufficient data in collect results during testing (#114)

63.26 v0.6.0(11/10/2020)

- Support new method: ResNeSt and VGG.
- Support new dataset: CIFAR10.
- Provide new tools to do model inference, model conversion from pytorch to onnx.

63.26.1 New Features

- Add model inference. (#16)
- Add pytorch2onnx. (#20)
- Add PIL backend for transform `Resize`. (#21)
- Add ResNeSt. (#25)
- Add VGG and its pretained models. (#27)
- Add CIFAR10 configs and models. (#38)
- Add albumentations transforms. (#45)
- Visualize results on image demo. (#58)

63.26.2 Improvements

- Replace urlretrieve with urlopen in dataset.utils. (#13)
- Resize image according to its short edge. (#22)
- Update ShuffleNet config. (#31)
- Update pre-trained models for shufflenet_v2, shufflenet_v1, se-resnet50, se-resnet101. (#33)

63.26.3 Bug Fixes

- Fix init_weights in `shufflenet_v2.py`. (#29)
- Fix the parameter `size` in test_pipeline. (#30)
- Fix the parameter in cosine lr schedule. (#32)
- Fix the convert tools for mobilenet_v2. (#34)
- Fix crash in CenterCrop transform when image is greyscale (#40)
- Fix outdated configs. (#53)

CHAPTER

SIXTYFOUR

COMPATIBILITY OF MMCLASSIFICATION 0.X

64.1 MMClassification 0.20.1

64.1.1 MMCV compatibility

In Twins backbone, we use the `PatchEmbed` module of MMCV, and this module is added after MMCV 1.4.2. Therefore, we need to update the mmcv version to 1.4.2.

CHAPTER

SIXTYFIVE

FREQUENTLY ASKED QUESTIONS

We list some common troubles faced by many users and their corresponding solutions here. Feel free to enrich the list if you find any frequent issues and have ways to help others to solve them. If the contents here do not cover your issue, please create an issue using the provided templates and make sure you fill in all required information in the template.

65.1 Installation

- Compatibility issue between MMCV and MMClassification; "AssertionError: MMCV==xxx is used but incompatible. Please install mmcv>=xxx, <=xxx."

  Compatible MMClassification and MMCV versions are shown as below. Please choose the correct version of MMCV to avoid installation issues.

MMClassification version	MMCV version
dev	mmcv>=1.7.0, <1.9.0
0.25.0 (master)	mmcv>=1.4.2, <1.9.0
0.24.1	mmcv>=1.4.2, <1.9.0
0.23.2	mmcv>=1.4.2, <1.7.0
0.22.1	mmcv>=1.4.2, <1.6.0
0.21.0	mmcv>=1.4.2, <=1.5.0
0.20.1	mmcv>=1.4.2, <=1.5.0
0.19.0	mmcv>=1.3.16, <=1.5.0
0.18.0	mmcv>=1.3.16, <=1.5.0
0.17.0	mmcv>=1.3.8, <=1.5.0
0.16.0	mmcv>=1.3.8, <=1.5.0
0.15.0	mmcv>=1.3.8, <=1.5.0
0.15.0	mmcv>=1.3.8, <=1.5.0
0.14.0	mmcv>=1.3.8, <=1.5.0
0.13.0	mmcv>=1.3.8, <=1.5.0
0.12.0	mmcv>=1.3.1, <=1.5.0
0.11.1	mmcv>=1.3.1, <=1.5.0
0.11.0	mmcv>=1.3.0
0.10.0	mmcv>=1.3.0
0.9.0	mmcv>=1.1.4
0.8.0	mmcv>=1.1.4
0.7.0	mmcv>=1.1.4
0.6.0	mmcv>=1.1.4

Note: Since the `dev` branch is under frequent development, the MMCV version dependency may be inaccurate.

If you encounter problems when using the `dev` branch, please try to update MMCV to the latest version.

