PaddleClas

Jun 18, 2020

Contents:

CHAPTER 1

tutorials

1.1 Installation

1.1.1 Introducation

This document introduces how to install PaddleClas and its requirements.

1.1.2 Install PaddlePaddle

Python 3.5, CUDA 9.0, CUDNN7.0 nccl2.1.2 and later version are required at first, For now, PaddleClas only support training on the GPU device. Please follow the instructions in the Installation if the PaddlePaddle on the device is lower than v1.7

Install PaddlePaddle

```
pip install paddlepaddle-gpu --upgrade
```

or compile from source code, please refer to Installation.

Verify Installation

```
import paddle.fluid as fluid
fluid.install_check.run_check()
```

Check PaddlePaddle version

```
python -c "import paddle; print(paddle.__version__)"
```

Note:

- Make sure the compiled version is later than v1.7
- Indicate **WITH_DISTRIBUTE=ON** when compiling, Please refer to Instruction for more details.

1.1.3 Install PaddleClas

**Clone PaddleClas: **

```
cd path_to_clone_PaddleClas
git clone https://github.com/PaddlePaddle/PaddleClas.git
```

Install requirements

```
pip install --upgrade -r requirements.txt
```

1.2 Trial in 30mins

Based on the flowers102 dataset, it takes only 30 mins to experience PaddleClas, include training varieties of backbone and pretrained model, SSLD distillation, and multiple data augmentation, Please refer to Installation to install at first.

1.2.1 Preparation

- enter insatallation dir

```
cd path_to_PaddleClas
```

- enter `dataset/flowers102`, download and decompress flowers102 dataset.

```
cd dataset/flowers102
wget https://www.robots.ox.ac.uk/~vgg/data/flowers/102/102flowers.tgz
wget https://www.robots.ox.ac.uk/~vgg/data/flowers/102/imagelabels.mat
wget https://www.robots.ox.ac.uk/~vgg/data/flowers/102/setid.mat
tar -xf 102flowers.tgz
```

- create train/val/test label files

```
python generate_flowers102_list.py jpg train > train_list.txt
python generate_flowers102_list.py jpg valid > val_list.txt
python generate_flowers102_list.py jpg test > extra_list.txt
cat train_list.txt extra_list.txt > train_extra_list.txt
```

Note: In order to offer more data to SSLD training task, train_list.txt and extra_list.txt will merge into train_extra_list.txft

- return `PaddleClas` dir

```
cd ../../
```

1.2.2 Environment

Set PYTHONPATH

```
export PYTHONPATH=./:$PYTHONPATH
```

Download pretrained model

```
python tools/download.py -a ResNet50_vd -p ./pretrained -d True
python tools/download.py -a ResNet50_vd_ssld -p ./pretrained -d True
python tools/download.py -a MobileNetV3_large_x1_0 -p ./pretrained -d True
```

Paramters

- `architecture`(shortname: a): model name.
- `path`(shortname: p) download path.
- `decompress`(shortname: d) whether to decompress.
- All experiments are running on the NVIDIA® Tesla® V100 sigle card.

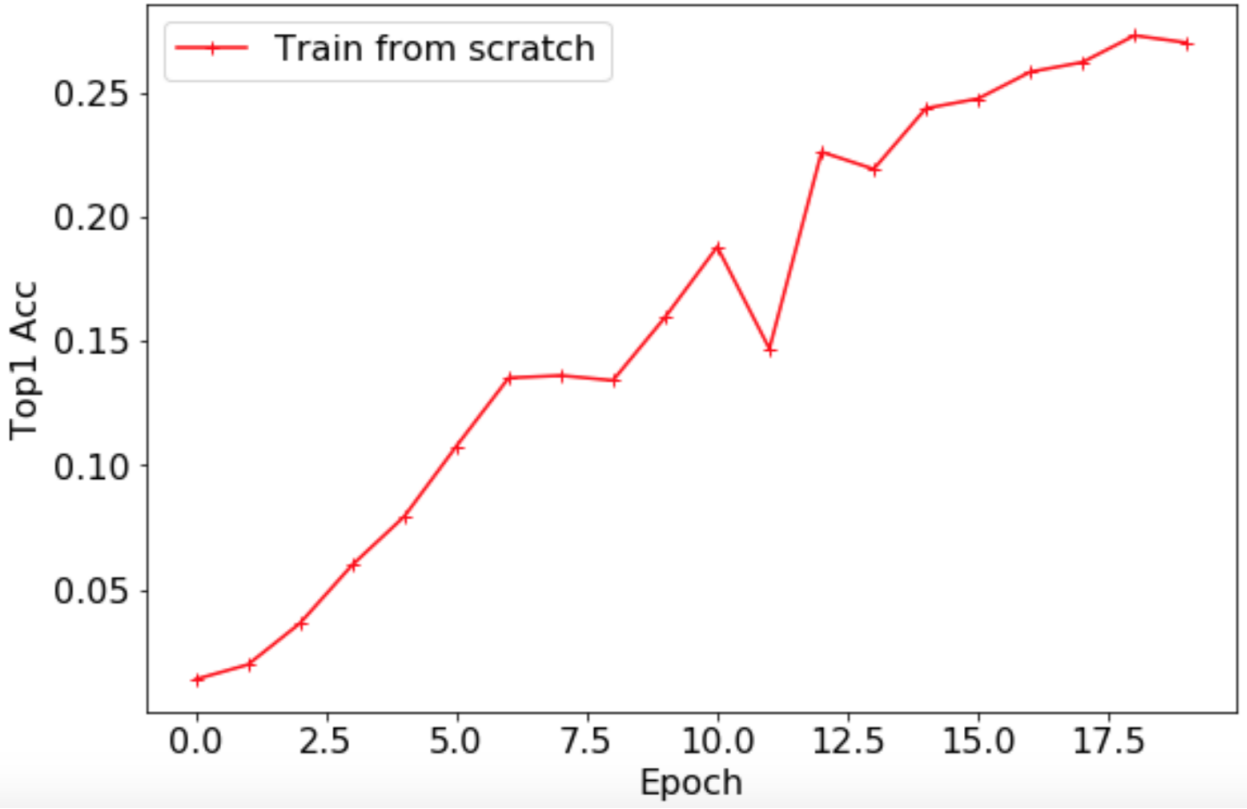
1.2.3 Training

Train from scratch

- Train ResNet50_vd

```
export CUDA_VISIBLE_DEVICES=0
python -m paddle.distributed.launch \
    --selected_gpus="0" \
    tools/train.py \
        -c ./configs/quick_start/ResNet50_vd.yaml
```

The validation `Top1 Acc` curve is showmn below.

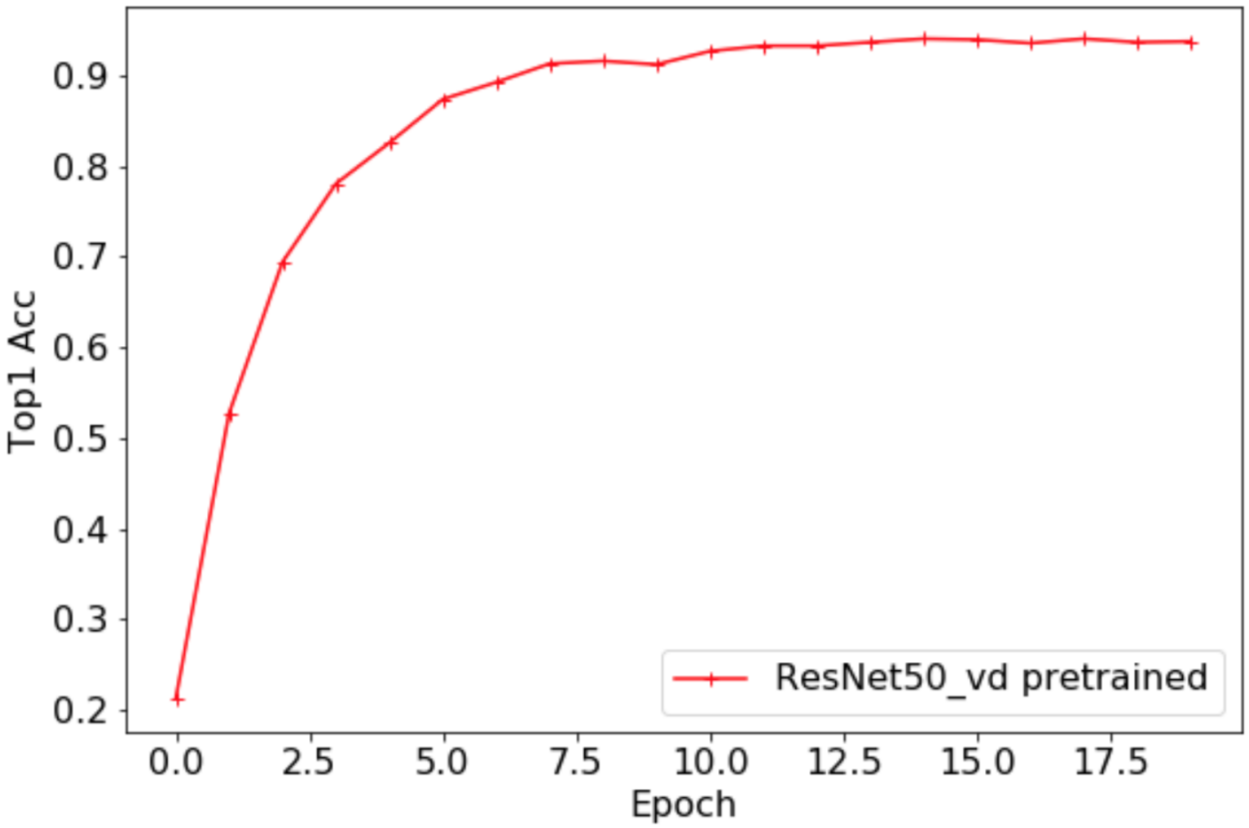


Finetune - ResNet50_vd pretrained model (Acc 79.12%)

- finetune ResNet50_vd_ model pretrained on the 1000-class Imagenet dataset

```
export CUDA_VISIBLE_DEVICES=0
python -m paddle.distributed.launch \
    --selected_gpus="0" \
    tools/train.py \
        -c ./configs/quick_start/ResNet50_vd_finetune.yaml
```

The validation `Top1 Acc` curve is shown below



Compare with training from scratch, it improve by 65% to 94.02%

SSLD finetune - ResNet50_vd_ssld pretrained model (Acc 82.39%)

Note: when finetuning model, which has been trained by SSLD, please use smaller learning rate in the middle of net.

```
ARCHITECTURE:
    name: 'ResNet50_vd'
    params:
        lr_mult_list: [0.1, 0.1, 0.2, 0.2, 0.3]
pretrained_model: "./pretrained/ResNet50_vd_ssld_pretrained"
```

Tringing script

```
export CUDA_VISIBLE_DEVICES=0
python -m paddle.distributed.launch \
```

(continues on next page)

(continued from previous page)

```
    --selected_gpus="0" \
    tools/train.py \
        -c ./configs/quick_start/ResNet50_vd_ssld_finetune.yaml
```

Compare with finetune on the 79.12% pretrained model, it improve by 0.9% to 95%.

More architecture - MobileNetV3

Training script

```
export CUDA_VISIBLE_DEVICES=0
python -m paddle.distributed.launch \
    --selected_gpus="0" \
    tools/train.py \
        -c ./configs/quick_start/MobileNetV3_large_x1_0_finetune.yaml
```

Compare with ResNet50_vd pretrained model, it decrease by 5% to 90%. Different architecture generates different performance, actually it is a task-oriented decision to apply the best performance model, should consider the inference time, storage, heterogeneous device, etc.

RandomErasing

Data augmentation works when training data is small.

Training script

```
export CUDA_VISIBLE_DEVICES=0
python -m paddle.distributed.launch \
    --selected_gpus="0" \
    tools/train.py \
        -c ./configs/quick_start/ResNet50_vd_ssld_random_erasing_finetune.yaml
```

It improves by 1.27% to 96.27%

- Save ResNet50_vd pretrained model to experience next chapter.

```
cp -r output/ResNet50_vd/19/  ./pretrained/flowers102_R50_vd_final/
```

Distillation

- Use extra_list.txt as unlabeled data, Note:
  - Samples in the `extra_list.txt` and `val_list.txt` don't have intersection
  - Because of in the source code, label information is unused, This is still unlabeled distillation
  - Teacher model use the pretrained_model trained on the flowers102 dataset, and student model use the MobileNetV3_large_x1_0 pretrained model(Acc 75.32%) trained on the ImageNet1K dataset

```
total_images: 7169
ARCHITECTURE:
    name: 'ResNet50_vd_distill_MobileNetV3_large_x1_0'
pretrained_model:
    - "./pretrained/flowers102_R50_vd_final/ppcls"
```

(continues on next page)

(continued from previous page)

```
    - "./pretrained/MobileNetV3_large_x1_0_pretrained/"
TRAIN:
    file_list: "./dataset/flowers102/train_extra_list.txt"
```

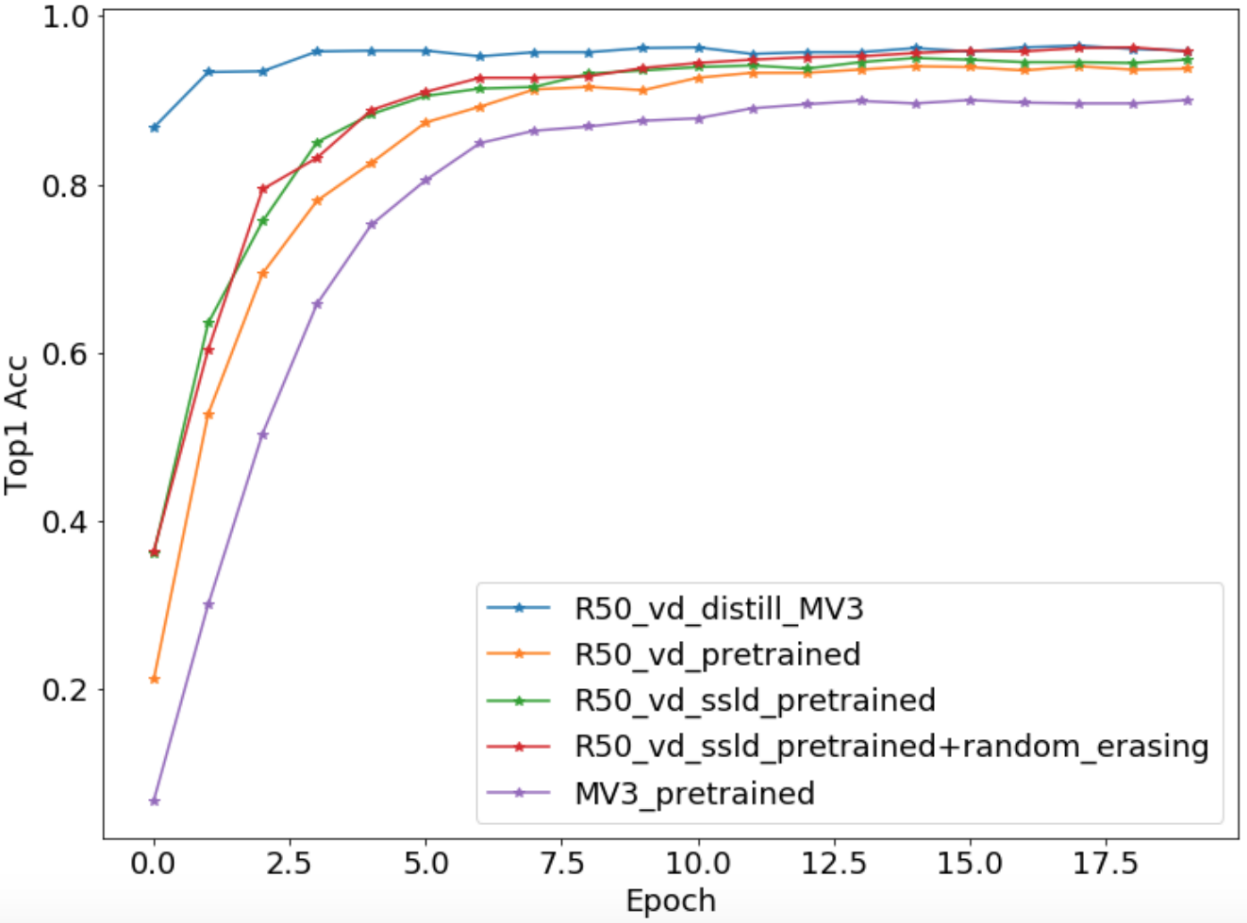
Final training script

```
export CUDA_VISIBLE_DEVICES=0
python -m paddle.distributed.launch \
    --selected_gpus="0" \
    tools/train.py \
        -c ./configs/quick_start/R50_vd_distill_MV3_large_x1_0.yaml
```

It significantly imporve by 6.47% to 96.47% with more unlabeled data and teacher model.

All accuracy

The whole accuracy curves are shown below



- **NOTE**: As flowers102 is a small dataset, validatation accuracy maybe float 1%.
- Please refer to Getting_started for more details

1.3 Data

1.3.1 Introduction

This document introduces the preparation of ImageNet1k and flowers102

1.3.2 Dataset

- Data format

Please follow the steps mentioned below to organize data, include train_list.txt and val_list.txt

```
# delimiter: "space"

ILSVRC2012_val_00000001.JPEG 65
...
```

ImageNet1k

After downloading data, please organize the data dir as below

```
PaddleClas/dataset/imagenet/
|_ train/
|  |_ n01440764
|  |  |_ n01440764_10026.JPEG
|  |  |_ ...
|  |_ ...
|  |
|  |_ n15075141
|     |_ ...
|     |_ n15075141_9993.JPEG
|_ val/
|  |_ ILSVRC2012_val_00000001.JPEG
|  |_ ...
|  |_ ILSVRC2012_val_00050000.JPEG
|_ train_list.txt
|_ val_list.txt
```

Flowers102 Dataset

Download Data then decompress:

```
jpg/
setid.mat
imagelabels.mat
```

Please put all the files under `PaddleClas/dataset/flowers102`

generate generate_flowers102_list.py and train_list.txtval_list.txt

```
python generate_flowers102_list.py jpg train > train_list.txt
python generate_flowers102_list.py jpg valid > val_list.txt
```

Please organize data dir as below

```
PaddleClas/dataset/flowers102/
|_ jpg/
|  |_ image_03601.jpg
|  |_ ...
|  |_ image_02355.jpg
|_ train_list.txt
|_ val_list.txt
```

1.4 Getting Started

Please refer to Installation to setup environment at first, and prepare ImageNet1K data by following the instruction mentioned in the data

1.4.1 Setup

Setup PYTHONPATH

```
export PYTHONPATH=path_to_PaddleClas:$PYTHONPATH
```

1.4.2 Training and validating

PaddleClas support `tools/train.py` and `tools/eval.py` to start training and validating.

Training

```
# PaddleClas use paddle.distributed.launch to start multi-cards and multiprocess␣
↪training.
# Set FLAGS_selected_gpus to indicate GPU cards

python -m paddle.distributed.launch \
    --selected_gpus="0,1,2,3" \
    tools/train.py \
        -c ./configs/ResNet/ResNet50_vd.yaml
```

- log:

```
epoch:0    train    step:13    loss:7.9561    top1:0.0156    top5:0.1094    lr:0.
↪100000    elapse:0.193
```

add -o params to update configuration

```
python -m paddle.distributed.launch \
    --selected_gpus="0,1,2,3" \
    tools/train.py \
        -c ./configs/ResNet/ResNet50_vd.yaml \
        -o use_mix=1 \
    --vdl_dir=./scalar/
```

- log:

```
epoch:0    train    step:522    loss:1.6330    lr:0.100000    elapse:0.210
```

or modify configuration directly to config fileds, please refer to config for more details.

use visuldl to visulize training loss in the real time

```
visualdl --logdir ./scalar --host <host_IP> --port <port_num>
```

finetune

- please refer to Trial for more details.

validation

```
python tools/eval.py \
    -c ./configs/eval.yaml \
    -o ARCHITECTURE.name="ResNet50_vd" \
    -o pretrained_model=path_to_pretrained_models

modify `configs/eval.yaml filed: `ARCHITECTURE.name` and filed: `pretrained_model` to␣
↪config valid model or add -o params to update config directly.


**NOTE: ** when loading the pretrained model, should ignore the suffix ```.pdparams```

## Predict

PaddlePaddle supprot three predict interfaces
Use predicator interface to predict
First, export inference model

```bash
python tools/export_model.py \
    --model=model_name \
    --pretrained_model=pretrained_model_dir \
    --output_path=save_inference_dir
```

Second, start predicator enginee

```
python tools/infer/predict.py \
    -m model_path \
    -p params_path \
    -i image path \
    --use_gpu=1 \
    --use_tensorrt=True
```
```

please refer to inference for more details.

#Configuration

1.5 Introduction

This document introduces the configuration(filed in `config/*.yaml`) of PaddleClas.

1.5.1 Basic

1.5.2 Optimizer & Learning rate

learning rate

optimizer

1.5.3 reader

processing

mix preprocessing

CHAPTER 2

models

2.1 Model Library Overview

2.1.1 Overview

Based on the ImageNet1k classification dataset, the 23 classification network structures supported by PaddleClas and the corresponding 117 image classification pretrained models are shown below. Training trick, a brief introduction to each series of network structures, and performance evaluation will be shown in the corresponding chapters.

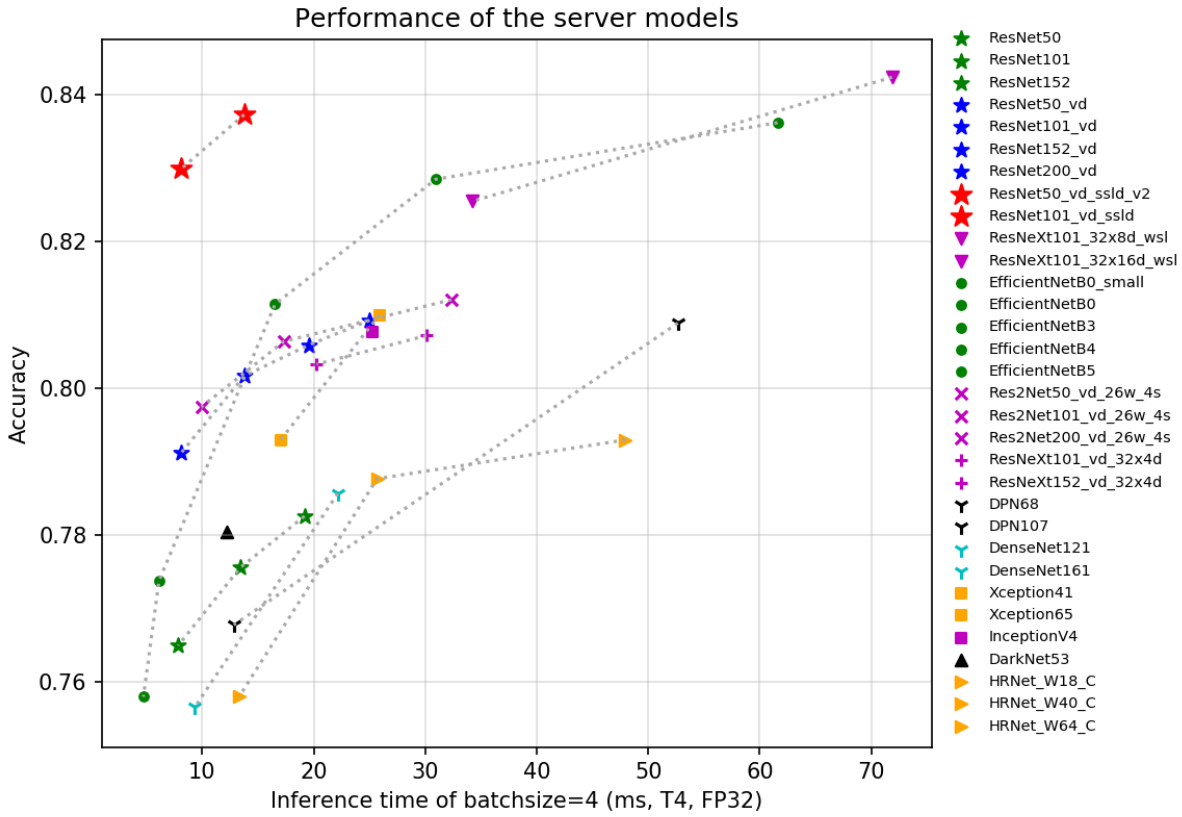
2.1.2 Evaluation environment

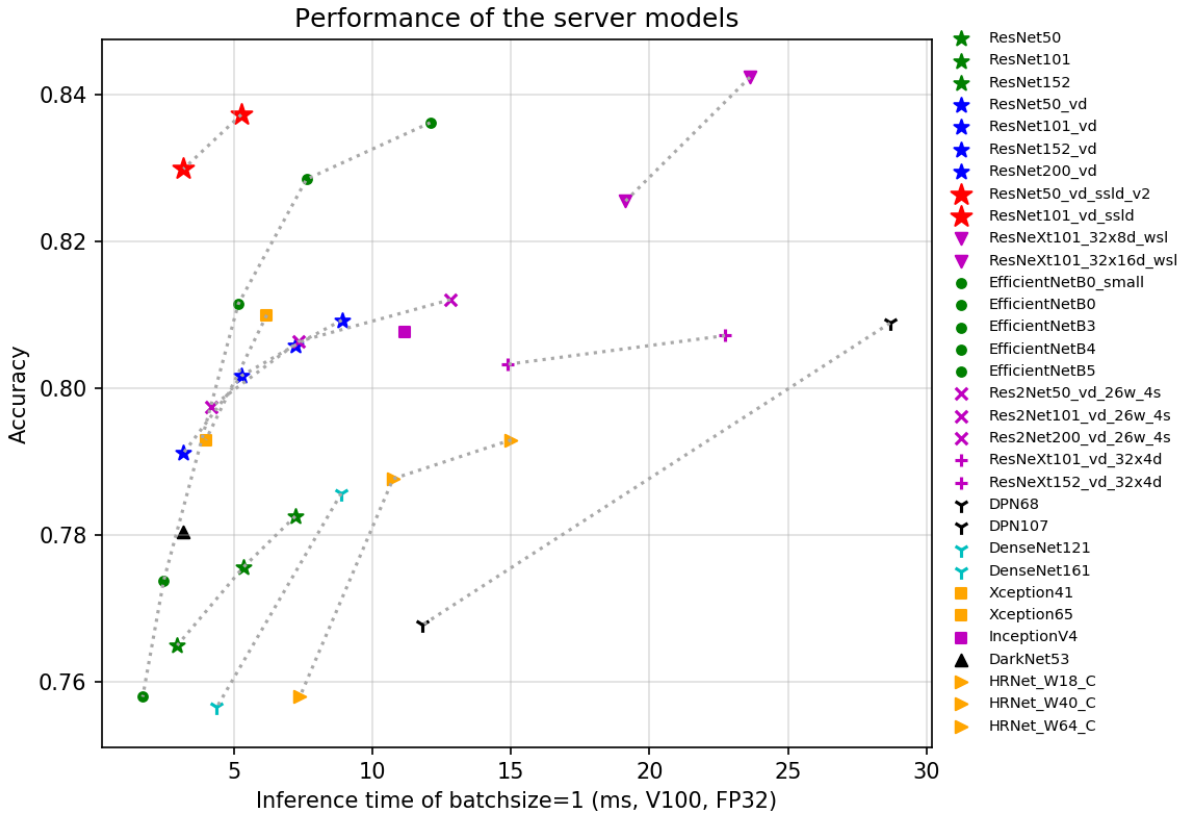
- CPU evaluation environment is based on Snapdragon 855 (SD855).
- The GPU evaluation environment is based on V100 and TensorRT, and the evaluation script is as follows.

