NCTU DLP Lab3-Report
Diabetic Retinopathy Detection using ResNet

廖家鴻 0786009

2020/4/21

Outline

- ➢ **Introduction**
- ➢ **Experimental Setup**
  - A. The details of your model (ResNet)
  - B. The details of your Dataloader
  - C. Describing your evaluation through the confusion matrix
- ➢ **Experimental Result**
  - A. The highest testing accuracy
    - ◆ Screenshot
    - ◆ Anything you want to present
  - B. Comparison Figures
    - Plotting the comparison figures
      (RseNet18/50, with/without pretraining)
- ➢ **Discussion and extra experiments**

Introduction

- Experimental Setup
  - A. The details of your model (ResNet)
  - B. The details of your Dataloader
  - C. Describing your evaluation through the confusion matrix

在Experimental Setup中，會說明：
1. resnet在coding時，如何load pretrained weight、fc層的修改及如何做finetune。
2. 呈現dataloader的coding
3. 最佳model的confusion matrix解析。

- Experimental Result
  - A. The highest testing accuracy
    - ◆Screenshot
    - ◆Anything you want to present
  - B. Comparison Figures

    (RseNet18/50, with/without pretraining)

在Experimental result中，會呈現：
1. 最佳model的test_acc=82.6%，及其相關參數與model設定。
2. 兩張test_acc的training curve來比較resnet18&resnet50的pretraining與否之好壞。

- Discussion and extra experiments

最後，我會分享pytorch的tranform來做此應用場景的data augmentation的心得。以及大概比較resnet最後額外加一層fc的效果。

Experimental Setup

Experimental Setup-Detail of Model Setting

```python
if ResNet18:
    DLmodel = models.resnet18(pretrained=pretrain, progress=True)
    model_name = 'ResNet18'
    in_feat = DLmodel.fc.in_features
    #(224,224)input的話=512，(512,512)input的話=2048
else:
    DLmodel = models.resnet50(pretrained=pretrain, progress=True)
    model_name = 'ResNet50'
    in_feat = DLmodel.fc.in_features
    #(224,224)input的話=2048，(512,512)input的話=8912
```

ResNet的coding設置如左上圖，

其中pretrained=True的話代表會load pretrained weight進來，否則就是重頭train model。

另外，由於本次的視網膜病變嚴重程度分類五等，所以需要改resnet最後面全連接層(fc)的設置。

在接上5個output神經元前，resnet末端神經元個數會因為input圖片的像素而變(但resnet本身weights個數不變)。以resnet18來說，若input size=(224,224)，則有512neurons；若input size=(512,512)，則有2048neurons。同理，對resnet50來說，則分別為2048與8192個neurons。

而我以”DLmodel.fc.in_features”來因應各種不同input size來adaptive調整

```python
###### Replace the last fully-connected layer ######
### Parameters of newly constructed modules have requires_grad=True by default
num_classes = 5
mid_feat = mid_feat
model_name = model_name +'_addFC' +str(mid_feat)
if mid_feat == 0:
    DLmodel.fc = nn.Linear(in_feat, num_classes).to(device)
else:
    DLmodel.fc = nn.Sequential(nn.Linear(in_feat, mid_feat),
                               nn.ReLU(inplace=True),
                               nn.Linear(mid_feat, num_classes)).to(device)
```

左下圖則為是否再為resnet添加一層fc，並且若有添的話，此fc的activation為relu。

最後的output neurons都是添加為5個，因應5等嚴重程度。

Experimental Setup-Detail of Model Setting

```python
########### Create Model object and set it ot GPU ############
########### Define optimizer/loss function ################
def set_parameter_requires_grad(model, feature_extracting):
    if feature_extracting:
        for param in model.parameters():
            param.requires_grad = False
```

```python
DLmodel.to(device)
##### Decide to do finetuing or feature_extraction #####
##### finetuing (update all weights) && feature_extraction (Only train the FC-layer)
set_parameter_requires_grad(DLmodel, feat_ext)


##### Replace the last fully-connected layer #####
### Parameters of newly constructed modules have requires_grad=True by default
num_classes = 5
mid_feat = mid_feat
model_name = model_name +'_addFC' +str(mid_feat)
if mid_feat == 0:
    DLmodel.fc = nn.Linear(in_feat, num_classes).to(device)
else:
    DLmodel.fc = nn.Sequential(nn.Linear(in_feat, mid_feat),
                                nn.ReLU(inplace=True),
                                nn.Linear(mid_feat, num_classes)).to(device)
```

