Deep Learning

Lab4:Diabetic Retinopathy Detection

311512064 鄧書桓

1. Introduction:

本次實驗是用來分析糖尿病所引發的視網膜病變，藉由編寫自訂義的 dataloader 與利用 pytorch 所提供的 ResNet18,ResNet50 網路架構與 pretrained weight 來實現。此外，還需比較有無使用 pretrain 的差異與繪製 confusion matrix 來判斷他們的好壞。

2. Experiment setups:

A. The detail of your model(ResNet)

Initial model:

```python
def initialize_model(model_name, num_classes, feature_extract, use_pretrained):

    model_ft = None
    # input_size = 0

    if model_name == "resnet18":
        """ Resnet18
        """
        if use_pretrained:
            model_ft = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)
        else:
            model_ft = models.resnet18()
        set_parameter_requires_grad(model_ft, feature_extract)
        num_ftrs = model_ft.fc.in_features
        model_ft.fc = nn.Linear(num_ftrs, num_classes)
        # input_size = 521

    elif model_name == "resnet50":
        """ Resnet50
        """
        if use_pretrained:
            model_ft = models.resnet50(weights=models.ResNet50_Weights.DEFAULT)
        else:
            model_ft = models.resnet50()
        set_parameter_requires_grad(model_ft, feature_extract)
        num_ftrs = model_ft.fc.in_features
        model_ft.fc = nn.Linear(num_ftrs, num_classes)
        # input_size = 521

    return model_ft
```

```python
def initialize_model(model_name, num_classes, feature_extract, use_pretrained):

    model_ft = None
    # input_size = 0

    if model_name == "resnet18":
        """ Resnet18
        """
        if use_pretrained:
            model_ft = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)
        else:
            model_ft = models.resnet18()
        set_parameter_requires_grad(model_ft, feature_extract)
        num_ftrs = model_ft.fc.in_features
        model_ft.fc = nn.Linear(num_ftrs, num_classes)
        # input_size = 521

    elif model_name == "resnet50":
        """ Resnet50
        """
        if use_pretrained:
            model_ft = models.resnet50(weights=models.ResNet50_Weights.DEFAULT)
        else:
            model_ft = models.resnet50()
        set_parameter_requires_grad(model_ft, feature_extract)
        num_ftrs = model_ft.fc.in_features
        model_ft.fc = nn.Linear(num_ftrs, num_classes)
        # input_size = 521

    return model_ft
```

這邊使用 pytorch 所提供的 resnet18 與 resnet50 網路架構來實現，並沒有自行實作出各層的網路架構。另外，這邊使用都使用.DEFAULT 來當作預訓練的權重版本，由於其是在 ImageNet 數據集上的大量圖像做訓練，我認為較符合此作業的應用範疇。此外，還有像是.IMAGENET1K_V1(僅使用 1000 個圖像類別)、.IMAGENET1K_V2(與 V1 相比有更多類別)、SELFIE(使用自拍數據集，用於臉部辨識與分析)和.CIFAR10(使用 CIFAR-10 數據集進行訓練的，用於圖像分類應用)，但他們訓練出來的結果都比不上 default，因此最終沒有使用它們。

