Lab4: Diabetic Retinopathy Detection

張千祐
Department of Computer Science
National Yang Ming Chiao Tung University
qianyou.cs11@nycu.edu.tw

1 Introduction

在這次 lab 中，目標是希望從圖像的輸入，具體來說是左右眼視網膜的圖像，來做對於糖尿病所引發視網膜病變的分類，依據嚴重程度分爲五類，資料集來源是 kaggle，希望能夠透過在 2015 年發表的 Resnet 作爲預測模型，分別用 resnet18 和 resnet50，並且需要比較有無 pretraining 帶來的表現差異，過程中我們可以自行調整 hyper parameters 來取得更高的準確率，最後畫出準確率對 epoch 曲線來視覺化結果。

2 Experiment setups

2.1 The details of my models

我使用的是 resnet v1 的版本，唯獨需要在最後 fc layer 將 out_features 的值改成我們要預測的分類個數，也就是 5，根據 NVIDIA 提供的文檔指出：

> The difference between v1 and v1.5 is that, in the bottleneck blocks which requires downsampling, v1 has stride = 2 in the first 1x1 convolution, whereas v1.5 has stride = 2 in the 3x3 convolution.

可以看出兩者差別在於 bottleneck block，而文檔也指出這差異造成 resnet v1.5 比起 v1.0 有些微的預測準確率的進步，但由於我看到這資訊時時間已經來不及改用 v1.5 版本，因此在此作業還是使用 v1.0 版本的 resnet。

2.2 The details of my dataloader

2.2.1 ___init___

在這部分主要是初始化一些變數，像是根目錄位置、圖檔名稱等，另外我還使用 torchvision 裡的 transforms 定義了 transform 的流程，transform 主要作爲 data augmentation 的角色，目的是爲了產生更多樣化的訓練資料，在我的 transform 裡頭包含了 RandomVectivalFlip(), RandomHorizontalFlip(), RandomRotation(), ToTensor() 的轉化。

2.2.2 ___len___

很單純的回傳圖檔的長度，也就是在資料集裡總共有多少張圖檔。

2.2.3 ___getitem___

這邊則是要取得在 index 位置的圖檔，因此把路徑上的圖檔先經過前處理（前處理的部分在後面會說明），接著透過 init 定義的 transform 進行轉換，最後連著圖檔的 label 一起回講。

2.3 Hyper parameters

- Batch size=4
- Learning rate=1e-3
- Epochs=10(resnet18), 5(resnet50)
- Optimizer=SGD, Momentum=0.9, Weight_decay=5e-4
- Loss function=Cross Entropy Loss

```
class RetinopathyLoader(data.Dataset):
    def __init__(self, root, mode):
        """
        Args:
            root (string): Root path of the dataset.
            mode : Indicate procedure status(training or testing)

            self.img_name (string list): String list that store all image names.
            self.label (int or float list): Numerical list that store all ground truth label values.
        """
        self.root = root
        self.img_name, self.label = getData(mode)
        self.mode = mode
        self.transform = transforms.Compose([
            transforms.RandomVerticalFlip(),
            transforms.RandomHorizontalFlip(),
            transforms.RandomRotation(180),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.320, 0.223, 0.160], std=[0.304, 0.219, 0.175]),
        ])
        print("> Found %d images..." % (len(self.img_name)))

    def __len__(self):
        """'return the size of dataset"""
        return len(self.img_name)

    def __getitem__(self, index):
        """something you should implement here"""
        path = self.root + self.img_name[index] + '.jpeg'
        label = self.label[index]
        img = cv2.imread(path)

        # Filter out black background
        gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
        _,thresh = cv2.threshold(gray,10,255,cv2.THRESH_BINARY)

        # Find out the minimum rectangle that bounds the foreground
        x, y, w, h = cv2.boundingRect(thresh)
        img = img[y:y+h, x:x+w]
        s = max(img.shape[0:2])

        # Pad rectangle with black color so that it becomes square
        img = Image.fromarray(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        img = ImageOps.pad(img, (s, s), color='black')

        # resize to (512, 512)
        img = img.resize((512, 512))

        # transform
        img = self.transform(img)
        """
        step1. Get the image path from 'self.img_name' and load it.
```

```
                hint : path = root + self.img_name[index] + '.jpeg'

        step2. Get the ground truth label from self.label

        step3. Transform the .jpeg rgb images during the training phase, such as resizing, random flipping,
                rotation, cropping, normalization etc. But at the beginning, I suggest you follow the hints.

                In the testing phase, if you have a normalization process during the training phase, you only need
                to normalize the data.

                hints : Convert the pixel value to [0, 1]
                        Transpose the image shape from [H, W, C] to [C, H, W]

            step4. Return processed image and label
        """

        return img, label
```

