Lab3 Diabetic Retinopathy Detection

311554046 林愉修

1. Introduction

分別使用 ResNet18 以及 ResNet50 來做 transfer learning (用pretrained weights) 和直接對 training dataset 做 learning (Random initialized weights)，任務是對視網膜的照片去做分類，有 0 ~ 4 共 5 種類別，代表視網膜病變的嚴重程度，數字越大代表越嚴重，並且比較兩種 model 以及有沒有用 pretrained weights 的差異。

我的程式設計為先跑 pretrained 再跑沒有 pretrained 的 model，並可以利用 argparse 來傳入 ResNet18、50的選擇以及超參數等等。

```python
def parse_argument():
    parser = argparse.ArgumentParser()
    parser.add_argument('-b', type=int, default=4, help='Batch size, default: 4')
    parser.add_argument('-c', type=int, default=5, help='Number of classes in dataset, default: 5')
    parser.add_argument('-t', type=check_network_type, default=18, help='ResNet18 (18) or ResNet50 (50), default: 18')
    parser.add_argument('-f', type=bool, default=True, help='Feature extract or not, default: True')
    parser.add_argument('-nf', type=int, default=5, help='How many epochs to train featrue extraction, then fine-tuning, default: 5')
    parser.add_argument('-e', type=int, default=5, help='Number of epochs, default: 5')
    parser.add_argument('-l', type=float, default=1e-3, help='Learning rate, default: 1e-3')
    parser.add_argument('-m', type=float, default=0.9, help='Momentum, default: 0.9')
    parser.add_argument('-w', type=float, default=5e-4, help='Weight decay, default: 5e-4')
    parser.add_argument('-wl', type=bool, default=False, help='Class-weighted loss, default: False')

    return parser.parse_args()

def check_network_type(input):
    int_value = int(input)

    if int_value != 18 and int_value != 50:
        raise argparse.ArgumentTypeError(f"Network type should be 18 or 50")

    return int_value
```

2. Experiment setups

A. The details of your model (ResNet)

使用 torchvision.models 內的 resnet18 以及 resnet50，將 fully-connected layer 的 output_feature 設為 5 ，並且 pretrained weights 分別使用 ResNet18_Weights.IMAGENET1K_V1 以及 ResNet50_Weights.IMAGENET1K_V2，而 random initialized weights model 則直接將 weights 設為 `None` 即可。

> Transfer learning 可分為以下兩種：

> - Feature extraction
>   將 pretrained model 前面幾層所有的 parameters 皆 freeze 住，只訓練並更新最後一個 layer 的 parameters。
> - Fine-tuning
>   將 pretrained model 的所有 parameters 都隨著訓練做更新(也有人說是有 unfreeze 一些 layer 的 parameters) 。

我將 pretrained model 設計為先做 feature extraction，再做 fine-tuning，因為聽說這樣比較快達到收斂，並且feature extraction 的 epoch 數可以自己設定。

- ResNet class
  為 torch.nn.Module 的 subclass，並且可以設定 output_feature 個數，使用 ResNet18 或是 ResNet50 等等。

```
class ResNet(nn.Module):
    def __init__(self, num_classes=5, net_type=18, weights=None, feature_extract=False):
        super(ResNet, self).__init__()

        self.feature_extract = feature_extract

        if weights is not None:
            self.mode = 'with pretraining'
        else:
            self.mode = 'w/o pretraining'

        self.net_type = net_type

        if net_type == 50:
            self.model = resnet50(weights=weights)
        else:
            self.model = resnet18(weights=weights)

        set_parameter_require_grad(self.model, self.feature_extract)
        self.num_ftrs = self.model.fc.in_features
        self.model.fc = nn.Linear(self.num_ftrs, num_classes)

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

        return x

def set_parameter_require_grad(model, feature_extracting):
    if feature_extracting:
        for param in model.parameters():
            param.requires_grad = False
    else:
        for param in model.parameters():
            param.requires_grad = True
```

