311551148 lab4 Report

1. Introduction

這次 lab 我們要利用 ResNet18 以及 ResNet50 來實作糖尿病所引發的視網膜病變偵測，我們也需要實作 pytroch dataloader 以及資料集的 preprocessing，最後要使用 confusion matrix 來 evaluate model。

2. Experiments setups

甲、The details of your model

```
class resnet(nn.Module):
    def __init__(self, model, pretrained, num_class):
        super(resnet, self).__init__()

        self.model_name = model
        self.num_class = num_class

        if self.model_name == "resnet18":
            self.resnet = models.resnet18(pretrained=pretrained)
        elif self.model_name == "resnet50":
            self.resnet = models.resnet50(pretrained=pretrained)

        ## Reinitialize the last layer
        fc_in_dim  = self.resnet.fc.in_features
        fc_out_dim = self.num_class
        self.resnet.fc = nn.Linear(fc_in_dim, fc_out_dim)

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

model 的部分因為可以使用 pytorch 定義好的 model，所以很單純，我是使用 ResNet v1.5，相比原本 paper 提出的架構，ResNet v1.5 把 stride 移到 conv 3*3 的部分，可以有更快更好更穩定的結果。唯一要改動的事最後一層 classification layer，原本 pretrian 的模型是設計給 1000 class classification，但我們只需要做 5 個類別的分類，所以把他最後的 fully connect layer 改成 output 5 channel。

乙、The details of your dataloader

```
img_path = f"{self.root}/processed_data/{self.img_name[index]}.jpeg"
label = self.label[index]

## Define transformations
transformations = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.ToTensor(), ## Scales the pixel to the range [0, 1]
    transforms.Normalize((0.3749, 0.2602, 0.1857),(0.2526, 0.1780, 0.1291)) ## Normalization
])

img = Image.open(img_path)
img = transformations(img)

return img, label
```

附圖的 code 是__get_item__的 code，使用 torchvision 的 transform 來做 data augmentation，包含 random horizontal flip 以及 ramdom vertical flip，再把資料轉成 tensor 方便之後訓練使用，最後在把資料

normalize，因為 resize 在 data preprocessing 就已經做好了，所以讀進來的圖片都已經是 crop 好 512*512，使用 PIL 來讀取指定路徑的圖片檔案，最後 get item 回傳這個 index 所對應到的圖片以及相對應的 label。

丙、Describe your evaluation through the confusion matrix

```
## Calculate confusion matrix
num_class = len(set(labels.tolist()))
confusion_matrix = np.zeros((num_class, num_class))
for idx in range(len(labels)):
    confusion_matrix[labels[idx]][preds[idx]] += 1

## Normalize
confusion_matrix = confusion_matrix / confusion_matrix.sum(axis=1)[np.newaxis].T

## Plot
textcolors = ("black", "white")
fig, ax = plt.subplots()
img = ax.imshow(confusion_matrix, cmap=plt.cm.Blues)
for i in range(confusion_matrix.shape[0]):
    for j in range(confusion_matrix.shape[1]):
        ax.text(
            j, i, "{:.2f}".format(confusion_matrix[i, j]),
            ha="center", va="center",
            color=textcolors[confusion_matrix[i, j] > 0.5]
        )

plt.colorbar(img)
plt.title(f"Normalized Confusion Matrix (ResNet{args.model[-2:]})")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")

pretrained_str = "w" if args.pretrained else "wo"

plt.savefig(f"{args.result_path}/cm_{args.model}_{pretrained_str}_pretrained.png")
```

上面是我 plot confusion matrix 的 code，input 會是 ground truth label 以及 prediction，因為總共有 5 個 class，所以 confusion matrix 會使一個 5*5 matrix，其中(I, j)代表的是 ground truth = I 被 predict 成 j 的 sample 數量，所以走訪一次 prediction list 就可以把 confusion matrix 計算出來，有計算好的 5*5 confusion matrix 後使用 plt 畫出來。

3. Data preprocessing

甲、How you preprocessing your data?

```
for i in tqdm(glob.glob(os.path.join("data", "*.jpeg"))):
    new_path = i.replace('data', 'processed_data')
    img = Image.open(i)
    width, height = img.size
    new_width = min(width, height)

    left = (width - new_width)/2
    top = (height - new_width)/2
    right = (width + new_width)/2
    bottom = (height + new_width)/2

