PaddleClas

2020 年 07 月 15 日

Contents

CHAPTER 1

初级使用

1.1 安装说明

1.1.1 一、简介

本章将介绍如何安装 PaddleClas 及其依赖项。

1.1.2 二、安装 PaddlePaddle

运行 PaddleClas 需要 PaddlePaddle Fluid v1.7 或更高版本。请参照安装文档中的说明进行操作。

如果已经安装好了 cuda、cudnn、nccl 或者安装好了 docker、nvidia-docker 运行环境，可以 pip 安装最新 GPU 版本 PaddlePaddle

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

也可以从源码编译安装 PaddlePaddle，请参照安装文档中的说明进行操作。

使用以下命令可以验证 PaddlePaddle 是否安装成功。

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

查看 PaddlePaddle 版本的命令如下：

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

注意：

- 从源码编译的 PaddlePaddle 版本号为 0.0.0，请确保使用了 Fluid v1.7 之后的源码编译。
- PaddleClas 基于 PaddlePaddle 高性能的分布式训练能力，若您从源码编译，请确保打开编译选项，**WITH_DISTRIBUTE=ON**。具体编译选项参考编译选项表

运行环境需求：

- Python3 (当前只支持 Linux 系统)
- CUDA >= 9.0
- cuDNN >= 5.0
- nccl >= 2.1.2

1.1.3 三、安装 PaddleClas

克隆 PaddleClas 模型库：

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

安装 Python 依赖库：

Python 依赖库在requirements.txt中给出，可通过如下命令安装：

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

visualdl 可能出现安装失败，请尝试

```
pip3 install --upgrade visualdl==2.0.0b3 -i https://mirror.baidu.com/pypi/simple
```

1.2 30 分钟玩转 PaddleClas

基于 flowers102 数据集，30 分钟体验 PaddleClas 不同骨干网络的模型训练、不同预训练模型、SSLD 知识蒸馏方案和数据增广的效果。请事先参考安装指南配置运行环境和克隆 PaddleClas 代码。

1.2.1 一、数据和模型准备

- 进入 PaddleClas 目录。

```
cd path_to_PaddleClas
```

- 进入 `dataset/flowers102` 目录，下载并解压 flowers102 数据集.

```
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
```

- 制作 train/val/test 标签文件

```
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
```

注意：这里将 train_list.txt 和 extra_list.txt 合并成 train_extra_list.txt，是为了之后在进行知识蒸馏时，使用更多的数据提升无标签知识蒸馏任务的效果。

- 返回 `PaddleClas` 根目录

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

1.2.2 二、环境准备

2.1 设置 PYTHONPATH 环境变量

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

下载预训练模型

通过 tools/download.py 下载所需要的预训练模型。

```
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
```

参数说明：

- `architecture`（简写 a）：模型结构
- `path`（简写 p）：下载路径

- decompress （简写 d）：是否解压

2.2 环境说明

- 下面所有的训练过程均在单卡 V100 机器上运行。

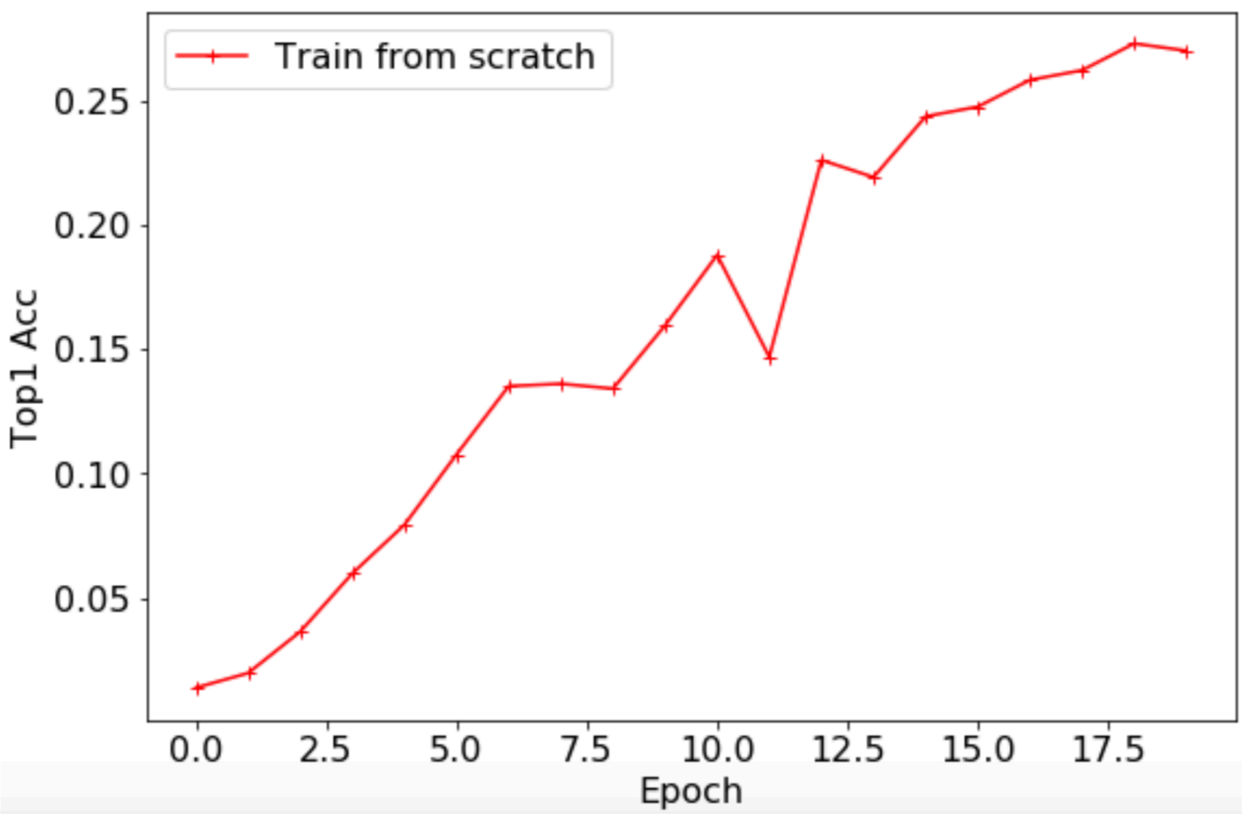
1.2.3 三、模型训练

3.1 零基础训练：不加载预训练模型的训练

- 基于 ResNet50_vd 模型，训练脚本如下所示。

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

验证集的 Top1 Acc 曲线如下所示，最高准确率为 0.2735。

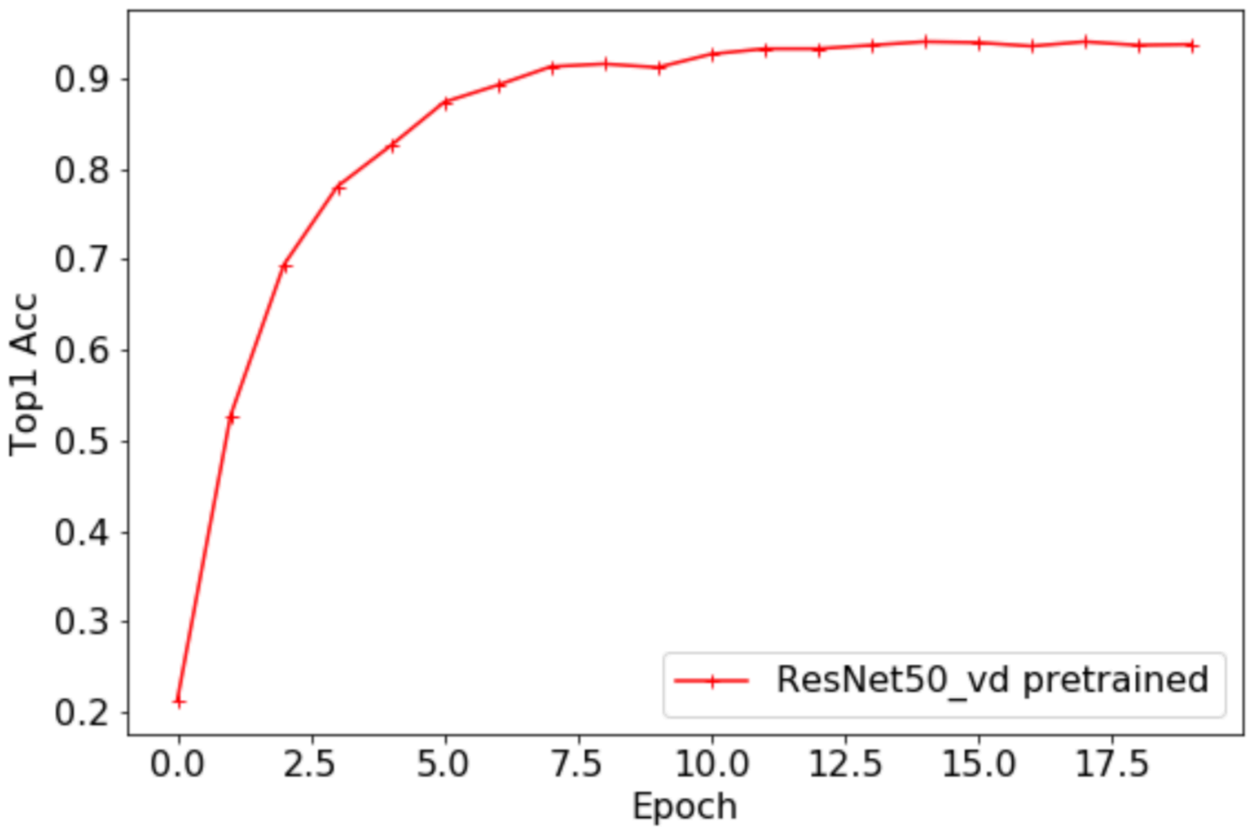


3.2 模型微调-基于 ResNet50_vd 预训练模型 (准确率 79.12%)

- 基于 ImageNet1k 分类预训练模型进行微调，训练脚本如下所示。

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

验证集的 Top1 Acc 曲线如下所示，最高准确率为 0.9402，加载预训练模型之后，flowers102 数据集精度大幅提升，绝对精度涨幅超过 65%。



3.3 SSLD 模型微调-基于 ResNet50_vd_ssld 预训练模型 (准确率 82.39%)

需要注意的是，在使用通过知识蒸馏得到的预训练模型进行微调时，我们推荐使用相对较小的网络中间层学习率。

```
ARCHITECTURE:
    name: 'ResNet50_vd'
    params:
```

(下页继续)

(续上页)

```
        lr_mult_list: [0.1, 0.1, 0.2, 0.2, 0.3]
pretrained_model: "./pretrained/ResNet50_vd_ssld_pretrained"
```

训练脚本如下。

```
export CUDA_VISIBLE_DEVICES=0
python -m paddle.distributed.launch \
    --selected_gpus="0" \
    tools/train.py \
        -c ./configs/quick_start/ResNet50_vd_ssld_finetune.yaml
```

最终 flowers102 验证集上精度指标为 0.95，相对于 79.12% 预训练模型的微调结构，新数据集指标可以再次提升 0.9%。

3.4 尝试更多的模型结构-MobileNetV3

训练脚本如下所示。

```
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
```

最终 flowers102 验证集上的精度为 0.90，比加载了预训练模型的 ResNet50_vd 的精度差了 5%。不同模型结构的网络在相同数据集上的性能表现不同，需要根据预测耗时以及存储的需求选择合适的模型。

3.5 数据增广的尝试-RandomErasing

训练数据量较小时，使用数据增广可以进一步提升模型精度，基于 `3.3` 节中的训练方法，结合 RandomErasing 的数据增广方式进行训练，具体的训练脚本如下所示。

```
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
```

最终 flowers102 验证集上的精度为 0.9627，使用数据增广可以使得模型精度再次提升 1.27%。

- 如果希望体验 `3.6` 节的知识蒸馏部分，可以首先保存训练得到的 ResNet50_vd 预训练模型到合适的位置，作为蒸馏时教师模型的预训练模型。脚本如下所示。

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

3.6 知识蒸馏小试牛刀

- 使用 flowers102 数据集进行模型蒸馏，为了进一步提提升模型的精度，使用 extra_list.txt 充当无标签数据，在这里有几点需要注意：
  - extra_list.txt 与 val_list.txt 的样本没有重复，因此可以用于扩充知识蒸馏任务的训练数据。
  - 即使引入了有标签的 extra_list.txt 中的图像，但是代码中没有使用标签信息，因此仍然可以视为无标签的模型蒸馏。
  - 蒸馏过程中，教师模型使用的预训练模型为 flowers102 数据集上的训练结果，学生模型使用的是 ImageNet1k 数据集上精度为 75.32% 的 MobileNetV3_large_x1_0 预训练模型。

配置文件中数据数量、模型结构、预训练地址以及训练的数据配置如下：

```
total_images: 7169
ARCHITECTURE:
    name: 'ResNet50_vd_distill_MobileNetV3_large_x1_0'
pretrained_model:
    - "./pretrained/flowers102_R50_vd_final/ppcls"
    - "./pretrained/MobileNetV3_large_x1_0_pretrained/”
TRAIN:
    file_list: "./dataset/flowers102/train_extra_list.txt"
```

最终的训练脚本如下所示。

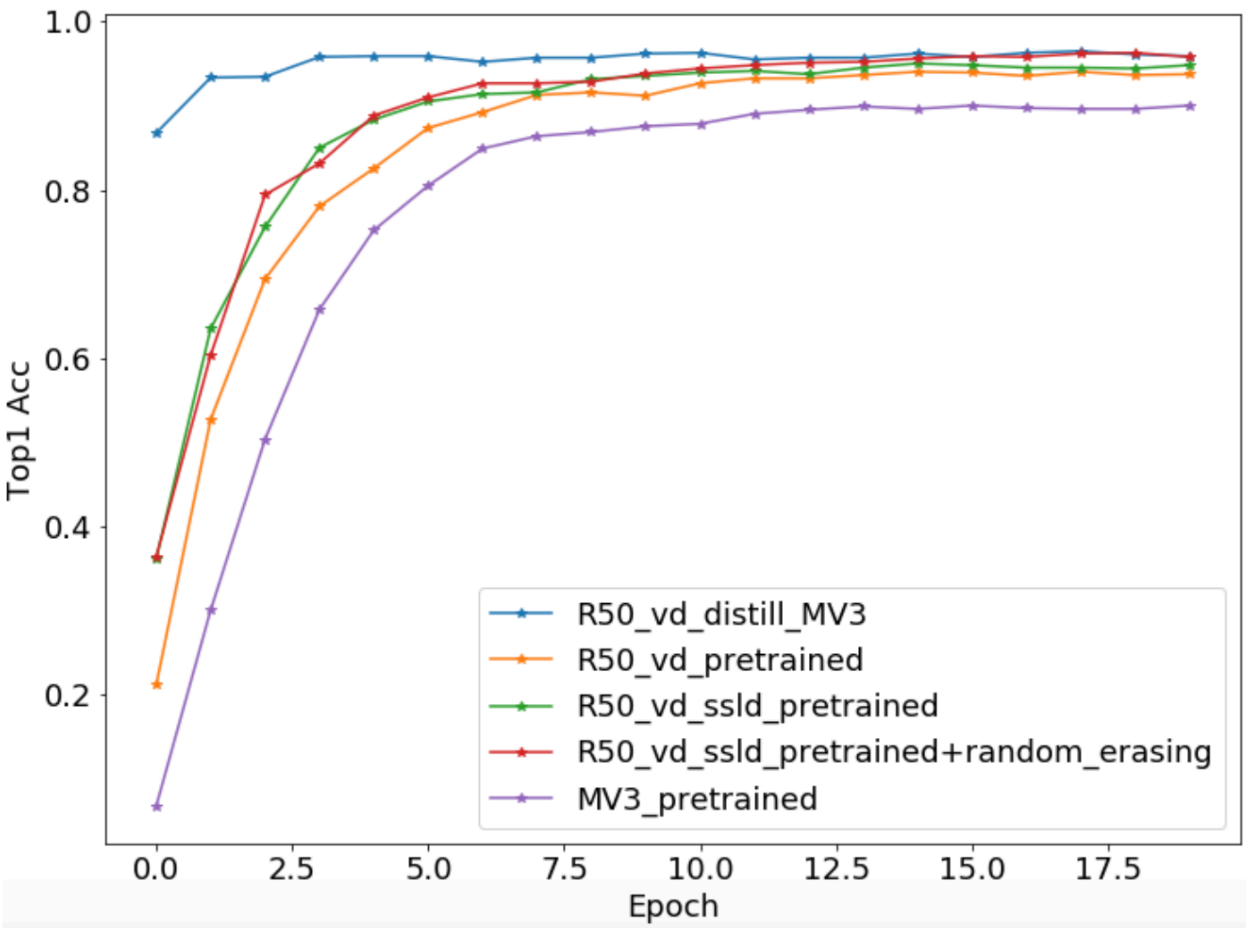
```
export CUDA_VISIBLE_DEVICES=0
python -m paddle.distributed.launch \
    --selected_gpus="0" \
    tools/train.py \
        -c ./configs/quick_start/R50_vd_distill_MV3_large_x1_0.yaml
```

最终 flowers102 验证集上的精度为 0.9647，结合更多的无标签数据，使用教师模型进行知识蒸馏，MobileNetV3 的精度涨幅高达 6.47%。

3.6 精度一览

- 下表给出了不同训练 yaml 文件对应的精度。

下图给出了不同配置文件在迭代过程中的 `Top1 Acc` 的精度曲线变化图。



- **注意**：flowers102 数据集图片数量较少，因此进行训练时，验证集的精度指标可能会有 1% 左右的波动。
- 更多训练及评估流程，请参考开始使用文档

1.3 数据说明

1.3.1 1. 简介

本文档介绍 ImageNet1k 和 flowers102 数据准备过程。

1.3.2 2. 数据集准备

- 数据格式按照如下结构组织数据，其中 train_list.txt 和 val_list.txt 的格式形如

```
# 每一行采用" 空格" 分隔图像路径与标注
```

(下页继续)

(续上页)

```
ILSVRC2012_val_00000001.JPEG 65
...
```

ImageNet1k

从官方下载数据后，按如下组织数据

```
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

从VGG 官方网站下载后的数据，解压后包括

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

将以上文件放置在 PaddleClas/dataset/flowers102/下

通过运行 generate_flowers102_list.py 生成 train_list.txt 和 val_list.txt

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

按照如下结构组织数据：

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

1.4 开始使用

请事先参考安装指南配置运行环境，并根据数据说明文档准备 ImageNet1k 数据，本章节下面所有的实验均以 ImageNet1k 数据集为例。

1.4.1 一、设置环境变量

设置 PYTHONPATH 环境变量：

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

1.4.2 二、模型训练与评估

PaddleClas 提供模型训练与评估脚本：`tools/train.py` 和 `tools/eval.py`

2.1 模型训练

按照如下方式启动模型训练。

```
# PaddleClas 通过 launch 方式启动多卡多进程训练
# 通过设置 FLAGS_selected_gpus 指定 GPU 运行卡号

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

- 输出日志示例如下：

```
epoch:0    train    step:13    loss:7.9561    top1:0.0156    top5:0.1094    lr:0.100000 ␣
↪  elapse:0.193
```

可以通过添加-o 参数来更新配置：

```
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/
```

- 输出日志示例如下：

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

也可以直接修改模型对应的配置文件更新配置。具体配置参数参考配置文档。

训练期间可以通过 VisualDL 实时观察 loss 变化，启动命令如下：

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

2.2 模型微调

- 30 分钟玩转 PaddleClas中包含大量模型微调的示例，可以参考该章节在特定的数据集上进行模型微调。

2.3 模型评估

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

可以更改 configs/eval.yaml 中的 `ARCHITECTURE.name` 字段和 pretrained_model 字段来配置评估模型，也可以通过-o 参数更新配置。

注意： 加载预训练模型时，需要指定预训练模型的前缀，例如预训练模型参数所在的文件夹为 `output/ResNet50_vd/19`，预训练模型参数的名称为 `output/ResNet50_vd/19/ppcls.pdparams`，则 `pretrained_model` 参数需要指定为 `output/ResNet50_vd/19/ppcls`，PaddleClas 会自动补齐.pdparams 的后缀。

1.4.3 三、模型推理

PaddlePaddle 提供三种方式进行预测推理，接下来介绍如何用预测引擎进行推理：首先，对训练好的模型进行转换：

```
python tools/export_model.py \
    --model=模型名字 \
    --pretrained_model=预训练模型路径 \
    --output_path=预测模型保存路径
```

之后，通过预测引擎进行推理：

```
python tools/infer/predict.py \
    -m model 文件路径 \
    -p params 文件路径 \
    -i 图片路径 \
    --use_gpu=1 \
    --use_tensorrt=True
```

更多使用方法和推理方式请参考分类预测框架。

1.5 配置说明

1.5.1 简介

本文档介绍了 PaddleClas 配置文件 (configs/*.yaml) 中各参数的含义，以便您更快的自定义或修改超参数配置。

1.5.2 配置详解

基础配置

学习率与优化器

学习率

优化器

数据读取器与数据处理

数据处理

mix 处理

CHAPTER 2

模型库

2.1 模型库概览

2.1.1 概述

基于 ImageNet1k 分类数据集，PaddleClas 支持的 23 种系列分类网络结构以及对应的 117 个图像分类预训练模型如下所示，训练技巧、每个系列网络结构的简单介绍和性能评估将在相应章节展现。

2.1.2 评估环境

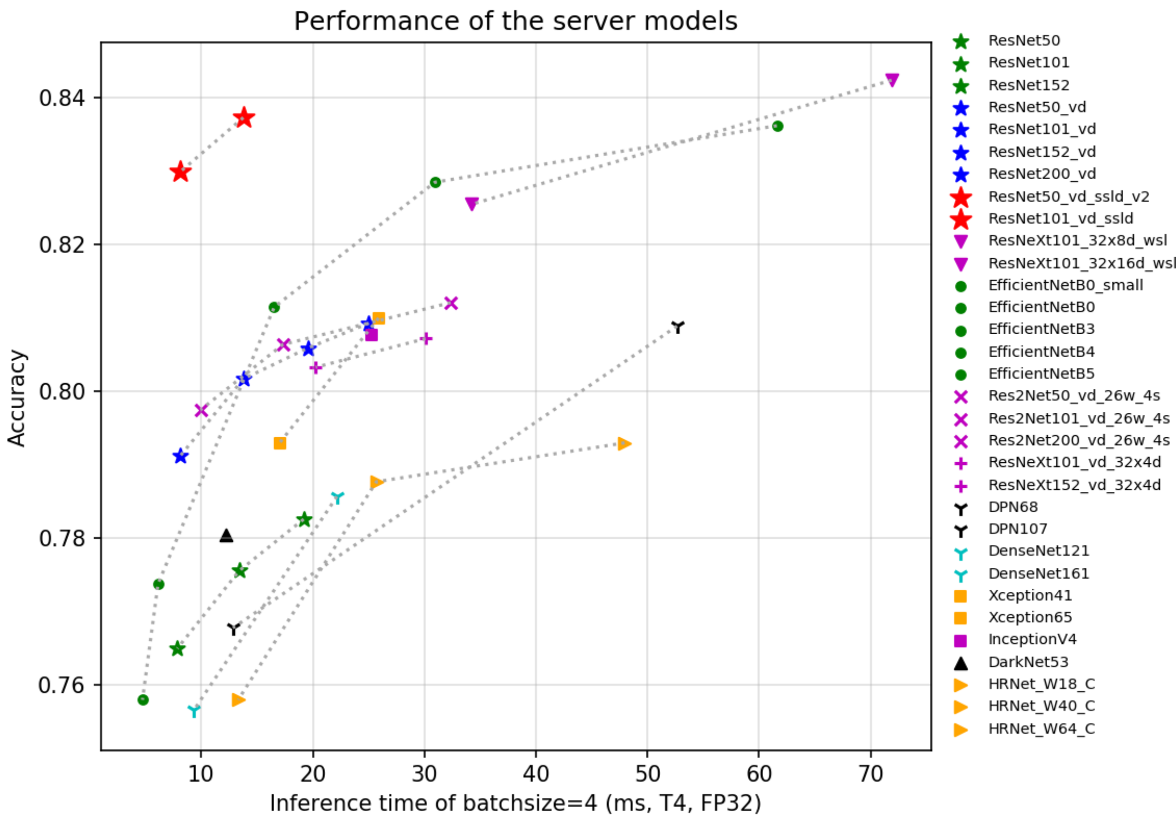
- CPU 的评估环境基于骁龙 855（SD855）。
- GPU 评估环境基于 V100 和 TensorRT，评估脚本如下。

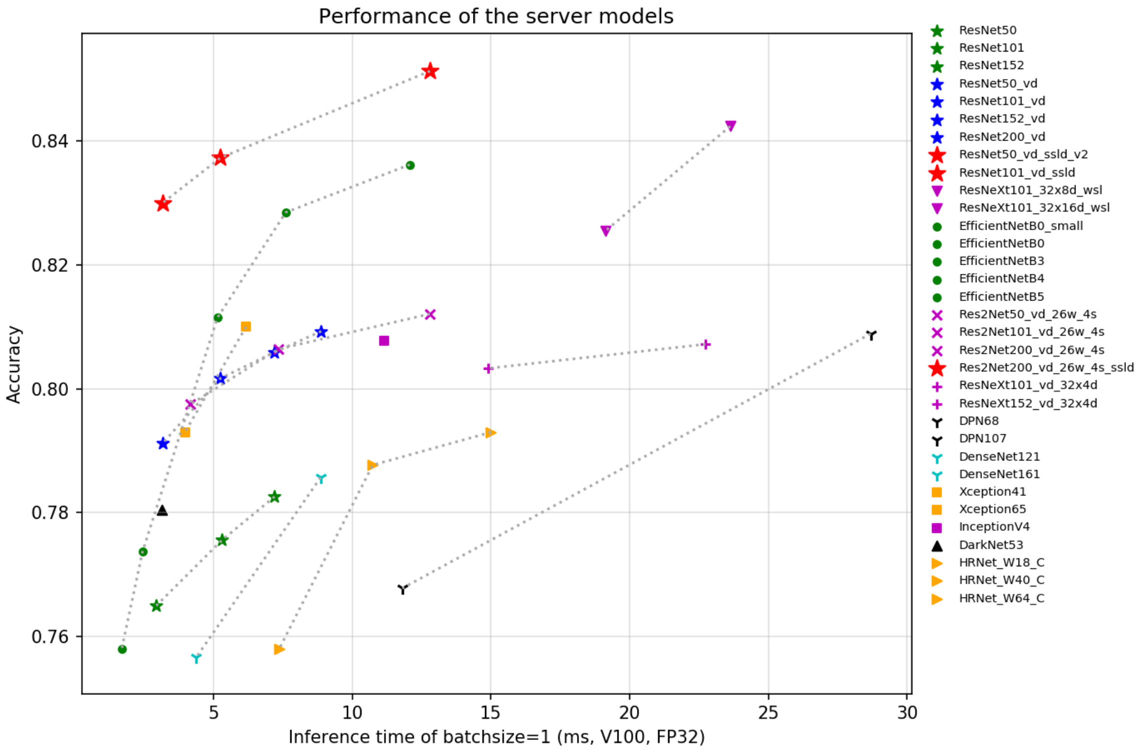
```
#!/usr/bin/env bash

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 \
```

