DLP-LAB04-2:
Diabetic Retinopathy Detection

學號：310611008 姓名：張祐誠

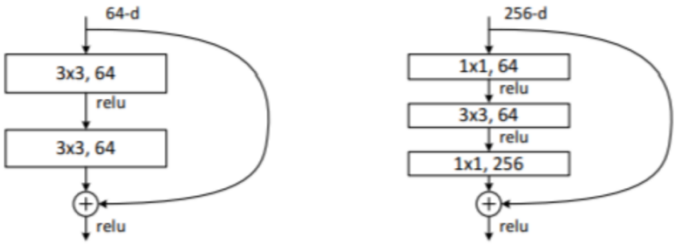
Introduction:

在這次 LAB 中，要達到以下事項：

1. 改寫 dataloader 來讀取我們要訓練的圖片和對圖片進行預處理
2. 利用 resnet18 和 resnet50 的架構對視網膜疾病資料進行訓練
3. 利用 confusion matrix 對我們訓練出的 model 進行評估

Resnet:

Resnet 的特別在於他在每個 block 外面多增加了一個路徑，如下圖所示，



這樣的設計提供了一種解決 gradient vanishing 問題的方式，這類的問題常常發生在深層網路之中；而這種設計也減少了不少的運算量，讓深度網路能夠更容易的訓練。上圖左是 resnet18 中使用的基本 block，右邊則是 resnet50 中使用的 bottleneck block。

Pretrained weight:

在訓練一個模型時，我們可以先用不同且廣泛的 dataset 先對 model backbone 做訓練，然後利用訓練好的參數作為我們當前任務的初始值。利用 pretrained weight 訓練的手法稱為 fine tune。預期使用 pretrained weight 的優點如下：

1.加快模型收斂的時間：因為事先有訓練過，已經可以對同性質任務中的特徵有認識，因此可以減少訓練的時間。

2.減少訓練所需要的資料量：概念基本上與前一點類似，因為對同性質任務中有一些概念的認識，因此只要能學習當前任務的特徵就可以了。

Experiment Setups:

A. Details of model

```python
def model_init(model:str, use_pre_weight: bool, featrure_extract = False):

    Resnet = {
        'resnet18': models.resnet18(pretrained=use_pre_weight),
        'resnet50': models.resnet50(pretrained=use_pre_weight)
    }

    model = Resnet[model]
    set_parameter_requires_grad(model=model, feature_extracting=featrure_extract)
    num_ftrs = model.fc.in_features
    model.fc = nn.Linear(num_ftrs, 5)

    return model
```

本次實作使用了 resnet 18 跟 resnet 50 兩種架構，我直接使用 torch.model 裡面的模型使用，因爲預設的 output 有 1000 個 class，所以更改成本次 lab 的 data 裡的 5 種分類。

```python
def set_parameter_requires_grad(model, feature_extracting):
    if feature_extracting:
        for param in model.parameters():
            param.requires_grad = False
```

其中 fine tune 裡 feature extracting 是對於更新權重的部分，如果是 True 的話，就只會更新最後一層輸出的權重而已，相對的如果是 False 的話，則會更新所有的權重。在經過實驗後，在本次 lab 當 feature_extracting 為 False 時可以有比較好的訓練結果。

B. Details of dataloader

```python
def __getitem__(self, index):
    PATH = self.root +'/'+ self.img_name[index] + '.jpeg'
    img_data = cv2.imread(PATH)
    img_rgb = cv2.cvtColor(img_data, cv2.COLOR_BGR2RGB)

    if self.mode == 'train':
        transform = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]),
            transforms.RandomRotation((-180, +180)),
            transforms.RandomHorizontalFlip()
        ])
    else:
        transform = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
        ])

    img = transform(img_rgb)

    label = self.label[index]
```

