Lab 3 Report - Retinopathy Detection

- Introduction (20%)

  此次實驗是基於PyTorch實作兩個neural network - ResNet18及ResNet50，而的dataset的資料內容為視網膜照片，目的是辨別因為糖尿病而造成的視網膜病變嚴重程度。

  在nenural network方面，兩個network的差異除了深度外，所組成的block也不相同，ResNet18會由basic block組成，而ResNet50則會由bottleneck block組成。

  在實作方面，TA提供的`getData` 及`RetinopathyDataset` 作為基礎建立data loader，並嘗試利用一個`ResNet` class處理ResNet18和ResNet50兩個network的建立。

- Experiment setups (30%)

  1. The details of your model (ResNet)

     - Common parts

       1. `ResNet`

          - `get_pretrained_weights`

            ```
            def get_pretrained_weights(architecture):
                if architecture == 'resnet18':
                    return ResNet18_Weights.IMAGENET1K_V1
                else:
                    return ResNet50_Weights.IMAGENET1K_V1
            ```

            根據architecture來決定pretrained weights的給定。

          - `generate_blocks`

```python
    # Generate blocks for conv_2, conv_3, conv_4, and conv_5.
    def generate_blocks(self, block, num_of_blocks, in_channels, stride=1):
        down_sample = None

        # If the number of channels, width, or height will be changed, do down_sample for residual.
        if stride != 1 or self.current_channels != in_channels * block.expansion:
            down_sample = nn.Sequential(
                nn.Conv2d(
                    in_channels=self.current_channels,
                    out_channels=in_channels * block.expansion,
                    kernel_size=1,
                    stride=stride,
                    bias=False,
                ),
                nn.BatchNorm2d(
                    num_features=in_channels * block.expansion,
                ),
            )
```

```python
        # Build first block for conv_2, conv_3, conv_4, and conv_5.
        layers = [
            block(
                in_channels=self.current_channels,
                out_channels=in_channels,
                stride=stride,
                down_sample=down_sample,
            ),
        ]

        # After building the first block, the current_channels will be changed (because of convolution
        # in first block).
        self.current_channels = in_channels * block.expansion
        # Build remaining blocks for conv_2, conv_3, conv_4, and conv_5.
        layers += [
            block(
                in_channels=self.current_channels,
                out_channels=in_channels,
            ) for _ in range(1, num_of_blocks)]

        return nn.Sequential(*layers)
```

用於建立conv_2, conv_3, conv_4, and conv_5，每層皆由多個block組成，block數可由`num_of_blocks`來控制，而由於block可能為basic block或是bottleneck block，因此以參數傳入。

- `__init__`

    1. Pretrained

```python
        if pretrain:
            pretrained_weights = self.get_pretrained_weights(architecture)
            pretrained_resnet = getattr(torch_models, architecture)(weights=pretrained_weights)
            self.conv_1 = nn.Sequential(
                getattr(pretrained_resnet, 'conv1'),
                getattr(pretrained_resnet, 'bn1'),
                getattr(pretrained_resnet, 'relu'),
                getattr(pretrained_resnet, 'maxpool')
            )

            # Layers
            self.conv_2 = getattr(pretrained_resnet, 'layer1')
            self.conv_3 = getattr(pretrained_resnet, 'layer2')
            self.conv_4 = getattr(pretrained_resnet, 'layer3')
            self.conv_5 = getattr(pretrained_resnet, 'layer4')

            self.classify = nn.Sequential(
                getattr(pretrained_resnet, 'avgpool'),
                nn.Flatten(),
                nn.Linear(getattr(pretrained_resnet, 'fc').in_features, out_features=50),
                nn.ReLU(
                    inplace=True,
                ),
                nn.Dropout(
                    p=0.25,
                ),
                nn.Linear(
                    in_features=50,
                    out_features=5,
                ),
            )
```