- Using Albumentations

  If you would like to use `albumentations`, we suggest using `pip install -r requirements/albu.txt` or `pip install -U albumentations --no-binary qudida,albumentations`.

  If you simply use `pip install albumentations>=0.3.2`, it will install `opencv-python-headless` simultaneously (even though you have already installed `opencv-python`). Please refer to the official documentation for details.

65.2 Coding

- Do I need to reinstall mmcls after some code modifications?

  If you follow *the best practice* and install mmcls from source, any local modifications made to the code will take effect without reinstallation.

- How to develop with multiple MMClassification versions?

  Generally speaking, we recommend to use different virtual environments to manage MMClassification in different working directories. However, you can also use the same environment to develop MMClassification in different folders, like mmcls-0.21, mmcls-0.23. When you run the train or test shell script, it will adopt the mmcls package in the current folder. And when you run other Python script, you can also add ``PYTHONPATH=`pwd` `` at the beginning of your command to use the package in the current folder.

  Conversely, to use the default MMClassification installed in the environment rather than the one you are working with, you can remove the following line in those shell scripts:

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

CHAPTER

SIXTYSIX

NPU (HUAWEI ASCEND)

66.1 Usage

66.1.1 General Usage

Please install MMCV with NPU device support according to the tutorial.

Here we use 8 NPUs on your computer to train the model with the following command:

```
bash ./tools/dist_train.sh configs/resnet/resnet50_8xb32_in1k.py 8 --device npu
```

Also, you can use only one NPU to train the model with the following command:

```
python ./tools/train.py configs/resnet/resnet50_8xb32_in1k.py --device npu
```

66.1.2 High-performance Usage on ARM server

Since the scheduling ability of ARM CPUs when processing resource preemption is not as good as that of X86 CPUs during multi-card training, we provide a high-performance startup script to accelerate training with the following command:

```
# The script under the 8 cards of a single machine is shown here
bash tools/dist_train_arm.sh configs/resnet/resnet50_8xb32_in1k.py 8 --device npu --cfg-
↪options data.workers_per_gpu=$(($(nproc)/8))
```

For resnet50 8 NPUs training with batch_size(data.samples_per_gpu)=512, the performance data is shown below:

CPU	Start Script	IterTime(s)
ARM(Kunpeng920 *4)	./tools/dist_train.sh	~0.9(0.85-1.0)
ARM(Kunpeng920 *4)	./tools/dist_train_arm.sh	~0.8(0.78s-0.85)

66.2 Models Results

Model	Top-1 (%)	Top-5 (%)	Config	Download
ResNet-50	76.38	93.22	config	model | log
ResNetXt-32x4d-50	77.55	93.75	config	model | log
HRNet-W18	77.01	93.46	config	model | log
ResNetV1D-152	79.11	94.54	config	model | log
SE-ResNet-50	77.64	93.76	config	model | log
VGG-11	68.92	88.83	config	model | log
ShuffleNetV2 1.0x	69.53	88.82	config	model | log
MobileNetV2	71.758	90.394	config	model | log
MobileNetV3-Small	67.522	87.316	config	model | log
**CSPResNeXt50*	77.10	93.55	config	model | log
**EfficientNet-B4(AA + AdvProp)*	75.55	92.86	config	model | log
***DenseNet121*	72.62	91.04	config	model | log

Notes:

- If not specially marked, the results are almost same between results on the NPU and results on the GPU with FP32.
- (*) The training results of these models are lower than the results on the readme in the corresponding model, mainly because the results on the readme are directly the weight of the timm of the eval, and the results on this side are retrained according to the config with mmcls. The results of the config training on the GPU are consistent with the results of the NPU.
- (**) The accuracy of this model is slightly lower because config is a 4-card config, we use 8 cards to run, and users can adjust hyperparameters to get the best accuracy results.

All above models are provided by Huawei Ascend group.

CHAPTER

SIXTYSEVEN

INDICES AND TABLES

- genindex
- search