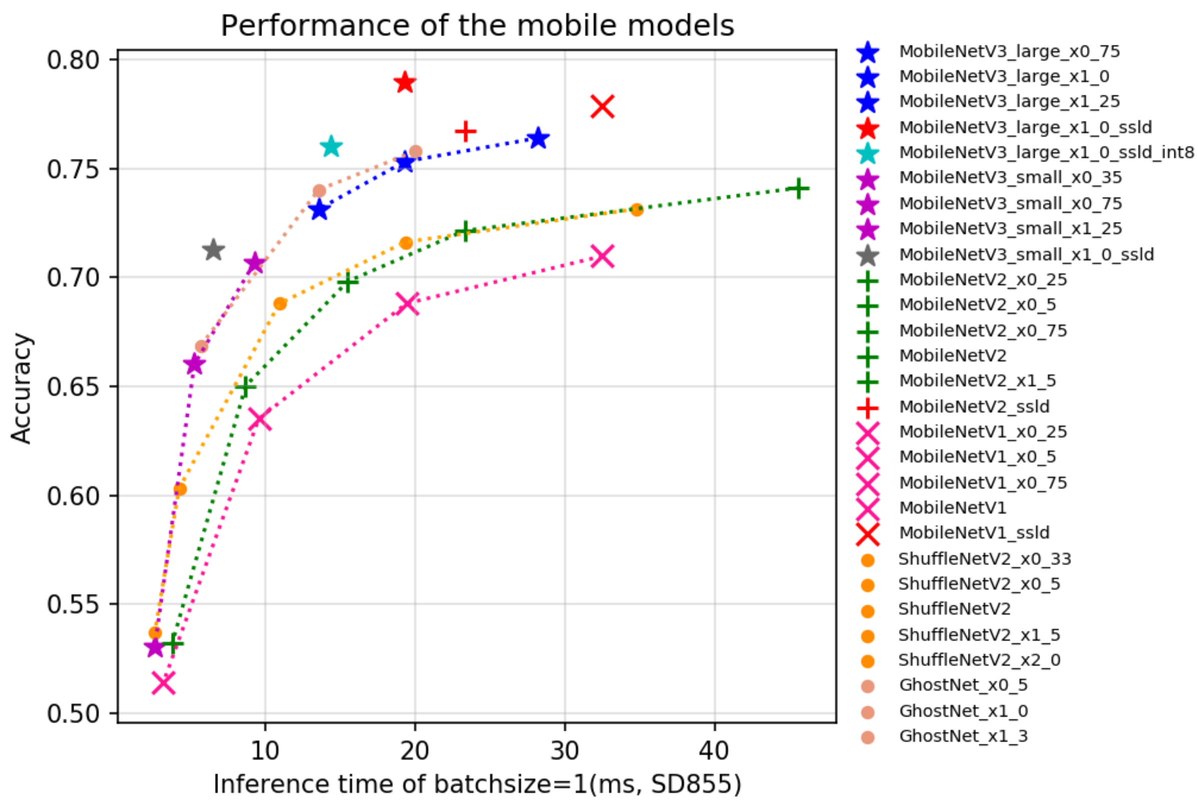
(下页继续)

(续上页)

```
    --use_tensorrt=True \
    --use_fp16=False \
    --batch_size=1
```




Performance of the server models
Accuracy
Inference time of batchsize=1 (ms, V100, FP32)
0.76
0.78
0.80
0.82
0.84
5
10
15
20
25
30
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
Res2Net200_vd_26w_4s_ssld
ResNeXt101_vd_32x4d
ResNeXt152_vd_32x4d
DPN68
DPN107
DenseNet121
DenseNet161
Xception41
Xception65
InceptionV4
DarkNet53
HRNet_W18_C
HRNet_W40_C
HRNet_W64_C



如果您觉得此文档对您有帮助，欢迎 star 我们的项目：https://github.com/PaddlePaddle/PaddleClas

2.1.3 预训练模型列表及下载地址

- ResNet 及其 Vd 系列
  - ResNet 系列 [1](论文地址)
    * ResNet18
    * ResNet34
    * ResNet50
    * ResNet101
    * ResNet152
  - ResNet_vc、ResNet_vd 系列 [2](论文地址)
    * 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

- 移动端系列
  - MobileNetV3 系列 [3](论文地址)
    * 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 系列 [4](论文地址)
    * MobileNetV2_x0_25
    * MobileNetV2_x0_5
    * MobileNetV2_x0_75
    * MobileNetV2

    * MobileNetV2_x1_5
    * MobileNetV2_x2_0
    * MobileNetV2_ssld
  - MobileNetV1 系列 [5](论文地址)
    * MobileNetV1_x0_25
    * MobileNetV1_x0_5
    * MobileNetV1_x0_75
    * MobileNetV1
    * MobileNetV1_ssld
  - ShuffleNetV2 系列 [6](论文地址)
    * ShuffleNetV2_x0_25
    * ShuffleNetV2_x0_33
    * ShuffleNetV2_x0_5
    * ShuffleNetV2
    * ShuffleNetV2_x1_5
    * ShuffleNetV2_x2_0
    * ShuffleNetV2_swish
  - GhostNet 系列 [23](论文地址)
    * GhostNet_x0_5
    * GhostNet_x1_0
    * GhostNet_x1_3
- SEResNeXt 与 Res2Net 系列
  - ResNeXt 系列 [7](论文地址)
    * ResNeXt50_32x4d
    * ResNeXt50_64x4d
    * ResNeXt101_32x4d
    * ResNeXt101_64x4d
    * ResNeXt152_32x4d
    * ResNeXt152_64x4d
  - ResNeXt_vd 系列

    * ResNeXt50_vd_32x4d
    * ResNeXt50_vd_64x4d
    * ResNeXt101_vd_32x4d
    * ResNeXt101_vd_64x4d
    * ResNeXt152_vd_32x4d
    * ResNeXt152_vd_64x4d
  - SE_ResNet_vd 系列 [*8*](论文地址)
    * SE_ResNet18_vd
    * SE_ResNet34_vd
    * SE_ResNet50_vd
  - SE_ResNeXt 系列
    * SE_ResNeXt50_32x4d
    * SE_ResNeXt101_32x4d
  - SE_ResNeXt_vd 系列
    * SE_ResNeXt50_vd_32x4d
    * SENet154_vd
  - Res2Net 系列 [*9*](论文地址)
    * Res2Net50_26w_4s
    * Res2Net50_vd_26w_4s
    * Res2Net50_14w_8s
    * Res2Net101_vd_26w_4s
    * Res2Net200_vd_26w_4s
    * Res2Net200_vd_26w_4s_ssld
- Inception 系列
  - GoogLeNet 系列 [*10*](论文地址)
    * GoogLeNet
  - Inception 系列 [*11*](论文地址)
    * InceptionV4
  - Xception 系列 [*12*](论文地址)
    * Xception41

    * Xception41_deeplab
    * Xception65
    * Xception65_deeplab
    * Xception71

- HRNet 系列
  - HRNet 系列 [*13*](论文地址)
    * HRNet_W18_C
    * HRNet_W30_C
    * HRNet_W32_C
    * HRNet_W40_C
    * HRNet_W44_C
    * HRNet_W48_C
    * HRNet_W64_C
- DPN 与 DenseNet 系列
  - DPN 系列 [*14*](论文地址)
    * DPN68
    * DPN92
    * DPN98
    * DPN107
    * DPN131
  - DenseNet 系列 [*15*](论文地址)
    * DenseNet121
    * DenseNet161
    * DenseNet169
    * DenseNet201
    * DenseNet264
- EfficientNet 与 ResNeXt101_wsl 系列
  - EfficientNet 系列 [*16*](论文地址)
    * EfficientNetB0_small
    * EfficientNetB0

        * EfficientNetB1
        * EfficientNetB2
        * EfficientNetB3
        * EfficientNetB4
        * EfficientNetB5
        * EfficientNetB6
        * EfficientNetB7
    - ResNeXt101_wsl 系列 [*17*](论文地址)
        * ResNeXt101_32x8d_wsl
        * ResNeXt101_32x16d_wsl
        * ResNeXt101_32x32d_wsl
        * ResNeXt101_32x48d_wsl
        * Fix_ResNeXt101_32x48d_wsl
- 其他模型
    - AlexNet 系列 [*18*](论文地址)
        * AlexNet
    - SqueezeNet 系列 [*19*](论文地址)
        * SqueezeNet1_0
        * SqueezeNet1_1
    - VGG 系列 [*20*](论文地址)
        * VGG11
        * VGG13
        * VGG16
        * VGG19
    - DarkNet 系列 [*21*](论文地址)
        * DarkNet53
    - ACNet 系列 [*22*](论文地址)
        * ResNet50_ACNet_deploy

注意：以上模型中 EfficientNetB1-B7 的预训练模型转自pytorch 版 EfficientNet，ResNeXt101_wsl 系列预训练模型转自官方 repo，剩余预训练模型均基于飞桨训练得到的，并在 configs 里给出了相应的训练超参数。

2.1.4 参考文献

[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.

[23] Han K, Wang Y, Tian Q, et al. GhostNet: More features from cheap operations[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020: 1580-1589.

2.2 训练技巧

2.2.1 1. 优化器的选择

自深度学习发展以来，就有很多关于优化器的研究者工作，优化器的目的是为了让损失函数尽可能的小，从而找到合适的参数来完成某项任务。目前业界主要用到的优化器有 SGD、RMSProp、Adam、AdaDelt 等，其中由于带 momentum 的 SGD 优化器广泛应用于学术界和工业界，所以我们发布的模型也大都使用该优化器来实现损失函数的梯度下降。带 momentum 的 SGD 优化器有两个劣势，其一是收敛速度慢，其二是初始学习率的设置需要依靠大量的经验，然而如果初始学习率设置得当并且迭代轮数充足，该优化器也会在众多的优化器中脱颖而出，使得其在验证集上获得更高的准确率。一些自适应学习率的优化器如 Adam、RMSProp 等，收敛速度往往比较快，但是最终的收敛精度会稍差一些。如果追求更快的收敛速度，我们推荐使用这些自适应学习率的优化器，如果追求更高的收敛精度，我们推荐使用带 momentum 的 SGD 优化器。

2.2.2 2. 学习率以及学习率下降策略的选择

学习率的选择往往和优化器以及数据和任务有关系。这里主要介绍以 momentum+SGD 作为优化器训练 ImageNet-1k 的学习率以及学习率下降的选择。

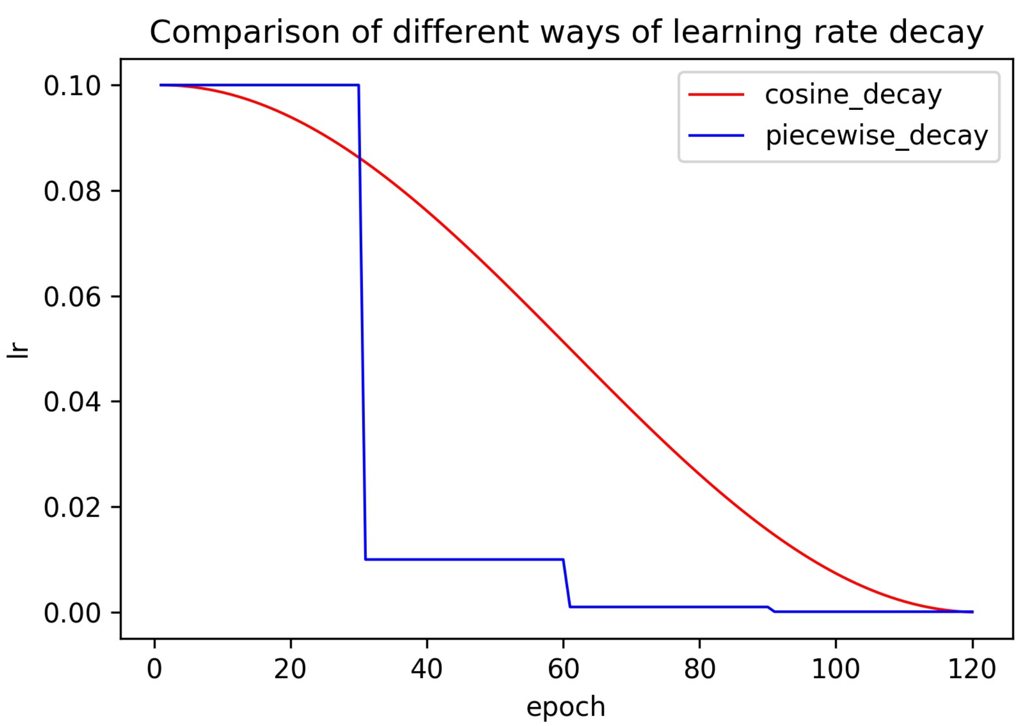
学习率的概念：

学习率是通过损失函数的梯度调整网络权重的超参数的速度。学习率越低，损失函数的变化速度就越慢。虽然使用低学习率可以确保不会错过任何局部极小值，但也意味着将花费更长的时间来进行收敛，特别是在被

困在高原区域的情况下。

学习率下降策略:

在整个训练过程中，我们不能使用同样的学习率来更新权重，否则无法到达最优点，所以需要在训练过程中调整学习率的大小。在训练初始阶段，由于权重处于随机初始化的状态，损失函数相对容易进行梯度下降，所以可以设置一个较大的学习率。在训练后期，由于权重参数已经接近最优值，较大的学习率无法进一步寻找最优值，所以需要设置一个较小的学习率。在训练整个过程中，很多研究者使用的学习率下降方式是 piecewise_decay，即阶梯式下降学习率，如在 ResNet50 标准的训练中，我们设置的初始学习率是 0.1，每 30epoch 学习率下降到原来的 1/10，一共迭代 120epoch。除了 piecewise_decay，很多研究者也提出了学习率的其他下降方式，如 polynomial_decay（多项式下降）、exponential_decay（指数下降）,cosine_decay（余弦下降）等，其中 cosine_decay 无需调整超参数，鲁棒性也比较高，所以成为现在提高模型精度首选的学习率下降方式。Cosine_decay 和 piecewise_decay 的学习率变化曲线如下图所示，容易观察到，在整个训练过程中，cosine_decay 都保持着较大的学习率，所以其收敛较为缓慢，但是最终的收敛效果较 peicewise_decay 更好一些。



另外，从图中我们也可以看到，cosine_decay 里学习率小的轮数较少，这样会影响到最终的精度，所以为了使得 cosine_decay 发挥更好的效果，建议迭代更多的轮数，如 200 轮。

warmup 策略

如果使用较大的 batch_size 训练神经网络时，我们建议您使用 warmup 策略。Warmup 策略顾名思义就是让学习率先预热一下，在训练初期我们不直接使用最大的学习率，而是用一个逐渐增大的学习率去训练网络，

当学习率增大到最高点时，再使用学习率下降策略中提到的学习率下降方式衰减学习率的值。实验表明，在 batch_size 较大时，warmup 可以稳定提升模型的精度。在训练 MobileNetV3 等 batch_size 较大的实验中，我们默认将 warmup 中的 epoch 设置为 5，即先用 5epoch 将学习率从 0 增加到最大值，再去做相应的学习率衰减。

2.2.3 3.batch_size 的选择

batch_size 是训练神经网络中的一个重要的超参数，该值决定了一次将多少数据送入神经网络参与训练。在论文 [1] 中，作者通过实验发现，当 batch_size 的值与学习率的值呈线性关系时，收敛精度几乎不受影响。在训练 ImageNet 数据时，大部分的神经网络选择的初始学习率为 0.1，batch_size 是 256，所以根据实际的模型大小和显存情况，可以将学习率设置为 0.1*k,batch_size 设置为 256*k。

2.2.4 4.weight_decay 的选择

过拟合是机器学习中常见的一个名词，简单理解即为模型在训练数据上表现很好，但在测试数据上表现较差，在卷积神经网络中，同样存在过拟合的问题，为了避免过拟合，很多正则方式被提出，其中，weight_decay 是其中一个广泛使用的避免过拟合的方式。Weight_decay 等价于在最终的损失函数后添加 L2 正则化，L2 正则化使得网络的权重倾向于选择更小的值，最终整个网络中的参数值更趋向于 0，模型的泛化性能相应提高。在各大深度学习框架的实现中，该值表达的含义是 L2 正则前的系数，在 paddle 框架中，该值的名称是 l2_decay，所以以下都称其为 l2_decay。该系数越大，表示加入的正则越强，模型越趋于欠拟合状态。在训练 ImageNet 的任务中，大多数的网络将该参数值设置为 1e-4，在一些小的网络如 MobileNet 系列网络中，为了避免网络欠拟合，该值设置为 1e-5~4e-5 之间。当然，该值的设置也和具体的数据集有关系，当任务的数据集较大时，网络本身趋向于欠拟合状态，可以将该值适当减小，当任务的数据集较小时，网络本身趋向于过拟合状态，可以将该值适当增大。下表展示了 MobileNetV1_x0_25 在 ImageNet-1k 上使用不同 l2_decay 的精度情况。由于 MobileNetV1_x0_25 是一个比较小的网络，所以 l2_decay 过大会使网络趋向于欠拟合状态，所以在该网络中，相对 1e-4，3e-5 是更好的选择。

另外，该值的设置也和训练过程中是否使用其他正则化有关系。如果训练过程中的数据预处理比较复杂，相当于训练任务变的更难，可以将该值适当减小，下表展示了在 ImageNet-1k 上，ResNet50 在使用 randaugment 预处理方式后使用不同 l2_decay 的精度。容易观察到，在任务变难后，使用更小的 l2_decay 有助于模型精度的提升。

综上所述，l2_decay 可以根据具体的任务和模型去做相应的调整，通常简单的任务或者较大的模型，推荐使用较大的 l2_decay, 复杂的任务或者较小的模型，推荐使用较小的 l2_decay。

2.2.5 5.label_smoothing 的选择

Label_smoothing 是深度学习中的一种正则化方法，其全称是 Label Smoothing Regularization(LSR)，即标签平滑正则化。在传统的分类任务计算损失函数时，是将真实的 one hot 标签与神经网络的输出做相应的交叉熵计算，而 label_smoothing 是将真实的 one hot 标签做一个标签平滑的处理，使得网络学习的标签不再是一个 hard label，而是一个有概率值的 soft label，其中在类别对应的位置的概率最大，其他位置概率是一个非常小的数。具体的计算方式参见论文 [2]。在 label_smoothing 里，有一个 epsilon 的参数值，该值描述

了将标签软化的程度，该值越大，经过 label smoothing 后的标签向量的标签概率值越小，标签越平滑，反之，标签越趋向于 hard label，在训练 ImageNet-1k 的实验里通常将该值设置为 0.1。在训练 ImageNet-1k 的实验中，我们发现，ResNet50 大小级别及其以上的模型在使用 label_smooting 后，精度有稳定的提升。下表展示了 ResNet50_vd 在使用 label_smoothing 前后的精度指标。

同时，由于 label_smoohing 相当于一种正则方式，在相对较小的模型上，精度提升不明显甚至会有所下降，下表展示了 ResNet18 在 ImageNet-1k 上使用 label_smoothing 前后的精度指标。可以明显看到，在使用 label_smoothing 后，精度有所下降。

综上所述，较大的模型使用 label_smoohing 可以有效提升模型的精度，较小的模型使用 label_smoohing 可能会降低模型的精度，所以在决定是否使用 label_smoohing 前，需要评估模型的大小和任务的难易程度。

2.2.6 6. 针对小模型更改图片的 crop 面积与拉伸变换程度

在 ImageNet-1k 数据的标准预处理中，random_crop 函数中定义了 scale 和 ratio 两个值，两个值分别确定了图片 crop 的大小和图片的拉伸程度，其中 scale 的默认取值范围是 0.08-1(lower_scale-upper_scale),ratio 的默认取值范围是 3/4-4/3(lower_ratio-upper_ratio)。在非常小的网络训练中，此类数据增强会使得网络欠拟合，导致精度有所下降。为了提升网络的精度，可以使其数据增强变的更弱，即增大图片的 crop 区域或者减弱图片的拉伸变换程度。我们可以分别通过增大 lower_scale 的值或缩小 lower_ratio 与 upper_scale 的差距来实现更弱的图片变换。下表列出了使用不同 lower_scale 训练 MobileNetV2_x0_25 的精度，可以看到，增大图片的 crop 区域面积后训练精度和验证精度均有提升。

2.2.7 7. 使用数据增广方式提升精度

一般来说，数据集的规模对性能影响至关重要，但是图片的标注往往比较昂贵，所以有标注的图片数量往往比较稀少，在这种情况下，数据的增广尤为重要。在训练 ImageNet-1k 的标准数据增广中，主要使用了 random_crop 与 random_flip 两种数据增广方式，然而，近些年，越来越多的数据增广方式被提出，如 cutout、mixup、cutmix、AutoAugment 等。实验表明，这些数据的增广方式可以有效提升模型的精度，下表列出了 ResNet50 在 8 种不同的数据增广方式的表现，可以看出，相比 baseline，所有的数据增广方式均有收益，其中 cutmix 是目前最有效的数据增广。更多数据增广的介绍请参考**数据增广章节**。

2.2.8 8. 通过 train_acc 和 test_acc 确定调优策略

在训练网络的过程中，通常会打印每一个 epoch 的训练集准确率和验证集准确率，二者刻画了该模型在两个数据集上的表现。通常来说，训练集的准确率比验证集准确率微高或者二者相当是比较不错的状态。如果发现训练集的准确率比验证集高很多，说明在这个任务上已经过拟合，需要在训练过程中加入更多的正则，如增大 l2_decay 的值，加入更多的数据增广策略，加入 label_smoothing 策略等；如果发现训练集的准确率比验证集低一些，说明在这个任务上可能欠拟合，需要在训练过程中减弱正则效果，如减小 l2_decay 的值，减少数据增广方式，增大图片 crop 区域面积，减弱图片拉伸变换，去除 label_smoothing 等。

2.2.9 9. 通过已有的预训练模型提升自己的数据集的精度

在现阶段计算机视觉领域中，加载预训练模型来训练自己的任务已成为普遍的做法，相比从随机初始化开始训练，加载预训练模型往往可以提升特定任务的精度。一般来说，业界广泛使用的预训练模型是通过训练 128 万张图片 1000 类的 ImageNet-1k 数据集得到的，该预训练模型的 fc 层权重是是一个 k*1000 的矩阵，其中 k 是 fc 层以前的神经元数，在加载预训练权重时，无需加载 fc 层的权重。在学习率方面，如果您的任务训练的数据集特别小（如小于 1 千张），我们建议你使用较小的初始学习率，如 0.001（batch_size:256, 下同），以免较大的学习率破坏预训练权重。如果您的训练数据集规模相对较大（大于 10 万），我们建议你尝试更大的初始学习率，如 0.01 或者更大。

如果您觉得此文档对您有帮助，欢迎 star 我们的项目：https://github.com/PaddlePaddle/PaddleClas

2.2.10 参考文献

[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 及其 Vd 系列

2.3.1 概述

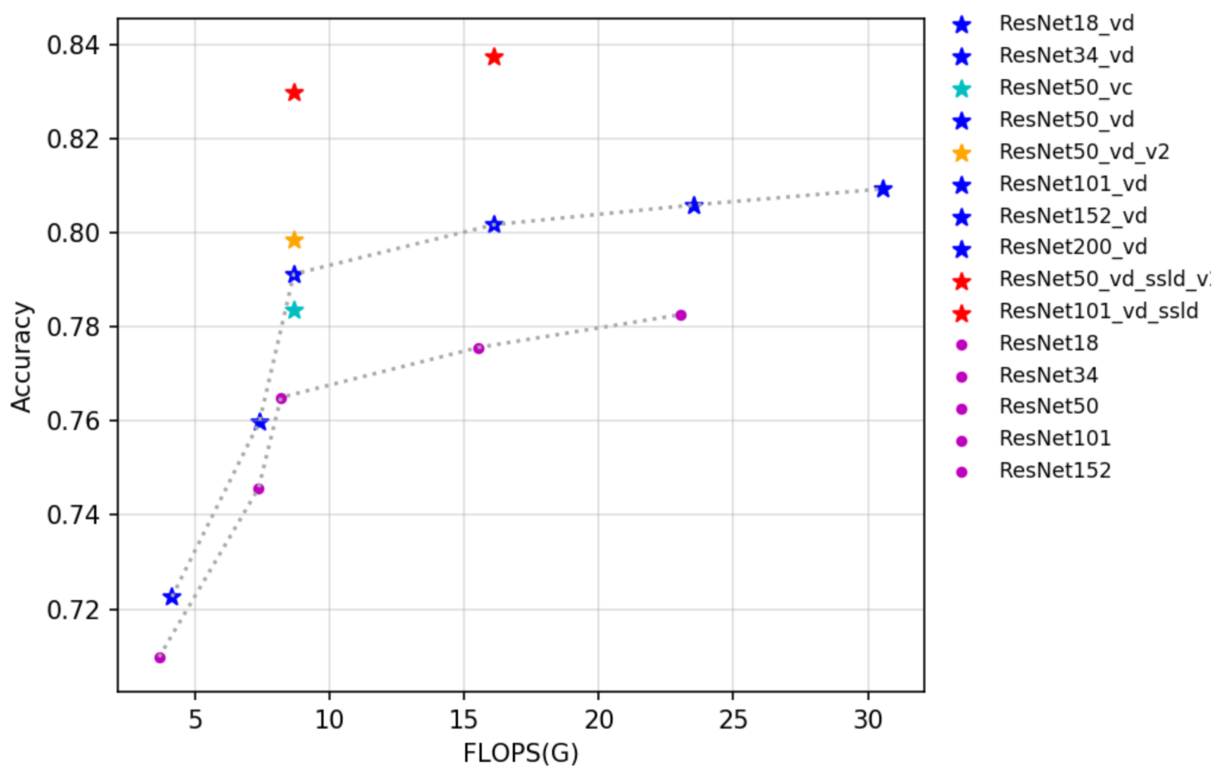
ResNet 系列模型是在 2015 年提出的，一举在 ILSVRC2015 比赛中取得冠军，top5 错误率为 3.57%。该网络创新性的提出了残差结构，通过堆叠多个残差结构从而构建了 ResNet 网络。实验表明使用残差块可以有效地提升收敛速度和精度。

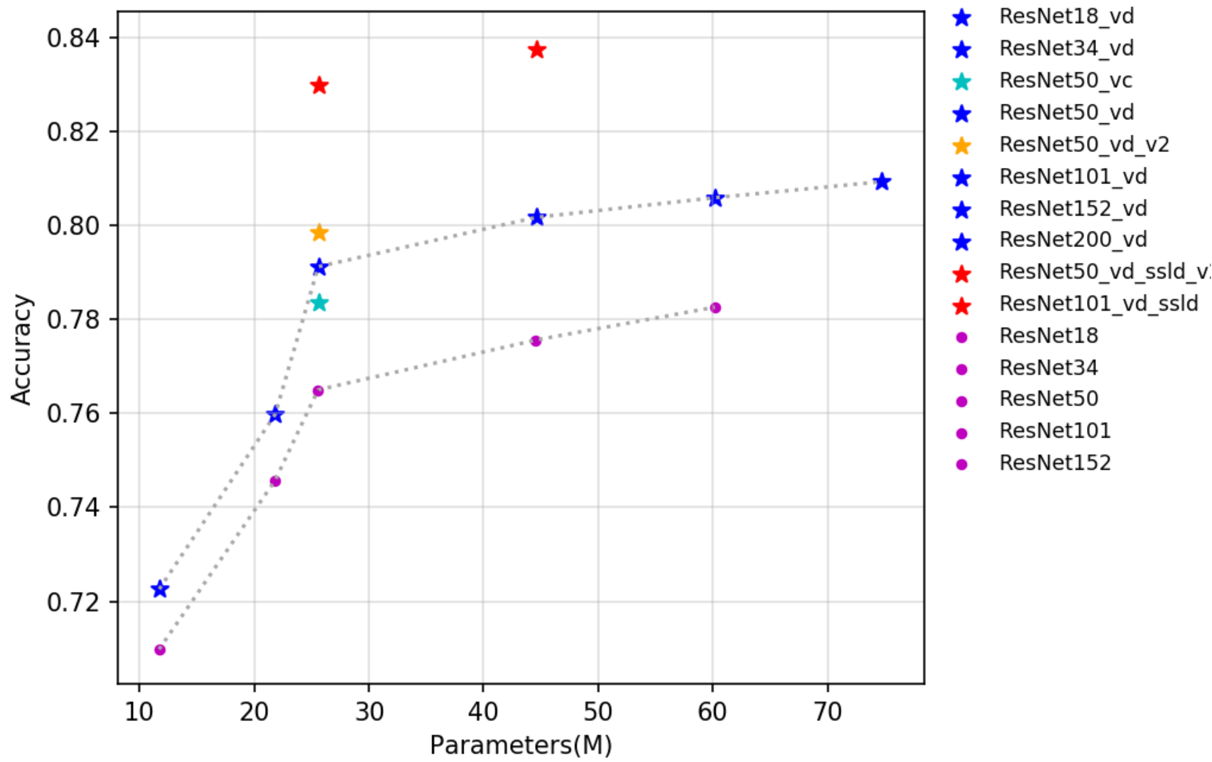
斯坦福大学的 Joyce Xu 将 ResNet 称为「真正重新定义了我们看待神经网络的方式」的三大架构之一。由于 ResNet 卓越的性能，越来越多的来自学术界和工业界学者和工程师对其结构进行了改进，比较出名的有 Wide-ResNet, ResNet-vc ,ResNet-vd, Res2Net 等，其中 ResNet-vc 与 ResNet-vd 的参数量和计算量与 ResNet 几乎一致，所以在此我们将其与 ResNet 统一归为 ResNet 系列。