pretrained model包含NN架構及weights，讀入後寫入各層layer即可。

2. Not pretrained

```python
        else:
            self.current_channels = 64

            self.conv_1 = nn.Sequential(
                nn.Conv2d(
                    in_channels=3,
                    out_channels=64,
                    kernel_size=7,
                    stride=2,
                    padding=3,
                    bias=False,
                ),
                nn.BatchNorm2d(
                    num_features=64,
                ),
                nn.ReLU(
                    inplace=True,
                ),
                nn.MaxPool2d(
                    kernel_size=3,
                    stride=2,
                    padding=1,
                ),
            )
```

```python
        # Layers
        self.conv_2 = self.generate_blocks(
            block=block,
            num_of_blocks=layers[0],
            in_channels=64,
        )
        self.conv_3 = self.generate_blocks(
            block=block,
            num_of_blocks=layers[1],
            in_channels=128,
            stride=2,
        )
        self.conv_4 = self.generate_blocks(
            block=block,
            num_of_blocks=layers[2],
            in_channels=256,
            stride=2,
        )
        self.conv_5 = self.generate_blocks(
            block=block,
            num_of_blocks=layers[3],
            in_channels=512,
            stride=2,
        )
```

```python
        self.classify = nn.Sequential(
            nn.AdaptiveAvgPool2d(
                output_size=(1, 1),
            ),
            nn.Flatten(),
            nn.Linear(
                in_features=512 * block.expansion,
                out_features=50,
            ),
            nn.ReLU(
                inplace=True,
            ),
            nn.Dropout(
                p=0.25,
            ),
            nn.Linear(
                in_features=50,
                out_features=5,
            ),
        )
```

除了`conv_1`，其他layer皆以`generate_blocks`來生成，實現共用構建ResNet18及ResNet50的程式碼。

- `forward`

```python
    def forward(self, inputs: TensorDataset):
        partial_results = inputs
        for idx in range(1, 6):
            partial_results = getattr(self, f'conv_{idx}')(partial_results)
        return self.classify(partial_results)
```

- ReseNet18

1. `BasicBlock`

- `__init__`

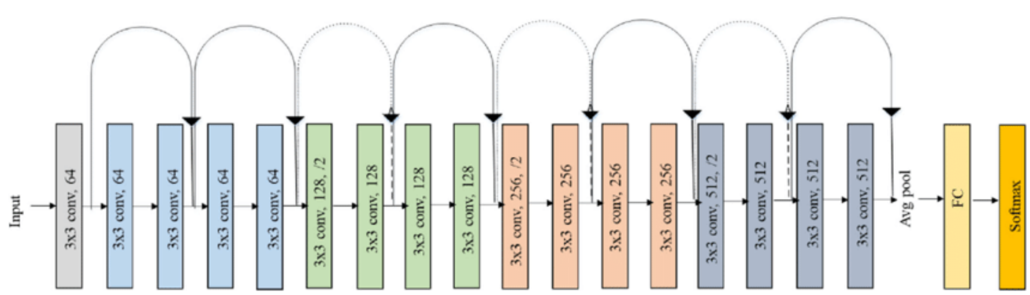
```python
def __init__(self, in_channels, out_channels, stride=1, down_sample=None):
    super(BasicBlock, self).__init__()

    self.activation = nn.ReLU(
        inplace=True,
    )
    self.block = nn.Sequential(
        nn.Conv2d(
            in_channels=in_channels,
            out_channels=out_channels,
            kernel_size=3,
            stride=stride,
            padding=1,
            bias=False,
        ),
        nn.BatchNorm2d(
            num_features=out_channels,
        ),
        self.activation,
        nn.Conv2d(
            in_channels=out_channels,
            out_channels=out_channels,
            kernel_size=3,
            padding=1,
            bias=False,
        ),
        nn.BatchNorm2d(
            num_features=out_channels,
        ),
    )
    self.down_sample = down_sample
```

