Lab4 Diabetic Retinopathy Detection

1. Introduction

   Use the Resnet 50 and 18 model to detect diabetic retinopathy , and show the confusion matrix .

   model.py : return Resnet 50 , 18 , with , w/o pretrained model .

   train.py : training code .

   test.py : testing code .

   dataLoader.py : my own dataloader .

2. Experiment setups

   A : Details of model:

```python
def resnet(ver,pre=True,frez=False):
    if ver == 50:
        resnet = models.resnet50(pretrained=pre)
    else:
        resnet = models.resnet18(pretrained=pre)

    if frez == True:
        for param in resnet.parameters():
            param.requires_grad = False

    fc_inputs = resnet.fc.in_features
    resnet.fc = nn.Sequential(
        nn.Linear(fc_inputs, 512),
        nn.LeakyReLU(),
        nn.Linear(512, 5)
    )

    return resnet
```

   In model.py , I change the last layer to two fully connection layer and one activation function layer , and you can set the parameters to get the different model .

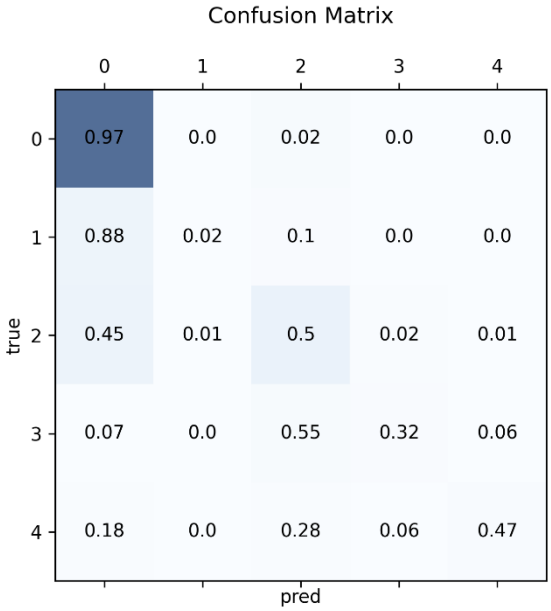
   B : Details of DataLoader :

```python
class dataloader(Dataset):
    def __init__(self, data_path, imagepath, mode='train'):
        self.mode = mode
        self.path = [data_path,imagepath]
        self.img_list = np.array(pd.read_csv(self.path[0]+'/'+self.mode+'_img.csv',header=None))
        self.label_list = np.array(pd.read_csv(self.path[0]+'/'+self.mode+'_label.csv',header=None))
        self.img_name = []
        self.label = []

        for i in range(self.img_list.shape[0]):
            self.img_name.append(self.path[1]+'/'+str(self.img_list[i][0])+'.jpeg')
            self.label.append(int(self.label_list[i][0]))

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

    def __getitem__(self, idx):
        image = cv2.imread(self.img_name[idx])
        label = self.label[idx]
        transform = transforms.Compose([transforms.ToPILImage(),
                                        transforms.Resize((224)),
                                        transforms.RandomHorizontalFlip(),
                                        transforms.RandomVerticalFlip(),
                                        transforms.ToTensor(),
                                        transforms.Normalize(mean=[0.5769, 0.3852, 0.2649],std=[0.1061, 0.0809, 0.0555])])

        image_tensor = transform(image).float()
        sample = {"Image": image_tensor, "label": label}

        return sample
```

In __getitem__ , I use the transforms to prevent overfitting . And it will return the dictionary which includes image and label .

C : Evaluation of confusion matrix

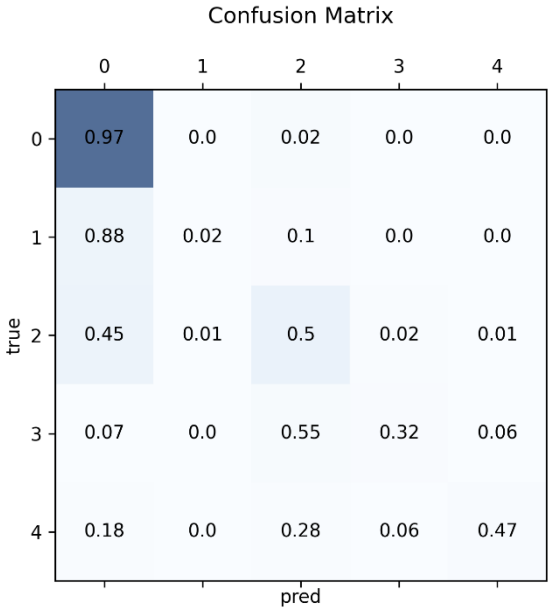
In evaluation of confusion matrix , It can clear present the detection rate between each two classes . So we expect the higher values located on the upper left and lower right diagonal .