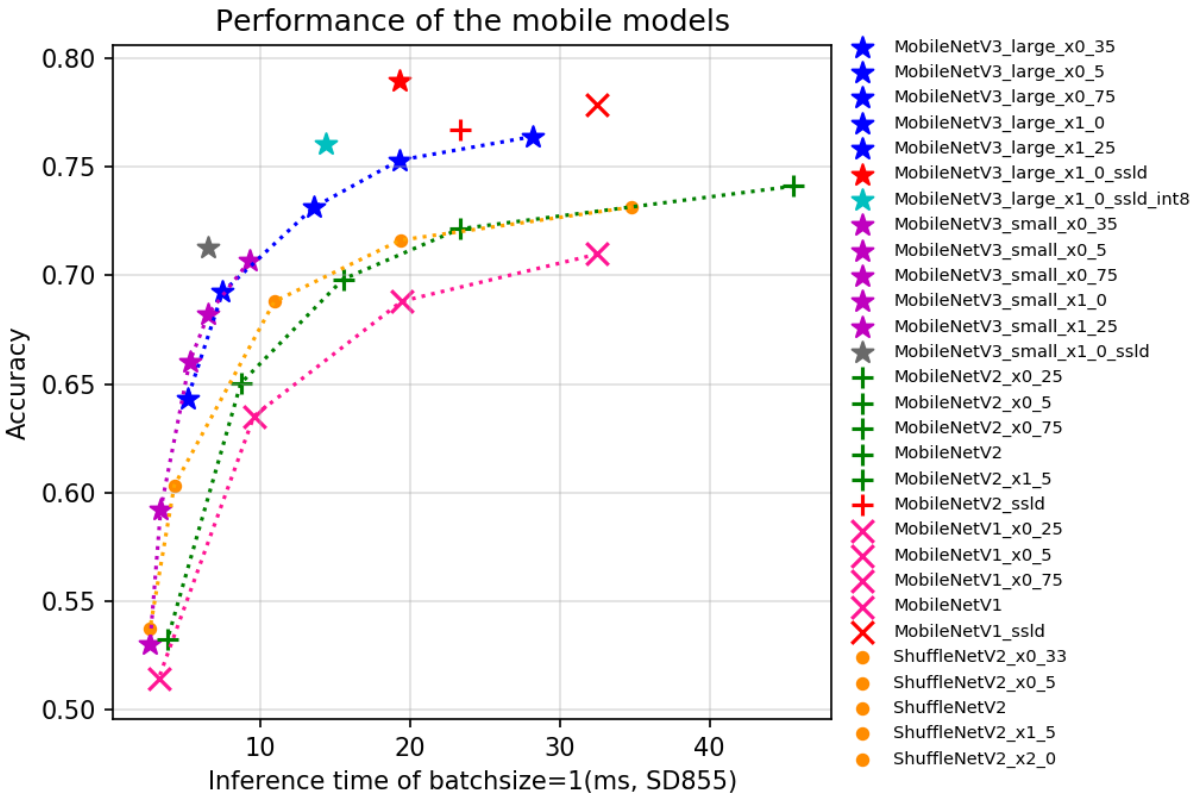
```
#!/usr/bin/env bash

export PYTHONPATH=$PWD:$PYTHONPATH

python tools/infer/predict.py \
    --model_file='pretrained/infer/model' \
    --params_file='pretrained/infer/params' \
    --enable_benchmark=True \
    --model_name=ResNet50_vd \
    --use_tensorrt=True \
    --use_fp16=False \
    --batch_size=1
```


Performance of the server models
Accuracy
0.84
0.82
0.80
0.78
0.76
10
20
30
40
50
60
70
Inference time of batchsize=4 (ms, T4, FP32)
ResNet50
ResNet101
ResNet152
ResNet50_vd
ResNet101_vd
ResNet152_vd
ResNet200_vd
ResNet50_vd_ssld_v2
ResNet101_vd_ssld
ResNeXt101_32x8d_wsl
ResNeXt101_32x16d_wsl
EfficientNetB0_small
EfficientNetB0
EfficientNetB3
EfficientNetB4
EfficientNetB5
Res2Net50_vd_26w_4s
Res2Net101_vd_26w_4s
Res2Net200_vd_26w_4s
ResNeXt101_vd_32x4d
ResNeXt152_vd_32x4d
DPN68
DPN107
DenseNet121
DenseNet161
Xception41
Xception65
InceptionV4
DarkNet53
HRNet_W18_C
HRNet_W40_C
HRNet_W64_C


Performance of the server models
Accuracy
0.84
0.82
0.80
0.78
0.76
5
10
15
20
25
30
Inference time of batchsize=1 (ms, V100, FP32)
ResNet50
ResNet101
ResNet152
ResNet50_vd
ResNet101_vd
ResNet152_vd
ResNet200_vd
ResNet50_vd_ssld_v2
ResNet101_vd_ssld
ResNeXt101_32x8d_wsl
ResNeXt101_32x16d_wsl
EfficientNetB0_small
EfficientNetB0
EfficientNetB3
EfficientNetB4
EfficientNetB5
Res2Net50_vd_26w_4s
Res2Net101_vd_26w_4s
Res2Net200_vd_26w_4s
ResNeXt101_vd_32x4d
ResNeXt152_vd_32x4d
DPN68
DPN107
DenseNet121
DenseNet161
Xception41
Xception65
InceptionV4
DarkNet53
HRNet_W18_C
HRNet_W40_C
HRNet_W64_C



If you think this document is helpful to you, welcome to give a star to our project:https://github.com/PaddlePaddle/PaddleClas

2.1.3 Pretrained model list and download address

- ResNet and ResNet_vd series
  - ResNet series[*1*](paper link)
    - ResNet18
    - ResNet34
    - ResNet50
    - ResNet101
    - ResNet152
  - ResNet_vcResNet_vd series[*2*](paper link)
    - ResNet50_vc
    - ResNet18_vd
    - ResNet34_vd
    - ResNet50_vd
    - ResNet50_vd_v2

        * ResNet101_vd
        * ResNet152_vd
        * ResNet200_vd
        * ResNet50_vd_ssld
        * ResNet50_vd_ssld_v2
        * Fix_ResNet50_vd_ssld_v2
        * ResNet101_vd_ssld
- Mobile and Embedded Vision Applications Network series
    - MobileNetV3 series[*3*](paper link)
        * MobileNetV3_large_x0_35
        * MobileNetV3_large_x0_5
        * MobileNetV3_large_x0_75
        * MobileNetV3_large_x1_0
        * MobileNetV3_large_x1_25
        * MobileNetV3_small_x0_35
        * MobileNetV3_small_x0_5
        * MobileNetV3_small_x0_75
        * MobileNetV3_small_x1_0
        * MobileNetV3_small_x1_25
        * MobileNetV3_large_x1_0_ssld
        * MobileNetV3_large_x1_0_ssld_int8
        * MobileNetV3_small_x1_0_ssld
    - MobileNetV2 series[*4*](paper link)
        * MobileNetV2_x0_25
        * MobileNetV2_x0_5
        * MobileNetV2_x0_75
        * MobileNetV2
        * MobileNetV2_x1_5
        * MobileNetV2_x2_0
        * MobileNetV2_ssld
    - MobileNetV1 series[*5*](paper link)
        * MobileNetV1_x0_25
        * MobileNetV1_x0_5
        * MobileNetV1_x0_75
        * MobileNetV1
        * MobileNetV1_ssld

  - ShuffleNetV2 series[*6*](paper link)
    - ShuffleNetV2_x0_25
    - ShuffleNetV2_x0_33
    - ShuffleNetV2_x0_5
    - ShuffleNetV2
    - ShuffleNetV2_x1_5
    - ShuffleNetV2_x2_0
    - ShuffleNetV2_swish
- SEResNeXt and Res2Net series
  - ResNeXt series[*7*](paper link)
    - ResNeXt50_32x4d
    - ResNeXt50_64x4d
    - ResNeXt101_32x4d
    - ResNeXt101_64x4d
    - ResNeXt152_32x4d
    - ResNeXt152_64x4d
  - ResNeXt_vd series
    - ResNeXt50_vd_32x4d
    - ResNeXt50_vd_64x4d
    - ResNeXt101_vd_32x4d
    - ResNeXt101_vd_64x4d
    - ResNeXt152_vd_32x4d
    - ResNeXt152_vd_64x4d
  - SE_ResNet_vd series[*8*](paper link)
    - SE_ResNet18_vd
    - SE_ResNet34_vd
    - SE_ResNet50_vd
  - SE_ResNeXt series
    - SE_ResNeXt50_32x4d
    - SE_ResNeXt101_32x4d
  - SE_ResNeXt_vd series
    - SE_ResNeXt50_vd_32x4d
    - SENet154_vd
  - Res2Net series[*9*](paper link)
    - Res2Net50_26w_4s
    - Res2Net50_vd_26w_4s

        * Res2Net50_14w_8s
        * Res2Net101_vd_26w_4s
        * Res2Net200_vd_26w_4s
- Inception series
    - GoogLeNet series[*10*](paper link)
        * GoogLeNet
    - Inception series[*11*](paper link)
        * InceptionV4
    - Xception series[*12*](paper link)
        * Xception41
        * Xception41_deeplab
        * Xception65
        * Xception65_deeplab
        * Xception71
- HRNet series
    - HRNet series[*13*](paper link)
        * HRNet_W18_C
        * HRNet_W30_C
        * HRNet_W32_C
        * HRNet_W40_C
        * HRNet_W44_C
        * HRNet_W48_C
        * HRNet_W64_C
- DPN and DenseNet series
    - DPN series[*14*](paper link)
        * DPN68
        * DPN92
        * DPN98
        * DPN107
        * DPN131
    - DenseNet series[*15*](paper link)
        * DenseNet121
        * DenseNet161
        * DenseNet169
        * DenseNet201
        * DenseNet264

- EfficientNet and ResNeXt101_wsl series
  - EfficientNet series[*16*](paper link)
    - EfficientNetB0_small
    - EfficientNetB0
    - EfficientNetB1
    - EfficientNetB2
    - EfficientNetB3
    - EfficientNetB4
    - EfficientNetB5
    - EfficientNetB6
    - EfficientNetB7
  - ResNeXt101_wsl series[*17*](paper link)
    - ResNeXt101_32x8d_wsl
    - ResNeXt101_32x16d_wsl
    - ResNeXt101_32x32d_wsl
    - ResNeXt101_32x48d_wsl
    - Fix_ResNeXt101_32x48d_wsl
- Other models
  - AlexNet series[*18*](paper link)
    - AlexNet
  - SqueezeNet series[*19*](paper link)
    - SqueezeNet1_0
    - SqueezeNet1_1
  - VGG series[*20*](paper link)
    - VGG11
    - VGG13
    - VGG16
    - VGG19
  - DarkNet series[*21*](paper link)
    - DarkNet53
  - ACNet series[*22*](paper link)
    - ResNet50_ACNet_deploy

Note: The pretrained models of EfficientNetB1-B7 in the above models are transferred from pytorch version of EfficientNet, and the ResNeXt101_wsl series of pretrained models are transferred from Official repo, the remaining pretrained models are obtained by training with the PaddlePaddle framework, and the corresponding training hyperparameters are given in configs.

2.1.4 References

[1] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[2] He T, Zhang Z, Zhang H, et al. Bag of tricks for image classification with convolutional neural networks[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019: 558-567.

[3] Howard A, Sandler M, Chu G, et al. Searching for mobilenetv3[C]//Proceedings of the IEEE International Conference on Computer Vision. 2019: 1314-1324.

[4] Sandler M, Howard A, Zhu M, et al. Mobilenetv2: Inverted residuals and linear bottlenecks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 4510-4520.

[5] Howard A G, Zhu M, Chen B, et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications[J]. arXiv preprint arXiv:1704.04861, 2017.

[6] Ma N, Zhang X, Zheng H T, et al. Shufflenet v2: Practical guidelines for efficient cnn architecture design[C]//Proceedings of the European Conference on Computer Vision (ECCV). 2018: 116-131.

[7] Xie S, Girshick R, Dollár P, et al. Aggregated residual transformations for deep neural networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 1492-1500.

[8] Hu J, Shen L, Sun G. Squeeze-and-excitation networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 7132-7141.

[9] Gao S, Cheng M M, Zhao K, et al. Res2net: A new multi-scale backbone architecture[J]. IEEE transactions on pattern analysis and machine intelligence, 2019.

[10] Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2015: 1-9.

[11] Szegedy C, Ioffe S, Vanhoucke V, et al. Inception-v4, inception-resnet and the impact of residual connections on learning[C]//Thirty-first AAAI conference on artificial intelligence. 2017.

[12] Chollet F. Xception: Deep learning with depthwise separable convolutions[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 1251-1258.

[13] Wang J, Sun K, Cheng T, et al. Deep high-resolution representation learning for visual recognition[J]. arXiv preprint arXiv:1908.07919, 2019.

[14] Chen Y, Li J, Xiao H, et al. Dual path networks[C]//Advances in neural information processing systems. 2017: 4467-4475.

[15] Huang G, Liu Z, Van Der Maaten L, et al. Densely connected convolutional networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 4700-4708.

[16] Tan M, Le Q V. Efficientnet: Rethinking model scaling for convolutional neural networks[J]. arXiv preprint arXiv:1905.11946, 2019.

[17] Mahajan D, Girshick R, Ramanathan V, et al. Exploring the limits of weakly supervised pretraining[C]//Proceedings of the European Conference on Computer Vision (ECCV). 2018: 181-196.

[18] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[C]//Advances in neural information processing systems. 2012: 1097-1105.

[19] Iandola F N, Han S, Moskewicz M W, et al. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size[J]. arXiv preprint arXiv:1602.07360, 2016.

[20] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.

[21] Redmon J, Divvala S, Girshick R, et al. You only look once: Unified, real-time object detection[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 779-788.

[22] Ding X, Guo Y, Ding G, et al. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks[C]//Proceedings of the IEEE International Conference on Computer Vision. 2019: 1911-1920.

2.2 Tricks for Training

2.2.1 Choice of Optimizers:

Since the development of deep learning, there have been many researchers working on the optimizer. The purpose of the optimizer is to make the loss function as small as possible, so as to find suitable parameters to complete a certain task. At present, the main optimizers used in model training are SGD, RMSProp, Adam, AdaDelt and so on. The SGD optimizers with momentum is widely used in academia and industry, so most of models we release are trained by SGD optimizer with momentum. But the SGD optimizer with momentum has two disadvantages, one is that the convergence speed is slow, the other is that the initial learning rate is difficult to set, however, if the initial learning rate is set properly and the models are trained in sufficient iterations, the models trained by SGD with momentum can reach higher accuracy compared with the models trained by other optimizers. Some other optimizers with adaptive learning rate such as Adam, RMSProp and so on tent to converge faster, but the final convergence accuracy will be slightly worse. If you want to train a model in faster convergence speed, we recommend you use the optimizers with adaptive learning rate, but if you want to train a model with higher accuracy, we recommend you to use SGD optimizer with momentum.

2.2.2 Choice of Learning Rate and Learning Rate Declining Strategy:

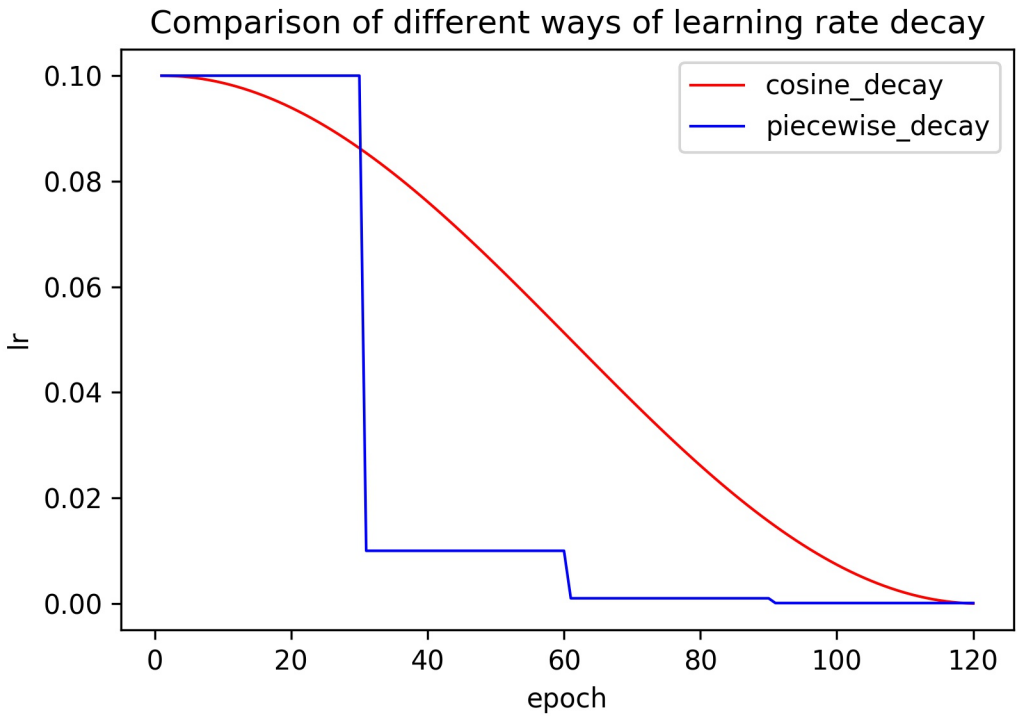
The choice of learning rate is related to the optimizer, data set and tasks. Here we mainly introduce the learning rate of training ImageNet-1K with momentum + SGD as the optimizer and the choice of learning rate decline.

Concept of Learning Rate

the learning rate is the hyperparameter to control the learning speed, the lower the learning rate, the slower the change of the loss value, though using a low learning rate can ensure that you will not miss any local minimum, but it also means that the convergence speed is slow, especially when the gradient is trapped in a gradient plateau area.

Learning Rate Decline Strategy

During training, if we always use the same learning rate, we cannot get the model with highest accuracy, so the learning rate should be adjust during training. In the early stage of training, the weights are in a random initialization state and the gradients are tended to descent, so we can set a relatively large learning rate for faster convergence. In the late stage of training, the weights are close to the optimal values, the optimal value cannot be reached by a relatively large learning rate, so a relatively smaller learning rate should be used. During training, many researchers use the piecewise_decay learning rate reduction strategy, which is a stepwise decline learning rate. For example, in the training of ResNet50, the initial learning rate we set is 0.1, and the learning rate drops to 1/10 every 30 epoches, the total epoches for training is 120. Besides the piecewise_decay, many researchers also proposed other ways to decrease the learning rate, such as polynomial_decay, exponential_decay and cosine_decay and so on, among them, cosine_decay has become the preferred learning rate reduction method for improving model accuracy beacause there is no need to adjust hyperparameters and the robustness is relatively high. The learning rate curves of cosine_decay and piecewise_decay are shown in the following figures, it is easy to observe that during the entire training process, cosine_decay keeps a relatively large learning rate, so its convergence is slower, but the final convergence accuracy is better than the one using piecewise_decay.



In addition, we can also see from the figures that the number of epoches with a small learning rate in cosine_decay is fewer, which will affect the final accuracy, so in order to make cosine_decay play a better effect, it is recommended to use cosine_decay in large epoched, such as 200 epoches.

Warmup Strategy

If a large batch_size is adopted to train nerual network, we recommend you to adopt warmup strategy. as the name suggests, the warmup strategy is to let model learning first warm up, we do not directly use the initial learning rate at the begining of training, instead, we use a gradually increasing learning rate to train the model, when the increasing learning rate reaches the initial learning rate, the learning rate reduction method mentioned in the learning rate reduction strategy is then used to decay the learning rate. Experiments show that when the batch size is large, warmup strategy can improve the accuracy. Some model training with large batch_size such as MobileNetV3 training, we set the epoch in warmup to 5 by default, that is, first in 5 epoches, the learning rate increases from 0 to initial learning rate, then learning rate decay begins.

2.2.3 Choice of Batch_size

Batch_size is an important hyperparameter in training neural networks, batch_size determines how much data is sent to the neural network to for training at a time. In the paper [1], the author found in experiments that when batch_size is linearly related to the learning rate, the convergence accuracy is hardly affected. When training ImageNet data, an initial learning rate of 0.1 are commonly chosen for training, and batch_size is 256, so according to the actual model size and memory, you can set the learning rate to 0.1*k, batch_size to 256*k.

2.2.4 Choice of Weight_decay

Overfitting is a common term in machine learning. A simple understanding is that the model performs well on the training data, but it performs poorly on the test data. In the convolutional neural network, there also exists the problem

of overfitting. To avoid overfitting, many regular ways have been proposed. Among them, weight_decay is one of the widely used ways to avoid overfitting. After the final loss function, L2 regularization(weight_decay) is added to the loss function, with the help of L2 regularization, the weight of the network tend to choose a smaller value, and finally the parameters in the entire network tends to 0, and the generalization performance of the model is improved accordingly. In different kinds of Deep learning frame, the meaning of L2_decay is the coefficient of L2 regularization, on paddle, the name of this value is L2_decay, so in the following the value is called L2_decay. the larger the coefficient, the more the model tends to be underfitting. In the task of training ImageNet, this parameter is set to 1e-4 in most network. In some small networks such as MobileNet networks, in order to avoid network underfitting, the value is set to 1e-5 ~ 4e-5. Of course, the setting of this value is also related to the specific data set, When the data set is large, the network itself tends to be under-fitted, and the value can be appropriately reduced. When the data set is small, the network tends to overfit itself, so the value can be increased appropriately. The following table shows the accuracy of MobileNetV1_x0_25 using different l2_decay on ImageNet-1k. Since MobileNetV1_x0_25 is a relatively small network, the large l2_decay will make the network tend to be underfitting, so in this network, 3e-5 are better choices compared with 1e-4.

In addition, the setting of L2_decay is also related to whether other regularization is used during training. If the data argument during the training is more complicated, which means that the training becomes more difficult, L2_decay can be appropriately reduced. The following table shows that the precision of ResNet50 using a different l2_decay on ImageNet-1K. It is easy to observe that after the training becomes difficult, using a smaller l2_decay helps to improve the accuracy of the model.

In summary, l2_decay can be adjusted according to specific tasks and models. Usually simple tasks or larger models are recommended to use Larger l2_decay, complex tasks or smaller models are recommended to use smaller l2_decay.

2.2.5 Choice of Label_smoothing

Label_smoothing is a regularization method in deep learning. Its full name is Label Smoothing Regularization (LSR), that is, label smoothing regularization. In the traditional classification task, when calculating the loss function, the real one hot label and the output of the neural network are calculated in cross-entropy formula, the label smoothing aims to make the real one hot label become smooth label, which makes the neural network no longer learn from the hard labels, but the soft labels with a probability value, where the probability of the position corresponding to the category is the largest and the probability of other positions are very small value, specific calculation method can be seen in the paper[2]. In label-smoothing, there is an epsilon parameter describing the degree of softening the label. The larger epsilon, the smaller the probability and smoother the label, on the contrary, the label tends to be hard label. during training on ImageNet-1K, the parameter is usually set to 0.1. In the experiments of training ResNet50, when using label_smoothing, the accuracy is higher than the one without label_smoothing, the following table shows the performance of ResNet50_vd with label smoothing and without label smoothing.

But, because label smoothing can be regarded as a regular way, on relatively small models, the accuracy improvement is not obvious or even decreases, the following table shows the accuracy performance of ResNet18 with label smoothing and without label smoothing on ImageNet-1K, it can be clearly seen that after using label smoothing, the accuracy of ResNet has decreased.

In summary, the use of label_smoohing for larger models can effectively improve the accuracy of the model, and the use of label_smoohing for smaller models may reduce the accuracy of the model, so before deciding whether to use label_smoohing, you need to evaluate the size of the model and the difficulty of the task.

2.2.6 Change the Crop Area and Stretch Transformation Degree of the Images for Small Models

In the standard preprocessing of ImageNet-1k data, two values of scale and ratio are defined in the random_crop function. These two values respectively determine the size of the image crop and the degree of stretching of the image. The default value of scale is 0.08-1(lower_scale-upper_scale), the default value range of ratio is 3/4-4/3(lower_ratio-upper_ratio). In small network training, such data argument will make the network underfitting, resulting in a decrease

in accuracy. In order to improve the accuracy of the network, you can make the data argument weaker, that is, increase the crop area of the images or weaken the degree of stretching and transformation of the images, we can achieve weaker image transformation by increasing the value of lower_scale or narrowing the gap between lower_ratio and upper_scale. The following table lists the accuracy of training MobileNetV2_x0_25 with different lower_scale. It can be seen that the training accuracy and validation accuracy are improved after increasing the crop area of the images

2.2.7 Use Data Augmentation to Improve Accuracy

In general, the size of the data set is critical to the performances, but the annotation of images are often more expensive, so the number of annotated images are often scarce. In this case, the data argument is particularly important. In the standard data augmentation for training on ImageNet-1k, two data augmentation methods which are random_crop and random_flip are mainly used. However, in recent years, more and more data augmentation methods have been proposed, such as cutout, mixup, cutmix, AutoAugment, etc. Experiments show that these data augmentation methods can effectively improve the accuracy of the model. The following table lists the performance of ResNet50 in 8 different data augmentation methods. It can be seen that compared to the baseline, all data augmentation methods can be useful for the accuracy of ResNet50, among them cutmix is currently the most effective data argument. More data argument can be seen here**Data Argument**.

2.2.8 Determine the Tuning Strategy by Train_acc and Test_acc

In the process of training the network, the training set accuracy rate and validation set accuracy rate of each epoch are usually printed. Generally speaking, the accuracy of the training set is slightly higher than the accuracy of the validation set or the same are good state in training, but if you find that the accuracy of training set is much higher than the one of validation set, it means that overfitting happens in your task, which need more regularization, such as increase the value of L2_decay, using more data argument or label smoothing and so on. If you find that the accuracy of training set is lower than the one of validation set, it means that underfitting happens in your task, which recommend you to decrease the value of L2_decay, using fewer data argument, increase the area of the crop area of the images, weaken the stretching transformation of the images, remove label_smoothing, etc.

2.2.9 Improve the Accuracy of Your Own Data Set with Existing Pre-trained Models

In the field of computer vision, it has become common to load pre-trained models to train one's own tasks. Compared with starting training from random initialization, loading pre-trained models can often improve the accuracy of specific tasks. In general, the pre-trained model widely used in the industry is obtained from the ImageNet-1k dataset. The fc layer weight of the pre-trained model is a matrix of k*1000, where k is The number of neurons before, and the weights of the fc layer is not need to load because of the different tasks. In terms of learning rate, if your training data set is particularly small (such as less than 1,000), we recommend that you use a smaller initial learning rate, such as 0.001 (batch_size: 256, the same below), to avoid a large learning rate undermine pre-training weights, if your training data set is relatively large (greater than 100,000), we recommend that you try a larger initial learning rate, such as 0.01 or greater.

> If you think this guide is helpful to you, welcome to star our repo:https://github.com/PaddlePaddle/PaddleClas

2.2.10 Reference

[1]P. Goyal, P. Dolla r, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch SGD: training imagenet in 1 hour. CoRR, abs/1706.02677, 2017.

[2]C.Szegedy,V.Vanhoucke,S.Ioffe,J.Shlens,andZ.Wojna. Rethinking the inception architecture for computer vision. CoRR, abs/1512.00567, 2015.

2.3 ResNet and ResNet_vd series

2.3.1 Overview

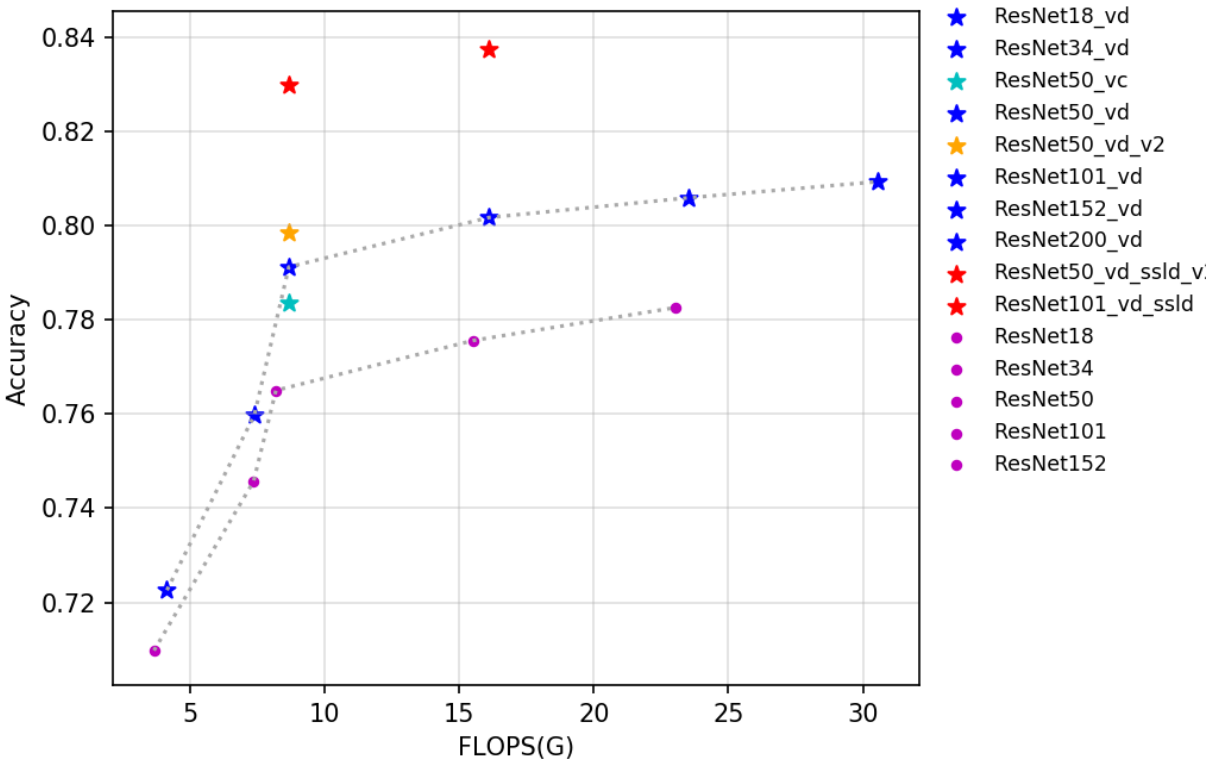
The ResNet series model was proposed in 2015 and won the championship in the ILSVRC2015 competition with a top5 error rate of 3.57%. The network innovatively proposed the residual structure, and built the ResNet network by stacking multiple residual structures. Experiments show that using residual blocks can improve the convergence speed and accuracy effectively.

