Lab3: Diabetic Retinopathy Detection

ID: 0856601
姓名: 黃兆宇

1. Introduction

In this homework, we need to train the Resnet18 and Resnet50 for a Dataset-Diabetic Retinopathy Detection network, the network can classify the images into five different classes.

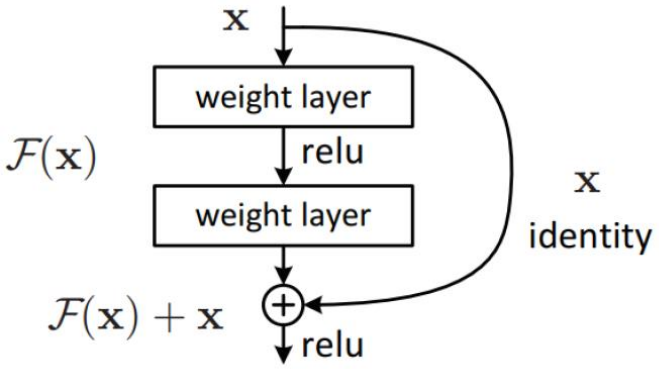
2. Experiment setups

A. The details of model

In this homework, I train Resnet18 and Resnet50 as the classifier, there are different kinds of Resnets that have different layer numbers, following picture show the architecture of the networks, the Resnet18 and Resnet50 is specified by red boxes.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
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

Resnet use short cut connection to solve the problem of gradient vanishing, the short cut connection would pass the data x directly to the output of layer.



Without the short cut connection, the gradient contributed by x value may become 0 during times of derivative between layers, but if adding the x value directly to the output, the x will contribute larger value to the gradient, which can solve the gradient vanishing problem in the training phase.

$$\mathbf{x}_L = \mathbf{x}_l + \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathcal{W}_i),$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\left(1 + \frac{\partial}{\partial \mathbf{x}_l}\sum_{i=l}^{L-1}\mathcal{F}(\mathbf{x}_i, \mathcal{W}_i)\right).$$

B. The details of your Dataloader

There are three functions in the dataloader (RetinopathyLoader), including __init__ 、__len__ 、__getitem__.

```
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

        if self.mode == 'train':
            self.data_transforms = \
            transforms.Compose([
                                transforms.RandomHorizontalFlip(),
                                transforms.ToTensor(),
                                transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
            ])
        else:
            self.data_transforms = \
            transforms.Compose([
                                transforms.ToTensor(),
                                transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
            ])


        print("> Found %d images..." % (len(self.img_name)))
```

The __init__ function initialize the data we need when loading model, including root, img_name, label, mode and transform.

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

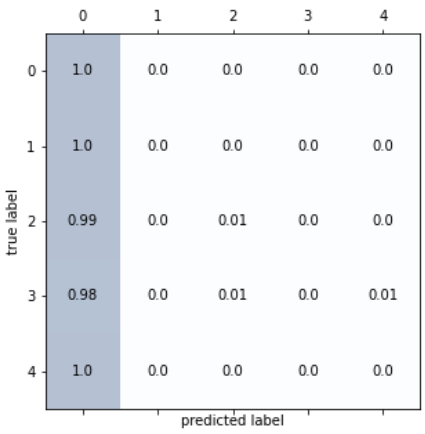
The function __len__ return the size of dataset.

```
def __getitem__(self, index):
    """something you should implement here"""

    """
       step1. Get the image path from 'self.img_name' and load it.
              hint : path = root + self.img_name[index] + '.jpeg'

       step2. Get the ground truth label from self.label

       step3. Transform the .jpeg rgb images during the training phase, such as resizing, random flipping,
              rotation, cropping, normalization etc. But at the beginning, I suggest you follow the hints.

              In the testing phase, if you have a normalization process during the training phase, you only need
              to normalize the data.

              hints : Convert the pixel value to [0, 1]
                      Transpose the image shape from [H, W, C] to [C, H, W]

          step4. Return processed image and label
    """
    # step1.
    path = self.root + self.img_name[index] + '.jpeg'
    img = Image.open(path)

    # step2.
    label = self.label[index]

    # step3. Use the transform.ToTensor() can accomplish two tasks in hint
    # Can also apply more transform on the image.
    img = self.data_transforms(img)

    return img, label
```

