Lab3 : Diabetic Retinopathy Detection

311511043 李承翰

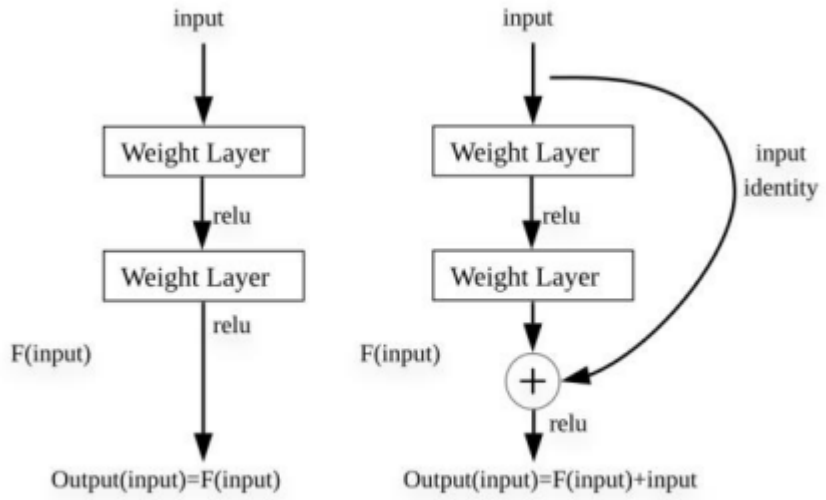
1. Introduction

在這次 Lab，我們需要自行設計 dataloader，並且利用 ResNet 網路架構分析糖尿病所引發的視網膜病變分類問題，比較有無 pretrain weight 情況下模型的表現，並透過 confusion matrix 來評估模型的性能。

而此次我們使用的 dataset 總共有 35123 張照片 ，每一張照片的解析度為 512 * 512，而 label class 總共有 5 個:0～4 分別代表著病變的嚴重程度

而這次我們所使用的 ResNet 網路指的是在一般的捲積神經網路當中加入殘差學習的概念，以往當神經網路層數增加的時候，就會出現退化(degradation)的問題，也就是當層數越來越深，模型準確率接近飽和，反而變得更不穩定，同時因為層數加深，所以也相當有可能會出現梯度消失的問題，而此時 ResNet 引入了殘差映射的概念，讓此問題得以被解決，也就使模型深度可以加深，在準確性提高的同時也能夠提升穩定性。

殘差映射(Residual Mapping) 就是在模型的訓練途徑當中曾加一條捷徑，讓 output(x) = f(x) + x, 透過這樣一條捷徑，即使模型在這一層當中沒有學習到任何東西，她也能夠確定模型的下一層輸入剛好等於這一層的輸入，如此一來模型就不會退化，使模型能夠原來越深。



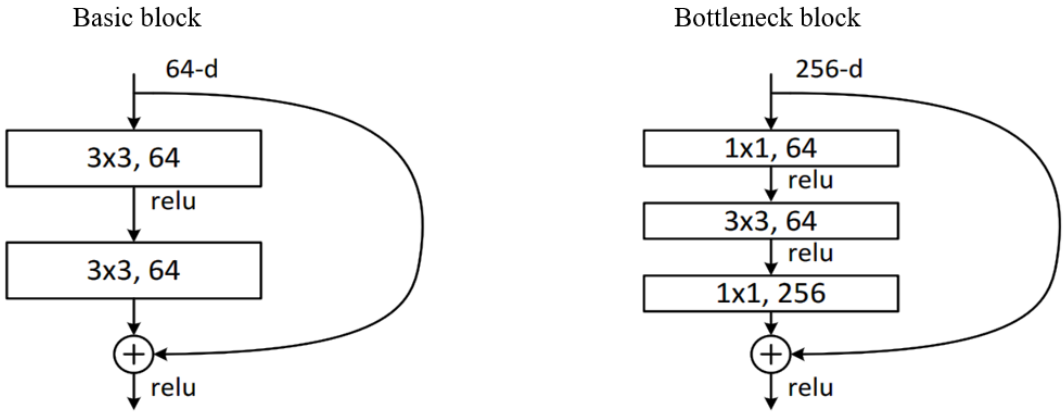
2. Experiment setup

A. The detail of model (ResNet):

這一次的 ResNet 模型不需要我們自己設計結構，在 torchvision 當中已經有這一個模型的架構，我們只要從 torchvision 當中將她引入即可。不過當初在 torchvision 當中這個模型的最後一層分類器是分類 1000 的 class 用的，因此我們需要將最後一層的分類器當中的參數進行改寫。

```
self.model.fc = nn.Linear(self.model.fc.in_features,5);
```