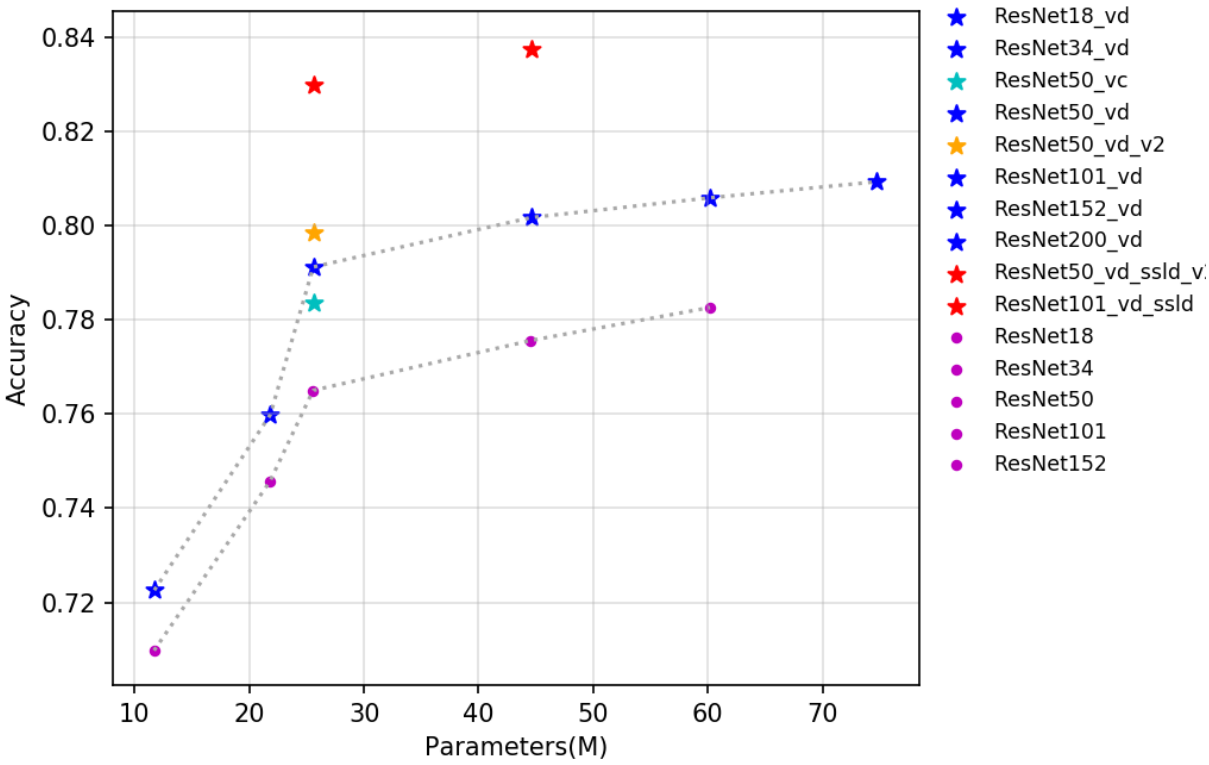
Joyce Xu of Stanford university calls ResNet one of three architectures that "really redefine the way we think about neural networks." Due to the outstanding performance of ResNet, more and more scholars and engineers from academia and industry have improved its structure. The well-known ones include wide-resnet, resnet-vc, resnet-vd, Res2Net, etc. The number of parameters and FLOPs of resnet-vc and resnet-vd are almost the same as those of ResNet, so we hereby unified them into the ResNet series.

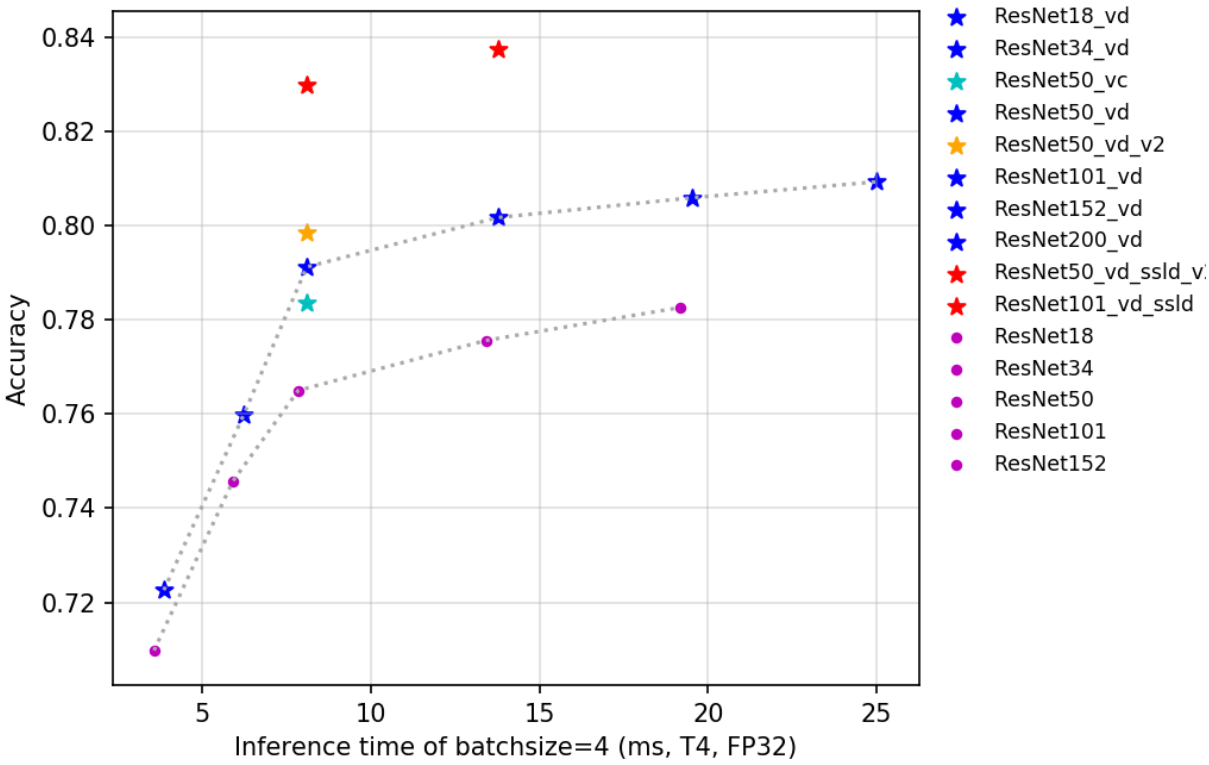
The models of the ResNet series released this time include 14 pre-trained models including ResNet50, ResNet50_vd, ResNet50_vd_ssld, and ResNet200_vd. At the training level, ResNet adopted the standard training process for training ImageNet, while the rest of the improved model adopted more training strategies, such as cosine decay for the decline of learning rate and the regular label smoothing method,mixup was added to the data preprocessing, and the total number of iterations increased from 120 epoches to 200 epoches.

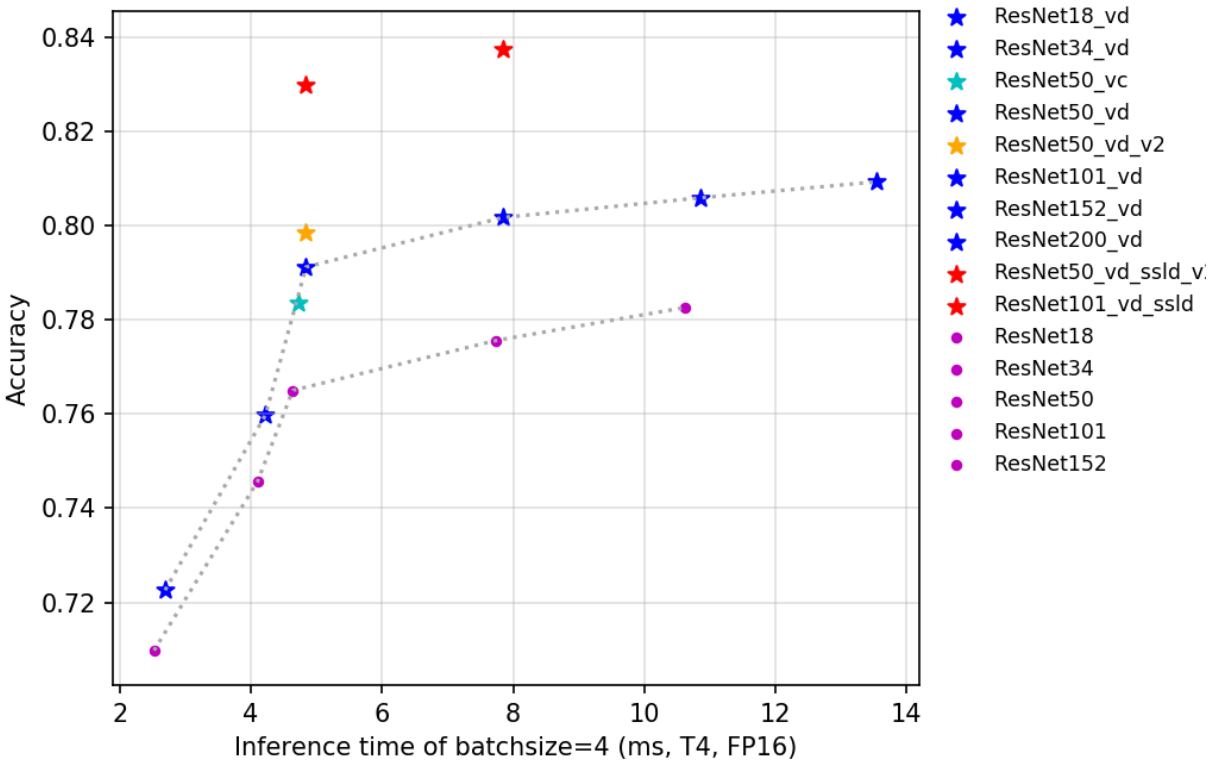
Among them, ResNet50_vd_v2 and ResNet50_vd_ssld adopted knowledge distillation, which further improved the accuracy of the model while keeping the structure unchanged. Specifically, the teacher model of ResNet50_vd_v2 is ResNet152_vd (top1 accuracy 80.59%), the training set is imagenet-1k, the teacher model of ResNet50_vd_ssld is ResNeXt101_32x16d_wsl (top1 accuracy 84.2%), and the training set is the combination of 4 million data mined by imagenet-22k and ImageNet-1k . The specific methods of knowledge distillation are being continuously updated.

The FLOPS, parameters, and inference time on the T4 GPU of this series of models are shown in the figure below.


0.84
0.82
0.80
0.78
0.76
0.74
0.72
Accuracy
5
10
15
20
25
30
FLOPS(G)
ResNet18_vd
ResNet34_vd
ResNet50_vc
ResNet50_vd
ResNet50_vd_v2
ResNet101_vd
ResNet152_vd
ResNet200_vd
ResNet50_vd_ssld_v
ResNet101_vd_ssld
ResNet18
ResNet34
ResNet50
ResNet101
ResNet152


0.84
0.82
0.80
0.78
0.76
0.74
0.72
Accuracy
10
20
30
40
50
60
70
Parameters(M)
ResNet18_vd
ResNet34_vd
ResNet50_vc
ResNet50_vd
ResNet50_vd_v2
ResNet101_vd
ResNet152_vd
ResNet200_vd
ResNet50_vd_ssld_v
ResNet101_vd_ssld
ResNet18
ResNet34
ResNet50
ResNet101
ResNet152


0.84
0.82
0.80
0.78
0.76
0.74
0.72
Accuracy
5
10
15
20
25
Inference time of batchsize=4 (ms, T4, FP32)
ResNet18_vd
ResNet34_vd
ResNet50_vc
ResNet50_vd
ResNet50_vd_v2
ResNet101_vd
ResNet152_vd
ResNet200_vd
ResNet50_vd_ssld_v
ResNet101_vd_ssld
ResNet18
ResNet34
ResNet50
ResNet101
ResNet152



As can be seen from the above curves, the higher the number of layers, the higher the accuracy, but the corresponding number of parameters, calculation and latency will increase. ResNet50_vd_ssld further improves the accuracy of top-1 of the ImageNet-1k validation set by using stronger teachers and more data, reaching 82.39%, refreshing the accuracy of ResNet50 series models.

2.3.2 Accuracy, FLOPS and Parameters

- Note: `ResNet50_vd_ssld_v2` is obtained by adding AutoAugment in training process on the basis of `ResNet50_vd_ssld` training strategy.`Fix_ResNet50_vd_ssld_v2` stopped all parameter updates of `ResNet50_vd_ssld_v2` except the FC layer,and fine-tuned on ImageNet1k dataset, the resolution is 320x320.

2.3.3 Inference speed based on V100 GPU

2.3.4 Inference speed based on T4 GPU

2.4 Mobile and Embedded Vision Applications Network series

2.4.1 Overview

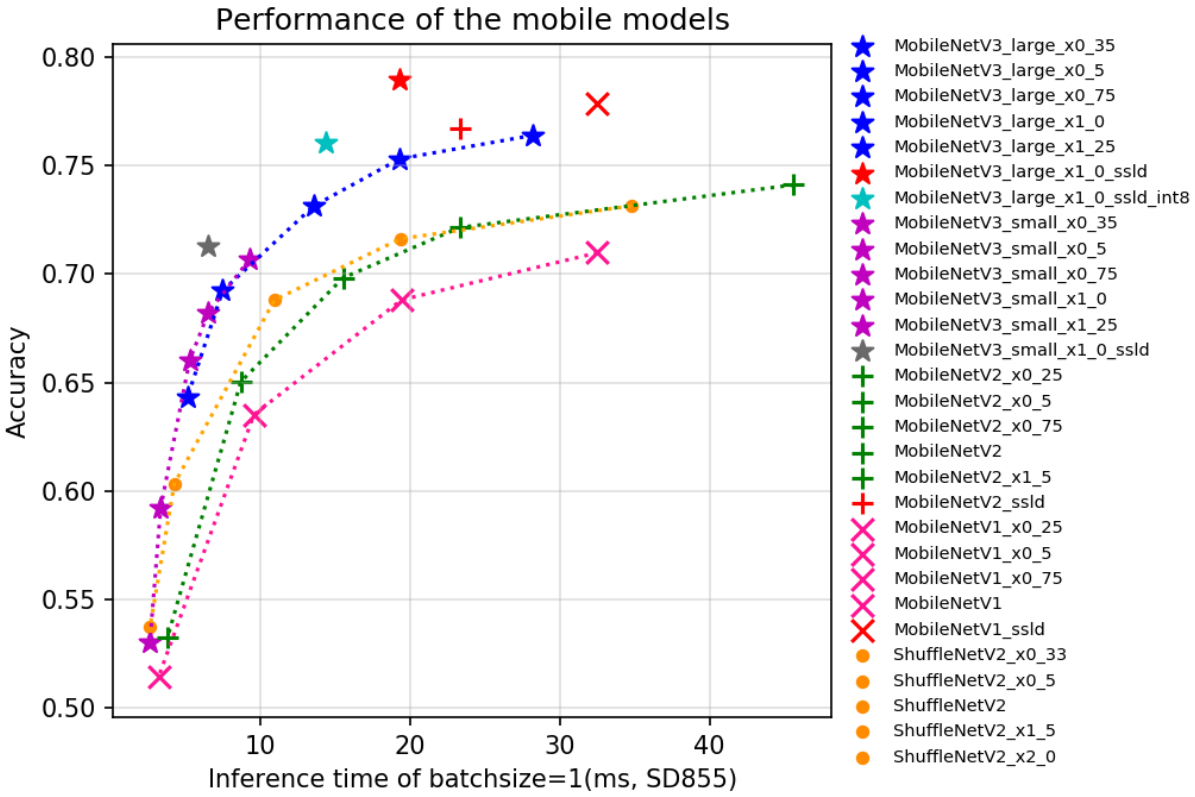
MobileNetV1 is a network launched by Google in 2017 for use on mobile devices or embedded devices. The network replaces the depthwise separable convolution with the traditional convolution operation, that is, the combination of

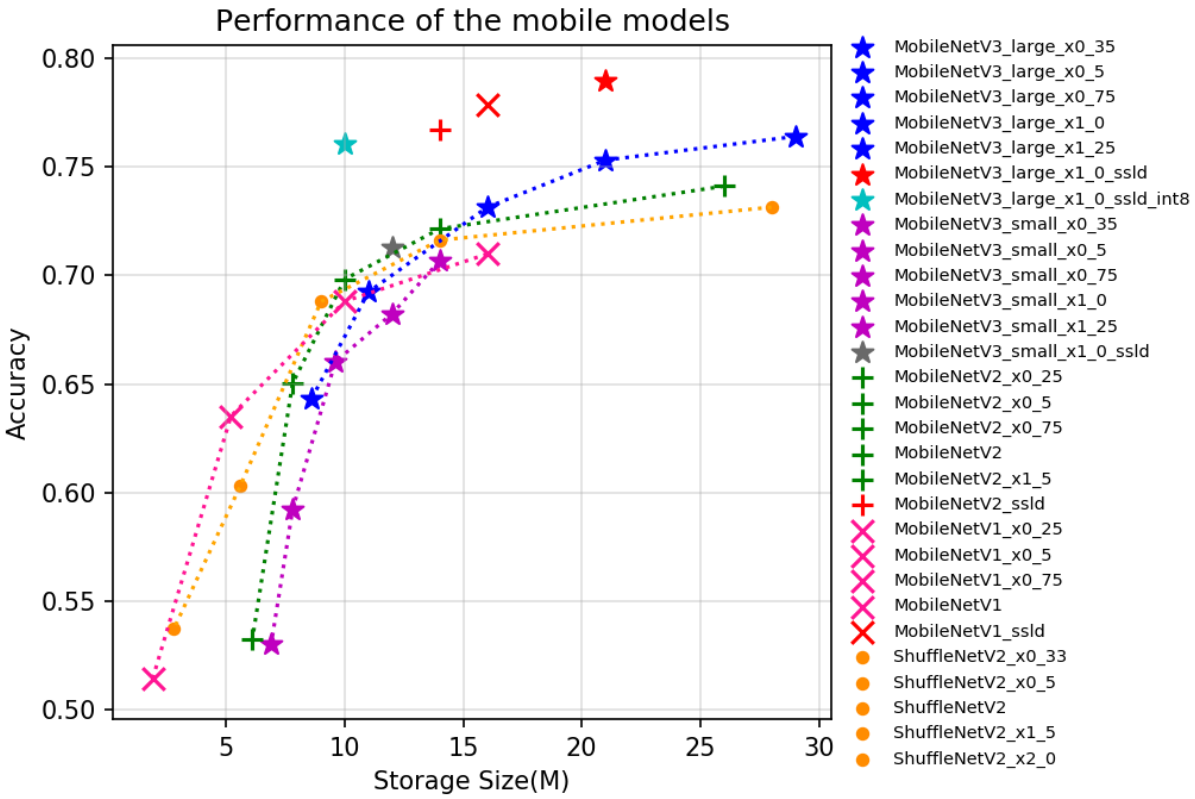
depthwise convolution and pointwise convolution. Compared with the traditional convolution operation, this combination can greatly save the number of parameters and computation. At the same time, MobileNetV1 can also be used for object detection, image segmentation and other visual tasks.

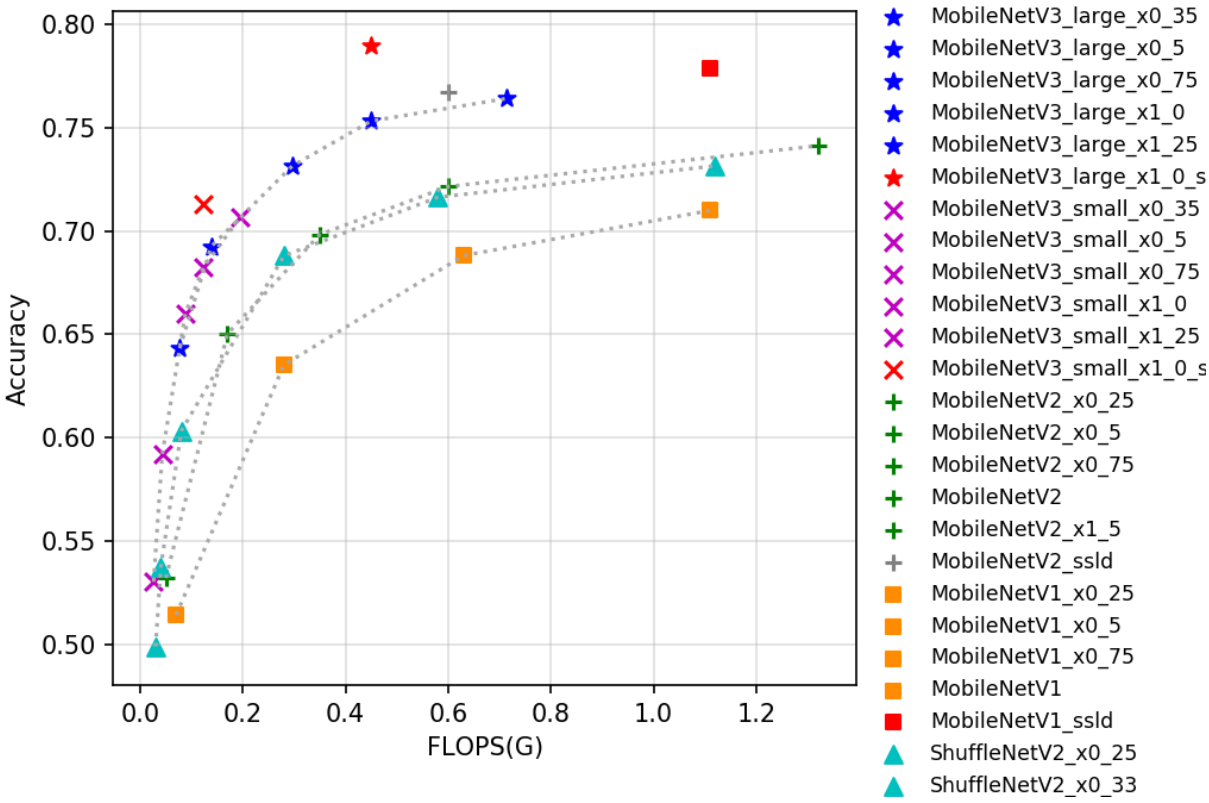
MobileNetV2 is a lightweight network proposed by Google following MobileNetV1. Compared with MobileNetV1, MobileNetV2 proposed Linear bottlenecks and Inverted residual block as a basic network structures, to constitute MobileNetV2 network architecture through stacking these basic module a lot. In the end, higher classification accuracy was achieved when FLOPS was only half of MobileNetV1.

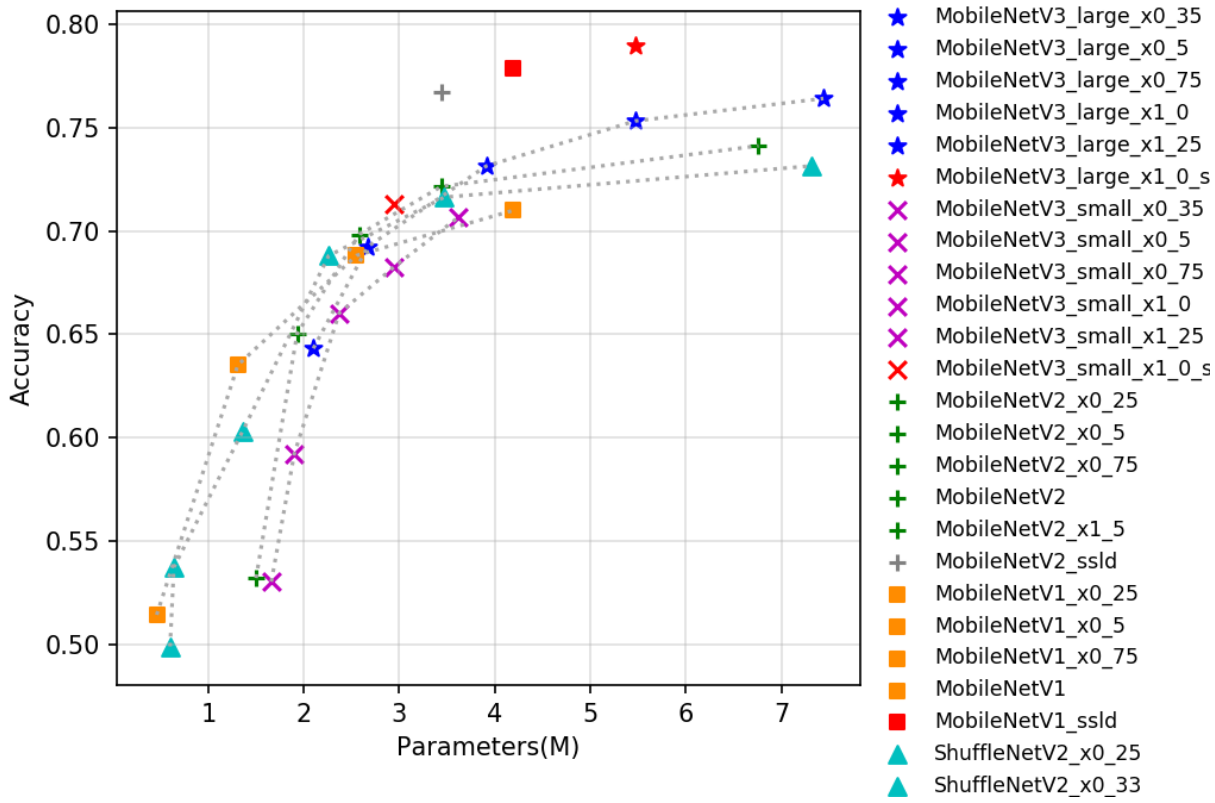
The ShuffleNet series network is the lightweight network structure proposed by MEGVII. So far, there are two typical structures in this series network, namely, ShuffleNetV1 and ShuffleNetV2. A Channel Shuffle operation in ShuffleNet can exchange information between groups and perform end-to-end training. In the paper of ShuffleNetV2, the author proposes four criteria for designing lightweight networks, and designs the ShuffleNetV2 network according to the four criteria and the shortcomings of ShuffleNetV1.

MobileNetV3 is a new and lightweight network based on NAS proposed by Google in 2019. In order to further improve the effect, the activation functions of relu and sigmoid were replaced with hard_swish and hard_sigmoid activation functions, and some improved strategies were introduced to reduce the amount of network computing.




Performance of the mobile models
Accuracy
Storage Size(M)
0.80
0.75
0.70
0.65
0.60
0.55
0.50
5
10
15
20
25
30
MobileNetV3_large_x0_35
MobileNetV3_large_x0_5
MobileNetV3_large_x0_75
MobileNetV3_large_x1_0
MobileNetV3_large_x1_25
MobileNetV3_large_x1_0_ssld
MobileNetV3_large_x1_0_ssld_int8
MobileNetV3_small_x0_35
MobileNetV3_small_x0_5
MobileNetV3_small_x0_75
MobileNetV3_small_x1_0
MobileNetV3_small_x1_25
MobileNetV3_small_x1_0_ssld
MobileNetV2_x0_25
MobileNetV2_x0_5
MobileNetV2_x0_75
MobileNetV2
MobileNetV2_x1_5
MobileNetV2_ssld
MobileNetV1_x0_25
MobileNetV1_x0_5
MobileNetV1_x0_75
MobileNetV1
MobileNetV1_ssld
ShuffleNetV2_x0_33
ShuffleNetV2_x0_5
ShuffleNetV2
ShuffleNetV2_x1_5
ShuffleNetV2_x2_0


0.80
0.75
0.70
0.65
0.60
0.55
0.50
Accuracy
0.0
0.2
0.4
0.6
0.8
1.0
1.2
FLOPS(G)
MobileNetV3_large_x0_35
MobileNetV3_large_x0_5
MobileNetV3_large_x0_75
MobileNetV3_large_x1_0
MobileNetV3_large_x1_25
MobileNetV3_large_x1_0_s
MobileNetV3_small_x0_35
MobileNetV3_small_x0_5
MobileNetV3_small_x0_75
MobileNetV3_small_x1_0
MobileNetV3_small_x1_25
MobileNetV3_small_x1_0_s
MobileNetV2_x0_25
MobileNetV2_x0_5
MobileNetV2_x0_75
MobileNetV2
MobileNetV2_x1_5
MobileNetV2_ssld
MobileNetV1_x0_25
MobileNetV1_x0_5
MobileNetV1_x0_75
MobileNetV1
MobileNetV1_ssld
ShuffleNetV2_x0_25
ShuffleNetV2_x0_33



Currently there are 32 pretrained models of the mobile series open source by PaddleClas, and their indicators are shown in the figure below. As you can see from the picture, newer lightweight models tend to perform better, and MobileNetV3 represents the latest lightweight neural network architecture. In MobileNetV3, the author used 1x1 convolution after global-avg-pooling in order to obtain higher accuracy,this operation significantly increases the number of parameters but has little impact on the amount of computation, so if the model is evaluated from a storage perspective of excellence, MobileNetV3 does not have much advantage, but because of its smaller computation, it has a faster inference speed. In addition, the SSLD distillation model in our model library performs excellently, refreshing the accuracy of the current lightweight model from various perspectives. Due to the complex structure and many branches of the MobileNetV3 model, which is not GPU friendly, the GPU inference speed is not as good as that of MobileNetV1.

