DL Lab3 report - Diabetic Retinopathy Detection

匯出pdf之後排版變得有點醜 😥😥
助教不介意的話可以到HackMD網址閱讀 謝謝 ~ ~ ~
https://hackmd.io/@ZdXM6gDQSTGTtZGr5jTo4Q/SJA4ueuh9 (https://hackmd.io/@ZdXM6gDQSTGTtZGr5jTo4Q/SJA4ueuh9)

Introduction

In this lab, we need to analysis diabetic retinopathy in the following three steps.

1. Write your own custom DataLoader through PyTorch framework.
2. Classify diabetic retinopathy grading via the ResNet architecture.
3. calculate the confusion matrix to evaluate the performance.

Requirments

- Implement the ResNet18、ResNet50 architecture and load parameters from a pretrained model
- Compare and visualize the accuracy trend between the pretrained model and without pretraining in same architectures, you need to plot each epoch accuracy during training phase and testing phase.
- Implement custom DataLoader
- Calculate the confusion matrix and plotting

Dataset

Diabetic Retinopathy Detection (kaggle)
https://www.kaggle.com/c/diabetic-retinopathy-detection#description (https://www.kaggle.com/c/diabetic-retinopathy-detection#description)

- Diabetic retinopathy is the leading cause of blindness in the working-age population of the developed world.
- This dataset provided with a large set of high-resolution retina images taken under a variety of

imaging conditions. Format: .jpeg

Class
0 - No DR
1 - Mild
2 - Moderate
3 - Severe
4 - Proliferative DR

Experiment setups

The details of your model (ResNet)

```
class ResNet(nn.Module):
    def __init__(self, num_classes=5, option='resnet18', pretrained=True):
        super(ResNet, self).__init__()
        if option == 'resnet18':
            self.model = models.resnet18(pretrained=pretrained)
        elif option == 'resnet50':
            self.model = models.resnet50(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_classes)

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

        return out
```

直接使用torchvision的ResNet18/50，然後把最後一層layer的output feature改為class的數量5。

```
ResNet18_w = ResNet(num_classes=5, option='resnet18', pretrained=True)
# feature extraction
params_to_update = []
for name, param in ResNet18_w.named_parameters():
    if param.requires_grad == True:
        params_to_update.append(param)
optimizer = optim.SGD(params_to_update, lr=lr, momentum=momentum, weight_decay=weight_de
_, _, _ = train_test(model=ResNet18_w, dataloaders=loader, criterion=criterion, optimize
# fine-tune
for param in ResNet18_w.parameters():
    param.requires_grad = True
optimizer = optim.SGD(ResNet18_w.parameters(), lr=lr, momentum=momentum, weight_decay=we
acc_train, acc_test, best_wts = train_test(model=ResNet18_w, dataloaders=loader, criteri
```

訓練時分為兩部分：feature extraction跟fine-tune，過程中也有嘗試過不做feature extraction直接做fine-tune，發現結果其實是差不多的。

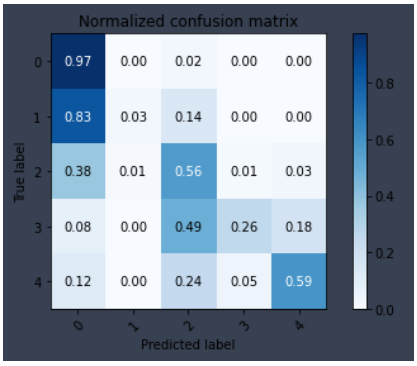
The details of your Dataloader

```
class RetinopathyLoader(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 l
        """
        self.root = root
        self.img_name, self.label = getData(mode)
        self.mode = mode

        if mode == 'train':
            self.transforms = transforms.Compose([transforms.RandomHorizontalFlip(),trar
                                                  transforms.Normalize([0.5769, 0.3852,
        else:
            self.transforms = transforms.Compose([transforms.ToTensor(),
                                                  transforms.Normalize([0.5769, 0.3852,

        print("> Found %d images..." % (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 resi
                  rotation, cropping, normalization etc. But at the beginning, I suggest

                  In the testing phase, if you have a normalization process during the t
                  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') #step1
        img = Image.open(path) #step1
        label = self.label[index] #step2
        img = self.transforms(img) #step3

        return img, label #step4


loader = {}

data_train = RetinopathyLoader(root='data', mode='train')
loader['train'] = DataLoader(dataset=data_train, batch_size=batch_size, shuffle=True)

data_test = RetinopathyLoader(root='data', mode='test')
loader['test'] = DataLoader(dataset=data_test, batch_size=batch_size, shuffle=False)
```

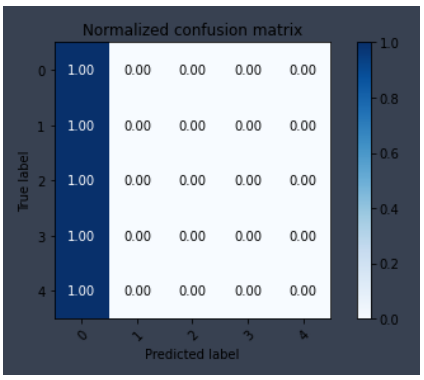
使用transfroms來實作data augmentation及normlization，然後使用PyTorch的DataLoader。

