Lab4-2 Report

Introduction

和 Lab4-1 一樣是做分類問題，這次是分類視網膜的病變。總共有五類，我們要用 ResNet 18 和 ResNet 50 這兩個 model 進行分類。這兩個 model 都在 torchvision 函式庫中宣告好了，因此我們只要負責呼叫、使用即可。而 dataloader 我們主要需要實作 getitem()的部分。它的功能不只是回傳圖片本身，更要做一些前處理，以增加模型預測的正確率。

Experiment setups

The details of your model (ResNet)

model 的細節 torchvision 函式庫都幫我們做好了。我們需要更改的地方在最後一層 fully connected layer、也就是 linear 層的部分要輸出 5 類 classes。

```
def get_model1():
    resnet18 = models.resnet18(pretrained = True)
    num_ftrs = resnet18.fc.in_features
    resnet18.fc = nn.Linear(num_ftrs, 5)
    return resnet18

def get_model2():
    resnet50 = models.resnet50(pretrained = True)
    num_ftrs = resnet50.fc.in_features
    resnet50.fc = nn.Linear(num_ftrs, 5)
    return resnet50
```

首先，先取得 input neuron 數，也就是上圖中的 num_ftrs，再把它和

output class 數(5)一起傳進 nn.Linear 裡面，最後指派給 resnet.fc 即完成。另外，resnet 還有一個 pretrained 參數可以使用，設定成 True 即可載入 pretrained weight。pretrained 的意思就是模型已經由先人經過大量的 dataset 訓練(ex: ImageNet)，並儲存 weights 和 biases 供後人使用。這樣我們就不需要從 random initialize 的參數開始訓練。當然也可以設定 pretained = False 將其關掉。

且 pretrained model 通常會搭配 feature extraction 使用。feature extraction 是指在前幾個 epoch 單獨使用一個 optimizer 只更新最後一層的 fc layer，其他中間層則不計算 gradient 也不更新。其目的是為了更加運用已經先訓練好的 weight，固定前面的 CNN layers 有取出 input image 的特徵的用意。以下是搭配 feature extraction 的 code:

```
def set_parameter_requires_grad(model, grad):
    if grad:
        for param in model.parameters():
            param.requires_grad = True
    else:
        for param in model.parameters():
            param.requires_grad = False

def get_model1():
    resnet18 = models.resnet18(pretrained = True)
    set_parameter_requires_grad(resnet18, False)
    num_ftrs = resnet18.fc.in_features
    resnet18.fc = nn.Linear(num_ftrs, 5)
    return resnet18

def get_model2():
    resnet50 = models.resnet50(pretrained = True)
    set_parameter_requires_grad(resnet50, False)
    num_ftrs = resnet50.fc.in_features
    resnet50.fc = nn.Linear(num_ftrs, 5)
    return resnet50
```

剛被指派的 fc 層預設的 requires_grad 為 true，故可以在只派之前將其他層都改成 false，也就是 set_parameter_requires_grad 函式在做的事。這樣 return 回去的 model 就是一個只有 fc 層的 require_grad 是 true、其他層都是 false 的 model。

The details of your Dataloader

我們要實作的是 RetinopathyLoader 裡的 getitem()函式，接下來 torch.utils.data.DataLoader 才能依據 getitem 拿取相應的資料。

```
# step 1
path = self.root + self.img_name[index] + '.jpeg'
# step 2
label = self.label[index]
# step 3
img = Image.open(path)
preprocess = transforms.Compose([
    transforms.Resize(256),
    #transforms.CenterCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.RandomCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

input_tensor = preprocess(img)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
input_tensor = input_tensor.to(device)
img = input_tensor

return img, label
```

首先，根據路徑和 index 取得 image 和 label，再對資料進行一些前處理並回傳。而這邊前處理的目的主要是為了將圖片轉成適合傳入模型的型態以及讓圖片資料多一點變化性和多樣性。上圖中使用的處理手段如下:

- Resize: Resize(256)表示 image 會被轉成 256×256 pixels。
- RandomHorizontalFlip: RandomHorizontalFlip(p=0.5)表示圖片有

0.5 的機率進行水平翻轉。

- RandomVerticalFlip: RandomVerticalFlip(p=0.5)表示圖片有 0.5 的機率進行垂直翻轉。
- RandomCrop: 依據給定的 size 隨機剪裁。
- ToTensor: 將 PIL image 或 ndarray 轉成 tensor 型態，並將值轉成[0, 1]。而影像大小(H x W x C)也會被轉成(C x H x W)。
- Normalize: 正規化，需給定影像平均值(mean)和標準差(std)。根據 ImageNet 資料集訓練，訓練集所計算出來的平均值(mean)和標準差(std)分別為 mean = [0.485, 0.456, 0.406] 和 std = [0.229, 0.224, 0.225]。

Describing your evaluation through the confusion matrix

confusion matrix 的畫法我直接用 sklearn.metrics 裡的 ConfusionMatrixDisplay 函式來畫。

Parameters:

y_true : *array-like of shape (n_samples,)*
True labels.

y_pred : *array-like of shape (n_samples,)*
The predicted labels given by the method `predict` of an classifier.

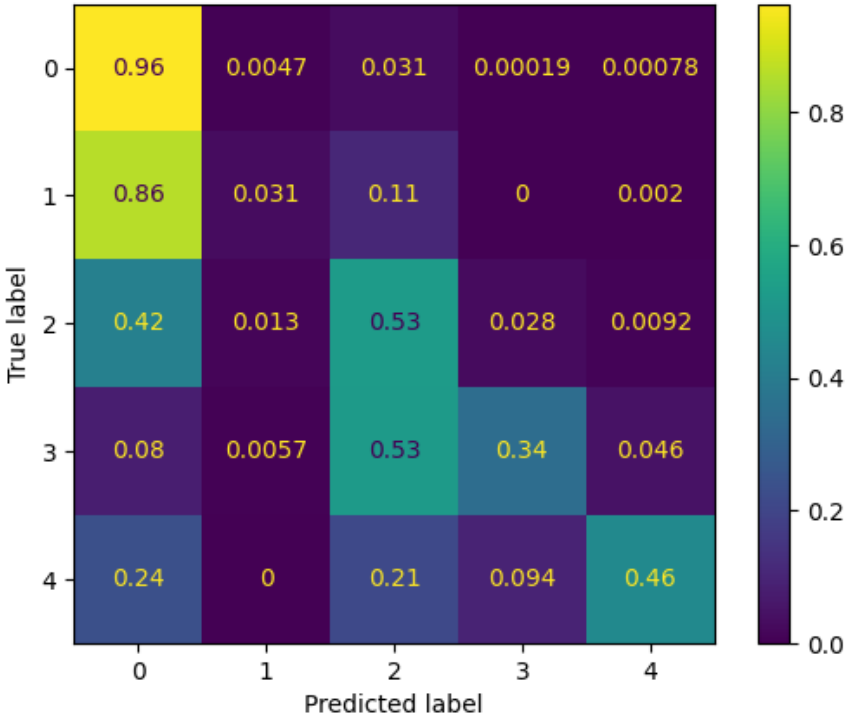
labels : *array-like of shape (n_classes,), default=None*
List of labels to index the confusion matrix. This may be used to reorder or select a subset of labels. If `None` is given, those that appear at least once in `y_true` or `y_pred` are used in sorted order.

sample_weight : *array-like of shape (n_samples,), default=None*
Sample weights.

normalize : *{'true', 'pred', 'all'}, default=None*
Either to normalize the counts display in the matrix:

- if `'true'`, the confusion matrix is normalized over the true conditions (e.g. rows);
- if `'pred'`, the confusion matrix is normalized over the predicted conditions (e.g. columns);
- if `'all'`, the confusion matrix is normalized by the total number of samples;
- if `None` (default), the confusion matrix will not be normalized.