Train:

```python
def train_model(model_name, model, dataloaders, dataloaders_eval, optimizer, criterion, num_epochs=5, phase='train'):
    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0
    train_acc_history = []
    eval_acc_history = []

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch+1, num_epochs ))
        print('-' * 10)

        # Each epoch has a training and validation phase
        if phase == 'train':
            model.train()  # Set model to training mode
        else:
            model.eval()   # Set model to evaluate mode or test mode

        running_loss = 0.0
        running_corrects = 0

        for inputs, labels in tqdm(dataloaders, desc='Epoch (train) %d' % (epoch + 1)):
            inputs = inputs.to(device)
            labels = labels.to(device)

            # zero the parameter gradients
            optimizer.zero_grad()

            # forward
            # track history if only in train
            with torch.set_grad_enabled(phase == 'train'):
```

```python
                outputs = model(inputs)
                loss = criterion(outputs, labels)

                _, preds = torch.max(outputs, 1)

                # backward + optimize only if in training phase
                if phase == 'train':
                    loss.backward()
                    optimizer.step()

            # statistics
            running_loss += loss.item() * inputs.size(0)
            running_corrects += torch.sum(preds == labels.data)

        epoch_loss = running_loss / len(dataloaders.dataset)
        epoch_acc = running_corrects.double() / len(dataloaders.dataset)
        print("acc = :", epoch_acc)
        print('{} Loss: {:.4f} Acc: {:.4f}'.format(
            phase, epoch_loss, epoch_acc))

        # evaluaing test data
        eval_accurarcy = evaluate(model, dataloaders_eval, epoch)
        eval_acc_history.append(eval_accurarcy)

        # deep copy the model
        if phase == 'train' and epoch_acc > best_acc:
            best_acc = epoch_acc
            best_model_wts = copy.deepcopy(model.state_dict())
```

```python
            #     train_acc_history.append(epoch_acc)
            # elif phase == 'test':
            #     train_acc_history.append(epoch_acc)
            train_acc_history.append(epoch_acc)
    model.load_state_dict(best_model_wts)

    if use_pretrained == True:
        torch.save(model.state_dict(), 'pretrain_' + model_name + '_train.pt')
        return model, train_acc_history, eval_acc_history
    else:
        torch.save(model.state_dict(), model_name + '_train.pt')
        return model, train_acc_history, eval_acc_history
```

B. The details of your Dataloader

```python
def __init__(self, root, mode, transform = False):
    """
    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

    # set the transform
    self.transform = transform

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

在__init__中取的 images 所在的 folder，並讀取得到的 transform。

```python
if torch.is_tensor(index):
    index = index.tolist()

path = os.path.join(self.root, self.img_name[index] + '.jpeg')
img = Image.open(path)

min_size = img.size[0] if img.size[0]<img.size[1] else img.size[1]

if self.transform :
    transforms_pre = transforms.Compose([transforms.CenterCrop(min_size),
                                         transforms.Resize((512, 512)),
                                         transforms.RandomHorizontalFlip(p = 0.5),
                                         transforms.RandomVerticalFlip(p = 0.5),
                                         transforms.RandomRotation(degrees = 10),
                                         transforms.ToTensor()])
    img = transforms_pre(img)
# vutils.save_image(img, './test/new_example.jpg')
else:
    transforms_pre = transforms.Compose([transforms.CenterCrop(min_size),
                                         transforms.Resize((512, 512)),
                                         transforms.ToTensor()])
    img = transforms_pre(img)

label = self.label[index]
return img, label
```

__getitem__會根據 index 讀取出對應的照片並透過初始化的擴充方式將資料擴充，並透過 PIL 讀取轉成 tensor 的形式，最後回傳轉換過後的 image 及其對應的 label。

C. Describing your evaluation through the confusion matrix

```python
def create_confusionmatrix(model, dataloaders):
    y_pred = []
    y_true = []
    model.to(device)
    model.eval()
    with torch.no_grad():
        for images, labels in tqdm(dataloaders, desc="create coonfusion matrix"):
            images = images.to(device)
            labels = labels.to(device)
            outputs = model(images)
            _, preds = torch.max(outputs, 1)

            # .view(-1)展平成一維張量
            # .detch()為斷開張量與計算圖的連接, 用來避免該張量後續進行反向傳播(去梯度操作)
            # confusion matrix 計算是使用numpy計算(使用cpu)
            y_pred.extend(preds.view(-1).detach().cpu().numpy())
            y_true.extend(labels.view(-1).detach().cpu().numpy())
    cf_matrix = confusion_matrix(y_true, y_pred)
    cf_matrix_norm = confusion_matrix(y_true, y_pred, normalize='true')
    return cf_matrix, cf_matrix_norm
```

一開始將預測結果與真實的 label 存成一個 Skylearn 將結果算

成混淆矩陣並 normalize，再利用 seaborn 與 pandas 等套件將

其繪製成圖表，圖上的數字分別代表該類數量與準確率。

