Experiment Report

311552024 詹偉翔

1. Introduction

In this lab, we will implement the image classification that is ResNet18 and ResNet50. Moreover, we will preprocess data by ourselves and using pre-Trained model. To see which models will be the best model with highest accuracy.

2. Experiment Setup

A. The detail of your model

I set the parameters below:

- learning rate = 0.005
- batch size = 12
- epochs = 10

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

This is architecture of ResNet18 & ResNet50.

i. ResNet18

There are 18 layers in this model and it use residual learning, so it called ResNet18. I build this network with the ResNet18 architecture. In my code, there is _make_layer is used to build the blocks that used in conv2 – conv5 which is in previous picture. I get the pretraining weight from pytorch, get their layers, and used in my network.

```python
class ResNet18(nn.Module):

    def __init__(self, image_channels , num_classes , pretrained = False):
        super(ResNet18, self).__init__()

        self.in_channels = 64

        self.pretrained = pretrained
        if  pretrained == True:
            model = models.resnet18(weights=ResNet18_Weights.DEFAULT)
            self.conv1 = getattr(model , 'conv1')
            self.bn1 = getattr(model , 'bn1')
            self.relu = getattr(model , 'relu')
            self.maxpool = getattr(model , 'maxpool')
            self.layer1 = getattr(model , 'layer1')
            self.layer2 = getattr(model , 'layer2')
            self.layer3 = getattr(model , 'layer3')
            self.layer4 = getattr(model , 'layer4')
            self.avgpool = getattr(model , 'avgpool')
        else:
            self.conv1 = nn.Conv2d(image_channels, 64, kernel_size=7, stride=2, padding=3)
            self.bn1 = nn.BatchNorm2d(64)
            self.relu = nn.ReLU()
            self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
            #resnet layers
            self.layer1 = self.__make_layer(64, 64, stride=1)
            self.layer2 = self.__make_layer(64, 128, stride=2)
            self.layer3 = self.__make_layer(128, 256, stride=2)
            self.layer4 = self.__make_layer(256, 512, stride=2)
            self.avgpool = nn.AdaptiveAvgPool2d((1, 1))

        self.fc = nn.Linear(512, num_classes)

    def __make_layer(self, in_channels, out_channels, stride):

        identity_downsample = None
        if stride != 1:
            identity_downsample = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=2, padding=1),
                nn.BatchNorm2d(out_channels)
            )
        return nn.Sequential(
            Block(in_channels, out_channels, identity_downsample=identity_downsample, stride=stride),
            Block(out_channels, out_channels)
        )

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.maxpool(x)

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)

        x = self.avgpool(x)
        x = x.view(x.shape[0], -1)
        x = self.fc(x)
        return x
```

Here is my ResNet18.

```python
class Block(nn.Module):

    def __init__(self, in_channels, out_channels, identity_downsample=None, stride=1):
        super(Block, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU()
        self.identity_downsample = identity_downsample

    def forward(self, x):
        identity = x
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.conv2(x)
        x = self.bn2(x)
        if self.identity_downsample is not None:
            identity = self.identity_downsample(identity)
        x += identity
        x = self.relu(x)
        return x
```

Here is my block.

ii. ResNet50

This is 50 layers in this model and it use residual learning, so it called ResNet50. I build this network with the ResNet18 architecture. In my code, there are the same functions I do in ResNet18, but there are some different in the block function and corresponding number of blocks.

```python
class ResNet50(nn.Module):
    def __init__(self, image_channels = 3 , num_classes = 5  , pretrained = False):
        super(ResNet50, self).__init__()
        self.expansion = 4
        layers = [3, 4, 6, 3]
        self.in_channels = 64

        self.pretrained = pretrained
        if  pretrained == True:
            model = models.resnet50(weights=ResNet50_Weights.DEFAULT)
            self.conv1 = getattr(model , 'conv1')
            self.bn1 = getattr(model , 'bn1')
            self.relu = getattr(model , 'relu')
            self.maxpool = getattr(model , 'maxpool')
            self.layer1 = getattr(model , 'layer1')
            self.layer2 = getattr(model , 'layer2')
            self.layer3 = getattr(model , 'layer3')
            self.layer4 = getattr(model , 'layer4')
            self.avgpool = getattr(model , 'avgpool')
        else:
            self.conv1 = nn.Conv2d(image_channels, 64, kernel_size=7, stride=2, padding=3)
            self.bn1 = nn.BatchNorm2d(64)
            self.relu = nn.ReLU()
            self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

            # ResNetLayers
            self.layer1 = self.__make_layer(layers[0], intermediate_channels=64, stride=1)
            self.layer2 = self.__make_layer(layers[1], intermediate_channels=128, stride=2)
            self.layer3 = self.__make_layer(layers[2], intermediate_channels=256, stride=2)
            self.layer4 = self.__make_layer(layers[3], intermediate_channels=512, stride=2)

            self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512 * self.expansion, num_classes)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.maxpool(x)

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)

        x = self.avgpool(x)
        x = x.view(x.shape[0], -1)
        x = self.fc(x)
        return x

    def __make_layer(self, num_residual_blocks, intermediate_channels, stride):
        layers = []

        identity_downsample = nn.Sequential(nn.Conv2d(self.in_channels, intermediate_channels*self.expansion, kernel_size=1, stride=stride),
                                            nn.BatchNorm2d(intermediate_channels*self.expansion))
        layers.append(Block( self.in_channels, intermediate_channels, identity_downsample, stride))
        self.in_channels = intermediate_channels * self.expansion # 256
        for i in range(num_residual_blocks - 1):
            layers.append(Block( self.in_channels, intermediate_channels)) # 256 -> 64, 64*4 (256) again
        return nn.Sequential(*layers)
```

