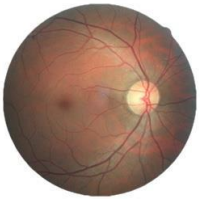
Deep Learning and Practice

#Lab04 Diabetic Retinopathy Detection　　309505002 鄭紹文

1. Introduction：

此次 Lab 中使用 ResNet18 和 ResNet50 兩種架構來實現糖尿病性視網膜病變(Diabetic Retinopathy)的分類，而其分成以下五個類別：

1. No DR
2. Mild
3. Moderate
4. Severe
5. Proliferative DR

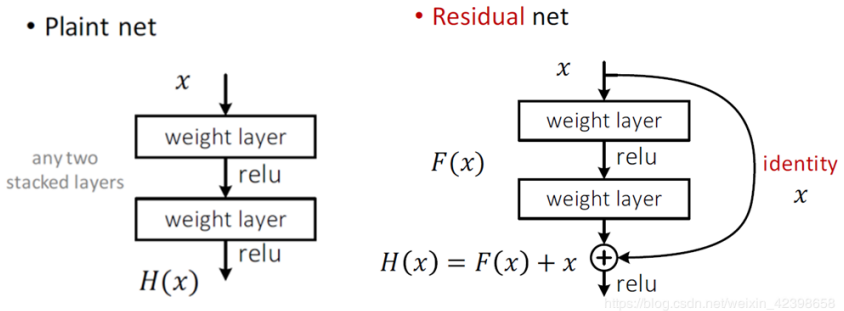


Dataset 中總共有 28099 張 train data 和 7025 張 test data，大小是 512 X 512 的 RGB.jpeg 圖片(W=512, H=512, C=3)，因為這次不同上次有先做 preprocessing，這次需要使用自定義的 DataLoader 來讀取資料，將其轉換成需要的格式訓練，並且比較有使用 pretrained model 以及沒有使用 pretrained model 的準確率，然後將結果用可視覺化的方式呈現，再計算 confusion matrix。

2. Experiment setups：

A. The details of model (ResNet)

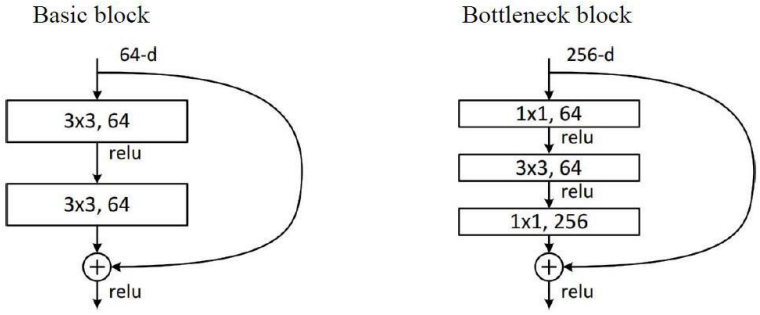
ResNet 出來前，當時的網路相較於現在都是非常淺的設計，原因在於當時較深的網路比較容易訓練不起來，有時會 overfitting，有時會在執行 back propagation 時， gradient 在多次相乘後趨近於 0(若 gradient < 1)，aka gradient vanish problem，導致 network 加深後反而導致效能變差。而 ResNet 提出的 residual learning 簡單地使得深層網路更容易訓練，也開啟了超深網路的時代。



ResNet運用Shortcut connection概念解決上述的問題，透過shortcut可將training goal更改為H(x) - x=F(x)。上圖中的weight layer可以先把想像成是convolution layer，同時可知對於一個layer structures，當輸入為x時，學習到的特徵記為H(x)，運用此概念希望將殘差學習成F(x) = H(x) – x，此時學習特徵將會變為F(x)+x。這樣做法是因為殘差學習相比原始特徵較來的容易學習，當殘差為0時，等同於做恆等

映射(Identity mapping)，此時網路的性能不會下降，實際上殘差不大會變為0，使得堆積層在輸入特徵基礎上學習到新的特徵，進而有更好的性能。直觀上可以想像ResNet網絡就能從旁邊的捷徑得到另外一種選擇，當今天發生了因為網路過深而梯度消失時，代表說網路已經失去作用，那即可F(x)當作是0，而此時結果就是Net = x，而不會造成不好的影響，所以ResNet最後的結果只會比傳統網路好，就算是最差的情況也頂多是效果一樣好而不會比較差。所以在ResNet的想法出現之後，網路也不會再受到過深的限制，而可以有越來越深的網路出現。