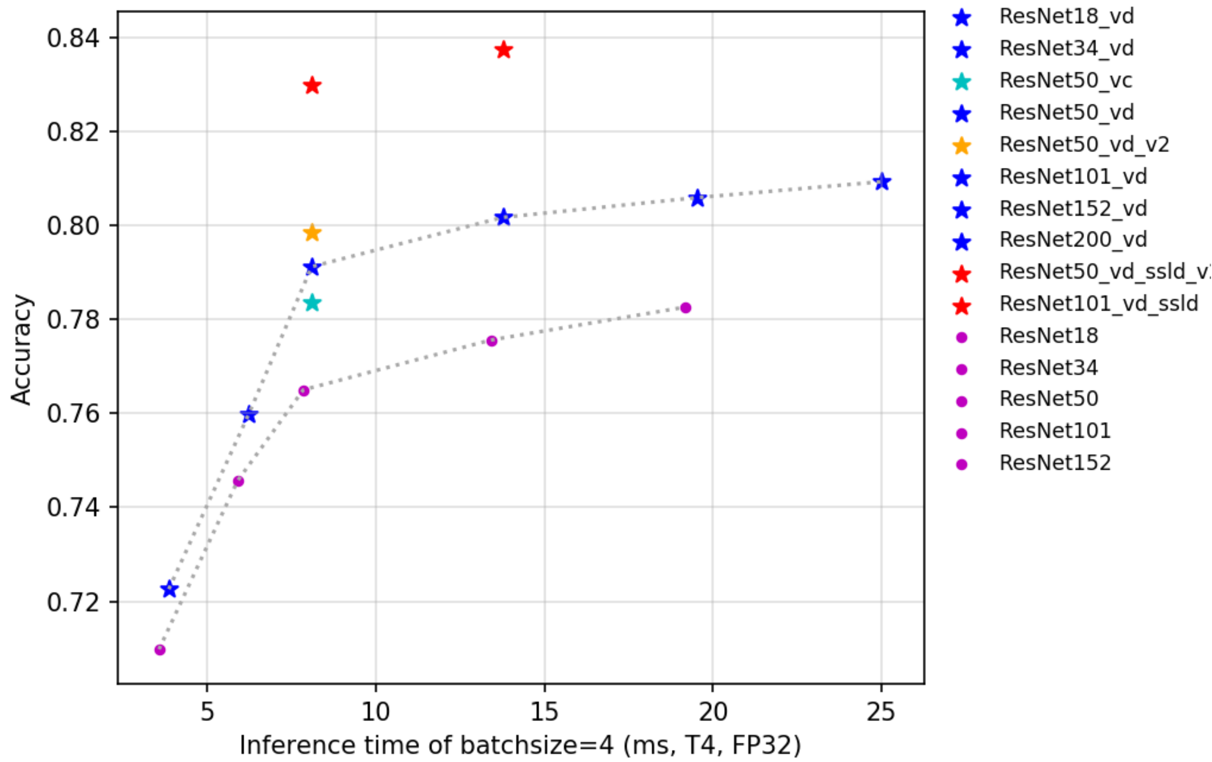
本次发布 ResNet 系列的模型包括 ResNet50，ResNet50_vd，ResNet50_vd_ssld，ResNet200_vd 等 14 个预训练模型。在训练层面上，ResNet 的模型采用了训练 ImageNet 的标准训练流程，而其余改进版模型采用了更多的训练策略，如 learning rate 的下降方式采用了 cosine decay，引入了 label smoothing 的标签正则方式，在数据预处理加入了 mixup 的操作，迭代总轮数从 120 个 epoch 增加到 200 个 epoch。

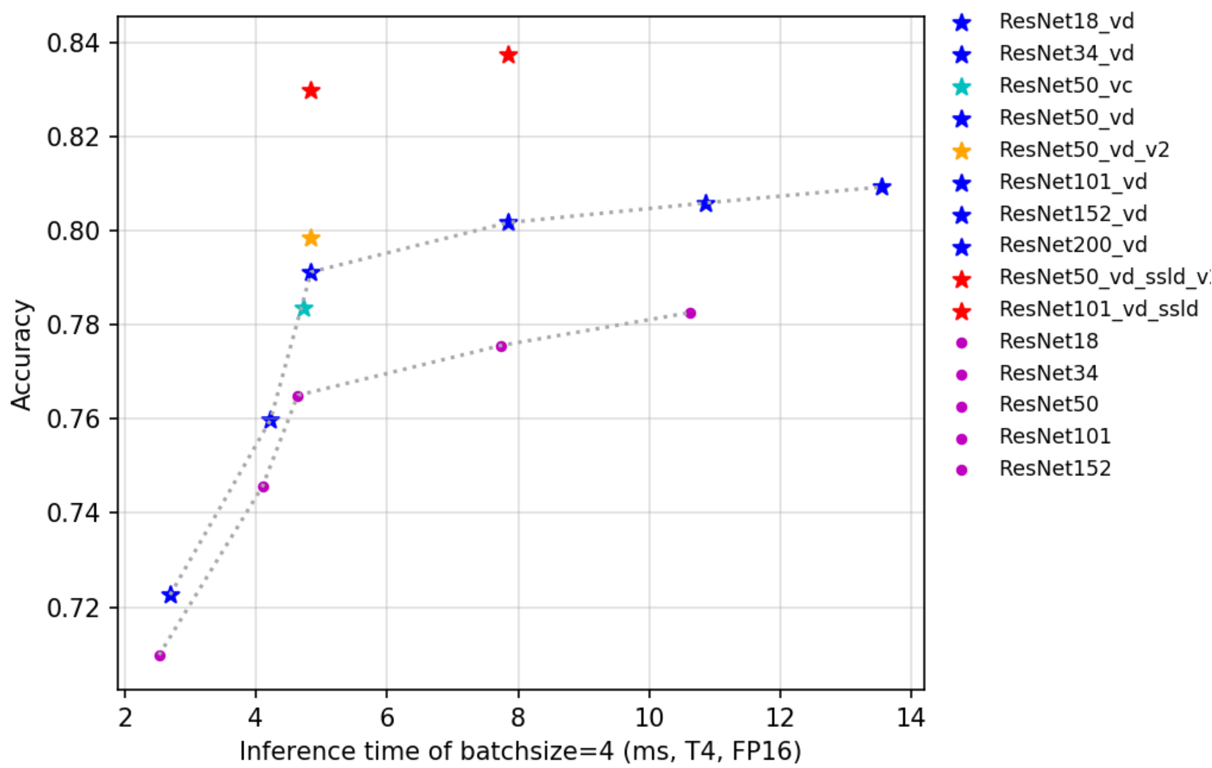
其中，ResNet50_vd_v2 与 ResNet50_vd_ssld 采用了知识蒸馏，保证模型结构不变的情况下，进一步提升了模型的精度，具体地，ResNet50_vd_v2 的 teacher 模型是 ResNet152_vd（top1 准确率 80.59%），数据选用的是 ImageNet-1k 的训练集，ResNet50_vd_ssld 的 teacher 模型是 ResNeXt101_32x16d_wsl（top1 准确率 84.2%），数据选用结合了 ImageNet-1k 的训练集和 ImageNet-22k 挖掘的 400 万数据。知识蒸馏的具体方法正在持续更新中。

该系列模型的 FLOPS、参数量以及 T4 GPU 上的预测耗时如下图所示。




Accuracy
Parameters(M)
ResNet18_vd
ResNet34_vd
ResNet50_vc
ResNet50_vd
ResNet50_vd_v2
ResNet101_vd
ResNet152_vd
ResNet200_vd
ResNet101_vd_ssld
ResNet18
ResNet34
ResNet50
ResNet101
ResNet152


Accuracy
0.84
0.82
0.80
0.78
0.76
0.74
0.72
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
ResNet101_vd_ssld
ResNet18
ResNet34
ResNet50
ResNet101
ResNet152



通过上述曲线可以看出，层数越多，准确率越高，但是相应的参数量、计算量和延时都会增加。ResNet50_vd_ssld 通过用更强的 teacher 和更多的数据，将其在 ImageNet-1k 上的验证集 top-1 精度进一步提高，达到了 82.39%，刷新了 ResNet50 系列模型的精度。

2.3.2 精度、FLOPS 和参数量

- 注：`ResNet50_vd_ssld_v2` 是在 `ResNet50_vd_ssld` 训练策略的基础上加上 AutoAugment 训练得到，`Fix_ResNet50_vd_ssld_v2` 是固定 `ResNet50_vd_ssld_v2` 除 FC 层外所有的网络参数，在 320x320 的图像输入分辨率下，基于 ImageNet1k 数据集微调得到。

2.3.3 基于 V100 GPU 的预测速度

2.3.4 基于 T4 GPU 的预测速度

2.4 移动端系列

2.4.1 概述

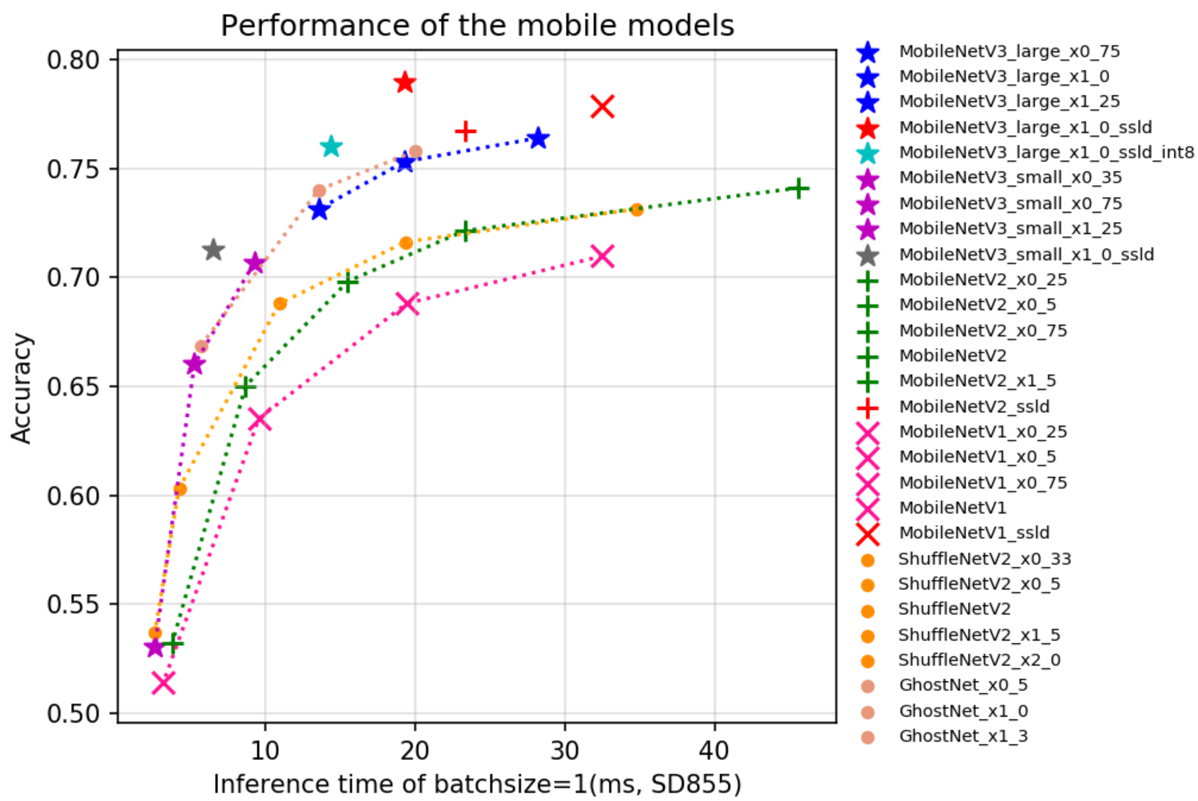
MobileNetV1 是 Google 于 2017 年发布的用于移动设备或嵌入式设备中的网络。该网络将传统的卷积操作替换深度可分离卷积，即 Depthwise 卷积和 Pointwise 卷积的组合，相比传统的卷积操作，该组合可以大大节省参数量和计算量。与此同时，MobileNetV1 也可以用于目标检测、图像分割等其他视觉任务中。

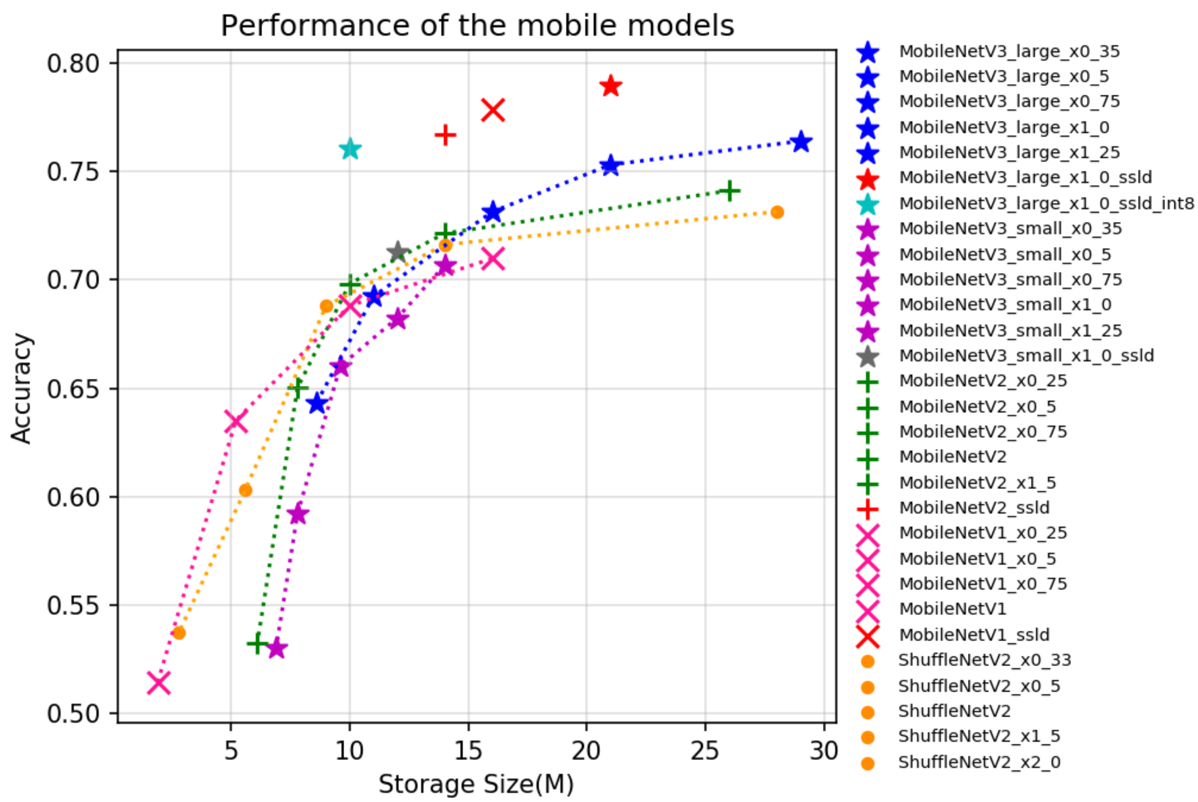
MobileNetV2 是 Google 继 MobileNetV1 提出的一种轻量级网络。相比 MobileNetV1，MobileNetV2 提出了 Linear bottlenecks 与 Inverted residual block 作为网络基本结构，通过大量地堆叠这些基本模块，构成了 MobileNetV2 的网络结构。最终，在 FLOPS 只有 MobileNetV1 的一半的情况下取得了更高的分类精度。

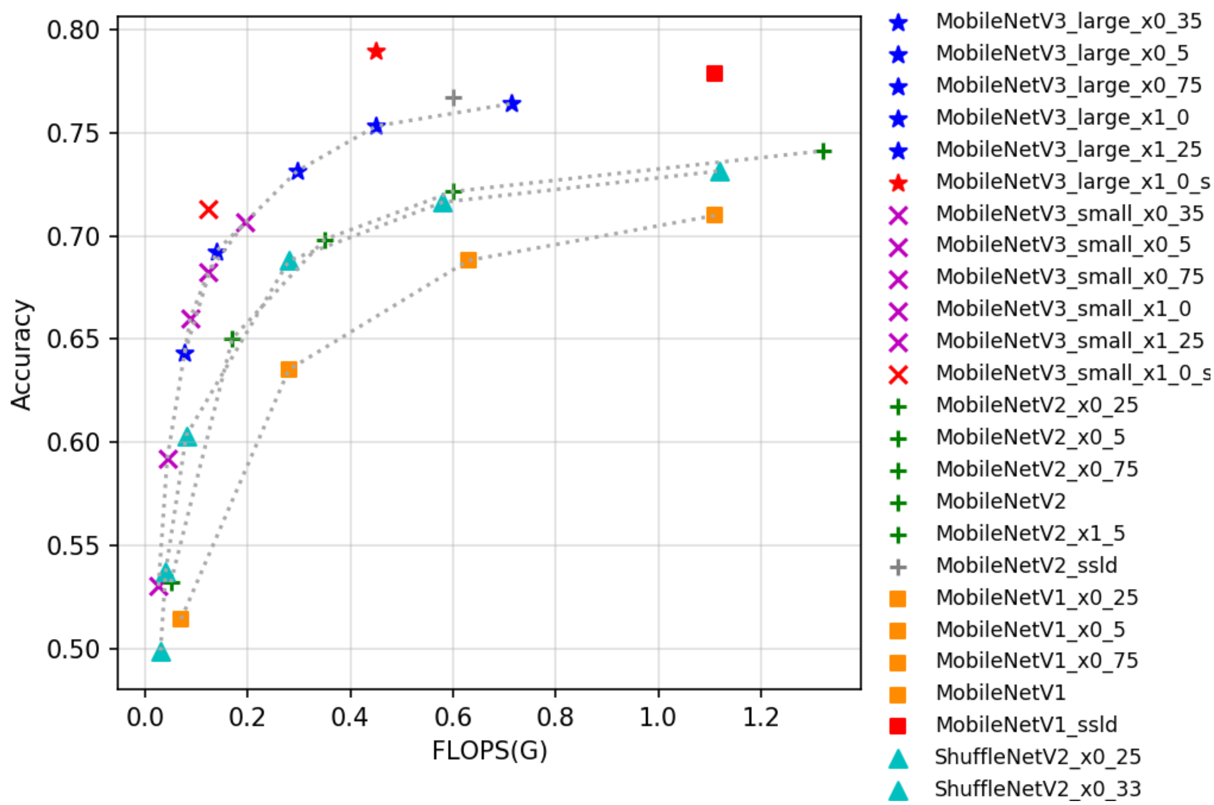
ShuffleNet 系列网络是旷视提出的轻量化网络结构，到目前为止，该系列网络一共有两种典型的结构，即 ShuffleNetV1 与 ShuffleNetV2。ShuffleNet 中的 Channel Shuffle 操作可以将组间的信息进行交换，并且可以实现端到端的训练。在 ShuffleNetV2 的论文中，作者提出了设计轻量级网络的四大准则，并且根据四大准则与 ShuffleNetV1 的不足，设计了 ShuffleNetV2 网络。

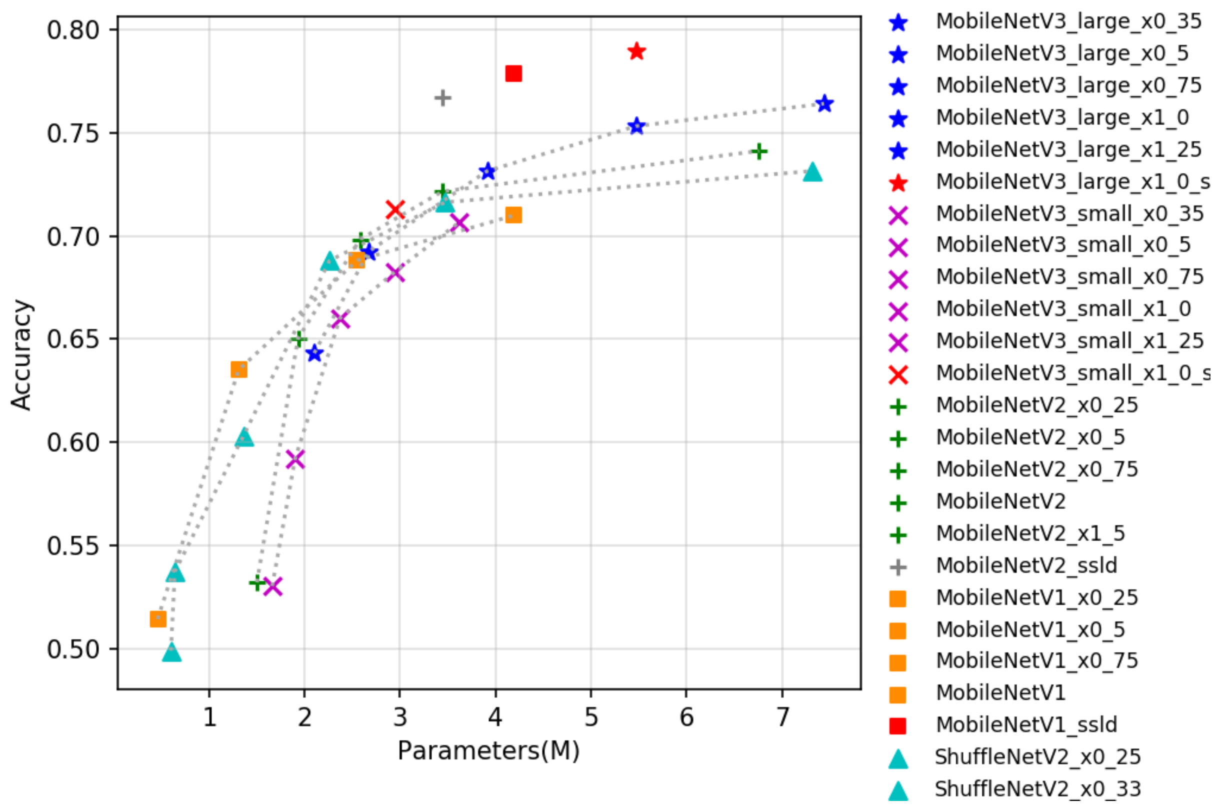
MobileNetV3 是 Google 于 2019 年提出的一种基于 NAS 的新的轻量级网络，为了进一步提升效果，将 relu 和 sigmoid 激活函数分别替换为 hard_swish 与 hard_sigmoid 激活函数，同时引入了一些专门减小网络计算量的改进策略。

GhosttNet 是华为于 2020 年提出的一种全新的轻量化网络结构，通过引入 ghost module，大大减缓了传统深度网络中特征的冗余计算问题，使得网络的参数量和计算量大大降低。


Performance of the mobile models
Accuracy
Inference time of batchsize=1(ms, SD855)
0.80
0.75
0.70
0.65
0.60
0.55
0.50
10
20
30
40
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_75
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
GhostNet_x0_5
GhostNet_x1_0
GhostNet_x1_3


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



目前 PaddleClas 开源的的移动端系列的预训练模型一共有 35 个，其指标如图所示。从图片可以看出，越新的轻量级模型往往有更优的表现，MobileNetV3 代表了目前主流的轻量级神经网络结构。在 MobileNetV3 中，作者为了获得更高的精度，在 global-avg-pooling 后使用了 1x1 的卷积。该操作大幅提升了参数量但对计算量影响不大，所以如果从存储角度评价模型的优异程度，MobileNetV3 优势不是很大，但由于其更小的计算量，使得其有更快的推理速度。此外，我们模型库中的 ssld 蒸馏模型表现优异，从各个考量角度下，都刷新了当前轻量级模型的精度。由于 MobileNetV3 模型结构复杂，分支较多，对 GPU 并不友好，GPU 预测速度不如 MobileNetV1。GhostNet 于 2020 年提出，通过引入 ghost 的网络设计理念，大大降低了计算量和参数量，同时在精度上也超过前期最高的 MobileNetV3 网络结构。