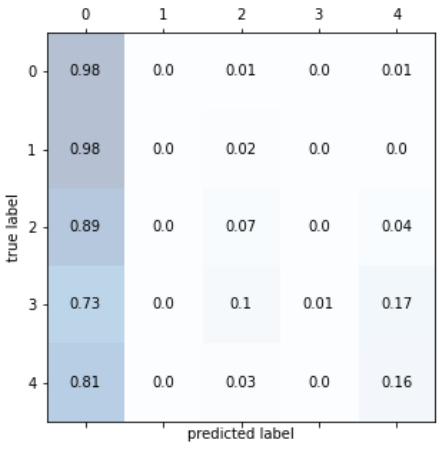
The function __getitem__ will be called when iterating the data through data loader, first it will load the image from the img's directory and implement the image transformation on each image, the transform,ToTensor() function can covert the pixel value to [0, 1] and transpose the image shape from [H, W, C] to [C, H, W].

C. Describing your evaluation through the confusion matrix

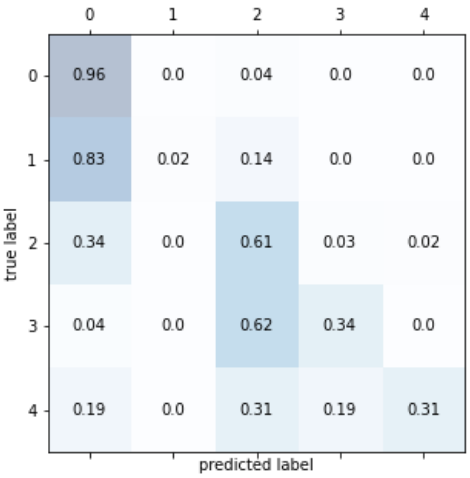
Resnet18 pretrained



Resnet50 pretrained



Resnet50 pretrained with data augmentation and normalization



By observing the confusion matrix, I find out that with data augmentation and normalization, the classification task will work well on classifying different classes, without applying these two transformation, the prediction result will be dominated by the class 0 due to the unbalance data numbers in the training set.

3. Experimental results

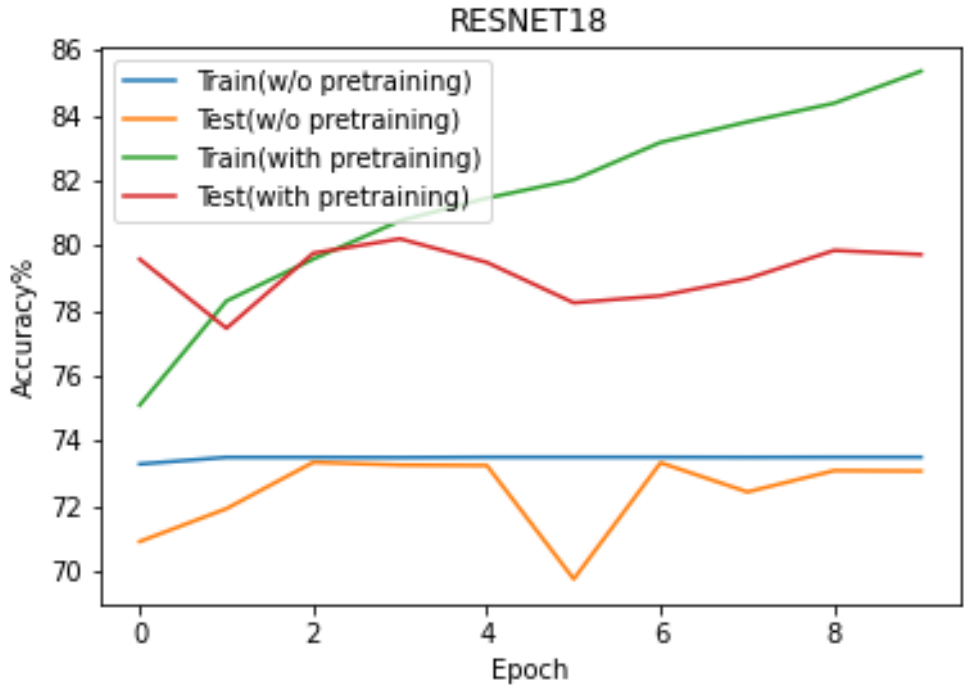
A The highest testing accuracy

The pretrained Resnet50 with data augmentation give the best accuracy:

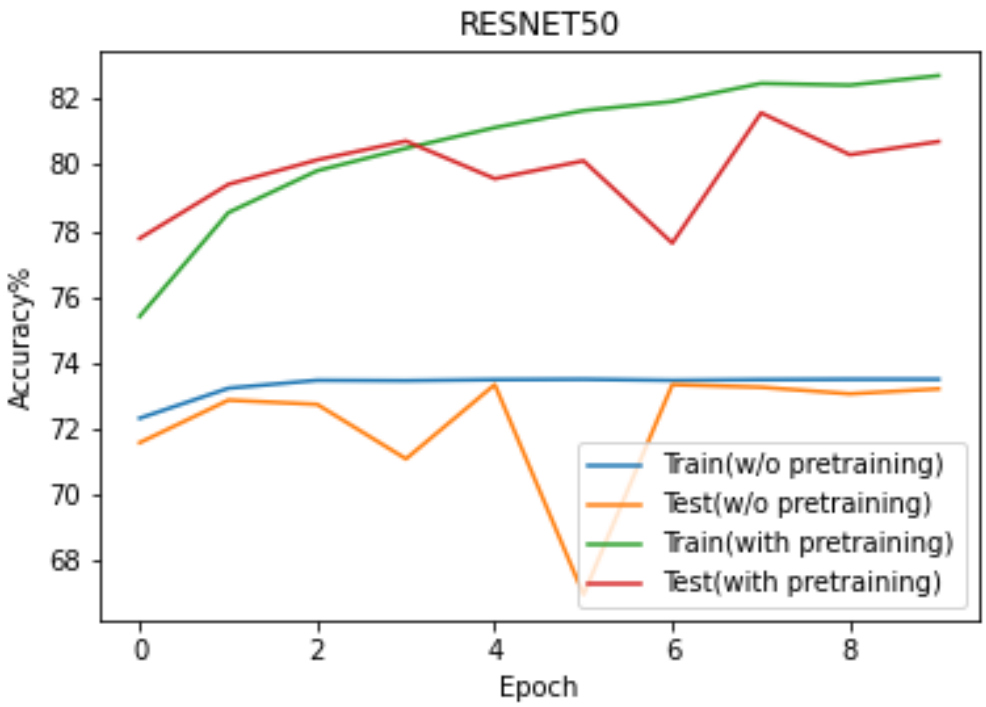
```
Best accuracy for pretrained RESNET50 : 81.58%
```

B. Comparison figures

Resnet18



Resnet50



Hyperparameters I use in this Lab

```
# Hyperparameters
EPOCHS = 10
BATCH_SIZE = 4
LR = 1e-3
MOMENTUM = 0.9
WEIGHT_DECAY = 5e-4
```

To achieve higher accuracy I add the data augmentation and image normalization on the training set to train the resnet50 model, the best accuracy increase from 80.94 to 81.58.

```
if self.mode == 'train':
    self.data_transforms = \
    transforms.Compose([
                        transforms.RandomHorizontalFlip(),
                        transforms.ToTensor(),
                        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
```

4. Discussion

In this homework, I learned how to write the custom data-loader in Pytorch, which make me more familiar with the data type and dimension I feed into the network. I also find out before applying in mine program, the pretrained model is trained on the dataset ImageNet which is different from the Diabetic images. Although the dataset are different, applying pretrained weight can make the accuracy arise a lot, about 7%~9%. By this observation, I suggest that maybe some classification ability of network is not depend on the high level semantic of the images, but depend on low level features like, corners, lines… And according to the experiment, data augmentation improve the classification result in this Lab.