```
def plot_confusion_matrix(cf_matrix, name):
    class_names = ['no DR', 'Mild', 'Moderate', 'Severe', 'Proliferative DR']
    df_cm = pd.DataFrame(cf_matrix, class_names, class_names)
    sns.heatmap(df_cm, annot=True, cmap='Oranges')
    plt.title(name)
    plt.xlabel("prediction")
    plt.ylabel("laTbel (ground truth)")
    plt.savefig('Confusion_matrix' + name + '.png')
    plt.clf()
```

3. Data Preprocessing

```
img_name = np.squeeze(pd.read_csv('test_img.csv'))
reolution_ft = 768

for i in tqdm(range(len(img_name))):
    path = os.path.join("./dataset/new_test", img_name[i] + '.jpeg')

    img = cv2.imread(path)
    min_size = img.shape[0] if img.shape[0]<img.shape[1] else img.shape[1]
    scale = reolution_ft/min_size


    width = int(img.shape[1] * scale)
    height = int(img.shape[0] * scale)
    dim = (width, height)
    resized_img = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
    cv2.imwrite('./dataset/test_resize/' + img_name[i] + '.jpeg', resized_img)
```

由於原始資料的解析度太高了(都 2000 起跳)，會導致在 dataloader 讀取資時花費太多時間，因此這邊在做任何 transforms 前先降維，以此來提高訓練的時間。

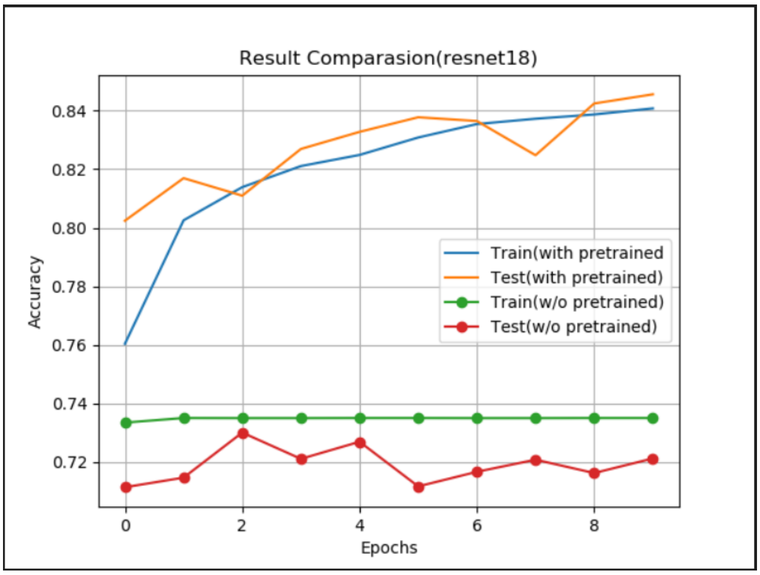
```
min_size = img.size[0] if img.size[0]<img.size[1] else img.size[1]

