DLP HW3

310551053 Chieh-Ming Jiang

July 29th 2021

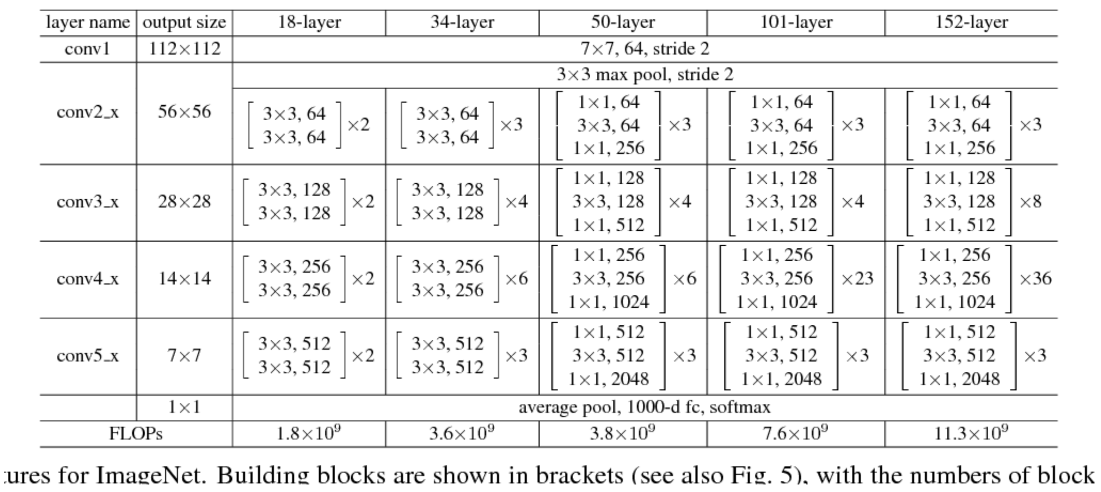
1 Introduction

In this assignment, I am going to implement ResNet with pytorch to analysis diabetic retinopathy dataset. This dataset is from kaggle, which contains 35124 images,including 28,099 training data and 7025 testing data. Last but not least, i will use confusion matrix to show what the model predicts, and what the accuracy is.

Figure 1: Diabetic Retinopathy figure

2 Experiment setups

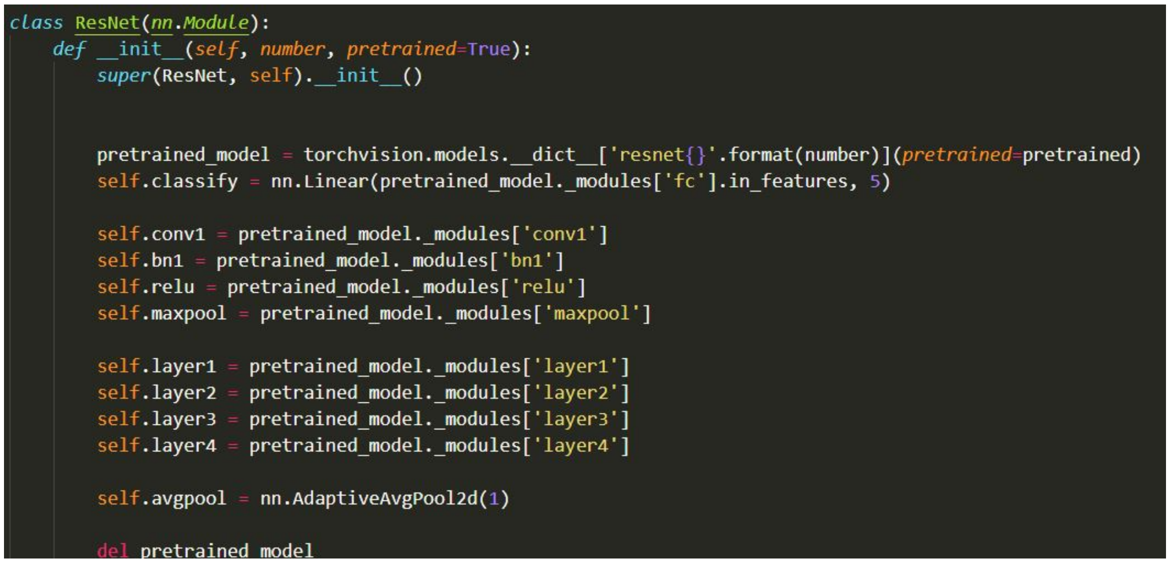
1. Model
   I am using the model of torchvision to construct the ResNet-18 and ResNet-50. As a consequence, it is easy to determine where to use pre-trained parameters. Moreover, because there are five labels in diabetic retinopathy dataset, we have to make the output of last layer the same.



