1. Introduction

這次作業就是用resnet18，resnet50，用圖片去預測class。
這份作業讓大家學習:
(1)如何製作customized dataset，還有實作data augmentation。
(2)使用別人pretrained/unpretrained的model，並且學習如何train pretrained model。
(3)學習如何繪畫confusion matrix。

2. Experiment setups

A. The details of your model (ResNet)

我的model是參考data.zip中的model architecture實作的，也就是將torchvision.models.Resnet18/50引入model的不同layer中，最後再用flatten攤平後，才做classify。

其中Resnet本體分成9層，分別是
conv1
bn1
relu
maxpool
layer1
layer2
layer3
layer4
classify
然後basic block/bottleneck block就包在layer1,2,3,4內，可以解決gradiant vanish。
至於說如何知道torchvision.models.Resnet18/50有幾層，可以使用model.modules()查看。

```python
class ResNet(nn.Module):
    def __init__(self, Layer=18,Pretrained=True):
        super(ResNet, self).__init__()
        if Layer==18:
            self.classify = nn.Linear(512, 5)
        if Layer==50:
            self.classify = nn.Linear(2048, 5)

        pretrained_model = torchvision.models.__dict__['resnet{}'.format(Layer)](pretrained=Pretrained)
        self.conv1 = pretrained_model._modules['conv1']
        self.bn1 = pretrained_model._modules['bn1']
        self.relu = pretrained_model._modules['relu']
        self.maxpool = pretrained_model._modules['maxpool']

        self.layer1 = pretrained_model._modules['layer1']
        self.layer2 = pretrained_model._modules['layer2']
        self.layer3 = pretrained_model._modules['layer3']
        self.layer4 = pretrained_model._modules['layer4']

        self.avgpool = nn.AdaptiveAvgPool2d(1)

        del pretrained_model

    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)
        # print(x.shape)
        x = torch.flatten(x,start_dim=1)

        x = self.classify(x)

        return x
```

B. The details of your Dataloader

Dataloader也是跟上次一樣，直接呼叫torch的Dataloader。

```python
train_custumized_dataset = RetinopathyLoaDER("data","train")
test_custumized_dataset = RetinopathyLoaDER("data","test")
train_loader = DataLoader(train_custumized_dataset,batch_size=batch_size)
test_loader = DataLoader(test_custumized_dataset,batch_size=batch_size)
```

這次不太一樣的地方，是dataset要自己寫一個custumized_dataset，不能直接拉TensorDataset用。

至於custumized_dataset，我的寫法如下

首先是__init__

```python
def getData(mode):
    if mode== 'train':
        img = pd.read_csv('train_img.csv')
        label = pd.read_csv('train_label.csv')
        return np.squeeze(img.values),np.squeeze(label.values)
    else:
        img = pd.read_csv('test_img.csv')
        label = pd.read_csv('test_label.csv')
        return np.squeeze(img.values),np.squeeze(label.values)

class RetinopathyLoaDER(data.Dataset):
    def __init__(self, root,mode):
        self.root = root
        self.img_name, self.label = getData(mode)
        self.mode = mode
        self.train_transform = transforms.Compose([transforms.RandomHorizontalFlip(),
transforms.RandomVerticalFlip(), transforms.RandomRotation(45), transforms.ToTensor()])

        self.test_transform = transforms.ToTensor()
        print(">Found %d images..." %(len(self.img_name)))
```

先將dataset的相關資訊存起來，像說位置，檔名，train/test，還有data_augmentation都先存起來。

data_augmentation的話，如果是train_dataset，我就對他做任意水平/垂直翻轉和旋轉後，再對其做nomalize和把dimension變成[C,H,W]