B. The details of your Dataloader

RetinopathyLoader 為torch.utils.data.Dataset 這個 abstract class 的 subclass，並實作其 function，其中我將 transforms 作為instance variable，來存放 torchvision.transforms 的 function，並且在 `__getitem__` 先利用 PIL.Image.open 讀入照片，再將照片經過 transforms 處理，接著抓取該照片的對應 label。

```
class RetinopathyLoader(data.Dataset):
    def __init__(self, root, mode, transforms=None):
        """
        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.transforms (torchvision.transforms): Data transforms function.
        """
        self.root = root
        self.img_name, self.label = getData(mode)
        self.mode = mode
        self.transforms = transforms
        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):
        img_path = os.path.join(self.root, f'{self.img_name[index]}.jpeg')
        img = Image.open(img_path)

        if self.transforms is not None:
            img = self.transforms(img)

        label = self.label[index]

        return img, label
```

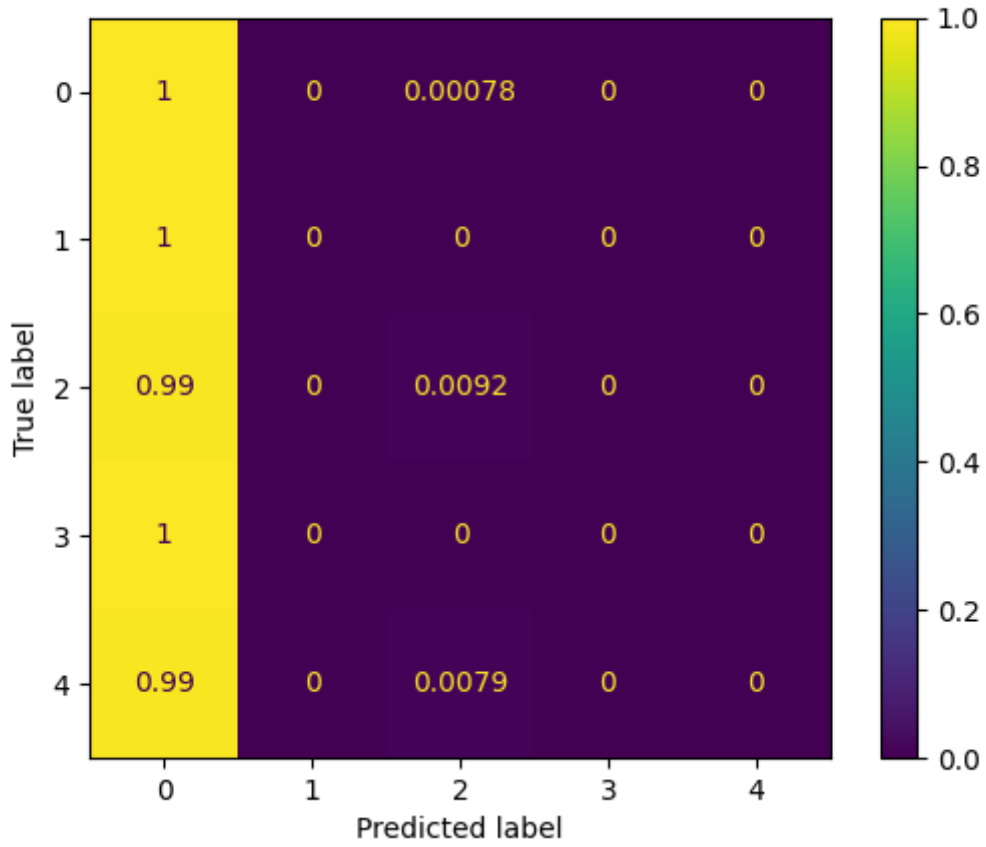
實作好 Dataset 後，將其放入 torch.utils.data.DataLoader 以 mini batch 來取得 input 及 label。

```
data_transform = transforms.Compose([
        transforms.RandomHorizontalFlip(),
        transforms.RandomVerticalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
print(f"Initializing Datasets and Dataloaders...")

image_dataset = {x: RetinopathyLoader('data/', x, data_transform) for x in ['train', 'test']}

dataloaders_dict = {x: DataLoader(image_dataset[x], batch_size=batch_size, shuffle=True, num_workers=4) for x in ['train', 'test']}
```