基於spec上的方式建立basic block，主要包含兩次的convolution，並保持輸入輸出channel數不變。

- `forward`

```python
def forward(self, inputs):
    outputs = self.block(inputs)
    if self.down_sample is not None:
        inputs = self.down_sample(inputs)
    outputs = self.activation(outputs + inputs)

    return outputs
```

2. `resnet_18`



根據此圖建立ResNet18，主要包含四層 (藍綠橙灰)，每層由兩個 `BasicBlock` 所組成，因此 `layers` 設為[2, 2, 2, 2] (後四層)，如下圖:

```python
def resnet_18(pretrain=False):
    return ResNet(
        architecture='resnet18',
        block=BasicBlock,
        layers=[2, 2, 2, 2],
        pretrain=pretrain,
    )
```

- ReseNet50

1. `BottleneckBlock`

    - `__init__`

```python
def __init__(self, in_channels, out_channels, stride=1, down_sample=None):
    super(BottleneckBlock, self).__init__()

    external_channels = out_channels * self.expansion
    self.activation = nn.ReLU(
        inplace=True,
    )
    self.block = nn.Sequential(
        nn.Conv2d(
            in_channels=in_channels,
            out_channels=out_channels,
            kernel_size=1,
            bias=False,
        ),
        nn.BatchNorm2d(
            num_features=out_channels,
        ),
        self.activation,
        nn.Conv2d(
            in_channels=out_channels,
            out_channels=out_channels,
            kernel_size=3,
            stride=stride,
            padding=1,
            bias=False,
        ),
```

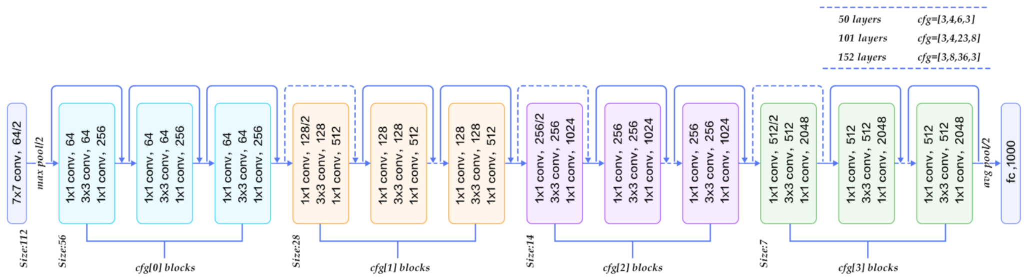
```python
        nn.BatchNorm2d(
            num_features=out_channels,
        ),
        self.activation,
        nn.Conv2d(
            in_channels=out_channels,
            out_channels=external_channels,
            kernel_size=1,
            bias=False,
        ),
        nn.BatchNorm2d(
            num_features=external_channels,
        ),
    )
    self.down_sample = down_sample
```