只需要傳 true_label 和 prediction 結果，並指定 normalize = true 即可。舉一個畫出來的結果為例:



縱軸是 true label、橫軸是 predicted label。舉例來說，上圖中的 0.96 那格就是預測是 0、實際也是 0 的比例是 0.96；0.86 那格就是預測是 0、實際是 1 的比例為 0.86。而我指定 normalize = true 代表用真實的數量(true label)來 normalize，即每個 row 總和為 1。

Experimental results

Hyper Parameters
Batch size = 16
Learning rate = 1e-3
Epochs = 20
Optimizer: SGD momentum = 0.9 weight_decay = 5e-4

Loss function: torch.nn.CrossEntropyLoss()

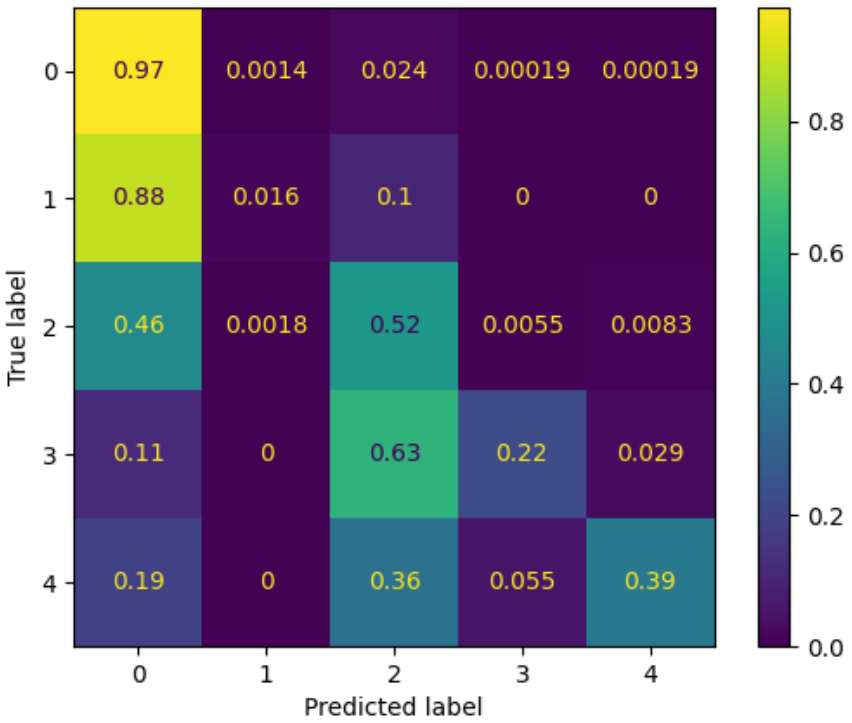
The highest testing accuracy

用 pretrained 會有較高的 accuracy，故以下只呈現用 pretrained 的結果。

ResNet18

```
epoch 18:
trainig accuracy: tensor(81.7147, device='cuda:0')  loss: 956.5676323547959
testing accuracy: tensor(80.5125, device='cuda:0')

epoch 19:
trainig accuracy: tensor(82.0599, device='cuda:0')  loss: 936.7826828733087
testing accuracy: tensor(80.5125, device='cuda:0')

epoch 20:
trainig accuracy: tensor(81.9887, device='cuda:0')  loss: 929.44252468645 57
testing accuracy: tensor(80.8256, device='cuda:0')

max testing accuracy 80.82562255859375% at epoch 20
```