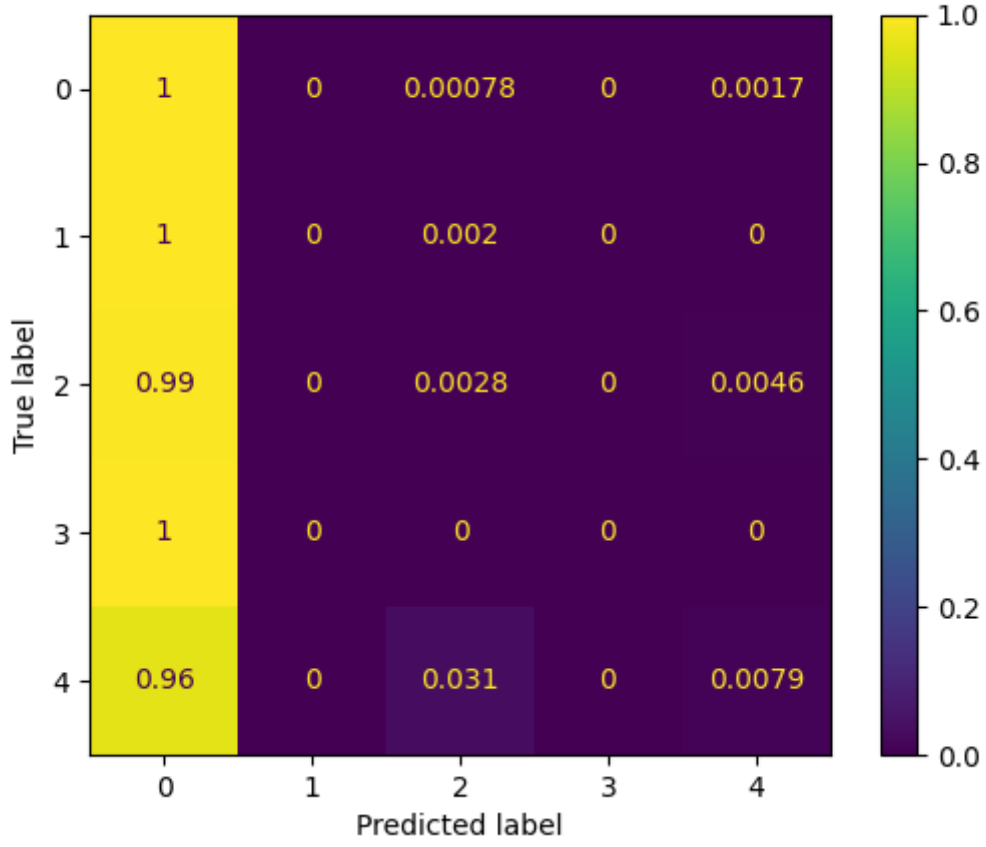
C. Describing your evaluation through the confusion matrix

