Lab4-2 Diabetic Retinopathy Detection

Deep Learning and Practice

資科工碩1 游騰德 310551054

Introduction

This lab is to implement ResNet18 and ResNet50 to analysis diabetic retinopathy. The networks to be implemented will be trained and tested on Diabetic Retinopathy Detection dataset from kaggle. This dataset contains 35124 images, including 28099 and 7025 of training and testing images respectively. The main goal is to take the input data and classify them into 5 classes. Additionally, the dataloader is required to be customized. Finally, confusion matrices and comparison figures will be calculated and shown to evaluate models performance. Please refer to the following of the report for further implementing details.

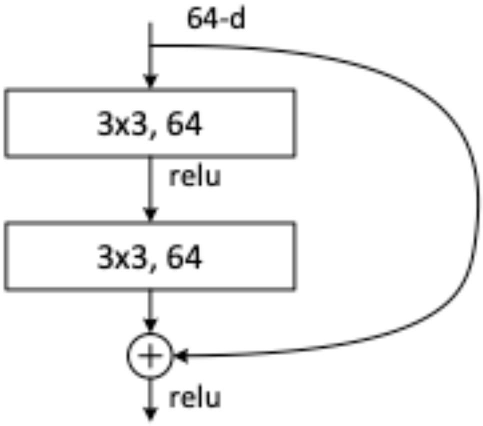
Experiment setups

Model details

layer name	output size	18-layer	34-layer	50-layer
conv1	112×112	7×7, 64, stride 2		
conv2_x	56×56	3×3 max pool, strid		
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc,		
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9

ResNet architectures

ResNet18

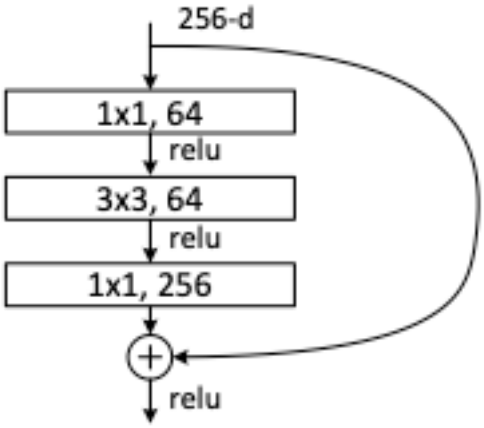


Basic block

```
BasicBlock(
  (conv1): Conv2d
  (bn1): BatchNorm2d
  (relu): ReLU
  (conv2): Conv2d
  (bn2): BatchNorm2d
  (downsample): Sequential(
    (0): Conv2d
    (1): BatchNorm2d
  )
)

ResNet18(
  (classify): Linear(in_features=512, out_features=5, bias=True)
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock
    (1): BasicBlock
  )
  (layer2): Sequential(
    (0): BasicBlock
    (1): BasicBlock
  )
  (layer3): Sequential(
    (0): BasicBlock
    (1): BasicBlock
  )
  (layer4): Sequential(
    (0): BasicBlock
    (1): BasicBlock
  )
  (avgpool): AdaptiveAvgPool2d(output_size=1)
)
```

ResNet50



Bottleneck block

```
Bottleneck(
  (conv1): Conv2d
  (bn1): BatchNorm2d
  (conv2): Conv2d
  (bn2): BatchNorm2d
  (conv3): Conv2d
  (bn3): BatchNorm2d
  (relu): ReLU
  (downsample): Sequential(
    (0): Conv2d
    (1): BatchNorm2d
  )
)

ResNet50(
  (classify): Linear(in_features=2048, out_features=5, bias=True)
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck
    (1): Bottleneck
    (2): Bottleneck
  )
  (layer2): Sequential(
    (0): Bottleneck
    (1): Bottleneck
    (2): Bottleneck
    (3): Bottleneck
  )
  (layer3): Sequential(
    (0): Bottleneck
    (1): Bottleneck
    (2): Bottleneck
    (3): Bottleneck
    (4): Bottleneck
    (5): Bottleneck
  )
  (layer4): Sequential(
    (0): Bottleneck
    (1): Bottleneck
    (2): Bottleneck
  )
  (avgpool): AdaptiveAvgPool2d(output_size=1)
)
```

Hyperparameters

- Batch size = 8
- Learning rate = 1e-3*
- Epochs = 10 (ResNet18), 5 (ResNet50)
- Optimizer: SGD
- Loss function: Cross Entropy Loss
- Weight decay = 5e-4*

* Finetuned after several epochs of training.