如果是test_dataset，我則不對他做data_augmentation，只做nomalize和把dimension變成[C,H,W]。

接下來是__len__和__getitem__，這兩個函式是為了讓Dataloader能正常使用dataset而生的。

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

def __getitem__(self,index):
    path = self.root +'/' +self.img_name[index]+'.jpeg'
    img_Im = Image.open(path)
    if self.mode == "train":
        img = self.train_transform(img_Im)
    else:
        img = self.test_transform(img_Im)
    label = self.label[index]
    return img, label
```

__len__就回傳有多少筆data，__getitem__則是讀取照片，對照片做/不做augmentation，再將照片和label return出去。

C. Describing your evaluation through the confusion matrix

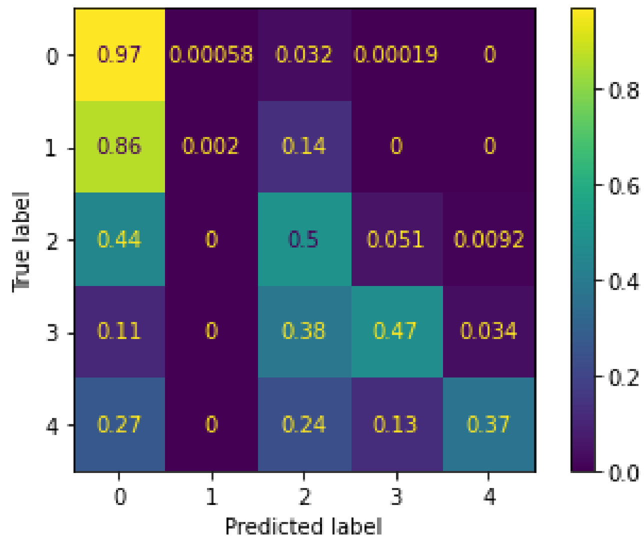
confusion matrix就是看說ground_truth和對應prediction分別分佈在哪裡，能更有效知道分佈狀況。

而有時ground truth分佈非常不均時，直接看每格對應的數字可能會失真，應該要做normalization。

舉個例子，假設一個村落正常人10000人，該年正常人死了20人，癌症病患2人，該年癌症病患死了1人。看起來死亡人數正常人比癌症病患多很多，但實際上致死率的話，癌症病患死亡率顯然比較高的。

而這個dataset也有分布不均的特性，True label=0的data特別多，佔了7成，所以最後我對每個ground_truth做了normalize。

Resnet18 pretrained True confusion matrix



像這張是有pretrained過的resnet18最高test_acc的confusion matrix，可以看到true label=0預測的特別準，而其他的資料比較少，預測的不太準。尤其是true label=1的，大多都預測成了true label=0。