我另外設置了左上圖的function來決定model weights是否可以update，並且在resnet末端與接新的fc層之前調用此function。

若feature_extraction=True，則代表該次train的resnet本體的weights不update(無論pretrained weights有無load進來)，而只update fc層的參數。

若有load pretrained weights以及feature_extraction=False，則整個model的參數都會update，此時稱之為finetuning。

在我這次training實驗的觀察中，有load pretrained weights、然後有添加一層256的fc的model的效能最佳。

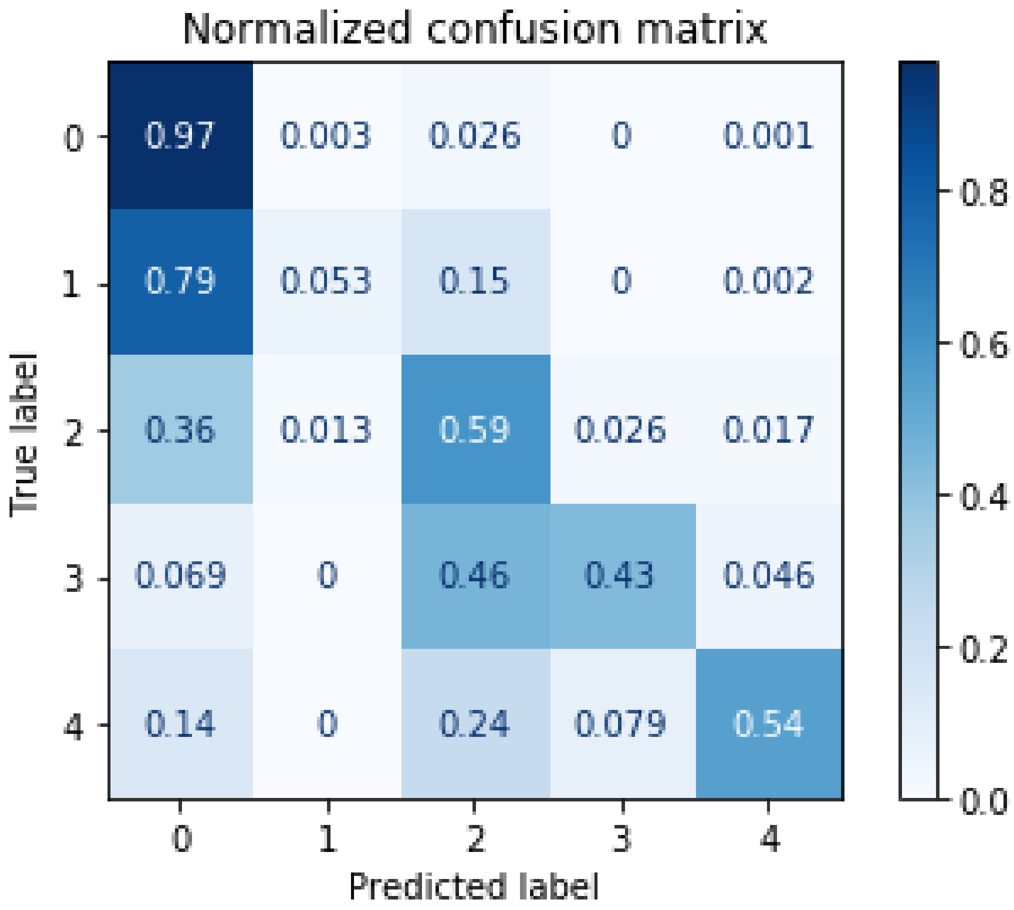
Experimental Setup-Detail of dataloader

```python
def __getitem__(self, idx):

    if torch.is_tensor(idx):
        idx = idx.tolist()
    #step1.Get the image path from 'self.img_name' and load
    img_fn = self.img_name[idx]+'.jpeg'   # img_fn = [self.
    img_fn = join(self.root,img_fn)
    image_fn = Image.open(img_fn).convert('RGB')

    #step2. Get the ground truth label from self.label
    label_fn = torch.Tensor([self.label[idx]]).long()

    #step3. Transform the images during the training phase
    #       Do normalization only during testing phase
    if self.transform:
        image_fn = self.transform(image_fn)
    else:
        image_fn = io.imread(img_fn)
        image_fn = np.array(image_fn)/ 255
        image_fn = torch.FloatTensor(image_fn)

    #step4. Return processed image and label
    sample = {'image': image_fn, 'label': label_fn}

    return sample
```