Dataloader

The custom dataset inherits **torch.utils.data.Dataset** and override the following methods:

__len__

Returns the size of the dataset.

__getitem__

Support the indexing to returns the query data.

There are mainly 3 steps in this function: image loading, label loading, and data transformation.

To load the image, I utilize **Pillow.Image** to open the image file according to **train_img.csv** or **test_img.csv** file.

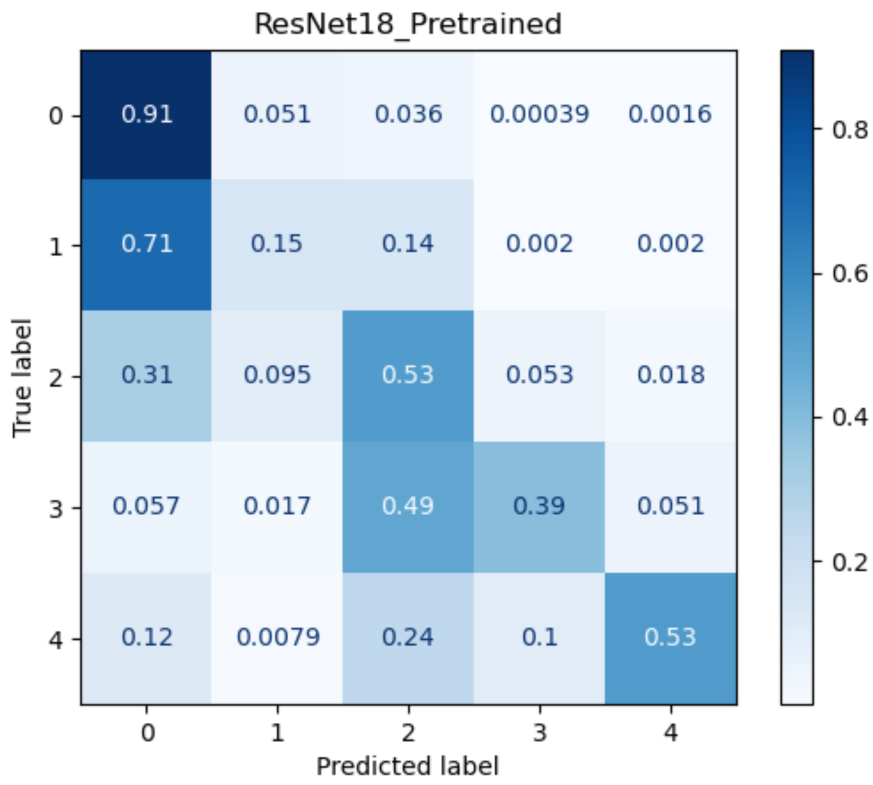
To transform the data, I utilize **transforms.ToTensor** to transpose the image shape from [H, W, C] to [C, H, W], and **transforms.Normalize** to apply normalization. Additionally, I apply **transforms.RandomHorizontalFlip()** and **transforms.RandomVerticalFlip()** for data augmentation while training.

I tried to apply histogram equalization by **transforms.functional.equalize** to all images, but did not results in good performance. Further details will be discussed at the end of the report.

