Deep Learning and Practice Lab 4

309552007

袁鈺勛

A. Introduction

這次作業是要利用 pytorch 實作出 ResNet 來分類視網膜照片，分類出他因為糖尿病而造成的視網膜病變嚴重程度，在程式中有提供 argument 來控制 model 的各個參數，利用'-t'可以選擇要使用 ResNet18 還是 ResNet50，用'-c'可以控制是否要一起訓練 pretrained model 和 w/o pretrained model 來進行比較。

```
parser = ArgumentParser(description='ResNet')
parser.add_argument('-t', '--target_model', default='ResNet18', type=check_model_type, help='ResNet18 or ResNet50')
parser.add_argument('-c', '--comparison', default=1, type=check_comparison_type,
                    help='Whether compare the accuracies of w/ pretraining and w/o pretraining models')
parser.add_argument('-p', '--pretrain', default=0, type=check_pretrain_type,
                    help='Train w/ pretraining model or w/o pretraining model when "comparison" is false')
parser.add_argument('-l', '--load', default=0, type=check_load_type,
                    help='Whether load the stored model and accuracies')
parser.add_argument('-s', '--show_only', default=0, type=check_show_type, help='Whether only show the results')
parser.add_argument('-b', '--batch_size', default=4, type=int, help='Batch size')
parser.add_argument('-lr', '--learning_rate', default=1e-3, type=float, help='Learning rate')
parser.add_argument('-e', '--epochs', default=10, type=int, help='Number of epochs')
parser.add_argument('-o', '--optimizer', default='sgd', type=check_optimizer_type, help='Optimizer')
parser.add_argument('-m', '--momentum', default=0.9, type=float, help='Momentum factor for SGD')
parser.add_argument('-w', '--weight_decay', default=5e-4, type=float, help='Weight decay (L2 penalty)')
parser.add_argument('-v', '--verbosity', default=0, type=check_verbosity_type, help='Verbosity level')
```

B. Experiment setups

1. The details of your model (ResNet)

實作方式下面分為 basic block、bottleneck block，以及由 blocks 組成的 ResNet:

a. Basic block

```python
class BasicBlock(nn.Module):
    """
    output = (channels, H, W) -> conv2d (3×3) -> (channels, H, W) -> conv2d (3×3) -> (channels, H, W) + (channels, H
    """
    expansion: int = 1

    def __init__(self, in_channels: int, out_channels: int, stride: int = 1, down_sample: Optional[nn.Module] = 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(out_channels),
            self.activation,
            nn.Conv2d(
```

```python
                in_channels=out_channels,
                out_channels=out_channels,
                kernel_size=3,
                padding=1,
                bias=False),
            nn.BatchNorm2d(out_channels),
        )
        self.down_sample = down_sample

    def forward(self, inputs: TensorDataset) -> Tensor:
        """
        Forward propagation
        :param inputs: input data
        :return: results
        """
        residual = inputs
        outputs = self.block(inputs)
        if self.down_sample is not None:
            residual = self.down_sample(inputs)

        outputs = self.activation(outputs + residual)
```

```python
        return outputs
```

上面的圖為根據 basic block 建立的，其中的 down_sample 是為了要因應在 ResNet 中跨 convolution layer 時 channel 數會變化而需要使用的，要讓 input 以及 output 的 channel 數以及 height 和 width 相等。

