Lab3 Diabetic Retionopathy Detection by ResNet

0856029 Yu-Shao Liu

April 20, 2020

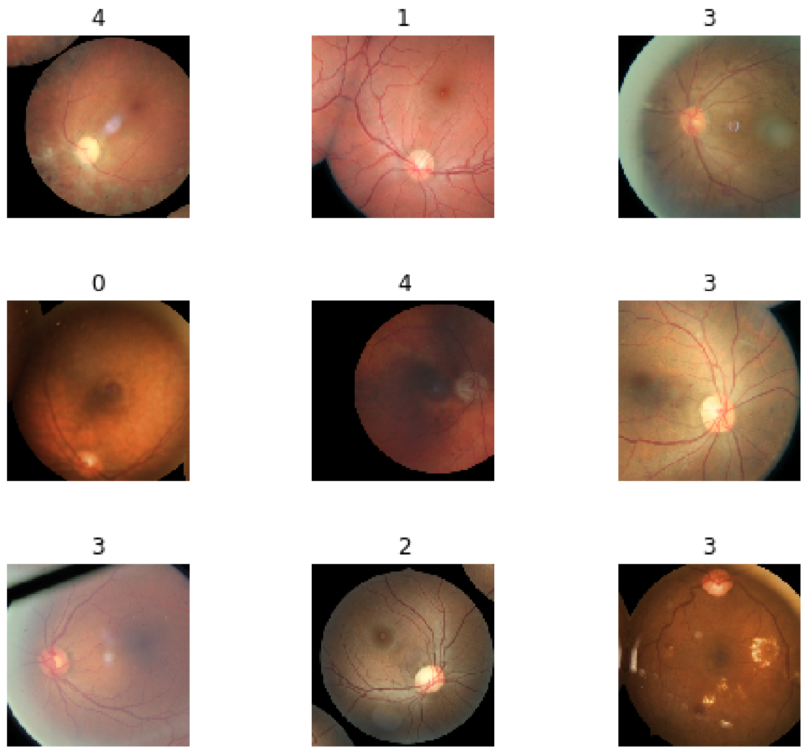
1 Introduction

It is more difficult to train Deep Neural Network while the network gets deeper due to gradient vanishing problem. ResNet add the "skip connection" into its network architecture, so that the gradient propagation might be more smooth.

In this lab, we are going to use ResNet18 and ResNet50 to classify 5 classes of Retinopathy images and compare results of whether fine tuning pretrained parameters.

1.1 Retinopathy Images

Both training and testing data are extracted from the Diabetic Retinopathy Detection dataset on kaggle. Each image are processed into 512 x 512 with RGB channels.

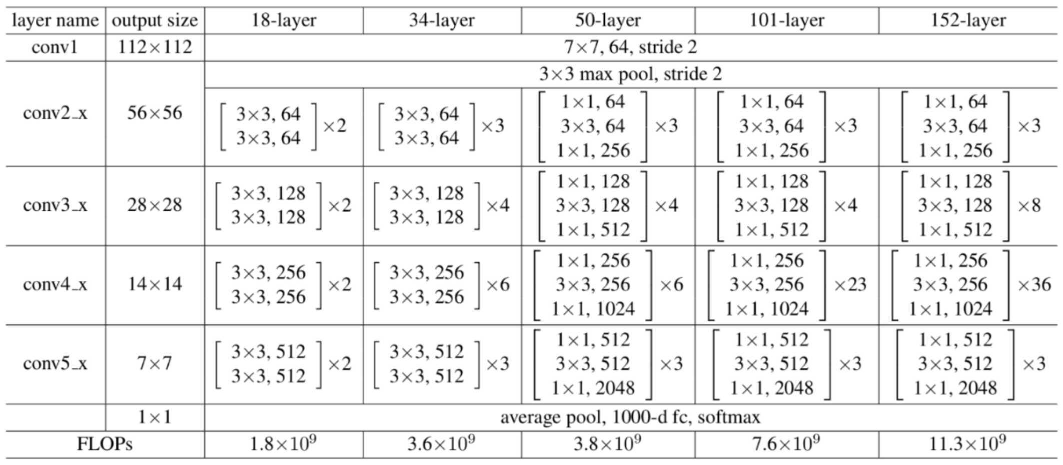


2 Experimental Setup

We compare ResNet18 and ResNet50 both with and without pretrained parameters.

