Lab4 Diabetic Retinopathy Detection

邱以中

311551040

1. Introduction

在這次 lab4 中，我們的目標是要預測眼部的黃斑部病變，輸入會是眼睛的 image，輸出則是 5 個 label(0~4)，代表黃斑部病變的嚴重程度，我們要做的事就是根據給定的影像、label，訓練 resnet18, resnet50 (包含 pretrained)的 model 來進行正確的預測。

2. Experiment setups

A. The details of your model (ResNet)

- **Pretrained**

使用 torchvision 提供的 resnet18, resnet50 model, 並載入 pretrain 好的 weight，然後為了符合我們要預測的 5 個類別，會將原本最後一層的 FC 層替換掉，改成 output 為 5 的 FC 層

```
178  class ResNet18_pretrain(nn.Module):
179      def __init__(self,num_classes) -> None:
180          super().__init__()
181          self.model = resnet18(pretrained=True)
182          self.model.fc = nn.Linear(self.model.fc.in_features, num_classes)
183
184      def forward(self,x):
185          x = self.model(x)
186          return x
187
188  class ResNet50_pretrain(nn.Module):
189      def __init__(self,num_classes) -> None:
190          super().__init__()
191          resnet = resnet50(pretrained=True)
192          self.features = nn.Sequential(*list(resnet.children())[:-1])
193          self.fc = nn.Linear(resnet.fc.in_features, num_classes)
194
195      def forward(self,x):
196          x = self.features(x)
197          x = x.view(x.size(0), -1)
198          x = self.fc(x)
199          return x
```

- **Without pretrained**

在 without pretrain 的版本中，我則是嘗試手寫一個完整的 resnet 架構，在 resnet50 中使用 bottleneck block 來進行建構，在 resnet18 中使用 basic block 來進行建構，並將每個 block 的 input 與 output 進行相加來建構殘差網路，並且為了讓 intput 與 output 的 dimension 保持一致，在每個 block 當中都會使用 downsample 來檢查 input channel 與

output channel * expansion 是否有一致，若是沒有則是用 1x1 的 convolution 來達成 升維/降維 的操作。

Bottleneck block

```python
# resnet50 sub block
class BottleneckBlock(nn.Module):

    expansion = 4
    def __init__(self, in_channels, out_channels, stride=1):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.conv3 = nn.Conv2d(out_channels, out_channels*self.expansion, kernel_size=1, bias=False)
        self.bn3 = nn.BatchNorm2d(out_channels*self.expansion)
        self.relu = nn.ReLU(inplace=True)

        if stride != 1 or in_channels != out_channels*self.expansion:
            self.downsample = nn.Sequential(
                nn.Conv2d(in_channels, out_channels*self.expansion, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels*self.expansion)
            )
        else:
            self.downsample = nn.Identity()

    def forward(self,x):
        identity = x

        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)
        out = self.relu(out)

        out = self.conv3(out)
        out = self.bn3(out)

        identity = self.downsample(identity)

        out += identity
        out = self.relu(out)
        return out
```

Basic block

```python
# resnet18 sub block
class BasicBlock(nn.Module):

    expansion = 1
    def __init__(self, in_channels, out_channels, stride=1):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)

        if stride != 1 or in_channels != out_channels:
            self.downsample = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels)
            )
        else:
            self.downsample = nn.Identity()

    def forward(self,x):
        identity = x

        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)

        identity = self.downsample(identity)

        out += identity
        out = self.relu(out)

        return out
```

B. The details of your Dataloader

在 init 階段，我們會根據不同的 mode，讀取相對應的 csv 檔，取得 training/testing 的 image 檔名與其對應的 label

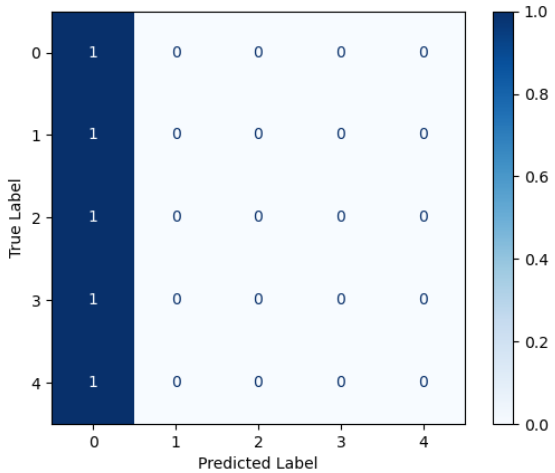
```python

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
            self.label (int or float list): Numerical list that stor
        """
        self.root = root
        self.img_name, self.label = getData(mode)
        self.mode = mode