b. Bottleneck block

```python
class BottleneckBlock(nn.Module):
    """
    output = (channels * 4, H, W) -> conv2d (1×1) -> (channels, H, W) -> conv2d (3×3) -> (channels, H, W)
             -> conv2d (1×1) -> (channels * 4, H, W) + (channels * 4, H, W)
    """
    expansion: int = 4

    def __init__(self, in_channels: int, out_channels: int, stride: int = 1, down_sample: Optional[nn.Module] = 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(out_channels),
            self.activation,
            nn.Conv2d(in_channels=out_channels,
                      out_channels=out_channels,
```

```
                    kernel_size=3,
                    stride=stride,
                    padding=1,
                    bias=False),
            nn.BatchNorm2d(out_channels),
            self.activation,
            nn.Conv2d(in_channels=out_channels,
                    out_channels=external_channels,
                    kernel_size=1,
                    bias=False),
            nn.BatchNorm2d(external_channels),
        )
        self.down_sample = down_sample

    def forward(self, inputs: TensorDataset) -> Tensor:
        """
        Forward propagation
        :param inputs: input data
        :return: results
        """
        residual = inputs
        outputs = self.block(inputs)
```

```
        if self.down_sample is not None:
            residual = self.down_sample(inputs)

        outputs = self.activation(outputs + residual)

        return outputs
```

上面的圖為根據 bottleneck block 建立的，其中的 down_sample 和 basic block 是一樣的用途。

c. ResNet

```
class ResNet(nn.Module):
    def __init__(self, architecture: str, block: Type[Union[BasicBlock, BottleneckBlock]], layers: List[int],
                 pretrain: bool):
        super(ResNet, self).__init__()

        if pretrain:
            pretrained_resnet = getattr(torch_models, architecture)(pretrained=True)
            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)
            )

            del pretrained_resnet
        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(64),
                nn.ReLU(inplace=True),
```

```
            nn.MaxPool2d(kernel_size=3,
                         stride=2,
                         padding=1)
        )

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

```
        self.classify = nn.Sequential(
            nn.AdaptiveAvgPool2d((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)
        )

    def make_layer(self, block: Type[Union[BasicBlock, BottleneckBlock]], num_of_blocks: int, in_channels: int,
                   stride: int = 1) -> nn.Sequential:
        """
        Make a layer with given block
        :param block: block to be used to compose the layer
        :param num_of_blocks: number of blocks in this layer
        :param in_channels: channels used in the blocks
        :param stride: stride
        :return: convolution layer composed with given block
        """
        down_sample = None
```

```
        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(in_channels * block.expansion),
            )

        layers = [
            block(in_channels=self.current_channels,
                  out_channels=in_channels,
                  stride=stride,
                  down_sample=down_sample)
        ]
        self.current_channels = in_channels * block.expansion
        layers += [block(in_channels=self.current_channels, out_channels=in_channels) for _ in range(1, num_of_

        return nn.Sequential(*layers)
```

```
    def forward(self, inputs: TensorDataset) -> Tensor:
        """
        Forward propagation
        :param inputs: input data
        :return: results
        """
        partial_results = inputs
        for idx in range(1, 6):
            partial_results = getattr(self, f'conv_{idx}')(partial_results)
        return self.classify(partial_results)
```

上圖為 ResNet 的基本架構，會利用前面的 basic block 或 bottleneck block 來組成 convolution layer 2~5，如果是 pretrained model 的話，就會從 pytorch 載入 pretrained ResNet 來建構，只有

fully connected layer 是非 pretrained，目前是使用兩層 fully connected layers。Class 中的 make_layer function 用於建立 convolution layers，其中的 down_sample 便是用於 convolution layer 之間的 channel 以及 height 和 width 變換。

```
def resnet_18(pretrain: bool = False) -> ResNet:
    """
    Get ResNet18
    :param pretrain: whether use pretrained model
    :return: ResNet18
    """
    return ResNet(architecture='resnet18', block=BasicBlock, layers=[2, 2, 2, 2], pretrain=pretrain)


def resnet_50(pretrain: bool = False) -> ResNet:
    """
    Get ResNet50
    :param pretrain: whether use pretrained model
    :return: ResNet50
    """
    return ResNet(architecture='resnet50', block=BottleneckBlock, layers=[3, 4, 6, 3], pretrain=pretrain)
```

根據上面的 ResNet class 以及所需的 block 和 layer 數，便可以建立 ResNet18 和 ResNet50。

2. The details of your Dataloader