此次ResNet要做兩種，分別是ResNet18和ResNet50，兩者分別是用basic block和bottleneck block組成。




```python
# Bottleneck：三個卷基層分別是1x1，3x3，1x1，分別用來降低維度，卷積處理，提高維度
# 目的是 -> 減少參數數量，Bottleneck相比較BasicBlock在參數的數目上較少但經度差不多

class BottleneckBlock(nn.Module):
    '''
    x = (in, H, W) -> conv2d(1x1) -> conv2d -> (out, H, W) -> conv2d(1x1) -> (out*4, H, W) + x
    '''
    expansion = 4

    def __init__(self, in_channels, out_channels, stride=1, kernel_size=3, downsample=None):
        super(BottleneckBlock, self).__init__()
        padding = int(kernel_size/2)
        self.activation = nn.ReLU(inplace=True)
        self.block = nn.Sequential(
            # 1x1 的卷積是為了降維，減少通道數
            nn.Conv2d(in_channels, out_channels, kernel_size=1, bias=False),
            nn.BatchNorm2d(out_channels),
            self.activation,
            # 3x3 的卷積是為了改變圖片大小，不改變通道數
            nn.Conv2d(
                out_channels, out_channels,
                kernel_size=kernel_size, stride=stride, padding=padding, bias=False
            ),
            nn.BatchNorm2d(out_channels),
            self.activation,

             # 1x1 的卷積是為了升維，增加通道數，增加到 planes * 4
            nn.Conv2d(out_channels, out_channels * self.expansion, kernel_size=1, bias=False),
            nn.BatchNorm2d(out_channels * self.expansion),
        )
        self.downsample = downsample

    def forward(self, x):
        residual = x
        out = self.block(x)

        #若上一個Residual Block的輸出維度和當前維度不同，則對這個x進行downsample，若維度依樣則直接網間(out += residual)
        if self.downsample is not None:
            residual = self.downsample(x)

        out += residual
        out = self.activation(out)

        return out
```

```python
# 兩個 3*3 前後的維度相同
class BasicBlock(nn.Module):
    '''
    x = (in, H, W) -> conv2d -> (out, H, W) -> conv2d -> (out, H, W) + x
    '''
    expansion = 1

    def __init__(self, in_channels, out_channels, stride=1, kernel_size=3, downsample=None):
        super(BasicBlock, self).__init__()
        padding = int(kernel_size/2)
        self.activation = nn.ReLU(inplace=True)
        self.block = nn.Sequential(
            nn.Conv2d(
                in_channels, out_channels,
                kernel_size=kernel_size, padding=padding, stride=stride, bias=False
            ),
            nn.BatchNorm2d(out_channels),
            self.activation,

            nn.Conv2d(
                out_channels, out_channels,
                kernel_size=kernel_size, padding=padding, bias=False
            ),
            nn.BatchNorm2d(out_channels),
        )
        self.downsample = downsample

    def forward(self, x):
        residual = x
        #x 賦值給 residual，用於後面的 shortcut 連線
        out = self.block(x)

        if self.downsample is not None:
            #遇到降維或升維時要保證能夠相加
            residual = self.downsample(x)

        out += residual
        out = self.activation(out)

        return out