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

ures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of block

Figure 2: ResNet architecture

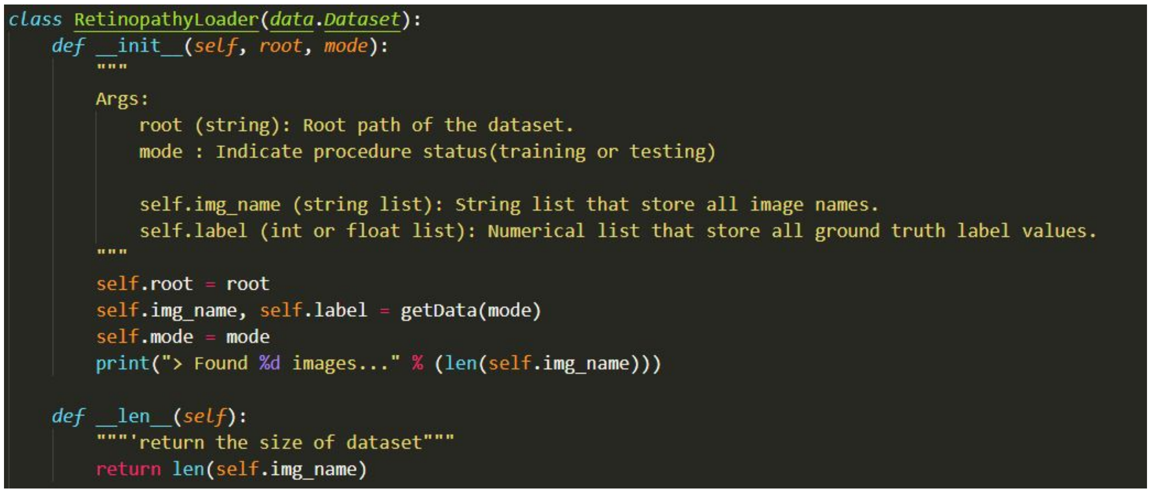


```
class ResNet(nn.Module):
    def __init__(self, number, pretrained=True):
        super(ResNet, self).__init__()


        pretrained_model = torchvision.models.__dict__['resnet{}'.format(number)](pretrained=pretrained)
        self.classify = nn.Linear(pretrained_model._modules['fc'].in_features, 5)

        self.conv1 = pretrained_model._modules['conv1']
        self.bn1 = pretrained_model._modules['bn1']
        self.relu = pretrained_model._modules['relu']
        self.maxpool = pretrained_model._modules['maxpool']

        self.layer1 = pretrained_model._modules['layer1']
        self.layer2 = pretrained_model._modules['layer2']
        self.layer3 = pretrained_model._modules['layer3']
        self.layer4 = pretrained_model._modules['layer4']

        self.avgpool = nn.AdaptiveAvgPool2d(1)

        del pretrained_model
```