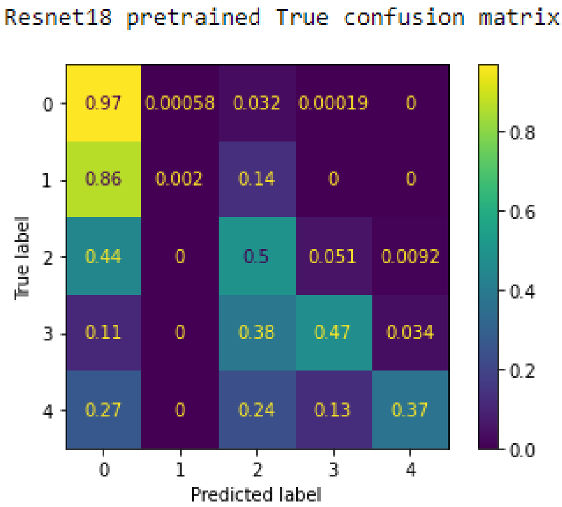
3. Experimental results

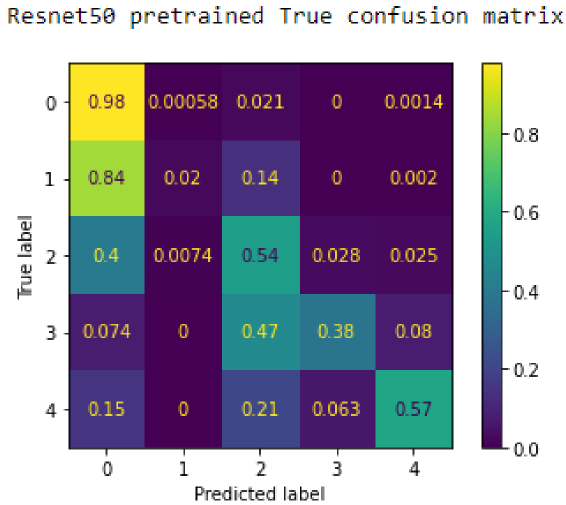
A. The highest testing accuracy

Screenshot

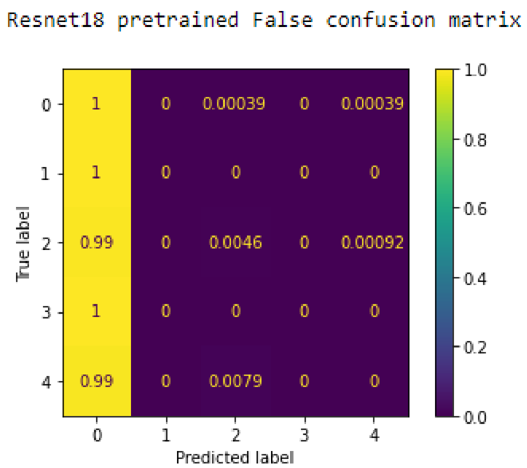
```
highest_acc:
resnet18_pretrained=True_test :  81.59430604982207 %
resnet18_pretrained=False_test :  73.36654804270462 %
resnet50_pretrained=True_test :  82.14946619217082 %
resnet50_pretrained=False_test :  73.35231316725978 %
```

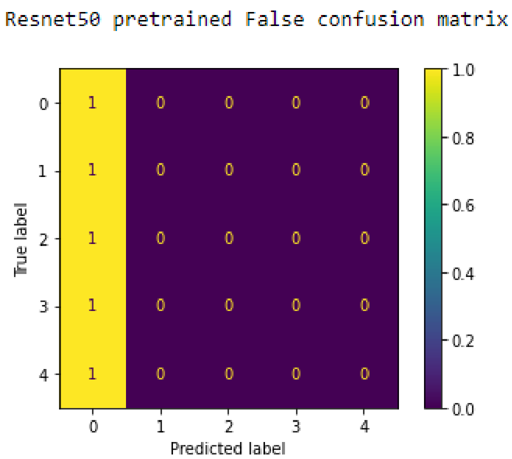
Anything you want to present





從confusion matrix中可以看見，pretrained model在 true label=0時都預測得特別好，反之true label=1時預測的比較差，我猜是因為true label=0占了快7成，而true label=1又跟true label=0時很像。



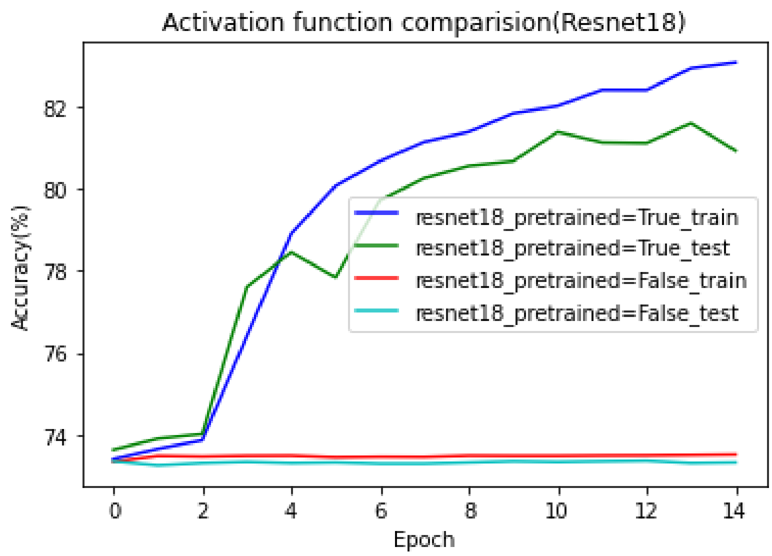


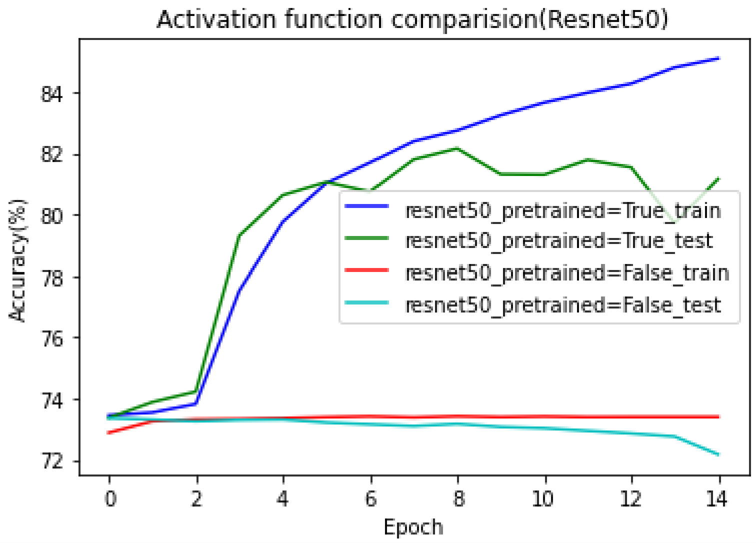
可以看到unpretrained model似乎有偏好只猜0的狀況，畢竟true label=0占了快7成，無腦猜0也會有73%左右，認真猜還不一定能猜贏無腦猜。

B. Comparison figures

Plotting the comparison figures

(RseNet18/50, with/without pretraining)





4. Discussion

A. Anything you want to share

1.一開始我完全沒做data augmentation，也沒讓pretrained model的前幾個epoch凍結pretrained backbone，導致我的resnet18的test accuracy在前幾次都有overfitting的現象。

