DLP HW4

309553008

1. Introduction

這次作業在實作 ResNet18 與 ResNet50，然後分別測試有無使用 pretrained model 的情況正確率會有何差異，input 是 512*512 的眼球照片，以此照片來判斷該眼睛的主人糖尿病嚴重性(5 個 class)

2. Experiment

- Detail of your model

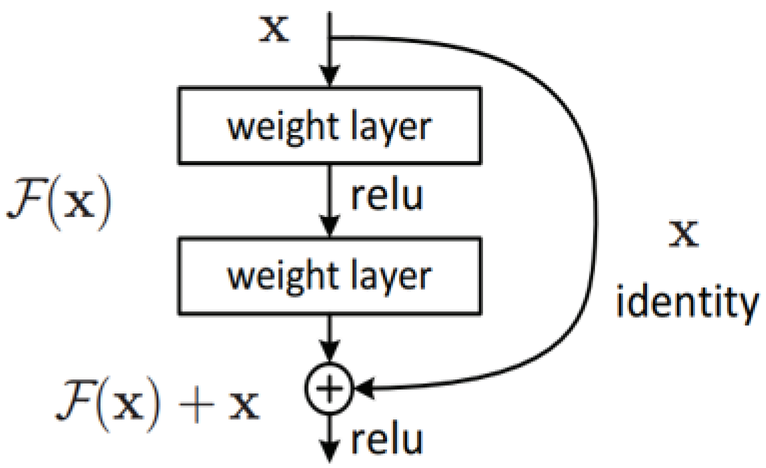
  這是 ResNet18 裡的 Residual Block，本來我想把 downsampling 寫在這一塊，畢竟是在 Residual Block 裡發生的事，但是維度一直弄錯(雖然最後發現不是因為這裡的問題，但是已經改了架構就不改回去了)

```python
class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, in_channel, out_channel, kernel_size=3, stride=1, downsampling=None):
        super(BasicBlock, self).__init__()
        padding = int(kernel_size/2)
        self.activation = nn.ReLU(inplace=True)
        self.block = nn.Sequential(
            nn.Conv2d(in_channel, out_channel, kernel_size=kernel_size, padding=padding,stride=stride, bias=False),
            nn.BatchNorm2d(out_channel),
            self.activation,
            nn.Conv2d(out_channel, out_channel, kernel_size=kernel_size, padding=padding, bias=False),
            nn.BatchNorm2d(out_channel)
        )
        self.downsampling = downsampling
        # downsampling
        # self.shortcut = nn.Sequential()
        # if stride != 1 or in_channel != out_channel:
        #     self.shortcut = nn.Sequential(
        #         nn.Conv2d(in_channel, out_channel, kernel_size=1, stride=stride, bias=False),
        #         nn.BatchNorm2d(out_channel)
        #     )

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

        if self.downsampling is not None:
            input_x = self.downsampling(x)

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

        return out
```

這是 ResNet50 裡的 Residual Block

```python
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, downsampling=None):
        super(BottleneckBlock, 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=1, bias=False),
            nn.BatchNorm2d(out_channels),
            self.activation,
            nn.Conv2d(out_channels, out_channels, kernel_size=kernel_size, stride=stride, padding=padding, bias=False),
            nn.BatchNorm2d(out_channels),
            self.activation,
            nn.Conv2d(out_channels, out_channels * self.expansion, kernel_size=1, bias=False),
            nn.BatchNorm2d(out_channels * self.expansion),
        )
        self.downsampling = downsampling
        # downsampling
        # self.shortcut = nn.Sequential()
        # if stride != 1 or in_channel != out_channel*self.expansion:
        #     self.shortcut = nn.Sequential(
        #         nn.Conv2d(in_channel, out_channel*self.expansion, kernel_size=1, stride=stride, bias=False),
        #         nn.BatchNorm2d(outchannel=self.expansion)
        #     )

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

        if self.downsampling is not None:
            input_x = self.downsampling(x)

        out += input_x
        out = F.relu(out)

        return out
```