if self.transform :
    transforms_pre = transforms.Compose([transforms.CenterCrop(min_size),
                                         transforms.Resize((512, 512)),
                                         transforms.RandomHorizontalFlip(p = 0.5),
                                         transforms.RandomVerticalFlip(p = 0.5),
                                         transforms.RandomRotation(degrees = 10),
                                         transforms.ToTensor()])
    img = transforms_pre(img)
```

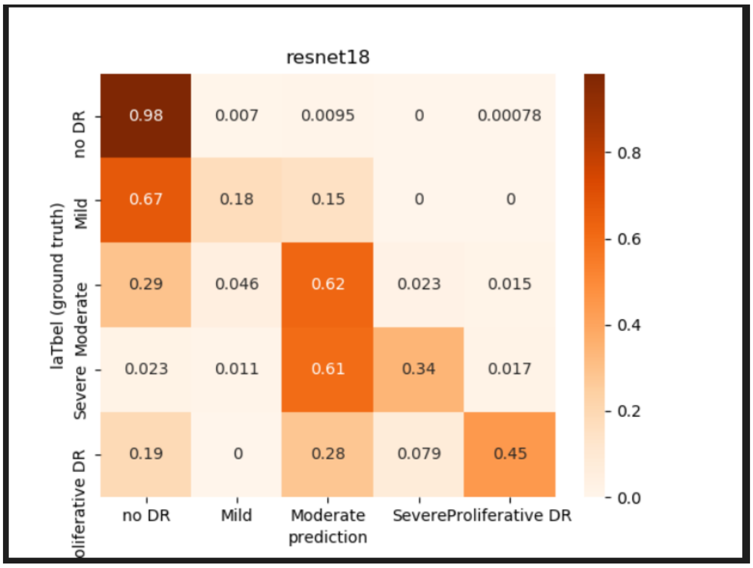
接著，由於 resnet 的輸入維 512*512，且資料集的眼球皆為圖像正中心，因此這邊根據最小的長或寬來做中間擷取，獲得正方形的過高解析度圖，接著做 resize，以此來避免原圖的失真，最後使用 augmentation 來強化原始的數據集，這邊使用隨機水平、垂直翻轉與整個選轉，以此來達成上述目的。

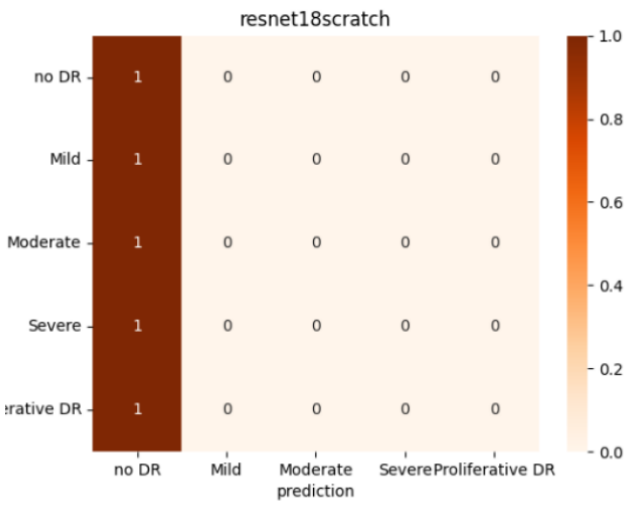
。

4. Experimental results

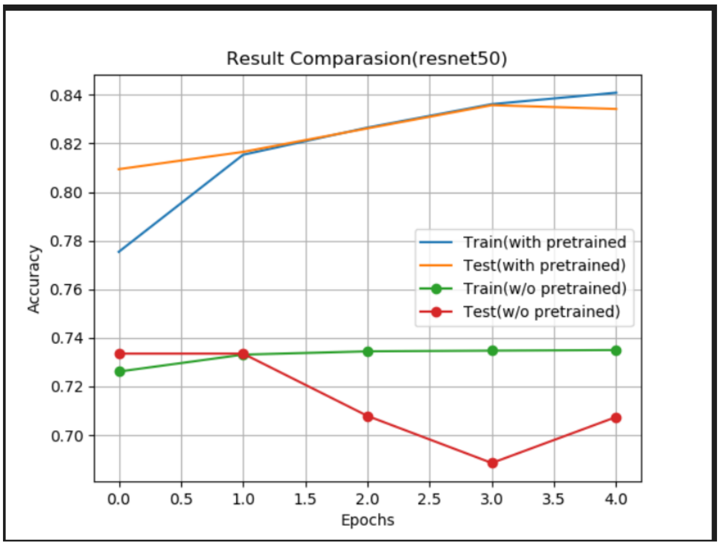
ResNet18:

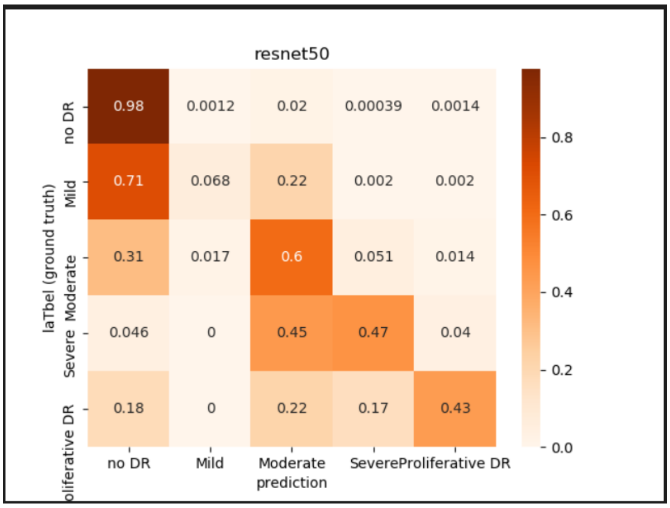


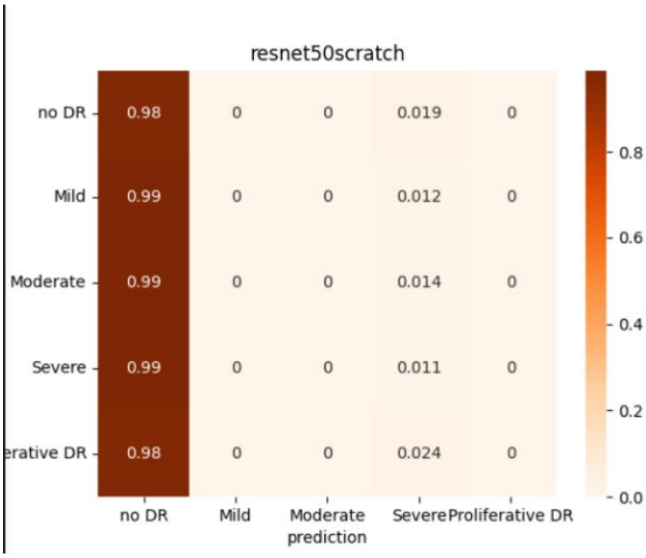

resnet18
laTbel (ground truth)
no DR
Mild
Moderate
Severe
oliferative DR
0.98
0.007
0.0095
0
0.00078
0.67
0.18
0.15
0
0
0.29
0.046
0.62
0.023
0.015
0.023
0.011
0.61
0.34
0.017
