Lab3 : Diabetic Retinopathy Detection

311511043 李承翰

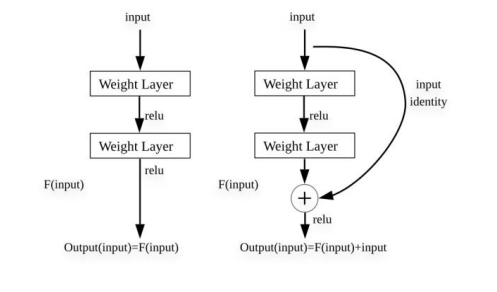
1. Introduction

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。

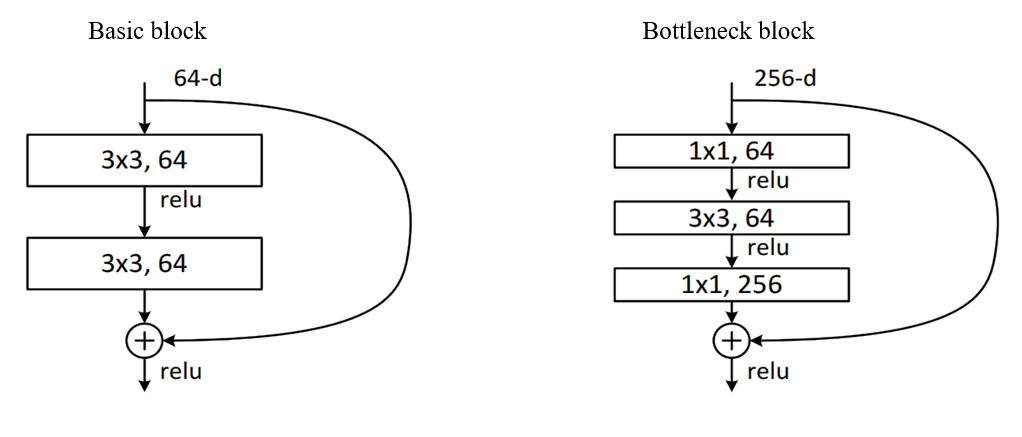
殘差映射(Residual Mapping) 就是在模型的訓練途徑當中曾加一條捷徑，讓output(x) = f(x) + x, 透過這樣一條捷徑，即使模型在這一層當中沒有學習到任何東西，她也能夠確定模型的下一層輸入剛好等於這一層的輸入，如此一來模型就不會退化，使模型能夠原來越深。



1. Experiment setup
2. 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。

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。

1. 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。

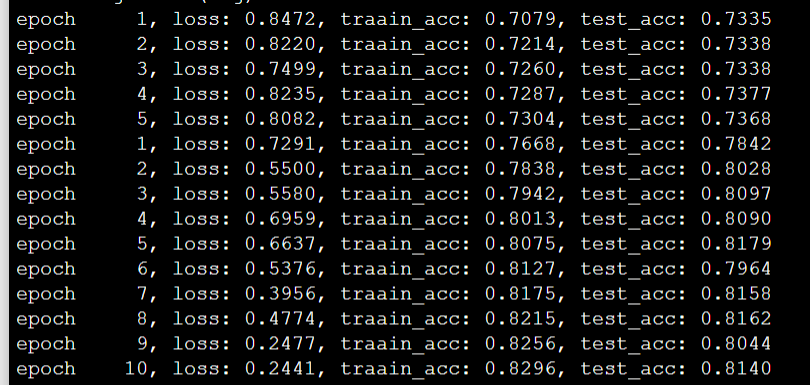
1. Describing your evaluation through the confusion matrix

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。

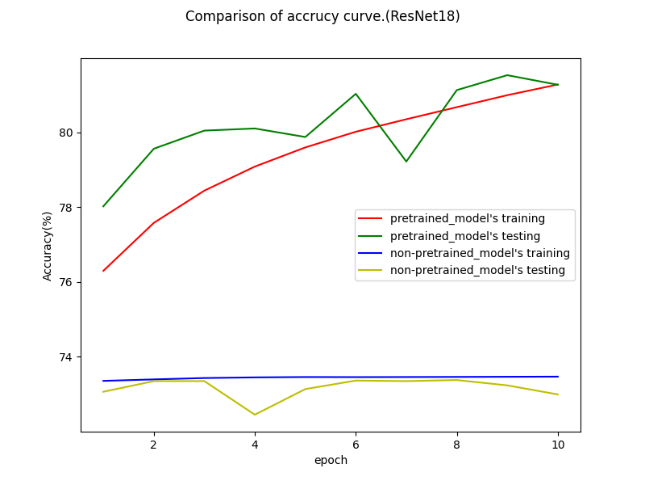
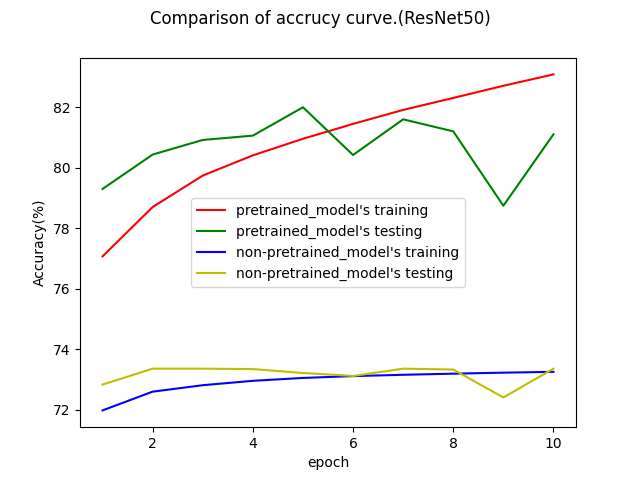
1. Experimental results
2. The highest testing accuracy:

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% |



1. 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。

1. 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。