2.4.2 Accuracy, FLOPS and Parameters

2.4.3 Inference speed and storage size based on SD855

2.4.4 Inference speed based on T4 GPU

2.5 SEResNeXt and Res2Net series

2.5.1 Overview

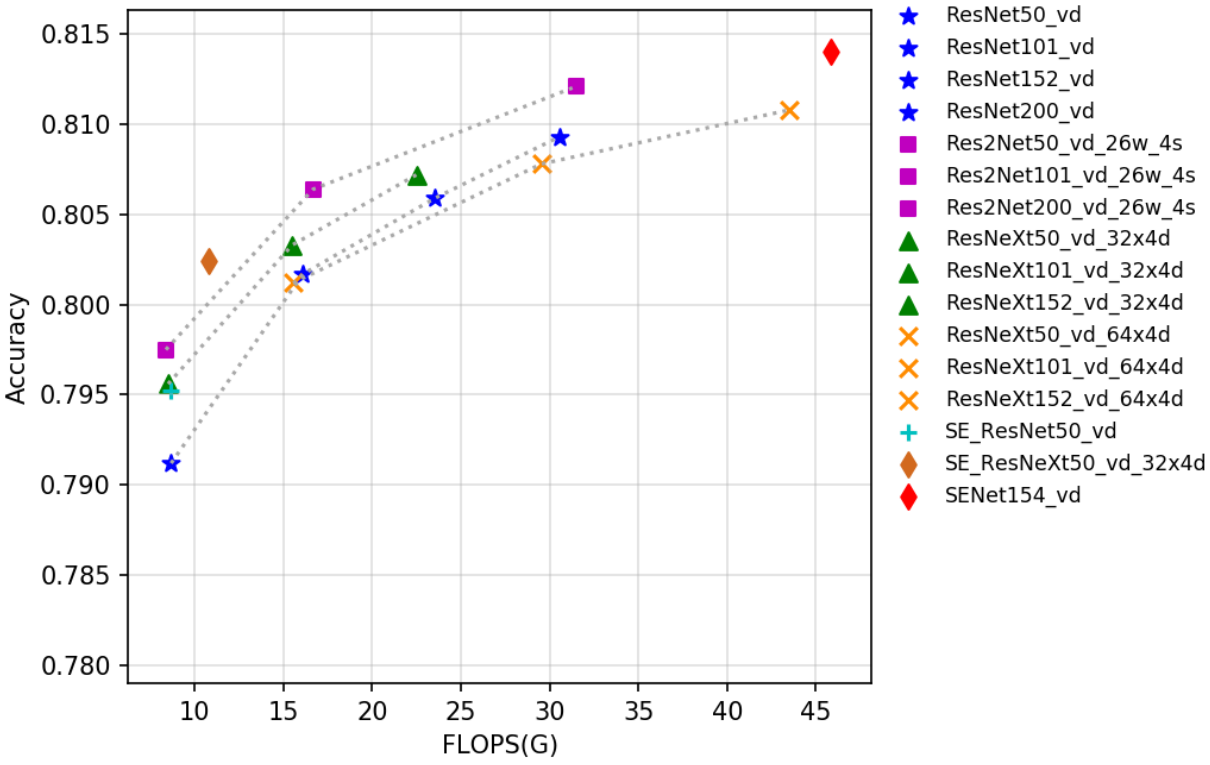
ResNeXt, one of the typical variants of ResNet, was presented at the CVPR conference in 2017. Prior to this, the methods to improve the model accuracy mainly focused on deepening or widening the network, which increased the

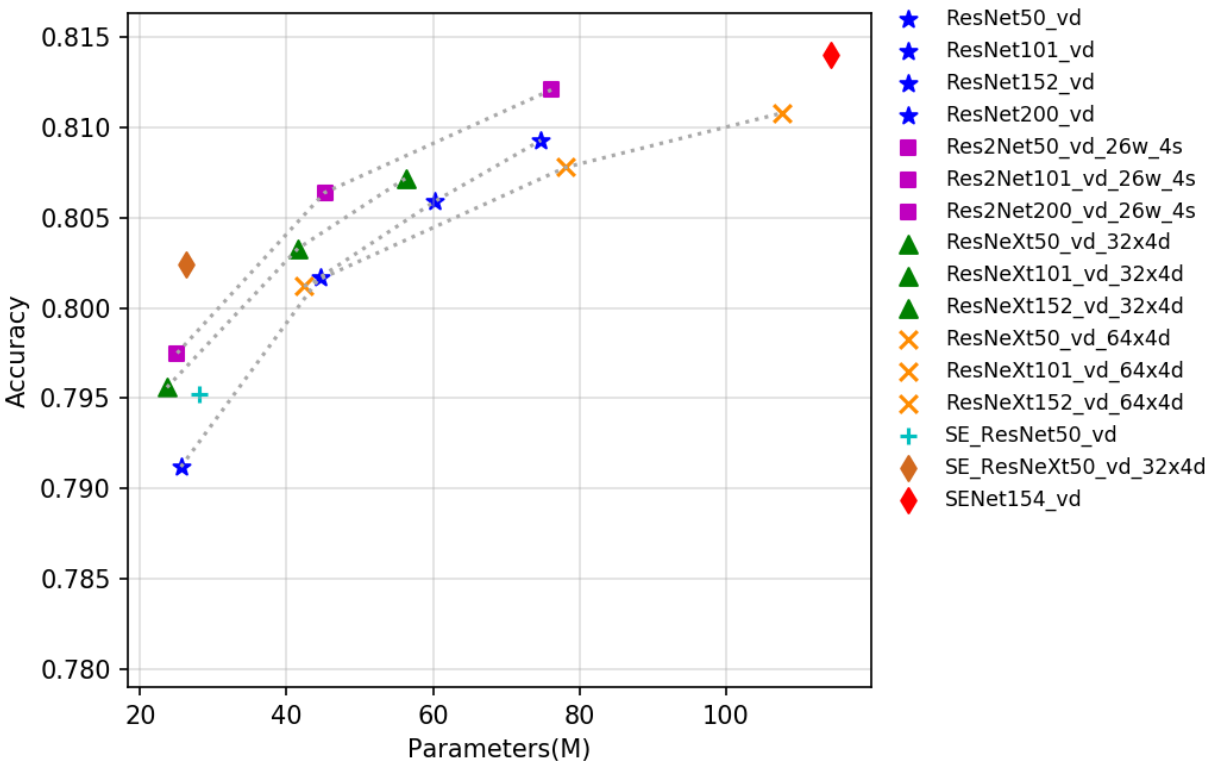
number of parameters and calculation, and slowed down the inference speed accordingly. The concept of cardinality was proposed in ResNeXt structure. The author found that increasing the number of channel groups was more effective than increasing the depth and width through experiments. It can improve the accuracy without increasing the parameter complexity and reduce the number of parameters at the same time, so it is a more successful variant of ResNet.

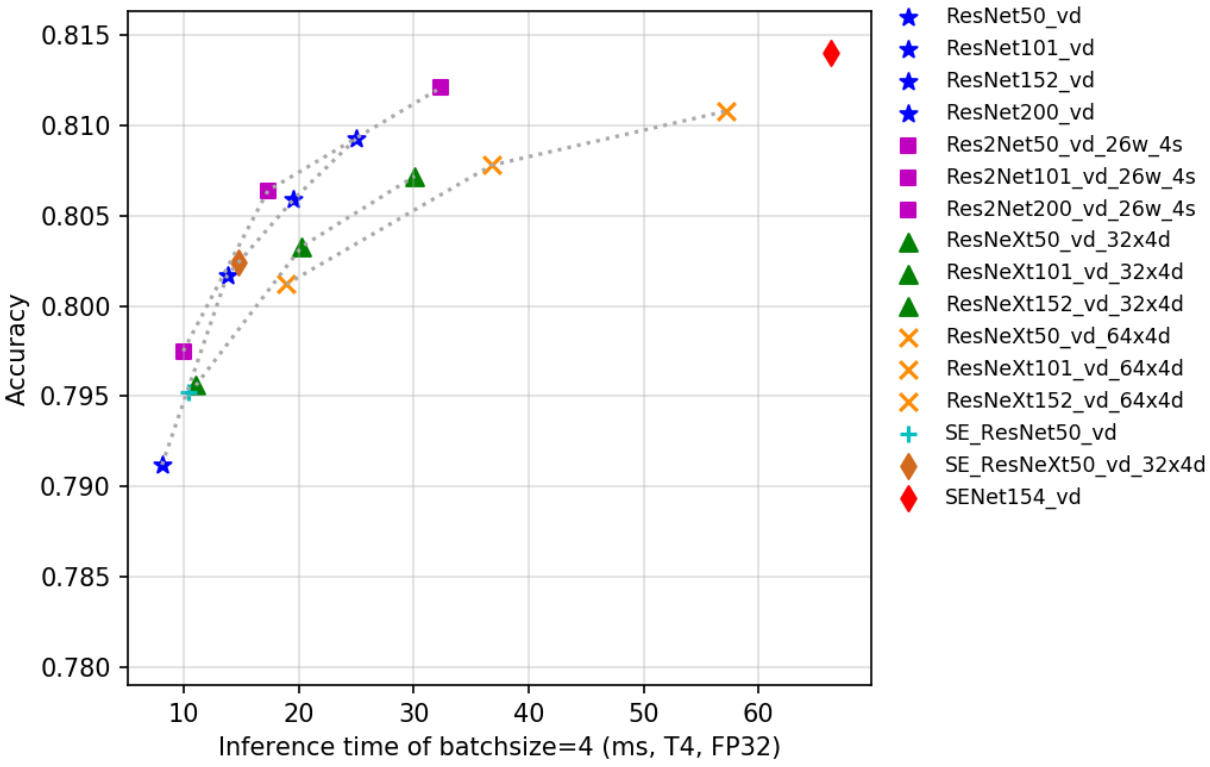
SENet is the winner of the 2017 ImageNet classification competition. It proposes a new SE structure that can be migrated to any other network. It controls the scale to enhance the important features between each channel, and weaken the unimportant features. So that the extracted features are more directional.

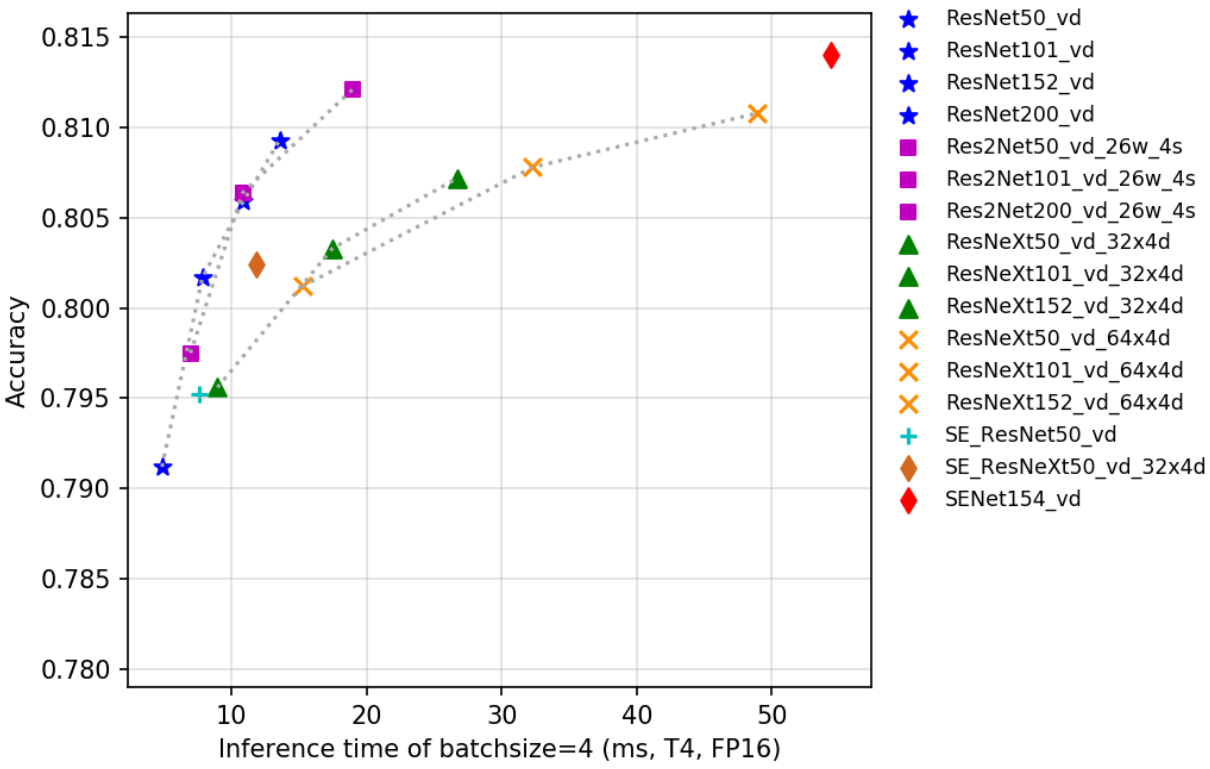
Res2Net is a brand-new improvement of ResNet proposed in 2019. The solution can be easily integrated with other excellent modules. Without increasing the amount of calculation, the performance on ImageNet, CIFAR-100 and other data sets exceeds ResNet. Res2Net, with its simple structure and superior performance, further explores the multi-scale representation capability of CNN at a more fine-grained level. Res2Net reveals a new dimension to improve model accuracy, called scale, which is an essential and more effective factor in addition to the existing dimensions of depth, width, and cardinality. The network also performs well in other visual tasks such as object detection and image segmentation.

The FLOPS, parameters, and inference time on the T4 GPU of this series of models are shown in the figure below.




0.815
0.810
0.805
0.800
0.795
0.790
0.785
0.780
Accuracy
20
40
60
80
100
Parameters(M)
ResNet50_vd
ResNet101_vd
ResNet152_vd
ResNet200_vd
Res2Net50_vd_26w_4s
Res2Net101_vd_26w_4s
Res2Net200_vd_26w_4s
ResNeXt50_vd_32x4d
ResNeXt101_vd_32x4d
ResNeXt152_vd_32x4d
ResNeXt50_vd_64x4d
ResNeXt101_vd_64x4d
ResNeXt152_vd_64x4d
SE_ResNet50_vd
SE_ResNeXt50_vd_32x4d
SENet154_vd


0.815
0.810
0.805
0.800
0.795
0.790
0.785
0.780
Accuracy
10
20
30
40
50
60
Inference time of batchsize=4 (ms, T4, FP32)
ResNet50_vd
ResNet101_vd
ResNet152_vd
ResNet200_vd
Res2Net50_vd_26w_4s
Res2Net101_vd_26w_4s
Res2Net200_vd_26w_4s
ResNeXt50_vd_32x4d
ResNeXt101_vd_32x4d
ResNeXt152_vd_32x4d
ResNeXt50_vd_64x4d
ResNeXt101_vd_64x4d
ResNeXt152_vd_64x4d
SE_ResNet50_vd
SE_ResNeXt50_vd_32x4d
SENet154_vd



At present, there are a total of 24 pretrained models of the three categories open sourced by PaddleClas, and the indicators are shown in the figure. It can be seen from the diagram that under the same Flops and Params, the improved model tends to have higher accuracy, but the inference speed is often inferior to the ResNet series. On the other hand, Res2Net performed better. Compared with group operation in ResNeXt and SE structure operation in SEResNet, Res2Net tended to have better accuracy in the same Flops, Params and inference speed.

2.5.2 Accuracy, FLOPS and Parameters

2.5.3 Inference speed based on V100 GPU

2.5.4 Inference speed based on T4 GPU

2.6 Inception series

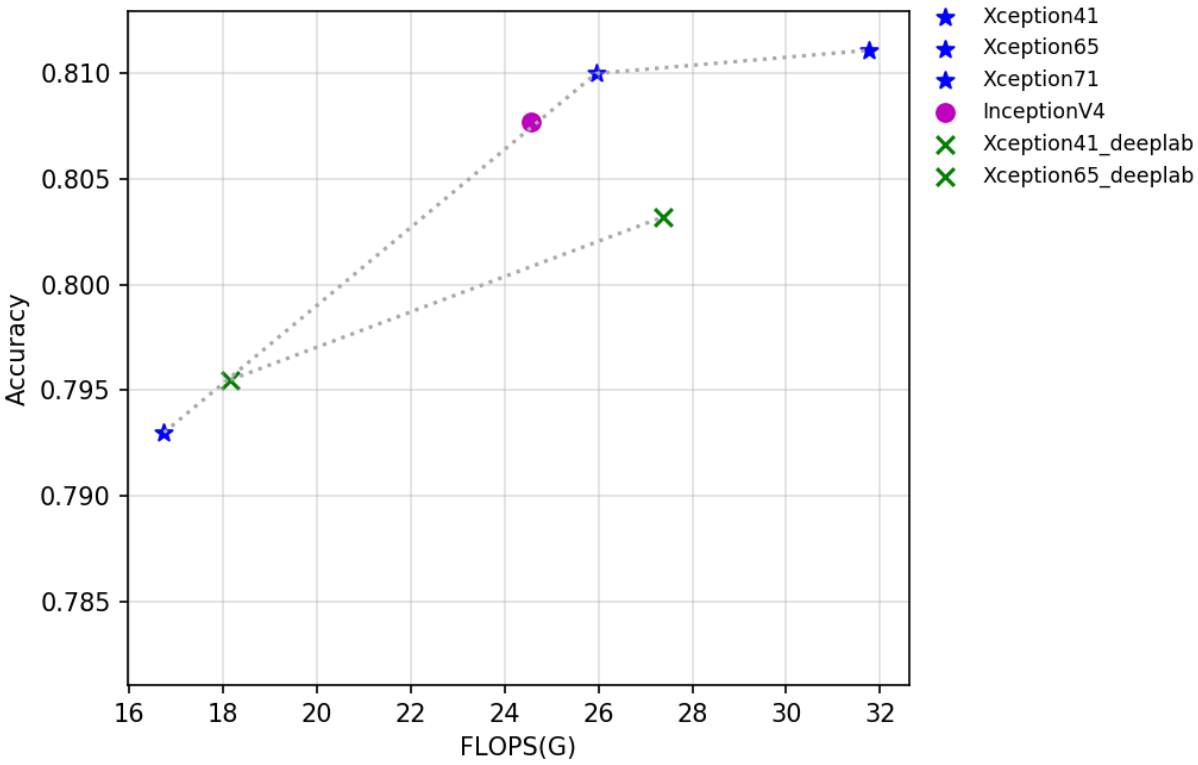
2.6.1 Overview

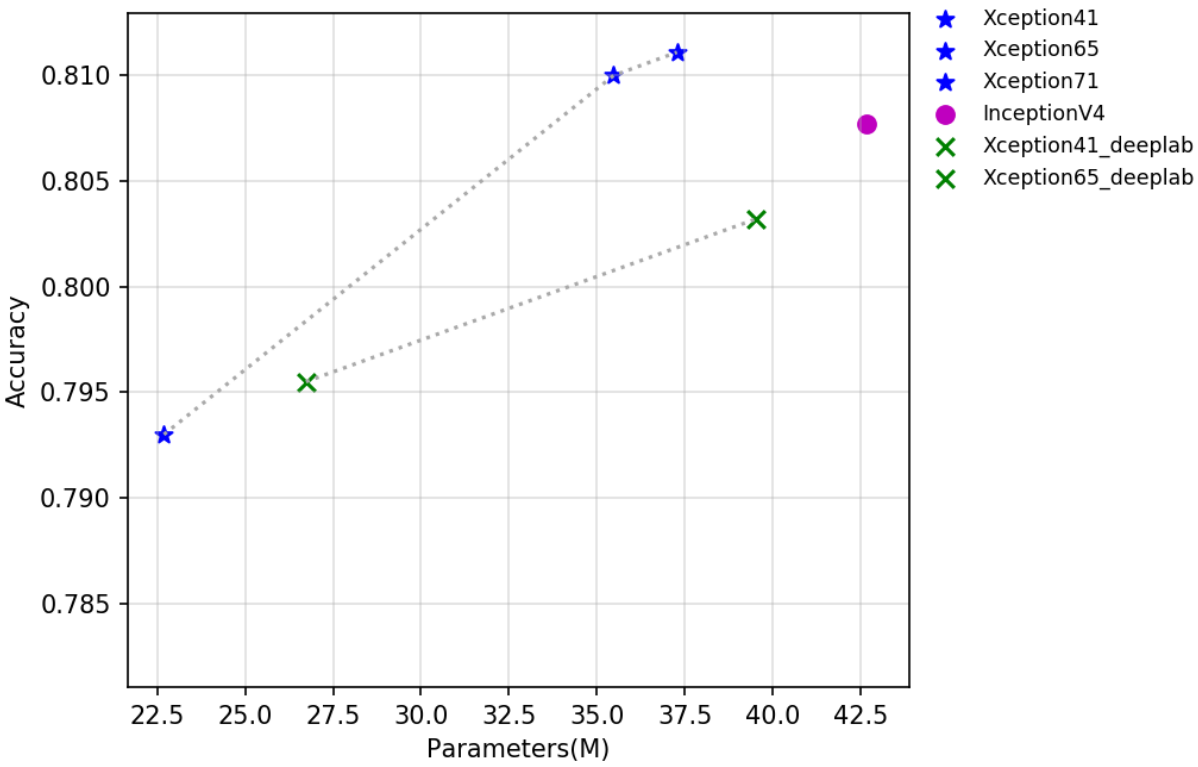
GoogLeNet is a new neural network structure designed by Google in 2014, which, together with VGG network, became the twin champions of the ImageNet challenge that year. GoogLeNet introduces the Inception structure for the first time, and stacks the Inception structure in the network so that the number of network layers reaches 22, which is also the mark of the convolutional network exceeding 20 layers for the first time. Since 1x1 convolution is used in the Inception structure to reduce the dimension of channel number, and Global pooling is used to replace the traditional method of processing features in multiple fc layers, the final GoogLeNet network has much less FLOPS and parameters than VGG network, which has become a beautiful scenery of neural network design at that time.

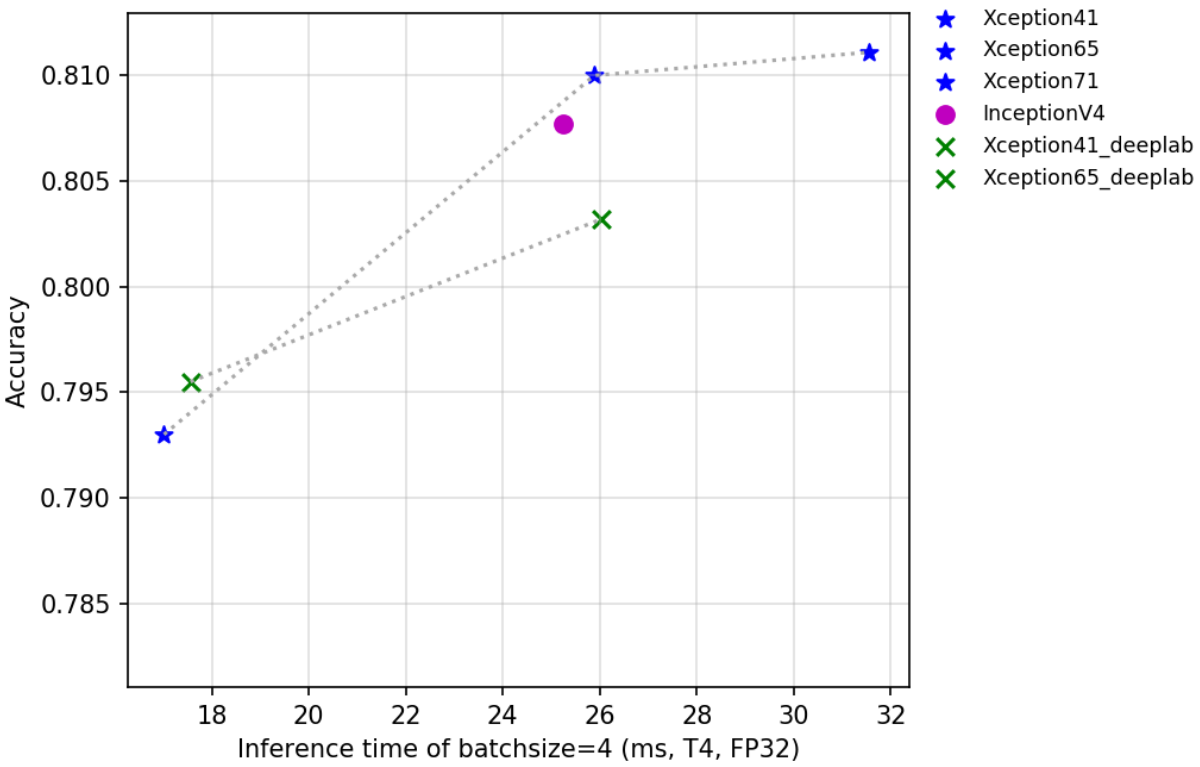
Xception is another improvement to InceptionV3 that Google proposed after Inception. In Xception, the author used the depthwise separable convolution to replace the traditional convolution operation, which greatly saved the network FLOPS and the number of parameters, but improved the accuracy. In DeeplabV3+, the author further improved the Xception and increased the number of Xception layers, and designed the network of Xception65 and Xception71.

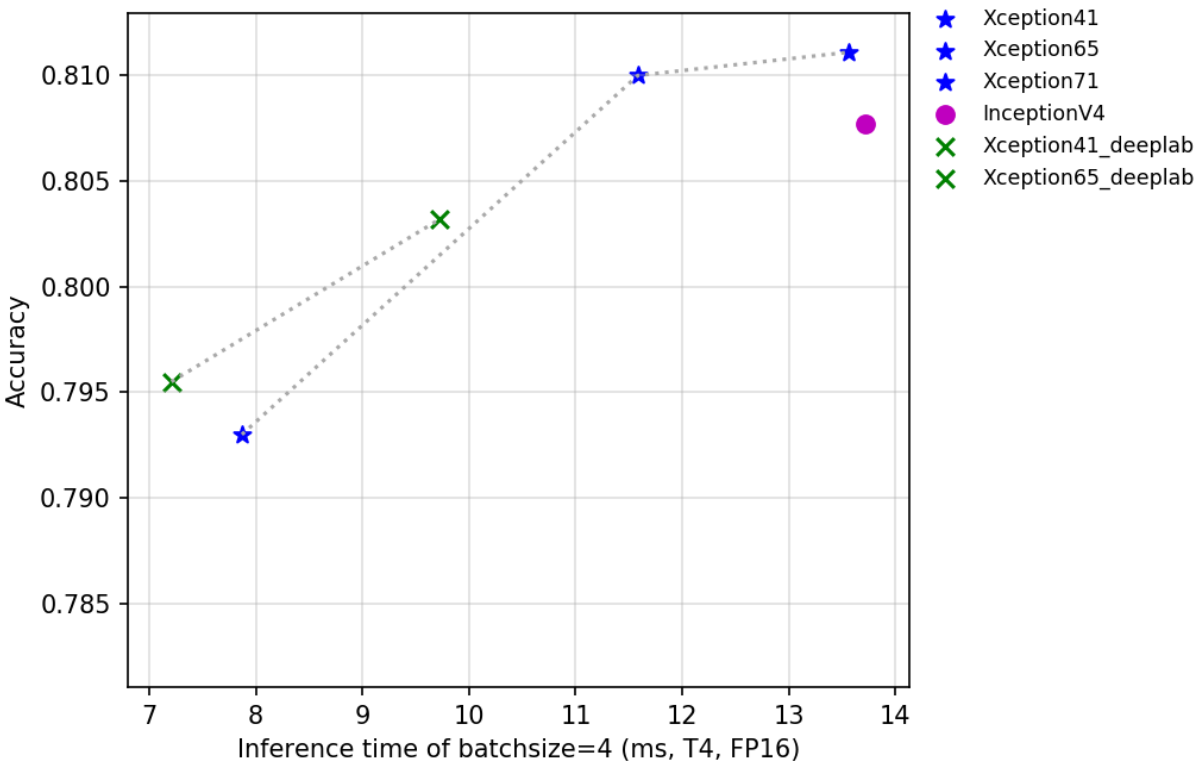
InceptionV4 is a new neural network designed by Google in 2016, when residual structure were all the rage, but the authors believe that high performance can be achieved using only Inception structure. InceptionV4 uses more Inception structure to achieve even greater precision on Imagenet-1k.

The FLOPS, parameters, and inference time on the T4 GPU of this series of models are shown in the figure below.




Xception41
Xception65
Xception71
InceptionV4
Xception41_deeplab
Xception65_deeplab
Accuracy
0.810
0.805
0.800
0.795
0.790
0.785
22.5
25.0
27.5
30.0
32.5
35.0
37.5
40.0
42.5
Parameters(M)


Xception41
Xception65
Xception71
InceptionV4
Xception41_deeplab
Xception65_deeplab
Accuracy
0.810
0.805
0.800
0.795
0.790
0.785
18
20
22
24
26
28
30
32
Inference time of batchsize=4 (ms, T4, FP32)



The figure above reflects the relationship between the accuracy of Xception series and InceptionV4 and other indicators. Among them, Xception_deeplab is consistent with the structure of the paper, and Xception is an improved model developed by PaddleClas, which improves the accuracy by about 0.6% when the inference speed is basically unchanged. Details of the improved model are being updated, so stay tuned.