基於spec上的方式建立basic block，主要包含三次的convolution，且輸出channel數為輸入的channel數的四倍。

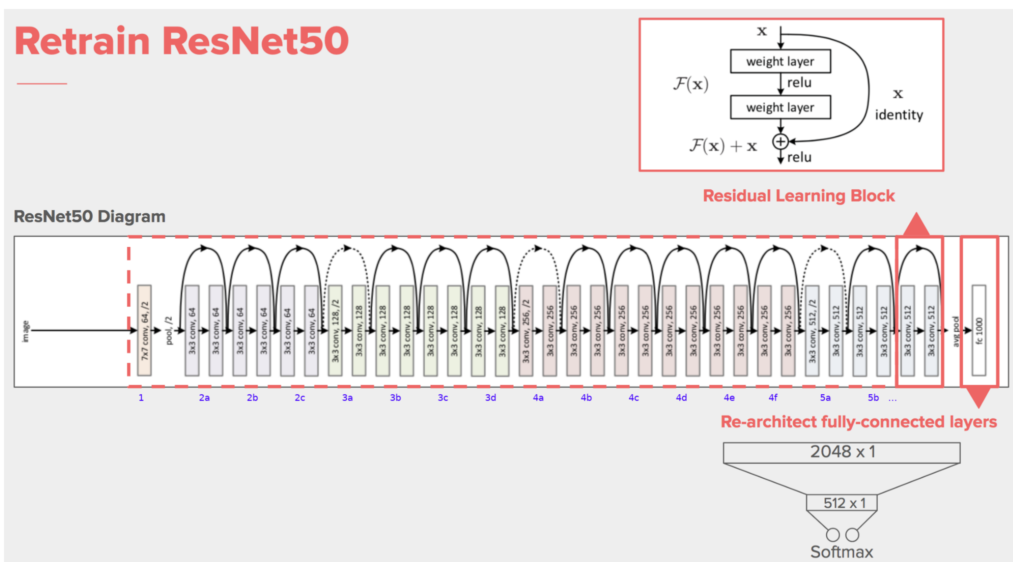
    - `forward`

```python
def forward(self, inputs):
    outputs = self.block(inputs)
    if self.down_sample is not None:
        inputs = self.down_sample(inputs)
    outputs = self.activation(outputs + inputs)

    return outputs
```

2. `resnet_50`



根據此圖建立ResNet50，主要包含四層 (藍黃紅綠)，每層由兩個 `BottleneckBlock` 所組，但 `layers` 設為[3, 4, 6, 3] (後四層)，目的是與paper保持一致，如下圖:




```python
def resnet_50(pretrain=False):
    return ResNet(
        architecture='resnet50',
        block=BottleneckBlock,
        layers=[3, 4, 6, 3],
        pretrain=pretrain,
    )
```

2. The details of your data loader

- `__init__`

```python
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.image_name, self.label = getData(mode)
    self.mode = mode
    print("> Found %d images..." % (len(self.image_name)))
```

與TA預設的`__init__` function相同。

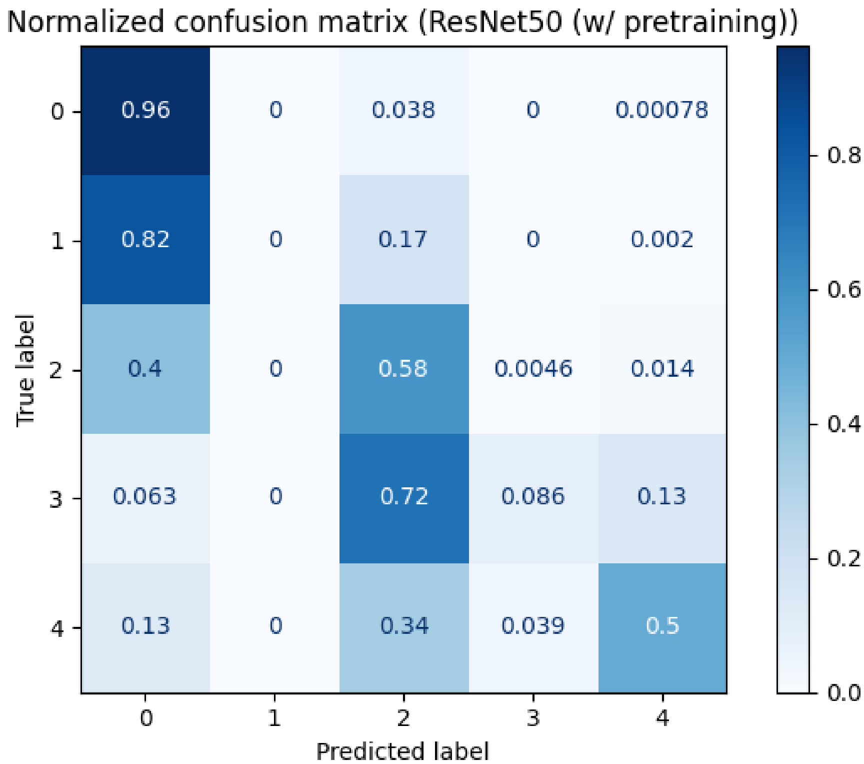
- `__getitem__`

```python
def __getitem__(self, index):
    # Load image.
    image_path = os.path.join(self.root, f'{self.image_name[index]}.jpeg')
    # image = mpimg.imread(image_path)
    image = PIL.Image.open(image_path)

    # Get the ground truth label.
    label = self.label[index]

    # Transform the .jpeg rgb images during the training phase.
    # Convert the pixel value to [0, 1].
    # image = np.where(image < 128, 0, 1)
    # Transpose the image shape from [H, W, C] to [C, H, W].
    # image = np.transpose(image, (2, 0, 1))
    transform = transforms.Compose([
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.RandomVerticalFlip(p=0.5),
        transforms.ToTensor()
    ])
    image = transform(image)
```

