lab4 report

學號：311512049 姓名：陳緯翰

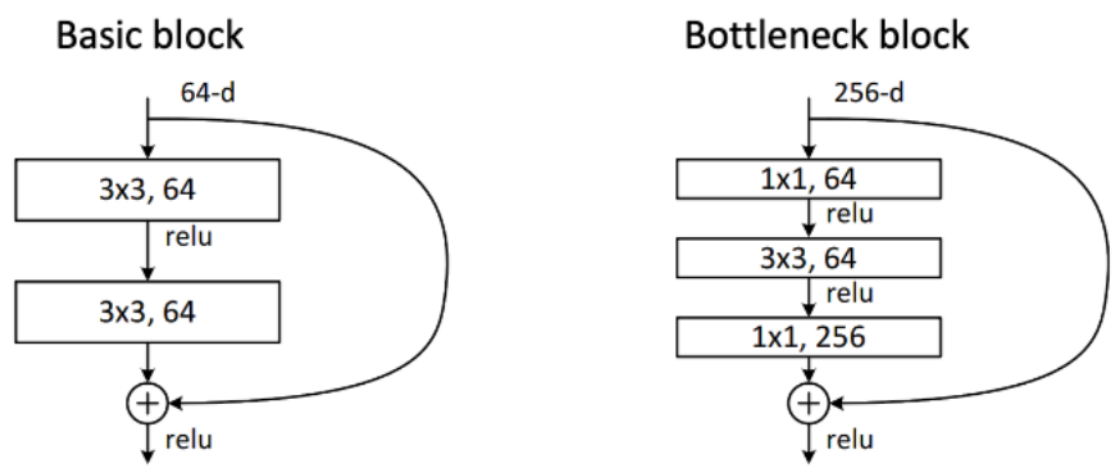
Introduction (20%)

- 分析糖尿病所引發視網膜病變
- 編寫自定義的 dataloader
- 實踐 ResNet18,ResNet50 等網路架構，並 pretrained
- 比較有無 pretrain 並可視化準確率
- 繪製混淆矩陣
- 這次資料集依照危險程度被分成五類

Experiment setups (30%)

a. The details of your model (ResNet)

ReseNet 有兩種建立方式，一種是用 pytorch 刻出建立其結構，另一種是從 torchvision 裡面的 model 這個模組將 ResNet 抓出來做使用，在下面我介紹我的 ResNet 建立方式首先先建立一個 block，分別有兩種建立方式分別是 bottleneck block 和basic block，而ResNet18所使用的是basic block，Resnet50所使用的是bottleneck block，而 BottleNeck Block 的好處是所需參數比 basic block 少但卻能達到一樣的結果。



以下是我的Resnet18和Resnet50的架構：

```
class Basicblock(nn.Module):
    expansion = 1

    def __init__(self, in_channel, channel, stride=1):
        super(Basicblock, self).__init__()
        self.conv1 = nn.Conv2d(in_channel, channel, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(channel)
        self.conv2 = nn.Conv2d(channel, channel, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(channel)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channel != self.expansion*channel:
            self.shortcut = nn.Sequential(
```

```
                nn.Conv2d(in_channel, self.expansion*channel, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion*channel)
            )

    def forward(self, x):
        out = nn.ReLU()(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = nn.ReLU()(out)
        return out

class Bottleneck(nn.Module):
    expansion = 4

    def __init__(self, in_channel, channel, stride=1):
        super(Bottleneck, self).__init__()
        self.conv1 = nn.Conv2d(in_channel, channel, kernel_size=1, bias=False)
        self.bn1 = nn.BatchNorm2d(channel)
        self.conv2 = nn.Conv2d(channel, channel, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(channel)
        self.conv3 = nn.Conv2d(channel, self.expansion*channel, kernel_size=1, bias=False)
        self.bn3 = nn.BatchNorm2d(self.expansion*channel)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channel != self.expansion*channel:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channel, self.expansion*channel, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion*channel)
            )

    def forward(self, x):
        out = nn.ReLU()(self.bn1(self.conv1(x)))
        out = nn.ReLU()(self.bn2(self.conv2(out)))
        out = self.bn3(self.conv3(out))
        out += self.shortcut(x)
        out = nn.ReLU()(out)
        return out

class ResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=4):
        super(ResNet, self).__init__()
        self.in_channel = 64

        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
        self.classify = nn.Sequential(
                nn.AdaptiveAvgPool2d((1, 1)),
                nn.Flatten(),
                nn.Linear(in_features=512 * block.expansion, out_features=50),
                nn.ReLU(inplace=True),
                nn.Dropout(p=0.25),
                nn.Linear(in_features=50, out_features=5))
        # self.linear = nn.Linear(512*block.expansion, num_classes)

    def _make_layer(self, block, channel, num_blocks, stride):
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_channel, channel, stride))
            self.in_channel = channel * block.expansion
        return nn.Sequential(*layers)

    def forward(self, x):
        out = nn.ReLU()(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = self.classify(out)
        return out

def ResNet18():
    return ResNet(Basicblock, [2,2,2,2])

def ResNet50():
    return ResNet(Bottleneck, [3,4,6,3])
```

Data Preprocessing (20%)

a. How you preprocessed your data?