```
self.root = root
self.img_name, self.label = get_data(mode)
self.mode = mode
trans = []
if transformations:
    trans += transformations
trans.append(transforms.ToTensor())
self.transform = transforms.Compose(trans)
print("> Found %d images ..." % (len(self.img_name)))
```

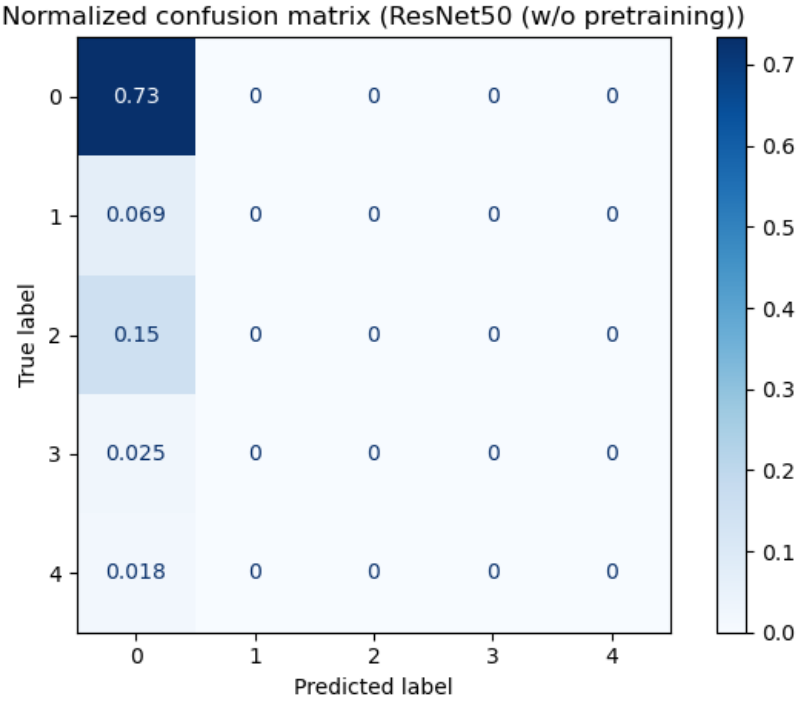
Dataloader 會在__init__ function 中取得 images 所在的 folder，然後從 csv 讀出檔案名和對應的 label，並且根據得到的 transformations 來建立 transforms。

```
path = os.path.join(self.root, f'{self.img_name[index]}.jpeg')
img = self.transform(PIL.Image.open(path))
label = self.label[index]