2 個最重要的地方都是(forward)一開始 input 會先存下來，然後跟最後經過 Residual Block 後的 output 加在一起，如圖



至於為了要讓維度一樣，可能一開始的 input 會經過一個 convolution 跟一個 batch Normalize 做 down sampling。

至於除了 Residual Block 以外 ResNet 都長得一樣(不管 18、50 還是 152)，所

以我看大家都會把 Residual Block 抓出來寫，其他的地方會長這樣

```python
class ResNet(nn.Module):
    def __init__(self, block, layers, num_classes=5, start_in_channels=64):
        super(ResNet, self).__init__()

        self.current_in_channels = start_in_channels
        self.first = nn.Sequential(
            nn.Conv2d(3, self.current_in_channels, kernel_size=7, stride=2, padding=3, bias=False),
            nn.BatchNorm2d(self.current_in_channels),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2, padding=1),
        )
        self.layers = layers
        channels = self.current_in_channels
        for i, l in enumerate(layers):
            setattr(self, 'layer'+str(i+1),
                    self._make_layer(block, channels, l, stride=(2 if i!=0 else 1) ))
            channels*=2

        self.avgpool = nn.AdaptiveAvgPool2d(1)
        self.fc = nn.Linear(self.current_in_channels, num_classes)

    def _make_layer(self, block, in_channels, blocks, stride=1):
        downsampling=None
        if stride != 1 or self.current_in_channels != in_channels * block.expansion:
            downsampling = nn.Sequential(
                nn.Conv2d(self.current_in_channels, in_channels * block.expansion, kernel_size = 1, stride=stride, bias=False),
                nn.BatchNorm2d(in_channels * block.expansion)
            )

        layers = []
        layers.append(block(self.current_in_channels, in_channels, stride=stride, downsampling=downsampling))
        self.current_in_channels = in_channels * block.expansion
        for i in range(1, blocks):
            layers.append(block(self.current_in_channels, in_channels))

        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.first(x)
        for i in range(len(self.layers)):
            x = getattr(self, 'layer'+str(i+1))(x)
        x = self.avgpool(x)
        # flatten
        x = x.view(x.size(0), -1)
        x = self.fc(x)

        return x
```

下面這張圖有很好的架構圖

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

以 ResNet18 為例，照著架，值得一提的是會需要 down sampling 的時候都是 stride 不是 1(所以出來後 size 會對不上)或者餵進來的 in/output 本來就不一樣，所以可以很簡單的用一行 if 來判斷

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )

  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )

  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )

  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AvgPool2d(kernel_size=7, stride=1, padding=0)
  (fc): Linear(in_features=512, out_features=1000, bias=True)
```

- Detail of your dataloader

  Transforms.Compose 可以將一連串的 transform 包在一起像這邊就是做了 Flip，如果都不需要操作的話可以像被註解掉的那行直接用 transforms.ToTensor 轉成 tensor 就能丟進 pytorch 自己的 dataloader 了

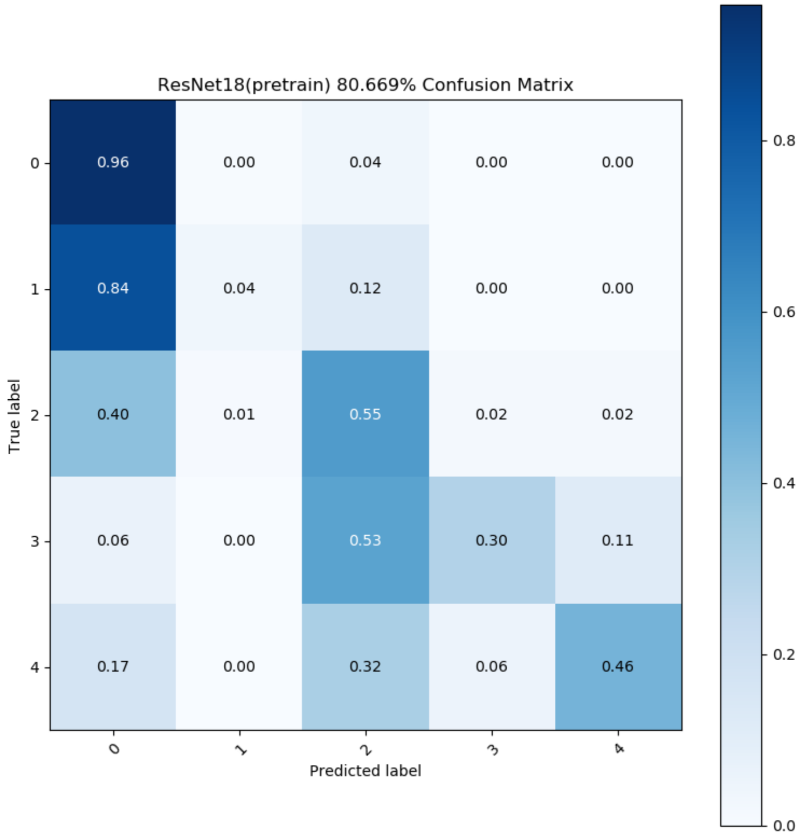
```python
lass RetinopathyLoader(data.Dataset):
    def __init__(self, root, mode):
        """
        Args:
            root (string): Root path of the dataset.
            mode : Indicate procedure status(training or testing)