Here is my ResNet50.

```python
class Block(nn.Module):
    def __init__(self, in_channels, out_channels, identity_downsample=None, stride=1):
        super(Block, self).__init__()
        self.expansion = 4

        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=1, padding=0)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=stride, padding=1)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.conv3 = nn.Conv2d(out_channels, out_channels * self.expansion, kernel_size=1, stride=1, padding=0)
        self.bn3 = nn.BatchNorm2d(out_channels * self.expansion)
        self.relu = nn.ReLU()
        self.identity_downsample = identity_downsample

    def forward(self, x):
        identity = x
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.conv2(x)
        x = self.bn2(x)
        x = self.relu(x)
        x = self.conv3(x)
        x = self.bn3(x)

        if self.identity_downsample is not None:
            identity = self.identity_downsample(identity)

        x += identity
        x = self.relu(x)
        return x
```

Here is my block.

B. The details of your Dataloader

1. we get the image name and label from csv file.
2. Implement the __getitem__ function. We do the preprocessing and resize the image to 512x512 in here.
3. Return our image and its label.

C. Describing your evaluation through the confusion matrix

I store the label and prediction in each list, and then use ConfusionMatrixDisplay.from_prediction() to get the confusion matrix with labels is 0~4 and normalization is true.

```
def cofusion_matrix(net, name , Pre_Train = "No Pretraining"):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    checkpoint = torch.load("./models/{}_{}.pt".format(name,Pre_Train))
    net.load_state_dict(checkpoint['model_state_dict'])
    net.eval()
    Test_Load_IMG = RetinopathyLoader("./new_test/" , 'test')
    test_load = DataLoader(Test_Load_IMG , batch_size=4 ,num_workers=8)
    prediction_list = []
    label_list = []

    for _,(inputs,labels) in enumerate(test_load):
        inputs, labels = inputs.to(device , dtype = torch.float), labels.to(device , dtype = torch.long)
        outputs = net(inputs)
        pred = outputs.argmax(dim = 1)
        pred= pred.to('cpu')
        labels=labels.to('cpu')
        for i in pred:
            prediction_list.append(i)
        for j in labels:
            label_list.append(j)
    disp = ConfusionMatrixDisplay.from_predictions(label_list , prediction_list , labels=[0,1,2,3,4] , normalize='true')
    # disp.plot()
    plt.title(f"{name}_{Pre_Train}_Normalized confusion matrix")
    plt.savefig(f"./results/{name}_{Pre_Train}_confusion matrix.jpg")
    plt.show()
```

Here is my confusion matrix function.

3. Data Preprocessing

A. How you preprocessed your data?

i. I find the boundary of images.

ii. I count each pixel in each line. If all pixel in line is lower than 25, I think it is black and cut it down.