在 data loader 中，我先使用 opencv 讀取每張圖片並轉成 rgb 圖像；在 training 中，先把圖像轉成 tensor(包含 reshape)，然後將圖片 normalize，再隨機旋轉、翻轉，盡可能讓圖片隨機性提高。

train_img
2576_right
6592_left
30349_right
1741_right

	2576_right
0	6592_left
1	30349_right
2	1741_right
3	33869_left
4	28423_right

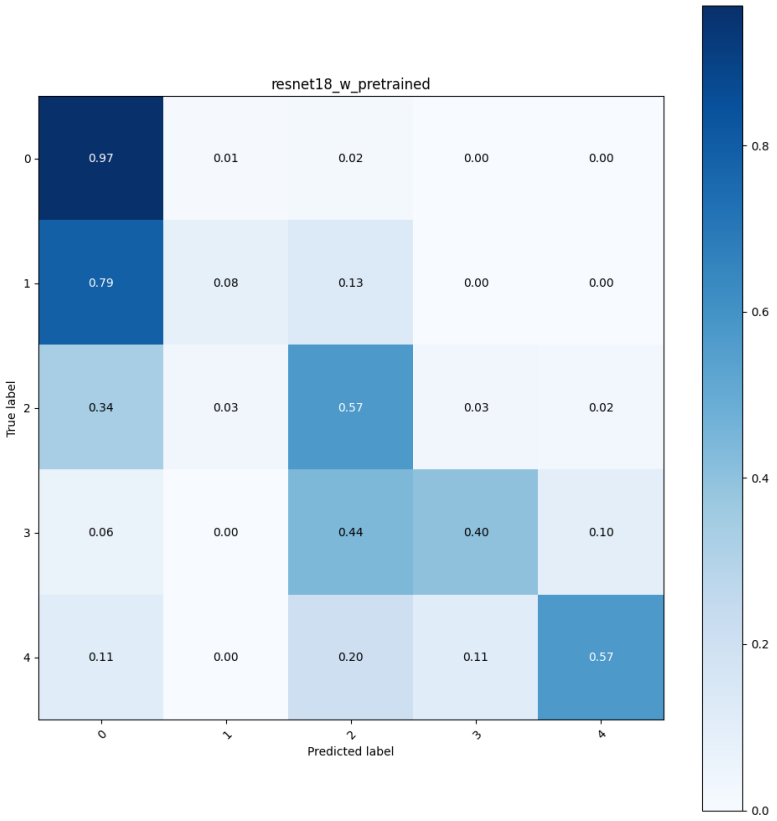
	train_img
0	2576_right
1	6592_left
2	30349_right
3	1741_right
4	33869_left

另外我在所有 csv 中都加入 header，如左圖，因為 pandas 讀取的設定，會自動將第一個設為 header，也就是說如果沒有 header 的話，'2576_right' 並不會被放入我們的 training dataset，如中圖。更改後讀取結果如右圖。

```python
def showTensorImg(img):
    import matplotlib.pyplot as plt
    img = img / 2 + 0.5
    npImg = img.numpy()
    plt.imshow(np.transpose(npImg, (1, 2, 0)))
    plt.show()
```

另外如果在檢視讀取的 data 的時候，因為 tensor 資料型態沒版法被顯示，所以我有另外寫 function 在我有需要時可以顯示讀取的圖片。

C. Evaluation through the confusion matrix



我們以這次測試的 resnet18 結果為例，對角線上是預測資料跟實際資料相符的位置，所以直覺上來說，當預測結果都落在對角線上是我們想要的。
另外我們有一些可以評斷預測結果的指標：