            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.transforms = transforms.Compose([transforms.RandomHorizontalFlip(p=0.5), transforms.RandomVerticalFlip(p=0.5), transforms.ToTensor()])
        #self.transforms = transforms.ToTensor()
        print("> Found %d images..." % (len(self.img_name)))
```

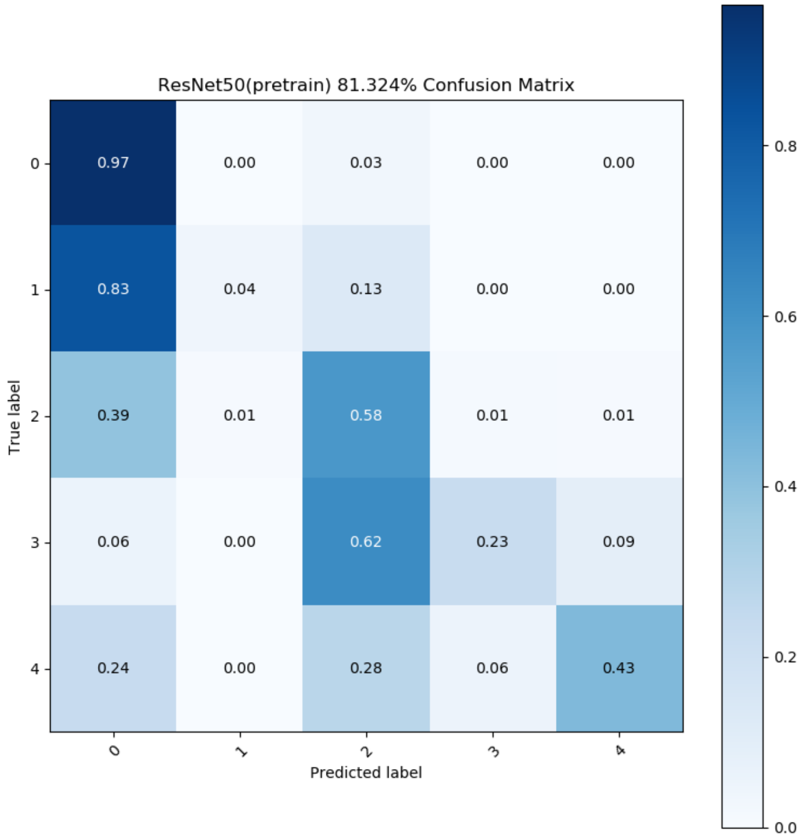
這邊將圖&label 讀進來，如果要做什麼操作的話可以寫進 Compose 裡再一起做 transform