Figure 3: ResNet

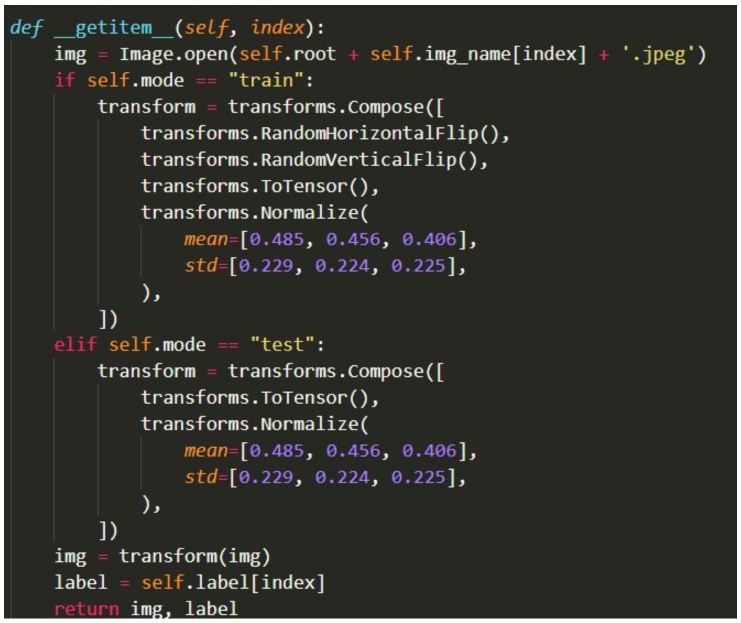
2. Dataloader
   In this case, we need to custom our dataloader for training and testing. As a consequence, we need to write __init__ , __len__ and __getitem__ in the specific class. In __len__ , it returns the length of the whole dataset. In __getitem__ , I will first find the image according to the path and index. Then do some transformation in training data and testing data re-

spectively. In training data, I will do RandomHorizontalFlip() and RandomVerticalFlip() to augment the training data. Then in both training and testing dataset, I use ToTensor() and Normalize(mean=[0.485, 0.456, 0.406],std=[0.229, 0.224, 0.225],) to normalize the data. The value of mean and std are referenced from imagenet, which is a considerable large image dataset.



```python
class RetinopathyLoader(data.Dataset):
    def __init__(self, root, mode):
        """
        Args:
            root (string): Root path of the dataset.
            mode : Indicate procedure status(training or testing)

            self.img_name (string list): String list that store all image names.
            self.label (int or float list): Numerical list that store all ground truth label values.
        """
        self.root = root
        self.img_name, self.label = getData(mode)
        self.mode = mode
        print("> Found %d images..." % (len(self.img_name)))

    def __len__(self):
        """'return the size of dataset"""
        return len(self.img_name)
```

Figure 4: Dataloder-1



```
def __getitem__(self, index):
    img = Image.open(self.root + self.img_name[index] + '.jpeg')
    if self.mode == "train":
        transform = transforms.Compose([
            transforms.RandomHorizontalFlip(),
            transforms.RandomVerticalFlip(),
            transforms.ToTensor(),
            transforms.Normalize(
                mean=[0.485, 0.456, 0.406],
                std=[0.229, 0.224, 0.225],
            ),
        ])
    elif self.mode == "test":
        transform = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize(
                mean=[0.485, 0.456, 0.406],
                std=[0.229, 0.224, 0.225],
            ),
        ])
    img = transform(img)
    label = self.label[index]
    return img, label
```

Figure 5: Dataloder-2

3. Confusion matrix
   The confusion matrix shows the relation of predicted class and ground truth class. In this assignment, I will use normalized confusion matrix.

