1.Introduction

ResNet 全名為 Residual Neural Network。在 ResNet 出現以前，太深的神經網路架構常出現梯度消失的問題，因而難以訓練。如下圖，ResNet 網路架構中設計了 Residual Block 單元，新增了一條捷徑用來複製前層的輸出，透過此捷徑，當前層作 back-propagation 時，如果當層的參數逼近於零時，仍可以選擇捷徑的路徑，跳過當層，直接回流至後層。在 ResNet 出現後，以 residual block / residual learning 為主架構的網路接連地在各個論文中出現，正式開啟了深層數網路的時代。

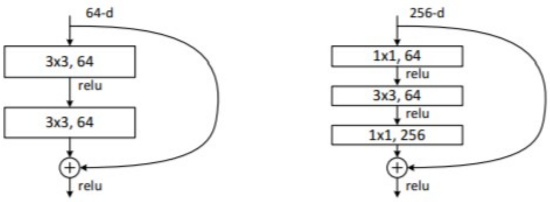


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

2.Experiment setups

A.The details of your model (ResNet)

直接 from torchvision import models，這次實驗中用到了 ResNet-18 和 ResNet-50。
需修改最後一層 layer 的 output feature 為 5 種 classes。
在宣告類別時，pretrained=True 它就會自己下載 torch.utils.model_zoo 提供的 pretrained model。在使用 pretrained model 做訓練時，先做 feature extraction ，只訓練最後一層 layer，用 pretrained model 對所有當前訓練圖片提取特徵（這些圖片是這是任務要處理的新問題），把所有圖片對應的特徵存儲起來，作為新的訓練輸入。過幾個 epoch 後，再做 finetuning，構造一個新的淺層網絡，訓練全部 layer，數個 epoch 後得到最後結果。

```
"""
resnet50 with pretrained weights
    first feature extraction for few epochs, then finefuning for some epochs
"""
model_with=ResNet50(num_class=num_class,pretrained=True)
# feature extraction
print('~~~ feature extraction ~~~')
params_to_update=[]
for name,param in model_with.named_parameters():
    if param.requires_grad:
        params_to_update.append(param)
optimizer=optim.SGD(params_to_update,lr=lr,momentum=momentum,weight_decay=weight_decay)
df_firststep=train(model_with,loader_train,loader_test,Loss,optimizer,epochs_feature_extraction,device,num_class,'re
# finetuning
print('~~~ finetuning ~~~')
for param in model_with.parameters():
    param.requires_grad=True
optimizer=optim.SGD(model_with.parameters(),lr=lr,momentum=momentum,weight_decay=weight_decay)
df_secondstep=train(model_with,loader_train,loader_test,Loss,optimizer,epochs_fine_tuning,device,num_class,'resnet5
df_with_pretrained=pd.concat([df_firststep,df_secondstep],axis=0,ignore_index=True)
```

```python
class ResNet50(nn.Module):
    def __init__(self,num_class,pretrained=False):
        """
        Args:
            num_class: #target class
            pretrained:
                True: the model will have pretrained weights, and only the last
                False: random initialize weights, and all layer's 'require_grad'
        """
        super(ResNet50,self).__init__()
        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_class)

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

B.The details of your Dataloader

需要實做出 RetinopathyLoader 類別，然後放進 torch.utils.data 提供的 DataLoader 裡。

```python
dataset_train=RetinopathyLoader(img_path='data',mode='train')
loader_train=DataLoader(dataset=dataset_train,batch_size=batch_size,shuffle=True,num_workers=4)
```

先繼承了 torch.utils.data 的 Dataset，然後覆寫他的__getitem__，讓 DataLoader 能從路徑找到 input 和 target。並在建構函式__init__中對圖片做正規化處理。