$$precision = \frac{True\ positive}{True\ positive + False\ positive}$$

$$recall = \frac{True\ positive}{True\ positive + False\ negative}$$

$$f1\ score = \frac{2 * precision * recall}{precision + recall}$$

通常我們關注的 accuracy 是指落在對角線上除以整體資料的總數，但在一些狀況中並不顯得實用，像這次判斷疾病便是其中之一。

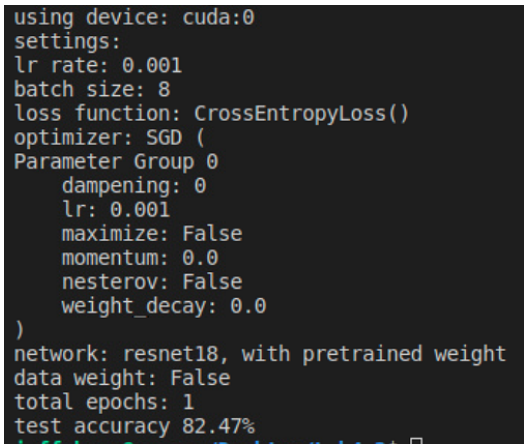
所以比起看總共對了多少 testing set，上面的指標也可以提供我們參考，Precision 的意涵是判斷為陽性樣本中有多少是正確的，而 recall 則是意涵著本來就有疾病的人，被診斷出的機率是多少，而 f1 是兩者調和平均的結果。以這次疾病分類來說，我們更傾向減少偽陰性發生的概率，所以 recall 會是我們可以重視的指標。

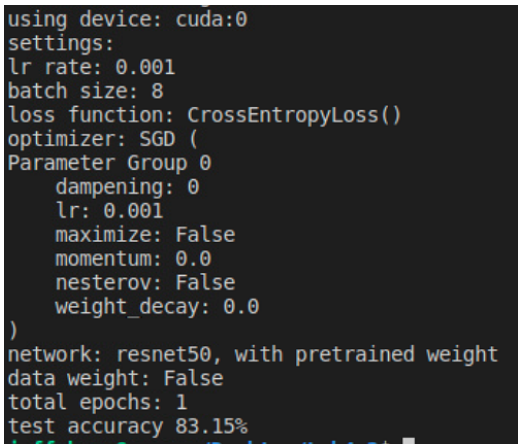
而計算這些數值在 scikit-learn 函式庫裡已經有 classification report 可以直接使用，提供我們快速的計算 confusion matrix 隱含的資訊。

Experiment Results:

A. The highest testing accuracy:

Screen shot of with two models





Highest testing accuracy (%):

	Without pretrained weight	With pretrained weight
Resnet 18	73.77	**82.47**
Resnet 50	73.36	**83.15**

Extra thing to present:

Regularization term:

在 optimizer 設定中可以設定 weight decay，也就是 regularization，這樣可以避免 over fitting 的問題。在這次的 case 中，我嘗試設定與否，結果如下

Highest testing accuracy (%):

	Without regularization	With regularization
Resnet 18 with pretrained weight	80.7	**82.47**
Resnet 50 with pretrained weight	81.43	**83.15**

在這次 lab 中並沒能提升 testing data 準確率，所以我將 regularization term，也就是 weight decay 設為零。

Momentum:

另外一個可以可以設定的參數是 momentum，這一項讓我們在梯度下降時可以更快收斂到最小值，同樣的我也嘗試設定與沒設定，結果如下：

Epoch to testing accuracy >= 81%:

	Without momentum	With momentum
Resnet 18 with pretrained weight	11	**7**
Resnet 50 with pretrained weight	8	**6**

可以看到有 momentum 可以讓我們更快的收斂，所以設定 momentum = 0.9。

Other hyper parameters:

Learning rate = 1e-3;

Epochs = 20;

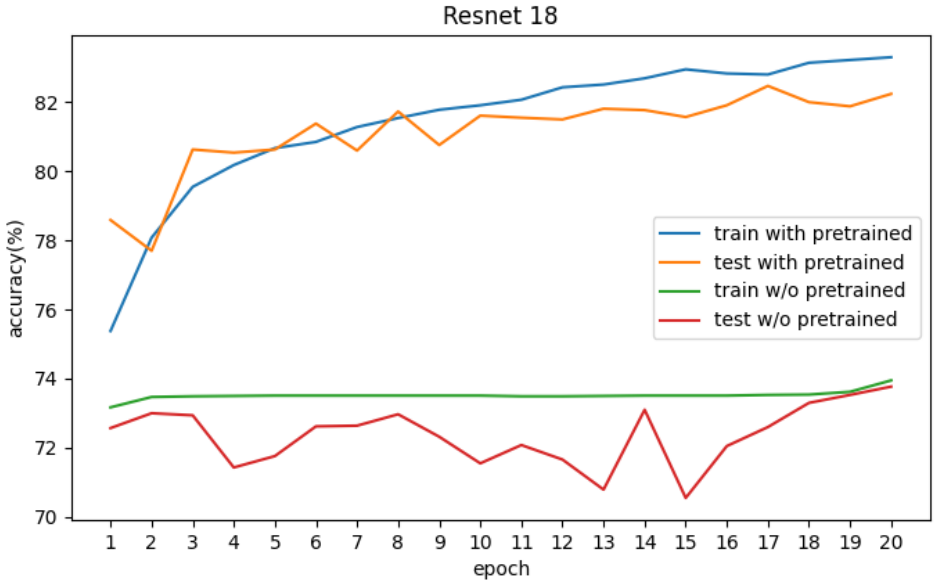
Loss function = Cross Entropy;