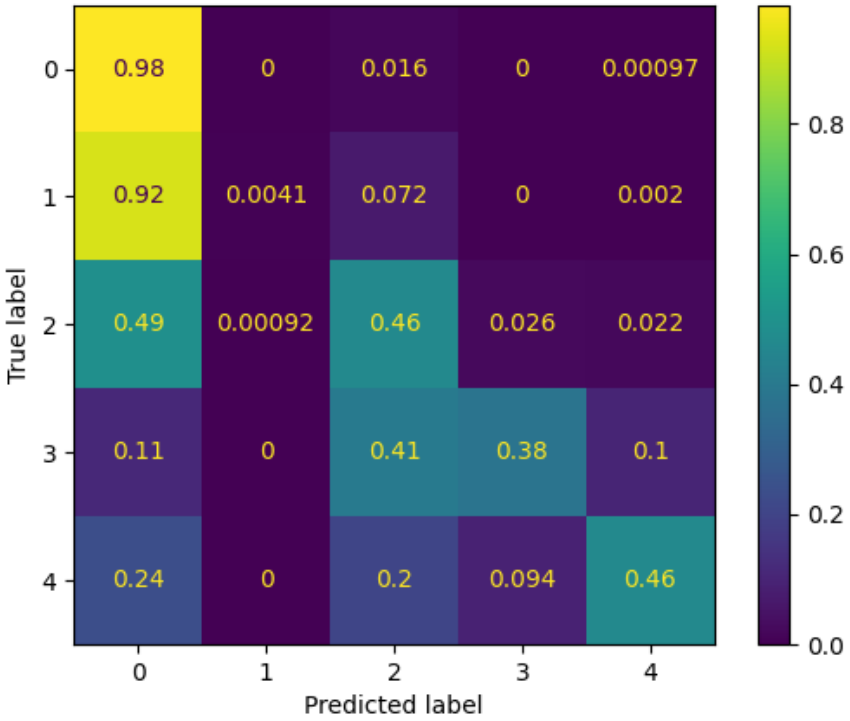
accuracy: 80.8%



ResNet50

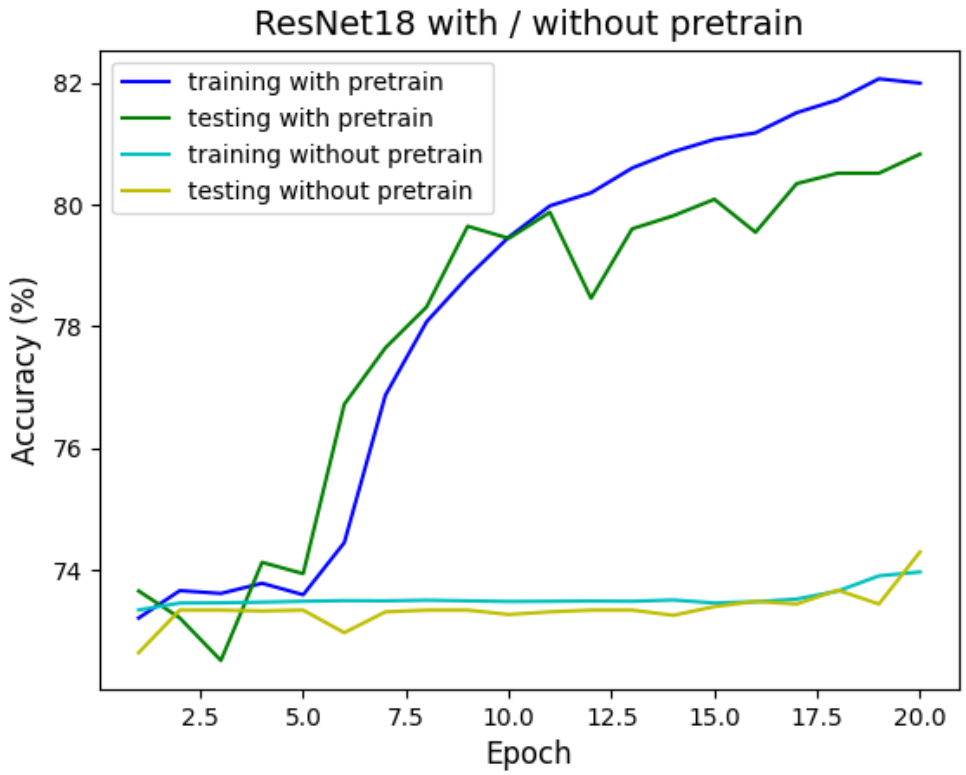
```
epoch 18:
trainig accuracy: tensor(82.8321, device='cuda:0')  loss: 886.6231434866786
testing accuracy: tensor(81.0107, device='cuda:0')

epoch 19:
trainig accuracy: tensor(83.0385, device='cuda:0')  loss: 867.6077251583338
testing accuracy: tensor(80.2135, device='cuda:0')

epoch 20:
trainig accuracy: tensor(83.1560, device='cuda:0')  loss: 867.049497641623
testing accuracy: tensor(80.5552, device='cuda:0')

max testing accuracy 81.01067352294922% at epoch 18
```

