Lab3

鄭余玄

1 Introduction

In this lab, we are going to implement ResNet-18 and ResNet-50 with and without pretrain. I will explain the details of my implementation in Section 2. In addition, experiment setups, the best accuracy results, and the confusion matrix evaluation are describe in Section 3.

2 Method

In this section, I will introduce the basic of building block of ResNet, follow with the model inputs loader `DataLoader`, the main ResNet architecture, and finally the evaluation of confusion matrix. The pretrained part of this lab is using models from `torchvision` and replace the final fully connected layer to output five categories. The following of this section describes the non-pretrained part.

2.1 Residual Block

The structure of a residual block in shown in Figure 1. The original weight layer $\mathcal{F}(x)$ are added with itself x, which resulted in

$$\mathcal{H}(x) = \mathcal{F}(x) + x = (\mathcal{F} + I)(x)$$

, and its gradient will become

$$\frac{\partial \mathcal{H}}{\partial x} = x\left(I + \frac{\partial \mathcal{F}}{\partial x}\right)$$

. Even if the gradient of $\mathcal{F}$ vanish to 0, the gradient of the residual block $\mathcal{H}$ still holds good property whose gradient is close to an identity matrix. Therefore, using the residual block is a solution of vanishing gradients.

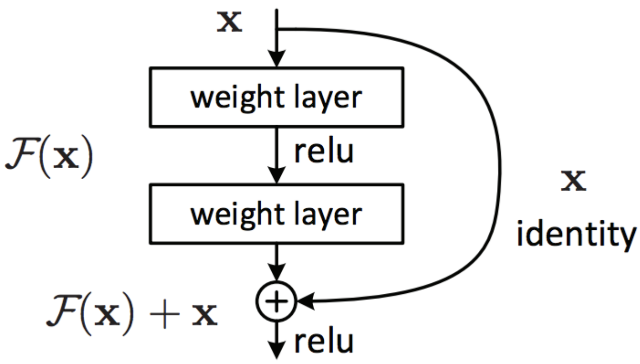


Figure 1: Residual Block

2.2 DataLoader

pytorch provides an interface (`utils.data.DataLoader`) to load data from `utils.data.Dataset`. The `RetinopathyLoader` not only needs to load images and labels, but also performs a series of transforms on the images. The image transforms could be defined using `torchvision`:

```
self.data_transform = {
    'train': transforms.Compose([
        transforms.RandomVerticalFlip(),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
    ]),
    'test': transforms.Compose([
        transforms.ToTensor(),
    ]),
}
```

During the training phase, images are augmented by random flipping; while in the training phase, the images only converts its format from (N, H, W, C) to (N, C, H, W) (`transform.ToTensor()`). In addition, in both phases images are normalized a constant (255) after these transforms.

2.3 ResNet

The overall structure in shown in Figure 2. Note that there are down sampling layers in `conv3_1`, `conv4_1`, and `conv5_1` with stride 2. ResNet-18 uses `BasicBlock` as the residual block while ResNet-50 uses `Bottlenck` with dimension expansion factor of 4 (ex: in `conv5_x`, $512 \times 4 = 2048$). Due to the similarity, the `conv2_x` to `conv5_x` could be written in `ModuleList`:

```
muls = [1] + [block.expansion] * 3
channels = [64, 64, 128, 256, 512]
strides = [1, 2, 2, 2]
self.convs = nn.ModuleList([
    get_conv(in_ * m, out_, layer, stride)
    for m, in_, out_, layer, stride in
        zip(muls, channels[:-1], channels[1:], layers, strides)
])
```

The output dimension is also related the block expansion factor:

```
self.flat_dim = 512 * block.expansion
self.classify = nn.Linear(in_features=self.flat_dim, out_features=5)
```

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 2: ResNet Architecture

2.4 Confusion Matrix

The confusion matrix can be calculated during the evaluation with the package `sklearn`.

```
y_true = np.append(y_true, labels.cpu())
y_pred = np.append(y_pred, predicted.cpu())
```

After casting `torch.Tensor` to `numpy.arrays`, we simply needs to pass these two arrays into the function provided in the documentation of `sklearn`.

3 Result

In all experiments, I use the same setting: `nn.CrossEntropyLoss` as the loss function, and `optim.SGD` as the optimizer (with learning rate $1e-3$, momentum 0.9 and weight decay $5e-4$). The input batch size of ResNet-18 is 128 and ResNet-50 is 64 (limited by the memory).