2.6.2 Accuracy, FLOPS and Parameters

2.6.3 Inference speed based on V100 GPU

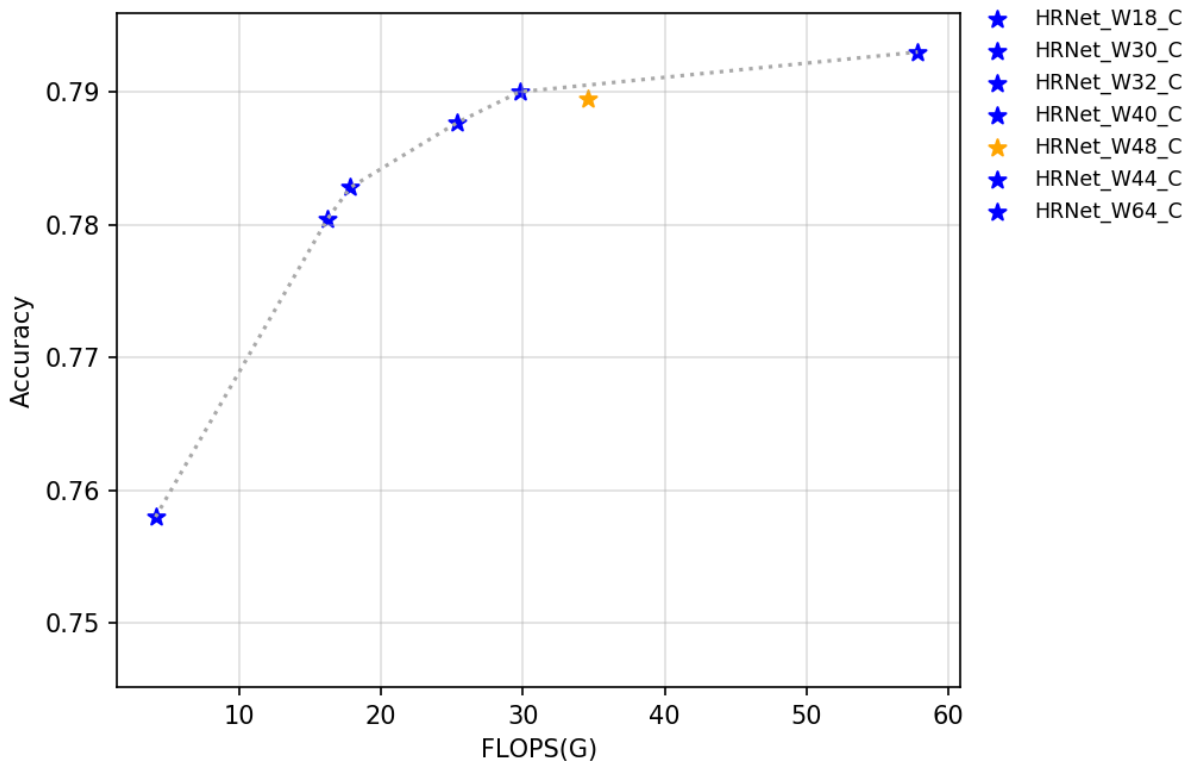
2.6.4 Inference speed based on T4 GPU

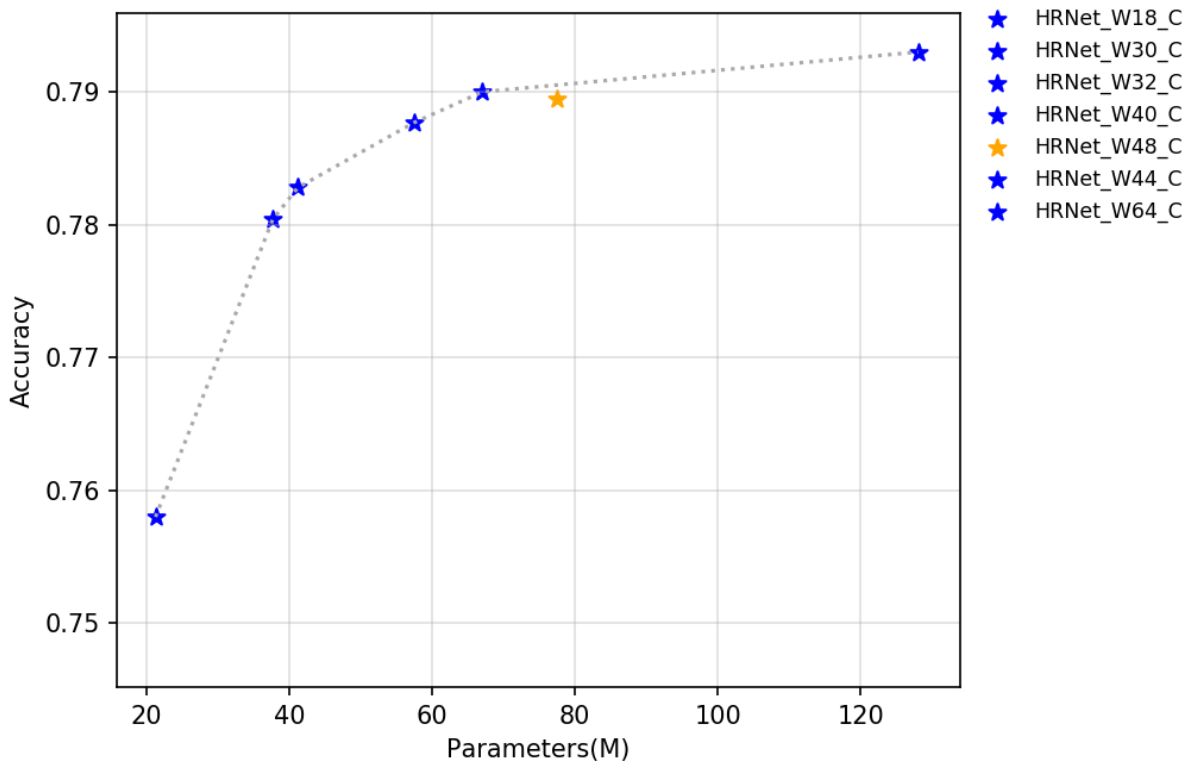
2.7 HRNet series

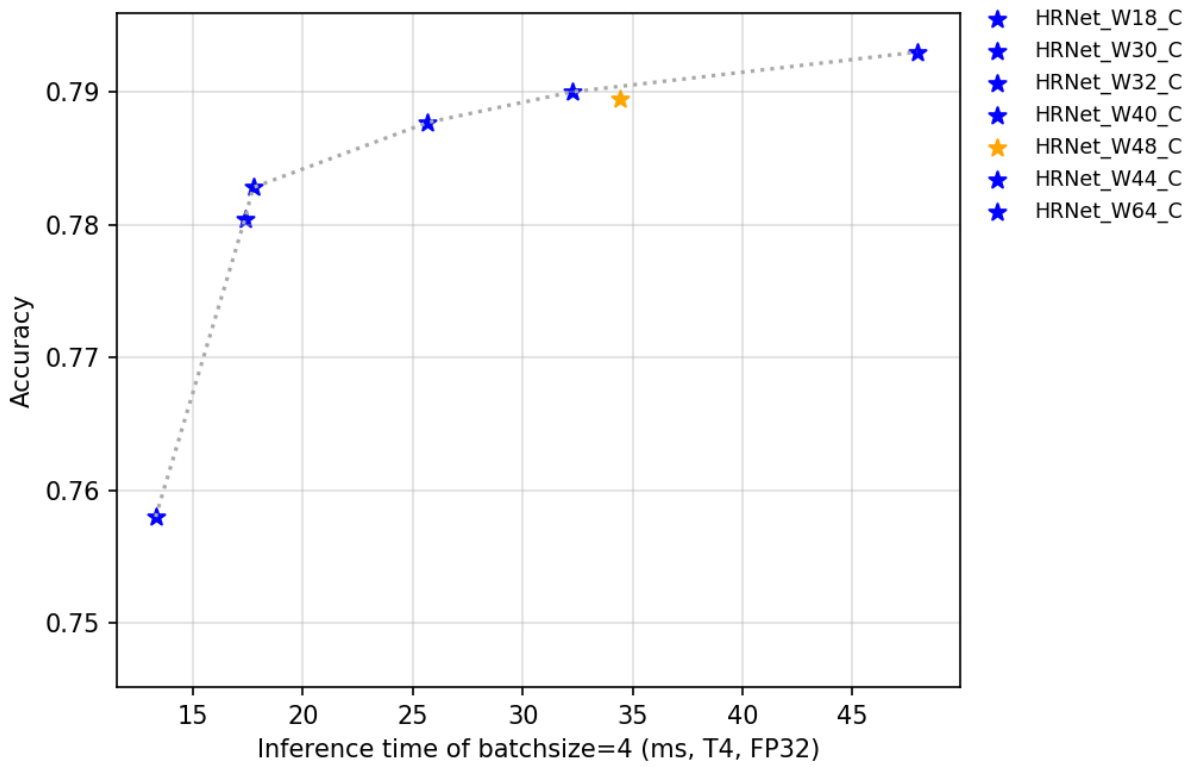
2.7.1 Overview

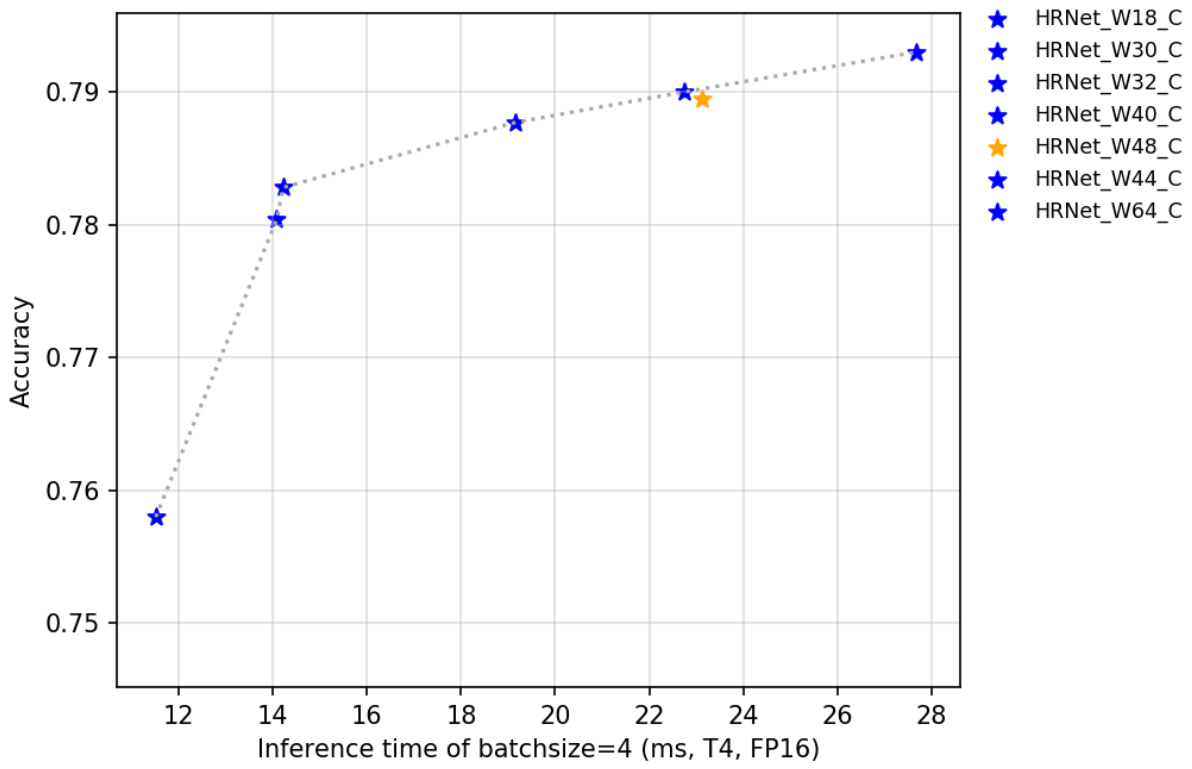
HRNet is a brand new neural network proposed by Microsoft research Asia in 2019. Different from the previous convolutional neural network, this network can still maintain high resolution in the deep layer of the network, so the heat map of the key points predicted is more accurate, and it is also more accurate in space. In addition, the network performs particularly well in other visual tasks sensitive to resolution, such as detection and segmentation.

The FLOPS, parameters, and inference time on the T4 GPU of this series of models are shown in the figure below.


HRNet_W18_C
HRNet_W30_C
HRNet_W32_C
HRNet_W40_C
HRNet_W48_C
HRNet_W44_C
HRNet_W64_C
Accuracy
0.79
0.78
0.77
0.76
0.75
10
20
30
40
50
60
FLOPS(G)


HRNet_W18_C
HRNet_W30_C
HRNet_W32_C
HRNet_W40_C
HRNet_W48_C
HRNet_W44_C
HRNet_W64_C
Accuracy
0.79
0.78
0.77
0.76
0.75
20
40
60
80
100
120
Parameters(M)


HRNet_W18_C
HRNet_W30_C
HRNet_W32_C
HRNet_W40_C
HRNet_W48_C
HRNet_W44_C
HRNet_W64_C
Accuracy
0.79
0.78
0.77
0.76
0.75
15
20
25
30
35
40
45
Inference time of batchsize=4 (ms, T4, FP32)



At present, there are 7 pretrained models of such models open-sourced by PaddleClas, and their indicators are shown in the figure. Among them, the reason why the accuracy of the HRNet_W48_C indicator is abnormal may be due to fluctuations in training.

2.7.2 Accuracy, FLOPS and Parameters

2.7.3 Inference speed based on V100 GPU

2.7.4 Inference speed based on T4 GPU

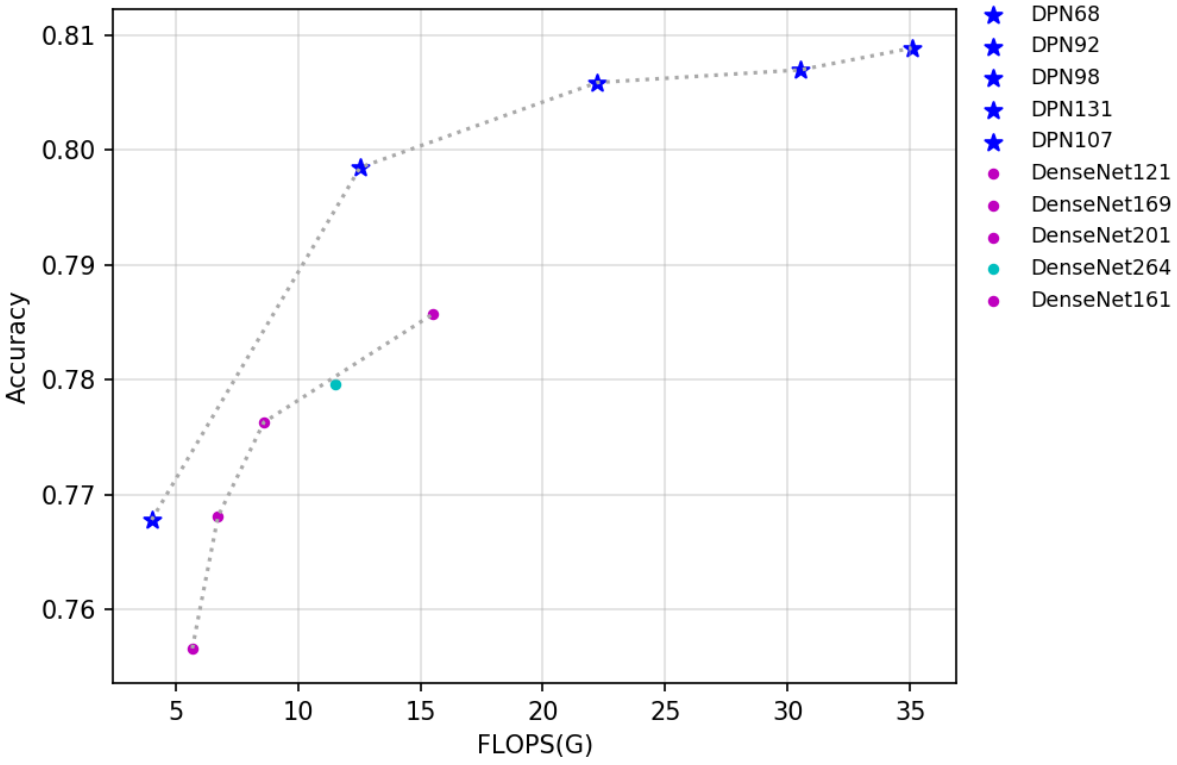
2.8 DPN and DenseNet series

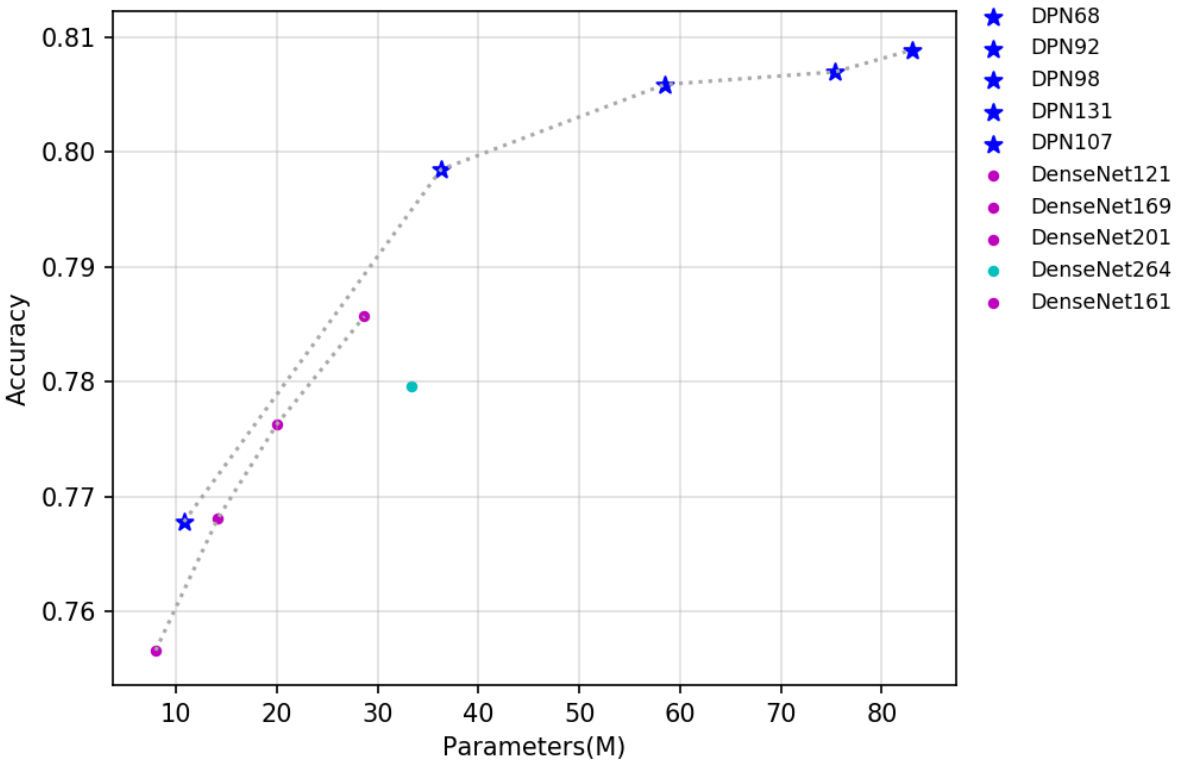
2.8.1 Overview

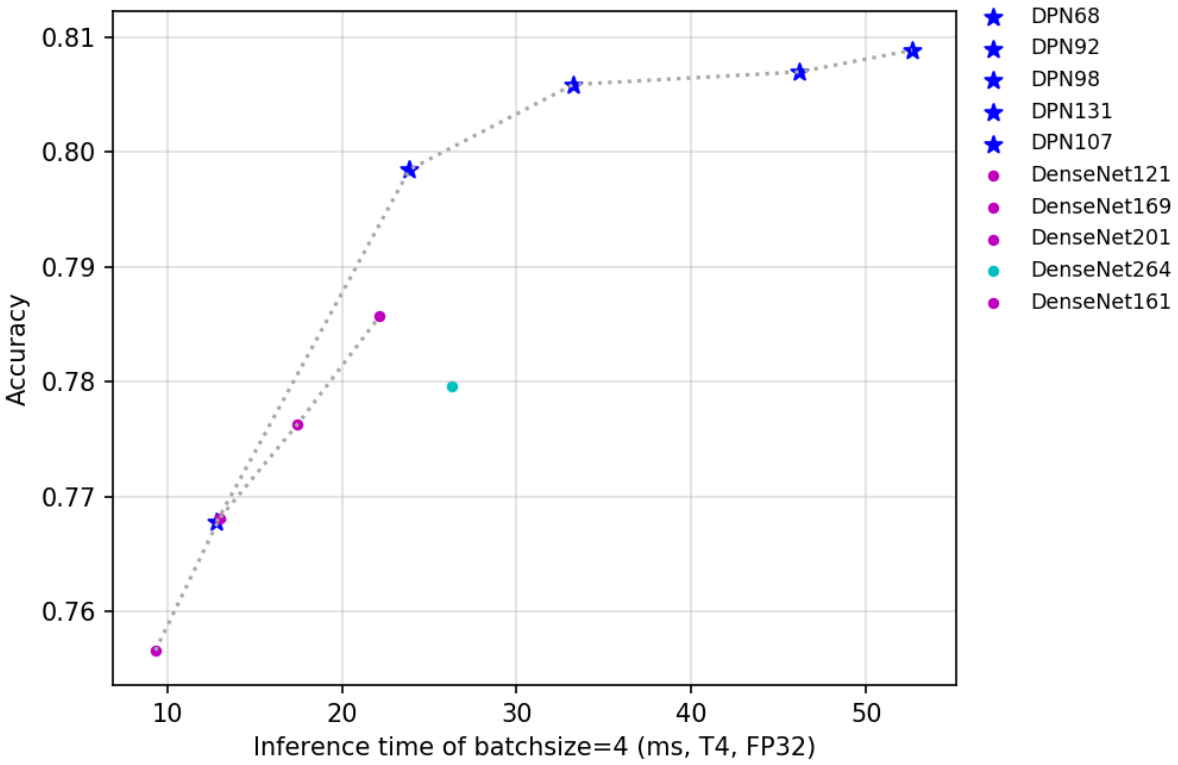
DenseNet is a new network structure proposed in 2017 and was the best paper of CVPR. The network has designed a new cross-layer connected block called dense-block. Compared to the bottleneck in ResNet, dense-block has designed a more aggressive dense connection module, that is, connecting all the layers to each other, and each layer will accept all the layers in front of it as its additional input. DenseNet stacks all dense-blocks into a densely connected network. The dense connection makes DenseNet easier to backpropagate, making the network easier to train and converge. The full name of DPN is Dual Path Networks, which is a network composed of DenseNet and ResNeXt, which proves that DenseNet can extract new features from the previous level, and ResNeXt essentially reuses the extracted features . The author further analyzes and finds that ResNeXt has high reuse rate for features, but low redundancy, while DenseNet can create new features, but with high redundancy. Combining the advantages of the two structures, the

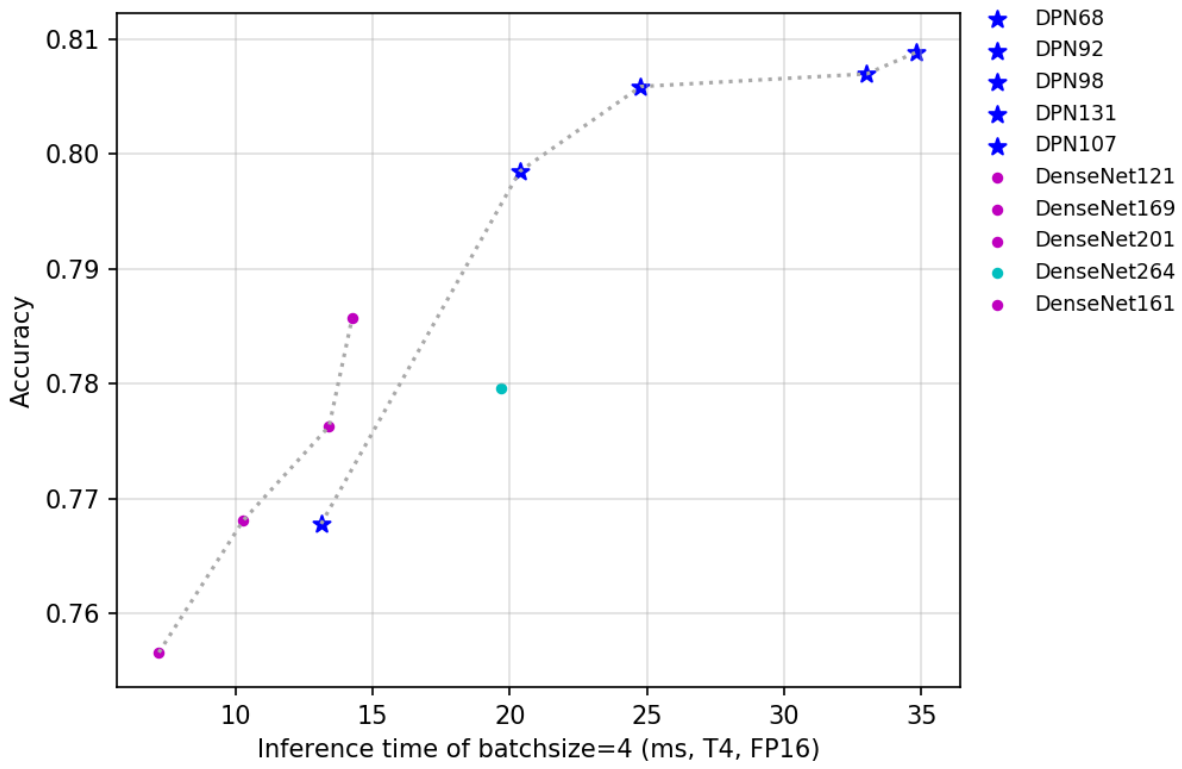
author designed the DPN network. In the end, the DPN network achieved better results than ResNeXt and DenseNet under the same FLOPS and parameters.

The FLOPS, parameters, and inference time on the T4 GPU of this series of models are shown in the figure below.




0.81
0.80
0.79
0.78
0.77
0.76
Accuracy
10
20
30
40
50
60
70
80
Parameters(M)
DPN68
DPN92
DPN98
DPN131
DPN107
DenseNet121
DenseNet169
DenseNet201
DenseNet264
DenseNet161


0.81
0.80
0.79
0.78
0.77
0.76
Accuracy
10
20
30
40
50
Inference time of batchsize=4 (ms, T4, FP32)
DPN68
DPN92
DPN98
DPN131
DPN107
DenseNet121
DenseNet169
DenseNet201
DenseNet264
DenseNet161



The pretrained models of these two types of models (a total of 10) are open sourced in PaddleClas at present. The indicators are shown in the figure above. It is easy to observe that under the same FLOPS and parameters, DPN has higher accuracy than DenseNet. However,because DPN has more branches, its inference speed is slower than DenseNet. Since DenseNet264 has the deepest layers in all DenseNet networks, it has the largest parameters,DenseNet161 has the largest width, resulting the largest FLOPs and the highest accuracy in this series. From the perspective of inference speed, DenseNet161, which has a large FLOPs and high accuracy, has a faster speed than DenseNet264, so it has a greater advantage than DenseNet264.

For DPN series networks, the larger the model's FLOPs and parameters, the higher the model's accuracy. Among them, since the width of DPN107 is the largest, it has the largest number of parameters and FLOPs in this series of networks.