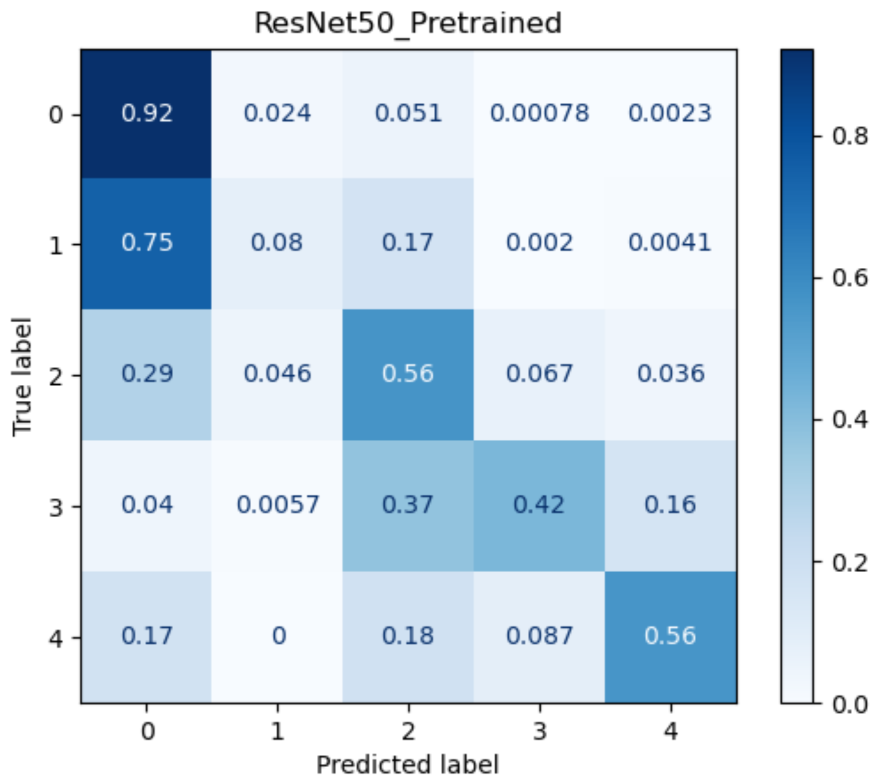
Confusion matrix

A confusion matrix describes the performance of a classification model on a set of test data with known true values. Normalized confusion matrices of my final pre-trained model in this lab are listed.

ResNet18



ResNet50



Experimental results

Highest testing accuracy

Screenshot

```
Inference
> Found 7025 images...
ResNet18_Pretrained Test Acc:           0.8216370106761566
ResNet50_Pretrained Test Acc:           0.8220640569395018
```

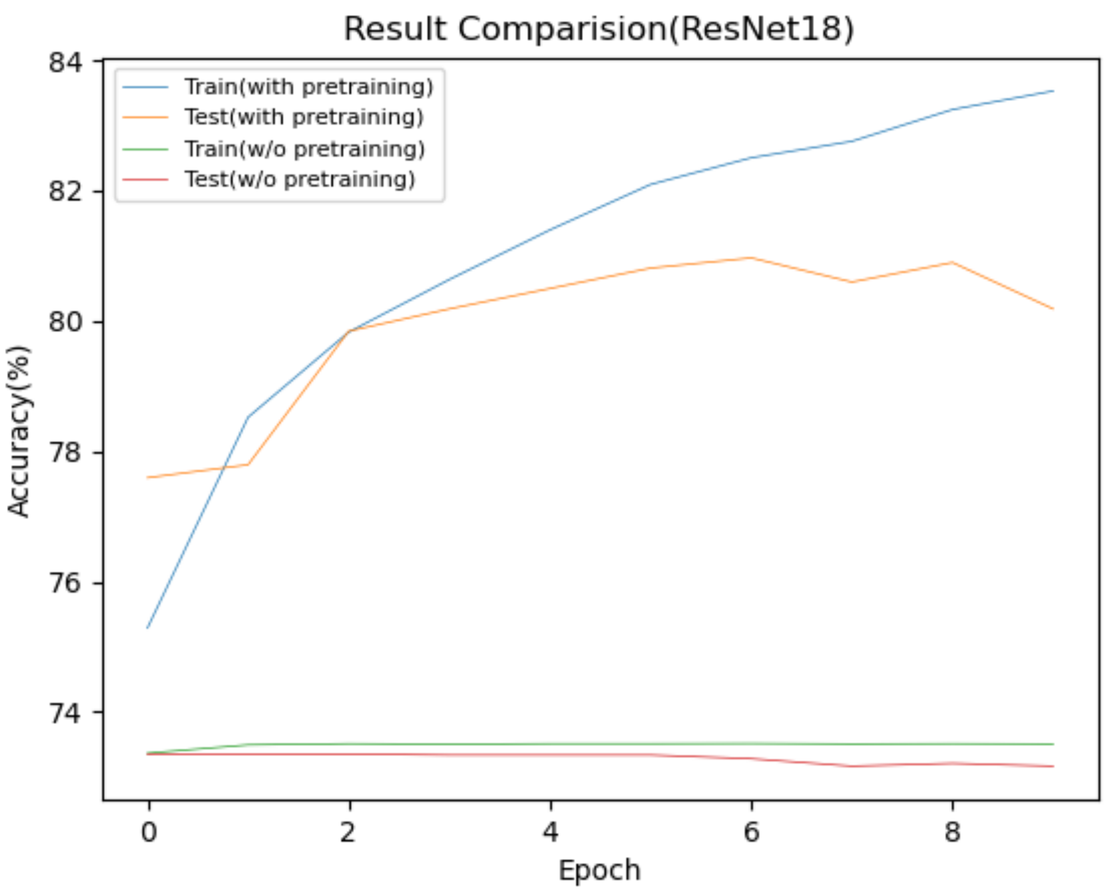
- Best of ResNet18: w/ pre-trained weights, **82.16%** accuracy.
- Best of ResNet50: w/ pre-trained weights, **82.21%** accuracy.