```

B. The details of Dataloader：

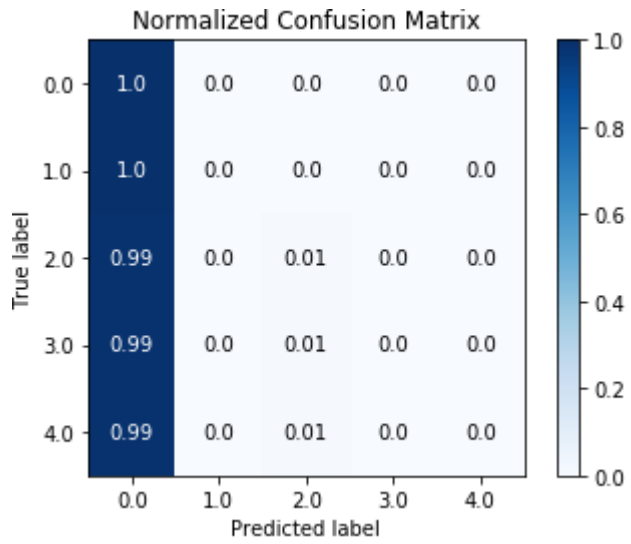
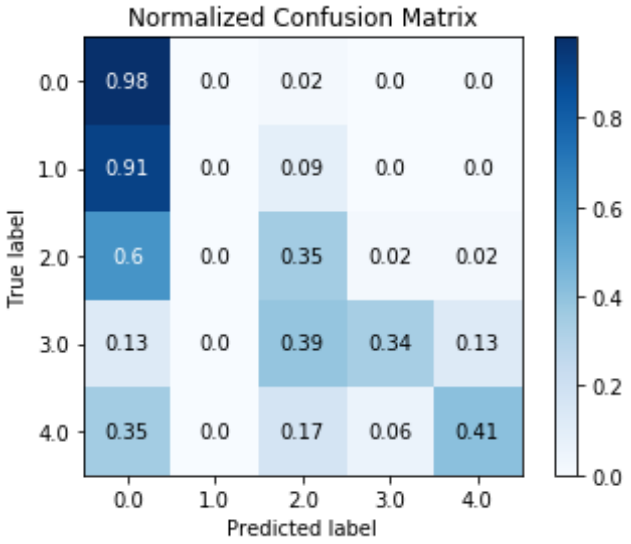
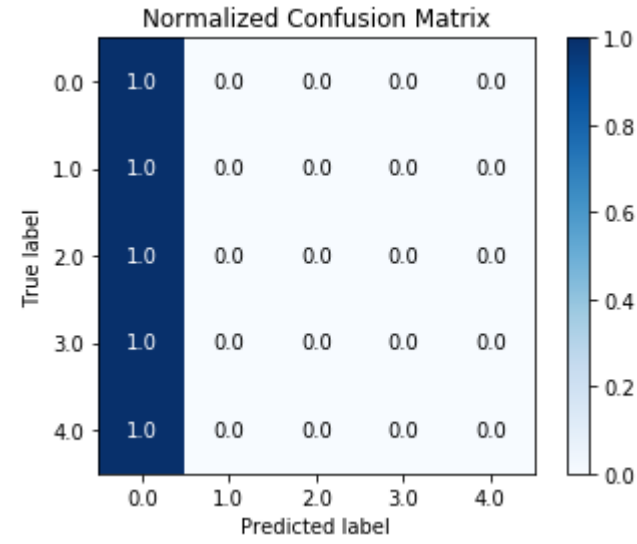
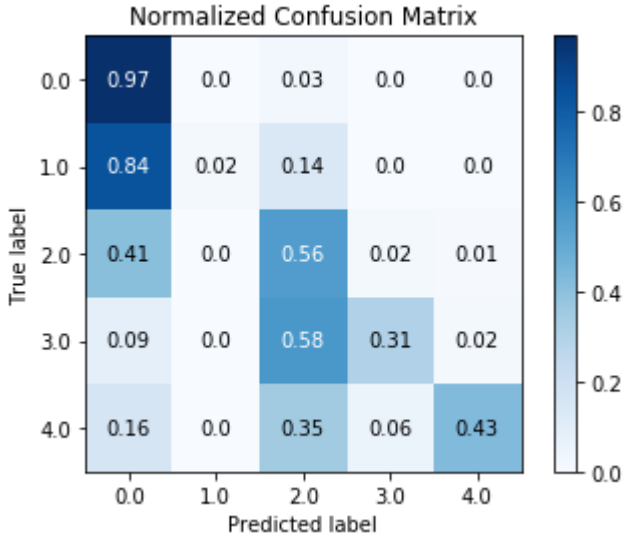
因為PyTorch中convolution layer需要的圖片格式和用PIL.Image.open讀進來的格式不太一樣，所以使用torchvision.transforms來做轉換*([H, W, C] to [C, H, W])*。Class RetinopathyLoader中__getitem__的部分，首先要告訴Dataloader資料放在何處，並且要在讀取後進行何種操作，圖中顯示讀進來後對data image做to.Tensor()再做Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])，這部分是因為之後用到pretrained model，其透過查閱pytorch官方資料可以發現該pretrained model是建立在經過此normalize的data set上。為了增進效果，在讀檔時額外做將原本資料水平翻轉、垂直翻轉來增加train data的數量以提高準確率。