iii. I resize the image to 512x512

B. What makes your method special?

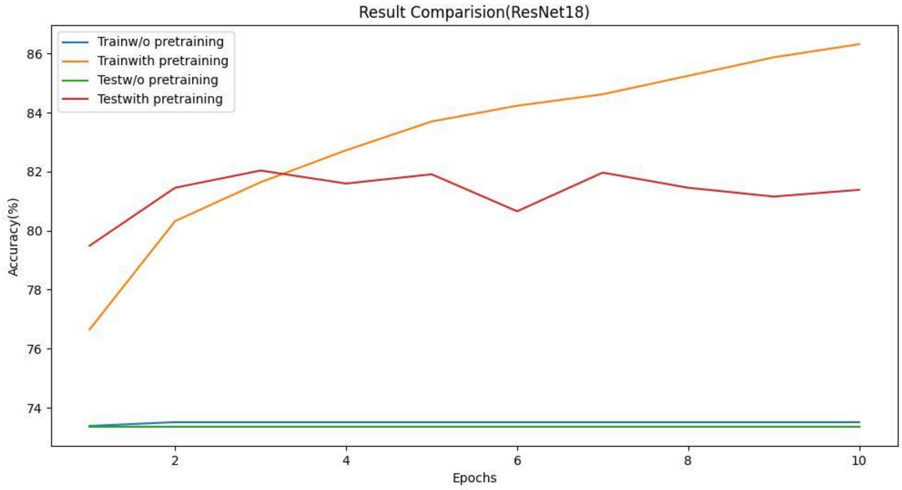
I think the cutting black side is the special way in my method, because the model can only focus on the eye.

4. Experiment Results

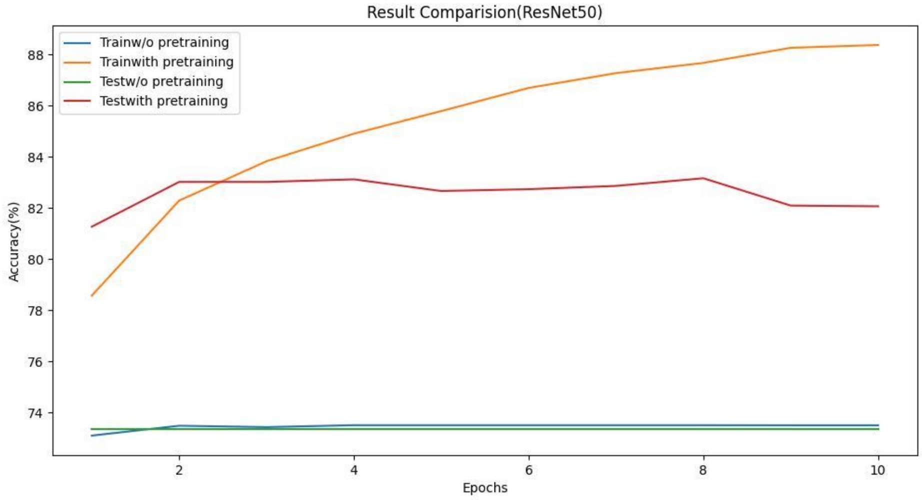
A. The highest testing models

```
ResNet18_Yes Pretraining accuracy is 82.08
ResNet18_No Pretraining accuracy is 73.35
ResNet50_Yes Pretraining accuracy is 82.32
ResNet50_No Pretraining accuracy is 73.30
```

- ResNet18



- ResNet50

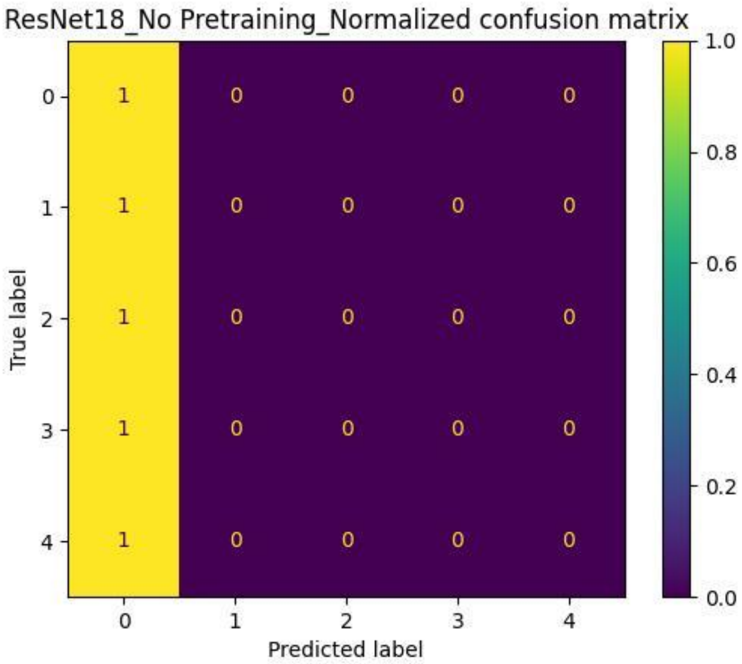


The models without pre-trained will be not able to predict well and can’t be trained better than previous epochs. But there is better performance in pretraining models.