Dataloader的__getitem__的coding如左圖，按照助教的四個step來做。

至於transform的部分與相關心得，將在報告最後一部分討論。

Experimental Setup-Confusion Matrix



左圖即是本次lab我這邊跑最好的test_acc=82.6%的confusion matrix圖示。

Normalized的方式，其實類似算sensitivity，即matrix各條row上的個數值除以該所屬true label的總數，即橫向row element的值加總會等於1。(model判正確的數值為matrix裡左上到右下的對角線elements值)

而true label個數統計如左下圖所示，明顯為imbalance data的課題。因為label=0的個數最多，所以model似乎都傾向將各個input判成0這一類別，即data以正常的視網膜居多。

第二個現象是，雖然1這一類的總數在training時並非最少，但卻在test的inference時只有5%的案例會被判成功，而有79%會被判成正常的視網膜。

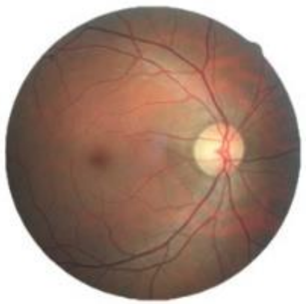
整體來說，1類很難判對，2~4類正確大約一半，而又此data是0類佔大多數，model傾向將input判成0類，也能將accuracy拉高。所以confusion matrix可以了解model判定各類別的能力到哪裡。

Test label統計：

```
{0: 5153, 1: 488, 2: 1082, 3: 175, 4: 127}
```

Training label統計：

```
{0: 20655, 1: 1955, 2: 4210, 3: 698, 4: 581}
```



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

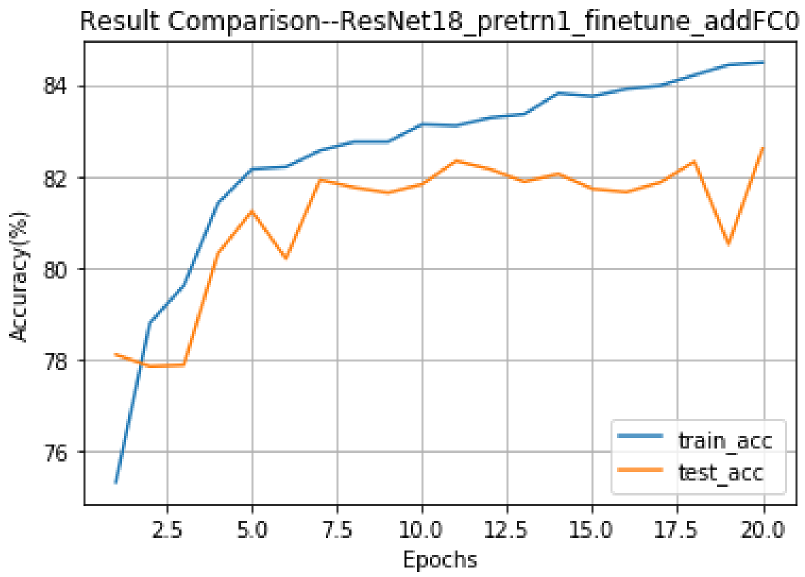
Experimental Results

Experimental Result – highest test_acc

```
Epoch 20/20
train_avg_loss: 0.4568, train_acc: 84.480%
test_avg_loss: 0.5464, test_acc: 82.605%

Result of ResNet18_pretrn1_finetune_addFC0:
The best test accuracy is 82.60% at epoch=20

Training time taken: 175.0 minutes 48.6 seconds
```

```
########### learning rate scheduling ######
def adjust_learning_rate(optimizer, epoch):
    if epoch < 3:
        lr = 0.005
    else: #epoch < 20:
        lr = 0.001
```