```
epoch:  1  train acc:  74.62899035552867  test acc:  76.69750889679716
epoch:  2  train acc:  78.13445318338731  test acc:  78.6049822064057
epoch:  3  train acc:  79.95302323926119  test acc:  78.41992882562278
epoch:  4  train acc:  81.41570874408342  test acc:  77.72241992882562
epoch:  5  train acc:  82.5189508523435  test acc:  73.59430604982207
epoch:  6  train acc:  83.88554752838179  test acc:  73.43772241992883
epoch:  7  train acc:  84.76458236947934  test acc:  76.27046263345196
epoch:  8  train acc:  85.88917755080251  test acc:  73.55160142348754
epoch:  9  train acc:  86.73618278230542  test acc:  75.23131672597864
epoch:  10  train acc:  88.51916438307413  test acc:  72.12811387900356
epoch:  11  train acc:  89.99252642442792  test acc:  72.61209964412811
epoch:  12  train acc:  91.095768532688  test acc:  64.75444839857651
epoch:  13  train acc:  92.04242143848535  test acc:  74.43416370106762
epoch:  14  train acc:  93.90013879497491  test acc:  74.1779359430605
epoch:  15  train acc:  94.51226022278372  test acc:  71.08896797153025
```

為了解決這個現象，所以我對dataset做了data augmentation。
並且在前3epoch只train network的最後一層(flatten)。

```python
#Train
def set_paremeter_requires_grads(layer,is_required):
    if is_required:
        for param in layer.parameters():
            param.requires_grad = True
    else:
        for param in layer.parameters():
            param.requires_grad = False
def append_model_grad(model_grad,model):
    for param in model.parameters():
        if param.requires_grad == True:
            model_grad.append(param)

def train(model,train_loader,Momentum,wd,lr,epoch_i,pretrained,device):
    if pretrained:
        ct = 0
        model_grad=[]

        for children in model.children():
            ct += 1
            if (ct > 1) & (epoch_i<3):
                set_paremeter_requires_grads(children,0)
            else:
                set_paremeter_requires_grads(children,1)

        append_model_grad(model_grad,model)
        optimizer = optim.SGD(model_grad,momentum=Momentum,lr = lr,weight_decay = wd)

    else:
        optimizer = optim.SGD(model.parameters(),momentum=Momentum,lr = lr,weight_decay = wd)
    #etc.....
```

作法也就是把model的layer用for迴圈跑過
選擇將要凍結的model，requires_grad =False，並且參數不要加入optimizer中。

2.一開始做batch size = 4，最高%數圖如下。

```
highest_acc:
resnet18_pretrained=True_test :  80.46975088967972 %
resnet18_pretrained=False_test :  73.35231316725978 %
resnet50_pretrained=True_test :  80.44128113879003 %
resnet50_pretrained=False_test :  73.35231316725978 %
```

為了讓模型能到82%，所以我就調成batch size=32，也就是之前提到的結果。