Listing 1: Class RetinopathyLoader

2.4 Describe my evaluation through the confusion matrix

3 Data preprocessing

3.1 How did I preprocess my data?

3.1.1 Observation

首先是我觀察到資料集裡面的每張影像外圍都有多餘的黑色背景，所以第一個想法是希望篩掉多餘的黑色背景，使得模型的輸入有更少的值爲 0 的部分。第二是我觀察到每張影像未必是正方形，而模型的輸入兩個維度必須是一樣的，因此我會想要把矩形的短邊用黑色做 padding。最後是因□模型輸入維度必須是 (512, 512)，因此我會在最後一個步驟將圖片 resize 成規定的維度。

3.1.2 Steps

1. 一開始，我會先用 opencv 的 function 將 BGR 三色的影像輸入轉爲灰階影像
2. 接著用灰階影像篩出原影像當中只含有視網膜的矩形
3. 將矩形用黑色做 padding
4. Resize 成 (512, 512)

3.2 What makes my method special?

我認爲我的方法不一定是特別的，但是卻很直觀也很符合直覺，而且有幾項好處，第一是能夠維持 input image 長寬的比例，第二是使得資料值爲 0 的地方（對預測沒幫助）變少，第三是通用性，能夠適用於各種的大小的影像。

4 Experimental results

4.1 The highest testing accuracy

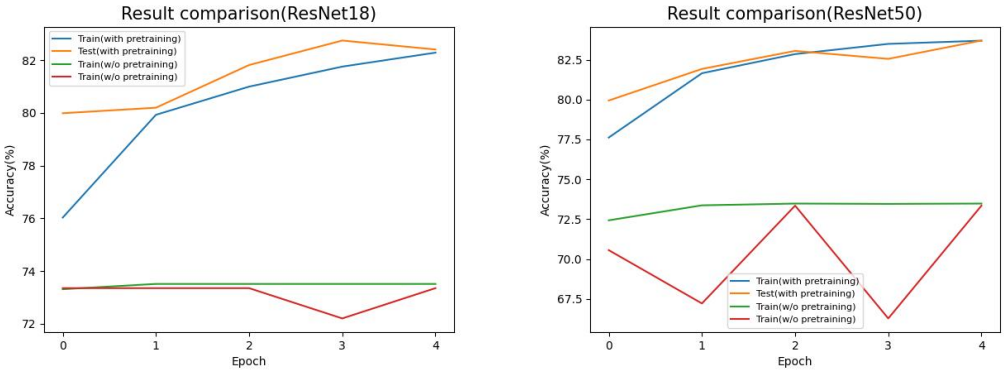
以最高 testing accuracy 來說，在沒有 pretraining 的時候 ResNet18 和 ResNet50 的表現是一樣的，而在有 pretraining 的時候，ResNet50 些微的比 ResNet18 高出一點。

	With pretraining	W/o pretraining
ResNet18	82.75%	73.35%
ResNet50	83.7%	73.35%

Table 1: Table of the highest testing accuracy

4.2 Comparison figures

從兩者的 learning curve 中可以發現，有 pretraining 的 model 預測準確率會穩定隨著訓練的 epoch 數量增加而提升，而沒有 pretraining 的則不然，會顯得較爲曲折。



(a) ResNet18 (b) ResNet50

Figure 1: Screenshot of comparison result

5 Discussion

5.1 With & Without pretraining

從實驗中可以發現有無 pretraining 對於預測準確率影響非常大，在 ResNet18 和 ResNet50 兩者都是差了快 10% 左右的準確率，因此在訓練時載入預訓練過的 weight 對於訓練速度和效果都是很顯著的提升。