```python
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 = os.path.join(self.root, self.img_name[index] + '.jpeg')
    img = PIL.Image.open(path)

    img = self.transforms(img)
    label = self.label[index]
    return img, label
```

- Describing your evaluation through the confusion matrix

  看的出來預測到旁邊一個 label 的機率還是很高，然後比較奇怪的是 predicted 中不管是 ResNet18 還是 50 預測是 class 1 的數目都很少，我猜可能是 class 1 本身的 data 就不多，有 data unbalance 的問題，但我沒有實際去看 data



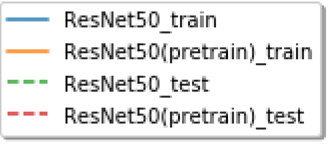


3. Expermental result

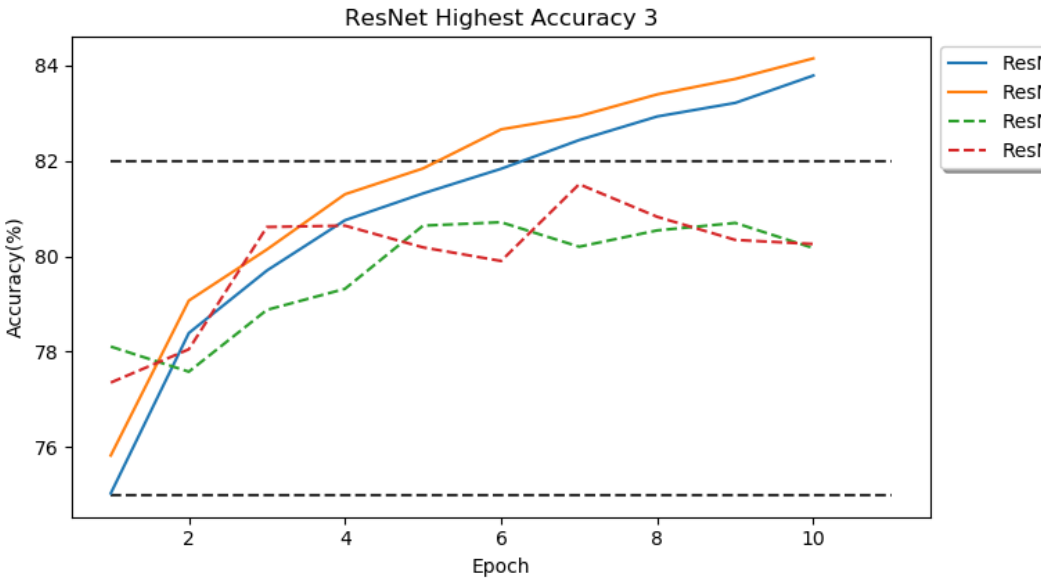
- The hightest testing accuracy