為了讓訓練中batch size能較大，我有寫一個將原版相片先做resize的程式而且並不會影響照片的比例， 為的是使得整體訓練速度變快。

- dataset_preparasion.py如下

  由於我們這次所吃的資料是彩色圖片，檔案的 size 相當大， 因此我使用dataset_preparasion.py 這個檔案將原本長寬皆3000多個 pixel 的原圖片先轉成比例相當但是只有768 pixel 的圖片，再交給助教所給的 dataloader.py 來做裁切跟 resize 成 512 。此舉有些許降低訓練時間 ，程式如下：

```
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("./data/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 = resolution_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('./data/test_resize/' + img_name[i] + '.jpeg', resized_img)
```

- dataloader.py如下：

  Dataloader 會在__init__ function 中取得 images 所在的 folder，並讀取得到的 augmentation 方法做資料擴充，並印出數量，在__getitem__函示則會根據取得的 index 取出對應的照片並透過初始化的擴充方式將資料擴充後，透過 PIL 讀取並轉成對應的 tensor，最後回傳轉換過後的 image 以及其對應的 label

```
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)))

        # crop the image in center
        self.center_crop = transforms.CenterCrop(512)
        self.to_tensor = transforms.ToTensor() # transforms.ToTensor()，它将输入数据转换为张量形式。这个操作会将像素值从0-255的范围映射到0-1之

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

    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:get the image path use os
          path = os.path.join(self.root, self.img_name[index] + '.jpeg')

          # step2:get the ground truth label from  self.label
          label = self.label[index]

          # step3:transform the .jpeg rgb images
          img = Image.open(path)

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

          # img = self.center_crop(img)
          # img = self.to_tensor(img)
          # print(img.shape)
          return img, label



  # print the np.squeeze(img.values), np.squeeze(label.values)
  img, label = getData('train')
  print(img.shape, label.shape)

  img = pd.read_csv('train_img.csv')
  label = pd.read_csv('train_label.csv')
  print(np.squeeze(img.values))
  print(np.squeeze(label.values))
```

b. how to train and test

我是先做training的動作，在完成每個epoch的同時會將每個epoch的weight接存下來，在training完之後再利用 **model.load_state_dict**將model weight匯入去型model evaluate也就是testing的動作，最後在將每個epoch所存取來accuracy list 拿去繪製出下方的比較圖，實際code如下：

```
def train(name, model, device, optimizer, EPOCH):
    train_acc_list = []
    for epoch in range(EPOCH):
        train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True, num_workers=4)
        model.train()
        loop = tqdm(train_loader, total=len(train_loader), leave=True)
        train_acc = 0

        for data, target in loop:
            data, target = data.to(device), target.to(device)
            optimizer.zero_grad()
            output = model.forward(data)
            loss = nn.CrossEntropyLoss()(output, target)
            loss.backward()
            optimizer.step()
            loop.set_description('Epoch: {}, Loss: {:.4f}'.format(epoch+1, loss.item()))
            _, train_pred = torch.max(output,1)
            train_acc += (train_pred == target).sum().item()
            # print(train_acc)
        torch.save(model, './model/{}_{}.pth'.format(name, epoch+1))
        print("{}_model_saved{}.pth".format(name, epoch+1))
        # 每個epoch的訓練結果
        train_acc_list.append(train_acc/len(train_loader.dataset)*100)
        print(train_acc_list)
        torch.cuda.empty_cache()
    return train_acc_list

def test(name, model, device, EPOCH):
    test_acc_list = []
```

```
    for epoch in range(EPOCH):
        test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False, num_workers=4)
        model.load_state_dict(torch.load('./model/{}_{}.pth'.format(name, epoch+1)))
        model.to(device)
        model.eval()   # 不會更新參數
        loop = tqdm(test_loader, total=len(test_loader), leave=True)
        test_acc = 0
        for data, target in loop:
            data, target = data.to(device), target.to(device)
            output = model.forward(data)
            _, test_pred = torch.max(output,1)
            test_acc += (test_pred == target).sum().item()
        test_acc_list.append(test_acc/len(test_loader.dataset)*100)
        print(test_acc_list)
        torch.cuda.empty_cache()
    return test_acc_list
```

c. polt the confusion matrix

首先將預測結果與真實 label 存成一個矩陣，並利用 Skylearn 將結果算成混淆矩陣並normalize，再利用 seaborn 與 pandas 等套件繪製成圖表，圖上的數字分別代表該類數量與準確率。