Load Data

```python
train_dataset = dataloader.RetinopathyLoader('./data', 'train')
test_dataset  = dataloader.RetinopathyLoader('./data', 'test')

augmentation = [
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
]
print("> dataset with augmentation with{}".format(augmentation))

train_dataset_with_augementation = dataloader.RetinopathyLoader('./data', 'train', augmentation=augmentation)
```

```python
def getData(mode):
    if mode == 'train':
        img = pd.read_csv('train_img.csv', header=None)
        label = pd.read_csv('train_label.csv', header=None)
        return np.squeeze(img.values), np.squeeze(label.values)
    else:
        img = pd.read_csv('test_img.csv', header=None)
        label = pd.read_csv('test_label.csv', header=None)
        return np.squeeze(img.values), np.squeeze(label.values)


class RetinopathyLoader(data.Dataset):
    def __init__(self, root, mode, augmentation=None):
        """
        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.mode = mode
        self.img_name, self.label = getData(mode)

        trans = []
        if augmentation:
            trans += augmentation
        trans += [transforms.ToTensor(),
                  transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])]
        self.transforms = transforms.Compose(trans)


        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"""

        """
            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
        """

        # path = self.root + self.img_name[index] + '.jpeg'
        path = os.path.join(self.root, self.img_name[index] + '.jpeg')

        img = Image.open(path)
        img = self.transforms(img)

        label = self.label[index]

        return img, label
```

C. Describing your evaluation through the confusion matrix：

ResNet18 without pretrained *(epoch：10, batch_size：20)*	**ResNet18 with pretrained** *(epoch：10, batch_size：20)*
	
ResNet50 without pretrained *(epoch：5, batch_size：10)*	**ResNet50 with pretrained** *(epoch：5, batch_size：10)*
	

透過表格可發現 pretrained model 的版本，分類 class0 的準確率較高，但再分類 class1 的部分最差，可能 class 1 沒有很強烈的特徵好做分辨，可能還需要做其他的處理，其餘 class 表現的挺普通的。至於為何沒有 pretrained 的版本長得很怪，這部分是我蠻大的疑問。