2.8.2 Accuracy, FLOPS and Parameters

2.8.3 Inference speed based on V100 GPU

2.8.4 Inference speed based on T4 GPU

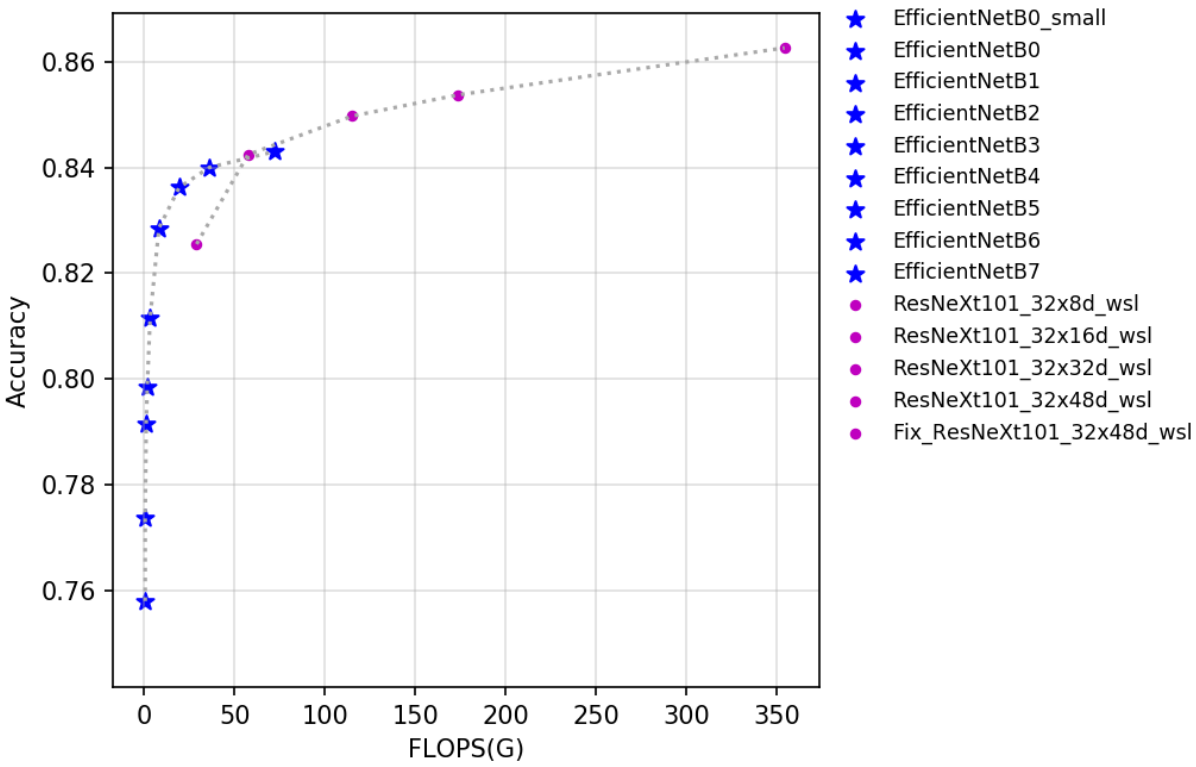
2.9 EfficientNet and ResNeXt101_wsl series

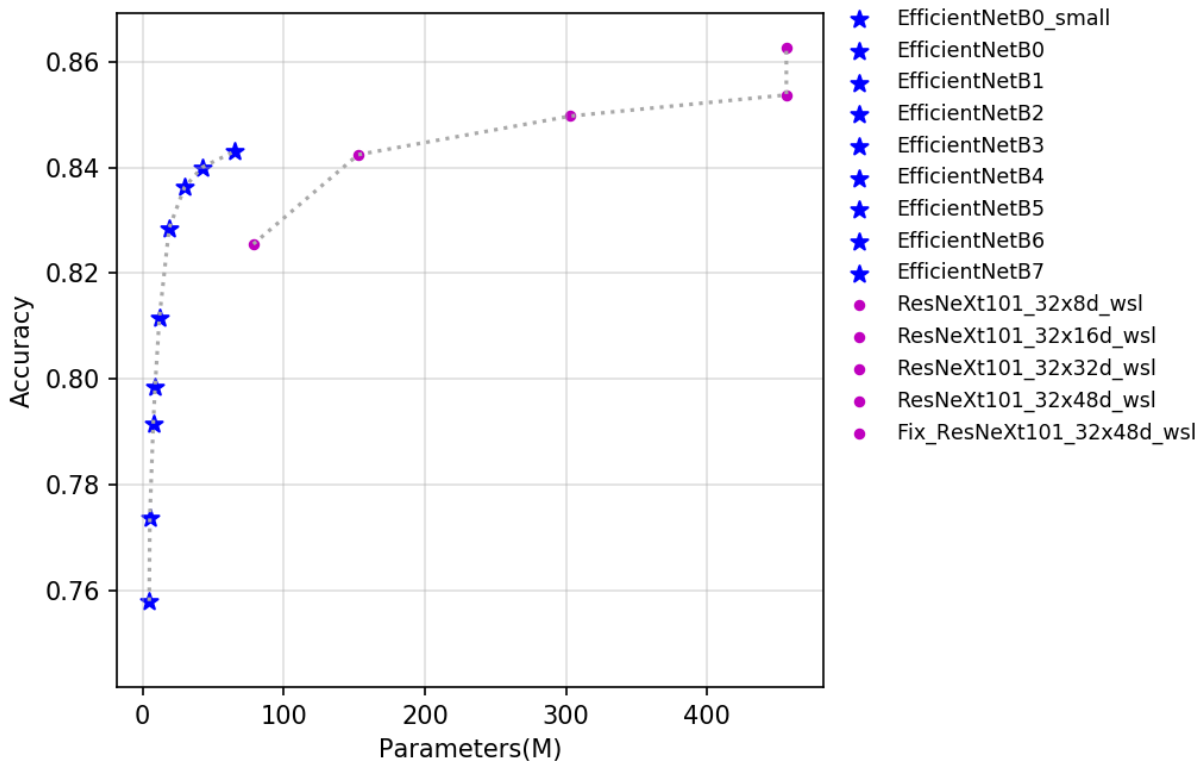
2.9.1 Overview

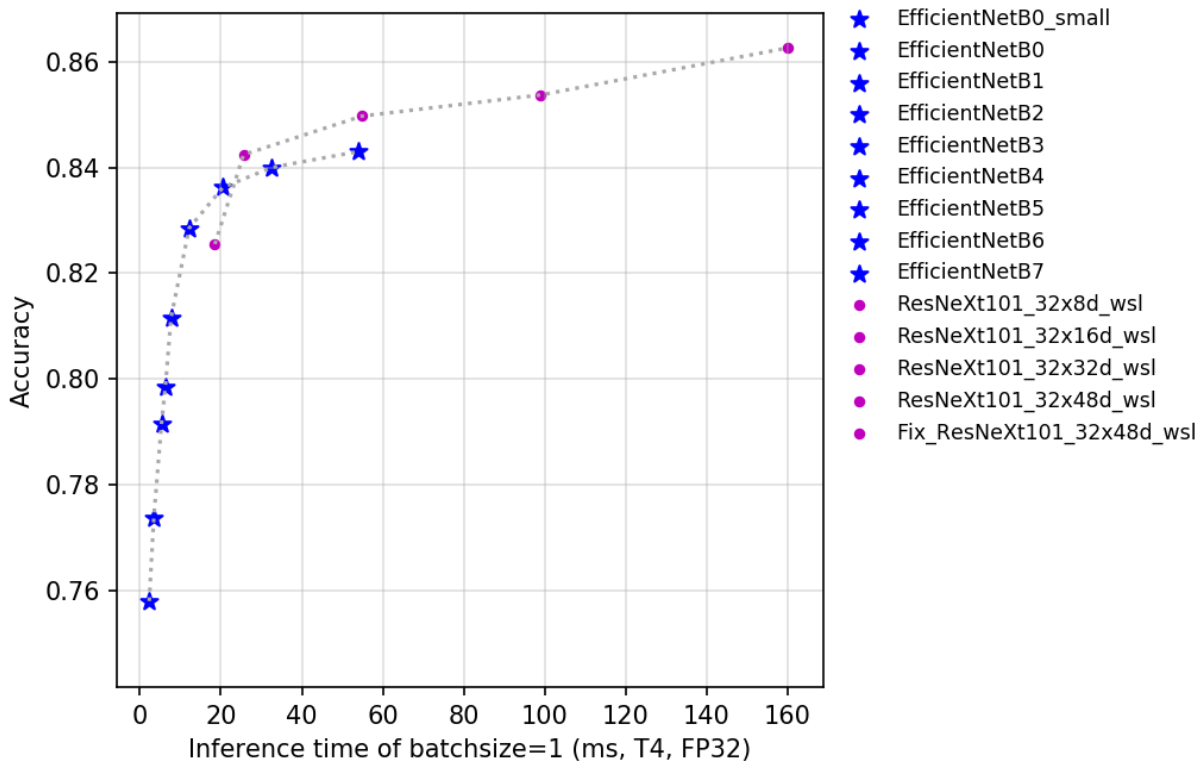
EfficientNet is a lightweight NAS-based network released by Google in 2019. EfficientNetB7 refreshed the classification accuracy of ImageNet-1k at that time. In this paper, the author points out that the traditional methods to improve the performance of neural networks mainly start with the width of the network, the depth of the network, and the resolution of the input picture. However, the author found that balancing these three dimensions is essential for improving accuracy and efficiency through experiments. Therefore, the author summarized how to balance the three dimensions at the same time through a series of experiments. At the same time, based on this scaling method, the author built a total of 7 networks B1-B7 in the EfficientNet series on the basis of EfficientNetB0, and with the same FLOPS and parameters, the accuracy reached state-of-the-art effect.

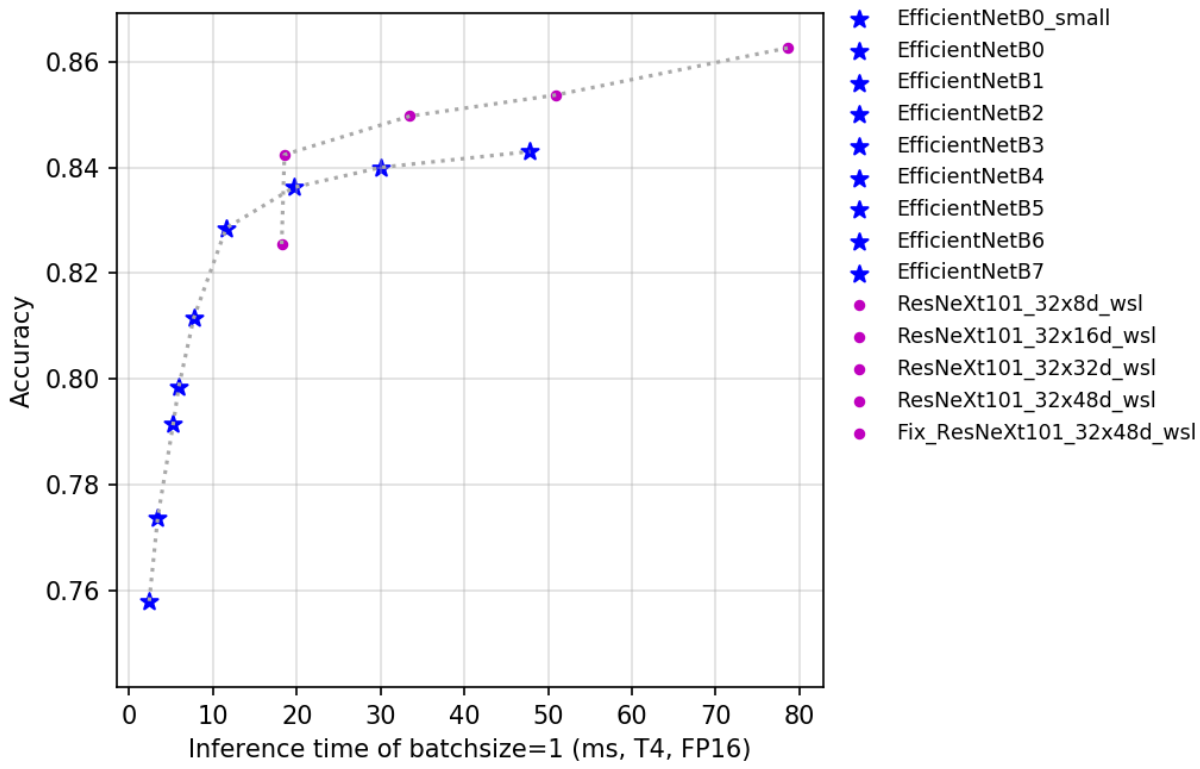
ResNeXt is an improved version of ResNet that proposed by Facebook in 2016. In 2019, Facebook researchers studied the accuracy limit of the series network on ImageNet through weakly-supervised-learning. In order to distinguish the previous ResNeXt network, the suffix of this series network is WSL, where WSL is the abbreviation of weakly-supervised-learning. In order to have stronger feature extraction capability, the researchers further enlarged the network width, among which the largest ResNeXt101_32x48d_wsl has 800 million parameters. It was trained under 940 million weak-labeled images, and the results were finetune trained on imagenet-1k. Finally, the acc-1 of imagenet-1k reaches 85.4%, which is also the network with the highest precision under the resolution of 224x224 on imagenet-1k so far. In Fix-ResNeXt, the author used a larger image resolution, made a special Fix strategy for the inconsistency of image data preprocessing in training and testing, and made ResNeXt101_32x48d_wsl have a higher accuracy. Since it used the Fix strategy, it was named Fix-ResNeXt101_32x48d_wsl.

The FLOPS, parameters, and inference time on the T4 GPU of this series of models are shown in the figure below.


0.86
0.84
0.82
0.80
0.78
0.76
Accuracy
0
50
100
150
200
250
300
350
FLOPS(G)
EfficientNetB0_small
EfficientNetB0
EfficientNetB1
EfficientNetB2
EfficientNetB3
EfficientNetB4
EfficientNetB5
EfficientNetB6
EfficientNetB7
ResNeXt101_32x8d_wsl
ResNeXt101_32x16d_wsl
ResNeXt101_32x32d_wsl
ResNeXt101_32x48d_wsl
Fix_ResNeXt101_32x48d_wsl


0.86
0.84
0.82
0.80
0.78
0.76
Accuracy
0
100
200
300
400
Parameters(M)
EfficientNetB0_small
EfficientNetB0
EfficientNetB1
EfficientNetB2
EfficientNetB3
EfficientNetB4
EfficientNetB5
EfficientNetB6
EfficientNetB7
ResNeXt101_32x8d_wsl
ResNeXt101_32x16d_wsl
ResNeXt101_32x32d_wsl
ResNeXt101_32x48d_wsl
Fix_ResNeXt101_32x48d_wsl


0.86
0.84
0.82
0.80
0.78
0.76
Accuracy
0
20
40
60
80
100
120
140
160
Inference time of batchsize=1 (ms, T4, FP32)
EfficientNetB0_small
EfficientNetB0
EfficientNetB1
EfficientNetB2
EfficientNetB3
EfficientNetB4
EfficientNetB5
EfficientNetB6
EfficientNetB7
ResNeXt101_32x8d_wsl
ResNeXt101_32x16d_wsl
ResNeXt101_32x32d_wsl
ResNeXt101_32x48d_wsl
Fix_ResNeXt101_32x48d_wsl



At present, there are a total of 14 pretrained models of the two types of models that PaddleClas open source. It can be seen from the above figure that the advantages of the EfficientNet series network are very obvious. The ResNeXt101_wsl series model uses more data, and the final accuracy is also higher. EfficientNet_B0_small removes SE_block based on EfficientNet_B0, which has faster inference speed.

2.9.2 Accuracy, FLOPS and Parameters

2.9.3 Inference speed based on V100 GPU

2.9.4 Inference speed based on T4 GPU

2.10 Other networks

2.10.1 Overview

In 2012, AlexNet network proposed by Alex et al. won the ImageNet competition by far surpassing the second place, and the convolutional neural network and even deep learning attracted wide attention. AlexNet used relu as the activation function of CNN to solve the gradient dispersion problem of sigmoid when the network is deep. During the training, Dropout was used to randomly lose a part of the neurons, avoiding the overfitting of the model. In the network, overlapping maximum pooling is used to replace the average pooling commonly used in CNN, which avoids the fuzzy effect of average pooling and improves the feature richness. In a sense, AlexNet has exploded the research and application of neural networks.

SqueezeNet achieved the same precision as AlexNet on Imagenet-1k, but only with 1/50 parameters. The core of the network is the Fire module, which used the convolution of 1x1 to achieve channel dimensionality reduction, thus greatly saving the number of parameters. The author created SqueezeNet by stacking a large number of Fire modules.

VGG is a convolutional neural network developed by researchers at Oxford University's Visual Geometry Group and DeepMind. The network explores the relationship between the depth of the convolutional neural network and its performance. By repeatedly stacking the small convolutional kernel of 3x3 and the maximum pooling layer of 2x2, the multi-layer convolutional neural network is successfully constructed and has achieved good convergence accuracy. In the end, VGG won the runner-up of ILSVRC 2014 classification and the champion of positioning.

DarkNet53 is designed for object detection by YOLO author in the paper. The network is basically composed of 1x1 and 3x3 kernel, with a total of 53 layers, named DarkNet53.

2.10.2 Accuracy, FLOPS and Parameters

2.10.3 Inference speed based on V100 GPU

2.10.4 Inference speed based on T4 GPU

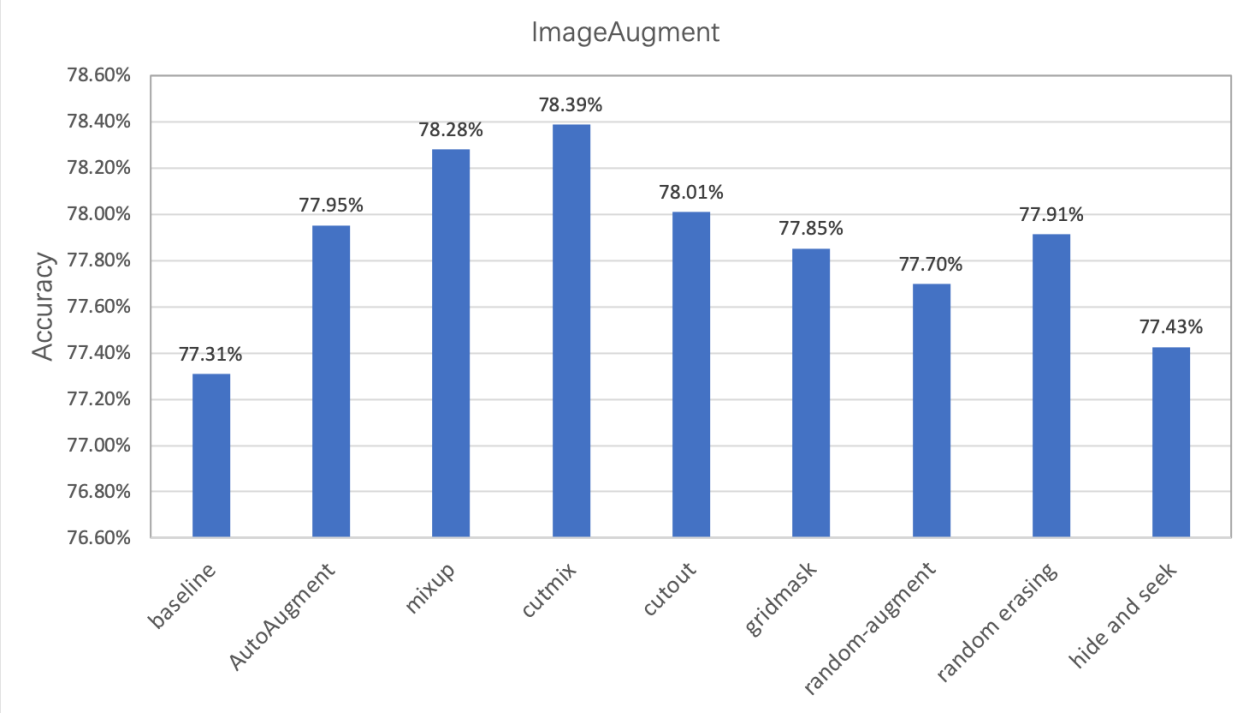
CHAPTER 3

advanced_tutorials

3.1 image_augmentation

3.1.1 Image Augmentation

Image augmentation is a commonly used regularization method in image classification task, which is often used in scenarios with insufficient data or large model. In this chapter, we mainly introduce 8 image augmentation methods besides standard augmentation methods. Users can apply these methods in their own tasks for better model performance. Under the same conditions, These augmentation methods' performance on ImageNet1k dataset is shown as follows.



3.1.2 Common image augmentation methods

If without special explanation, all the examples and experiments in this chapter are based on ImageNet1k dataset with the network input image size set as 224.

The standard data augmentation pipeline in ImageNet classification tasks contains the following steps.

1. Decode image, abbreviated as `ImageDecode`.
2. Randomly crop the image to size with 224x224, abbreviated as `RandCrop`.
3. Randomly flip the image horizontally, abbreviated as `RandFlip`.
4. Normalize the image pixel values, abbreviated as `Normalize`.
5. Transpose the image from `[224, 224, 3]`(HWC) to `[3, 224, 224]`(CHW), abbreviated as `Transpose`.
6. Group the image data(`[3, 224, 224]`) into a batch(`[N, 3, 224, 224]`), where `N` is the batch size. It is abbreviated as `Batch`.

Compared with the above standard image augmentation methods, the researchers have also proposed many improved image augmentation strategies. These strategies are to insert certain operations at different stages of the standard augmentation method, based on the different stages of operation. We divide it into the following three categories.

1. Transformation. Perform some transformations on the image after `RandCrop`, such as AutoAugment and RandAugment.
2. Cropping. Perform some transformations on the image after `Transpose`, such as CutOut, RandErasing, HideAndSeek and GridMask.
3. Aliasing. Perform some transformations on the image after `Batch`, such as Mixup and Cutmix.

The following table shows more detailed information of the transformations.

PaddleClas integrates all the above data augmentation strategies. More details including principles and usage of the strategies are introduced in the following chapters. For better visualization, we use the following figure to show the changes after the transformations. And `RandCrop` is replaced with`Resize` for simplification.

3.1.3 Image Transformation

Transformation means performing some transformations on the image after `RandCrop`. It mainly contains AutoAugment and RandAugment.

AutoAugment

Addresshttps://arxiv.org/abs/1805.09501v1

Github repohttps://github.com/DeepVoltaire/AutoAugment

Unlike conventional artificially designed image augmentation methods, AutoAugment is an image augmentation solution suitable for a specific data set found by certain search algorithm in the search space of a series of image augmentation sub-strategies. For the ImageNet dataset, the final data augmentation solution contains 25 sub-strategy combinations. Each sub-strategy contains two transformations. For each image, a sub-strategy combination is randomly selected and then determined with a certain probability Perform each transformation in the sub-strategy.

In PaddleClas, `AutoAugment` is used as follows.

```
from ppcls.data.imaug import DecodeImage
from ppcls.data.imaug import ResizeImage
from ppcls.data.imaug import ImageNetPolicy
from ppcls.data.imaug import transform

size = 224

decode_op = DecodeImage()
resize_op = ResizeImage(size=(size, size))
autoaugment_op = ImageNetPolicy()

ops = [decode_op, resize_op, autoaugment_op]

imgs_dir = image_path
fnames = os.listdir(imgs_dir)
```

(continues on next page)

(continued from previous page)

```
for f in fnames:
    data = open(os.path.join(imgs_dir, f)).read()
    img = transform(data, ops)
```

The images after AutoAugment are as follows.

RandAugment

Address: https://arxiv.org/pdf/1909.13719.pdf

Github repo: https://github.com/heartInsert/randaugment

The search method of AutoAugment is relatively violent. Searching for the optimal strategy for this data set directly on the data set requires a lot of computation. In RandAugment, the author found that on the one hand, for larger models and larger datasets, the gains generated by the augmentation method searched using AutoAugment are smaller. On the other hand, the searched strategy is limited to certain dataset, which has poor generalization performance and not sutable for other datasets.

In RandAugment, the author proposes a random augmentation method. Instead of using a specific probability to determine whether to use a certain sub-strategy, all sub-strategies are selected with the same probability. The experiments in the paper also show that this method performs well even for large models.

In PaddleClas, RandAugment is used as follows.

```
from ppcls.data.imaug import DecodeImage
from ppcls.data.imaug import ResizeImage
from ppcls.data.imaug import RandAugment
from ppcls.data.imaug import transform

size = 224

decode_op = DecodeImage()
resize_op = ResizeImage(size=(size, size))
randaugment_op = RandAugment()

ops = [decode_op, resize_op, randaugment_op]

imgs_dir = image_path
fnames = os.listdir(imgs_dir)
```

(continues on next page)

(continued from previous page)

```
for f in fnames:
    data = open(os.path.join(imgs_dir, f)).read()
    img = transform(data, ops)
```

The images after `RandAugment` are as follows.

3.1.4 Image Cropping

Cropping means performing some transformations on the image after `Transpose`, setting pixels of the cropped area as certain constant. It mainly contains CutOut, RandErasing, HideAndSeek and GridMask.

Image cropping methods can be operated before or after normalization. The difference is that if we crop the image before normalization and fill the areas with 0, the cropped areas' pixel values will not be 0 after normalization, which will cause grayscale distribution change of the data.

The above-mentioned cropping transformation ideas are the similar, all to solve the problem of poor generalization ability of the trained model on occlusion images, the difference lies in that their cropping details.

Cutout

Address: https://arxiv.org/abs/1708.04552

Github repo: https://github.com/uoguelph-mlrg/Cutout

Cutout is a kind of dropout, but occludes input image rather than feature map. It is more robust to noise than noise. Cutout has two advantages: (1) Using Cutout, we can simulate the situation when the subject is partially occluded. (2) It can promote the model to make full use of more content in the image for classification, and prevent the network from focusing only on the saliency area, thereby causing overfitting.

In PaddleClas, `Cutout` is used as follows.

```
from ppcls.data.imaug import DecodeImage
from ppcls.data.imaug import ResizeImage
from ppcls.data.imaug import Cutout
from ppcls.data.imaug import transform

size = 224
```

(continues on next page)

(continued from previous page)

```
decode_op = DecodeImage()
resize_op = ResizeImage(size=(size, size))
cutout_op = Cutout(n_holes=1, length=112)

ops = [decode_op, resize_op, cutout_op]

imgs_dir = image_path
fnames = os.listdir(imgs_dir)
for f in fnames:
    data = open(os.path.join(imgs_dir, f)).read()
    img = transform(data, ops)
```

The images after `Cutout` are as follows.

RandomErasing

Address: https://arxiv.org/pdf/1708.04896.pdf

Github repo: https://github.com/zhunzhong07/Random-Erasing

RandomErasing is similar to the Cutout. It is also to solve the problem of poor generalization ability of the trained model on images with occlusion. The author also pointed out in the paper that the way of random cropping is complementary to random horizontal flipping. The author also verified the effectiveness of the method on pedestrian re-identification (REID). Unlike `Cutout`, in, `RandomErasing` is operateed on the image with a certain probability, size and aspect ratio of the generated mask are also randomly generated according to pre-defined hyperparameters.

In PaddleClas, `RandomErasing` is used as follows.

```
from ppcls.data.imaug import DecodeImage
from ppcls.data.imaug import ResizeImage
from ppcls.data.imaug import ToCHWImage
from ppcls.data.imaug import RandomErasing
from ppcls.data.imaug import transform

size = 224

decode_op = DecodeImage()
resize_op = ResizeImage(size=(size, size))
```

(continues on next page)

(continued from previous page)

```
randomerasing_op = RandomErasing()

ops = [decode_op, resize_op, tochw_op, randomerasing_op]

imgs_dir = image_path
fnames = os.listdir(imgs_dir)
for f in fnames:
    data = open(os.path.join(imgs_dir, f)).read()
    img = transform(data, ops)
    img = img.transpose((1, 2, 0))
```

The images after `RandomErasing` are as follows.

HideAndSeek

Address: https://arxiv.org/pdf/1811.02545.pdf

Github repo: https://github.com/kkanshul/Hide-and-Seek

Images are divided into some patches for `HideAndSeek` and masks are generated with certain probability for each patch. The meaning of the masks in different areas is shown in the figure below.

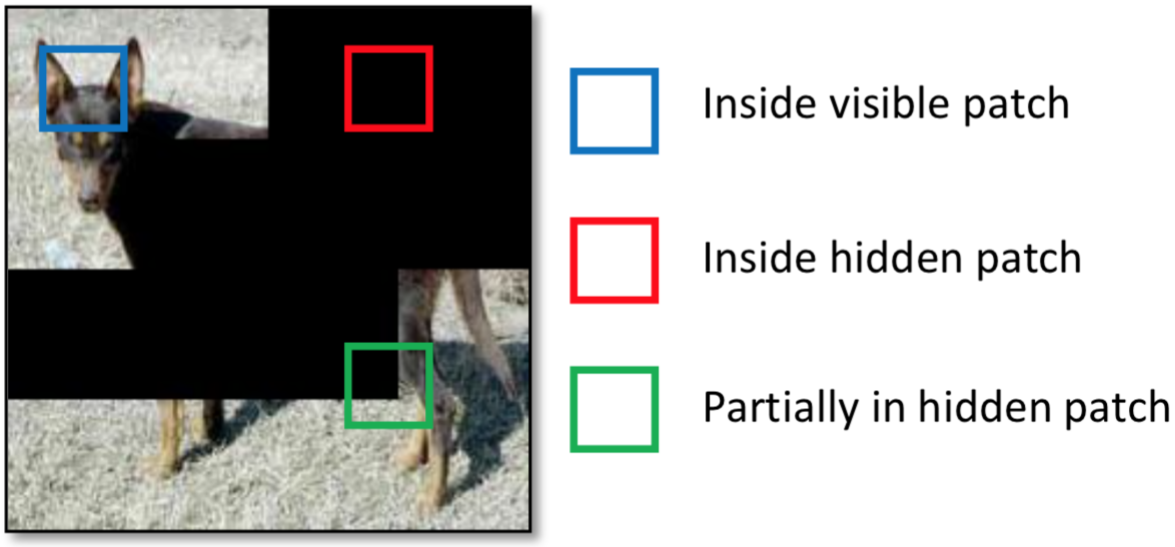


Fig. 3. There are three types of convolutional filter activations after hiding patches: a convolution filter can be completely within a visible region (blue box), completely within a hidden region (red box), or partially within a visible/hidden region (green box).