Observation

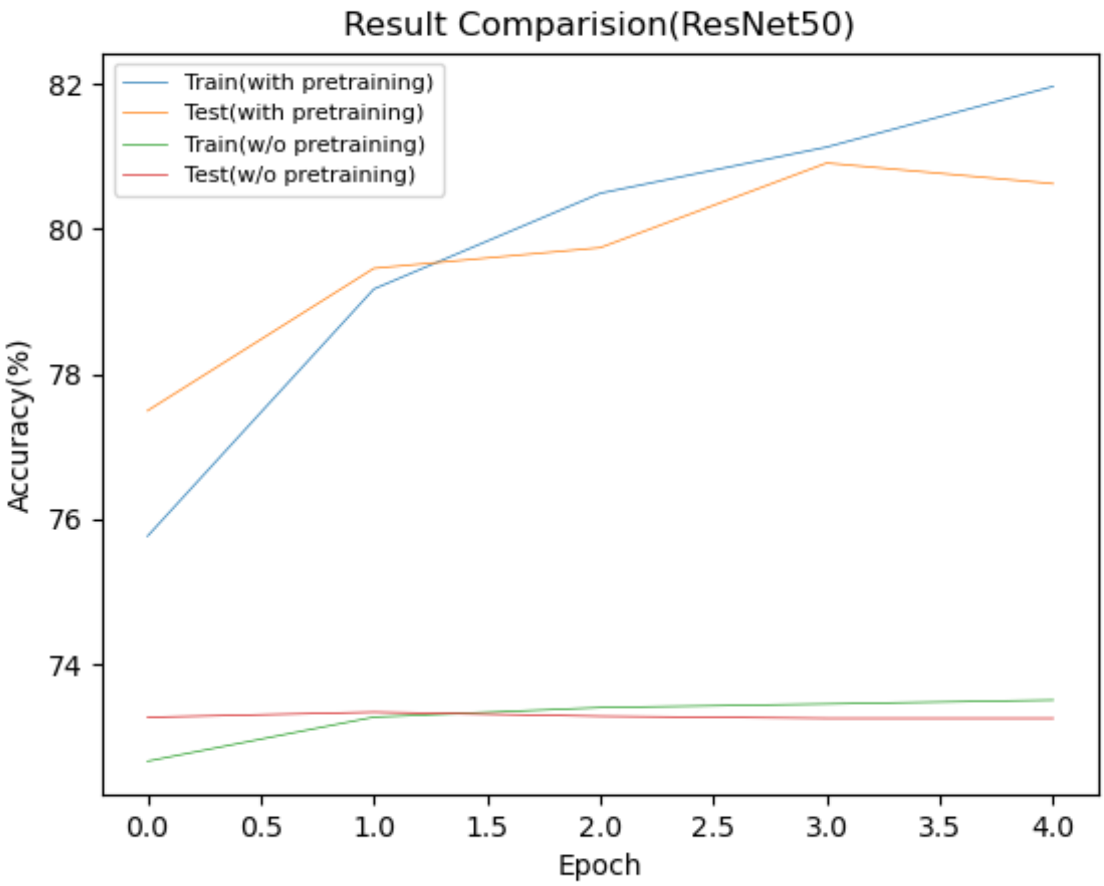
In this case, ResNet50 performs better than ResNet18 by achieving slightly higher accuracy. The FLOPs of ResNet50 is much more than that of ResNet18 as well.