而這次我們分別使用的 ResNet18、及 ResNet50 差別主要在層數不同，而在 ResNet50 的架構設計當中，因為其需要的參數量實在過大，它還設計了一個 bottleneck 的結構。



這次我們不只是單純的引入這個模型的架構，我們還要從 torchvision 當中引入 pretrained weight，使用 torch.util.model_zoo 當中預先提供的 weight，讓我們後續的學習可以更順利。

而在訓練當中，在採用 pretrained weight 之後，我們可以採用 先 linear probing 再 fine tune 的設計。一開始在 linear probing 時我們只訓練最後一層 fully connected 的 classifier 那層，在先行訓練幾個 epoch 之後再轉到 fine tune 階段，重新訓練整個模型的 weight。

B. The details of your dataloader:

這次我們要自行設計 RetinopathyLoader 這一個 class，它的功能類似一個 dataset 的介面，將這個 dataset 放進 pytorch 當中的 dataloader 裡面，我們就可以將資料們以 batch_size 進行分類，也可以決定在分配訓練用資料時是否需要進行 shuffle，讓每次的訓練資料以不同的順序餵進模型當中。

在 RetinopathyLoader 當中我們 override 了 __getitem__()這一個函數，讓 dataloader 可以根據 index 取出相應的圖片，並且在進行一些圖形的預處理之後回傳至 dataloader 之中。

而預處理的部分，這次我使用 RandomHorizontalFlip(), RandomVerticalFlip()將資料進行隨機的翻轉，接著使用 ToTensor()將圖片翻轉，從[H,W,C] 轉為[C,H,W]， 並把圖像的數值轉換到[0,1]的區間，最後在進行一次 Normalize 以加快模型的收斂。

而 Normalize 方面，這次我計算該張照片的 mean 及 std 來做為參數放進函數當中。

C. Describing your evaluation through the confusion matrix

在訓練過程當中在我們得到最高的 test_acc 時，我記錄下當下的

prediction list，並且以一個 5*5 的 table 表示 normalize 後其出現的機率，在表格當中可以看見實際 ground truth 下我們針對模型的 precision 以及 recall 分別為多少，以此來評斷我們模型的好壞。

3. Experimental results

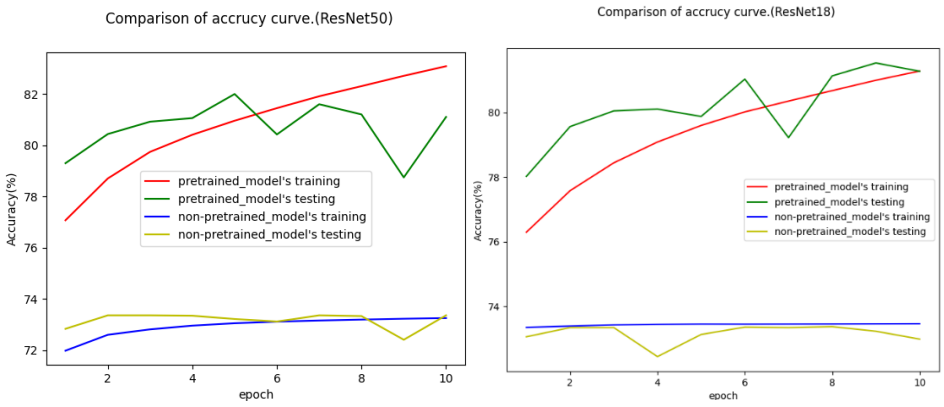
A. The highest testing accuracy:

表格中與截圖數據有些微不同，因為是不同次實驗當中擷取的

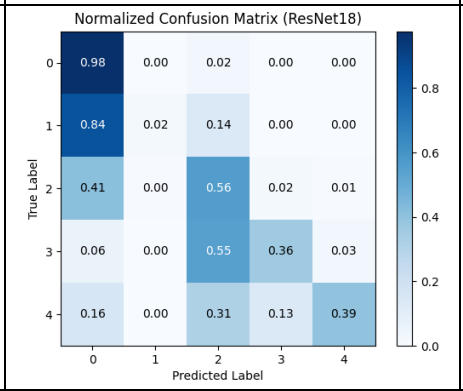
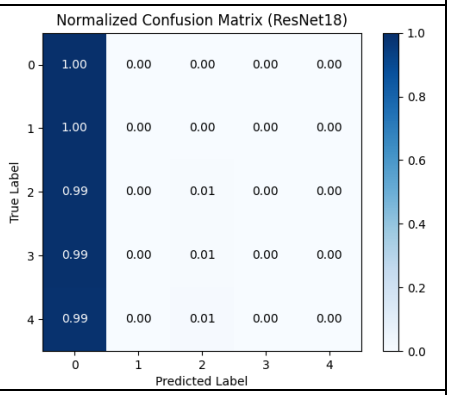
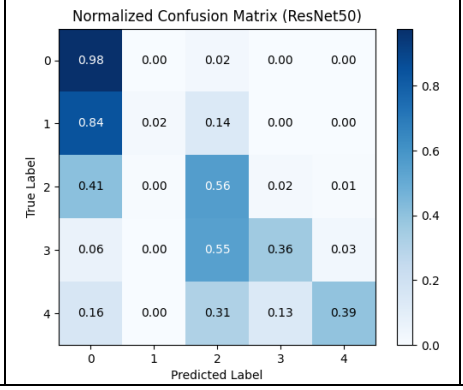
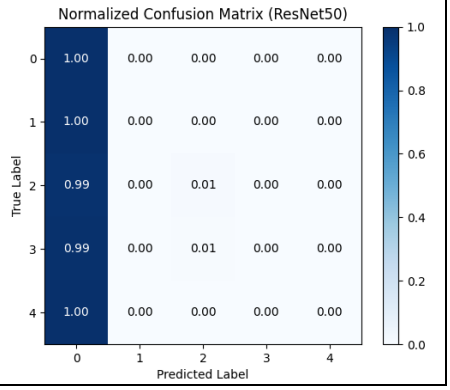
	With pre-train	Without pre-train
ResNet18	81.52%	73.38%
ResNet50	82.23%	73.35%

```
epoch   1, loss: 0.8472, traain_acc: 0.7079, test_acc: 0.7335
epoch   2, loss: 0.8220, traain_acc: 0.7214, test_acc: 0.7338
epoch   3, loss: 0.7499, traain_acc: 0.7260, test_acc: 0.7338
epoch   4, loss: 0.8235, traain_acc: 0.7287, test_acc: 0.7377
epoch   5, loss: 0.8082, traain_acc: 0.7304, test_acc: 0.7368
epoch   1, loss: 0.7291, traain_acc: 0.7668, test_acc: 0.7842
epoch   2, loss: 0.5500, traain_acc: 0.7838, test_acc: 0.8028
epoch   3, loss: 0.5580, traain_acc: 0.7942, test_acc: 0.8097
epoch   4, loss: 0.6959, traain_acc: 0.8013, test_acc: 0.8090
epoch   5, loss: 0.6637, traain_acc: 0.8075, test_acc: 0.8179
epoch   6, loss: 0.5376, traain_acc: 0.8127, test_acc: 0.7964
epoch   7, loss: 0.3956, traain_acc: 0.8175, test_acc: 0.8158
epoch   8, loss: 0.4774, traain_acc: 0.8215, test_acc: 0.8162
epoch   9, loss: 0.2477, traain_acc: 0.8256, test_acc: 0.8044
epoch  10, loss: 0.2441, traain_acc: 0.8296, test_acc: 0.8140
```

B. Comparison figures



從圖表當中可以發現，因為我們一開始有先對 pretrained model 進行一次 linear probing 之後才進到正式的訓練，所以在最一開始的時候兩個 pretrained model 就都有比較高的 accuracy。此外，即便經過了 10 個 epoch 的訓練，可以看見無論是 pretrained 還是 non-pretrained 的 model 在 testing accuracy 的部分，都與初使值相去不遠，兩者的差距只有一開始的 pretrain 與否而已，我認為這部分可能是因為我使用的 epoch 數不夠，再加上我使用了 weight decay 以及較大的 momentum，因此模型收斂的速度就變得很慢很慢，而使改變無法在 10 個 epoch 當中就出現。

	With pre-train	Without pre-train
ResNet18		
ResNet50		

從圖中我們可以看到在沒有無論是有沒有 pretrain 我們的模型只有在 0 的資料有較好的表現，只不過 pretrain model 在 0 以外的資料有更好的表現，而沒有 pretrain 的 model 幾乎把所有的資料都判別為 0。

而我推測這是跟我們的訓練資料相關，根據我的觀察，在訓練資料當中有超過 7 成的資料都是 label 為 0 的資料，因此才會造成這樣的結果，而因為 pretrain model 在接收到我們的訓練資料之前就曾經受過其他資料的訓練，因此不會受到這麼大程度的影響，因此在 0 以外的資料仍能夠有一定的表現。

4. Discussion

跟上一次 lab 相同，我們需要儲存一份 model，之後在 demo 時候將其 load 下來，然後把 testing data 餵給它，來看看這一份 model 的 accuracy 表現如何，而這次我發現在測試的時候得到的結果並不是一致的，而是每次都會有些許的變化，後來發現是因為再 RetinopathyLoader 當中，我們的__getitem__會將無論是 training 或者 testing 的 data 都加上了 RandomVerticalFlip 以及 RandomHorizontalFlip 才導致測試結果並不固定的情況，只要在__getitem__當中稍做修改就可以解決這個問題了。