Lab4-2 : Diabetic Retinopathy Detection

310605005 王映勻

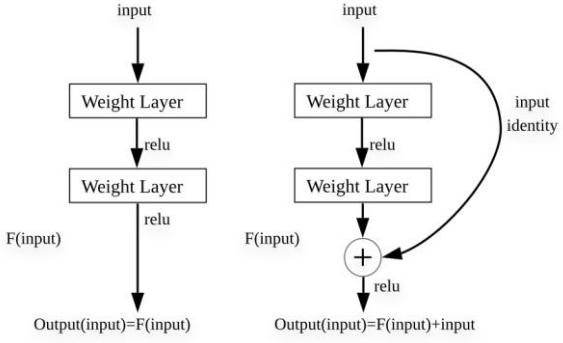
1. Introduction

這次 lab 要使用 pytorch 設計 DataLoader，利用 ResNet 網路架構分析糖尿病所引發視網膜病變的分類問題，比較有無 pretrained weight 分別在兩種 model 的表現，並透過 confusion matrix 評估結果。

Dataset 是使用 kaggle 的 Diabetic Retinopathy Detection，總共有 35124 張 512*512 的圖片，有五種 class label: 0~4，表示病變的嚴重程度。將其中 8 成的資料作為 training data，其餘 2 成當作 testing data。

在 ResNet 提出之前，太深的神經網路架構常出現退化(degradation)的問題，也就是隨著層數增加，模型準確率接近飽和且開始不穩定。殘差神經網路 ResNet (Residual Neural Network) 指的是在傳統卷積神經網路中加入殘差學習的思想，使得深層網路更容易訓練，也開啟了超深網路的時代。

ResNet 的關鍵「殘差映射」(Residual Mapping)，即是將模型要學習的目標從恆等映射函數 $H(x)$ 換成 $F(x) = H(x) - x$，也就是將目標轉為優化殘差，而不是直接學習恆等映射。下圖左邊為一般神經網路，右邊為 ResNet 的 Residual Block，可以發現 ResNet 只是多了一條捷徑(Shortcut connection)將輸入與輸出做相加，即 $H(x) = F(x) + x$。但是透過這樣做，即使模型多一層時沒學到任何東西 (殘差為 0)，也只會讓下一層輸入等於這層輸入 ($H(x) = x$)，不會讓模型退化，而實際上也不會剛好等於 0，因此每一層都可以學到一些新的更複雜的特徵，使模型能夠越來越深。



2. Experiment set up

A. The detail of your model (ResNet)

由於訓練卷積神經網路相當耗時，pytorch 中就有提供幾種常用的深度學習網絡架構以及其預訓練模型，如 VGG、ResNet 等，讓使用者不用每次都從隨機初始化參數開始訓練網絡。

這次實驗中用到的 ResNet-18 和 ResNet-50 就可以直接從 torchvision module 中 import，但原先網絡最後一層分類層 fc 是對 1000 種類型進行劃分，而針對我們的資料集則需要將 num_class 修改為 5。

要使用 pretrained model 進行 transfer learning 的話，將參數 pretrained 設置為 True 就會自動下載 torch.utils.model_zoo 提供的 pretrained model。用 pretrained model 先對所有資料提取特徵 (feature extraction)，透過設定’ param.requires_grad=False’ 將特徵層中參數固定住，使其在反向傳播的過程中不產生梯度的更新，只訓練最後一層全連接層，接著對模型進行 fine-tuning，訓練全部的 layer 數個 epoch 後得到最後結果。

B. The details of your Dataloader

要使用 pytorch 訓練 data，需要實做出 RetinopathyLoader，然後放進 torch.utils.data 提供的 DataLoader。RetinopathyLoader 繼承了 torch.utils.data 的 Dataset，主要覆寫__getitem__()的部分，讓 DataLoader 能找到 image 和 label，並在建構函式__init__()中對圖片做 transformation。

Transformation 包括 data augmentation 及 normalization，透過 transforms.Compose 將一系列的 transforms 的操作連結起來。經過測試，data augmentation 的部份使用 RandomHorizontalFlip(), RandomVerticalFlip()及 RandomRotation()，將資料做隨機的翻轉及旋轉，並利用 ToTensor()將 PIL.Image 數據進轉化為 torch.FloadTensor，將圖片的 shape 從 [H, W, C] 轉成 [C, H, W]，並把數值從 [0, 255] 轉換到 [0, 1.0]，最後對資料依照通道進行均值為 0，標準差為 1 的標準化，以加快模型收斂速

度。

transforms.Normalize()要輸入的參數為資料每個通道的均值及標準差，網路上很多人會使用 Imagenet 資料集的數值(mean=[0.485,0.456,0.406], std=[0.229,0.224,0.225])，但由於這次的資料集為較特殊的醫學影像，因此下面輸入的均值及標準差是取這次資料集其中七千張照片來計算而得。

```
transforms.Compose([transforms.RandomHorizontalFlip(),transforms.RandomVerticalFlip(),
                    transforms.RandomRotation(degrees=30),
                    transforms.ToTensor(),
                    transforms.Normalize((0.3775, 0.2618, 0.1873),(0.2918, 0.2088, 0.1684))])
```