0.19
0
0.28
0.079
0.45
no DR
Mild
Moderate
SevereProliferative DR
prediction
0.8
0.6
0.4
0.2
0.0


resnet18scratch
no DR
Mild
Moderate
Severe
erative DR
1
0
0
0
0
1
0
0
0
0
1
0
0
0
0
1
0
0
0
0
1
0
0
0
0
no DR
Mild
Moderate
SevereProliferative DR
prediction
1.0
0.8
0.6
0.4
0.2
0.0

Resnet50:


Result Comparasion(resnet50)
Accuracy
0.84
0.82
0.80
0.78
0.76
0.74
0.72
0.70
Train(with pretrained
Test(with pretrained)
Train(w/o pretrained)
Test(w/o pretrained)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Epochs





由上述的圖可知，由於 epoch 數的不同(ResNet18 為 10、ResNet50 為 5)，ResNet18 的表現比 ResNet50 的表現比較好，若 epoch 數相同的話，照理來說是 Resnet50 會較好，這是因為其參數與層數都較多，會使的模型對於高維度的 input 有更強的理解力。而有使用預訓練的成效很明顯優於沒有使用預訓練的成效，我認為是因為此 dataset 有一大部分為 label0，會導致模型只要輸出 0 即可完成預測，無法學到每個 label 各自不同的特徵。

除此之外，我還有使用過 class_weight 與 sample_weight 來解決 dataset 分布不均的問題，但是這樣反而會導致訓練出來的結果較差，不確定是什麼原因。

5. Discussion

這次的實驗還滿有趣的，體驗了如何自製 dataloader 與 finetune 其他人的 network，尤其是後者，讓我可以在不重建其他人的 model 與收集過多 dataset 的情況下，將其 network 應用在我的場合中，大大地提高 model 的使用效率。另外，在做此功課時，由於 dataset 過於龐大，若 batch size 不小心選太大，會導致顯卡的記憶體不構，進而使訓練失敗，因此還需多加注意。