2.1 ResNet

In this lab, just take the model that torchvision provides, and replace the dimension of the last layer with a new full connected layer so that we could fine-tune this network.



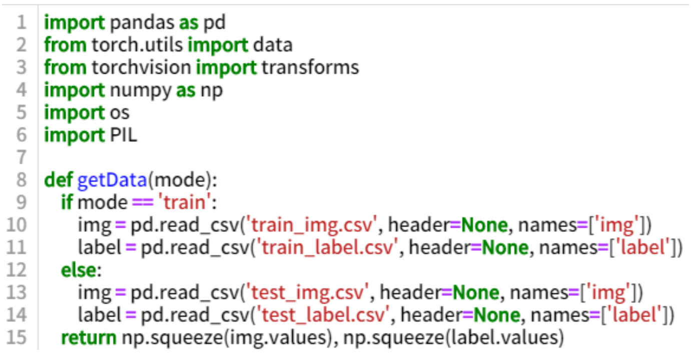
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 1: Network architecture of ResNet variants

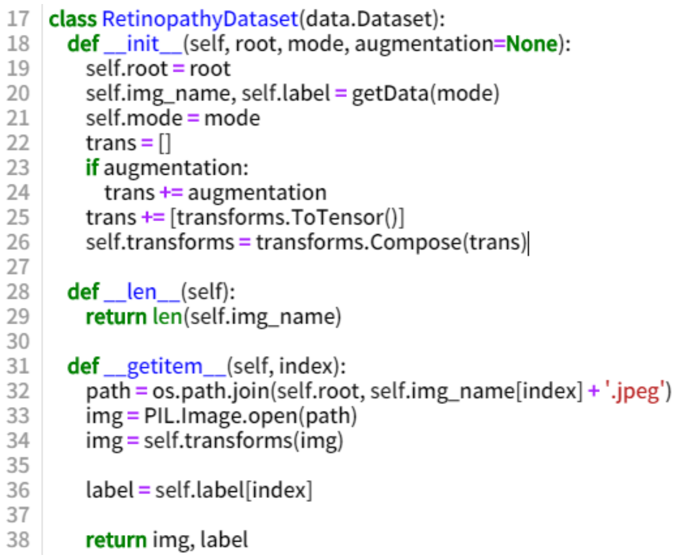
2.2 dataloader

Most of codes of `dataloader.py` are similar to codes TA provided, but it seems that the first row of data would become `pd.DataFrame`'s header and get discarded, so I add some parameters on `pd.read_csv` to reserver the first row of data.

To solve unbalanced data issue, there is a new parameter called `augemtation` in the class `RetinopathyDataset`, which could pass in a list of data preprocess operation in torchvision.transform.



```python
import pandas as pd
from torch.utils import data
from torchvision import transforms
import numpy as np
import os
import PIL

def getData(mode):
    if mode == 'train':
        img = pd.read_csv('train_img.csv', header=None, names=['img'])
        label = pd.read_csv('train_label.csv', header=None, names=['label'])
    else:
        img = pd.read_csv('test_img.csv', header=None, names=['img'])
        label = pd.read_csv('test_label.csv', header=None, names=['label'])
    return np.squeeze(img.values), np.squeeze(label.values)
```