Comparison figures

ResNet18



ResNet50

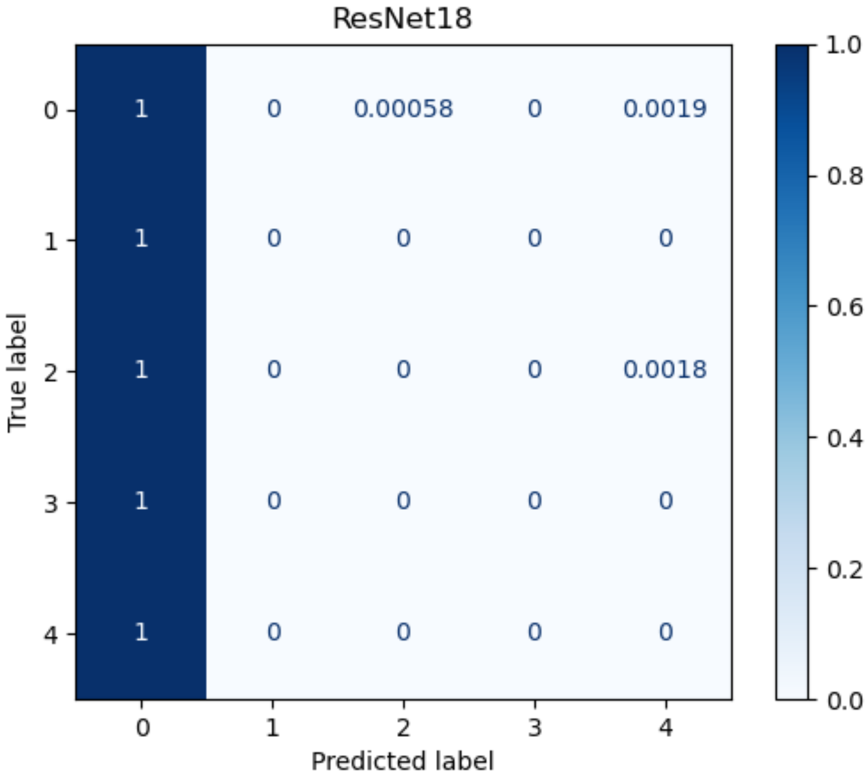


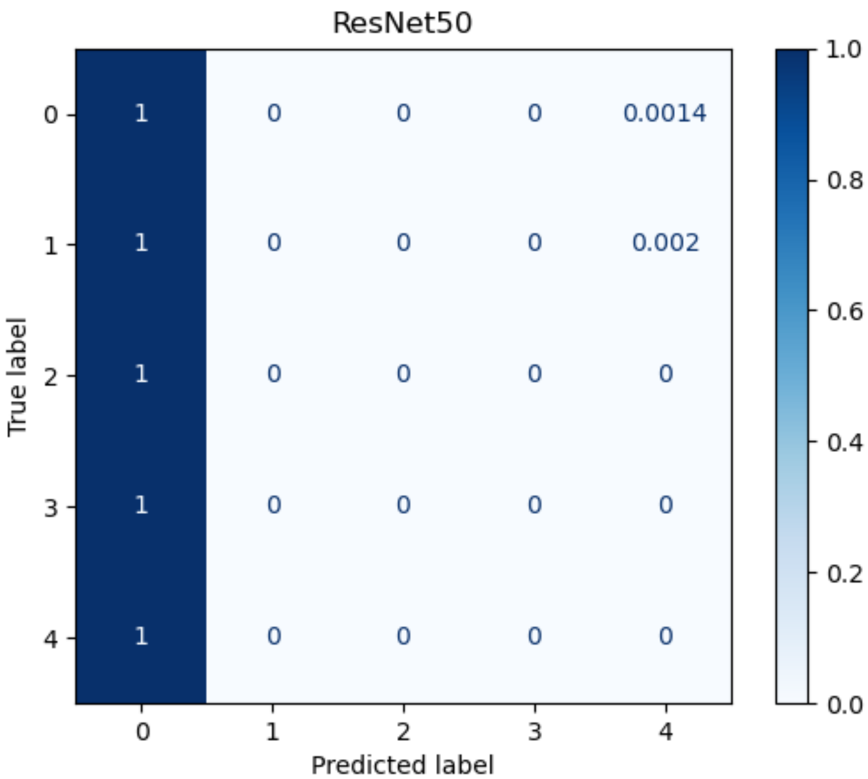
Discussion

- The performance of both models without pre-taining is found to be unacceptable.
- To achieve satisfactory performance, I finetuned the models after 10 epochs of ResNet18 training and 5 epochs of ResNet50 training.