3. Experiment results

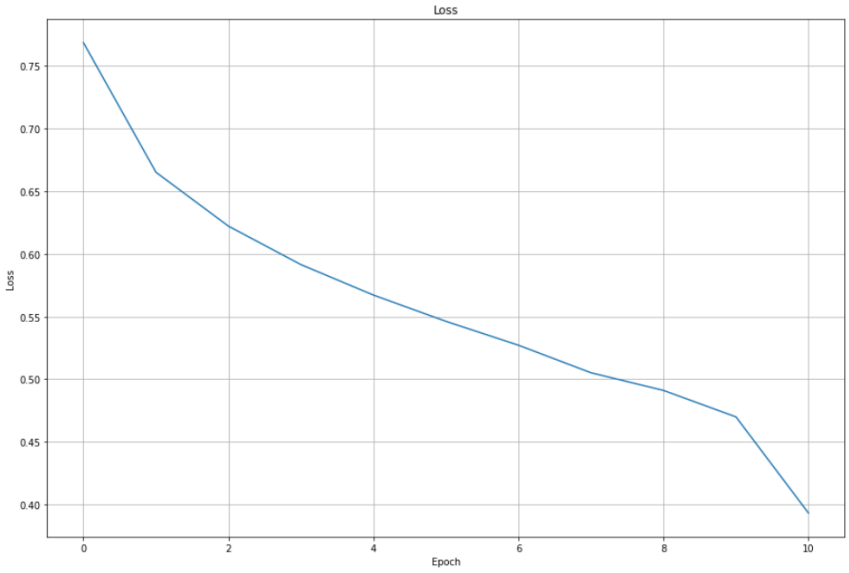
A : The highest testing accuracy : 80 %

I use the Resnet 50 with pretrained weight , batch size=32 , lr=0.001, epoch=11 , weight decay=0.001 , and testing accuracy is 80% .

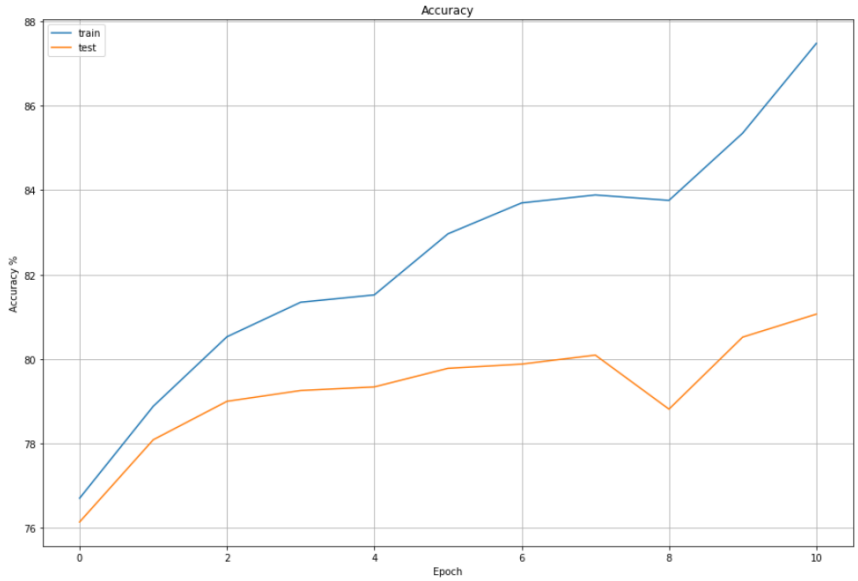


Although the accuracy is 80% , but all classes expect 0 are far below this value , and take a look in training data , the numbers of class 0 images is 20000 , accounting for 71% of the total , so I think it have a overfitting .

Loss curve :

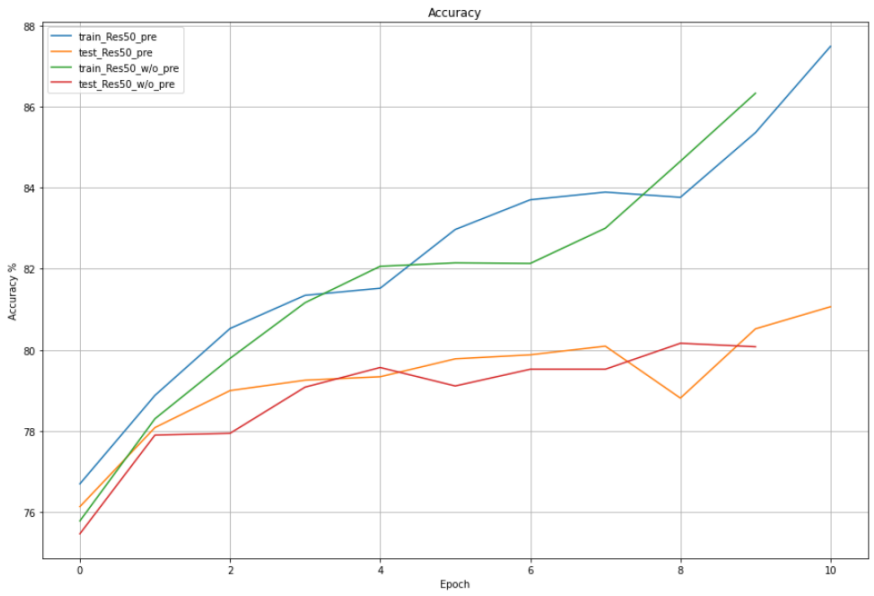


Accuracy curve :



B : Comparsion figures:

The final accuracy is no far between using pretrained weight or not , I think it is because I used pretrained weight without freezing the gradient backward .



4. Discussion

To overcome overfitting , I add the weight loss to the loss function :

```
if args.Loss:
    weights = [1.0, 11, 5, 30, 36]
    class_weights = torch.FloatTensor(weights)
    criterion = nn.CrossEntropyLoss(weight=class_weights).cuda()
else:
    criterion = nn.CrossEntropyLoss().cuda()
```

As a result , although the total accuracy is lower , but the accuracy of each class is better .