C. Describing your evaluation through the confusion matrix

首先，計算出模型的預測結果，以一個 5*5 的 table 記錄並對其數值做正規化處理，使每一行加起來為 1，以比例的方式呈現。接著，利用 matplotlib.pyplot 函式庫視覺化 label 和 predict 的結果。

3. Experimental results

A. The highest testing accuracy

ResNet18：

resnet18_no_pretraining

	epoch	acc_train	acc_test
0	1	73.194776	72.953737
1	2	73.475924	73.266904
2	3	73.486601	73.352313
3	4	73.504395	71.644128
4	5	73.479483	73.295374
5	6	73.497277	73.366548
6	7	73.472366	73.366548
7	8	73.475924	73.153025
8	9	73.497277	73.281139
9	10	73.490160	73.352313
10	11	73.465248	73.409253
11	12	73.468807	73.380783
12	13	73.490160	73.338078
13	14	73.475924	73.338078
14	15	73.458130	73.380783
15	16	73.475924	73.295374
16	17	73.479483	73.238434
17	18	73.465248	73.395018
18	19	73.486601	73.096085
19	20	73.465248	73.423488

resnet18_with_pretraining

	epoch	acc_train	acc_test
0	1	73.063098	73.366548
1	2	73.646749	73.437722
2	3	73.742838	73.822064
3	4	73.792662	73.864769
4	5	73.963486	74.192171
5	1	75.910175	76.597865
6	2	78.575750	79.117438
7	3	79.650521	78.733096
8	4	80.173672	80.483986
9	5	80.807146	80.498221
10	6	81.049148	80.128114
11	7	81.387238	80.654804
12	8	81.746681	80.911032
13	9	82.084772	80.711744
14	10	82.380156	79.686833
15	11	82.462009	80.555160
16	12	82.881953	81.195730
17	13	82.821453	81.010676
18	14	82.978042	80.811388
19	15	83.166661	81.921708

```
model = ResNet18(num_class=5, pretrained=True)
model.load_state_dict(torch.load(os.path.join('result18','resnet18_with_pretraining.pt')))
model = model.to(device)

_, acc = evaluate(model, loader_test, device, num_class=5)
print(acc)
```

```
82.36298932384342
```

ResNet50 :

resnet50_no_pretraining

	epoch	acc_train	acc_test
0	1	72.813979	73.352313
1	2	73.184099	73.124555
2	3	73.319335	70.975089
3	4	73.315776	71.715302
4	5	73.422542	72.811388
5	6	73.486601	73.295374
6	7	73.440336	73.096085
7	8	73.479483	72.256228
8	9	73.468807	71.957295
9	10	73.451012	72.640569
10	11	73.486601	72.967972
11	12	73.454571	73.352313
12	13	73.443895	70.918149
13	14	73.433218	72.868327
14	15	73.483042	73.281139
15	16	73.486601	72.854093
16	17	73.479483	73.096085
17	18	73.493719	69.907473
18	19	73.486601	72.797153
19	20	73.493719	72.811388

resnet50_with_pretraining

	epoch	acc_train	acc_test
0	1	73.461689	73.836299
1	2	73.636072	73.594306
2	3	73.924339	74.419929
3	4	73.853162	73.950178
4	5	73.935015	74.078292
5	1	77.319478	78.704626
6	2	79.700345	80.768683
7	3	80.913912	81.338078
8	4	81.344532	82.007117
9	5	81.885476	79.274021
10	6	82.369479	81.935943
11	7	82.732482	81.935943
12	8	82.974483	82.220641
13	9	83.458486	80.227758
14	10	83.775223	82.419929
15	11	83.757429	82.548043
16	12	83.949607	81.793594
17	13	84.241432	82.690391
18	14	84.529699	81.722420
19	15	84.679170	82.078292

```
model = ResNet50(num_class=5, pretrained=True)
model.load_state_dict(torch.load(os.path.join('resnet50','resnet50_with_pretraining.pt')))
model = model.to(device)

_, acc = evaluate(model, loader_test, device, num_class=5)
print(acc)
```

```
82.76156583629893
```