2.4.2 精度、FLOPS 和参数量

2.4.3 基于 SD855 的预测速度和存储大小

2.4.4 基于 T4 GPU 的预测速度

2.5 SEResNeXt 与 Res2Net 系列

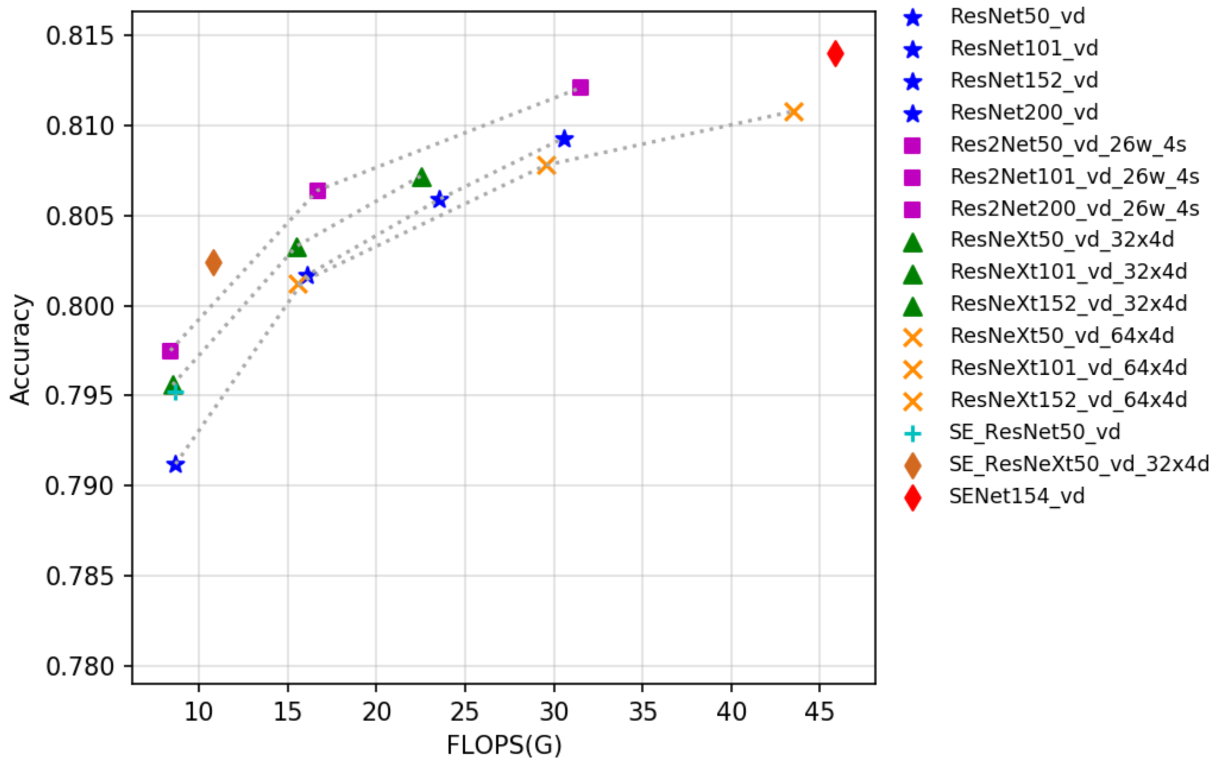
2.5.1 概述

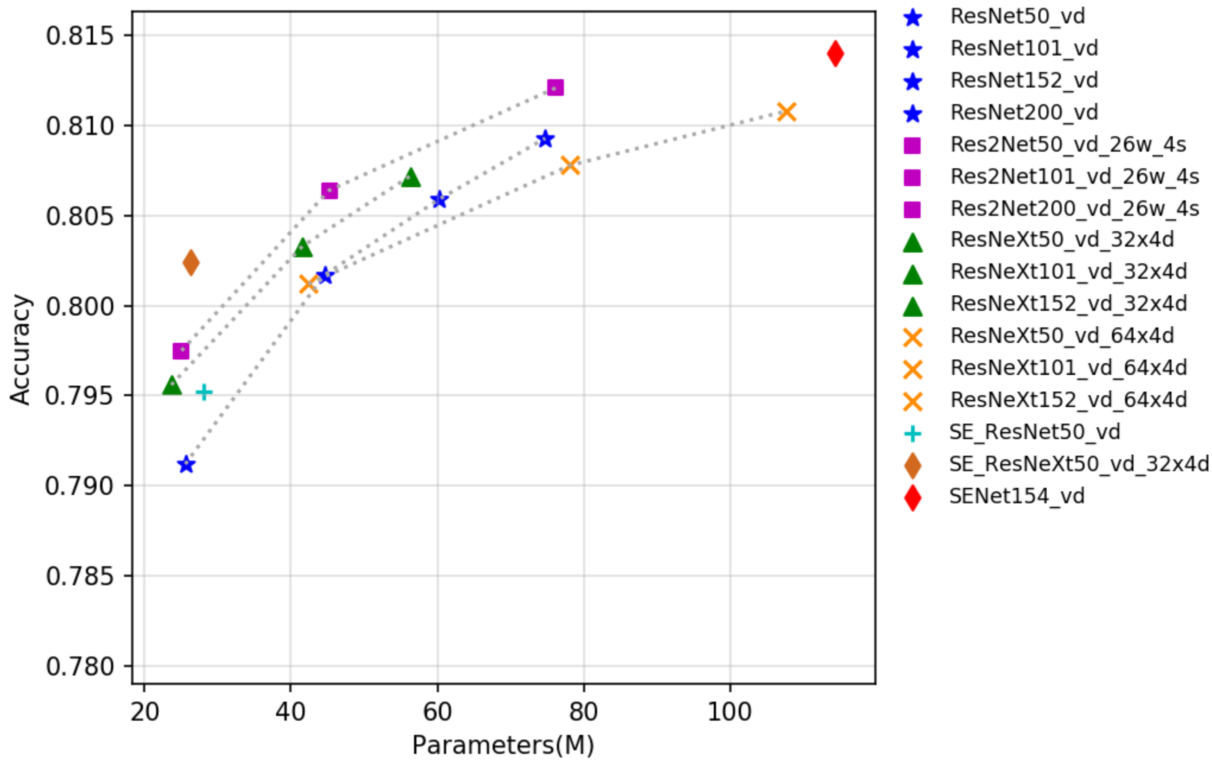
ResNeXt 是 ResNet 的典型变种网络之一，ResNeXt 发表于 2017 年的 CVPR 会议。在此之前，提升模型精度的方法主要集中在将网络变深或者变宽，这样增加了参数量和计算量，推理速度也会相应变慢。ResNeXt 结构提出了通道分组（cardinality）的概念，作者通过实验发现增加通道的组数比增加深度和宽度更有效。其可以在不增加参数复杂度的前提下提高准确率，同时还减少了参数的数量，所以是比较成功的 ResNet 的变种。

SENet 是 2017 年 ImageNet 分类比赛的冠军方案，其提出了一个全新的 SE 结构，该结构可以迁移到任何其他网络中，其通过控制 scale 的大小，把每个通道间重要的特征增强，不重要的特征减弱，从而让提取的特征指向性更强。

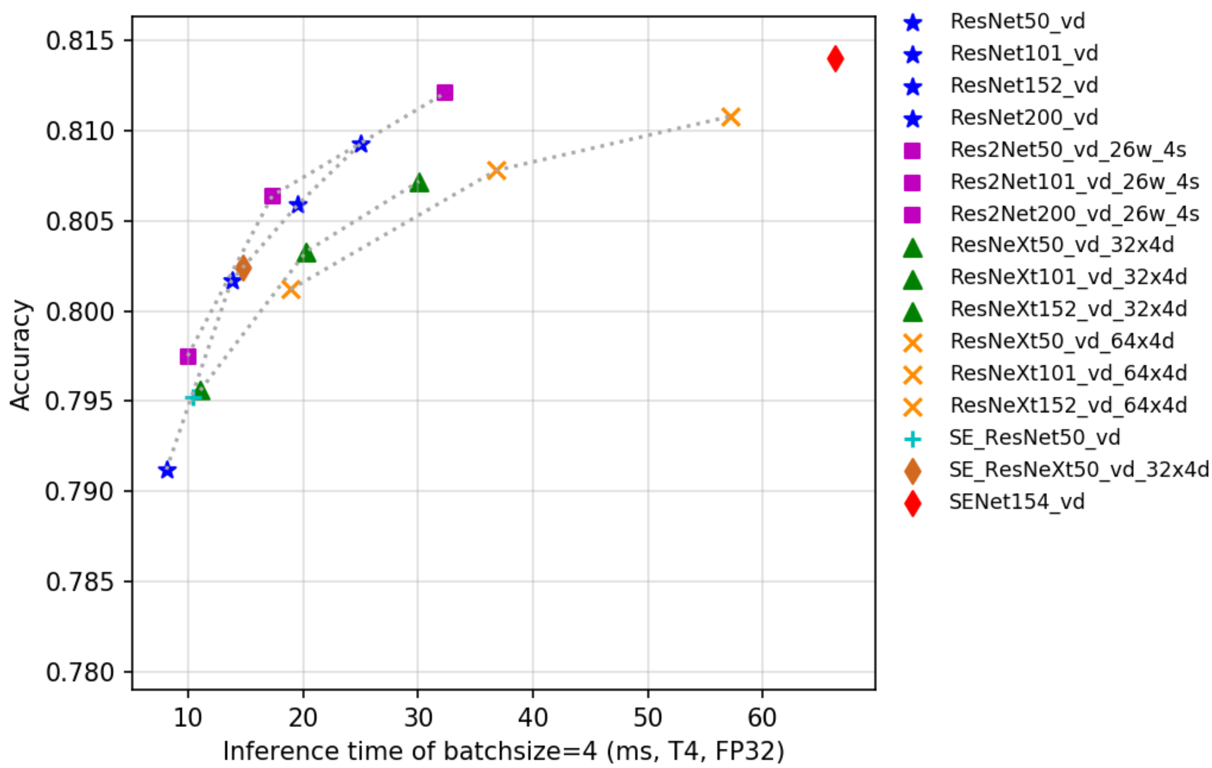
Res2Net 是 2019 年提出的一种全新的对 ResNet 的改进方案，该方案可以和现有其他优秀模块轻松整合，在不增加计算负载量的情况下，在 ImageNet、CIFAR-100 等数据集上的测试性能超过了 ResNet。Res2Net 结构简单，性能优越，进一步探索了 CNN 在更细粒度级别的多尺度表示能力。Res2Net 揭示了一个新的提升模型精度的维度，即 scale，其是除了深度、宽度和基数的现有维度之外另外一个必不可少的更有效的因素。该网络在其他视觉任务如目标检测、图像分割等也有相当不错的表现。

该系列模型的 FLOPS、参数量以及 T4 GPU 上的预测耗时如下图所示。

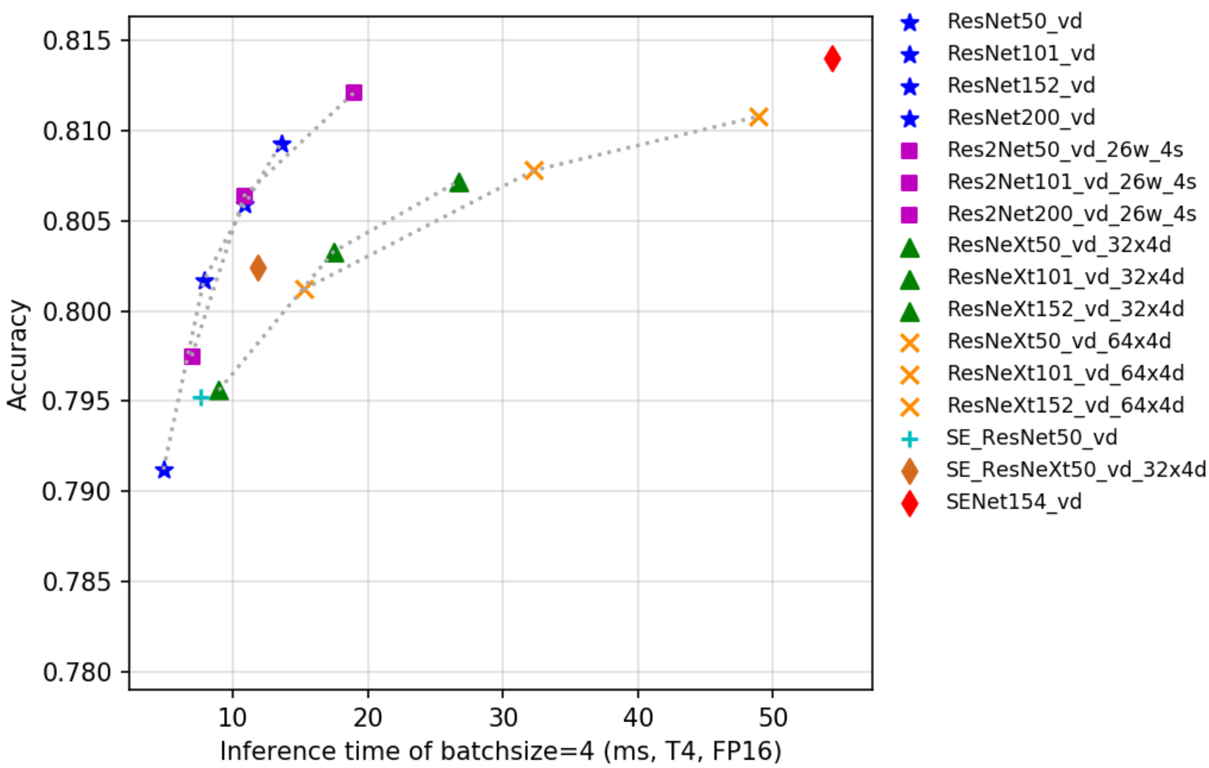

0.815
0.810
0.805
0.800
0.795
0.790
0.785
0.780
Accuracy
10
15
20
25
30
35
40
45
FLOPS(G)
ResNet50_vd
ResNet101_vd
ResNet152_vd
ResNet200_vd
Res2Net50_vd_26w_4s
Res2Net101_vd_26w_4s
Res2Net200_vd_26w_4s
ResNeXt50_vd_32x4d
ResNeXt101_vd_32x4d
ResNeXt152_vd_32x4d
ResNeXt50_vd_64x4d
ResNeXt101_vd_64x4d
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
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



目前 PaddleClas 开源的这三类的预训练模型一共有 24 个，其指标如图所示，从图中可以看出，在同样 Flops 和 Params 下，改进版的模型往往有更高的精度，但是推理速度往往不如 ResNet 系列。另一方面，Res2Net 表现也较为优秀，相比 ResNeXt 中的 group 操作、SEResNet 中的 SE 结构操作，Res2Net 在相同 Flops、Params 和推理速度下往往精度更佳。

2.5.2 精度、FLOPS 和参数量

2.5.3 基于 V100 GPU 的预测速度

2.5.4 基于 T4 GPU 的预测速度

2.6 Inception 系列

2.6.1 概述

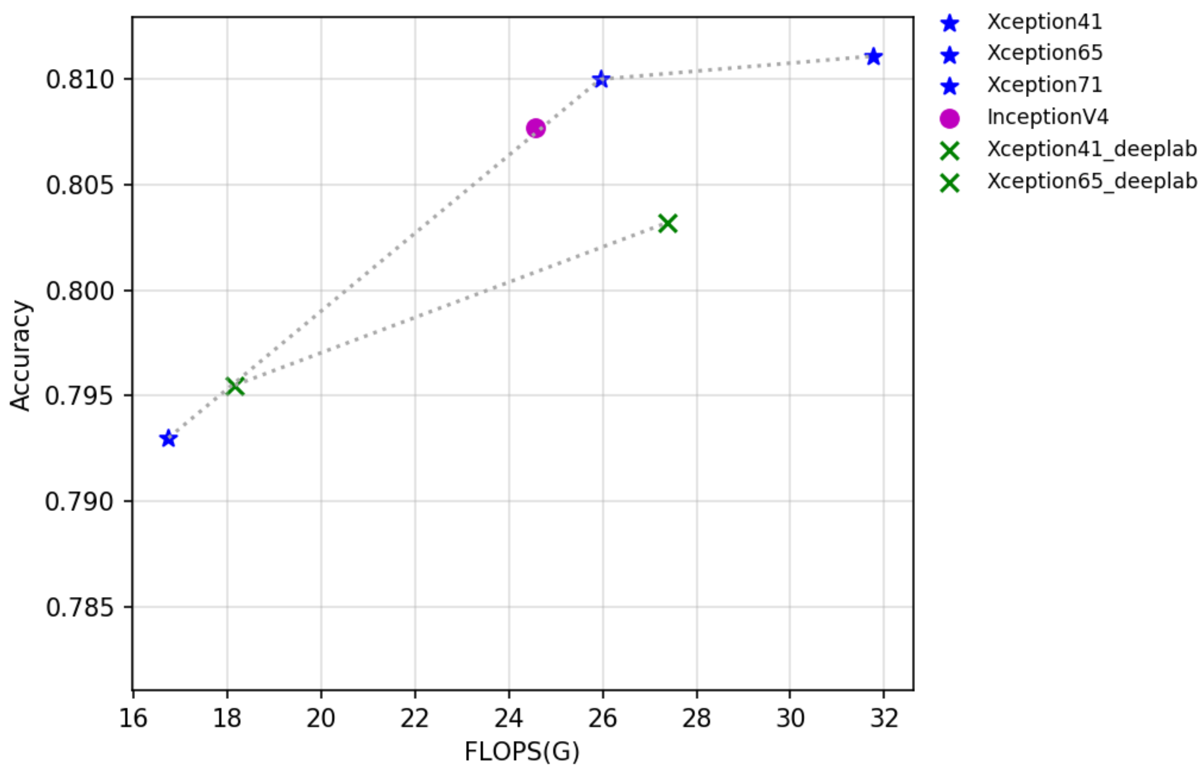
GoogLeNet 是 2014 年由 Google 设计的一种新的神经网络结构，其与 VGG 网络并列成为当年 ImageNet 挑战赛的双雄。GoogLeNet 首次引入 Inception 结构，在网络中堆叠该结构使得网络层数达到了 22 层，这也是卷积网络首次超过 20 层的标志。由于在 Inception 结构中使用了 1x1 的卷积用于通道数降维，并且使用

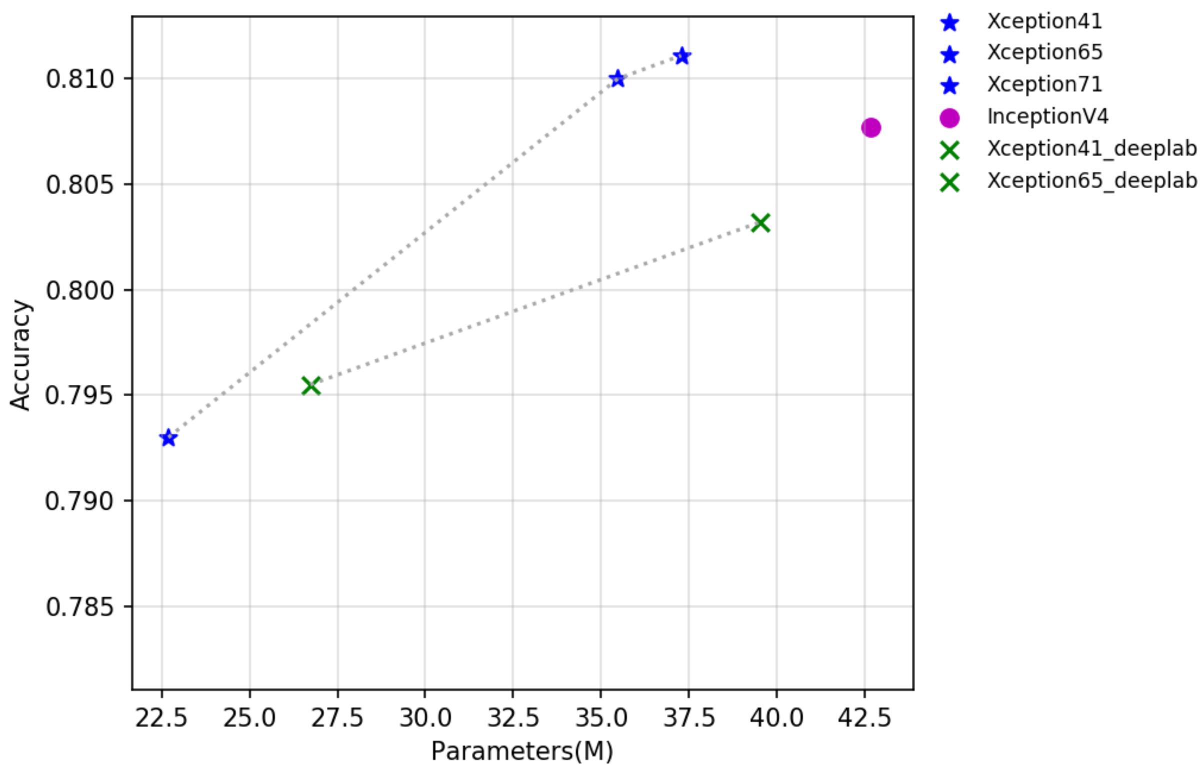
了 Global-pooling 代替传统的多 fc 层加工特征的方式，最终的 GoogLeNet 网络的 FLOPS 和参数量远小于 VGG 网络，成为当时神经网络设计的一道亮丽风景线。

Xception 是 Google 继 Inception 后提出的对 InceptionV3 的另一种改进。在 Xception 中，作者使用了深度可分离卷积代替了传统的卷积操作，该操作大大节省了网络的 FLOPS 和参数量，但是精度反而有所提升。在 DeeplabV3+ 中，作者将 Xception 做了进一步的改进，同时增加了 Xception 的层数，设计出了 Xception65 和 Xception71 的网络。

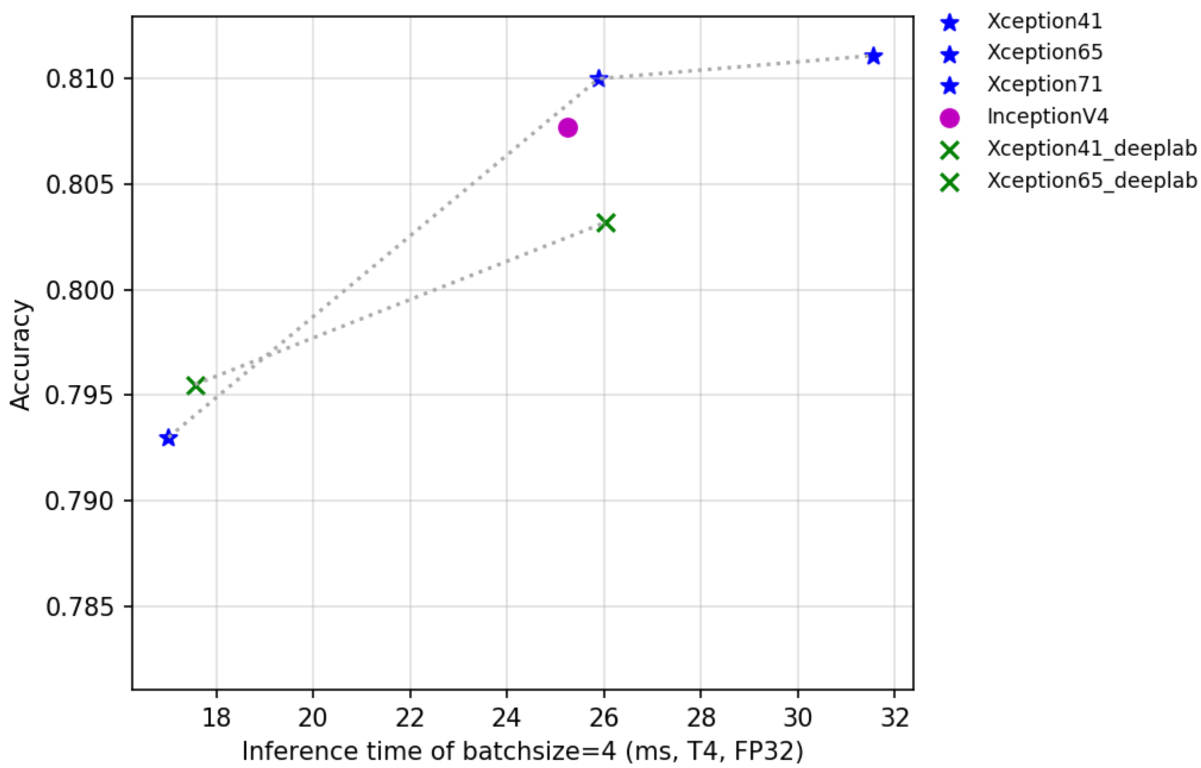
InceptionV4 是 2016 年由 Google 设计的新的神经网络，当时残差结构风靡一时，但是作者认为仅使用 Inception 结构也可以达到很高的性能。InceptionV4 使用了更多的 Inception module，在 ImageNet 上的精度再创新高。

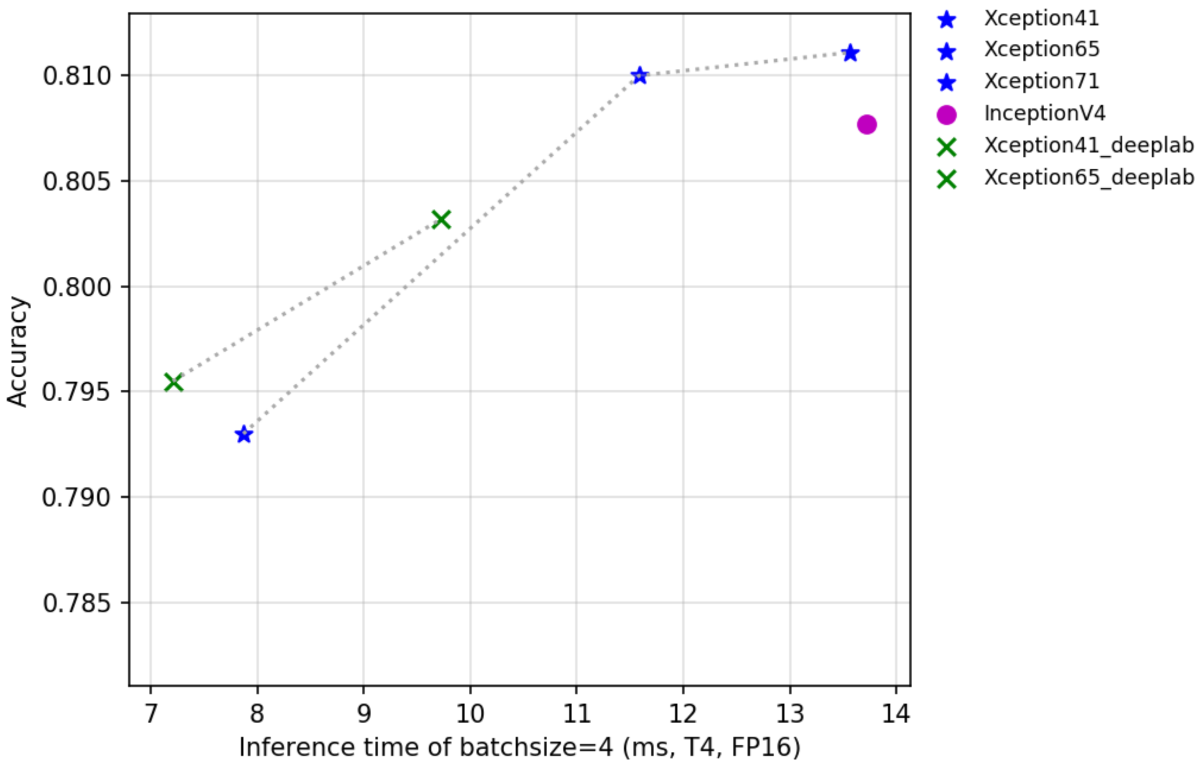
该系列模型的 FLOPS、参数量以及 T4 GPU 上的预测耗时如下图所示。





Xception41
Xception65
Xception71
InceptionV4
Xception41_deeplab
Xception65_deeplab
0.810
0.805
0.800
0.795
0.790
0.785
Accuracy
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)



上图反映了 Xception 系列和 InceptionV4 的精度和其他指标的关系。其中 Xception_deeplab 与论文结构保持一致，Xception 是 PaddleClas 的改进模型，在预测速度基本不变的情况下，精度提升约 0.6%。关于该改进模型的详细介绍正在持续更新中，敬请期待。