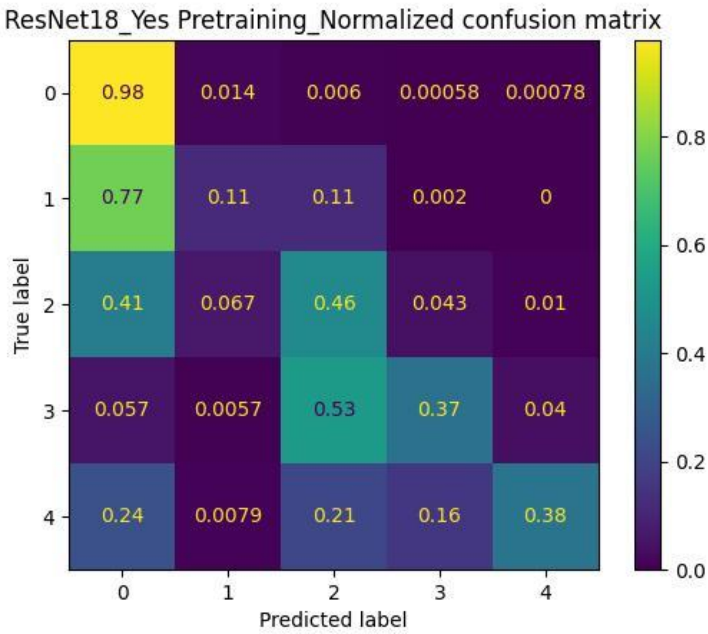
B. Comparison figures

i. ResNet18

- Without Pretraining

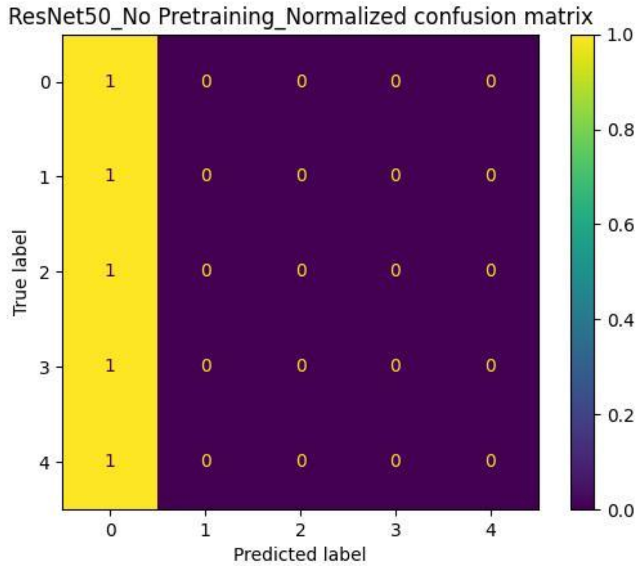


- With Pretraining

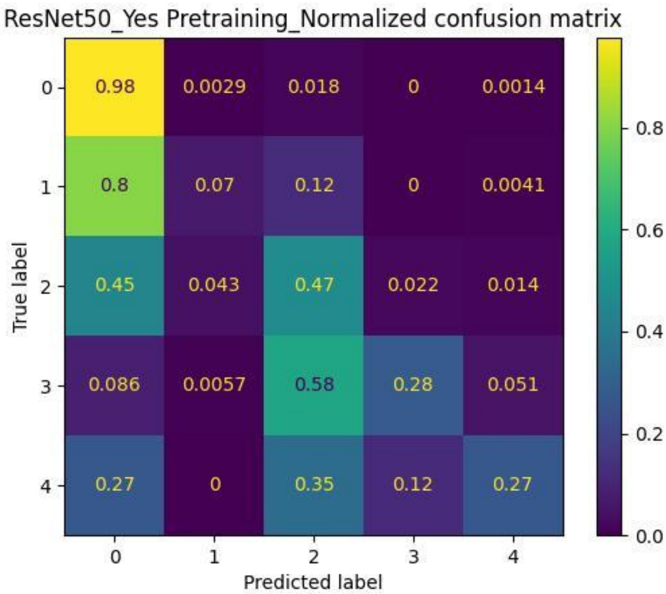


ii. ResNet50

- Without Pretraining



- With Pretraining



5. Discussion

There is a good problem that pretraining model can be trained better and better. But non-pretraining model can't. I think the main reason is imbalance dataset. It can be easy to be found out by confusion matrix. The non-pretraining models are all guessing the answer is 0.