accuracy: 81%

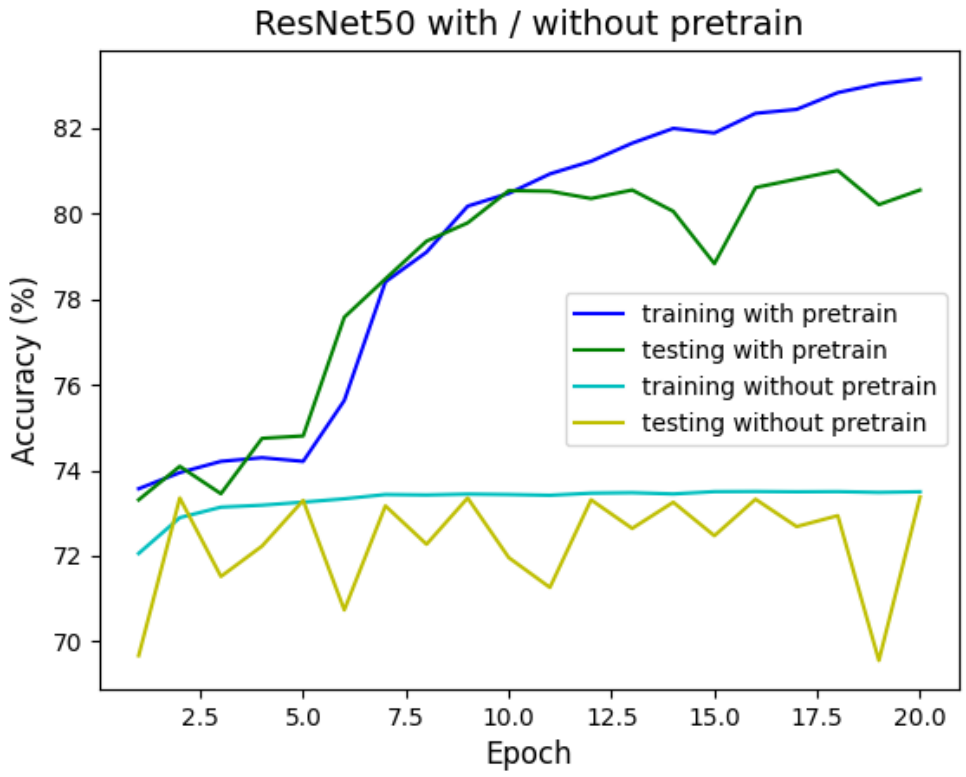


Comparison figures (ResNet18/50, with/without pretraining)

ResNet18



ResNet50



從上兩圖中可以看到，有用 pretrain 明顯比沒用 pretrain 表現得更好。另外，用 pretrain 因為前五個 epoch 在做 feature extraction，所以 accuracy 較低；過前五個 epoch 後 accuracy 會有一段明顯的提升。

Discussion

這組 training dataset 是一組 imbalance data，class 種類和其數量對應如下表:

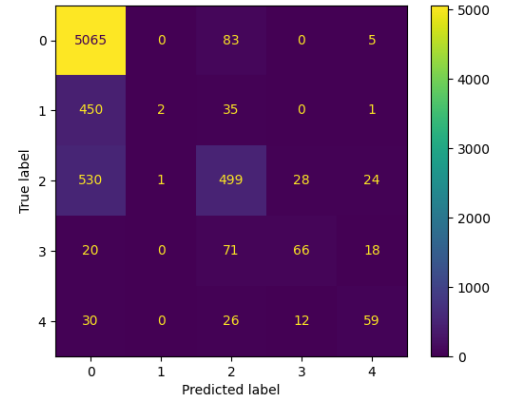
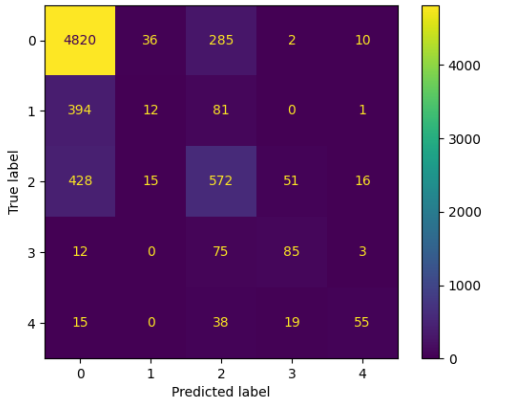
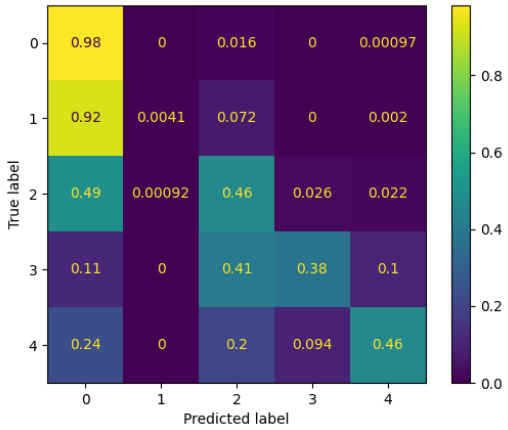
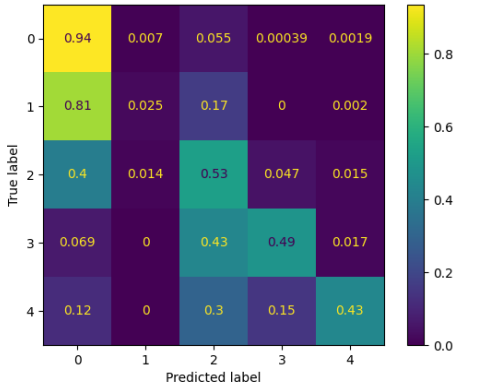
Label 0	20656	73.51%
Label 1	1955	6.96%
Label 2	4210	14.98%
Label 3	698	2.48%
Label 4	581	2.07%

根據 training dataset 的比例，若和 testing dataset 在分布上差不多，則全部輸出 0 都會有 73%的 accuracy。為了讓 model 對 label 0 之外的 classes 更敏感，我們可以在 CrossEntropyLoss 增加 classes weight 參數。

```
nSamples = [20656, 1955, 4210, 698, 581]
normedWeights = [1 - (x / 28100) for x in nSamples]
baseline = normedWeights[0]
normedWeights = [x / baseline for x in normedWeights]
print('each class weights: ' + str(normedWeights))
normedWeights = torch.FloatTensor(normedWeights).to(device)

loss_func = nn.CrossEntropyLoss(weight = normedWeights)
#loss_func = nn.CrossEntropyLoss()
num_epochs = 20
```

```
each class weights: [1.0, 3.5122246104245027, 3.2092960773777537, 3.681085437936593, 3.6968027941966684]
```

上圖的參數值意義為當 label 為 1 時，更新的 loss 量會是 label 0 的 3.51 倍，以此類推。這代表 model 會被 update 成更傾向預測 0 以外的 label。從以下的 confusion matrix 更能看出區別:

ResNet50 origin	ResNet50 with classes weight
	
	
Accuracy: 81%	Accuracy: 79%

從上表可以看出，我們犧牲了 label 0 的 accuracy，換取了其他 label 更高的 accuracy。但是 label 0 的數量還是太龐大了，因此 total accuracy 依然下降了 2%。若找到更適當的 classes weight 或 hyper parameters，就有可能使 total accuracy 提升。