    # Crop the center of the image
    im = img.crop((left, top, right, bottom))
    im = im.resize((512,512))
    im = im.save(new_path)
```

我把圖片讀進來之後，取 min(width, height)當作新圖片的長寬，會想這樣取是因為觀察到每張圖片長寬都不一定，並且幾乎都是上下的眼

睛會被切割到，所以直接取短邊當做新圖的長寬可以最小化流失的資料，計算出 center crop 的上下左右座標，使用 PIL crop 擷取圖片中間的部分，之後再把他 resize 成 512*512，另外存起來。

乙、What makes your method special?

這個方法很簡單，沒有使用到 detection 的部分，所以執行起來會比較快，另外幾乎沒有 data loss，並且因為是 center crop 成正方形，所以 resize 後眼睛還是會是正圓，在 data preprocessing 完把比較小的圖片存起來，之後 data loader 讀檔案的時候應該是會比較快。

4. Experiments result

甲、The highest testing accuracy

| | | |
|---|---|---|
| ResNet18 | With pretrain | 0.8219 |
| | Without pretrain | 0.7340 |
| ResNet50 | With pretrain | 0.8288 |
| | Without pretrain | 0.7335 |

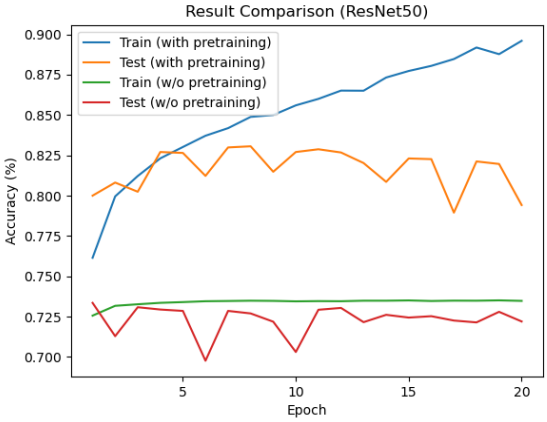
i. Screenshot


ii. Anything you want to present

上面的表格是兩種模型各自 with pretrain/ without pretrain 的 testing accuracy，可以看到 ResNet50 with pretrain 有最高的 testing accuracy，也可以發現 pretrain weight 對 testing 有很大的幫助，雖然 ResNet50 比 ResNet18 複雜很多，但是加上 pretrain weight 後 ResNet18 比 ResNet50 表現還好。

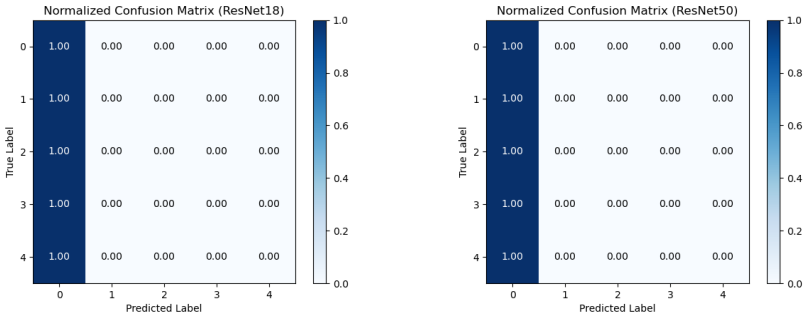
乙、Comparison figure

i. Plotting the comparison figurers

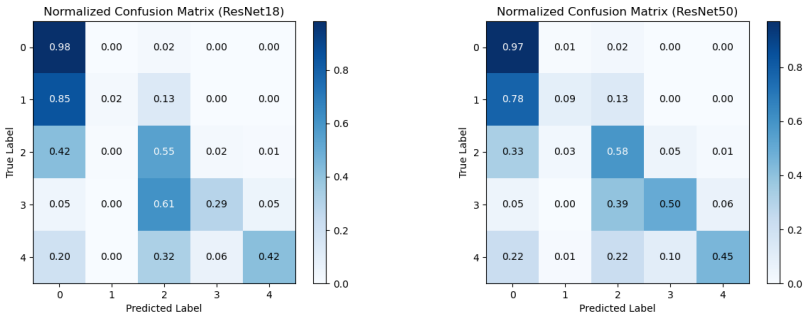




上圖畫出各個模型各自 train test 的 accuracy curve，可以看出 pretrain weight 非常重要

ii. (ResNet18/50, with/without pretraining)




上圖是沒有使用 pretrain weight 兩個模型的 confusion matrix，可以看出來所有的 sample 都被歸類到 class 0，因為資料集中大部分的 sample 都是 class 0，所以模型只要不管怎樣都預測 class 0 就可以得到蠻低的 loss，導致 performance 很差，後面可以看到要是使用 pretrain weight，就可以稍微解決這個問題，可以讓模型更能學習到圖片中特徵跟 label 的對應。

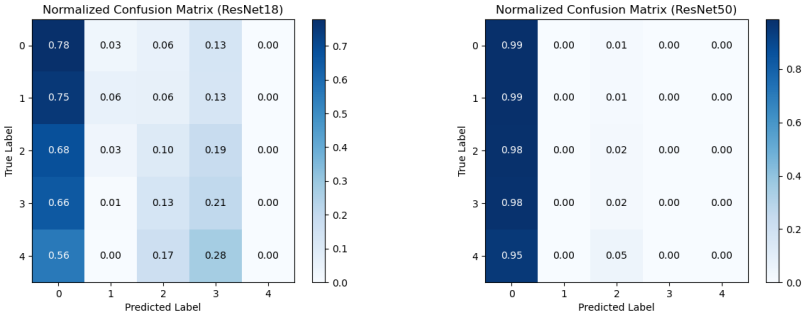



上圖是兩個模型使用 pretrain weight 後訓練完的 confusion matrix，可以很明顯看出來比沒有使用 pretrain weight 好非常多，從這次實驗可以看出使用 pretrain 在大型資料集的參數當初始參數可以解決資料集 inbalence 問題。

5. Discussion

甲、Anything you want to share

有注意到訓練資料非常不平衡，可以使用一些簡單的方法處理，像是在 loss function 給不同的類別不同的 loss weight，給那些比較少資料的類別更高的 weight，讓模型更注意這些類別，不然模型直接通通預測數量最多的那個類別也可以得到蠻小的 loss。




加上 loss weight 後的 testing accuracy 雖然有下降，但是從 confusion matrix 中可以看到模型不再是通通預測成 class 0，代表 loss weight 對 imbalance dataset 是有幫助的。