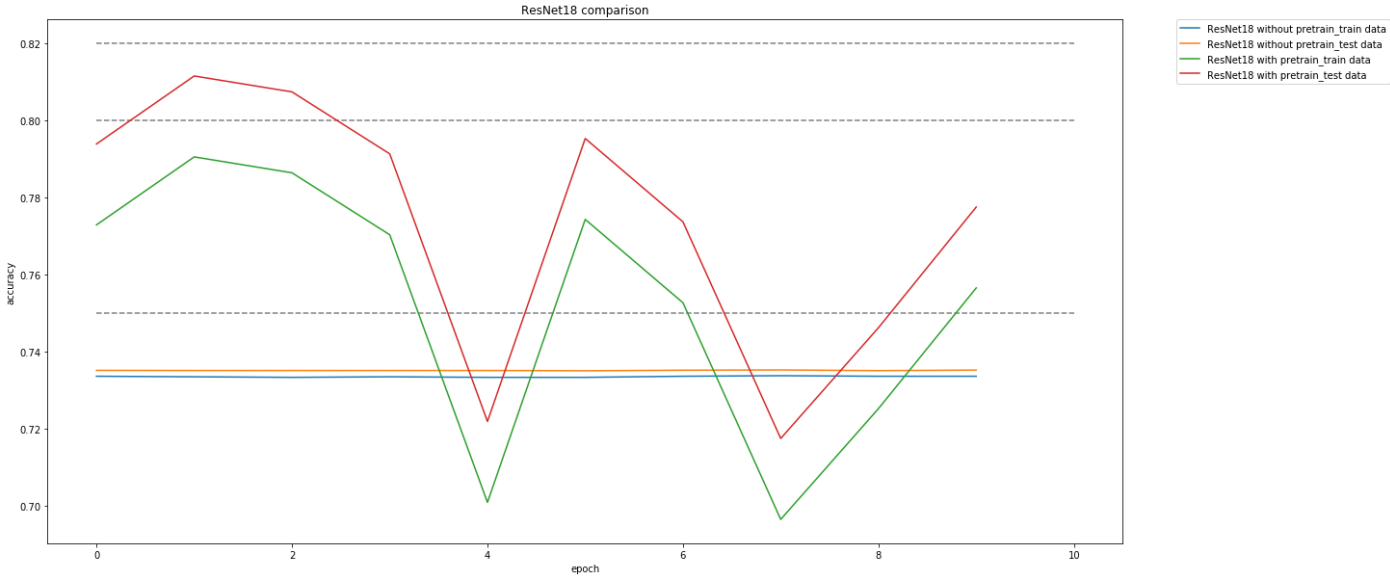
3. Experimental Results：

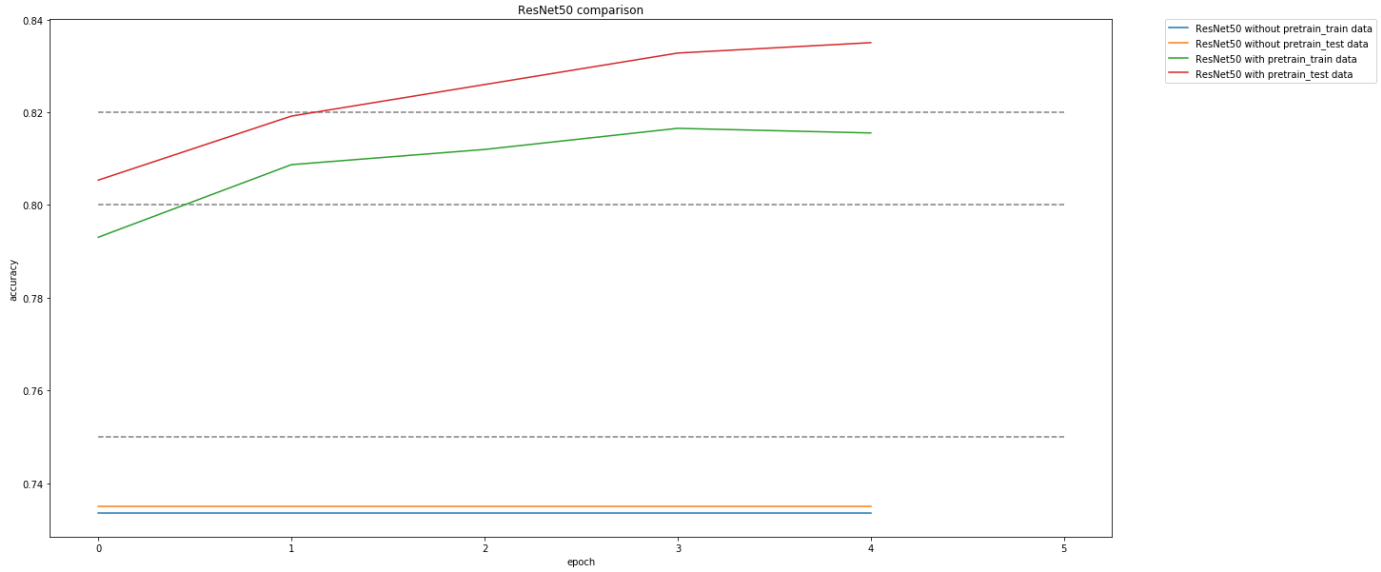
A. The highest testing accuracy：

ResNet18, ResNet50 Highest result
> ResNet18 with epoch:10, batch_size:20
> ResNet50 with epoch:05, batch_size:10

Net	without pretrained	with pretrained
ResNet18	0.734	0.790
ResNet50	0.734	0.835

B. Comparison figures：





由圖可以發現，ResNet50 的表現比 ResNet18 來的好，兩個 with pretrained 的也都比沒有 pretrained 來的優異，我想應該是因為 model 很大，需要夠多的 epoch 才 train 的成功，而相反從 train 好的 model 上做修改則能在較少的 epochs 上拿到較好的成果。

4. Discussion：

- **Optimizer Choose：**

  optimizer 的選擇很重要，起初嘗試使用 Adam，發現始終沒法讓 accuracy 上升，後來改成 SGD 才順利有進步，且在 SGD 選擇適當的 momentum 和 weight decay 都至關重要。

- **Backpropagation看ResNet的有效性：**

  BasicBlock結構使得backpropagation時，更不容易出現梯度消失的問題，其緣由於有Skip Connection的存在，梯度能輕鬆的通過各個Res blocks。

$$\mathbf{y}_l = h(\mathbf{x}_l) + \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l)$$ 函數f表示一個残差函數

$$\mathbf{x}_L = \mathbf{x}_l + \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathcal{W}_i),$$

為經過多次遞迴後得到第L層的表示式

$$\frac{\partial \mathcal{E}}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\left(1 + \frac{\partial}{\partial \mathbf{x}_l}\sum_{i=l}^{L-1}\mathcal{F}(\mathbf{x}_i, \mathcal{W}_i)\right).$$

上述為backpropagation第I層梯度，從上圖的式子中可以看出：第l層的梯度中，包含了第L層的梯度，簡而言之即第L層的梯度直接經過了Skip Connection傳送給第l層，而因為梯度消失問題主要是發生在淺層，現在因為有了Skip Connection可以直接把深層梯度傳送給淺層的方法，可以很有效的解決梯度消失的問題。