2.6.2 精度、FLOPS 和参数量

2.6.3 基于 V100 GPU 的预测速度

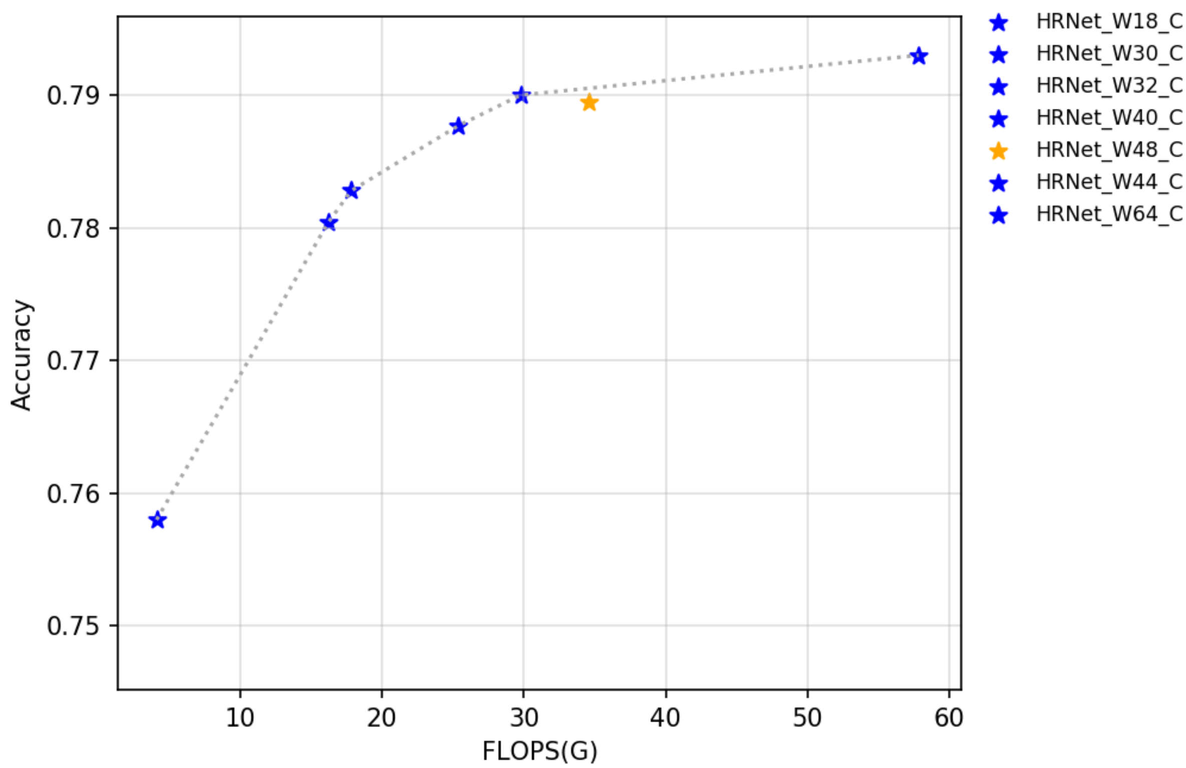
2.6.4 基于 T4 GPU 的预测速度

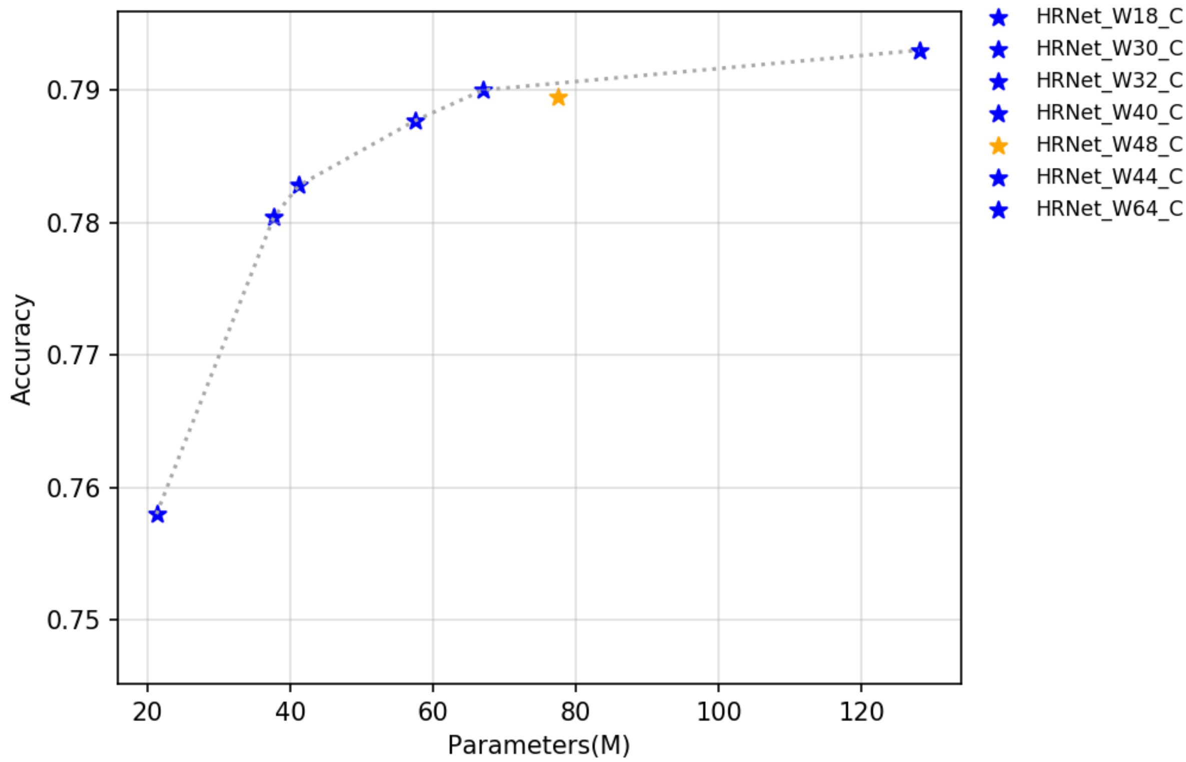
2.7 HRNet 系列

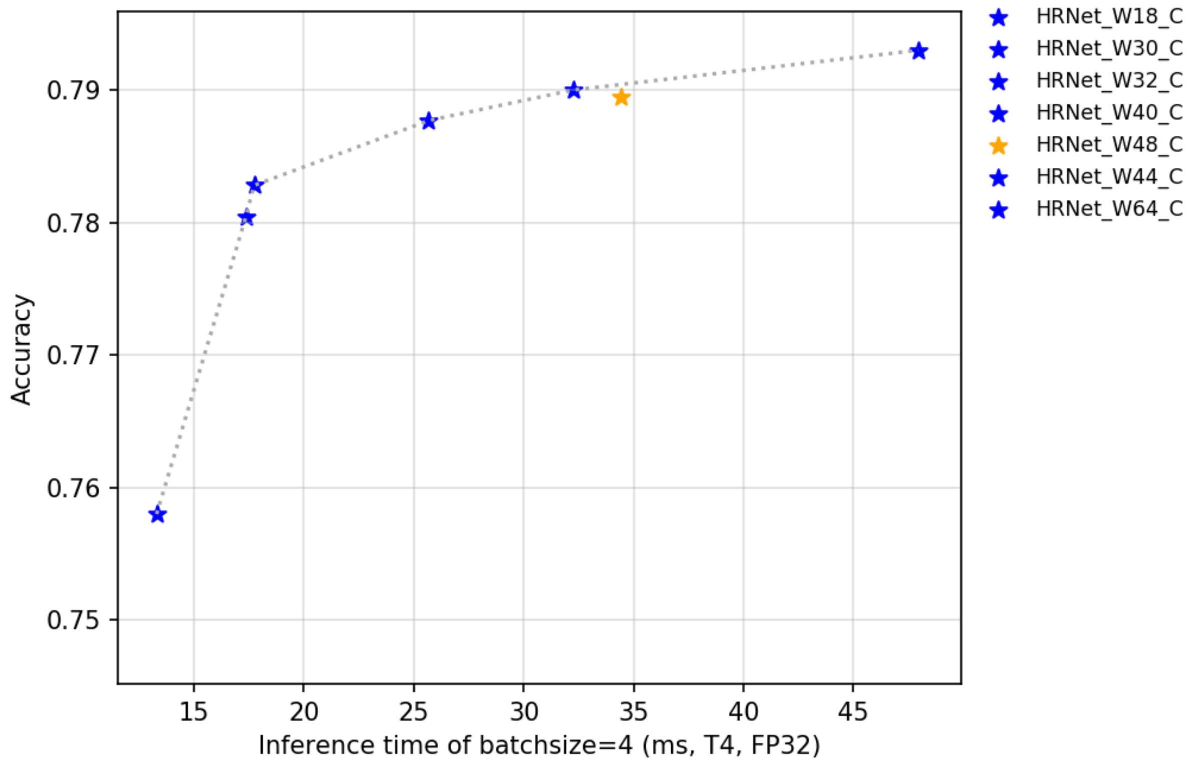
2.7.1 概述

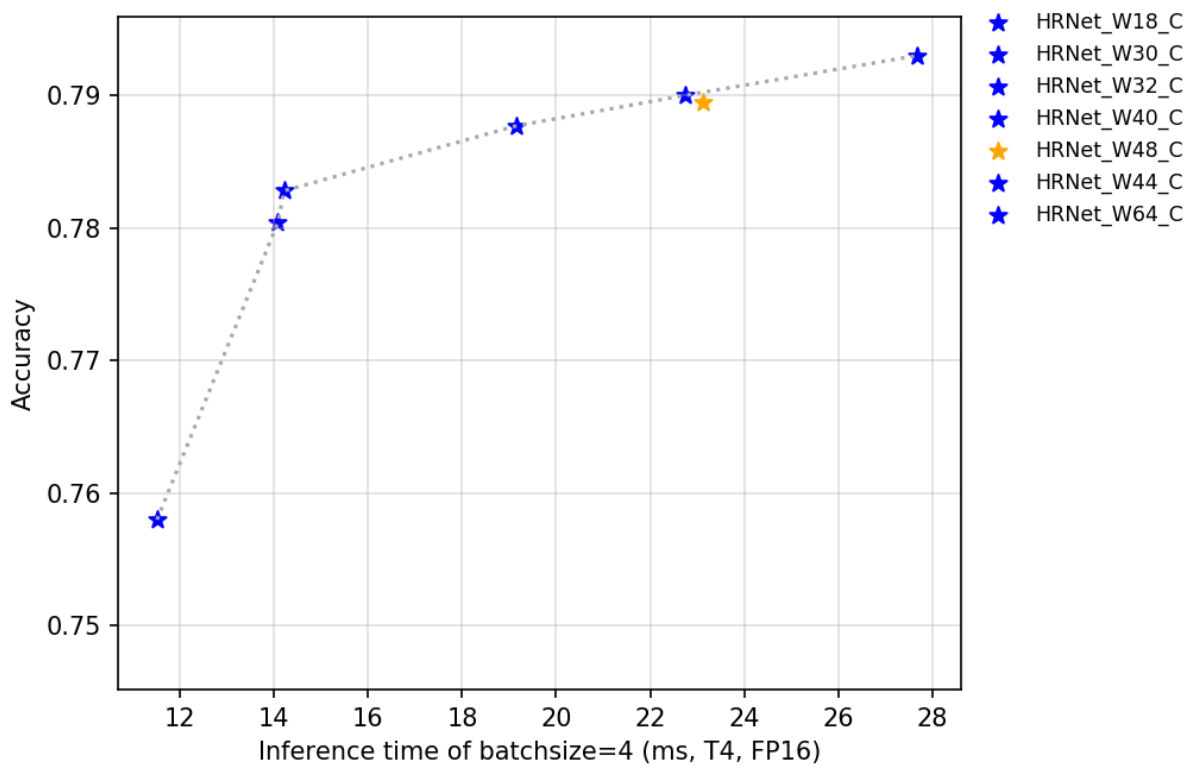
HRNet 是 2019 年由微软亚洲研究院提出的一种全新的神经网络，不同于以往的卷积神经网络，该网络在网络深层仍然可以保持高分辨率，因此预测的关键点热图更准确，在空间上也更精确。此外，该网络在对分辨率敏感的其他视觉任务中，如检测、分割等，表现尤为优异。

该系列模型的 FLOPS、参数量以及 T4 GPU 上的预测耗时如下图所示。


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


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)


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



目前 PaddleClas 开源的这类模型的预训练模型一共有 7 个，其指标如图所示，其中 HRNet_W48_C 指标精度异常的原因可能是因为网络训练的正常波动。

2.7.2 精度、FLOPS 和参数量

2.7.3 基于 V100 GPU 的预测速度

2.7.4 基于 T4 GPU 的预测速度

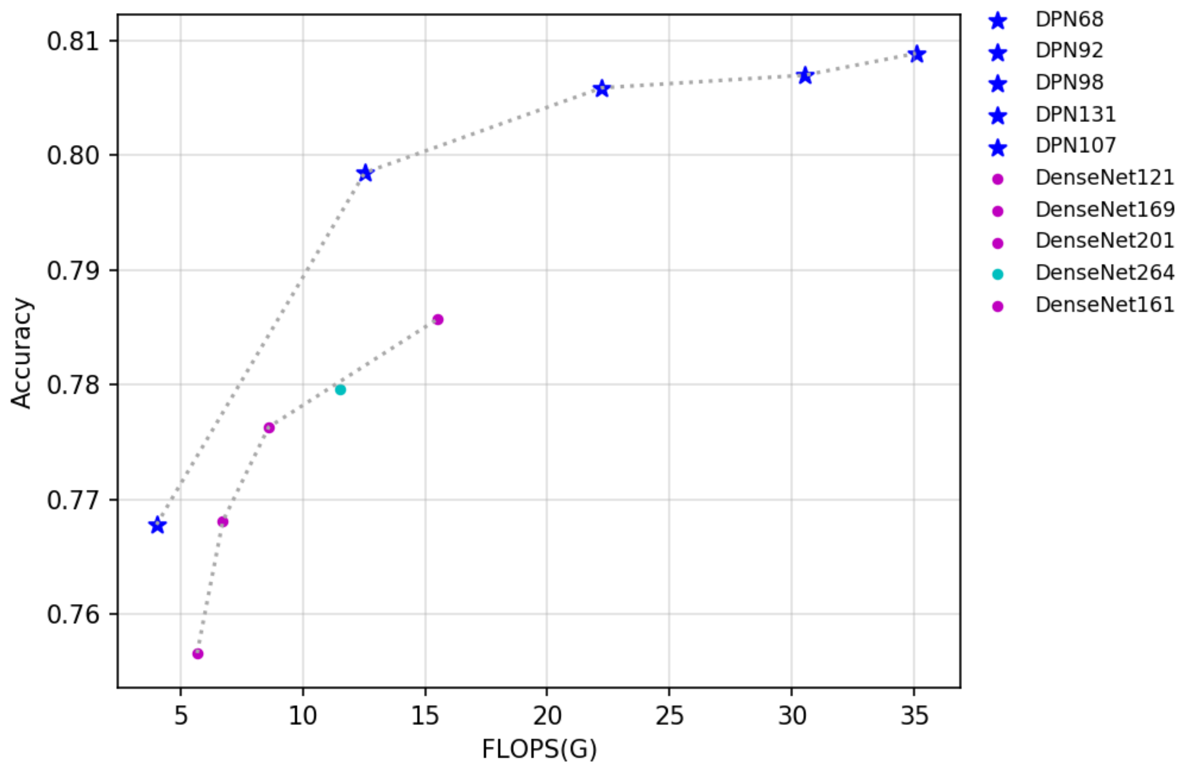
2.8 DPN 与 DenseNet 系列

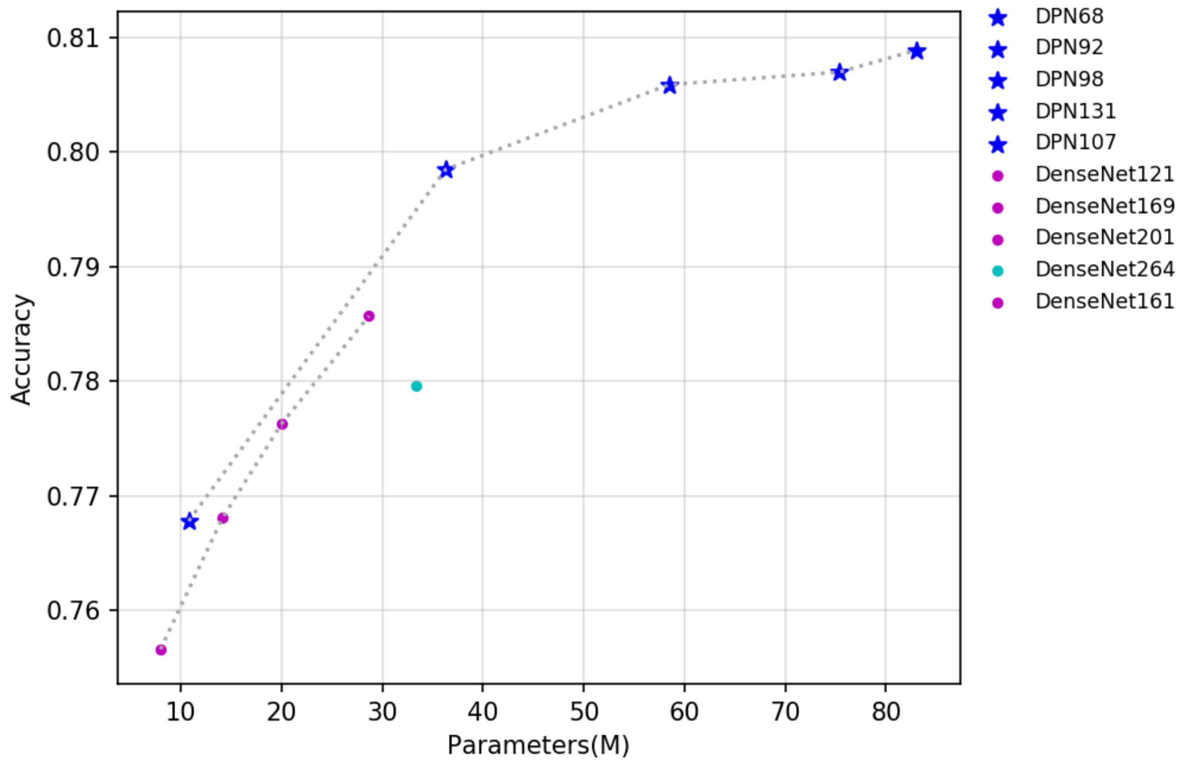
2.8.1 概述

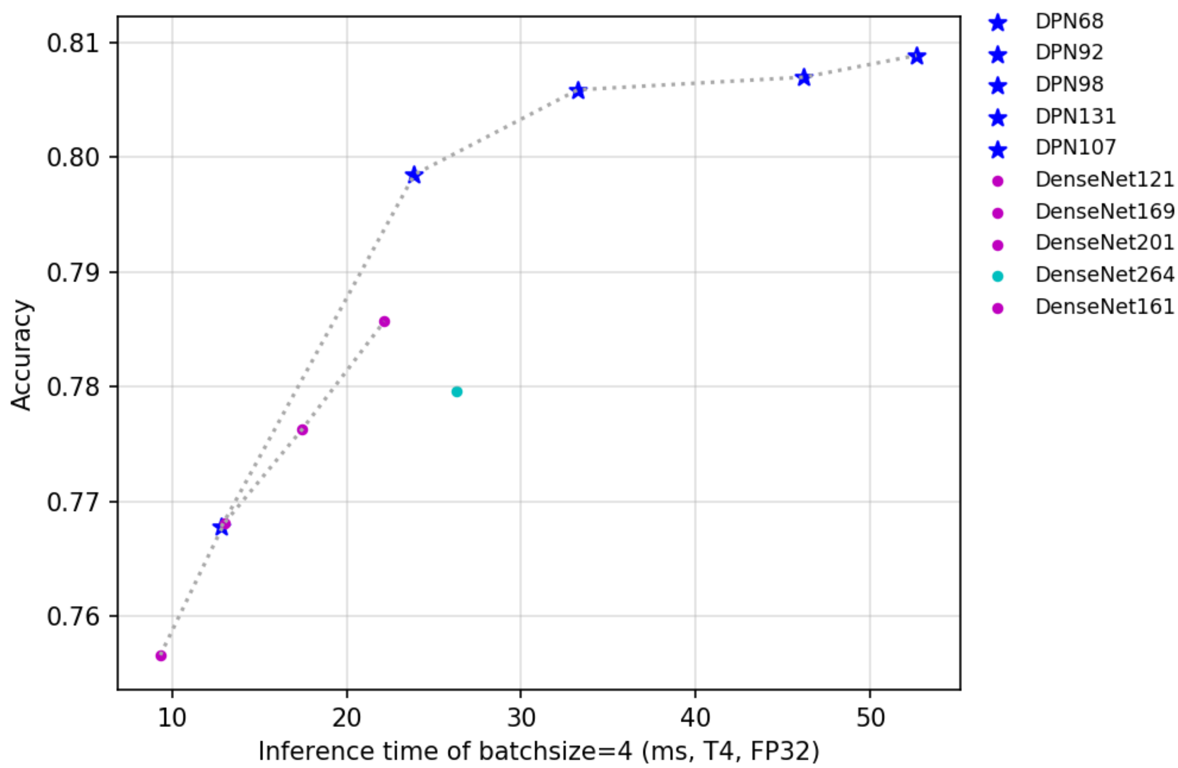
DenseNet 是 2017 年 CVPR best paper 提出的一种新的网络结构，该网络设计了一种新的跨层连接的 block，即 dense-block。相比 ResNet 中的 bottleneck，dense-block 设计了一个更激进的密集连接机制，即互相连接所有的层，每个层都会接受其前面所有层作为其额外的输入。DenseNet 将所有的 dense-block 堆叠，组合成了一个密集连接型网络。密集的连接方式使得 DenseNe 更容易进行梯度的反向传播，使得网络更容易训练。DPN 的全称是 Dual Path Networks，即双通道网络。该网络是由 DenseNet 和 ResNeXt 结合的一个网络，其证明了 DenseNet 能从靠前的层级中提取到新的特征，而 ResNeXt 本质上是对之前层级中已提取特征的

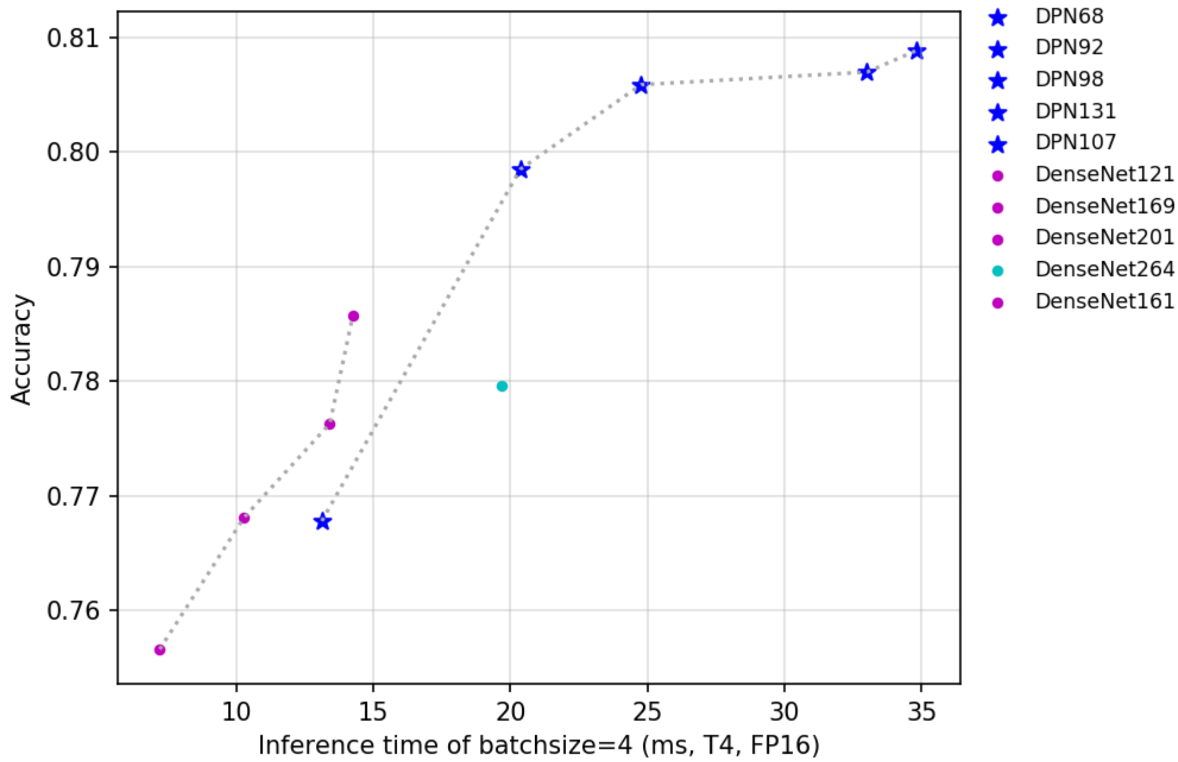
复用。作者进一步分析发现，ResNeXt 对特征有高复用率，但冗余度低，DenseNet 能创造新特征，但冗余度高。结合二者结构的优势，作者设计了 DPN 网络。最终 DPN 网络在同样 FLOPS 和参数量下，取得了比 ResNeXt 与 DenseNet 更好的结果。

该系列模型的 FLOPS、参数量以及 T4 GPU 上的预测耗时如下图所示。




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



目前 PaddleClas 开源的这两类模型的预训练模型一共有 10 个，其指标如上图所示，可以看到，在相同的 FLOPS 和参数量下，相比 DenseNet，DPN 拥有更高的精度。但是由于 DPN 有更多的分支，所以其推理速度要慢于 DenseNet。由于 DenseNet264 的网络层数最深，所以该网络是 DenseNet 系列模型中参数量最大的网络，DenseNet161 的网络的宽度最大，导致其是该系列中网络中计算量最大、精度最高的网络。从推理速度来看，计算量大且精度高的的 DenseNet161 比 DenseNet264 具有更快的速度，所以其比 DenseNet264 具有更大的优势。

对于 DPN 系列网络，模型的 FLOPS 和参数量越大，模型的精度越高。其中，由于 DPN107 的网络宽度最大，所以其是该系列网络中参数量与计算量最大的网络。