- In this lab, I applied histogram equalization via **transforms.functional.equalize** during data transformation.

Without pre-training

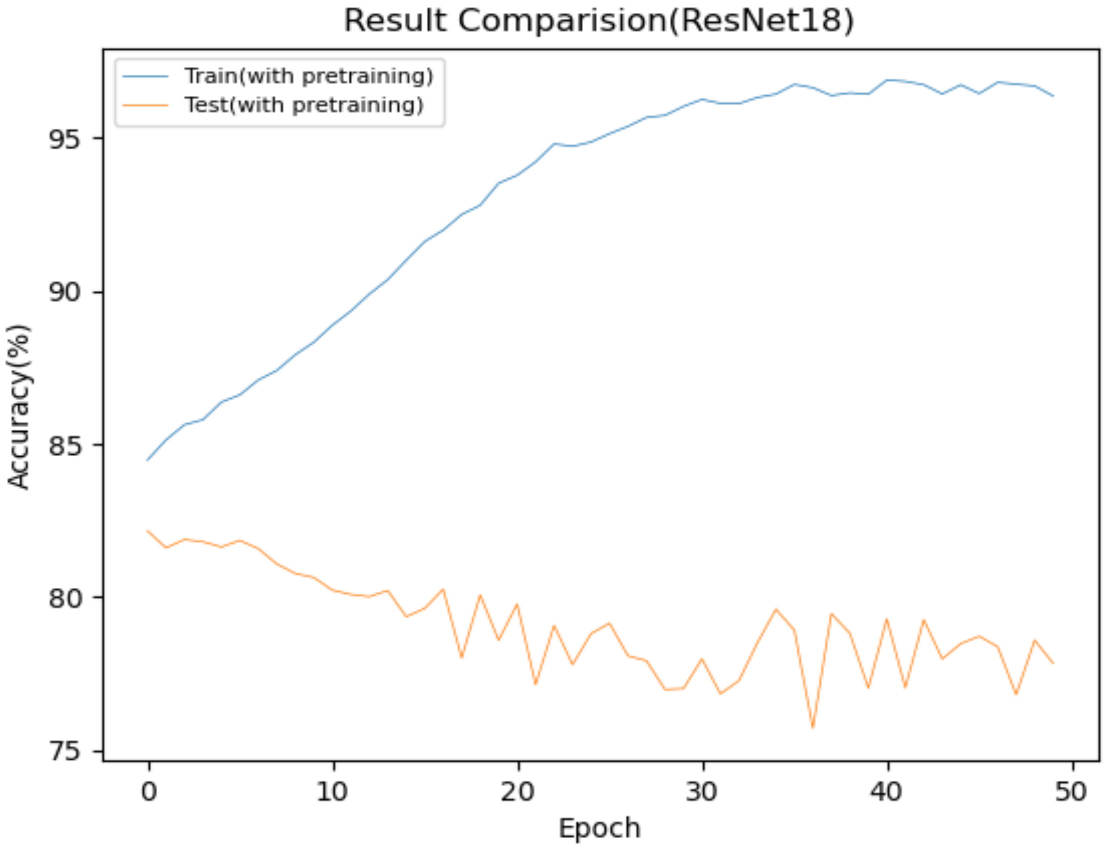
The confusion matrix of models without pre-training are listed below. Both models predict all data to class 0. My guess is that this caused by the extremely imbalance of data.