In PaddleClas, `HideAndSeek` is used as follows.

```
from ppcls.data.imaug import DecodeImage
from ppcls.data.imaug import ResizeImage
from ppcls.data.imaug import ToCHWImage
from ppcls.data.imaug import HideAndSeek
from ppcls.data.imaug import transform

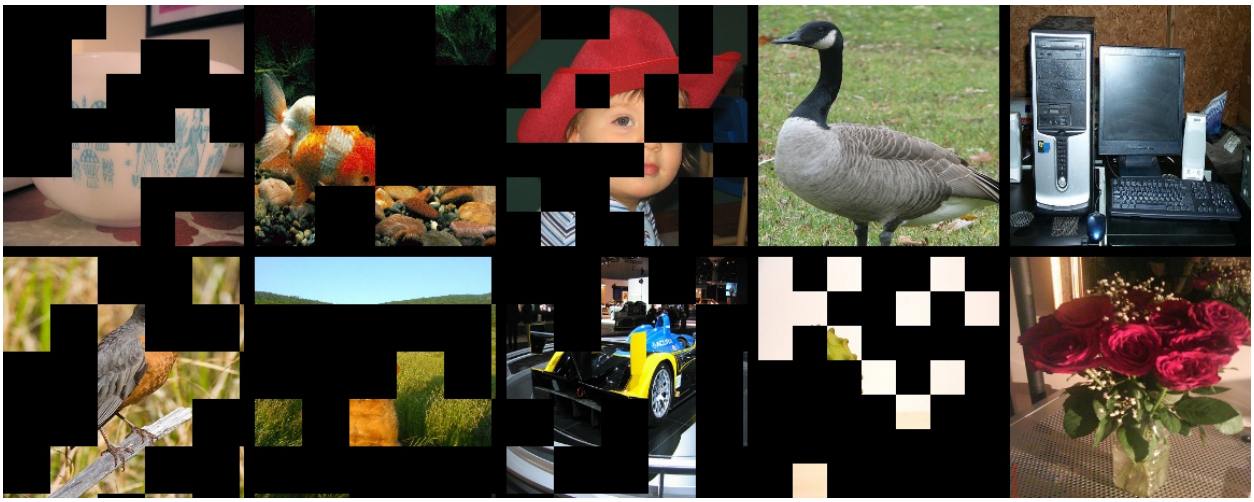
size = 224

decode_op = DecodeImage()
resize_op = ResizeImage(size=(size, size))
hide_and_seek_op = HideAndSeek()

ops = [decode_op, resize_op, tochw_op, hide_and_seek_op]

imgs_dir = image_path
fnames = os.listdir(imgs_dir)
for f in fnames:
    data = open(os.path.join(imgs_dir, f)).read()
    img = transform(data, ops)
    img = img.transpose((1, 2, 0))
```

The images after `HideAndSeek` are as follows.



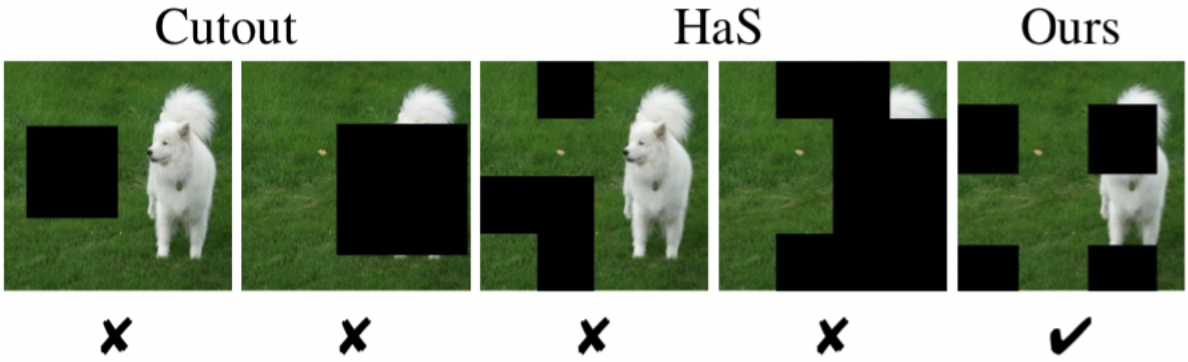
GridMask

Addresshttps://arxiv.org/abs/2001.04086

Github repohttps://github.com/akuxcw/GridMask

The author points out that the previous method based on image cropping has two problems, as shown in the following figure:

1. Excessive deletion of the area may cause most or all of the target subject to be deleted, or cause the context information loss, resulting in the images after enhancement becoming noisy data.
2. Reserving too much area has little effect on the object and context.



Therefore, it is the core problem to be solved how to if you avoid over-deletion or over-retention becomes the core problem to be solved.

`GridMask` is to generate a mask with the same resolution as the original image and multiply it with the original image. The mask grid and size are adjusted by the hyperparameters.

In the training process, there are two methods to use:

1. Set a probability p and use the GridMask to augment the image with probability p from the beginning of training.
2. Initially set the augmentation probability to 0, and the probability is increased with number of iterations from 0 to p.

It shows that the second method is better.

The usage of `GridMask` in PaddleClas is shown below.

```
from data.imaug import DecodeImage
from data.imaug import ResizeImage
from data.imaug import ToCHWImage
from data.imaug import GridMask
from data.imaug import transform

size = 224

decode_op = DecodeImage()
resize_op = ResizeImage(size=(size, size))
tochw_op = ToCHWImage()
gridmask_op = GridMask(d1=96, d2=224, rotate=1, ratio=0.6, mode=1, prob=0.8)

ops = [decode_op, resize_op, tochw_op, gridmask_op]

imgs_dir = image_path
fnames = os.listdir(imgs_dir)
for f in fnames:
    data = open(os.path.join(imgs_dir, f)).read()
    img = transform(data, ops)
    img = img.transpose((1, 2, 0))
```

The images after `GridMask` are as follows.

3.1.5 Image aliasing

Aliasing means performing some transformations on the image after `Batch`, which contains Mixup and Cutmix.

Data augmentation methods introduced before are based on single image while aliasing is carried on a certain batch to generate a new batch.

Mixup

Address: https://arxiv.org/pdf/1710.09412.pdf

Github repo: https://github.com/facebookresearch/mixup-cifar10

Mixup is the first solution for image aliasing, it is easy to realize and performs well not only on image classification but also on object detection. Mixup is usually carried out in a batch for simplification, so as `Cutmix`.

The usage of `Mixup` in PaddleClas is shown below.

```
from ppcls.data.imaug import DecodeImage
from ppcls.data.imaug import ResizeImage
from ppcls.data.imaug import ToCHWImage
from ppcls.data.imaug import transform
from ppcls.data.imaug import MixupOperator

size = 224

decode_op = DecodeImage()
resize_op = ResizeImage(size=(size, size))
tochw_op = ToCHWImage()
hide_and_seek_op = HideAndSeek()
mixup_op = MixupOperator()
cutmix_op = CutmixOperator()

ops = [decode_op, resize_op, tochw_op]

imgs_dir = image_path

batch = []
fnames = os.listdir(imgs_dir)
for idx, f in enumerate(fnames):
    data = open(os.path.join(imgs_dir, f)).read()
    img = transform(data, ops)
    batch.append( (img, idx) ) # fake label

new_batch = mixup_op(batch)
```

The images after `Mixup` are as follows.

Cutmix

Address: https://arxiv.org/pdf/1905.04899v2.pdf

Github repo: https://github.com/clovaai/CutMix-PyTorch

Unlike `Mixup` which directly adds two images, for Cutmix, an `ROI` is cut out from one image and Cutmix randomly cuts out an `ROI` from one image, and then covered onto the corresponding area in the another image. The usage of `Cutmix` in PaddleClas is shown below.

```
rom ppcls.data.imaug import DecodeImage
from ppcls.data.imaug import ResizeImage
from ppcls.data.imaug import ToCHWImage
from ppcls.data.imaug import transform
from ppcls.data.imaug import CutmixOperator

size = 224

decode_op = DecodeImage()
resize_op = ResizeImage(size=(size, size))
tochw_op = ToCHWImage()
hide_and_seek_op = HideAndSeek()
cutmix_op = CutmixOperator()

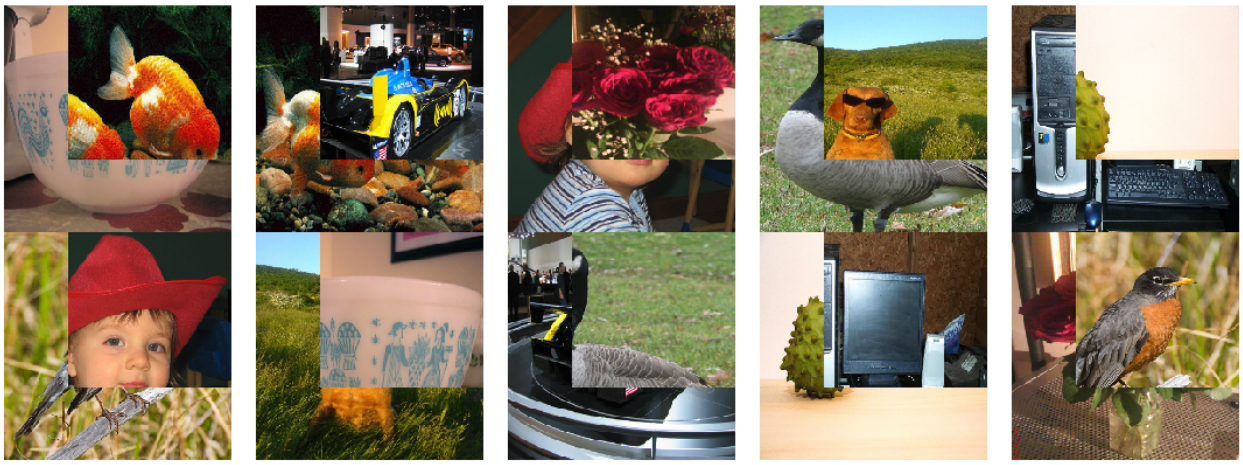
ops = [decode_op, resize_op, tochw_op]

imgs_dir = image_path

batch = []
fnames = os.listdir(imgs_dir)
for idx, f in enumerate(fnames):
    data = open(os.path.join(imgs_dir, f)).read()
    img = transform(data, ops)
    batch.append( (img, idx) ) # fake label

new_batch = cutmix_op(batch)
```

The images after `Cutmix` are as follows.



3.1.6 Experiments

Based on PaddleClas, Metrics of different augmentation methods on ImageNet1k dataset are as follows.

note:

- In the experiment here, for better comparison, we fixed the l2 decay to 1e-4. To achieve higher accuracy, we recommend trying to use a smaller l2 decay. Combined with data augmentaton, we found that reducing l2 decay from 1e-4 to 7e-5 can bring at least 0.3~0.5% accuracy improvement.
- We have not yet combined different strategies or verified, whch is our future work.

Data augmentation practice

Experiments about data augmentation will be introduced in detail in this section. If you want to quickly experience these methods, please refer to *Quick start PaddleClas in 30 miniutes*.

Configurations

Since hyperparameters differ from different augmentation methods. For better understanding, we list 8 augmentation configuration files in `configs/DataAugment` based on ResNet50. Users can train the model with `tools/run.sh`. The following are 3 of them.

RandAugment

Configuration of `RandAugment` is shown as follows. `Num_layers`(default as 2) and `magnitude`(default as 5) are two hyperparameters.

```
transforms:
    - DecodeImage:
        to_rgb: True
        to_np: False
        channel_first: False
    - RandCropImage:
        size: 224
    - RandFlipImage:
        flip_code: 1
    - RandAugment:
        num_layers: 2
        magnitude: 5
    - NormalizeImage:
        scale: 1./255.
        mean: [0.485, 0.456, 0.406]
        std: [0.229, 0.224, 0.225]
        order: ''
    - ToCHWImage:
```

Cutout

Configuration of `Cutout` is shown as follows. `n_holes`(default as 1) and `n_holes`(default as 112) are two hyperparameters.

```
transforms:
    - DecodeImage:
        to_rgb: True
        to_np: False
        channel_first: False
    - RandCropImage:
        size: 224
    - RandFlipImage:
        flip_code: 1
    - NormalizeImage:
        scale: 1./255.
        mean: [0.485, 0.456, 0.406]
        std: [0.229, 0.224, 0.225]
```

(continues on next page)

(continued from previous page)

```
            order: ''
        - Cutout:
            n_holes: 1
            length: 112
        - ToCHWImage:
```

Mixup

Configuration of `Mixup` is shown as follows. `alpha`(default as 0.2) is hyperparameter which users need to care about. What's more, `use_mix` need to be set as `True` in the root of the configuration.

```
    transforms:
        - DecodeImage:
            to_rgb: True
            to_np: False
            channel_first: False
        - RandCropImage:
            size: 224
        - RandFlipImage:
            flip_code: 1
        - NormalizeImage:
            scale: 1./255.
            mean: [0.485, 0.456, 0.406]
            std: [0.229, 0.224, 0.225]
            order: ''
        - ToCHWImage:
    mix:
        - MixupOperator:
            alpha: 0.2
```

Users can use the following command to start the training process, which can also be referred to `tools/run.sh`.

```
export PYTHONPATH=path_to_PaddleClas:$PYTHONPATH

python -m paddle.distributed.launch \
    --selected_gpus="0,1,2,3" \
    tools/train.py \
        -c ./configs/DataAugment/ResNet50_Cutout.yaml
```

Note

- When using augmentation methods based on image aliasing, users need to set `use_mix` in the configuration file as `True`. In addition, because the label needs to be aliased when the image is aliased, the accuracy of the training data cannot be calculated. The training accuracy rate was not printed during the training process.
- The training data is more difficult with data augmentation, so the training loss may be larger, the training set accuracy is relatively low, but it has better generalization ability, so the validation set accuracy is relatively higher.
- After the use of data augmentation, the model may tend to be underfitting. It is recommended to reduce `l2_decay` for better performance on validation set.

- hyperparameters exist in almost all agmenatation methods. Here we provide hyperparameters for ImageNet1k dataset. User may need to finetune the hyperparameters on specified dataset. More training tricks can be referred to **Tricks**.

  If this document is helpful to you, welcome to star our project: https://github.com/PaddlePaddle/PaddleClas

3.1.7 Reference

[1] Cubuk E D, Zoph B, Mane D, et al. Autoaugment: Learning augmentation strategies from data[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2019: 113-123.

[2] Cubuk E D, Zoph B, Shlens J, et al. Randaugment: Practical automated data augmentation with a reduced search space[J]. arXiv preprint arXiv:1909.13719, 2019.

[3] DeVries T, Taylor G W. Improved regularization of convolutional neural networks with cutout[J]. arXiv preprint arXiv:1708.04552, 2017.

[4] Zhong Z, Zheng L, Kang G, et al. Random erasing data augmentation[J]. arXiv preprint arXiv:1708.04896, 2017.

[5] Singh K K, Lee Y J. Hide-and-seek: Forcing a network to be meticulous for weakly-supervised object and action localization[C]//2017 IEEE international conference on computer vision (ICCV). IEEE, 2017: 3544-3553.

[6] Chen P. GridMask Data Augmentation[J]. arXiv preprint arXiv:2001.04086, 2020.

[7] Zhang H, Cisse M, Dauphin Y N, et al. mixup: Beyond empirical risk minimization[J]. arXiv preprint arXiv:1710.09412, 2017.

[8] Yun S, Han D, Oh S J, et al. Cutmix: Regularization strategy to train strong classifiers with localizable features[C]//Proceedings of the IEEE International Conference on Computer Vision. 2019: 6023-6032.

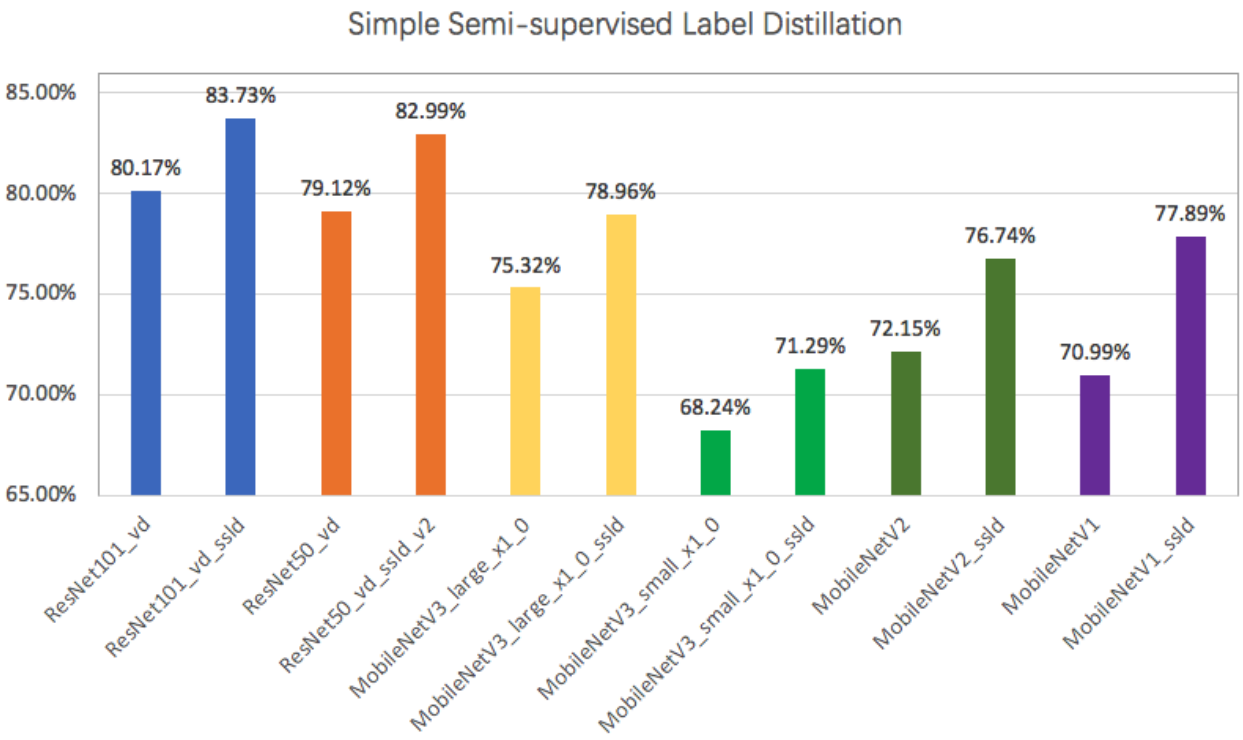
3.2 distillation

3.2.1 Introduction of model compression methods

In recent years, deep neural networks have been proven to be an extremely effective method to solve problems in the fields of computer vision and natural language processing. The deep learning methods performs better than traditional methods with suitable network structure and training process.

With enough training data, increasing parameters of the neural network by building a reasonabe network can significantly the model performance. But this increases the model complexity, which takes too much computation cost in real scenarios.

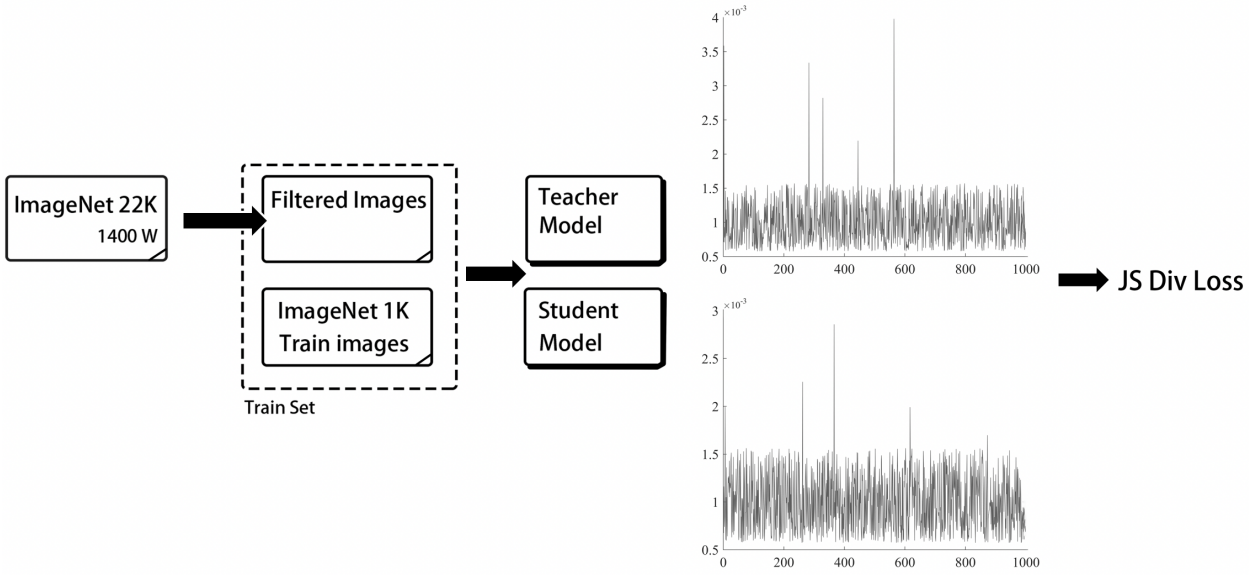
Parameter redundancy exists in deep neural networks. There are several methods to compress the model suck as pruning ,quantization, knowledge distillation, etc. Knowledge distillation refers to using the teacher model to guide the student model to learn specific tasks, ensuring that the small model has a relatively large effect improvement with the computation cost unchanged, and even obtains similar accuracy with the large model [1]. Combining some of the existing distillation methods [2,3], PaddleClas provides a simple semi-supervised label knowledge distillation solution (SSLD). Top-1 Accuarcy on ImageNet1k dataset has an improvement of more than 3% based on ResNet_vd and MobileNet series, which can be shown as below.



3.2.2 SSLD

Introduction

The following figure shows the framework of SSLD.



First, we select nearly 4 million images from ImageNet22k dataset, and integrate it with the ImageNet-1k training set to get a new dataset containing 5 million images. Then, we combine the student model and the teacher model into a new network, which outputs the predictions of the student model and the teacher model, respectively. The gradient of the entire network of the teacher model is fixed. Finally, we use JS divergence loss as the loss function for the training

process. Here we take MobileNetV3 distillation task as an example, and introduce key points of SSLD.

- Choice of the teacher model, During knowledge distillation, it may not be an optimal solution if the structure of the teacher model and the student model are too different. Under the same structure, the teacher model with higher accuracy leads to better performance for the student model during distillation. Compared with the 79.12% ResNet50_vd teacher model, using the 82.4% teacher model can bring a 0.4% accuracy improvement on Top-1 accuracy (`75.6%-> 76.0%`).
- Improvement of loss function. The most commonly used loss function for classification is cross entropy loss. We fint that when using soft label for training, KL divergence loss is almost useless to improve model performance compared to cross entropy loss, but The accuracy has a 0.2% improvement using JS divergence loss (`76.0%-> 76.2%`). Loss function in SSLD is JS divergence loss.
- More iteration number. It is only 120 for the baseline experiment. We can achieve a 0.9% improvement by setting it as 360 (`76.2%-> 77.1%`).
- There is not need for laleled data in SSLD, which leads to convenient training data expansion. label is not utilized when computing the loss function, therefore the unlabeled data can also be used to train the network. The label-free distillation strategy of this distillation solution has also greatly improved the upper performance limit of student models (`77.1%-> 78.5%`).
- ImageNet1k finetune. ImageNet1k training set is used for finetuning, which brings a 0.4% accuracy improvement (`75.8%-> 78.9%`).

Data selection

- An important feature of the SSLD distillation scheme is no need for labeled images, so the dataset size can be arbitrarily expanded. Considering the limitation of computing resources, we here only expand the training set of the distillation task based on the ImageNet22k dataset. For SSLD, we used the `Top-k per class` data sampling scheme [3]. Specific steps are as follows. * Deduplication of training set. We first deduplicate the ImageNet22k dataset and the ImageNet1k validation set based on the SIFT feature similarity matching method to prevent the added ImageNet22k training set from containing the ImageNet1k validation set images. Finally we removed 4511 similar images. Similar pictures with partial filtering are shown below.

– Obtain the soft label of the ImageNet22k dataset. For the ImageNet22k dataset after deduplication, we use the `ResNeXt101_32x16d_wsl` model to make predictions to obtain the soft label of each image. * Top-k data selection. There contains 1000 categories in ImageNet1k dataset. For each category, we find out images in the category with Top-k highest score, and finally generate a dataset whose image number does not exceed `1000 * k` (For some categories, there may contain less than k images). * The selected images are merged with the ImageNet1k training set to form the new dataset used for the final distillation model training, which contains 5 million images in all.

3.2.3 Experiments

The distillation solution that PaddleClas provides is combining common training with finetuning. Given a suitable teacher model, the large dataset(5 million) is used for common training and the ImageNet1k dataset is used for finetuning.

Choice of teacher model

In order to verify the influence of the model size difference between the teacher model and the student model on the distillation results as well as the teacher model accuracy, we conducted several experiments. The training strategy is unified as follows: `cosine_decay_warmup, lr = 1.3, epoch = 120, bs = 2048`, and the student models are all trained from scratch.

It can be shown from the table that:

> When the teacher model structure is the same, the higher the teacher model accuracy, the better the final student model will be.
>
> The size difference between the teacher model and the student model should not be too large, otherwise it will decrease the accuracy of the distillation results.

Therefore, during distillation, for the ResNet series student model, we use `ResNeXt101_32x16d_wsl` as the teacher model; for the MobileNet series student model, we use`ResNet50_vd_SSLD` as the teacher model.

Distillation using large-scale dataset

Training process is carried out on the large-scale dataset with 5 million images. Specifically, the following table shows more details of different models.

finetuning using ImageNet1k

Finetuning is carried out on ImageNet1k dataset to restore distribution between training set and test set. the following table shows more details of finetuning.

Data agmentation and Fix strategy

- Based on experiments mentioned above, we add AutoAugment [4] during training process, and reduced l2_decay from 4e-5 t 2e-5. Finally, the Top-1 accuracy on ImageNet1k dataset can reach 82.99%, with 0.6% improvement compared to the standard SSLD distillation strategy.
- For image classsification tasks, The model accuracy can be further improved when the test scale is 1.15 times that of training[5]. For the 82.99% ResNet50_vd pretrained model, it comes to 83.7% using 320x320 for the evaluation. We use Fix strategy to finetune the model with the training scale set as 320x320. During the process, the pre-preocessing pipeline is same for both training and test. All the weights except the fully connected layer are freezed. Finally the top-1 accuracy comes to **84.0%**.

3.2.4 Application of the distillation model

Instructions

- Adjust the learning rate of the middle layer. The middle layer feature map of the model obtained by distillation is more refined. Therefore, when the distillation model is used as the pretrained model in other tasks, if the same learning rate as before is adopted, it is easy to destroy the features. If the learning rate of the overall model training is reduced, it will bring about the problem of slow convergence. Therefore, we use the strategy of adjusting the learning rate of the middle layer. specifically: * For ResNet50_vd, we set up a learning rate list. The three conv2d convolution parameters before the resiual block have a uniform learning rate multiple, and the four resiual block conv2d have theirs own learning rate parameters, respectively. 5 values need to be set in the list. By the experiment, we find that when used for transfer learning finetune classification model, the learning rate list with `[0.1,0.1,0.2,0.2,0.3]` performs better in most tasks; while in the object detection tasks, `[0.05, 0.05, 0.05, 0.1, 0.15]` can bring greater accuracy gains. * For MoblileNetV3_large_1x0, because it contains 15 blocks, we set each 3 blocks to share a learning rate, so 5 learning rate values are required. We find that in classification and detection tasks, the learning rate list with `[0.25, 0.25, 0.5, 0.5, 0.75]` performs better in most tasks.
- Appropriate l2 decay. Different l2 decay values are set for different models during training. In order to prevent overfitting, l2 decay is ofen set as large for large models. L2 decay is set as `1e-4` for ResNet50, and `1e-5 ~ 4e-5` for MobileNet series models. L2 decay needs also to be adjusted when applied in other tasks. Taking Faster_RCNN_MobiletNetV3_FPN as an example, we found that only modifying l2 decay can bring up to 0.5% accuracy (mAP) improvement on the COCO2017 dataset.

Transfer learning

- To verify the effect of the SSLD pretrained model in transfer learning, we carried out experiments on 10 small datasets. Here, in order to ensure the comparability of the experiment, we use the standard preprocessing process trained by the ImageNet1k dataset. For the distillation model, we also add a simple search method for the learning rate of the middle layers of the distillation pretrained model.
- For ResNet50_vd, the baseline pretrained model Top-1 Acc is 79.12%, the other parameters are got by grid search. For distillation pretrained model, we add learning rate of the middle layers into the search space. The following table shows the results.
- It can be seen that on the above 10 datasets, combined with the appropriate middle layer learning rate, the distillation pretrained model can bring an average accuracy improvement of more than 1%.

Object detection

Based on the two-stage Faster/Cascade RCNN model, we verify the effect of the pretrained model obtained by distillation.

- ResNet50_vd

Training scale and test scale are set as 640x640, and some of the ablationstudies are as follows.

It can be seen here that for the baseline pretrained model, excessive adjustment of the middle-layer learning rate actually reduces the performance of the detection model. Based on this distillation model, we also provide a practical server-side detection solution. The detailed configuration and training code are open source, more details can be refer to [PaddleDetection] (https://github.com/PaddlePaddle/PaddleDetection/tree/master/configs/rcnn_enhance).

3.2.5 Practice

This section will introduce the SSLD distillation experiments in detail based on the ImageNet-1K dataset. If you want to experience this method quickly, you can refer to [** Quick start PaddleClas in 30 minutes**] (../../tutorials/quick_start.md), whose dataset is set as Flowers102.