Table 1 shows the best accuracy during testing.

	without-pretraining	with-pretraining
ResNet-18	73.54%	80.07%
ResNet-50	73.35%	81.14%

Table 1: (Best) Test Accuracy

The command to reproduce my experiments are: (with or without `pretrained` flag)

```
python lab3.py --net ResNet18 --batch_size 64 --weight_decay 0.0005 --pretrained
python lab3.py --net ResNet50 --batch_size 128 --weight_decay 0.0005 --pretrained
```

3.1 Comparison Figure

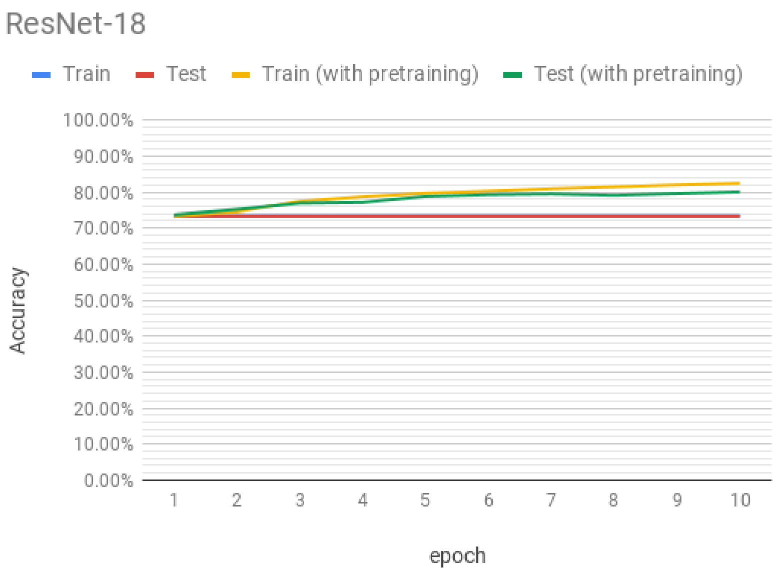
The comparison figure is shown in Figure 3. In both model without pretraining, both training and testing accuracy almost do not increase or decrease around 73%. In contrast, both pretrained models gradually increase their training and testing accuracy.

3.2 Confusion Matrix

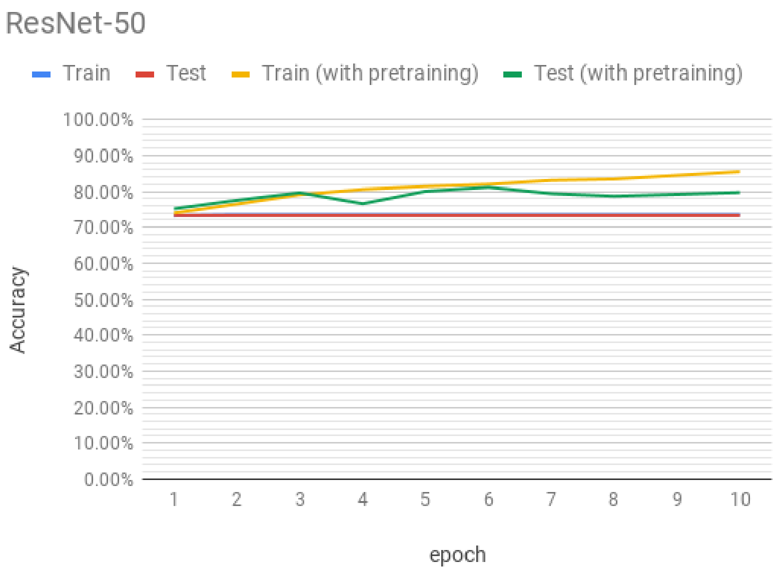
The confusion matrix in the final (10-th) epoch is shown in Figure 4. For both models without pretraining, it directly predicts class 0, and thus can explain the accuracy during training and testing, where class 0 has about 73% of samples shown in Figure 5. On the contrary, both pretrained models can predict the other classes more correctly, and therefore has a reasonable higher accuracy.

4 Discussion

In Figure 5, shows the problem of class imbalance. In addition, about 5% of input images are corrupted (no image, seriously distorted, etc.). Therefore, I think the key point to achieve higher accuracy is in the data augmentation. I think if I had more time, I would try more method to augment the input images.