我的 Confusion Matrix 是以 row (True label) 去做 Normalize 的。

- ResNet18 (w/o pretraining) Confusion Matrix

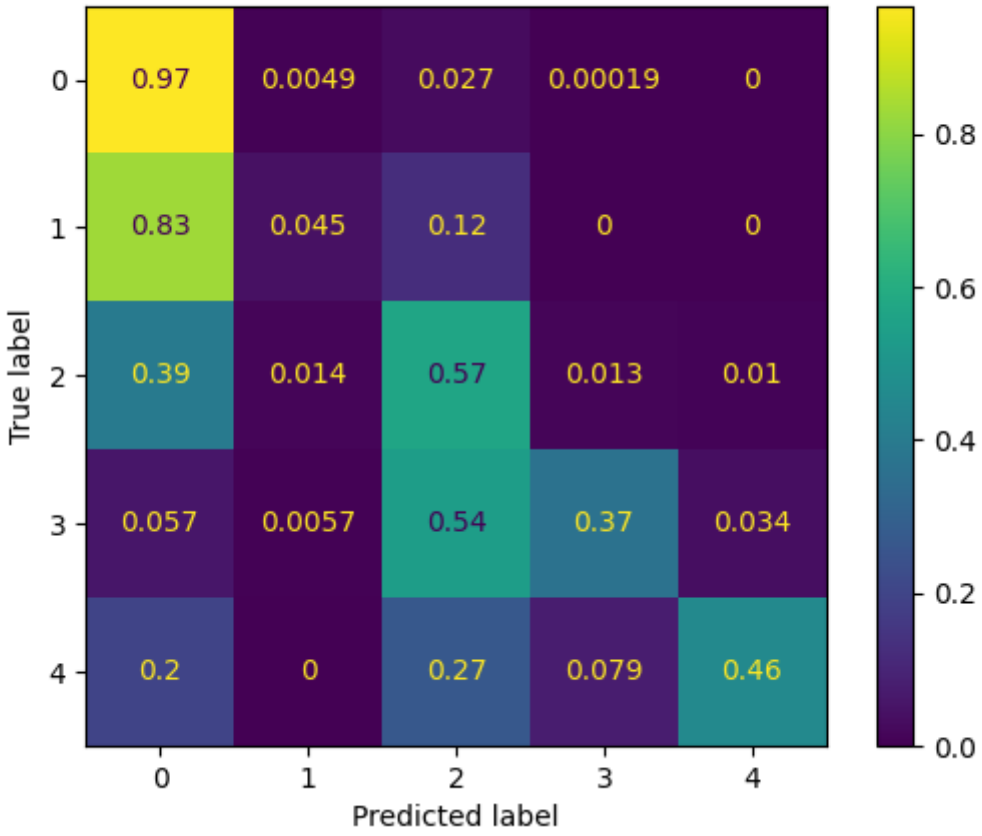


- ResNet50 (w/o pretraining) Confusion Matrix

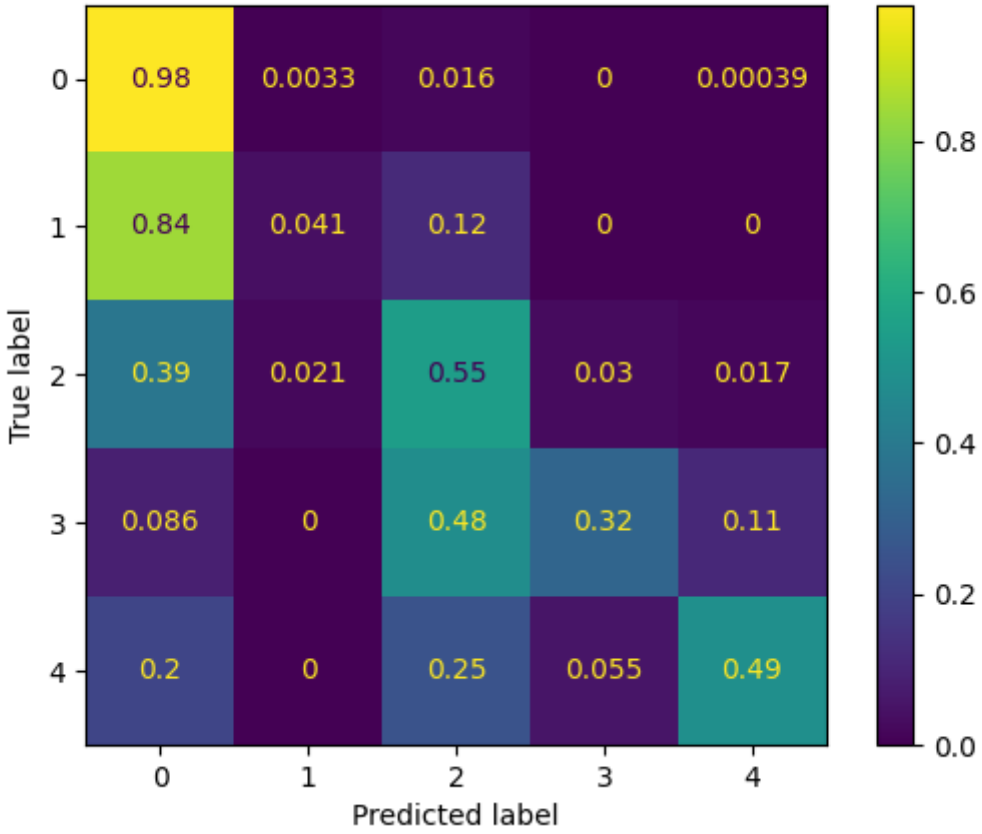


藉由上面兩張圖我們可以清楚看到，沒有 pretraining 的 model，predicted 出來的 label 幾乎都是 0，而在這樣的情況下，Test accuracy 都可以達到 **73%** 左右，所以我就去看了一下 training 跟 test 的 ground-truth label，發現其中 label 是 0 的比例在 training 跟 test data 也都大約是 **73%** 左右，是非常 imbalance 的 data，只要全部猜 0 的話，準確率都可以達到 73%，也難怪沒有 pretrained 的 model 直接去對 training data 做訓練會得到像這樣 prediction 幾乎都是 0 的情況。

- ResNet18 (with pretraining) Confusion Matrix



- ResNet50 (with pretraining) Confusion Matrix



藉由上面兩張圖，可以觀察到有 pretraining 的 model，似乎就稍微改善了這樣的狀況。

不過在 True label 為 1 的情況下，model 還是幾乎都 predict 為 0，並且 predict 2 的數量還比 1 多，同樣在 True label 為 3，也是 predict 為 2 的數量較多，因此我又去看了一下 train 跟 test 的 ground-truth label，果然 label 為 2 的數量又是遠遠多於除了 0 以外的其他 label，加上 1 跟 3 的照片可能分別也會跟 0、2 和 2、4 較為相似，才會有這樣的結果。