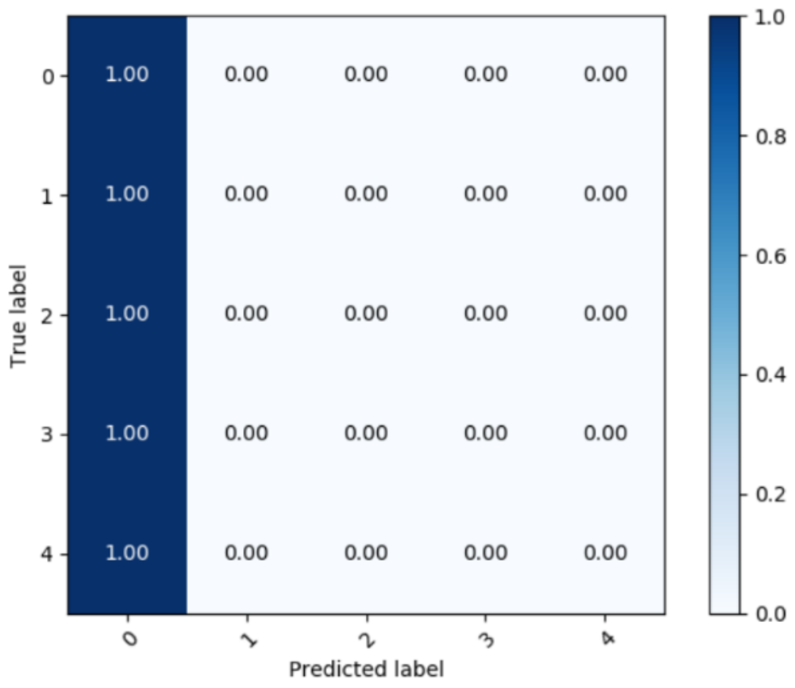


```python
class RetinopathyDataset(data.Dataset):
    def __init__(self, root, mode, augmentation=None):
        self.root = root
        self.img_name, self.label = getData(mode)
        self.mode = mode
        trans = []
        if augmentation:
            trans += augmentation
        trans += [transforms.ToTensor()]
        self.transforms = transforms.Compose(trans)

    def __len__(self):
        return len(self.img_name)

    def __getitem__(self, index):
        path = os.path.join(self.root, self.img_name[index] + '.jpeg')
        img = PIL.Image.open(path)
        img = self.transforms(img)

        label = self.label[index]

        return img, label
```

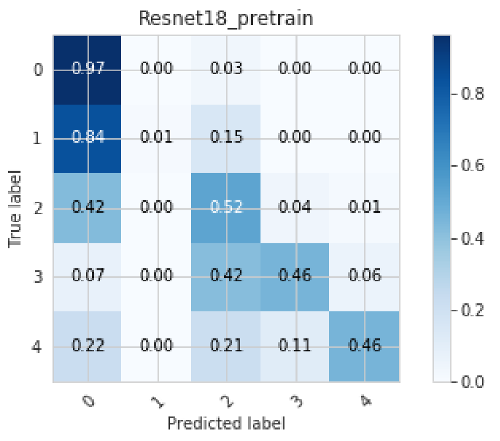
2.3 confusion matrix

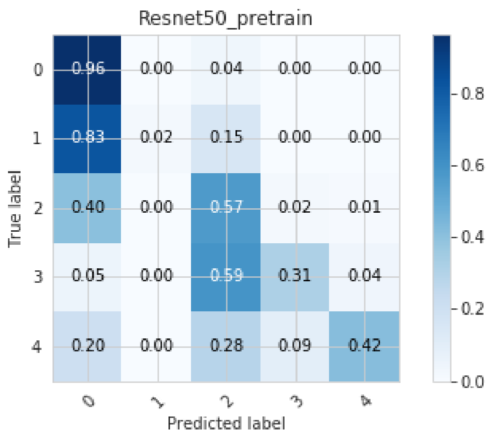
Without pretained parameters, models tend to predict "0" in all cases, getting about 70% accuracy.

Take the result of ResNet18 as example:



Using pretrained models, we could more easily escape some local optimum to get better results. We could find that there are more instances labelled "2", "3", or "4" correctly predicted. However, the instances labelled "1" are still hard to classify. Maybe this results shows that slight Retinopathy does not have obvious difference with normal Retina.