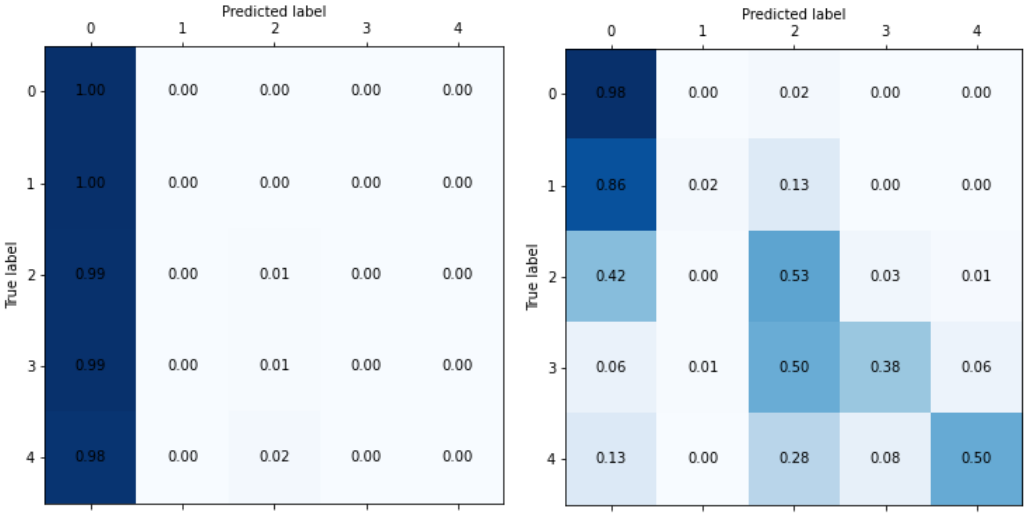
B. Comparison figures

ResNet18 :

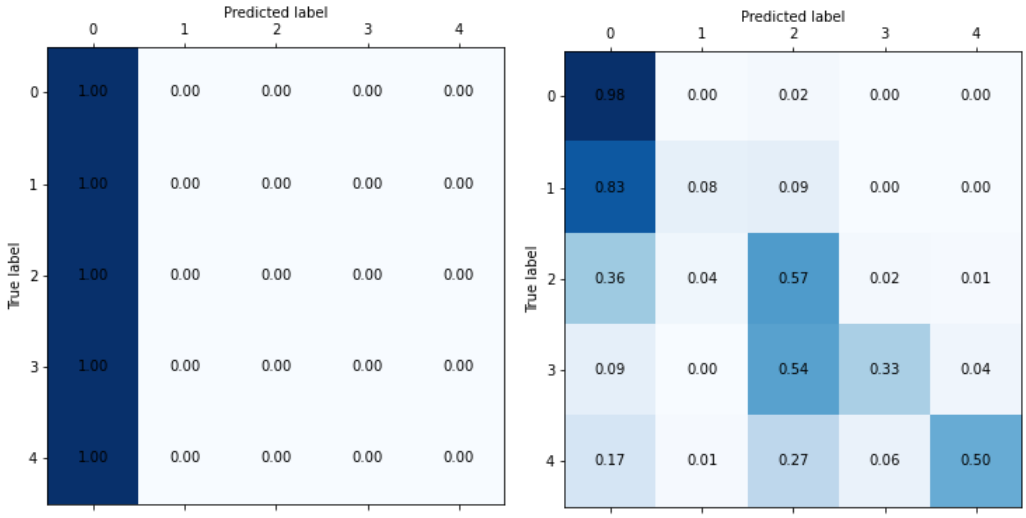
ResNet50 :

C. Confusion matrix

ResNet18 :



ResNet50 :



4. Discussion

(1) Transfer learning

從 confusion matrix 可以觀察到在沒有使用 pretrained model 的情況下，雖然正確率有百分之七十以上，但只是因為該資料集中 0 的類別占比特別高，因此即使全部都分到 0 的類別，也就是 0 以外的類別全都分類錯誤，正確率也不會太低。

而這樣過度不平衡的資料集，在沒有 pretrained model 的情況下模型訓練的 loss 很難降低，因此透過 transfer learning 的方式，將 pytorch 所提供有經過大量資料及訓練過的 pretrained model 繼承給這次任務模型，就可以有效降低訓練難度及時間，從實驗結果也可以明顯看出有使用 pretrained model 的模型表現較好。

(2) Different optimizer

在訓練過程中，嘗試調整 batch_size、learning rate 等參數，對結果影響都不大，雖然增加 fine-tuning 的 epoch 能增加 training 的表現，但 test 的部分效果也不顯著，可能是有 overfitting 的情況發生。因此就也嘗試不同的 optimizer，像是 Adam 及 RMSprop。

一開始使用的 Momentum SGD 是 SGD 的改良版，RMSprop 為其升級版，而 Adam 又為 RMSprop 的升級版。但經過實驗可發現 Adam 的效果和 Momentum SGD 差不多，甚至比 RMSprop 差一點，從這個結果中可以發現，理論上最好的 optimizer 不一定保證結果會最好，因此實驗最好嘗試不同的 optimizer，才能找到最適合該資料集的方法。

(3) 測試結果不穩定

將訓練完的模型權重儲存下來，利用 load_state_dict (torch.load(model_path)) 的方式載入，對測試資料集進行測試，發現計算出的正確率不太穩定。起初以為是 random seed 未固定的關係，但將相關參數固定後，仍有此問題，後來發現是在 dataloader 的部分，測試時不應

該對資料集做 data augmentation，拿掉這個部份後數值就有較穩定，但不曉得為何仍有些許差異，無法完全固定每次輸出。