3 Experimental results

1. Highest testing accuracy

Best testing accuracy		
	ResNet-18	ResNet-50
Pretrained	0.810961	**0.82078**
No pretrained	0.733523	0.733381

2. Comparison figures

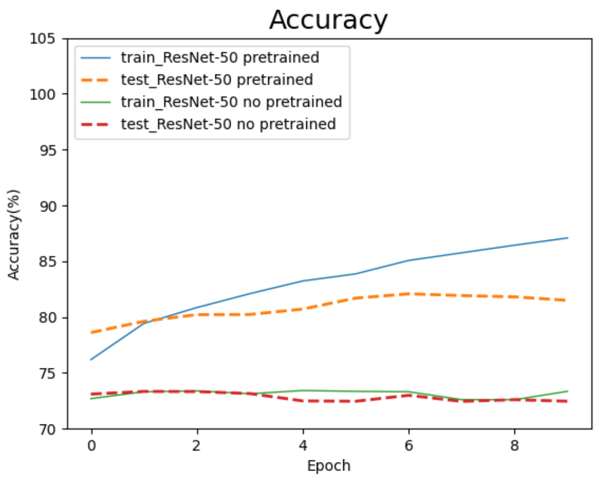


Figure 6: ResNet-50

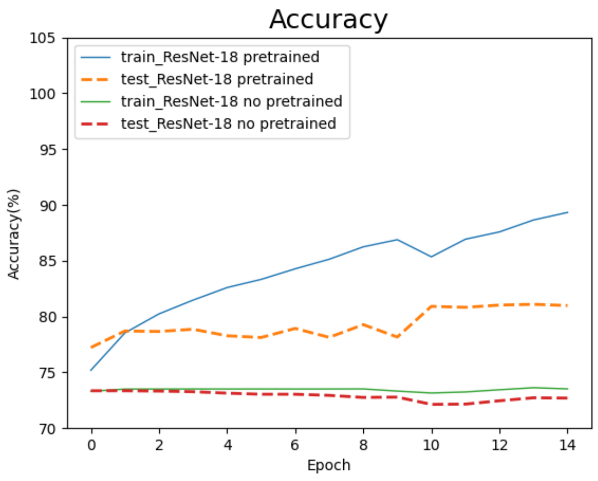


Figure 7: ResNet-18

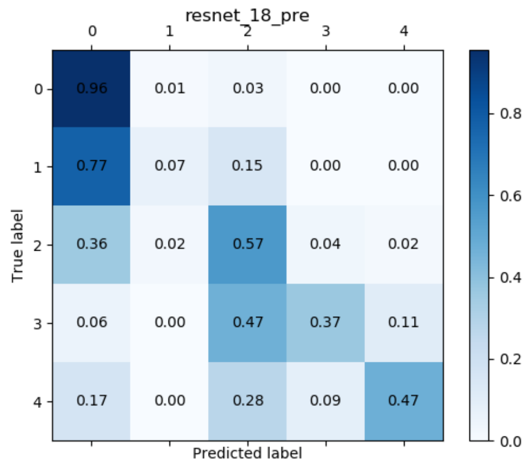


Figure 8: Confusion matrix of ResNet-18

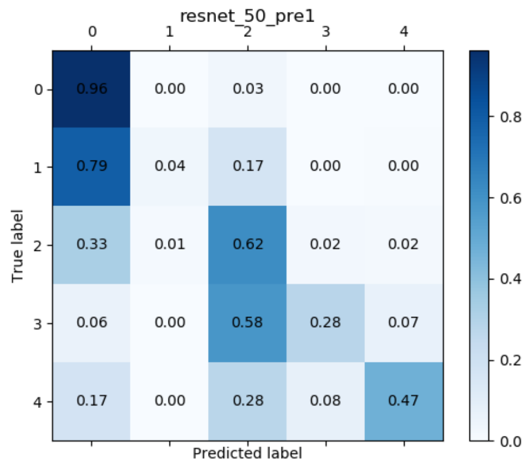


Figure 9: Confusion matrix of ResNet-50

4 Discussion

1. Fine-tune
   Originally, the best accuracy of pretrained ResNet-50 is about 81%, and pretty hard to improve. As a consequence, I saved the model after 10 epochs and adjust the hyper-parameters. I reduced the learning rate from

1e-3 to 1e-4 and increased the weight decay from 5e-4 to 5e-3. Then the accuracy improves to 82%.