```

在 getitem 這個 function 中，我們會定義如何讀取資料(image,label)，我們會根據 index 取得 image 檔名與 label ，並根據檔名讀取 image，之後再對 image 做 transform 然後回傳。

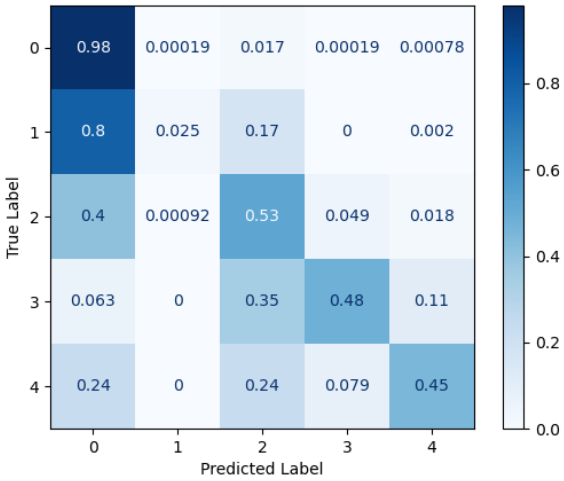
```
        """
        path = os.path.join(self.root, f'{self.img_name[index]}.jpeg')
        img = Image.open(path)
        width,height = img.size
        if self.mode == "train":
            transform = transforms.Compose([
                    # transforms.CenterCrop((height,height)),
                    transforms.RandomHorizontalFlip(0.5),
                    transforms.RandomVerticalFlip(0.5),
                    # transforms.Resize((512, 512)),
                    transforms.ToTensor()
                ])

        elif self.mode == "test":
            transform = transforms.Compose([
                    # transforms.CenterCrop((height,height)),
                    # transforms.Resize((512, 512)),
                    transforms.ToTensor()
                ])

        img = transform(img)
        label = self.label[index]


        return img, label
```

C. Describing your evaluation through the confusion matrix

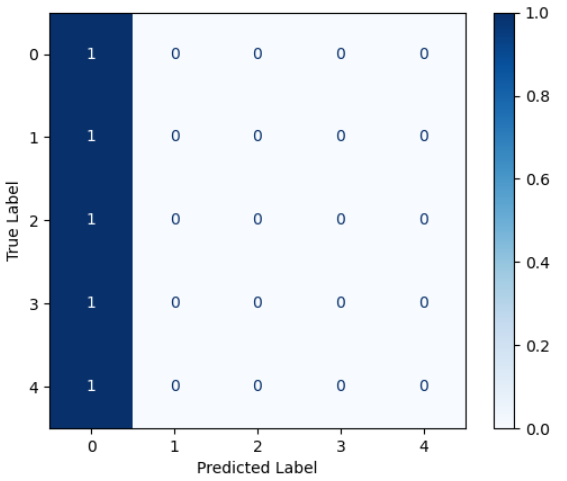
- **Resnet18**



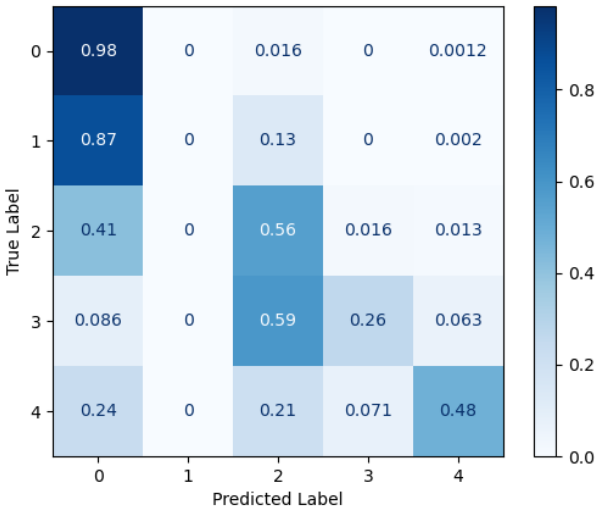
- **Resnet18 pretrained**



- **Resnet50**



- **Resnet50 pretrained**



由 confusion matrix 的結果可以發現，如果沒有 pretrain，在短短的幾個 epoch 中 model 其實沒辦法學得很好，幾乎對於所有輸入

model 都傾向輸出 label 0。而在 pretrained 的版本中，model 則是能較好的預測出不同的結果，不過可以發現在 predict 的結果中幾乎都沒有 label1，推測是因為 label1 的資料太少，所有 model 沒有辦法很好的進行學習。