2.8.2 精度、FLOPS 和参数量

2.8.3 基于 V100 GPU 的预测速度

2.8.4 基于 T4 GPU 的预测速度

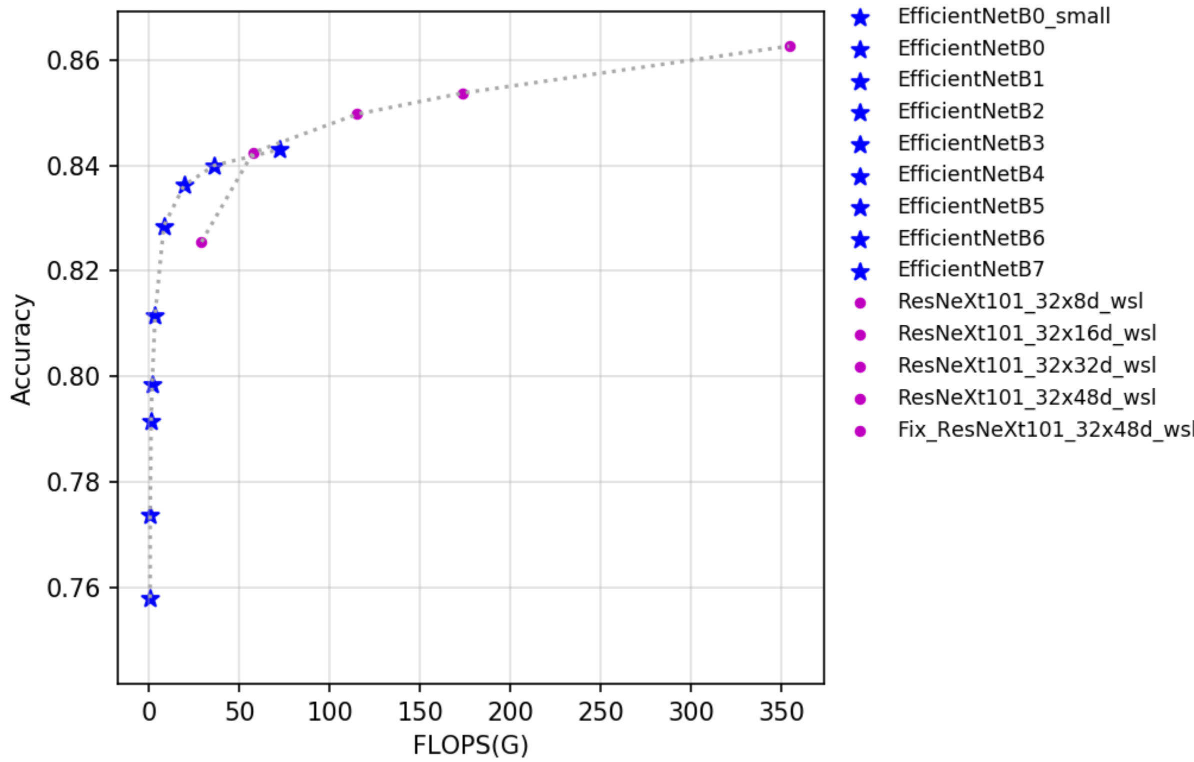
2.9 EfficientNet 与 ResNeXt101_wsl 系列

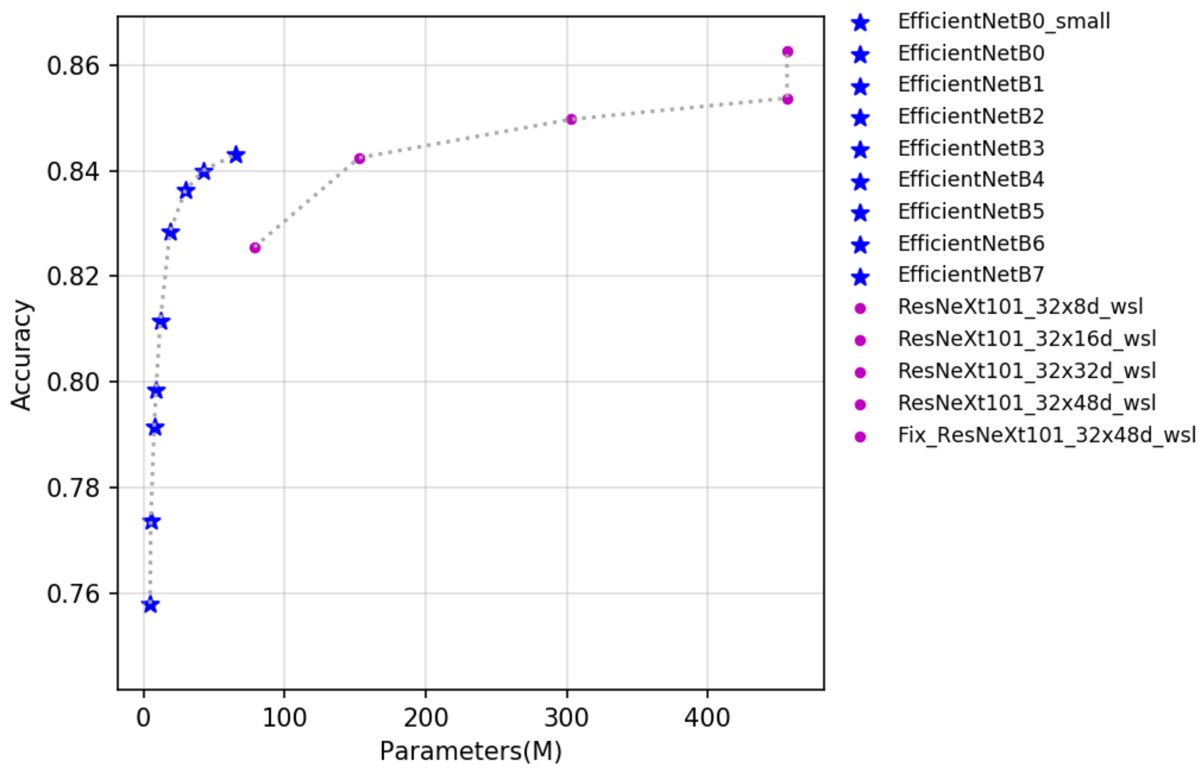
2.9.1 概述

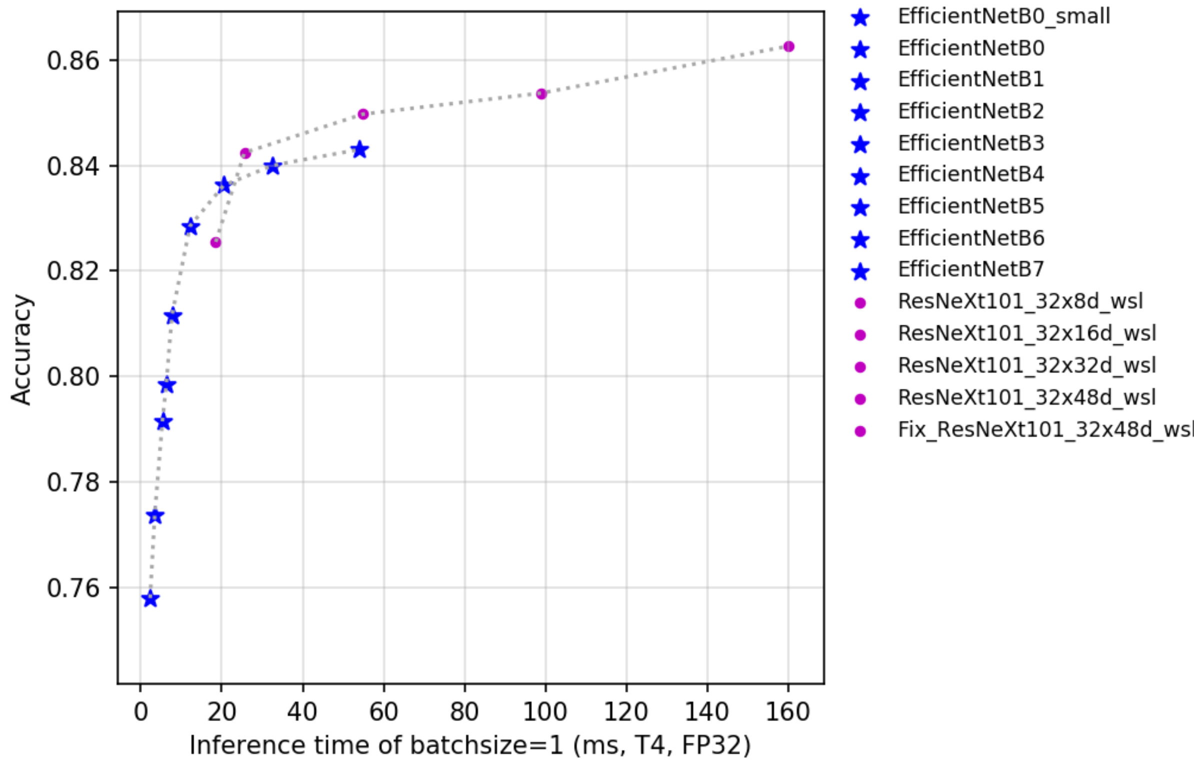
EfficientNet 是 Google 于 2019 年发布的一个基于 NAS 的轻量级网络，其中 EfficientNetB7 刷新了当时 ImageNet-1k 的分类准确率。在该文章中，作者指出，传统的提升神经网络性能的方法主要是从网络的宽度、网络的深度、以及输入图片的分辨率入手，但是作者通过实验发现，平衡这三个维度对精度和效率的提升至关重要，于是，作者通过一系列的实验中总结出了如何同时平衡这三个维度的放缩，与此同时，基于这种放缩方法，作者在 EfficientNet_B0 的基础上，构建了 EfficientNet 系列中 B1-B7 共 7 个网络，并在同样 FLOPS 与参数量的情况下，精度达到了 state-of-the-art 的效果。

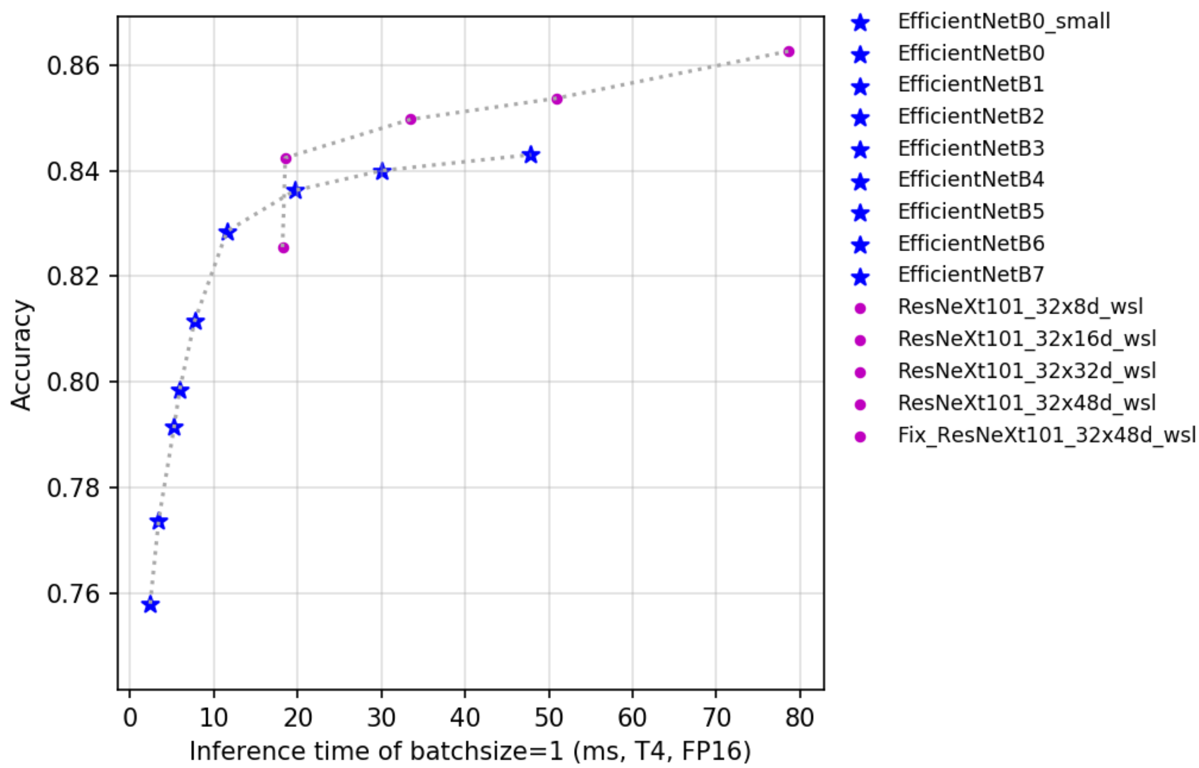
ResNeXt 是 facebook 于 2016 年提出的一种对 ResNet 的改进版网络。在 2019 年，facebook 通过弱监督学习研究了该系列网络在 ImageNet 上的精度上限，为了区别之前的 ResNeXt 网络，该系列网络的后缀为 wsl，其中 wsl 是弱监督学习（weakly-supervised-learning）的简称。为了能有更强的特征提取能力，研究者将其网络宽度进一步放大，其中最大的 ResNeXt101_32x48d_wsl 拥有 8 亿个参数，将其在 9.4 亿的弱标签图片下训练并在 ImageNet-1k 上做 finetune，最终在 ImageNet-1k 的 top-1 达到了 85.4%，这也是迄今为止在 ImageNet-1k 的数据集上以 224x224 的分辨率下精度最高的网络。Fix-ResNeXt 中，作者使用了更大的图像分辨率，针对训练图片和验证图片数据预处理不一致的情况下做了专门的 Fix 策略，并使得 ResNeXt101_32x48d_wsl 拥有了更高的精度，由于其用到了 Fix 策略，故命名为 Fix-ResNeXt101_32x48d_wsl。

该系列模型的 FLOPS、参数量以及 T4 GPU 上的预测耗时如下图所示。


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


Accuracy
Inference time of batchsize=1 (ms, T4, FP32)
0.76
0.78
0.80
0.82
0.84
0.86
0
20
40
60
80
100
120
140
160
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



目前 PaddleClas 开源的这两类模型的预训练模型一共有 14 个。从上图中可以看出 EfficientNet 系列网络优势非常明显，ResNeXt101_wsl 系列模型由于用到了更多的数据，最终的精度也更高。EfficientNet_B0_Small 是去掉了 SE_block 的 EfficientNet_B0，其具有更快的推理速度。

2.9.2 精度、FLOPS 和参数量

2.9.3 基于 V100 GPU 的预测速度

2.9.4 基于 T4 GPU 的预测速度

2.10 其他模型

2.10.1 概述

2012 年，Alex 等人提出的 AlexNet 网络在 ImageNet 大赛上以远超第二名的成绩夺冠，卷积神经网络乃至深度学习引起了广泛的关注。AlexNet 使用 relu 作为 CNN 的激活函数，解决了 sigmoid 在网络较深时的梯度弥散问题。训练时使用 Dropout 随机丢掉一部分神经元，避免了模型过拟合。网络中使用重叠的最大池化代替了此前 CNN 中普遍使用的平均池化，避免了平均池化的模糊效果，提升了特征的丰富性。从某种意义上说，AlexNet 引爆了神经网络的研究与应用热潮。

SqueezeNet 在 ImageNet-1k 上实现了与 AlexNet 相同的精度，但只用了 1/50 的参数量。该网络的核心是 Fire 模块，Fire 模块通过使用 1x1 的卷积实现通道降维，从而大大节省了参数量。作者通过大量堆叠 Fire 模块组成了 SqueezeNet。

VGG 由牛津大学计算机视觉组和 DeepMind 公司研究员一起研发的卷积神经网络。该网络探索了卷积神经网络的深度和其性能之间的关系，通过反复的堆叠 3x3 的小型卷积核和 2x2 的最大池化层，成功的构建了多层卷积神经网络并取得了不错的收敛精度。最终，VGG 获得了 ILSVRC 2014 比赛分类项目的亚军和定位项目的冠军。

DarkNet53 是 YOLO 作者在论文设计的用于目标检测的 backbone，该网络基本由 1x1 与 3x3 卷积构成，共 53 层，取名为 DarkNet53。

2.10.2 精度、FLOPS 和参数量

2.10.3 基于 V100 GPU 的预测速度

2.10.4 基于 T4 GPU 的预测速度

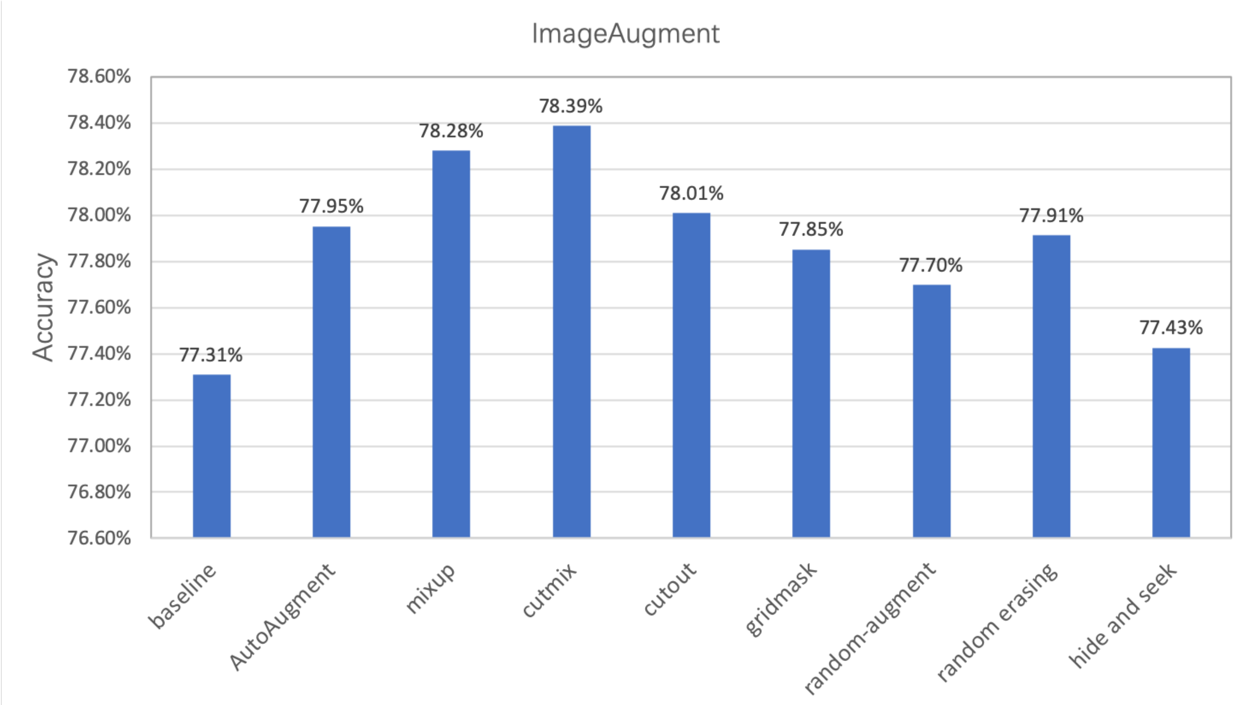
CHAPTER 3

高阶使用

3.1 图像增广

3.1.1 一、数据增广

在图像分类任务中，图像数据的增广是一种常用的正则化方法，常用于数据量不足或者模型参数较多的场景。在本章节中，我们将对除 ImageNet 分类任务标准数据增广外的 8 种数据增广方式进行简单的介绍和对比，用户也可以将这些增广方法应用到自己的任务中，以获得模型精度的提升。这 8 种数据增广方式在 ImageNet 上的精度指标如下所示。



3.1.2 二、常用数据增广方法

如果没有特殊说明，本章节中所有示例为 ImageNet 分类，并且假设最终输入网络的数据维度为：`[batch-size, 3, 224, 224]`

其中 ImageNet 分类训练阶段的标准数据增广方式分为以下几个步骤：

1. 图像解码：简写为 `ImageDecode`
2. 随机裁剪到长宽均为 224 的图像：简写为 `RandCrop`
3. 水平方向随机翻转：简写为 `RandFlip`
4. 图像数据的归一化：简写为 `Normalize`
5. 图像数据的重排，`[224, 224, 3]` 变为 `[3, 224, 224]`：简写为 `Transpose`
6. 多幅图像数据组成 batch 数据，如 `batch-size` 个 `[3, 224, 224]` 的图像数据拼组成 `[batch-size, 3, 224, 224]`：简写为 `Batch`

相比于上述标准的图像增广方法，研究者也提出了很多改进的图像增广策略，这些策略均是在标准增广方法的不同阶段插入一定的操作，基于这些策略操作所处的不同阶段，我们将其分为了三类：

1. 对 `RandCrop` 后的 224 的图像进行一些变换：AutoAugment，RandAugment
2. 对 `Transpose` 后的 224 的图像进行一些裁剪：CutOut，RandErasing，HideAndSeek，GridMask
3. 对 `Batch` 后的数据进行混合：Mixup，Cutmix

具体如下表所示：

PaddleClas 中集成了上述所有的数据增广策略，每种数据增广策略的参考论文与参考开源代码均在下面的介绍中列出。下文将介绍这些策略的原理与使用方法，并以下图为例，对变换后的效果进行可视化。为了说明问题，本章节中将 `RandCrop` 替换为 `Resize`。

3.1.3 三、图像变换类

图像变换类指的是对 `RandCrop` 后的 224 的图像进行一些变换，主要包括

- AutoAugment
- RandAugment

3.1 AutoAugment

论文地址：https://arxiv.org/abs/1805.09501v1

开源代码 github 地址：https://github.com/DeepVoltaire/AutoAugment

不同于常规的人工设计图像增广方式，AutoAugment 是在一系列图像增广子策略的搜索空间中通过搜索算法找到的适合特定数据集的图像增广方案。针对 ImageNet 数据集，最终搜索出来的数据增广方案包含 25 个子策略组合，每个子策略中都包含两种变换，针对每幅图像都随机的挑选一个子策略组合，然后以一定的概率来决定是否执行子策略中的每种变换。

PaddleClas 中 `AutoAugment` 的使用方法如下所示。

```
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
```

(下页继续)

(续上页)

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

ops = [decode_op, resize_op, autoaugment_op]

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

结果如下图所示。

3.2 RandAugment

论文地址：https://arxiv.org/pdf/1909.13719.pdf

开源代码 github 地址：https://github.com/heartInsert/randaugment

`AutoAugment` 的搜索方法比较暴力，直接在数据集上搜索针对该数据集的最优策略，其计算量很大。在 `RandAugment` 文章中作者发现，一方面，针对越大的模型，越大的数据集，使用 `AutoAugment` 方式搜索到的增广方式产生的收益也就越小；另一方面，这种搜索出的最优策略是针对该数据集的，其迁移能力较差，并不太适合迁移到其他数据集上。

在 `RandAugment` 中，作者提出了一种随机增广的方式，不再像 `AutoAugment` 中那样使用特定的概率确定是否使用某种子策略，而是所有的子策略都会以同样的概率被选择到，论文中的实验也表明这种数据增广方式即使在大模型的训练中也具有很好的效果。

PaddleClas 中 `RandAugment` 的使用方法如下所示。

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

结果如下图所示。

3.1.4 四、图像裁剪类

图像裁剪类主要是对 `Transpose` 后的 224 的图像进行一些裁剪，并将裁剪区域的像素值置为特定的常数（默认为 0），主要包括：

- CutOut
- RandErasing

- HideAndSeek
- GridMask

图像裁剪的这些增广并非一定要放在归一化之后，也有不少实现是放在归一化之前的，也就是直接对 uint8 的图像进行操作，两种方式的差别是：如果直接对 uint8 的图像进行操作，那么再经过归一化之后被裁剪的区域将不再是纯黑或纯白（减均值除方差之后像素值不为 0）。而对归一后之后的数据进行操作，裁剪的区域会是纯黑或纯白。

上述的裁剪变换思路是相同的，都是为了解决训练出的模型在有遮挡数据上泛化能力较差的问题，不同的是他们的裁剪方式、区域不太一样。

4.1 Cutout

论文地址：https://arxiv.org/abs/1708.04552

开源代码 github 地址：https://github.com/uoguelph-mlrg/Cutout

Cutout 可以理解为 Dropout 的一种扩展操作，不同的是 Dropout 是对图像经过网络后生成的特征进行遮挡，而 Cutout 是直接对输入的图像进行遮挡，相对于 Dropout 对噪声的鲁棒性更好。作者在论文中也进行了说明，这样做法有以下两点优势：(1) 通过 Cutout 可以模拟真实场景中主体被部分遮挡时的分类场景；(2) 可以促进模型充分利用图像中更多的内容来进行分类，防止网络只关注显著性的图像区域，从而发生过拟合。

PaddleClas 中 `Cutout` 的使用方法如下所示。

```
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

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

结果如下图所示。

4.2 RandomErasing

论文地址：https://arxiv.org/pdf/1708.04896.pdf

开源代码 github 地址：https://github.com/zhunzhong07/Random-Erasing

`RandomErasing` 与 `Cutout` 方法类似，同样是为了解决训练出的模型在有遮挡数据上泛化能力较差的问题，作者在论文中也指出，随机裁剪的方式与随机水平翻转具有一定的互补性。作者也在行人再识别（REID）上验证了该方法的有效性。与 `Cutout` 不同的是，在 `RandomErasing` 中，图片以一定的概率接受该种预处理方法，生成掩码的尺寸大小与长宽比也是根据预设的超参数随机生成。

PaddleClas 中 `RandomErasing` 的使用方法如下所示。

```
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))
randomerasing_op = RandomErasing()

ops = [decode_op, resize_op, tochw_op, randomerasing_op]

imgs_dir = 图像路径
fnames = os.listdir(imgs_dir)
for f in fnames:
```

（下页继续）

(续上页)

```
    data = open(os.path.join(imgs_dir, f)).read()
    img = transform(data, ops)
    img = img.transpose((1, 2, 0))
```

结果如下图所示。

4.3 HideAndSeek

论文地址：https://arxiv.org/pdf/1811.02545.pdf

开源代码 github 地址：https://github.com/kkanshul/Hide-and-Seek

`HideAndSeek` 论文将图像分为若干块区域 (patch)，对于每块区域，都以一定的概率生成掩码，不同区域的掩码含义如下图所示。

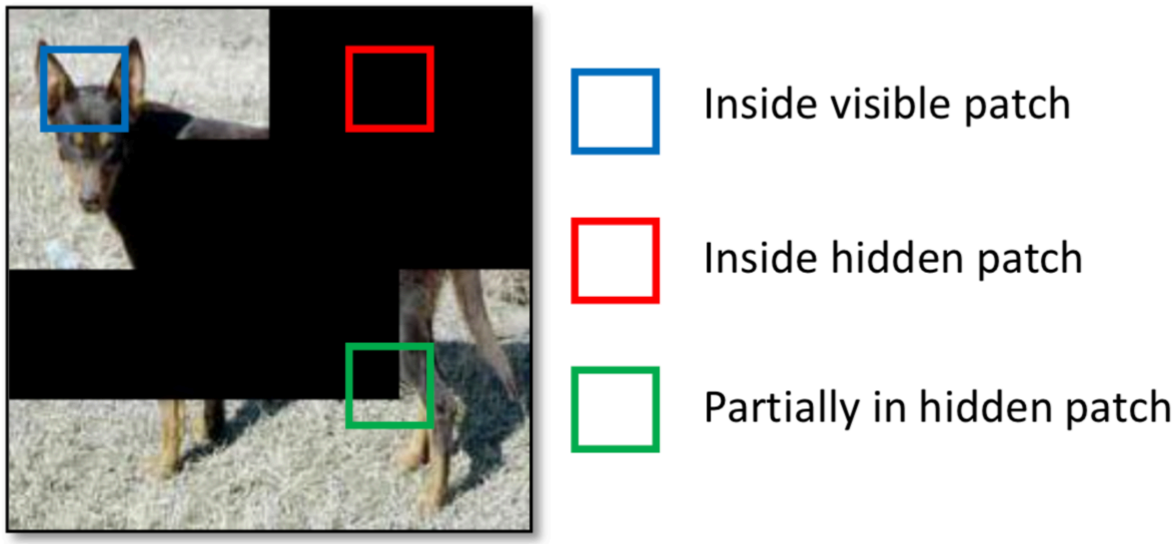


Fig. 3. There are three types of convolutional filter activations after hiding patches: a convolution filter can be completely within a visible region (blue box), completely within a hidden region (red box), or partially within a visible/hidden region (green box).

PaddleClas 中 HideAndSeek 的使用方法如下所示。

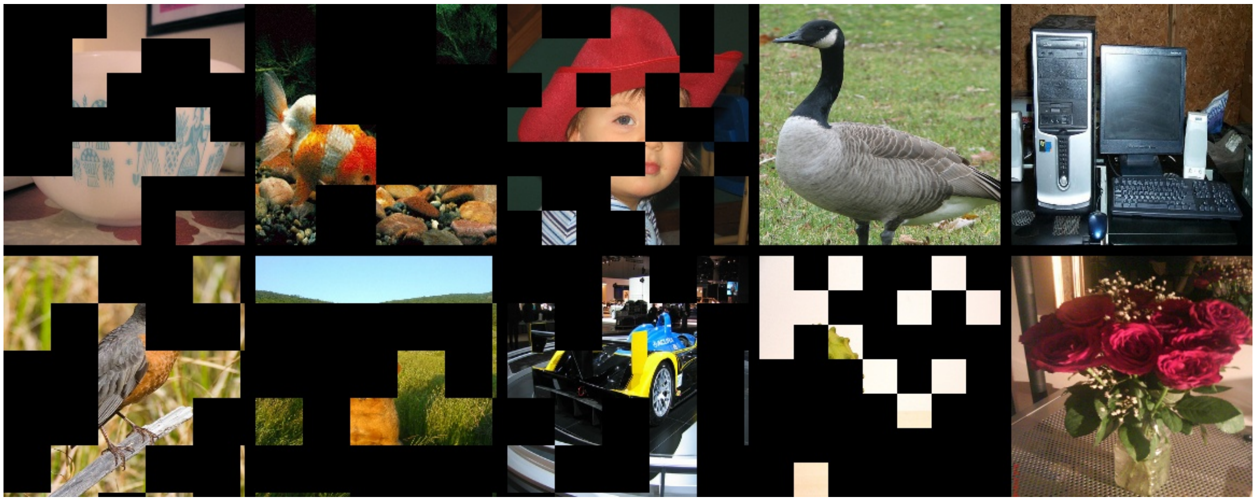
```
from ppcls.data.imaug import DecodeImage
from ppcls.data.imaug import ResizeImage
from ppcls.data.imaug import ToCHWImage
from ppcls.data.imaug import HideAndSeek
from ppcls.data.imaug import transform