Configuration

Distill ResNet50_vd using ResNeXt101_32x16d_wsl

Configuration of distilling `ResNet50_vd` using `ResNeXt101_32x16d_wsl` is as follows.

```
ARCHITECTURE:
    name: 'ResNeXt101_32x16d_wsl_distill_ResNet50_vd'
pretrained_model: "./pretrained/ResNeXt101_32x16d_wsl_pretrained/"
# pretrained_model:
#     - "./pretrained/ResNeXt101_32x16d_wsl_pretrained/"
#     - "./pretrained/ResNet50_vd_pretrained/"
use_distillation: True
```

Distill MobileNetV3_large_x1_0 using ResNet50_vd_ssld

The detailed configuration is as follows.

```
ARCHITECTURE:
    name: 'ResNet50_vd_distill_MobileNetV3_large_x1_0'
pretrained_model: "./pretrained/ResNet50_vd_ssld_pretrained/"
# pretrained_model:
#     - "./pretrained/ResNet50_vd_ssld_pretrained/"
#     - "./pretrained/ResNet50_vd_pretrained/"
use_distillation: True
```

Begin to train the network

If everything is ready, users can begin to train the network using the following command.

```
export PYTHONPATH=path_to_PaddleClas:$PYTHONPATH

python -m paddle.distributed.launch \
    --selected_gpus="0,1,2,3" \
    --log_dir=R50_vd_distill_MV3_large_x1_0 \
    tools/train.py \
        -c ./configs/Distillation/R50_vd_distill_MV3_large_x1_0.yaml
```

Note

- Before using SSLD, users need to train a teacher model on the target dataset firstly. The teacher model is used to guide the training of the student model.
- When using SSLD, users need to set `use_distillation` in the configuration file to`True`. In addition, because the student model learns soft-label with knowledge information, you need to turn off the `label_smoothing` option.

- If the student model is not loaded with a pretrained model, the other hyperparameters of the training can refer to the hyperparameters trained by the student model on ImageNet-1k. If the student model is loaded with the pre-trained model, the learning rate can be adjusted to `1/100~1/10` of the standard learning rate.
- In the process of SSLD distillation, the student model only learns the soft label, which makes the training process more difficult. It is recommended that the value of `l2_decay` can be decreased appropriately to obtain higher accuracy of the validation set.
- If users are going to add unlabeled training data, just the training list textfile needs to be adjusted for more data.

  If this document is helpful to you, welcome to star our project: https://github.com/PaddlePaddle/PaddleClas

3.2.6 Reference

[1] Hinton G, Vinyals O, Dean J. Distilling the knowledge in a neural network[J]. arXiv preprint arXiv:1503.02531, 2015.

[2] Bagherinezhad H, Horton M, Rastegari M, et al. Label refinery: Improving imagenet classification through label progression[J]. arXiv preprint arXiv:1805.02641, 2018.

[3] Yalniz I Z, Jégou H, Chen K, et al. Billion-scale semi-supervised learning for image classification[J]. arXiv preprint arXiv:1905.00546, 2019.

[4] Cubuk E D, Zoph B, Mane D, et al. Autoaugment: Learning augmentation strategies from data[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2019: 113-123.

[5] Touvron H, Vedaldi A, Douze M, et al. Fixing the train-test resolution discrepancy[C]//Advances in Neural Information Processing Systems. 2019: 8250-8260.

CHAPTER 4

application

4.1 Transfer learning in image classification

Transfer learning is an important part of machine learning, which is widely used in various fields such as text and images. Here we mainly introduce transfer learning in the field of image classification, which is often called domain transfer, such as migration of the ImageNet classification model to the specified image classification task, such as flower classification.

4.1.1 Hyperparameter search

ImageNet is the widely used dataset for image classification. A series of empirical hyperparameters have been summarized. High accuracy can be got using the hyperparameters. However, when applied in the specified dataset, the hyperparameters may not be optimal. There are two commonly used hyperparameter search methods that can be used to help us obtain better model hyperparameters.

Grid search

For grid search, which is also called exhaustive search, the optimal value is determined by finding the best solution from all solutions in the search space. The method is simple and effective, but when the search space is large, it takes huge computing resource.

Bayesian search

Bayesian search, which is also called Bayesian optimization, is realized by randomly selecting a group of hyperparameters in the search space. Gaussian process is used to update the hyperparameters, compute their expected mean and variance according to the performance of the previous hyperparameters. The larger the expected mean, the greater the probability of being close to the optimal solution. The larger the expected variance, the greater the uncertainty. Usually, the hyperparameter point with large expected mean is called `exporitation`, and the hyperparameter point with large variance is called `exploration`. Acquisition function is defined to balance the expected mean and variance. The currently selected hyperparameter point is viewed as the optimal position with maximum probability.

According to the above two search schemes, we carry out some experiments based on fixed scheme and two search schemes on 8 open source datasets. As the experimental scheme in [1], we search for 4 hyperparameters, the search space and The experimental results are as follows:

a fixed set of parameter experiments and two search schemes on 8 open source data sets. With reference to the experimental scheme of [1], we search for 4 hyperparameters, the search space and the experimental results are as follows:

- Fixed scheme.

```
lr=0.003l2 decay=1e-4label smoothing=Falsemixup=False
```

- Search space of the hyperparameters.

```
lr: [0.1, 0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001]

l2 decay: [1e-3, 3e-4, 1e-4, 3e-5, 1e-5, 3e-6, 1e-6]

label smoothing: [False, True]

mixup: [False, True]
```

It takes 196 times for grid search, and takes 10 times less for Bayesian search. The baseline is trained by using ImageNet1k pretrained model based on ResNet50_vd and fixed scheme. The follow shows the experiments.

- The above experiments verify that Bayesian search only reduces the accuracy by 0% to 0.4% under the condition of reducing the number of searches by about 10 times compared to grid search.
- The search space can be expaned easily using Bayesian search.

4.1.2 Large-scale image classification

In practical applications, due to the lack of training data, the classification model trained on the ImageNet1k data set is often used as the pretrained model for other image classification tasks. In order to further help solve practical problems, based on ResNet50_vd, Baidu open sourced a self-developed large-scale classification pretrained model, in which the training data contains 100,000 categories and 43 million pictures.

We conducted transfer learning experiments on 6 self-collected datasets,

using a set of fixed parameters and a grid search method, in which the number of training rounds was set to 20epochs, the ResNet50_vd model was selected, and the ImageNet pre-training accuracy was 79.12%. The comparison results of the experimental data set parameters and model accuracy are as follows:

Fixed scheme

```
lr=0.001l2 decay=1e-4label smoothing=Falsemixup=False
```

- The above experiments verified that for fixed parameters, compared with the pretrained model on ImageNet, using the large-scale classification model as a pretrained model can help us improve the model performance on a new dataset in most cases. Parameter search can be further helpful to the model performance.

4.1.3 Reference

[1] Kornblith, Simon, Jonathon Shlens, and Quoc V. Le. "Do better imagenet models transfer better?." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2019.

[2] Kolesnikov, Alexander, et al. "Large Scale Learning of General Visual Representations for Transfer." *arXiv preprint arXiv:1912.11370* (2019).

4.2 General object detection

4.2.1 Practical Server-side detection method base on RCNN

Introduction

- In recent years, object detection tasks have attracted widespread attention. PaddleClas open-sourced the ResNet50_vd_SSLD pretrained model based on ImageNet(Top1 Acc 82.4%). And based on the pretrained model, PaddleDetection provided the PSS-DET (Practical Server-side detection) with the help of the rich operators in PaddleDetection. The inference speed can reach 61FPS on single V100 GPU when COCO mAP is 41.6%, and 20FPS when COCO mAP is 47.8%.
- We take the standard `Faster RCNN ResNet50_vd FPN` as an example. The following table shows ablation study of PSS-DET.

Based on the ablation experiments, Cascade RCNN and larger inference scale(1000x1500) are used for better performance. The final COCO mAP is 47.8% and the following figure shows `mAP-Speed` curves for some common detectors.

pssdet

Note

For fair comparison, inference time for PSS-DET models on V100 GPU is transformed to Titan V GPU by multiplying by 1.2 times.

For more detailed information, you can refer to PaddleDetection.

4.2.2 Practical Mobile-side detection method base on RCNN

- This part is comming soon!

CHAPTER 5

extension

5.1 Prediction Framework

5.1.1 Introduction

Models for Paddle are stored in many different forms, which can be roughly divided into two categories

1. persistable modelthe models saved by fluid.save_persistables The weights are saved in checkpoint, which can be loaded to retrain, one scattered weight file saved by persistable stands for one persistable variable in the model, there is no structure information in these variable, so the weights should be used with the model structure.

```
resnet50-vd-persistable/
├── bn2a_branch1_mean
├── bn2a_branch1_offset
├── bn2a_branch1_scale
├── bn2a_branch1_variance
├── bn2a_branch2a_mean
├── bn2a_branch2a_offset
├── bn2a_branch2a_scale
├── ...
└── res5c_branch2c_weights
```

2. inference modelthe models saved by fluid.io.save_inference_model The model saved by this function cam be used for inference directly, compared with the ones saved by persistable, the model structure will be additionally saved in the model, with the weights, the model with trained weights can be reconstruction. as shown in the following figure, the structure information is saved in `model`

```
resnet50-vd-persistable/
├── bn2a_branch1_mean
├── bn2a_branch1_offset
├── bn2a_branch1_scale
├── bn2a_branch1_variance
├── bn2a_branch2a_mean
```

(continues on next page)

(continued from previous page)

```
├── bn2a_branch2a_offset
├── bn2a_branch2a_scale
├── ...
├── res5c_branch2c_weights
└── model
```

For convenience, all weight files will be saved into a `params` file when saving the inference model on Paddle, as shown below

```
resnet50-vd
├── model
└── params
```

Both the training engine and the prediction engine in Paddle support the model's e inference, but the back propagation is not performed during the inference, so it can be customized optimization (such as layer fusion, kernel selection, etc.) to achieve low latency and high throughput during inference. The training engine can support either the persistable model or the inference model, and the prediction engine only supports the inference model, so three different inferences are derived

1. prediction engine + inference model
2. training engine + inference model
3. training engine + inference model

Regardless of the inference method, it basically includes the following main steps

- Engine Build
- Make Data to Be Predicted
- Perform Predictions
- Result Analysis

There are two main differences in different inference methods: building the engine and executing the forecast. The following sections will be introduced in detail

5.1.2 Model Transformation

During training, we usually save some checkpoints (persistable models). These are just model weight files and cannot be directly loaded by the prediction engine to predict, so we usually find suitable checkpoints after the training and convert them to inference model. There are two main steps: 1. Build a training engine, 2. Save the inference model, as shown below.

```
import fluid

from ppcls.modeling.architectures.resnet_vd import ResNet50_vd

place = fluid.CPUPlace()
exe = fluid.Executor(place)
startup_prog = fluid.Program()
infer_prog = fluid.Program()
with fluid.program_guard(infer_prog, startup_prog):
    with fluid.unique_name.guard():
        image = create_input()
        image = fluid.data(name='image', shape=[None, 3, 224, 224], dtype='float32')
        out = ResNet50_vd.net(input=input, class_dim=1000)
```

(continues on next page)

(continued from previous page)

```
infer_prog = infer_prog.clone(for_test=True)
fluid.load(program=infer_prog, model_path=the path of persistable model, executor=exe)

fluid.io.save_inference_model(
        dirname='./output/',
        feeded_var_names=[image.name],
        main_program=infer_prog,
        target_vars=out,
        executor=exe,
        model_filename='model',
        params_filename='params')
```

A complete example is provided in the `tools/export_model.py`, just execute the following command to complete the conversion

```
python tools/export_model.py \
    --m=the name of model \
    --p=the path of persistable model\
    --o=the saved path of model and params
```

5.1.3 Prediction engine + inference model

The complete example is provided in the `tools/infer/predict.py`just execute the following command to complete the prediction:

```
python ./predict.py \
    -i=./test.jpeg \
    -m=./resnet50-vd/model \
    -p=./resnet50-vd/params \
    --use_gpu=1 \
    --use_tensorrt=True
```

Parameter Description

- `image_file`(shortening i)the path of images which are needed to predictsuch as `./test.jpeg`.
- `model_file`(shortening m)the path of weights foldersuch as `./resnet50-vd/model`.
- `params_file`(shortening p)the path of weights filesuch as `./resnet50-vd/params`.
- `batch_size`(shortening b)batch sizesuch as `1`.
- `ir_optim` whether to use `IR` optimization, default: True.
- `use_tensorrt`: whether to use TensorRT prediction engine, default:True.
- `gpu_mem` Initial allocation of GPU memory, the unit is M.
- `use_gpu`: whether to use GPU, default: True.
- `enable_benchmark`whether to use benchmark, default: False.
- `model_name`the name of model.

NOTE when using benchmark, we use tersorrt by default to make predictions on Paddle.

Building prediction engine

```
from paddle.fluid.core import AnalysisConfig
from paddle.fluid.core import create_paddle_predictor
config = AnalysisConfig(the path of model file, the path of params file)
config.enable_use_gpu(8000, 0)
config.disable_glog_info()
config.switch_ir_optim(True)
config.enable_tensorrt_engine(
        precision_mode=AnalysisConfig.Precision.Float32,
        max_batch_size=1)

# no zero copyfetch feed op
config.switch_use_feed_fetch_ops(False)

predictor = create_paddle_predictor(config)
```

Prediction Execution

```
import numpy as np

input_names = predictor.get_input_names()
input_tensor = predictor.get_input_tensor(input_names[0])
input = np.random.randn(1, 3, 224, 224).astype("float32")
input_tensor.reshape([1, 3, 224, 224])
input_tensor.copy_from_cpu(input)
predictor.zero_copy_run()
```

More parameters information can be refered in Paddle Python prediction API. If you need to predict in the environment of business, we recommand you to use Paddel C++ prediction APIa rich pre-compiled prediction library is provided in the offical websitePaddle C++ prediction library

By default, Paddle's wheel package does not include the TensorRT prediction engine. If you need to use TensorRT for prediction optimization, you need to compile the corresponding wheel package yourself. For the compilation method, please refer to Paddle's compilation guide. Paddle compilation

5.1.4 Training engine + persistable model prediction

A complete example is provided in the `tools/infer/infer.py`, just execute the following command to complete the prediction

```
python tools/infer/infer.py \
    --i=the path of images which are needed to predict \
    --m=the name of model \
    --p=the path of persistable model \
    --use_gpu=True
```

Parameter Description

- `image_file`(shortening i)the path of images which are needed to predictsuch as `./test.jpeg`
- `model_file`(shortening m)the path of weights foldersuch as `./resnet50-vd/model`
- `params_file`(shortening p)the path of weights filesuch as `./resnet50-vd/params`
- `use_gpu` : whether to use GPU, default: True.

Training Engine Construction

Since the persistable model does not contain the structural information of the model, it is necessary to construct the network structure first, and then load the weights to build the training engine

```
import fluid
from ppcls.modeling.architectures.resnet_vd import ResNet50_vd

place = fluid.CPUPlace()
exe = fluid.Executor(place)
startup_prog = fluid.Program()
infer_prog = fluid.Program()
with fluid.program_guard(infer_prog, startup_prog):
    with fluid.unique_name.guard():
        image = create_input()
        image = fluid.data(name='image', shape=[None, 3, 224, 224], dtype='float32')
        out = ResNet50_vd.net(input=input, class_dim=1000)
infer_prog = infer_prog.clone(for_test=True)
fluid.load(program=infer_prog, model_path=the path of persistable model, executor=exe)
```

Perform inference

```
outputs = exe.run(infer_prog,
        feed={image.name: data},
        fetch_list=[out.name],
        return_numpy=False)
```

For the above parameter descriptions, please refer to the official website fluid.Executor

5.1.5 Training engine + inference model prediction

A complete example is provided in `tools/infer/py_infer.py`, just execute the following command to complete the prediction

```
python tools/infer/py_infer.py \
    --i=the path of images \
    --d=the path of saved model \
    --m=the path of saved model file \
    --p=the path of saved weight file \
    --use_gpu=True
```

- `image_file`(shortening i)the path of images which are needed to predict `./test.jpeg`
- `model_file`(shortening m)the path of model file `./resnet50_vd/model`
- `params_file`(shortening p)the path of weights file `./resnet50_vd/params`
- `model_dir`(shortening d)the folder of model `./resent50_vd`
- `use_gpu`whether to use GPU, default: True

Training engine build

Since inference model contains the structure of model, we do not need to construct the model before, load the model file and weights file directly to bulid training engine.

```
import fluid

place = fluid.CPUPlace()
exe = fluid.Executor(place)
[program, feed_names, fetch_lists] = fluid.io.load_inference_model(
        the path of saved model,
        exe,
```

(continues on next page)

(continued from previous page)

```
        model_filename=the path of model file,
        params_filename=the path of weights file)
compiled_program = fluid.compiler.CompiledProgram(program)
```

`load_inference_model` Not only supports scattered weight file collection, but also supports a single weight file

Perform inference

```
outputs = exe.run(compiled_program,
        feed={feed_names[0]: data},
        fetch_list=fetch_lists,
        return_numpy=False)
```

For the above parameter descriptions, please refer to the official website fluid.Executor

5.2 Paddle-Lite

5.2.1 Introduction

Paddle-Lite is a set of lightweight inference engine which is fully functional, easy to use and then performs well. Lightweighting is reflected in the use of fewer bits to represent the weight and activation of the neural network, which can greatly reduce the size of the model, solve the problem of limited storage space of the mobile device, and the inference speed is better than other frameworks on the whole.

In PaddleClas, we uses Paddle-Lite to evaluate the performance on the mobile device, in this section we uses the `MobileNetV1` model trained on the `ImageNet1k` dataset as an example to introduce how to use `Paddle-Lite` to evaluate the model speed on the mobile terminal (evaluated on SD855)

5.2.2 Evaluation Steps

Export the Inference Model

- First you should transform the saved model during training to the special model which can be used to inference, the special model can be exported by `tools/export_model.py`, the specific way of transform is as follows.

```
python tools/export_model.py -m MobileNetV1 -p pretrained/MobileNetV1_pretrained/ -o␣
↪inference/MobileNetV1
```

Finally the `model` and `parmas` can be saved in `inference/MobileNetV1`.

Download Benchmark Binary File

- Use the adb (Android Debug Bridge) tool to connect the Android phone and the PC, then develop and debug. After installing adb and ensuring that the PC and the phone are successfully connected, use the following command to view the ARM version of the phone and select the pre-compiled library based on ARM version.

```
adb shell getprop ro.product.cpu.abi
```

- Download Benchmark_bin File

```
wget -c https://paddle-inference-dist.bj.bcebos.com/PaddleLite/benchmark_0/benchmark_
↪bin_v8
```

If the ARM version is v7, the v7 benchmark_bin file should be downloaded, the command is as follow.

```
wget -c https://paddle-inference-dist.bj.bcebos.com/PaddleLite/benchmark_0/benchmark_
↪bin_v7
```

Inference benchmark

After the PC and mobile phone are successfully connected, use the following command to start the model evaluation.

```
sh tools/lite/benchmark.sh ./benchmark_bin_v8 ./inference result_armv8.txt true
```

Where `./benchmark_bin_v8` is the path of the benchmark binary file, `./inference` is the path of all the models that need to be evaluated, `result_armv8.txt` is the result file, and the final parameter `true` means that the model will be optimized before evaluation. Eventually, the evaluation result file of `result_armv8.txt` will be saved in the current folder. The specific performances are as follows.

```
PaddleLite Benchmark
Threads=1 Warmup=10 Repeats=30
MobileNetV1                              min = 30.89100    max = 30.73600    average =␣
↪30.79750

Threads=2 Warmup=10 Repeats=30
MobileNetV1                              min = 18.26600    max = 18.14000    average =␣
↪18.21637

Threads=4 Warmup=10 Repeats=30
MobileNetV1                              min = 10.03200    max = 9.94300     average = 9.
↪97627
```

Here is the model inference speed under different number of threads, the unit is FPS, taking model on one threads as an example, the average speed of MobileNetV1 on SD855 is `30.79750FPS`.

Model Optimization and Speed Evaluation

- In II.III section, we mention that the model will be optimized before evaluation, here you can first optimize the model, and then directly load the optimized model for speed evaluation
- Paddle-Lite In Paddle-Lite, we provides multiple strategies to automatically optimize the original training model, which contain Quantify, Subgraph fusion, Hybrid scheduling, Kernel optimization and so on. In order to make the optimization more convenient and easy to use, we provide opt tools to automatically complete the optimization steps and output a lightweight, optimal and executable model in Paddle-Lite, which can be downloaded on Paddle-Lite Model Optimization Page. Here we take `MacOS` as our development environment, downloadopt_mac model optimization tools and use the following commands to optimize the model.

```
model_file="../MobileNetV1/model"
param_file="../MobileNetV1/params"
opt_models_dir="./opt_models"
mkdir ${opt_models_dir}
./opt_mac --model_file=${model_file} \
    --param_file=${param_file} \
    --valid_targets=arm \
```

(continues on next page)

(continued from previous page)

```
    --optimize_out_type=naive_buffer \
    --prefer_int8_kernel=false \
    --optimize_out=${opt_models_dir}/MobileNetV1
```

Where the `model_file` and `param_file` are exported model file and the file address respectively, after transforming successfully, the `MobileNetV1.nb` will be saved in `opt_models`

Use the benchmark_bin file to load the optimized model for evaluation. The commands are as follows.

```
bash benchmark.sh ./benchmark_bin_v8 ./opt_models result_armv8.txt
```

Finally the result is saved in `result_armv8.txt` and shown as follow.

```
PaddleLite Benchmark
Threads=1 Warmup=10 Repeats=30
MobileNetV1_lite              min = 30.89500    max = 30.78500    average = 30.84173

Threads=2 Warmup=10 Repeats=30
MobileNetV1_lite              min = 18.25300    max = 18.11000    average = 18.18017

Threads=4 Warmup=10 Repeats=30
MobileNetV1_lite              min = 10.00600    max = 9.90000     average = 9.96177
```

Taking the model on one threads as an example, the average speed of MobileNetV1 on SD855 is `30.84173FPS`.

More specific parameter explanation and Paddle-Lite usage can refer to Paddle-Lite docs

5.3 Model Quantification

Int8 quantization is one of the key features in PaddleSlim. It supports two kinds of training aware, **Dynamic strategy** and **Static strategy**, layer-wise and channel-wise quantization, and using PaddleLite to deploy models generated by PaddleSlim.

By using this toolkit, PaddleClas quantized the mobilenet_v3_large_x1_0 model whose accuracy is 78.9% after distilled. After quantized, the prediction speed is accelerated from 19.308ms to 14.395ms on SD855. The storage size is reduced from 21M to 10M. The top1 recognition accuracy rate is 75.9%. For specific training methods, please refer to PaddleSlim quant aware

5.4 Distributed Training

Distributed deep neural networks training is highly efficient in PaddlePaddle. And it is one of the PaddlePaddle's core advantage technologies. On image classification tasks, distributed training can achieve almost linear acceleration ratio. Fleet is High-Level API for distributed training in PaddlePaddle. By using Fleet, a user can shift from local machine paddlepaddle code to distributed code easily. In order to support both single-machine training and multi-machine training, PaddleClas uses the Fleet API interface. For more information about distributed training, please refer to Fleet API documentation.

5.5 Paddle Hub

PaddleHub is a pre-trained model application tool for PaddlePaddle. Developers can conveniently use the high-quality pre-trained model combined with Fine-tune API to quickly complete the whole process from model migration to

deployment. All the pre-trained models of PaddleClas have been collected by PaddleHub. For further details, please refer to PaddleHub website.

5.6 Model Service Deployment

5.6.1 Overview

Paddle Serving aims to help deep-learning researchers to easily deploy online inference services, supporting one-click deployment of industry, high concurrency and efficient communication between client and server and supporting multiple programming languages to develop clients.

Taking HTTP inference service deployment as an example to introduce how to use PaddleServing to deploy model services in PaddleClas.

5.6.2 Serving Install

It is recommends to use docker to install and deploy the Serving environment in the Serving official website, first, you need to pull the docker environment and create Serving-based docker.

```
nvidia-docker pull hub.baidubce.com/paddlepaddle/serving:0.2.0-gpu
nvidia-docker run -p 9292:9292 --name test -dit hub.baidubce.com/paddlepaddle/
↪serving:0.2.0-gpu
nvidia-docker exec -it test bash
```

In docker, you need to install some packages about Serving

```
pip install paddlepaddle-gpu
pip install paddle-serving-client
pip install paddle-serving-server-gpu
```

- If the installation speed is too slow, you can add `-i https://pypi.tuna.tsinghua.edu.cn/simple` following pip to speed up the process.
- If you want to deploy CPU service, you can install the cpu version of Serving, the command is as follow.

```
pip install paddle-serving-server
```

Export Model

Exporting the Serving model using `tools/export_serving_model.py`, taking ResNet50_vd as an example, the command is as follow.

```
python tools/export_serving_model.py -m ResNet50_vd -p ./pretrained/ResNet50_vd_
↪pretrained/ -o serving
```

finally, the client configures, model parameters and structure file will be saved in `ppcls_client_conf` and `ppcls_model`.

Service Deployment and Request

- Using the following commands to start the Serving.

```
python tools/serving/image_service_gpu.py serving/ppcls_model workdir 9292
```

`serving/ppcls_model` is the address of the Serving model just saved, `workdir` is the work directory, and `9292` is the port of the service.

- Using the following script to send an identification request to the Serving and return the result.

```
python tools/serving/image_http_client.py  9292 ./docs/images/logo.png
```

`9292` is the port for sending the request, which is consistent with the Serving starting port, and `./docs/images/logo.png` is the test image, the final top1 label and probability are returned.

- For more Serving deployment, such RPC inference service, you can refer to the Serving official website: https://github.com/PaddlePaddle/Serving/tree/develop/python/examples/imagenet

CHAPTER 6

Competition Support

PaddleClas stems from the Baidu's visual business applications and the exploration of frontier visual capabilities. It has helped us achieve leading results in many key events, and continues to promote more frontier visual solutions and landing applications.

- 1st place in 2018 Kaggle Open Images V4 object detection challenge
- 2nd place in 2019 Kaggle Open Images V5 object detection challenge
  - The report is avaiable here: https://arxiv.org/pdf/1911.07171.pdf
  - The pretrained model and code is avaiable here: source code
- 2nd place in Kacggle Landmark Retrieval Challenge 2019
  - The report is avaiable here: https://arxiv.org/abs/1906.03990
  - The pretrained model and code is avaiable here: source code
- 2nd place in Kaggle Landmark Recognition Challenge 2019
  - The report is avaiable here: https://arxiv.org/abs/1906.03990
  - The pretrained model and code is avaiable here: source code
- A-level certificate of three tasks: printed text OCR, face recognition and landmark recognition in the first multimedia information recognition technology competition

CHAPTER 7

Release Notes

- 2020.06.17
  - Add English documents
- 2020.06.12
  - Add support for training and evaluation on Windows or CPU.
- 2020.05.17
  - Add support for mixed precision training.
- 2020.05.09
  - Add user guide about Paddle Serving and Paddle-Lite.
  - Add benchmark about FP16/FP32 on T4 GPU.
- 2020.04.14
  - First commit.

CHAPTER 8

FAQ

- Why are the metrics different for different cards?
- A: Fleet is the default option for the use of PaddleClas. Each GPU card is taken as a single trainer and deals with different images, which cause the final small difference. Single card evalution is suggested to get the accurate results if you use `tools/eval.py`. You can also use `tools/eval_multi_platform.py` to evalute the models on multiple GPU cards, which is also supported on Windows and CPU.

- Q: Why `Mixup` or `Cutmix` is not used even if I have already add the data operation in the configuration file?
- A: When using `Mixup` or `Cutmix`, you also need to add `use_mix: True` in the configuration file to make it work properly.

- Q: During evaluation and inference, pretrained model address is assgined, but the weights can not be imported. Why?
- A: Prefix of the pretrained model is needed. For example, if the pretained weights are located in `output/ResNet50_vd/19`, with the filename `output/ResNet50_vd/19/ppcls.pdparams`, then `pretrained_model` in the configuration file needs to be `output/ResNet50_vd/19/ppcls`.

- Q: Why are the metrics 0.3% lower than that shown in the model zoo for `EfficientNet` series of models?
- A: Resize method is set as `Cubic` for `EfficientNet`(interpolation is set as 2 in OpenCV), while other models are set as `Bilinear`(interpolation is set as None in OpenCV). Therefore, you need to modify the interpolation explicitly in `ResizeImage`. Specifically, the following configuration is a demo for EfficientNet.

```
VALID:
    batch_size: 16
    num_workers: 4
    file_list: "./dataset/ILSVRC2012/val_list.txt"
```

(continues on next page)

(continued from previous page)

```
    data_dir: "./dataset/ILSVRC2012/"
    shuffle_seed: 0
    transforms:
        - DecodeImage:
            to_rgb: True
            to_np: False
            channel_first: False
        - ResizeImage:
            resize_short: 256
            interpolation: 2
        - CropImage:
            size: 224
        - NormalizeImage:
            scale: 1.0/255.0
            mean: [0.485, 0.456, 0.406]
            std: [0.229, 0.224, 0.225]
            order: ''
        - ToCHWImage:
```

- Q: What should I do if I want to transform the weights' format from `pdparams` to an earlier version(before Paddle1.7.0), which consists of the scattered files?
- A: You can use `fluid.load` to load the `pdparams` weights and use `fluid.io.save_vars` to save the weights as scattered files.