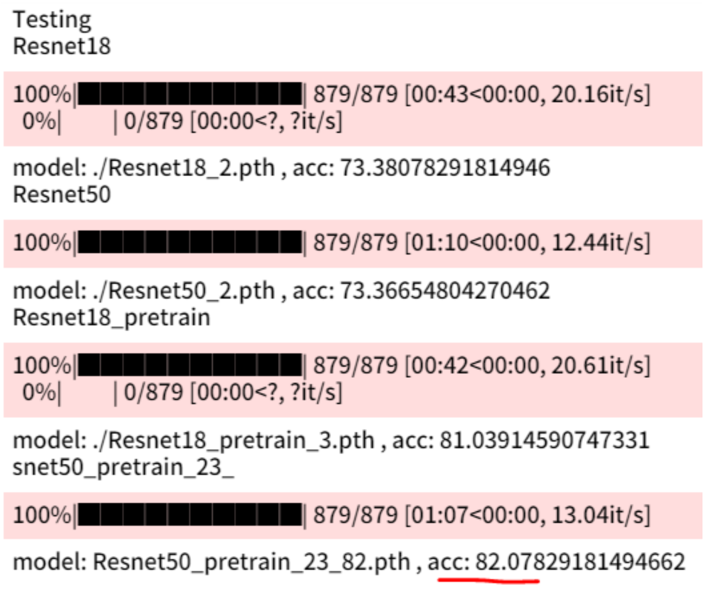
2.4 Hyperparameters

- Batch_size=8
- Epochs = 15 (resnet18), 8 (resnet50)
- Optimizer: SGD, momentum = 0.9, weight_decay = 5e-4, learning_rate=1e-3

3 Experimental results

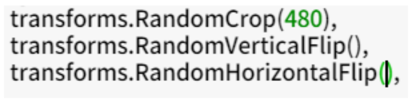
3.1 Highest accuracy

ResNet50 with finetuning pretrain models achieved 82% accuracy in 9 epochs.



```
Testing
Resnet18
100%|██████████| 879/879 [00:43<00:00, 20.16it/s]
  0%|          | 0/879 [00:00<?, ?it/s]
model: ./Resnet18_2.pth , acc: 73.38078291814946
Resnet50
100%|██████████| 879/879 [01:10<00:00, 12.44it/s]
model: ./Resnet50_2.pth , acc: 73.36654804270462
Resnet18_pretrain
100%|██████████| 879/879 [00:42<00:00, 20.61it/s]
  0%|          | 0/879 [00:00<?, ?it/s]
model: ./Resnet18_pretrain_3.pth , acc: 81.03914590747331
snet50_pretrain_23_
100%|██████████| 879/879 [01:07<00:00, 13.04it/s]
model: Resnet50_pretrain_23_82.pth , acc: 82.07829181494662
```

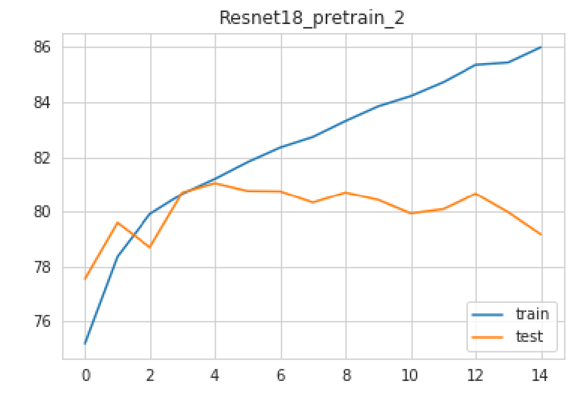
With these kind of transforms.

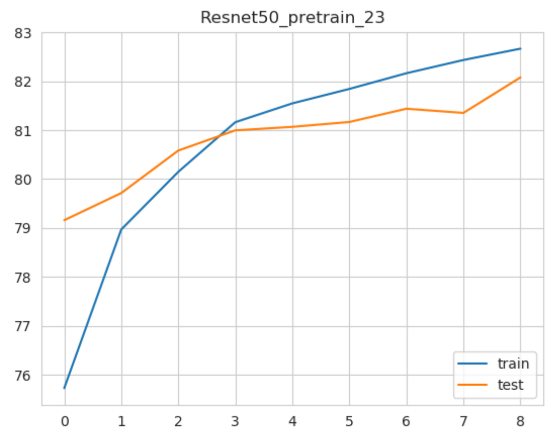


```
transforms.RandomCrop(480),
transforms.RandomVerticalFlip(),
transforms.RandomHorizontalFlip(),
```