Batch size = 8;

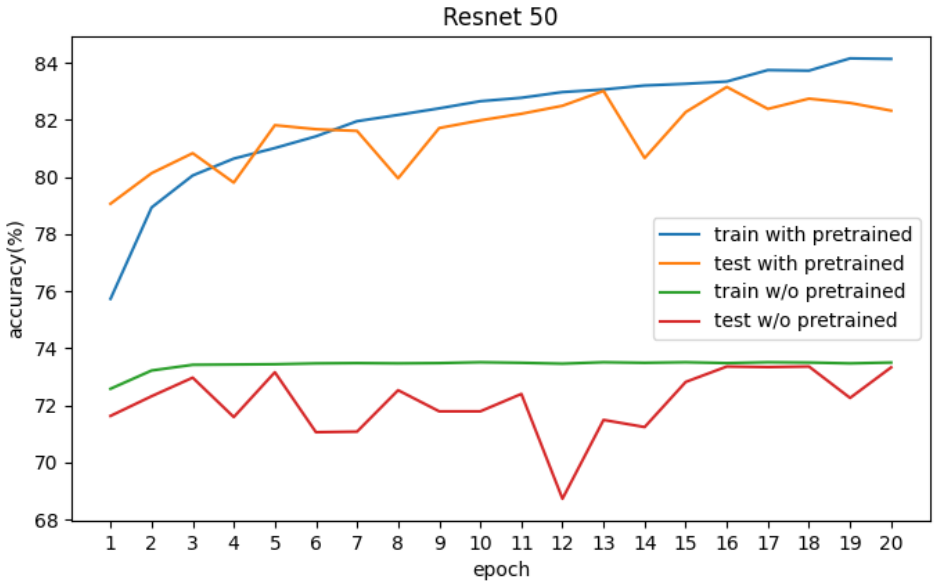
Optimizer = SGD;

B. Comparison Figure:

Resnet18:



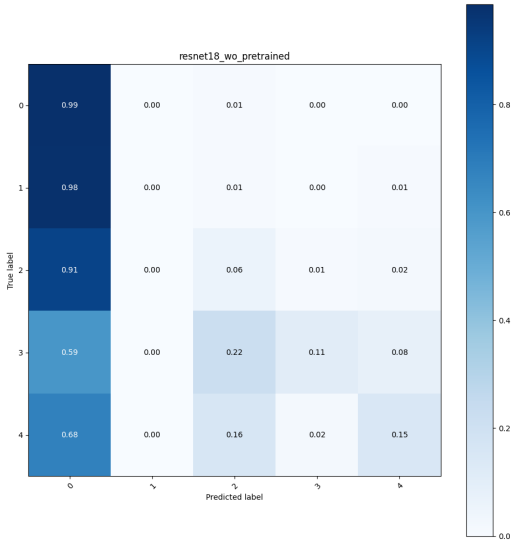
Resnet50:



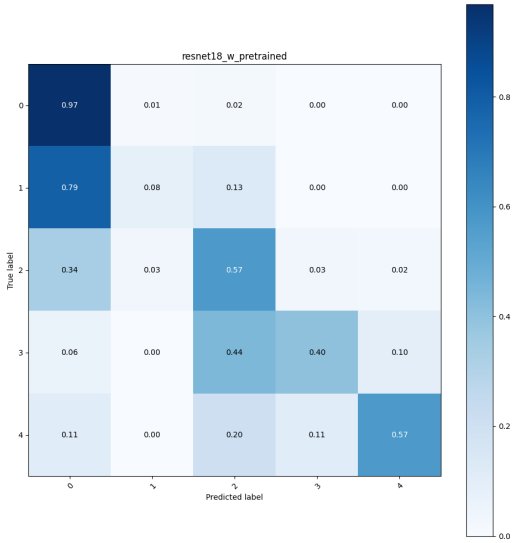
可以看到在這次 LAB 中，有 pretrained weight 的 model 都在 20 個 epoch 內讓 accuracy 提升到 82%以上，而沒有 pretrained weight 的 model 都只有在 73%附近震盪而已，所以有 pretrained weight 的狀況下確實可以減少模型收斂時間。繼續往下 train 不知道 train from scratch 能否提升他的 accuracy，如果有時間可以讓他再繼續 train 看看。

Confusion Matrix:

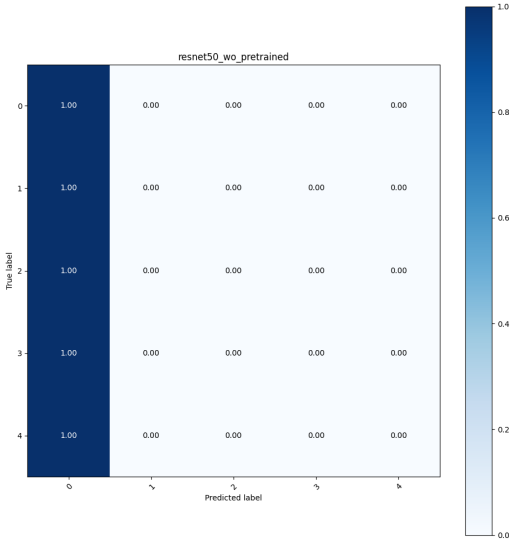
Resnet18 without pretrained:



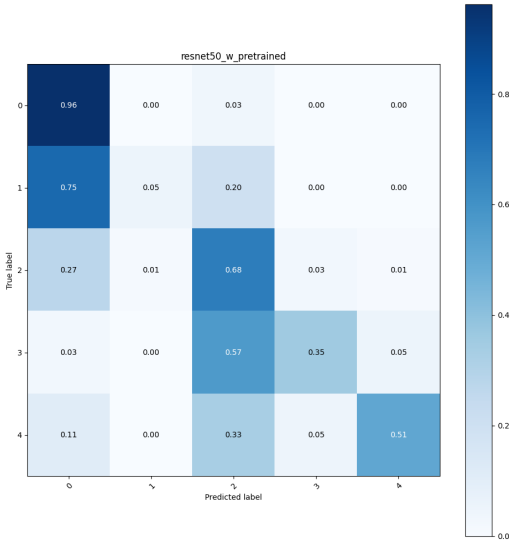
Resnet18 with pretrained:



Resnet50 without pretrained:



Resnet50 with pretrained:



可以看到接近的結論，有 pretrained weight 的 model 在訓練上也能得到比較好的分類方式，沒有 pretrained 可以說是幾乎沒有學習到任何特徵。

Discussion:

在這種疾病的資料很常會有資料權重失衡的問題，這次的資料集也是，在 training dataset 跟 testing dataset 最多和最少都差了 35-40 倍，換句話說，只要模型策略全部猜測陰性，他就會有 73%的 accuracy，很明顯這並不是我們希望的結果。

通常處理的方式有兩種：第一個是對於數量較少的 class 生成類似的 data；另一種是在計算 loss 的時候給數量較少的 data 較大的權重。而我嘗試了第二種方法來處理這個問題。