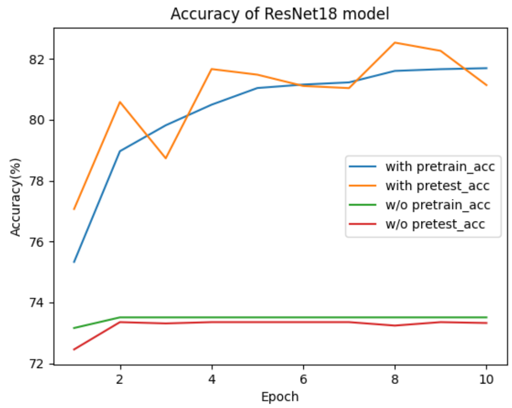
```
def makeconfusionmatrix(model, name):
    y_pred = []
    y_true = []

    # load the model weights
    model.load_state_dict(torch.load('./model/{}.pth'.format(name)))
    test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False, num_workers=4)
    model.to(device)
    model.eval()

    with torch.no_grad():
        for i, (images, target) in enumerate(test_loader):
            images=images.to(device)
            target=target.to(device)
            output = model.forward(images)
            _, preds = torch.max(output, 1)
            y_pred.extend(preds.view(-1).detach().cpu().numpy())
            y_true.extend(target.view(-1).detach().cpu().numpy())
            print(i,'/',len(test_loader))
    cf_matrix_normalize=confusion_matrix(y_true,y_pred,normalize='true')
    return cf_matrix_normalize

import pandas as pd
import seaborn as sns
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.show()
    # save the figure
    plt.savefig('confusion_matrix.png')
```

Experimental results (10%)

a. The highest testing accuracy

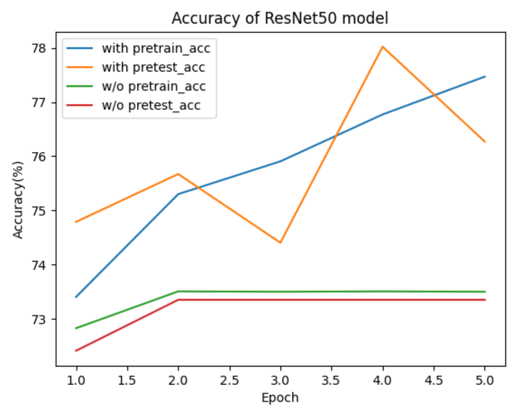
最高準確率我選用 pretrained 的 ResNet18 並將EPOCH = 10；BATCH_SIZE = 16；LR = 0.01，並透過 augmentation 將資料集水平與垂直翻轉與隨機旋轉進行擴充，最後最高準確率可以到達 **82.5338078291815**%如下圖

- ResNet18 in pretrain and not pretrain accuracy



max pretrain acc:81.6932986939037
max train acc:73.50795401971601
max pretest acc:**82.5338078291815**
max test acc:73.35231316725978

- ResNet50 in pretrain and not pretrain accuracy



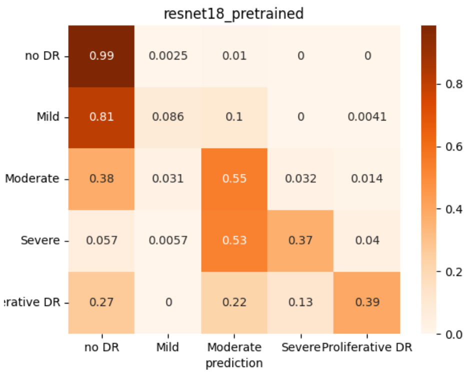
max pretrain acc: 77.46894907292075
max train acc:73.50795401971601
max pretest acc:78.02135231316726
max test acc:73.35231316725978