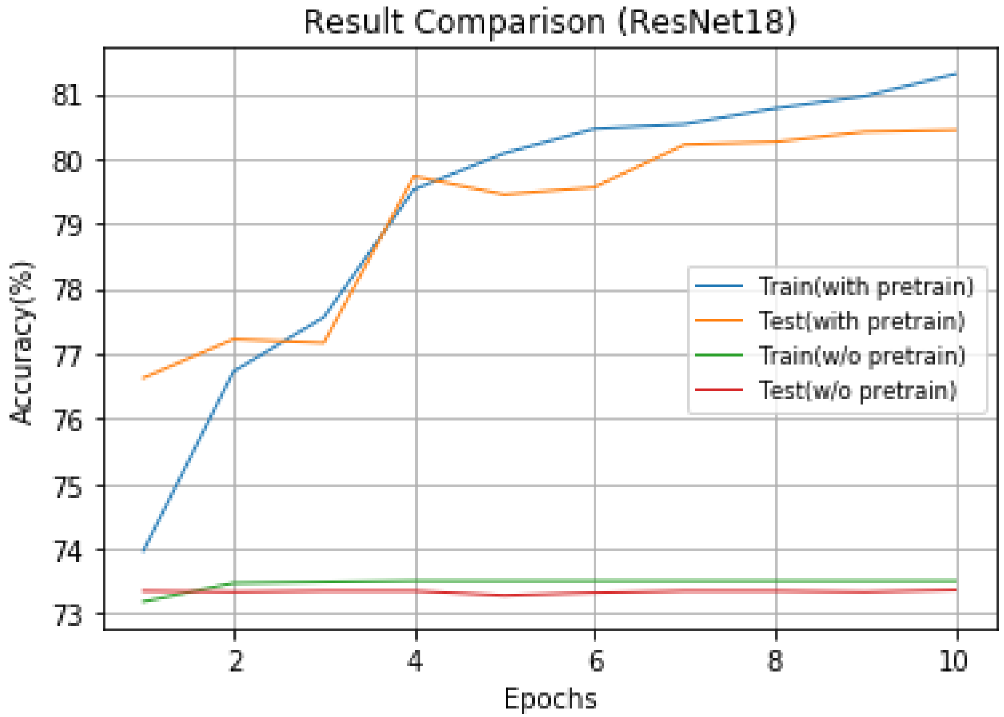
- Input size=512
- Batch size= 32
- Epochs = 20
- Optimizer: SGD
  Momentum = 0.9
  Weight_decay = 5e-4
- Loss function: CrossEntropyLoss()
- Resnet18 using pretrained weights
- Finetuing all weights and no additional immediate fc layer

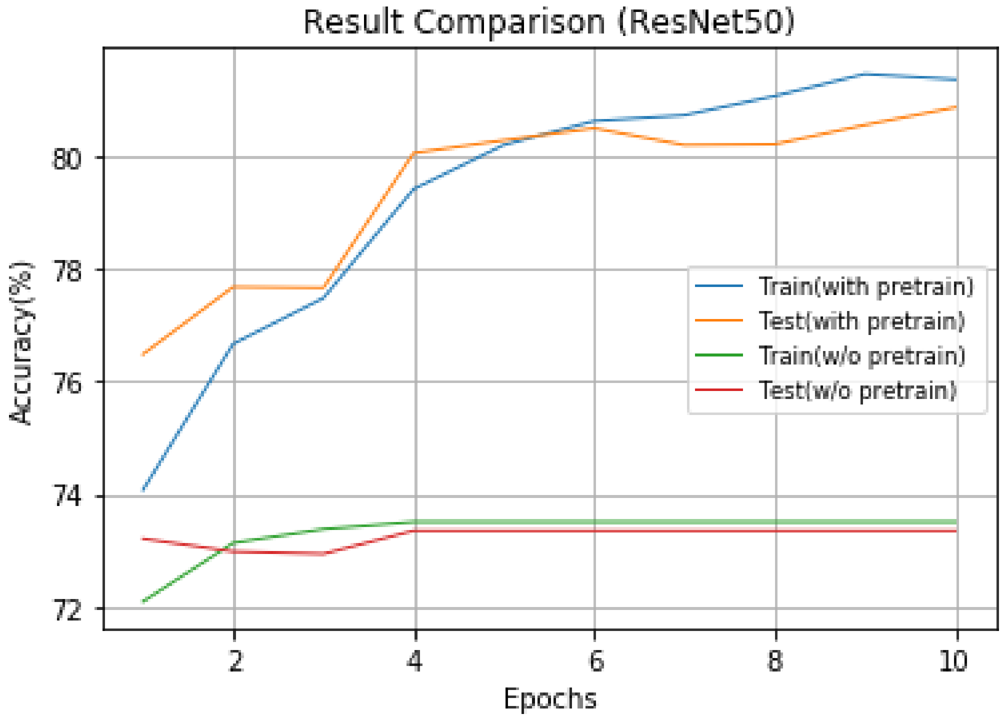



```
img_size=args.image_size
data_transforms = {
    'train':
        transforms.Compose([
            transforms.Resize((img_size,img_size)),
            # transforms.RandomResizedCrop((224,224)),
            transforms.RandomRotation(60, resample=False),
            # transforms.RandomAffine(0, shear=5, scale=(0
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            # normalize
            ]),
    'test':
        transforms.Compose([
            transforms.Resize((img_size,img_size)),
            transforms.ToTensor(),
            # normalize
            ])}
```