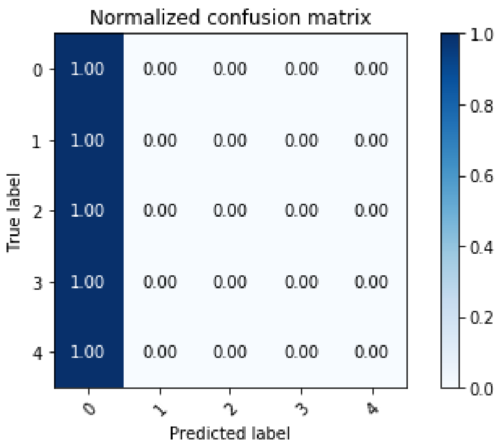


(a) ResNet-18 Accuracy

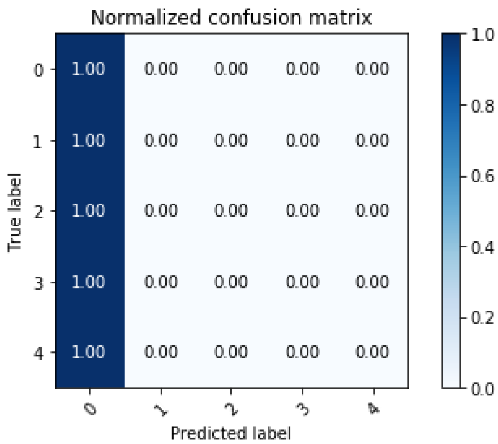


(b) ResNet-50 Accuracy

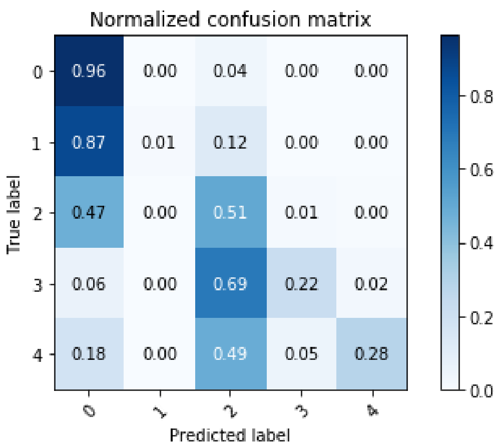
Figure 3: Accuracy



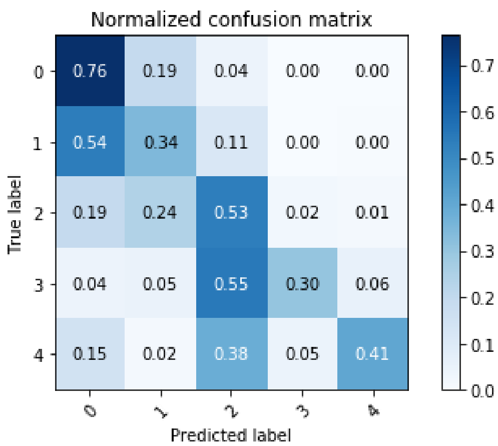
(a) ResNet-18



(b) ResNet-50



(c) ResNet-18 (with pretrained)



(d) ResNet-50 (with pretrained)

Figure 4: Confusion Matrix

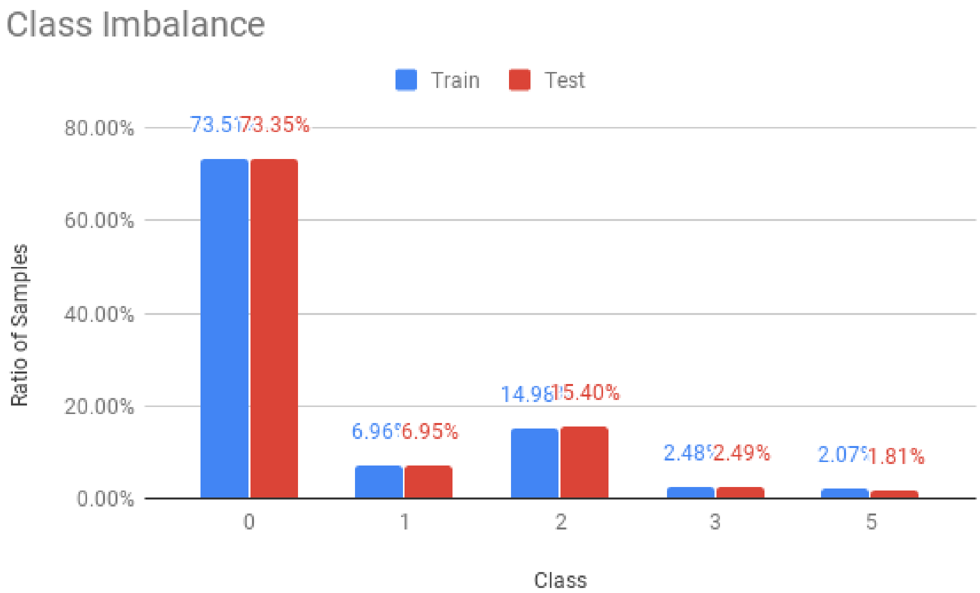


Figure 5: Class Imbalance