```python
class RetinopathyLoader(data.Dataset):
    def __init__(self, img_path, mode):
        """
        Args:
            img_path: Root path of the dataset.
            mode: training/testing

            self.img_names (string list): String list that store all image names.
            self.labels (int or float list): Numerical list that store all ground truth label values.
        """
        self.img_path = img_path
        self.mode = mode
        self.img_names=np.squeeze(pd.read_csv('train_img.csv' if mode=='train' else 'test_img.csv').values)
        self.labels=np.squeeze(pd.read_csv('train_label.csv' if mode=='train' else 'test_label.csv').values)
        assert len(self.img_names)==len(self.labels),'length not the same'
        self.data_len=len(self.img_names)
        self.transformations=transforms.Compose([transforms.RandomHorizontalFlip(),transforms.RandomVerticalFlip(),trans
                                                 transforms.Normalize((0.3749, 0.2602, 0.1857),(0.2526, 0.1780, 0.1291))])
        print(f'>> Found {self.data_len} images...')
    def __len__(self):
        return self.data_len
    def __getitem__(self, index):
        single_img_name=os.path.join(self.img_path,self.img_names[index]+'.jpeg')
        single_img=Image.open(single_img_name)  # read an PIL image
        img=self.transformations(single_img)
        label=self.labels[index]
        return img, label
```

C. Describing your evaluation through the confusion matrix

用 5*5 的 table 視覺化 label 和 predict 結果。並做正規化處理，使每一行加起來為 1。

```python
def evaluate(model,loader_test,device,num_class):
    print('------ start evaluate ------')
    """
    Args:
        model: resnet model
        loader_test: testing dataloader
        device: gpu/cpu
        num_class: #target class
    Returns:
        confusion_matrix: (num_class,num_class) ndarray
        acc: accuracy rate
    """
    confusion_matrix=np.zeros((num_class,num_class))

    with torch.set_grad_enabled(False):
        model.eval()
        correct=0
        for images,targets in loader_test:
            images,targets=images.to(device),targets.to(device,dtype=torch.long)
            predict=model(images)
            predict_class=predict.max(dim=1)[1]
            correct+=predict_class.eq(targets).sum().item()
            for i in range(len(targets)):
                confusion_matrix[int(targets[i])][int(predict_class[i])]+=1
        acc=100.*correct/len(loader_test.dataset)