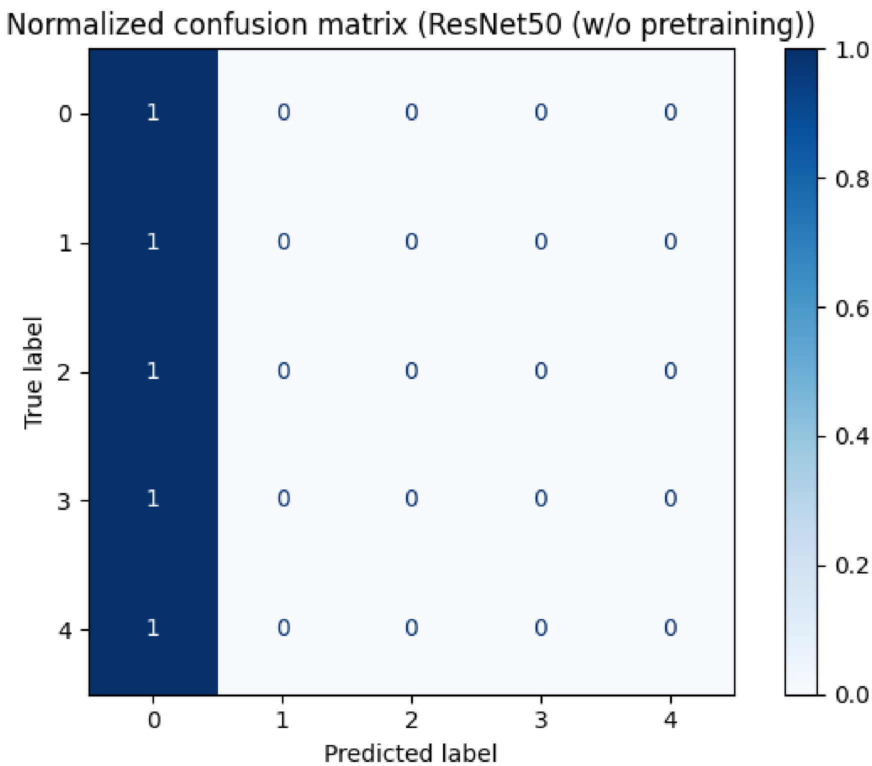
嘗試使用TA推薦的方式，先做一次二質化，並將shape由[H, W, C] 轉為[C, H, W]，及嘗試把圖片做垂直及水平反轉，增加圖片多樣性，發現第二種方式比較好。

3. Describing your evaluation through the confusion matrix

- Pretrained (ResNet50)



- Not pretrained (ResNet50)



- Discussion

  由上面兩張圖可以發現，pretrained效果對比沒有pretrained來的更好，在相同的epoch數目下，model更佳general，且由comparison graph還可發現，沒有pretrained的accuracy還有下降的趨勢。

- Experimental results (30%)