Describing your evaluation through the confusion matrix

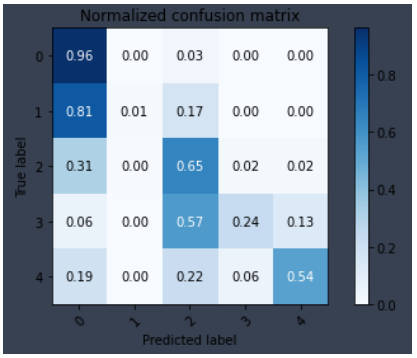
- ResNet18 with pretraining



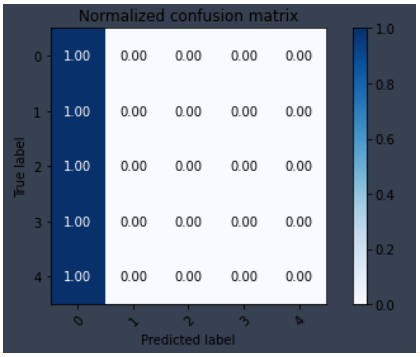
- ResNet18 without pretraining



- ResNet50 with pretraining



- ResNet50 without pretraining



可以發現without pretraining的model不管input什麼都會predict成class 0，但這樣acc也會有73~74%，表示data是非常imbalance的。

在with pretrainind的model中，可以發現class 0, 2, 4的分類的結果較好，大部分的data都可以被正確分類，而class 1大部分會被predict成class 0，class 3會被predict成class 2。

Experimental results

The highest testing accuracy

- Screenshot

  ResNet18 with pretrained

```
with torch.no_grad():
    preds = show_acc(option='resnet18', file_name='ResNet18_w_pretrained.pt', pretrained=True)
```

```
accuracy: 81.96%
```

  ResNet50 with pretrained

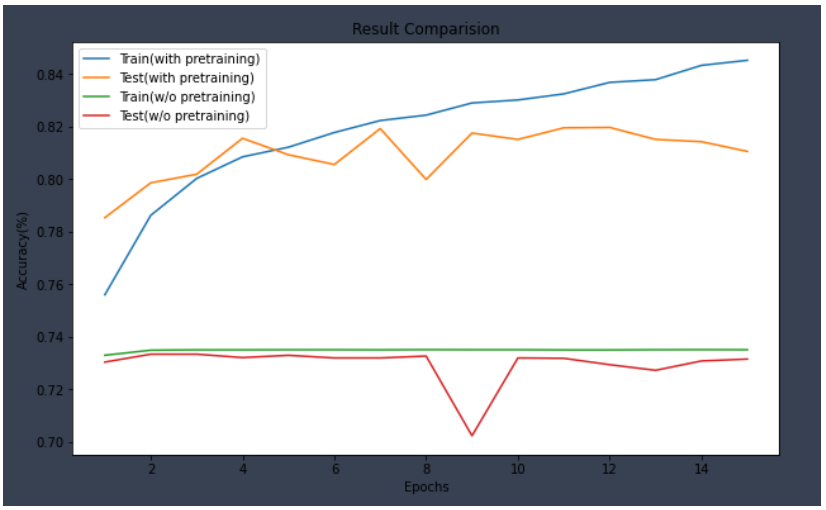
```
with torch.no_grad():
    preds = show_acc(option='resnet50', file_name='ResNet50_w_pretrained.pt', pretrained=True)
```

```
accuracy: 82.25%
```

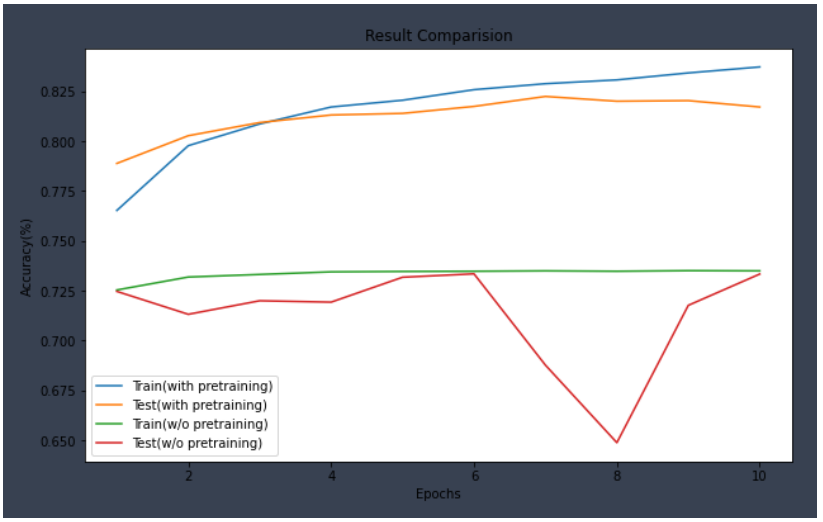
Comparison figures

- Plotting the comparison figures (RseNet18/50, with/without pretraining)

  ResNet18



  ResNet50



Discussion

- 可以發現在某些時候會有test data的accuracy高於train data的情形，我想可能是因為model會傾向於perdict成class 0，然後data是0的比例在test可能比train還要高，才會導致這種情形。
- 在run code的時候有發生out of memory的問題，發現有兩個原因：batch size設太大以及load best model進來做evaluate時沒有使用with torch.no_grad()。

- Hyper parameter

  batch_size = 8 *（更大會out of memory）*
  lr = 1e-3
  num_epochs_18 = 15 *（acc較epoch=10時上升）*
  num_epochs_50 = 10（*acc較epoch=5時上升）*
  momentum = 0.9
  weight_decay = 5e-4
  criterion = nn.CrossEntropyLoss()