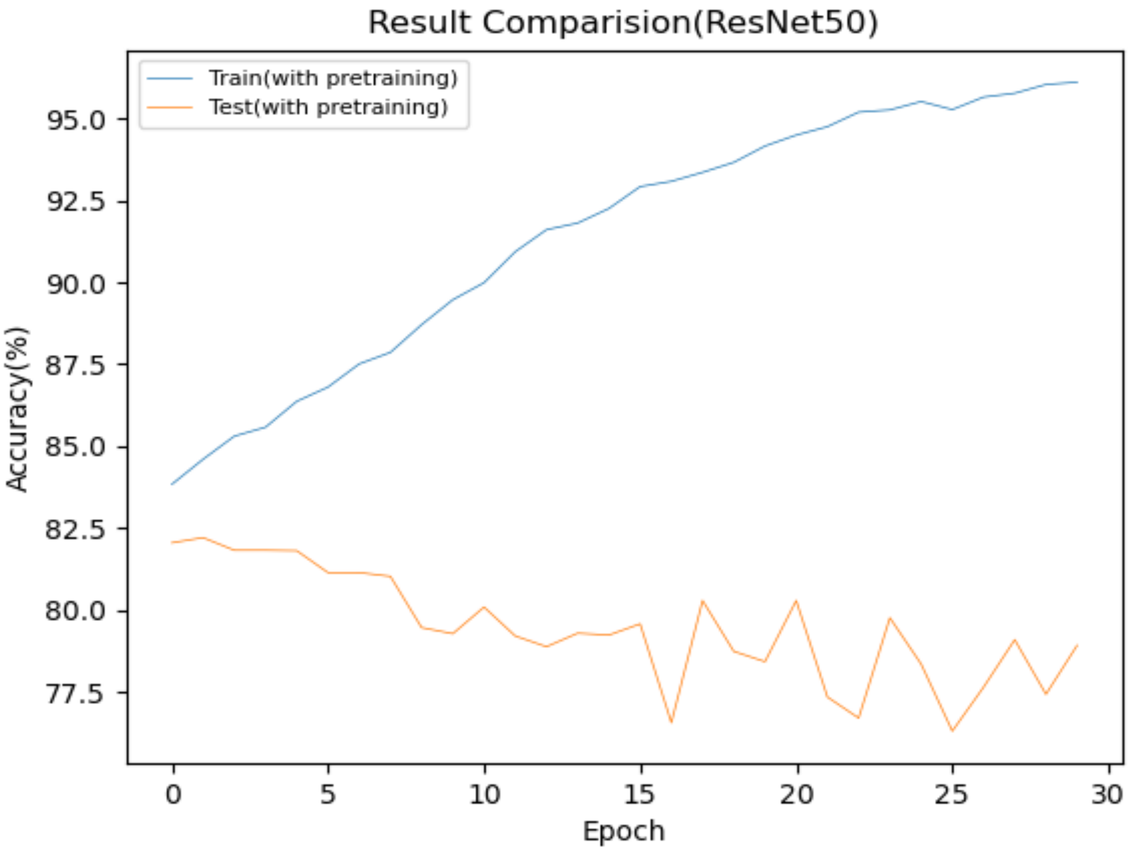




Finetune

I lowered the learning rate to 1e-4 and set weight decay to 5e-3 and kept train for several epochs. The performance increased at the beginning but end in overfitting.


Result Comparision(ResNet18)
Train(with pretraining)
Test(with pretraining)
Accuracy(%)
Epoch
75
80
85
90
95
0
10
20
30
40
50


Result Comparision(ResNet50)
Train(with pretraining)
Test(with pretraining)
Accuracy(%)
Epoch
77.5
80.0
82.5
85.0
87.5
90.0
92.5
95.0
0
5
10
15
20
25
30

Histogram equalization

Histogram equalization is an image processing technique to improve contrast of images by spreading out intensity values. As I applied histogram equalization to every input images, the highest accuracy of both ResNet18 and ResNet50 dropped by approximately 1%. This phenomena roughly shows that histogram equalization is not beneficial to training in this case. My guess is that the area near the image corners has too many black pixels which influence the results of histogram equalization. The overall performance is shown below for your reference.

```
Best Test Acc of ResNet18_Pretrained: 0.7997153024911032
Best Test Acc of ResNet50_Pretrained: 0.7994306049822064
```