3. Data Preprocessing

A. How you preprocessed your data?

在對 image 做 preprocess 時，我們會對 image 做 transform，對 image 做 center crop 去除周圍的黑色背景，保留重要的區域，將所有 image resize 到同樣大小，並在最後使用 totensor 將 image 轉換成 tensor 型態。除此之外，在 training 時我也有使用 random flip 來做資料增強。

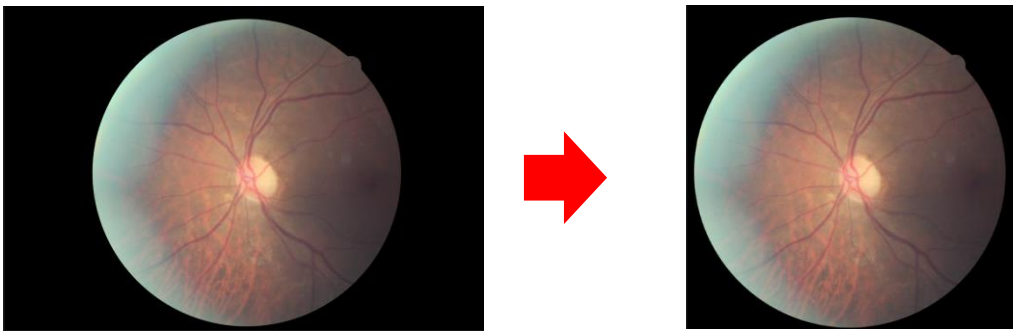
值得一提的是，在讀取資料時，我也有發現在 training image 中有資料損毀的情況發生，導致在讀取資料階段發生錯誤，因此我有事先找出資料損毀的圖片，並將他們的圖片與對應的 csv 資料進行刪除，就我發現的有這三張圖片。

```
21    # 38790 right
22    # 29126 right
23    # 8421 left
24
```

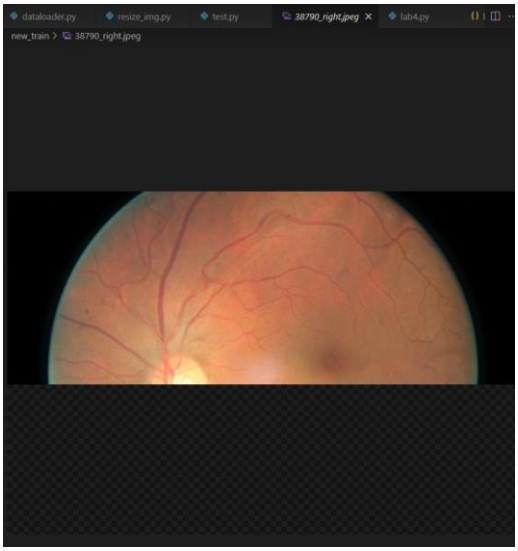
B. What makes your method special?

- 使用 center crop 讓 model 能關注在圖片中的主要部分，center crop 的大小設置為原圖片的 height 能最大程度保留眼睛的部分。
- 在 training 階段使用 random flip 來增加 model 的泛化能力，避免 overfitting。
- 事先去除損毀的圖片，來讓資料能正確讀取