```
ResNet Highest Accuracy 3
ResNet18(pretrain)_test : 80.71174377224199
ResNet50(pretrain)_test : 81.50889679715303
```

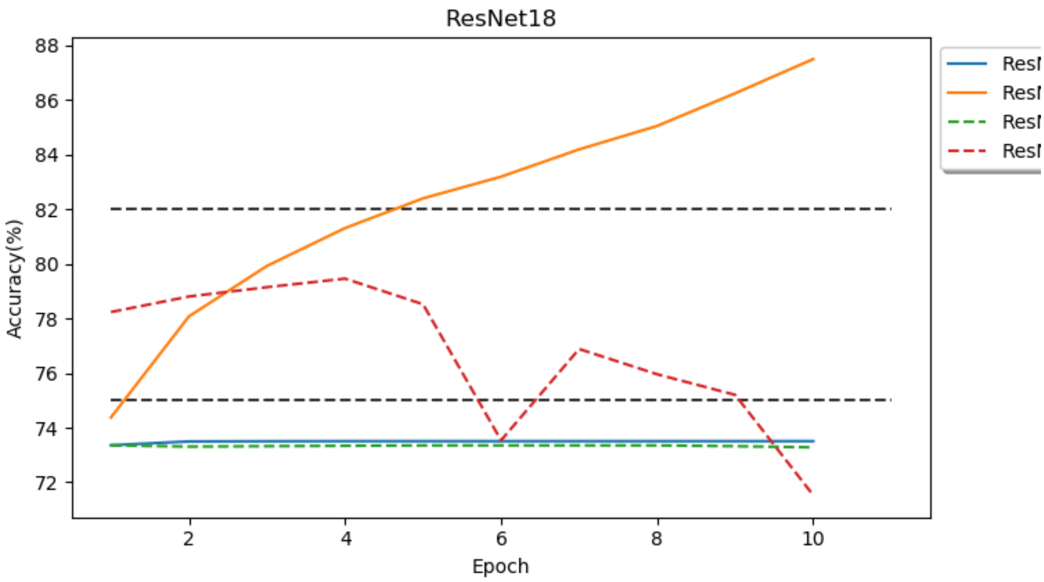
(實線是 train / 虛線是 test)

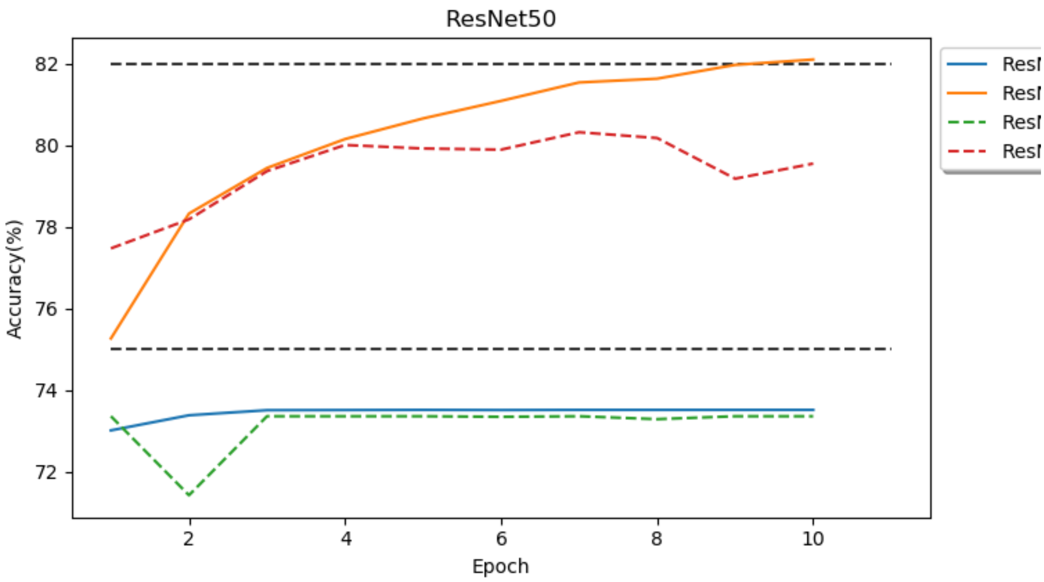


(我也不知道為甚麼存起來會被切掉...)



- Comparison figure





4. Disscussion

我參考了很多不同人寫的 ResNet 然後改寫了很多次，這次作業除了 Net 以外我把所有的東西直接複製進來我的 code 了(像是 save/load model, show 圖片)，不過因為是指定的架構，所以寫起來也大同小異，所以感覺改著改著已經跟別人寫的差不多有九成像了……

然後感覺把 data 做 shuffle 會有更好的正確率，但是時間不夠再讓我重 train 了，然後我發現在 dataloader 那邊如果我直接用
self.transforms = transforms.ToTensor()
而不是
self.transforms =
transforms.Compose([transforms.RandomHorizontalFlip(p=0.5),
transforms.RandomVerticalFlip(p=0.5), transforms.ToTensor()])
的話會快非常多，對水平跟垂直做隨機翻轉後慢到我以為我是在用 cpu 在跑還檢查了一下，我蠻不解的

最好的結果是用 pretrained model 然後只是把 batch size 調大(本來想開到 16，結果直接 ram 爆了，所以只有開到 8)

5. 參考:

https://github.com/csielee/2019DL/tree/master/lab3
https://github.com/xiaohu2015/DeepLearning_tutorials/blob/master/CNNs/ResNet50.py
https://iter01.com/525623.html

https://blog.csdn.net/sunqiande88/article/details/80100891
https://github.com/shanglianlm0525/PyTorch-Networks/blob/master/ClassicNetwork/ResNet.py
https://jennaweng0621.pixnet.net/blog/post/403589876-%5Bpytorch%5D-resnet%E7%B3%BB%E5%88%97%E7%A5%9E%E7%B6%93%E7%B6%B2%E8%B7%AF%E7%B5%90%E6%A7%8B%28resnet18%2C-resnet34%2C