3. Experimental results

A. The highest testing accuracy

- Screenshot
  使用 `ResNet50` ，batch size 設為 `8` ，learning rate 設為 `1e-3` ，momentum 設為 `0.9` ，並且 weight decay 設為 `5e-4` 。
  前 5 個 epoch 先做 feature extraction，後 15 個 epoch 做 fine-tuning。
  如下圖，對 Data 做 transform

```
Compose(
    RandomHorizontalFlip(p=0.5)
    RandomVerticalFlip(p=0.5)
    ToTensor()
    Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
)
```

得到最高的 Test accuracy 為 **82.36%**

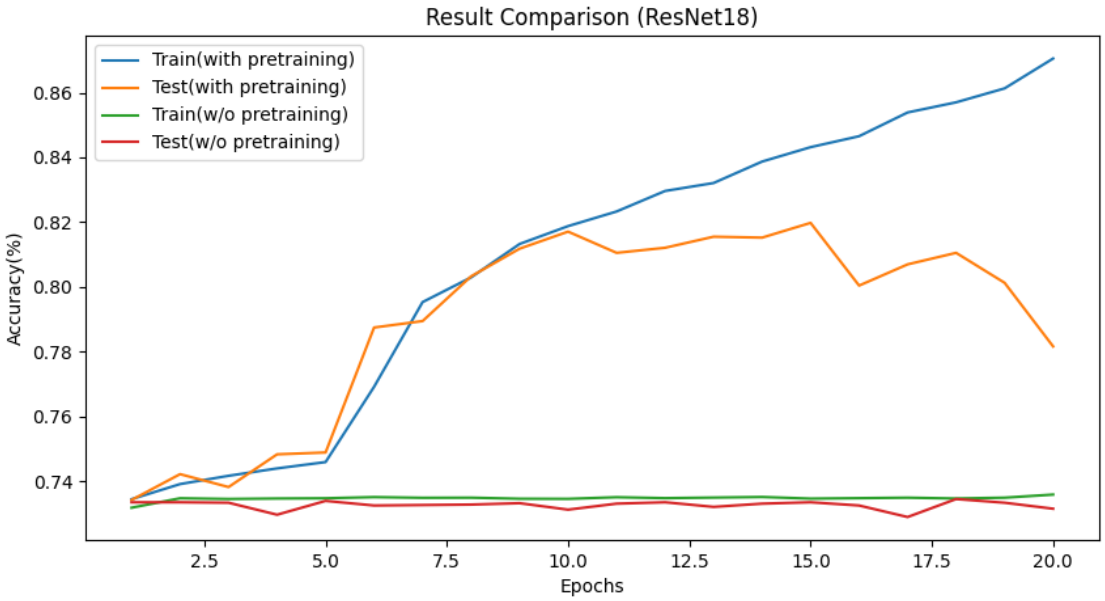
- Anything you want to present
  在做 data transform 時，我有先試過使用 pretrained model 所提供的 transforms (ResNet18_Weights.IMAGENET1K_V1.transforms)，如下圖

```
ImageClassification(
    crop_size=[224]
    resize_size=[256]
    mean=[0.485, 0.456, 0.406]
    std=[0.229, 0.224, 0.225]
    interpolation=InterpolationMode.BILINEAR
)
```