實驗結果：用resnet18的test_acc達到82.6%，transform使用如上圖、相關參數如右上list所示。

Experimental Result – Result comparision





實驗結果：

1. 不管是resnet18還是resnet50，明顯要都要用pretrained weight來finetune，accuracy才能train得上去。
2. 剛開始還不曉得在testing phase時，可以用torch.no_grad()來優化GPU memory的使用，所以input都resize成(400,400)，而test_acc在此最高也只能到81.62%

```
########### learning rate scheduling ######
def adjust_learning_rate(optimizer, epoch):
    if epoch < 3:
        lr = 0.005
    else: #epoch < 20:
        lr = 0.001
```

Discussion and Extra experiments

Discussion and extra experiments-1

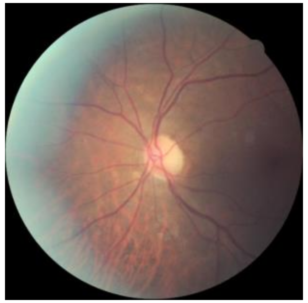
```python
############### train & test data preprocessing ##################
from RetinopathyLoader import RetinopathyLoader

# normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
img_size=args.image_size
data_transforms = {
    'train':
        transforms.Compose([
            transforms.Resize((img_size,img_size)),
            # transforms.RandomResizedCrop((224,224)),
            transforms.RandomRotation(60, resample=False), #expand=False, center=None),
            # transforms.RandomAffine(0, shear=5, scale=(0.95,1.05)),
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(), # normalize
            ]),
    'test':
        transforms.Compose([
            transforms.Resize((img_size,img_size)),
            transforms.ToTensor(), # normalize
            ])}
image_datasets = {
    'train':RetinopathyLoader(root='data/', mode='train', transform=data_transforms['train']),
     'test':RetinopathyLoader(root='data/', mode='test',  transform=data_transforms['test'])
                  }
dataloaders = {'train':DataLoader(image_datasets['train'],
                      batch_size=args.batch_size, shuffle=True, num_workers=4
                      ),
               'test':DataLoader(image_datasets['test'],
                      batch_size=args.batch_size, shuffle=False, num_workers=4
                      )}
```

Pytorch的transform設置如左圖，我曾在training phase嘗試用了resize、crop、affine、rotation、flip，心得如下：

RandomResizeCrop和RandomAffine的效果不好，推測原因在於：其實test data的樣子是視網膜圓圖圓心都會規規矩矩在中間，也不會變形，所以training去設定這個確實只會干擾model的性能。

Normalize若用normal distribution去調的話，原本內切圓與外切正方形之間，圖值為零的四個角落會變的不是零，這樣跟rotation同時做的時候可能會出現問題，因為雖然test data也做一樣的normalization，但test data是不rotate的，所以normalization確定只要到[0, 1]就好。

最後我只選用的resize、rotation、flip三種方式來做data augmentation。



Discussion and extra experiments-2


```
########### learning rate scheduling ######
def adjust_learning_rate(optimizer, epoch):
    if epoch < 3:
        lr = 0.005
    else: #epoch < 20:
        lr = 0.001
```

最後我有嘗試比較在resnet18末端是否加上一層FC with 256個neurons，比較起來：

1. 有加此FC的test_acc在初期爬升較快。
2. 從training過程來說，有加此FC的test_acc超過82%較多次一些，不過中間有段的”起伏”較沒加此fc的來的大。
3. 最後雖然是由沒加fc的model取勝
   (82.6% at epoch=20)，
   但沒再train更久，所以還是很難肯定有加此fc是否有比較好。

The End