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

(下页继续)

(续上页)

```
    img = img.transpose((1, 2, 0))
```

结果如下图所示。



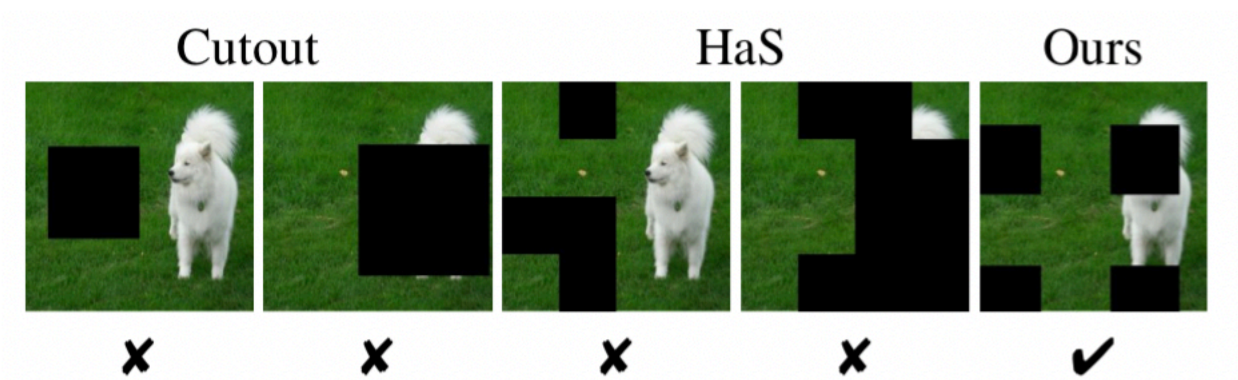
4.4 GridMask

论文地址：https://arxiv.org/abs/2001.04086

开源代码 github 地址：https://github.com/akuxcw/GridMask

作者在论文中指出，此前存在的基于对图像 crop 的方法存在两个问题，如下图所示：

1. 过度删除区域可能造成目标主体大部分甚至全部被删除，或者导致上下文信息的丢失，导致增广后的数据成为噪声数据；
2. 保留过多的区域，对目标主体及上下文基本产生不了什么影响，失去增广的意义。



因此如果避免过度删除或过度保留成为需要解决的核心问题。

`GridMask` 是通过生成一个与原图分辨率相同的掩码，并将掩码进行随机翻转，与原图相乘，从而得到增广后的图像，通过超参数控制生成的掩码网格的大小。

在训练过程中，有两种以下使用方法：

1. 设置一个概率 p，从训练开始就对图片以概率 p 使用 `GridMask` 进行增广。
2. 一开始设置增广概率为 0，随着迭代轮数增加，对训练图片进行 `GridMask` 增广的概率逐渐增大，最后变为 p。

论文中验证上述第二种方法的训练效果更好一些。

PaddleClas 中 `GridMask` 的使用方法如下所示。

```
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 = 图像路径
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))
```

结果如下图所示。

3.1.5 五、图像混叠

图像混叠主要对 Batch 后的数据进行混合，包括：

- Mixup
- Cutmix

前文所述的图像变换与图像裁剪都是针对单幅图像进行的操作，而图像混叠是对两幅图像进行融合，生成一幅图像，两种方法的主要区别为混叠的方式不太一样。

5.1 Mixup

论文地址：https://arxiv.org/pdf/1710.09412.pdf

开源代码 github 地址：https://github.com/facebookresearch/mixup-cifar10

Mixup 是最先提出的图像混叠增广方案，其原理简单、方便实现，不仅在图像分类上，在目标检测上也取得了不错的效果。为了便于实现，通常只对一个 batch 内的数据进行混叠，在 Cutmix 中也是如此。

如下是 imaug 中的实现，需要指出的是，下述实现会出现对同一幅进行相加的情况，也就是最终得到的图和原图一样，随着 batch-size 的增加这种情况出现的概率也会逐渐减小。

PaddleClas 中 Mixup 的使用方法如下所示。

```
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
```

(下页继续)

(续上页)

```
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 = 图像路径

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)
```

结果如下图所示。

5.2 Cutmix

论文地址：https://arxiv.org/pdf/1905.04899v2.pdf

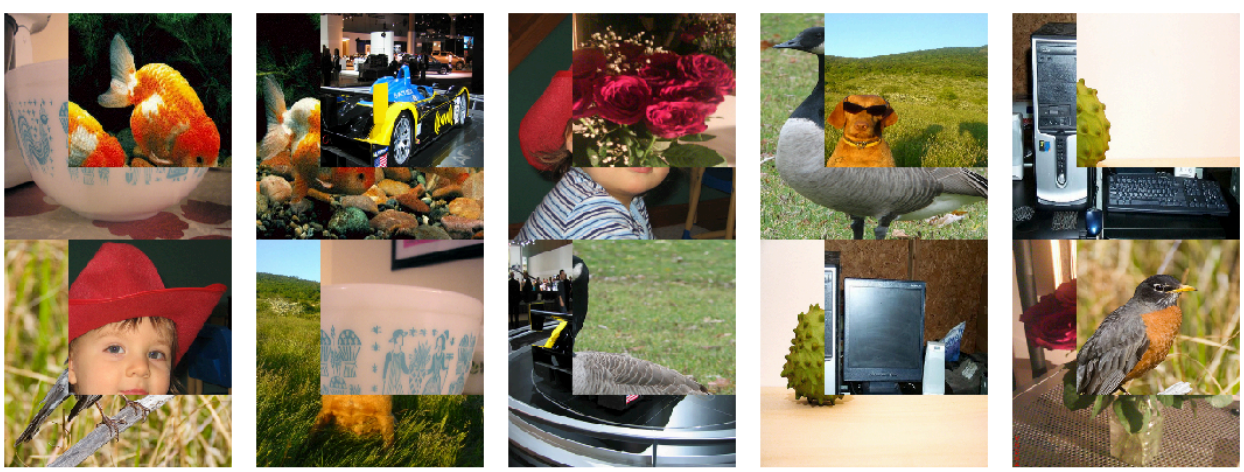
开源代码 github 地址：https://github.com/clovaai/CutMix-PyTorch

与 Mixup 直接对两幅图进行相加不一样，Cutmix 是从一幅图中随机裁剪出一个 ROI，然后覆盖当前图像中

对应的区域，代码实现如下所示：

```
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 = 图像路径

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)
```

结果如下图所示。



3.1.6 六、实验

基于 PaddleClas，在 ImageNet1k 数据集上的分类精度如下。

注意：

- 在这里的实验中，为了便于对比，我们将 l2 decay 固定设置为 1e-4，在实际使用中，我们推荐尝试使用更小的 l2 decay。结合数据增广，我们发现将 l2 decay 由 1e-4 减小为 7e-5 均能带来至少 0.3~0.5% 的精度提升。
- 我们目前尚未对不同策略进行组合并验证效果，这一块后续我们会开展更多的对比实验，敬请期待。

3.1.7 七、数据增广分类实战

本节将基于 ImageNet-1K 的数据集详细介绍数据增广实验，如果想快速体验此方法，可以参考*30 分钟玩转 PaddleClas*中基于 Flowers102 的数据增广实验。

7.1 参数配置

由于不同的数据增广方式含有不同的超参数，为了便于理解和使用，我们在 `configs/DataAugment` 里分别列举了 8 种训练 ResNet50 的数据增广方式的参数配置文件，用户可以在 `tools/run.sh` 里直接替换配置文件的路径即可使用。此处分别挑选了图像变换、图像裁剪、图像混叠中的一个示例展示，其他参数配置用户可以自查配置文件。

RandAugment

`RandAugment` 的图像增广方式的配置如下，其中用户需要指定其中的参数 `num_layers` 与 `magnitude`，默认的数值分别是 `2` 和 `5`。`RandAugment` 是在 uint8 的数据格式上转换的，所以其处理过程应该放在归一化操作（`NormalizeImage`）之前。

```yaml
    transforms:
        - DecodeImage:
            to_rgb: True
            to_np: False
            channel_first: False
        - RandCropImage:
            size: 224
        - RandFlipImage:
            flip_code: 1
        - RandAugment:
            num_layers: 2
            magnitude: 5
```

（下页继续）

(续上页)

```
        - NormalizeImage:
            scale: 1./255.
            mean: [0.485, 0.456, 0.406]
            std: [0.229, 0.224, 0.225]
            order: ''
        - ToCHWImage:
```

Cutout

Cutout 的图像增广方式的配置如下，其中用户需要指定其中的参数 n_holes 与 length，默认的数值分别是 1 和 112。类似其他图像裁剪类的数据增广方式，Cutout 既可以在 uint8 格式的数据上操作，也可以在归一化（NormalizeImage）后的数据上操作，此处给出的是在归一化后的操作。

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

Mixup

Mixup 的图像增广方式的配置如下，其中用户需要指定其中的参数 alpha，默认的数值是 0.2。类似其他图像混合类的数据增广方式，Mixup 是在图像做完数据处理后将每个 batch 内的数据做图像混叠，将混叠后的图像和标签输入网络中训练，所以其是在图像数据处理（图像变换、图像裁剪）后操作。另外，在配置文件中，需要将 use_mix 参数设置为 True。

```yaml
    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
```

7.2 启动命令

当用户配置完训练环境后，类似于训练其他分类任务，只需要将 tools/run.sh 中的配置文件替换成为相应的数据增广方式的配置文件即可。

其中 run.sh 中的内容如下：

```bash
export PYTHONPATH=path_to_PaddleClas:$PYTHONPATH

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

运行 run.sh：

```bash
sh tools/run.sh
```

7.3 注意事项

- 在使用图像混叠类的数据处理时，需要将配置文件中的 `use_mix` 设置为 `True`，另外由于图像混叠时需对 label 进行混叠，无法计算训练数据的准确率，所以在训练过程中没有打印训练准确率。
- 在使用数据增广后，由于训练数据更难，所以训练损失函数可能较大，训练集的准确率相对较低，但其有拥更好的泛化能力，所以验证集的准确率相对较高。
- 在使用数据增广后，模型可能会趋于欠拟合状态，建议可以适当的调小 `l2_decay` 的值来获得更高的验证集准确率。
- 几乎每一类图像增广均含有超参数，我们只提供了基于 ImageNet-1k 的超参数，其他数据集需要用户自己调试超参数，具体超参数的含义用户可以阅读相关的论文，调试方法也可以参考训练技巧的章节。

  如果您觉得此文档对您有帮助，欢迎 star 我们的项目：https://github.com/PaddlePaddle/PaddleClas

3.1.8 参考文献

[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.

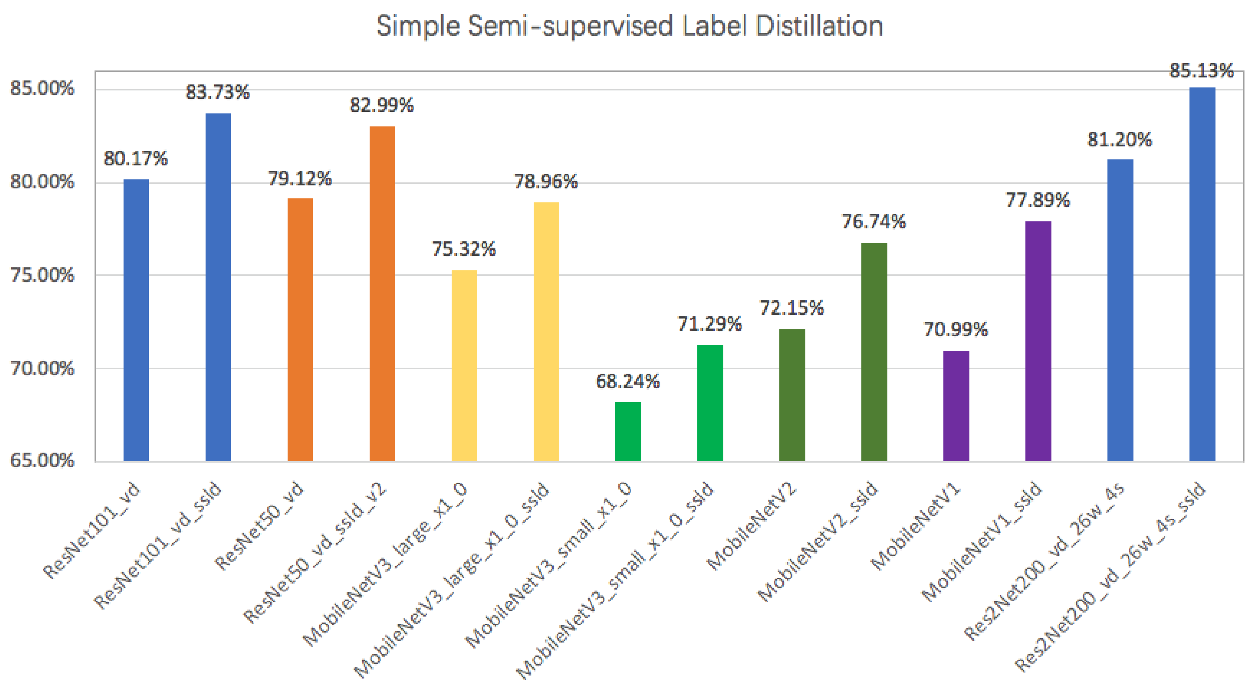
3.2 知识蒸馏

3.2.1 一、模型压缩方法简介

近年来，深度神经网络在计算机视觉、自然语言处理等领域被验证是一种极其有效的解决问题的方法。通过构建合适的神经网络，加以训练，最终网络模型的性能指标基本上都会超过传统算法。

在数据量足够大的情况下，通过合理构建网络模型的方式增加其参数量，可以显著改善模型性能，但是这又带来了模型复杂度急剧提升的问题。大模型在实际场景中使用的成本较高。

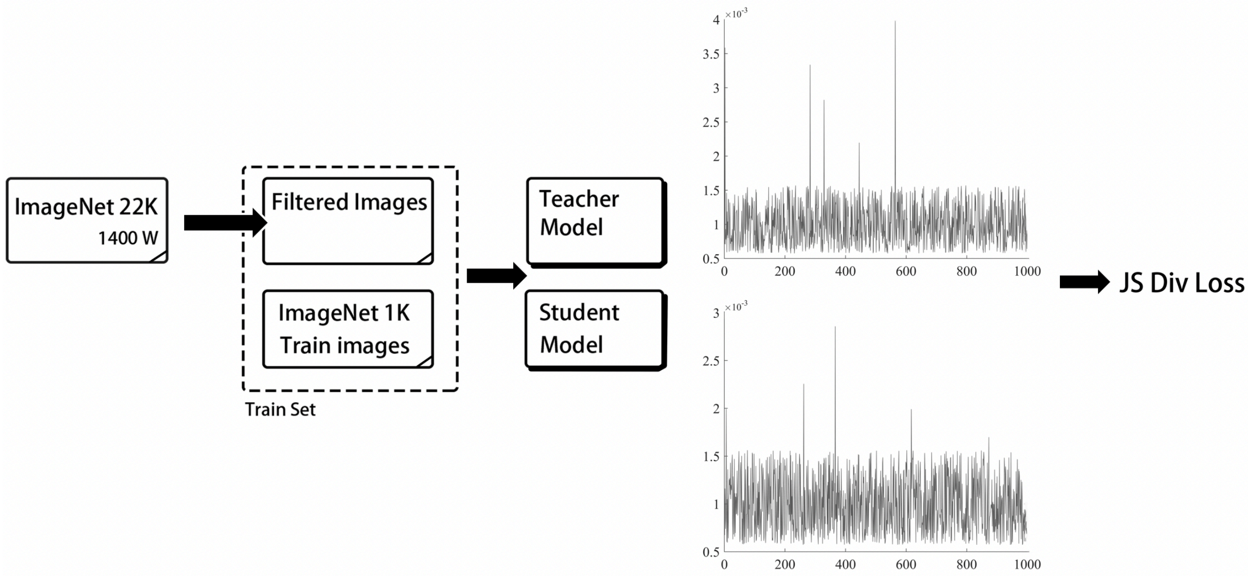
深度神经网络一般有较多的参数冗余，目前有几种主要的方法对模型进行压缩，减小其参数量。如裁剪、量化、知识蒸馏等，其中知识蒸馏是指使用教师模型 (teacher model) 去指导学生模型 (student model) 学习特定任务，保证小模型在参数量不变的情况下，得到比较大的性能提升，甚至获得与大模型相似的精度指标 [1]。PaddleClas 融合已有的蒸馏方法 [2,3]，提供了一种简单的半监督标签知识蒸馏方案（SSLD，Simple Semi-supervised Label Distillation），基于 ImageNet1k 分类数据集，在 ResNet_vd 以及 MobileNet 系列上的精度均有超过 3% 的绝对精度提升，具体指标如下图所示。



3.2.2 二、SSLD 蒸馏策略

2.1 简介

SSLD 的流程图如下图所示。



首先，我们从 ImageNet22k 中挖掘出了近 400 万张图片，同时与 ImageNet-1k 训练集整合在一起，得到了一个新的包含 500 万张图片的数据集。然后，我们将学生模型与教师模型组合成一个新的网络，该网络分别输出学生模型和教师模型的预测分布，与此同时，固定教师模型整个网络的梯度，而学生模型可以做正常的反向传播。最后，我们将两个模型的 logits 经过 softmax 激活函数转换为 soft label，并将二者的 soft label 做 JS 散度作为损失函数，用于蒸馏模型训练。下面以 MobileNetV3（该模型直接训练，精度为 75.3%）的知识蒸馏为例，介绍该方案的核心关键点（baseline 为 79.12% 的 ResNet50_vd 模型蒸馏 MobileNetV3，训练集为 ImageNet1k 训练集，loss 为 cross entropy loss，迭代轮数为 120epoch，精度指标为 75.6%）。

- 教师模型的选择。在进行知识蒸馏时，如果教师模型与学生模型的结构差异太大，蒸馏得到的结果反而不会有太大收益。相同结构下，精度更高的教师模型对结果也有很大影响。相比于 79.12% 的 ResNet50_vd 教师模型，使用 82.4% 的 ResNet50_vd 教师模型可以带来 0.4% 的绝对精度收益 (`75.6%->76.0%`)。
- 改进 loss 计算方法。分类 loss 计算最常用的方法就是 cross entropy loss，我们经过实验发现，在使用 soft label 进行训练时，相对于 cross entropy loss，KL div loss 对模型性能提升几乎无帮助，但是使用具有对称特性的 JS div loss 时，在多个蒸馏任务上相比 cross entropy loss 均有 0.2% 左右的收益 (`76.0%->76.2%`)，SSLD 中也基于 JS div loss 展开实验。
- 更多的迭代轮数。蒸馏的 baseline 实验只迭代了 120 个 epoch。实验发现，迭代轮数越多，蒸馏效果越好，最终我们迭代了 360epoch，精度指标可以达到 77.1%(`76.2%->77.1%`)。
- 无需数据集的真值标签，很容易扩展训练集。SSLD 的 loss 在计算过程中，仅涉及到教师和学生模型对于相同图片的处理结果（经过 softmax 激活函数处理之后的 soft label），因此即使图片数据不包含真值标签，也可以用来进行训练并提升模型性能。该蒸馏方案的无标签蒸馏策略也大大提升了学生模型的性能上限（`77.1%->78.5%`）。
- ImageNet1k 蒸馏 finetune。我们仅使用 ImageNet1k 数据，使用蒸馏方法对上述模型进行 finetune，最终仍然可以获得 0.4% 的性能提升 (`78.5%->78.9%`)。

2.2 数据选择

- SSLD 蒸馏方案的一大特色就是无需使用图像的真值标签，因此可以任意扩展数据集的大小，考虑到计算资源的限制，我们在这里仅基于 ImageNet22k 数据集对蒸馏任务的训练集进行扩充。在 SSLD 蒸馏任务中，我们使用了 `Top-k per class` 的数据采样方案 [3]。具体步骤如下。
  - 训练集去重。我们首先基于 SIFT 特征相似度匹配的方式对 ImageNet22k 数据集与 ImageNet1k 验证集进行去重，防止添加的 ImageNet22k 训练集中包含 ImageNet1k 验证集图像，最终去除了 4511 张相似图片。部分过滤的相似图片如下所示。

  - 大数据集 soft label 获取，对于去重后的 ImageNet22k 数据集，我们使用 `ResNeXt101_32x16d_wsl` 模型进行预测，得到每张图片的 soft label。
  - Top-k 数据选择，ImageNet1k 数据共有 1000 类，对于每一类，找出属于该类并且得分最高的 k 张图片，最终得到一个数据量不超过 `1000*k` 的数据集（某些类上得到的图片数量可能少于 k 张）。
  - 将该数据集与 ImageNet1k 的训练集融合组成最终蒸馏模型所使用的数据集，数据量为 500 万。

3.2.3 三、实验

- PaddleClas 的蒸馏策略为大数据集训练 `+ImageNet1k 蒸馏 finetune` 的策略。选择合适的教师模型，首先在挑选得到的 500 万数据集上进行训练，然后在 ImageNet1k 训练集上进行 finetune，最终得到蒸馏后的学生模型。

3.1 教师模型的选择

为了验证教师模型和学生模型的模型大小差异和教师模型的模型精度对蒸馏结果的影响，我们做了几组实验验证。训练策略统一为：`cosine_decay_warmup`，`lr=1.3`，`epoch=120`，`bs=2048`，学生模型均为从头训练。

从表中可以看出

教师模型结构相同时，其精度越高，最终的蒸馏效果也会更好一些。

教师模型与学生模型的模型大小差异不宜过大，否则反而会影响蒸馏结果的精度。

因此最终在蒸馏实验中，对于 ResNet 系列学生模型，我们使用 `ResNeXt101_32x16d_wsl` 作为教师模型；对于 MobileNet 系列学生模型，我们使用蒸馏得到的 `ResNet50_vd` 作为教师模型。

3.2 大数据蒸馏

基于 PaddleClas 的蒸馏策略为大数据集训练 `+imagenet1k finetune` 的策略。

针对从 ImageNet22k 挑选出的 400 万数据，融合 imagenet1k 训练集，组成共 500 万的训练集进行训练，具体地，在不同模型上的训练超参及效果如下。

3.3 ImageNet1k 训练集 finetune

对于在大数据集上训练的模型，其学习到的特征可能与 ImageNet1k 数据特征有偏，因此在这里使用 ImageNet1k 数据集对模型进行 finetune。finetune 的超参和 finetune 的精度收益如下。

3.4 数据增广以及基于 Fix 策略的微调

- 基于前文所述的实验结论，我们在训练的过程中加入自动增广 (AutoAugment)[4]，同时进一步减小了 l2_decay(4e-5->2e-5)，最终 ResNet50_vd 经过 SSLD 蒸馏策略，在 ImageNet1k 上的精度可以达到 82.99%，相比之前不加数据增广的蒸馏策略再次增加了 0.6%。
- 对于图像分类任务，在测试的时候，测试尺度为训练尺度的 1.15 倍左右时，往往在不需要重新训练模型的情况下，模型的精度指标就可以进一步提升 [5]，对于 82.99% 的 ResNet50_vd 在 320x320 的尺度下测试，精度可达 83.7%，我们进一步使用 Fix 策略，即在 320x320 的尺度下进行训练，使用与预测时相同的数据预处理方法，同时固定除 FC 层以外的所有参数，最终在 320x320 的预测尺度下，精度可以达到 **84.0%**。

3.4 实验过程中的一些问题

3.4.1 bn 的计算方法

- 在预测过程中，batch norm 的平均值与方差是通过加载预训练模型得到（设其模式为 test mode）。在训练过程中，batch norm 是通过统计当前 batch 的信息（设其模式为 train mode），与历史保存信息进行滑动平均计算得到，在蒸馏任务中，我们发现通过 train mode，即教师模型的 bn 实时变化的模

式，去指导学生模型，比通过 test mode 蒸馏，得到的学生模型性能更好一些，下面是一组实验结果。因此我们在该蒸馏方案中，均使用 train mode 去得到教师模型的 soft label。

3.4.2 模型名字冲突问题的解决办法

- 在蒸馏过程中，如果遇到命名冲突的问题，如使用 ResNet50_vd 蒸馏 ResNet34_vd，此时直接训练，会提示相同 variable 名称不匹配的问题，此时可以通过给学生模型或者教师模型中的变量名添加名称的方式解决该问题，如下所示。在训练之后也可以直接根据后缀区分学生模型和教师模型各自包含的参数。

```
def net(self, input, class_dim=1000):
    student = ResNet34_vd(postfix_name="_student")
    out_student = student.net( input, class_dim=class_dim )

    teacher = ResNet50_vd()
    out_teacher = teacher.net( input, class_dim=class_dim )
    out_teacher.stop_gradient = True

    return  out_teacher, out_student
```

- 训练完成后，可以通过批量重名的方式修改学生模型的参数，以上述代码为例，批量重命名的命令行如下。