我計算各個資料的權重方式為 最多數量的 class 的數量/每個 class 的數量，得到以下的權重：

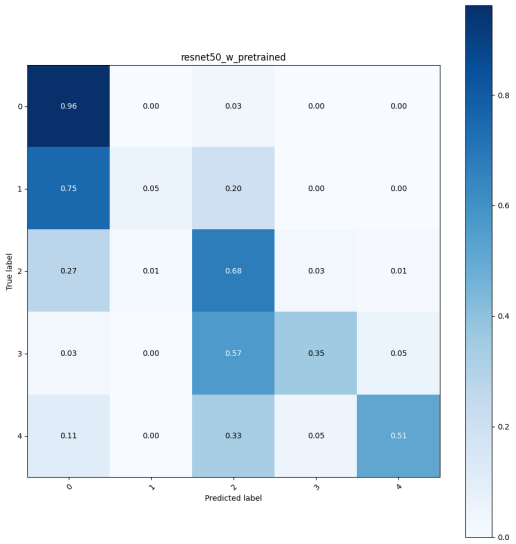
```
def getWeight(self):
    count = np.bincount(self.label)
    maxClass = np.nanmax(count)
    self.data_weight = maxClass/count
    print(self.data_weight)

    return torch.tensor(self.data_weight).float()
```

```
[ 1.         10.5657289   4.9064133  29.59312321 35.5524957 ]
> Found 7026 images...
```

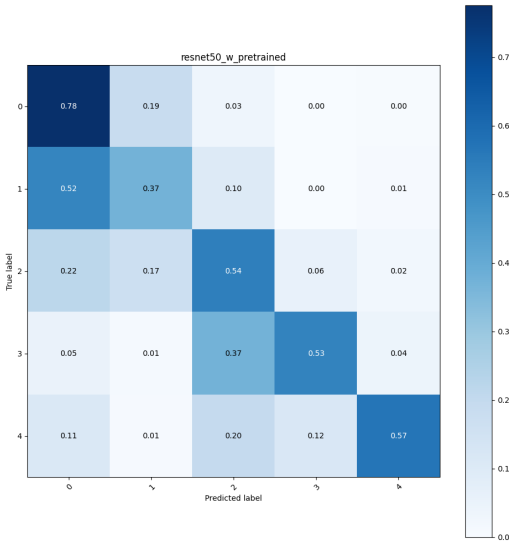
以 resnet50 with pretrained weight 為例，比較有給予 data 權重與否進行比較：

Confusion matrix:

Without data weight:



With data weight:



可以很明顯的看到資料更集中在對角線上，為了能支持這樣想法，我使用了 scikit-learn 的 classification report 找出相關的指標：

Without data weight:

	precision	recall	f1-score	support
0	0.88	0.96	0.92	5154
1	0.48	0.05	0.09	488
2	0.64	0.68	0.66	1082
3	0.64	0.35	0.46	175
4	0.71	0.51	0.59	127
accuracy			0.83	7026
macro avg	0.67	0.51	0.54	7026
weighted avg	0.81	0.83	0.80	7026

With data weight:

	precision	recall	f1-score	support
0	0.89	0.78	0.83	5154
1	0.14	0.37	0.20	488
2	0.65	0.54	0.59	1082
3	0.54	0.53	0.53	175
4	0.59	0.57	0.58	127
accuracy			0.70	7026
macro avg	0.56	0.56	0.55	7026
weighted avg	0.78	0.70	0.74	7026

這邊可以看到，給了 data 權重之後，整體的 accuracy 只剩下 70%，這個數字比全部猜陰性的 73%還糟，但我們可以很直觀的否定這樣的想法，因為 Confusion matrix 告訴我們在給了 data 權重之後，我們的結果更趨向對角線上了。

所以如同前面所述，我們可以參考其他指標來判斷我們的模型好壞，在這個 case 中，可以觀察 recall 的表現狀況，可以發現儘管 accuracy 更低了，可是除了陰性（class = 0）外，大部分的 class 的表現都變好了，所以至於哪個指標才是我們重視的，是 depend by case 的。在這次任務中，我會更傾向使用有 data 權重訓練出來的模型。