DLP Lab4 Diabetic Retinopathy Detection

1. Introduction

利用有 pretrained 和無 pretrained 的 ResNet18 與 ResNet50 架構，訓練並分析由糖尿病所引發視網膜病變的分類問題，總分類數有 5 種，分別為 0 ~ 4，代表視網膜病變的嚴重程度。本次 Lab 所用的 dataset 有 35124 張照片，我們將 28099 張當作 training dataset，7025 張當作 testing dataset，照片的解析度為 512 x 512。

2. Experiment setups

- The details of your model (ResNet)

  直接從 torchvision 引入 ResNet18 和 ResNet50，當需要 pretrain 的時候就將 pretrained 的參數設為 True，並先做 feature extraction，只訓練最後一層幾個 epoch，然後再做 fine tune，訓練整個 model（所有 layer）剩餘的數個 epoch。

```
class ResNet18(nn.Module):
  def __init__(self, num_class, pretrained=False):
    super(ResNet18, self).__init__()
    self.model = models.resnet18(pretrained=pretrained)
    if pretrained:
      for param in self.model.parameters():
        param.requires_grad = False
    num_neurons = self.model.fc.in_features
    self.model.fc = nn.Linear(num_neurons, num_class)

  def forward(self, x):
    out = self.model(x)
    return out
```

- The details of your Dataloader

  因為 pytorch 的 convolution layer 需要(N, C, H, W)，所以透過 torchvision 中的 transforms function 對每張圖片做 transform，使 dataset 能餵進 Dataloader。為了提高 accuracy 和避免 overfitting，所以對 data 做 augmentation 和 normaliztion。

```python
class 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.transformations = transforms.Compose([transforms.RandomHorizontalFlip(),transforms.RandomVerticalFlip(),transforms.ToTensor(),
                                                  transforms.Normalize((0.375, 0.265, 0.186),(0.253, 0.178, 0.129))])
        print(">> Found {} images...".format(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
        """

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

        return img, label
```

- Describing your evaluation through the confusion matrix

  先建立一個 5x5 的 confusion matrix，並在 evaluating 的時候更新且統計數量，最後再依每一列做 normalization。

```python
def evaluate(model, loader_test, device, num_class):
  confusion_matrix = np.zeros((num_class, num_class))
  with torch.no_grad():
    correct = 0
    for _, data in enumerate(loader_test, 0):
      inputs = data[0].to(device)
      labels = data[1].to(device, dtype=torch.long)
      predict = model(inputs)
      predict_class = predict.max(dim=1)[1]
      correct += predict_class.eq(labels).sum().item()
      for i in range(len(labels)):
        confusion_matrix[int(labels[i])][int(predict_class[i])] += 1
    correct = (correct / len(loader_test.dataset)) * 100.0

  confusion_matrix = confusion_matrix / confusion_matrix.sum(axis=1).reshape(num_class, 1)
```