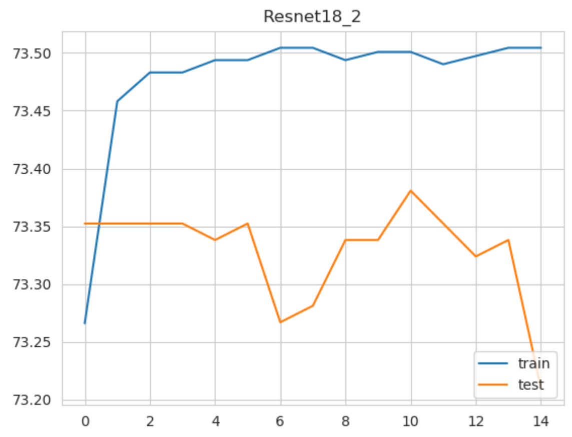
3.2 Result Comparison

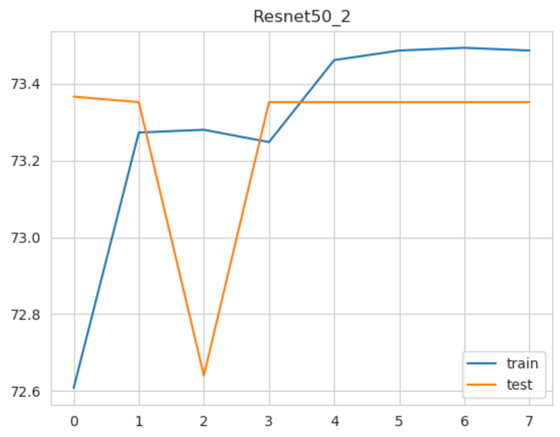
- Finetune Pretrained





- Without Pretrain



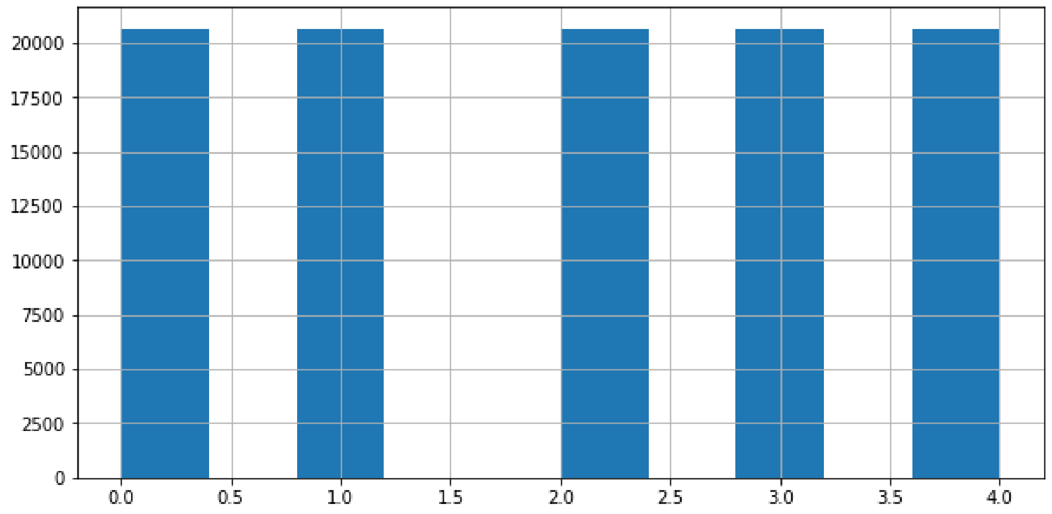


4 Discussion and extra experiments

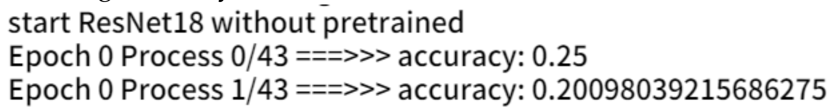
4.1 Balance data

Most of data are labelled 0, which means most of samples have healthy retina (not surprisingly). I tried to balance the number of each label using oversampling, but it takes more time to train, and less accuracy. This result is reasonable because before oversampling, model could always predict "0" and get about 70% accuracy. However, after oversampling, always predict "0" or other random guess could only get about 20% accuracy since there are 5 classes with same numbers. I guess imrpove accuracy from 20% to 82% is too hard to achieve, so oversampling is omitted.

- Data histogram after balanced



- Training accuracy of balanced data



```
start ResNet18 without pretrained
Epoch 0 Process 0/43 ===>>> accuracy: 0.25
Epoch 0 Process 1/43 ===>>> accuracy: 0.20098039215686275
```

4.2 Conclusion

ResNet is a powerful architecture of DNN, but it takes a lot of time to train or even just fine tune. This time-consuming model might be hard to deploy in products. We should consider light-weighted models or try model pruning and neural architecture search techniques.