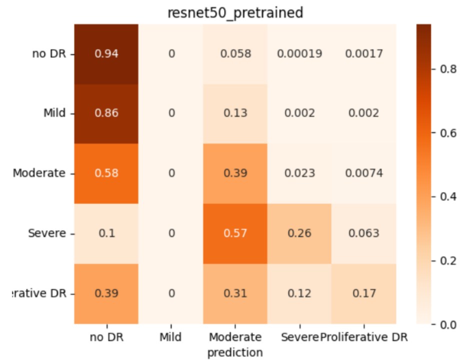
b. Comparison figures

從上圖可以看出有無 pretrained 的 ResNet，可以看出有 pretrained 過的model，在提升 train accuracy 方面非常效果非常顯著，而沒有pretrained 過的雖然 weight 都有改變，但修正不夠大，因此準確率沒有明顯上升。

c. Confusion Matrix comparison

- Resnet18 pretrain = True 的 confusion matrix
- Resnet50 pretrain = True 的 confusion matrix



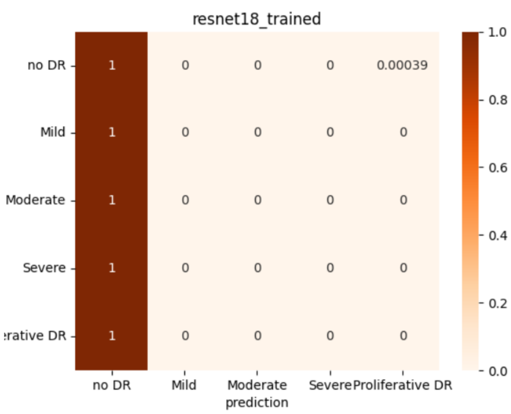


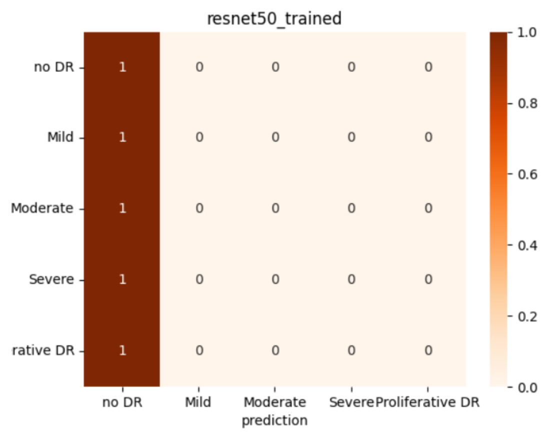
- Resnet18 pretrain = False 的

  confusion matrix

- Resnet50 pretrain = False 的

  confusion matrix





我們透過混淆矩陣觀察有 pretrain 過的 model 再分類也比較合理，但除了第一類，其他種類的準確率均不高，在實際情況下還有很大改進空間，我認為很有可能是因為資料不太平均，資料大部分都坐落在第一類，第二類的資料數量較少，因此很難預測到第二類的答案，因此第二類的準確率極低，而沒有 pretrain 過的矩陣效果非常差，幾乎將所有東西都分到第一類。

ResNet18/50 在有沒有 pretrained 也有類似的情況，第一類的分類結果還不錯但其他種類效果都非常差，沒有 pretrained 過的 model 準確率也沒有明顯上升並且很容易把結果全都分到第一類。

Discussion (20%)

a. Confusion Matrix

   Confusion Matrix在機器學習或深度學習中，最常見的就是分類，但要去判斷分類的好不好，單憑準確率是不夠的，因此混淆矩陣(Confusion Matrix)的各項指標會被拿來參考。

   混淆矩陣的 4 個元素(TP,TN,FP,FN)

   - TP(True Positive):正確預測成功的樣本
   - TF(True Negative):正確預測錯誤的樣本
   - FP(False Positive):錯誤預測成正樣本，實際上是負樣本
   - FN(False Negative):錯誤預測成負樣本

   而對混淆矩陣做 Normalization 則能看到每個種類的機率混淆矩陣有三種主要計算的方式，

- 第一種是準確率(Accuracy)計算公式為 AC=(TP+TN)/TP+FP+FN+TN)
- 第二種是精確率(Precision)=(TP)/(TP+FP)，代表判斷為陽性的樣本有機個是預測正確的。
- 第三種是召回率(Recall)=TP/(TP+FN) ，代表真實為陽性的樣本中有幾個是預測正確的。

b. Augmentation 資料增強

在本次作業我發現一種問題，因為大部分的資料都屬於第一類(NO DN)，而第二種種類的資料很少，因此透過混淆矩陣看出很難預測第二種的成果，因此若能增加第二種資料的話，對於整體的預測也能更為精準。而資料的擴充除了最基本的左右翻轉，若能做裁切旋轉也能有效提升準確率，本次實驗最高準確率也是在進行 Augmentation 下得到的。