經過 preprocess 的圖片



資料損毀的圖片



4. Experiment results

A. The highest testing accuracy

當我使用 pretrained 的 resnet18 時能達到最高 82.32 的 testing accuracy

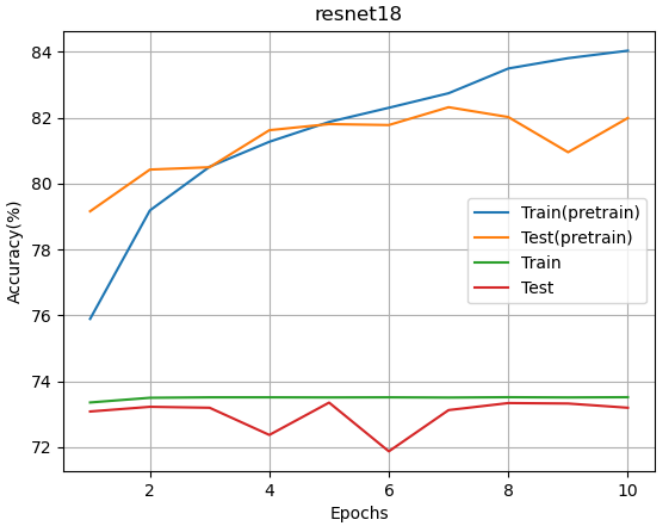
Hyper Parameters

- Batch size= 12
- Learning rate = 1e-3
- Epochs for ResNet-18 = 10
- Optimizer: SGD
- Momentum = 0.9
- Weight Decay = 5e-4
- Loss function: Cross Entropy Loss

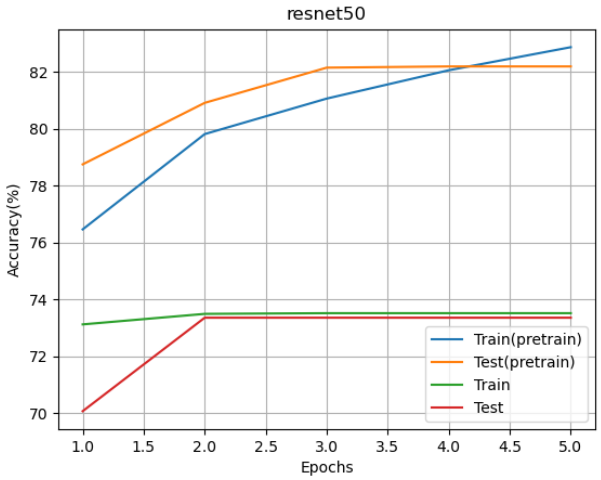
```
Epoch 6
train accuracy: 82.30353075170842
test accuracy: 81.77935943060498
Epoch 7
train accuracy: 82.74487471526196
test accuracy: 82.3202846975089
Epoch 8
train accuracy: 83.49587129840546
test accuracy: 82.02135231316726
Epoch 9
train accuracy: 83.80908314350798
test accuracy: 80.95373665480427
Epoch 10
train accuracy: 84.0368735763098
test accuracy: 81.99288256227759

python lab4.py --model resnet18_pretrain --epochs 10 --batch-size 12 --lr 0.001
```

B. Comparison figures

- Resnet18



- Resnet50



5. Discussion

A. 在這次的資料中，有發現資料損毀的情況發生，因為一開始沒有做特別的處理，所以導致一開始在實驗時有遇到一些困難，之後再做資料前處理時，我們應該提前想到可能有類似的情況發生，並且進行特殊的處理，來讓實驗能順利的進行。

B. 在這次的實驗中，我有發現每一個 epoch 花的時間非常的久，一開始光一個 epoch 都要花到 9~10 hr，原本以為是 model 的問題，後來發現在調整 dataloader 中的 num_worker 數量後時間有明顯的下降，所以由此得知原來 training 時間的 bottle neck 是在 cpu 讀取資料的速度上，而不是 gpu。但是就算我把 num_worker 調到 4，一個 epoch 還是需要花到 1~2 hr，而就我觀察的原因還是在於 image 太大了，一個 image 的 resolution 差不多為 2000*2000。後來我就想到，既然每次 transform

都要做 center crop 跟 resize，那何不一開始就對所有 image 做前處理並且存起來，這樣讀取的 image 就是 resize 後的，速度也應該會加快很多。因此我最後的做法是提前對所有 image 做 preprocess，並儲存到另一個資料夾當中，而經過這樣處理過後，我發現每個 training 的 epoch 大約只需要花上 10 多分鐘，大幅降低了訓練的時間。

C. 在做 resize 時，我也有嘗試不同的 resolution，一開始我是 resize 到 256x256，不過發現 accuracy 一直無法上升，後來改成 512x512 得到的 accuracy 就有比較高，由此可知 image 的 size 對於 model 的訓練以及預測還是有很大的影響。