  1. The highest testing accuracy

     - Parameters

       1. model → ResNet18
       2. batch_size → 32
       3. learning_rate → 1e-3
       4. epochs → 10
       5. optimizer → SGD

6. momentum → 0.9
7. weight_decay → 5e-4

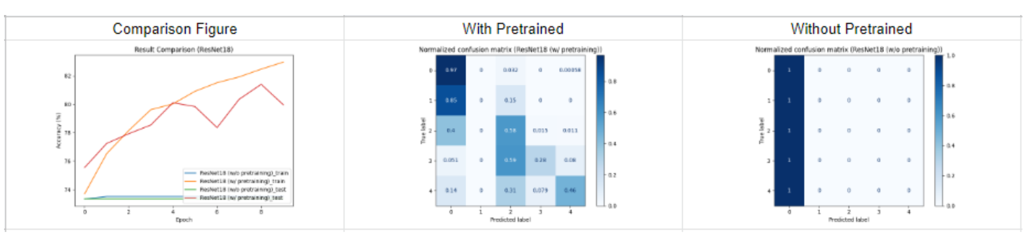
- Screenshot

```
ResNet18 (w/o pretraining)_train: 73.52 %
ResNet18 (w/ pretraining)_train:  82.96 %
ResNet18 (w/o pretraining)_test:  73.35 %
ResNet18 (w/ pretraining)_test:   81.40 %

Process finished with exit code 0
```

2. Comparison figures

- ResNet18

1. Parameters
    - model → ResNet18
    - batch_size → 32
    - learning_rate → 1e-3
    - epochs → 10
    - optimizer → SGD
    - momentum → 0.9
    - weight_decay → 5e-4
2. Results

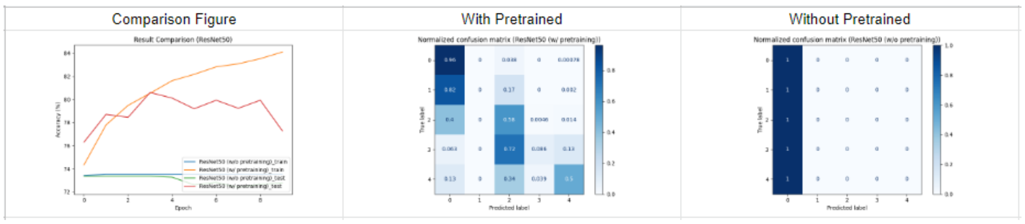


- ResNet50

1. Parameters

- model → ResNet18
- batch_size → 16
- learning_rate → 1e-3
- epochs → 10
- optimizer → SGD
- momentum → 0.9
- weight_decay → 5e-4

2. Results



- Discussion (20%)

1. Accuracy trend of ResNet18 & ResNet50

   由上面兩張comparison figure的結果可以發現，pretrained model在epoch提高時，train的accuracy但同時test的accuracy卻下降，可能的原因是epoch還不夠多。

2. Gradient Vanishing vs. Degradation

   一開始在做這個project時認為由於層數上升導致gradient在淺層的weight值變小，因此產生gradient vanishing的問題，而ResNet可以改善此問題。但在看過幾篇文章後發現，實際問題可能並非gradient vanishing，原因是他們有嘗試利用其他方式解決，諸如batch norm或drop out都無法改善gradient vanishing，因此ResNet作者提出了一個新的名詞，稱為degradation problem。而ResNet能改善此問題的原因在於Identify mapping，當輸出變成輸入 + 殘差，當殘差很小就可是為輸入，便可以視為此層無作用，我覺得可以理解為ResNet可以在training過程自己適度調節實際運作的layer層數，以最適當的符合training dataset的特性。

- Referenc

1. https://pytorch.org/tutorials/beginner/data_loading_tutorial.html
2. https://github.com/utkuozbulak/pytorch-custom-dataset-examples/blob/master/src/custom_dataset_from_file.py
3. https://medium.com/@hupinwei/深度學習-resnet之殘差學習-f3ac36701b2f