2. Class imbalance

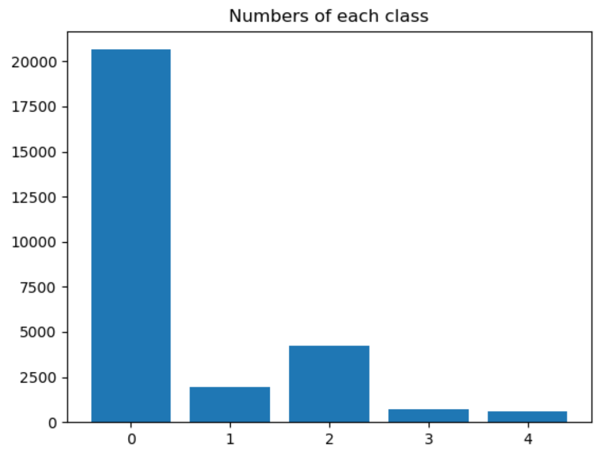
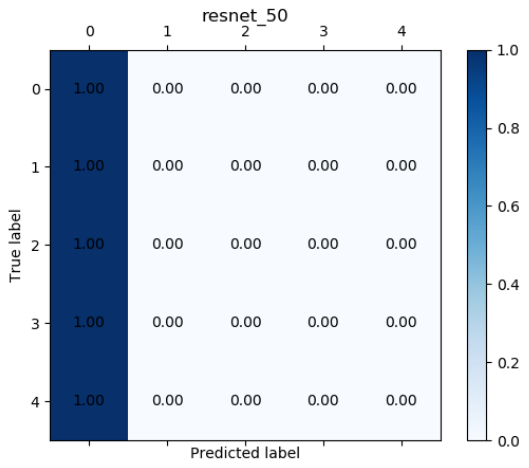


Figure 10: Numbers of each class

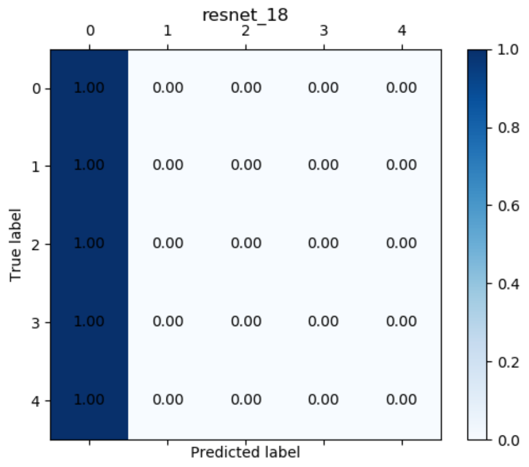
From the confusion matrix above, we could see the model often predicts the class 1 to class 0. I think it results from the images of class 0 are much more than the class 1.

3. Pretrained and not pretrained



True \ Predicted	0	1	2	3	4
0	1.00	0.00	0.00	0.00	0.00
1	1.00	0.00	0.00	0.00	0.00
2	1.00	0.00	0.00	0.00	0.00
3	1.00	0.00	0.00	0.00	0.00
4	1.00	0.00	0.00	0.00	0.00

Figure 11: ResNet-50 not pretrained



True \ Predicted	0	1	2	3	4
0	1.00	0.00	0.00	0.00	0.00
1	1.00	0.00	0.00	0.00	0.00
2	1.00	0.00	0.00	0.00	0.00
3	1.00	0.00	0.00	0.00	0.00
4	1.00	0.00	0.00	0.00	0.00

Figure 12: ResNet-18 not pretrained

From the confusion matrix of non-pretrained models, we could see the models always predict as class 0. However, the accuracy is still high as about 70%. As a consequence, it does not mean a model is good with only high accuracy.