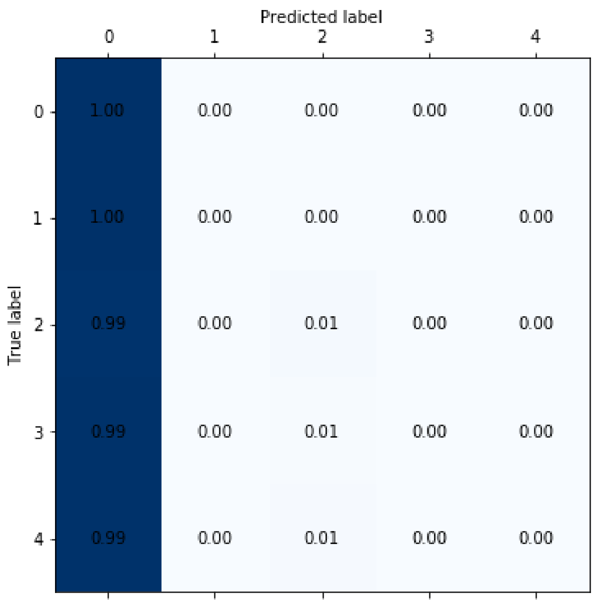
3. Experimental results

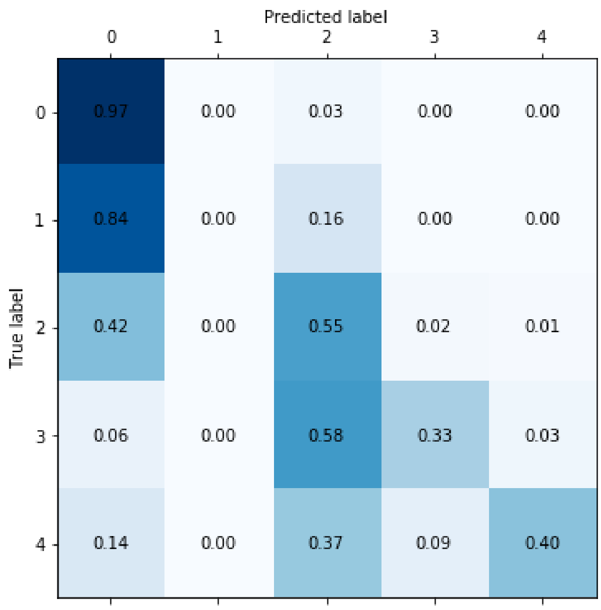
- The highest testing accuracy

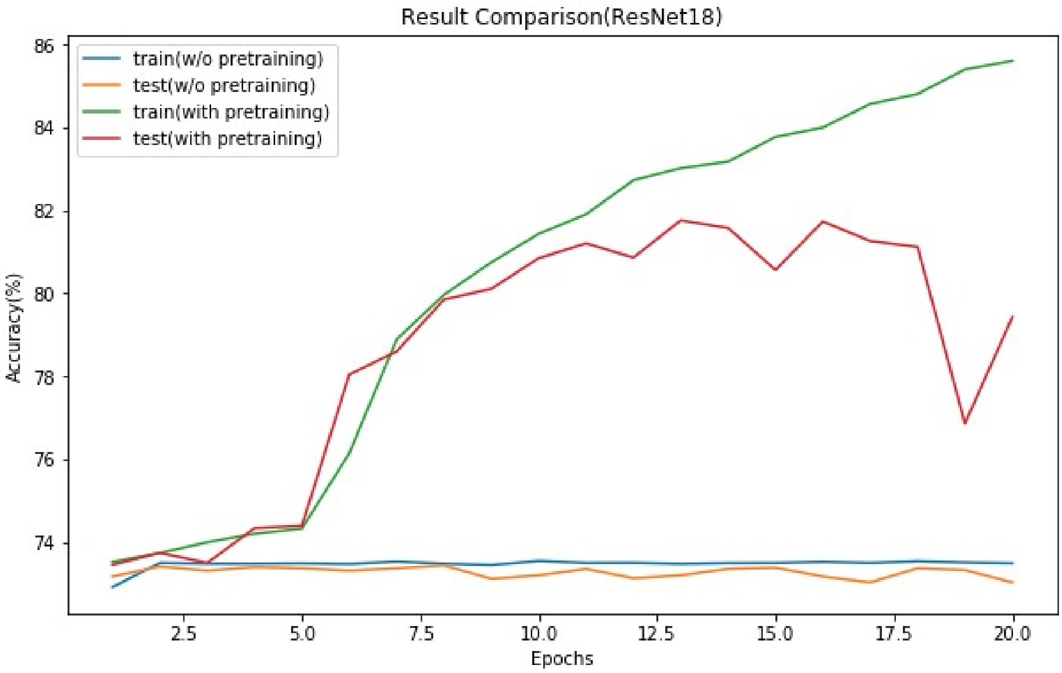
  最高的 testing accuracy 出現在有 pretrained 的 ResNet50 中，大約為 82.52%

	epoch	acc_train	acc_test
0	1	73.511513	73.722420
1	2	73.693014	74.249110
2	3	74.123634	74.733096
3	4	74.219723	74.832740
4	5	74.152105	74.106762
5	1	77.191359	79.416370
6	2	80.195025	80.341637
7	3	81.145236	77.693950
8	4	82.202214	81.252669
9	5	83.031425	80.199288
10	6	83.358838	81.665480
11	7	83.942489	82.519573
12	8	84.244991	81.323843
13	9	84.860671	81.594306
14	10	85.519058	82.192171
15	11	85.782412	80.882562
16	12	86.049326	81.551601
17	13	86.668565	79.516014
18	14	87.202392	81.879004
19	15	87.636571	80.911032

- Anything you want to present
  可以發現，無論是在 ResNet18 或 ResNet50 中，有 pretrained 的 training accuracy 與 testing accuracy 都比無 pretrained 還來得高，無 pretrained 的兩者 accuracy 差不多，但有 pretrained 的則是 ResNet50 的 accuracy 較高。
- Comparison figures
  - ResNet18