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

    return confusion_matrix,acc
```

3. Experimental results

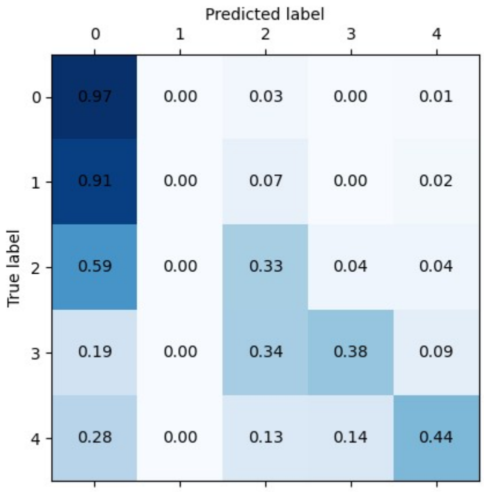
A. The highest testing accuracy

	epoch	acc_train	acc_test
0	1	72.248123	72.654804
1	2	73.493719	71.729537
2	3	73.507954	71.971530
3	4	73.507954	73.181495
4	5	73.507954	72.982206
5	6	73.507954	72.412811
6	7	73.507954	73.252669
7	8	73.493719	70.434164
8	9	73.504395	68.996441
9	10	73.511513	67.786477

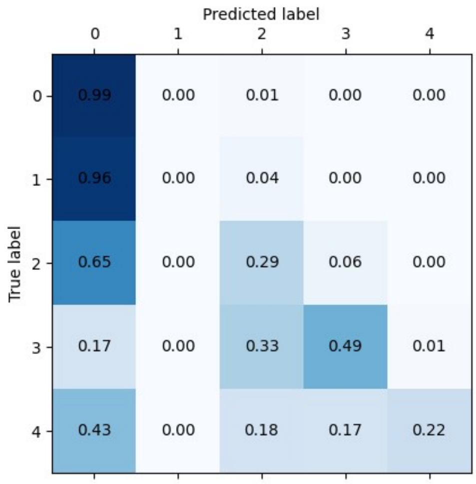
	epoch	acc_train	acc_test
0	1	57.671091	55.772242
1	2	57.728033	74.106762
2	3	57.859710	73.523132
3	4	57.945123	37.565836
4	1	70.611054	72.028470
5	2	74.429695	75.444840
6	3	76.063205	77.665480
7	4	77.081035	78.619217
8	5	77.565038	75.928826
9	6	77.988541	75.345196

B. Comparison figures

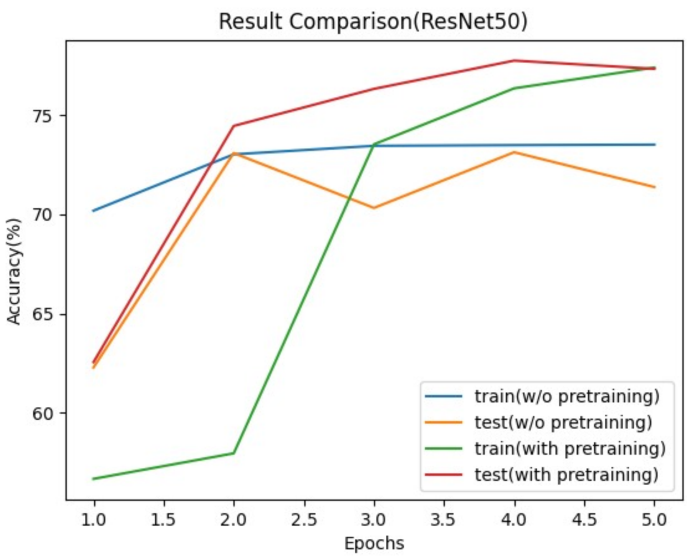
ResNet50 (with pretrained weights)

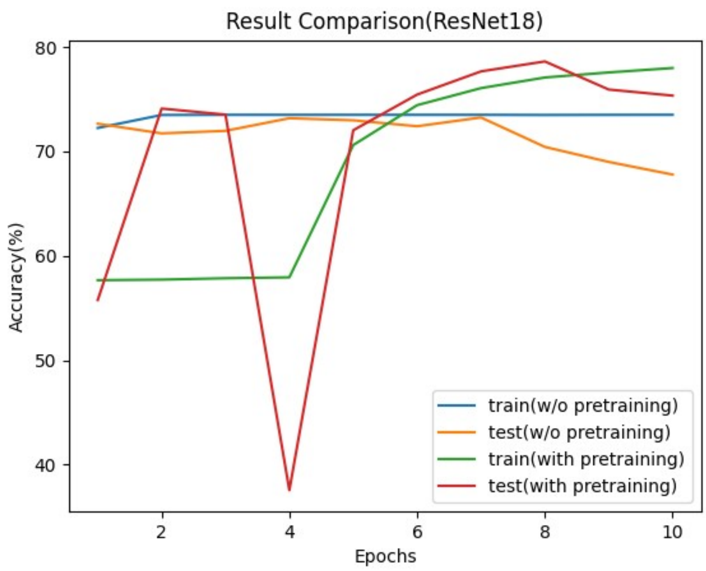


ResNet18 (with pretrained weights)



Result Comparison





4. Discussion

A. transfer learning

實驗結果明顯看出，有 pretrained model 的明顯較佳。這次實驗中，因為 Target Data（與進行的任務直接相關的資料）為壞的資料集（過度不平衡），沒有 pretrained model 的在訓練過程中幾乎不會進步。transfer learning 將訓練好已經學習過的 pretrained model 繼承給這次任務的欲訓練 model，省去了重新從頭訓練所需要的工作，降低訓練時特徵提取時間與淺層網絡訓練時間。應該是官方提供的 ResNet pretrained model 已經非常好，因而能順利運用在其他分類應用上。

B. a manual rescaling weight given to each class

因為資料過度不平衡，在使用 torch.nn.CrossEntropyLoss 作為 loss function 時，
給予不同類別不一樣的權重值。