return img, label
```

在__getitem__ function 會根據取得的 index 從 root folder 取出對應的 image，並將 image 轉成 Tensor 後再做建立的 transformations，最後回傳轉換過後的 image 以及其對應的 label。

3. Describing your evaluation through the confusion matrix



由 without pretraining 的 confusion matrix 可以看出 without pretraining 的都會將所有圖片歸為 label 0，可見 model 還不夠 general，所以會用 pretrained model 來找最高的 testing accuracy。

C. Experimental results

1. The highest testing accuracy

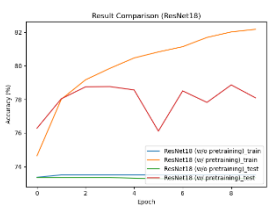
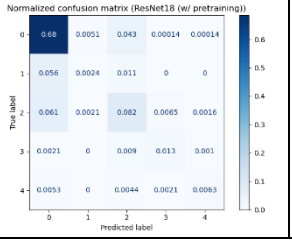
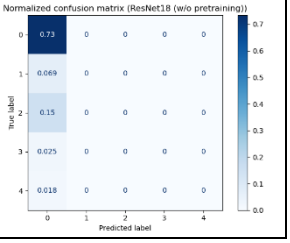
```
ResNet50 (w/ pretraining)_train: 83.41 %
ResNet50 (w/ pretraining)_test:  81.17 %
```

目前得到最高的 testing accuracy 是 ResNet50，只有 batch size 是 12，其他參數都是預設，fully connected layers 為 2。

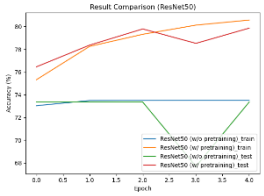
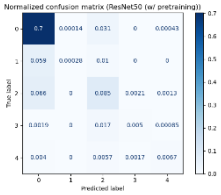
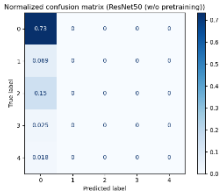
2. Comparison figures

以下的 confusion matrix 都是 all normalization，不是 row normalization。

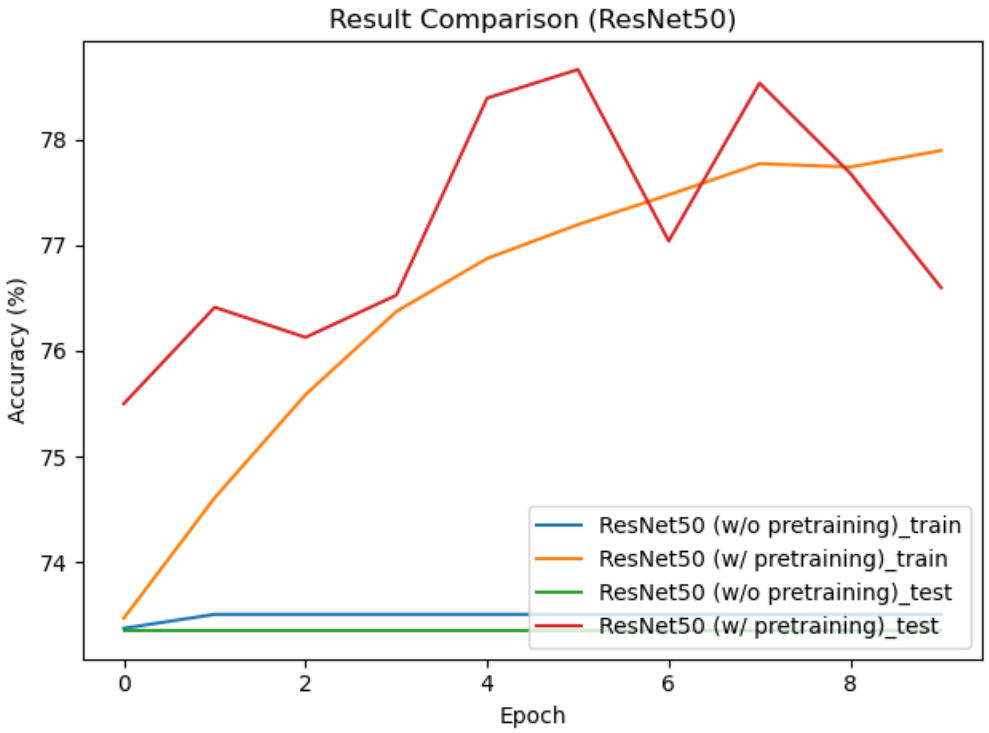
a. ResNet18

Comparison figure	With pretraining	Without pretraining
		

b. ResNet50

Comparison figure	With pretraining	Without pretraining
		

D. Discussion



上面兩張圖分別是 ResNet18 用三層 fully connected layers 跑 20 個 epochs 和 ResNet50 用三層 fully connected layers 跑 10 個 epochs，這兩個得出的最高 testing accuracy 沒有 default 的高，可能是因為 fc layers 較多，需要較多 epoch 才能收斂，所以目前還在嘗試其他設定看能不能將 testing accuracy 提升。在嘗試中有試過提升 batch size，因為 data 試 image，所以 dimension 會很高，因此以目前提供的 1060 的記憶體只能最高提升到 12。