```
cd model_final # enter model dir
for var in ./*_student; do cp "$var" "../student_model/${var%_student}"; done # batch␣
↪copy and rename
```

3.2.4 四、蒸馏模型的应用

4.1 使用方法

- 中间层学习率调整。蒸馏得到的模型的中间层特征图更加精细化，因此将蒸馏模型预训练应用到其他任务中时，如果采取和之前相同的学习率，容易破坏中间层特征。而如果降低整体模型训练的学习率，则会带来训练收敛速度慢的问题。因此我们使用了中间层学习率调整的策略。具体地：
  - 针对 ResNet50_vd，我们设置一个学习率倍数列表，res block 之前的 3 个 conv2d 卷积参数具有统一的学习率倍数，4 个 res block 的 conv2d 分别有一个学习率参数，共需设置 5 个学习率倍数的超参。在实验中发现。用于迁移学习 finetune 分类模型时，`[0.1,0.1,0.2,0.2,0.3]` 的中间层学习率倍数设置在绝大多数的任务中都性能更好；而在目标检测任务中，`[0.05,0.05,0.05,0.1,0.15]` 的中间层学习率倍数设置能够带来更大的精度收益。
  - 对于 MoblileNetV3_large_1x0，由于其包含 15 个 block，我们设置每 3 个 block 共享一个学习

率倍数参数，因此需要共 5 个学习率倍数的参数，最终发现在分类和检测任务中，`[0.25,0.25,0.5,0.5,0.75]` 的中间层学习率倍数能够带来更大的精度收益。

- 适当的 l2 decay。不同分类模型在训练的时候一般都会根据模型设置不同的 l2 decay，大模型为了防止过拟合，往往会设置更大的 l2 decay，如 ResNet50 等模型，一般设置为 `1e-4`；而如 MobileNet 系列模型，在训练时往往都会设置为 `1e-5~4e-5`，防止模型过度欠拟合，在蒸馏时亦是如此。在将蒸馏模型应用到目标检测任务中时，我们发现也需要调节 backbone 甚至特定任务模型模型的 l2 decay，和预训练蒸馏时的 l2 decay 尽可能保持一致。以 Faster RCNN MobiletNetV3 FPN 为例，我们发现仅修改该参数，在 COCO2017 数据集上就可以带来最多 0.5% 左右的精度 (mAP) 提升（默认 Faster RCNN l2 decay 为 1e-4，我们修改为 1e-5~4e-5 均有 0.3%~0.5% 的提升）。

4.2 迁移学习 finetune

- 为验证迁移学习的效果，我们在 10 个小的数据集上验证其效果。在这里为了保证实验的可对比性，我们均使用 ImageNet1k 数据集训练的标准预处理过程，对于蒸馏模型我们也添加了蒸馏模型中间层学习率的搜索。
- 对于 ResNet50_vd，baseline 为 Top1 Acc 79.12% 的预训练模型基于 grid search 搜索得到的最佳精度，对比实验则为基于该精度对预训练和中间层学习率进一步搜索得到的最佳精度。下面给出 10 个数据集上所有 baseline 和蒸馏模型的精度对比。
- 可以看出在上面 10 个数据集上，结合适当的中间层学习率倍数设置，蒸馏模型平均能够带来 1% 以上的精度提升。

4.3 目标检测

我们基于两阶段目标检测 Faster/Cascade RCNN 模型验证蒸馏得到的预训练模型的效果。

- ResNet50_vd

设置训练与评测的尺度均为 640x640，最终 COCO 上检测指标如下。

在这里可以看出，对于未蒸馏模型，过度调整中间层学习率反而降低最终检测模型的性能指标。基于该蒸馏模型，我们也提供了领先的服务端实用目标检测方案，详细的配置与训练代码均已开源，可以参考PaddleDetection。

3.2.5 五、SSLD 实战

本节将基于 ImageNet-1K 的数据集详细介绍 SSLD 蒸馏实验，如果想快速体验此方法，可以参考*30 分钟玩转 PaddleClas*中基于 Flowers102 的 SSLD 蒸馏实验。

5.1 参数配置

实战部分提供了 SSLD 蒸馏的示例，在 `ppcls/modeling/architectures/distillation_models.py` 中提供了 `ResNeXt101_32x16d_wsl` 蒸馏 `ResNet50_vd` 与 `ResNet50_vd_ssld` 蒸馏 `MobileNetV3_large_x1_0`

的示例，configs/Distillation 里分别提供了二者的配置文件，用户可以在 tools/run.sh 里直接替换配置文件的路径即可使用。

ResNeXt101_32x16d_wsl 蒸馏 ResNet50_vd

ResNeXt101_32x16d_wsl 蒸馏 ResNet50_vd 的配置如下，其中 pretrained model 指定了 ResNeXt101_32x16d_wsl（教师模型）的预训练模型的路径，该路径也可以同时指定教师模型与学生模型的预训练模型的路径，用户只需要同时传入二者预训练的路径即可（配置中的注释部分）。

```
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
```

ResNet50_vd_ssld 蒸馏 MobileNetV3_large_x1_0

类似于 ResNeXt101_32x16d_wsl 蒸馏 ResNet50_vd，ResNet50_vd_ssld 蒸馏 MobileNetV3_large_x1_0 的配置如下：

```
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
```

5.2 启动命令

当用户配置完训练环境后，类似于训练其他分类任务，只需要将 tools/run.sh 中的配置文件替换成为相应的蒸馏配置文件即可。

其中 run.sh 中的内容如下：

```
export PYTHONPATH=path_to_PaddleClas:$PYTHONPATH

python -m paddle.distributed.launch \
    --selected_gpus="0,1,2,3" \
```

（下页继续）

(续上页)

```
    --log_dir=R50_vd_distill_MV3_large_x1_0 \
    tools/train.py \
        -c ./configs/Distillation/R50_vd_distill_MV3_large_x1_0.yaml
```

运行 `run.sh`:

```
sh tools/run.sh
```

5.3 注意事项

- 用户在使用 SSLD 蒸馏之前，首先需要在目标数据集上训练一个教师模型，该教师模型用于指导学生模型在该数据集上的训练。
- 在用户使用 SSLD 蒸馏的时候需要将配置文件中的 `use_distillation` 设置为 `True`，另外由于学生模型学习带有知识信息的 soft-label，所以需要关掉 label_smoothing 选项，即将 `ls_epsilon` 中的值设置在 [0,1] 之外。
- 如果学生模型没有加载预训练模型，训练的其他超参数可以参考该学生模型在 ImageNet-1k 上训练的超参数，如果学生模型加载了预训练模型，学习率可以调整到原来的 1/10 或者 1/100。
- 在 SSLD 蒸馏的过程中，学生模型只学习 soft-label 导致训练目标变的更加复杂，建议可以适当的调小 `l2_decay` 的值来获得更高的验证集准确率。
- 若用户准备添加无标签的训练数据，只需要将新的训练数据放置在原本训练数据的路径下，生成新的数据 list 即可，另外，新生成的数据 list 需要将无标签的数据添加伪标签（只是为了统一读数据）。

  如果您觉得此文档对您有帮助，欢迎 star 我们的项目：https://github.com/PaddlePaddle/PaddleClas

3.2.6 参考文献

[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

应用拓展

4.1 图像分类迁移学习

迁移学习是机器学习领域的一个重要分支，广泛应用于文本、图像等各种领域，此处我们主要介绍的是图像分类领域的迁移学习，也就是我们常说的域迁移，比如将 ImageNet 分类模型迁移到我们自己场景的图像分类任务上，如花卉分类。

4.1.1 一、超参搜索

ImageNet 作为业界常用的图像分类数据被大家广泛使用，已经总结出一系列经验性的超参，使用这些超参往往能够得到不错的训练精度，而这些经验性的参数在迁移到自己的业务中时，有时效果不佳。有两种常用的超参搜索方法可以用于获得更好的模型超参。

1.1 网格搜索

网格搜索，即穷举搜索，通过查找搜索空间内所有的点，确定最优值。方法简单有效，但当搜索空间较大时，需要消耗大量的计算资源。

1.2 贝叶斯搜索

贝叶斯搜索，即贝叶斯优化，在搜索空间中随机选取超参数点，采用高斯过程，即根据上一个超参数点的结果，更新当前的先验信息，计算前面 n 个超参数点的后验概率分布，得到搜索空间中每一个超参数点的期望均值和方差，其中期望均值越大表示接近最优指标的可能性越大，方差越大表示不确定性越大。通常将选择

期望均值大的超参数点称为 exporitation，选择方差大的超参数点称为 exploration。在贝叶斯优化中通过定义 acquisition function 权衡期望均值和方差。贝叶斯搜索认为当前选择的超参数点是处于最大值可能出现的位置。

基于上述两种搜索方案，我们在 8 个开源数据集上将固定一组参数实验以及两种搜索方案做了对比实验，参照 [1] 的实验方案，我们对 4 个超参数进行搜索，搜索空间及实验结果如下所示：

- 固定参数：

```
初始学习率 lr=0.003, l2 decay=1e-4, label smoothing=False, mixup=False
```

- 超参搜索空间：

```
初始学习率 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]
```

网格搜索的搜索次数为 196 次，而贝叶斯搜索通过设置最大迭代次数（max_iter）和是否重复搜索（de_duplication）来确定搜索次数。我们设计了系列实验，baseline 为 ImageNet1k 校验集 Top1 Acc 为 79.12% 的 ResNet50_vd 预训练模型，并固定超参，在新数据集上 finetune 得到的模型。下表给出了固定参数、网格搜索以及贝叶斯搜索的精度与搜索次数对比。

- 精度与搜索次数对比：
- 上述实验验证了贝叶斯搜索相比网格搜索，在减少搜索次数 10 倍左右条件下，精度只下降 0%~0.4%。
- 当搜索空间进一步扩大时，例如将是否进行 AutoAugment，RandAugment，Cutout，Cutmix 以及 Dropout 这些正则化策略作为选择时，贝叶斯搜索能够在获取较优精度的前提下，有效地降低搜索次数。

4.1.2 二、大规模分类模型

在实际应用中，由于训练数据的匮乏，往往将 ImageNet1k 数据集训练的分类模型作为预训练模型，进行图像分类的迁移学习。为了进一步助力解决实际问题，基于 ResNet50_vd, 百度开源了自研的大规模分类预训练模型，其中训练数据为 10 万个类别，4300 万张图片。

我们在 6 个自有采集的数据集上进行迁移学习实验，采用一组固定参数以及网格搜索方式，其中训练轮数设置为 20epochs，选用 ResNet50_vd 模型，ImageNet 预训练精度为 79.12%。实验数据集参数以及模型精度的对比结果如下：

固定参数：

```
初始学习率 lr=0.001, l2 decay=1e-4, label smoothing=False, mixup=False
```

- 通过上述的实验验证了当使用一组固定参数时，相比于 ImageNet 预训练模型，使用大规模分类模型作为预训练模型在大多数情况下能够提升模型在新的数据集上得效果，通过参数搜索可以进一步提升精度。

4.1.3 参考文献

[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 通用目标检测

4.2.1 服务器端实用目标检测方案

简介

- 近年来,学术界和工业界广泛关注图像中目标检测任务。基于 SSLD 蒸馏方案训练得到的 ResNet50_vd 预训练模型 (ImageNet1k 验证集上 Top1 Acc 为 82.39%)，结合 PaddleDetection 中的丰富算子，飞桨提供了一种面向服务器端实用的目标检测方案 PSS-DET(Practical Server Side Detection)。基于 COCO2017 目标检测数据集，V100 单卡预测速度为为 61FPS 时，COCO mAP 可达 41.6%；预测速度为 20FPS 时，COCO mAP 可达 47.8%。

消融实验

- 我们以标准的 Faster RCNN ResNet50_vd FPN 为例，下表给出了 PSS-DET 不同的模块的速度与精度收益。

基于该实验结论，我们结合 Cascade RCNN，使用更大的训练与评估尺度 (1000x1500)，最终在单卡 V100 上速度为 20FPS，COCO mAP 达 47.8%。下图给出了目前类似速度的目标检测方法的速度与精度指标。

pssdet

注意

这里为了更方便地对比，我们将 V100 的预测耗时乘以 1.2 倍，近似转化为 Titan V 的预测耗时。

更加详细的代码、配置与预训练模型的地址可以参考PaddleDetection。

4.2.2 移动端实用目标检测方案

- 目前正在更新中，敬请期待！

CHAPTER 5

实用工具

5.1 分类预测框架

5.1.1 一、简介

Paddle 的模型保存有多种不同的形式，大体可分为两类：

1. persistable 模型（fluid.save_persistabels 保存的模型）一般做为模型的 checkpoint，可以加载后重新训练。persistable 模型保存的是零散的权重文件，每个文件代表模型中的一个 Variable，这些零散的文件不包含结构信息，需要结合模型的结构一起使用。

```
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 模型（fluid.io.save_inference_model 保存的模型）一般是模型训练完成后保存的固化模型，用于预测部署。与 persistable 模型相比，inference 模型会额外保存模型的结构信息，用于配合权重文

件构成完整的模型。如下所示，`model` 中保存的即为模型的结构信息。

```
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
  model
```

为了方便起见，paddle 在保存 inference 模型的时候也可以将所有的权重文件保存成一个 `params` 文件，如下所示：

```
resnet50-vd
  model
  params
```

在 Paddle 中训练引擎和预测引擎都支持模型的预测推理，只不过预测引擎不需要进行反向操作，因此可以进行定制型的优化（如层融合，kernel 选择等），达到低时延、高吞吐的目的。训练引擎既可以支持 persistable 模型，也可以支持 inference 模型，而预测引擎只支持 inference 模型，因此也就衍生出了三种不同的预测方式：

1. 预测引擎 + inference 模型
2. 训练引擎 + persistable 模型
3. 训练引擎 + inference 模型

不管是何种预测方式，基本都包含以下几个主要的步骤：

- 构建引擎
- 构建待预测数据
- 执行预测
- 预测结果解析

不同预测方式，主要有两方面不同：构建引擎和执行预测，以下的几个部分我们会具体介绍。

5.1.2 二、模型转换

在任务的训练阶段，通常我们会保存一些 checkpoint（persistable 模型），这些只是模型权重文件，不能直接被预测引擎直接加载预测，所以我们通常会在训练完之后，找到合适的 checkpoint 并将其转换为 inference

模型。主要分为两个步骤：1. 构建训练引擎，2. 保存 inference 模型，如下所示：

```
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=persistable 模型路径, 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')
```

在模型库的 `tools/export_model.py` 中提供了完整的示例，只需执行下述命令即可完成转换：

```
python tools/export_model.py \
    --m= 模型名称 \
    --p=persistable 模型路径 \
    --o=model 和 params 保存路径
```

5.1.3 三、预测引擎 + inference 模型预测

在模型库的 `tools/infer/predict.py` 中提供了完整的示例，只需执行下述命令即可完成预测：

```
python ./tools/infer/predict.py \
    -i=./test.jpeg \
```

(下页继续)

(续上页)

```
    -m=./resnet50-vd/model \
    -p=./resnet50-vd/params \
    --use_gpu=1 \
    --use_tensorrt=True
```

参数说明：

- `image_file`(简写 i)：待预测的图片文件路径，如 `./test.jpeg`
- `model_file`(简写 m)：模型文件路径，如 `./resnet50-vd/model`
- `params_file`(简写 p)：权重文件路径，如 `./resnet50-vd/params`
- `batch_size`(简写 b)：批大小，如 `1`
- `ir_optim`：是否使用 `IR` 优化，默认值：True
- `use_tensorrt`：是否使用 TesorRT 预测引擎，默认值：True
- `gpu_mem`：初始分配 GPU 显存，以 M 单位
- `use_gpu`：是否使用 GPU 预测，默认值：True
- `enable_benchmark`：是否启用 benchmark，默认值：False
- `model_name`：模型名字

注意：当启用 benchmark 时，默认开启 tersorrt 进行预测

构建预测引擎：

```python
from paddle.fluid.core import AnalysisConfig
from paddle.fluid.core import create_paddle_predictor
config = AnalysisConfig(model 文件路径, params 文件路径)
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 copy 方式需要去除 fetch feed op
config.switch_use_feed_fetch_ops(False)

predictor = create_paddle_predictor(config)
```

执行预测：

```
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()
```

更多预测参数说明可以参考官网 Paddle Python 预测 API。如果需要在业务的生产环境部署，也推荐使用 Paddel C++ 预测 API，官网提供了丰富的预编译预测库 Paddle C++ 预测库。

默认情况下，Paddle 的 wheel 包中是不包含 TensorRT 预测引擎的，如果需要使用 TensorRT 进行预测优化，需要自己编译对应的 wheel 包，编译方式可以参考 Paddle 的编译指南 Paddle 编译。

5.1.4 四、训练引擎 + persistable 模型预测

在模型库的 `tools/infer/infer.py` 中提供了完整的示例，只需执行下述命令即可完成预测：

```
python tools/infer/infer.py \
    --i= 待预测的图片文件路径 \
    --m= 模型名称 \
    --p=persistable 模型路径 \
    --use_gpu=True
```

参数说明：

- `image_file`(简写 i)：待预测的图片文件路径，如 `./test.jpeg`
- `model_file`(简写 m)：模型文件路径，如 `./resnet50-vd/model`
- `params_file`(简写 p)：权重文件路径，如 `./resnet50-vd/params`
- `use_gpu`：是否开启 GPU 训练，默认值：True

训练引擎构建：

由于 persistable 模型不包含模型的结构信息，因此需要先构建出网络结构，然后 load 权重来构建训练引擎。

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

place = fluid.CPUPlace()
exe = fluid.Executor(place)
startup_prog = fluid.Program()
```

(下页继续)

(续上页)

```
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=persistable 模型路径, executor=exe)
```

执行预测:

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

上述执行预测时候的参数说明可以参考官网 fluid.Executor

5.1.5 五、训练引擎 + inference 模型预测

在模型库的 `tools/infer/py_infer.py` 中提供了完整的示例，只需执行下述命令即可完成预测:

```
python tools/infer/py_infer.py \
    --i= 图片路径 \
    --d= 模型的存储路径 \
    --m= 保存的模型文件 \
    --p= 保存的参数文件 \
    --use_gpu=True
```

- `image_file`(简写 i): 待预测的图片文件路径，如 `./test.jpeg`
- `model_file`(简写 m): 模型文件路径，如 `./resnet50_vd/model`
- `params_file`(简写 p): 权重文件路径，如 `./resnet50_vd/params`
- `model_dir`(简写 d): 模型路径，如`./resent50_vd`
- `use_gpu`: 是否开启 GPU，默认值: True

训练引擎构建:

由于 inference 模型已包含模型的结构信息，因此不再需要提前构建模型结构，直接 load 模型结构和权重文件来构建训练引擎。

```
import fluid

place = fluid.CPUPlace()
exe = fluid.Executor(place)
[program, feed_names, fetch_lists] = fluid.io.load_inference_model(
        模型的存储路径,
        exe,
        model_filename= 保存的模型文件,
        params_filename= 保存的参数文件)
compiled_program = fluid.compiler.CompiledProgram(program)
```

`load_inference_model` 既支持零散的权重文件集合，也支持融合后的单个权重文件。

执行预测：

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

上述执行预测时候的参数说明可以参考官网 fluid.Executor

5.2 Paddle-Lite

5.2.1 一、简介

Paddle-Lite 是飞桨推出的一套功能完善、易用性强且性能卓越的轻量化推理引擎。轻量化体现在使用较少比特数用于表示神经网络的权重和激活，能够大大降低模型的体积，解决终端设备存储空间有限的问题，推理性能也整体优于其他框架。PaddleClas 使用 Paddle-Lite 进行了*移动端模型的性能评估*，本部分以 `ImageNet1k` 数据集的 `MobileNetV1` 模型为例，介绍怎样使用 `Paddle-Lite`，在移动端 (基于骁龙 855 的安卓开发平台) 对进行模型速度评估。

5.2.2 二、评估步骤

2.1 导出 inference 模型

- 首先需要将训练过程中保存的模型存储为用于预测部署的固化模型，可以使用 `tools/export_model.py` 导出 inference 模型，具体使用方法如下。

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

最终在 `inference/MobileNetV1` 文件夹下会保存得到 `model` 与 `parmas` 文件。

2.2 benchmark 二进制文件下载

- 使用 adb(Android Debug Bridge) 工具可以连接 Android 手机与 PC 端，并进行开发调试等。安装好 adb，并确保 PC 端和手机连接成功后，使用以下命令可以查看手机的 ARM 版本，并基于此选择合适的预编译库。

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

- 下载 benchmark_bin 文件

```
wget -c https://paddle-inference-dist.bj.bcebos.com/PaddleLite/benchmark_0/benchmark_bin_
↪v8
```

如果查看的 ARM 版本为 v7，则需要下载 v7 版本的 benchmark_bin 文件，下载命令如下。

```
wget -c https://paddle-inference-dist.bj.bcebos.com/PaddleLite/benchmark_0/benchmark_bin_
↪v7
```

2.3 模型速度 benchmark

PC 端和手机连接成功后，使用下面的命令开始模型评估。

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

其中./benchmark_bin_v8 为 benchmark 二进制文件路径，`./inference` 为所有需要评测的模型的路径，`result_armv8.txt` 为保存的结果文件，最后的参数 `true` 表示在评估之后会首先进行模型优化。最终在当前文件夹下会输出 `result_armv8.txt` 的评估结果文件，具体信息如下。

```
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
```

这里给出了不同线程数下的模型预测速度，单位为 FPS，以线程数为 1 为例，MobileNetV1 在骁龙 855 上的平均速度为 30.79750FPS。

2.4 模型优化与速度评估

- 在 2.3 节中提到了在模型评估之前对其进行优化，在这里也可以首先对模型进行优化，再直接加载优化后的模型进行速度评估。
- Paddle-Lite 提供了多种策略来自动优化原始的训练模型，其中包括量化、子图融合、混合调度、Kernel 优选等等方法。为了使优化过程更加方便易用，Paddle-Lite 提供了 opt 工具来自动完成优化步骤，输出一个轻量的、最优的可执行模型。可以在Paddle-Lite 模型优化工具页面下载。在这里以 MacOS 开发环境为例，下载opt_mac模型优化工具，并使用下面的命令对模型进行优化。

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

其中 model_file 与 param_file 分别是导出的 inference 模型结构文件与参数文件地址，转换成功后，会在 opt_models 文件夹下生成 MobileNetV1.nb 文件。

使用 benchmark_bin 文件加载优化后的模型进行评估，具体的命令如下。

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

最终 result_armv8.txt 中结果如下。

```
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
```

以线程数为 1 为例，MobileNetV1 在骁龙 855 上的平均速度为 30.84173FPS。

更加具体的参数解释与 Paddle-Lite 使用方法可以参考 Paddle-Lite 文档。

5.3 模型量化

模型量化是 PaddleSlim 的特色功能之一，支持动态和静态两种量化训练方式，对权重全局量化和 Channel-Wise 量化，同时以兼容 Paddle-Lite 的格式保存模型。PaddleClas 使用该量化工具，量化了 78.9% 的 mobilenet_v3_large_x1_0 的蒸馏模型, 量化后 SD855 上预测速度从 19.308ms 加速到 14.395ms，存储大小从 21M 减小到 10M，top1 识别准确率 75.9%。具体的训练方法可以参见 PaddleSlim 量化训练。

5.4 多机训练

分布式训练的高性能，是飞桨的核心优势技术之一，在分类任务上，分布式训练可以达到几乎线性的加速比。Fleet 是用于 PaddlePaddle 分布式训练的高层 API，基于这套接口用户可以很容易切换到分布式训练程序。为了可以同时支持单机训练和多机训练，PaddleClas 采用 Fleet API 接口，更多的分布式训练可以参考 Fleet API 设计文档。

5.5 Paddle Hub

PaddleHub 是飞桨生态的预训练模型应用工具，开发者可以便捷地使用高质量的预训练模型结合 Fine-tune API 快速完成模型迁移到部署的全流程工作。PaddleHub 收录了 PaddleClas 的所有预训练模型，更多使用细节请查看 PaddleHub 官网。

5.6 模型服务化部署

5.6.1 一、简介

Paddle Serving 旨在帮助深度学习开发者轻易部署在线预测服务，支持一键部署工业级的服务能力、客户端和服务端之间高并发和高效通信、并支持多种编程语言开发客户端。

该部分以 HTTP 预测服务部署为例，介绍怎样在 PaddleClas 中使用 PaddleServing 部署模型服务。

5.6.2 二、Serving 安装

Serving 官网推荐使用 docker 安装并部署 Serving 环境。首先需要拉取 docker 环境并创建基于 Serving 的 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
```

进入 docker 后，需要安装 Serving 相关的 python 包。

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

- 如果安装速度太慢，可以通过-i `https://pypi.tuna.tsinghua.edu.cn/simple` 更换源，加速安装过程。
- 如果希望部署 CPU 服务，可以安装 serving-server 的 cpu 版本，安装命令如下。

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

三、导出模型

使用 `tools/export_serving_model.py` 脚本导出 Serving 模型，以 `ResNet50_vd` 为例，使用方法如下。

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

最终在 serving 文件夹下会生成 `ppcls_client_conf` 与 `ppcls_model` 两个文件夹，分别存储了 client 配置、模型参数与结构文件。

四、服务部署与请求

- 使用下面的方式启动 Serving 服务。

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

其中 `serving/ppcls_model` 为刚才保存的 Serving 模型地址，`workdir` 为为工作目录，`9292` 为服务的端口号。

- 使用下面的脚本向 Serving 服务发送识别请求，并返回结果。

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

`9292` 为发送请求的端口号，需要与服务启动时的端口号保持一致，`./docs/images/logo.png` 为待识别的图像文件。最终返回 Top1 识别结果的类别 ID 以及概率值。

- 更多的服务部署类型，如 RPC 预测服务等，可以参考 Serving 的 github 官网：https://github.com/PaddlePaddle/Serving/tree/develop/python/examples/imagenet

CHAPTER 6

赛事支持

PaddleClas 的建设源于百度实际视觉业务应用的淬炼和视觉前沿能力的探索，助力多个视觉重点赛事取得领先成绩，并且持续推进更多的前沿视觉问题的解决和落地应用。

- 2018 年 Kaggle Open Images V4 图像目标检测挑战赛冠军
- 2019 年 Kaggle Open Images V5 图像目标检测挑战赛亚军
  - 技术报告可以参考：https://arxiv.org/pdf/1911.07171.pdf
  - 详细文档与开源的模型可以参考：OIDV5 目标检测 github 地址
- 2019 年 Kaggle 地标检索挑战赛亚军
  - 技术报告可以参考：https://arxiv.org/abs/1906.03990
  - 详细文档与开源的模型可以参考：2019 地标检索和识别 github 地址
- 2019 年 Kaggle 地标识别挑战赛亚军
  - 技术报告可以参考：https://arxiv.org/abs/1906.03990
  - 详细文档与开源的模型可以参考：2019 地标检索和识别 github 地址
- 首届多媒体信息识别技术竞赛中印刷文本 OCR、人脸识别和地标识别三项任务 A 级证书

CHAPTER 7

更新日志

- 2020.07.14
  - 添加 Res2Net200_vd_26w_4s_ssld 模型，在 ImageNet 上 Top-1 Acc 可达 85.13%。
  - 添加 Fix_ResNet50_vd_ssld_v2 模型,，在 ImageNet 上 Top-1 Acc 可达 84.0%。
- 2020.06.17
  - 添加英文文档。
- 2020.06.12
  - 添加对 windows 和 CPU 环境的训练与评估支持。
- 2020.05.17
  - 添加混合精度训练。
- 2020.05.09
  - 添加 Paddle Serving 使用文档。
  - 添加 Paddle-Lite 使用文档。
  - 添加 T4 GPU 的 FP32/FP16 预测速度 benchmark。
- 2020.04.10:
  - 第一次提交。

CHAPTER 8

FAQ

- Q: 多卡评估时，为什么每张卡输出的精度指标不相同？
- A: 目前 PaddleClas 基于 fleet api 使用多卡，在多卡评估时，每张卡都是单独读取各自 part 的数据，不同卡中计算的图片是不同的，因此最终指标也会有微量差异，如果希望得到准确的评估指标，可以使用单卡评估。

- Q: 在配置文件的 `TRAIN` 字段中配置了 `mix` 的参数，为什么 `mixup` 的数据增广预处理没有生效呢？
- A: 使用 mixup 时，数据预处理部分与模型输入部分均需要修改，因此还需要在配置文件中显式地配置 `use_mix: True`，才能使得 `mixup` 生效。

- Q: 评估和预测时，已经指定了预训练模型所在文件夹的地址，但是仍然无法导入参数，这么为什么呢？
- A: 加载预训练模型时，需要指定预训练模型的前缀，例如预训练模型参数所在的文件夹为 `output/ResNet50_vd/19`，预训练模型参数的名称为 `output/ResNet50_vd/19/ppcls.pdparams`，则 `pretrained_model` 参数需要指定为 `output/ResNet50_vd/19/ppcls`，PaddleClas 会自动补齐 `.pdparams` 的后缀。

- Q: 在评测 `EfficientNetB0_small` 模型时，为什么最终的精度始终比官网的低 0.3% 左右？
- A: `EfficientNet` 系列的网络在进行 resize 的时候，是使用 `cubic` 插值方式 (resize 参数的 interpolation 值设置为 2)，而其他模型默认情况下为 None，因此在训练和评估的时候需要显式地指定 resize

的 interpolation 值。具体地，可以参考以下配置中预处理过程中 ResizeImage 的参数。

```
VALID:
    batch_size: 16
    num_workers: 4
    file_list: "./dataset/ILSVRC2012/val_list.txt"
    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: 如果想将保存的 `pdparams` 模型参数文件转换为早期版本 (Paddle1.7.0 之前) 的零碎文件 (每个文件均为一个单独的模型参数)，该怎么实现呢？
- A: 可以首先导入 `pdparams` 模型，之后使用 `fluid.io.save_vars` 函数将模型保存为零散的碎文件。

*PaddlePaddle*2020