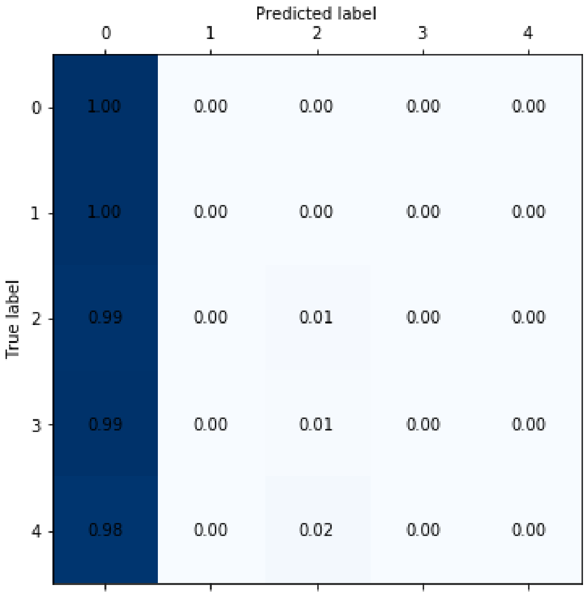




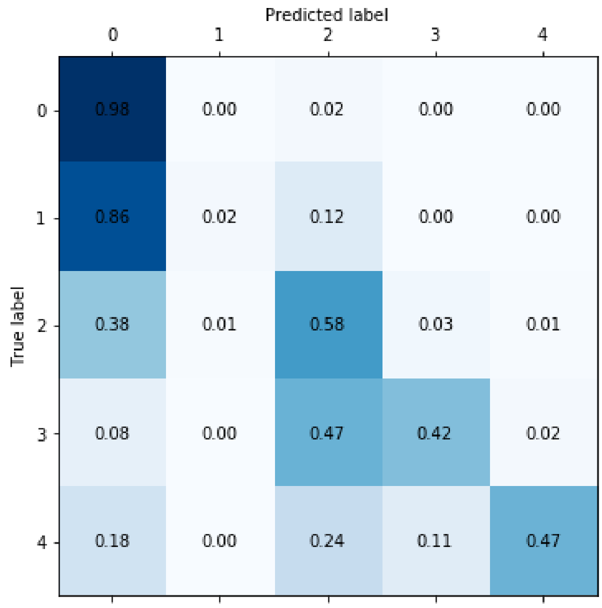


Result Comparison(ResNet18)
train(w/o pretraining)
test(w/o pretraining)
train(with pretraining)
test(with pretraining)
Accuracy(%)
Epochs

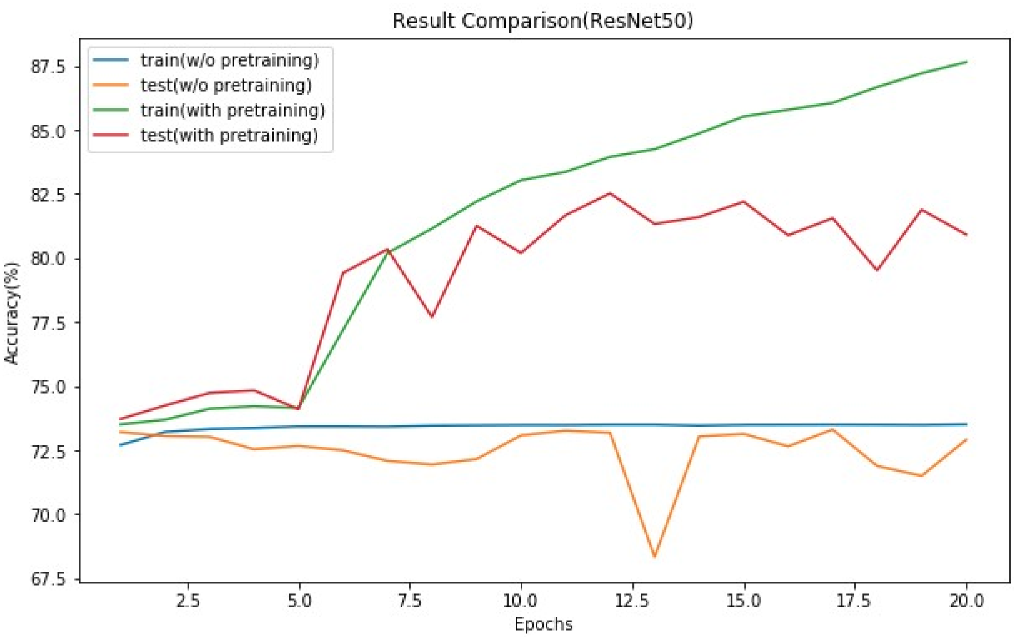
- ResNet50



Predicted label
True label
0 1 2 3 4
1.00 0.00 0.00 0.00 0.00
1.00 0.00 0.00 0.00 0.00
0.99 0.00 0.01 0.00 0.00
0.99 0.00 0.01 0.00 0.00
0.98 0.00 0.02 0.00 0.00



Predicted label
True label
0 1 2 3 4
0.98 0.00 0.02 0.00 0.00
0.86 0.02 0.12 0.00 0.00
0.38 0.01 0.58 0.03 0.01
0.08 0.00 0.47 0.42 0.02
0.18 0.00 0.24 0.11 0.47



Result Comparison(ResNet50)
train(w/o pretraining)
test(w/o pretraining)
train(with pretraining)
test(with pretraining)
Accuracy(%)
Epochs

- Discussion
  - Anything you want to share

    因為這次的 training dataset 很多，總共有 20899 張，所以在 training 時如果 batchsize 不小心調太大，會導致 GPU 記憶體不足無法計算，然後 ResNet50 又比 ResNet18 的參數量來得多，因此 ResNet50 所需的 batchsize 要比 ResNet18 更少，當然計算速度上又相對更久。