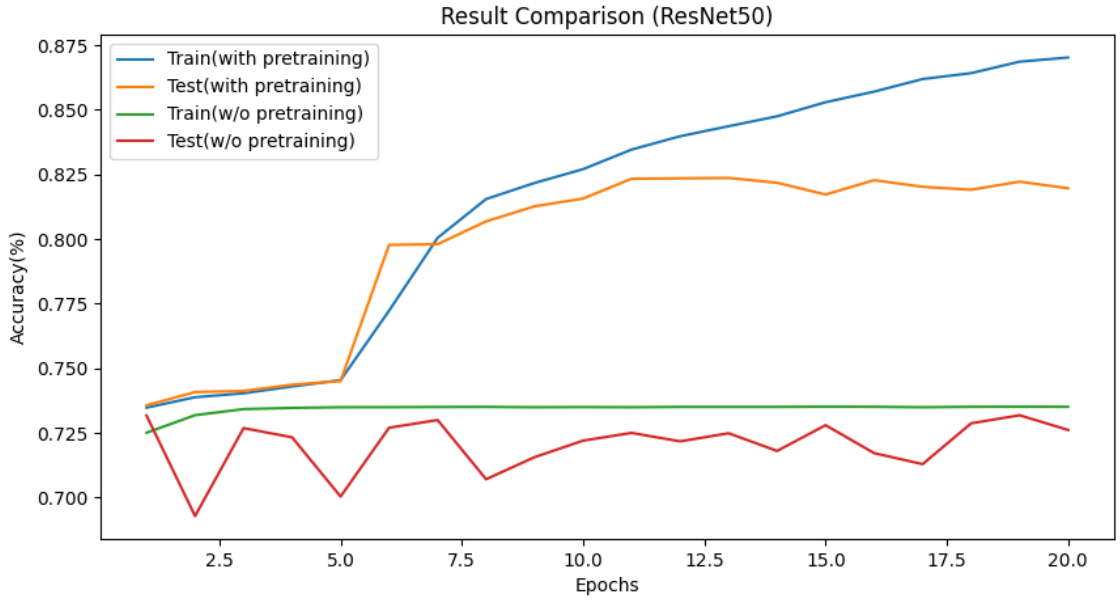
  發現使用這樣的 data transform 會造成 pretrained model overfitting，或許跟沒有 data augmentation 有關，因此我將其加入 torchvision.transforms.RandomHorizontalFlip 以及 torchvision.transforms.RandomVerticalFlip 雖然改善了overfitting 的問題，但是 testing accuracy 只有大概 **79%** 左右，後來發現原來是因為有先做 crop 的關係，可能因此造成 crop 到的部分是沒有明顯特徵的，並且在 ResNet 的 fully-connected layer 前有一個 torch.nn.AdaptiveAvgPool2d 可以確保 output size 都是一致的，因此我就將 crop 以及 resize 去掉，結果testing accuracy 就可以達到 **82%** 左右。

B. Comparison figures

- Plotting the comparison figures
  - ResNet18



  - ResNet50



4. Discussion

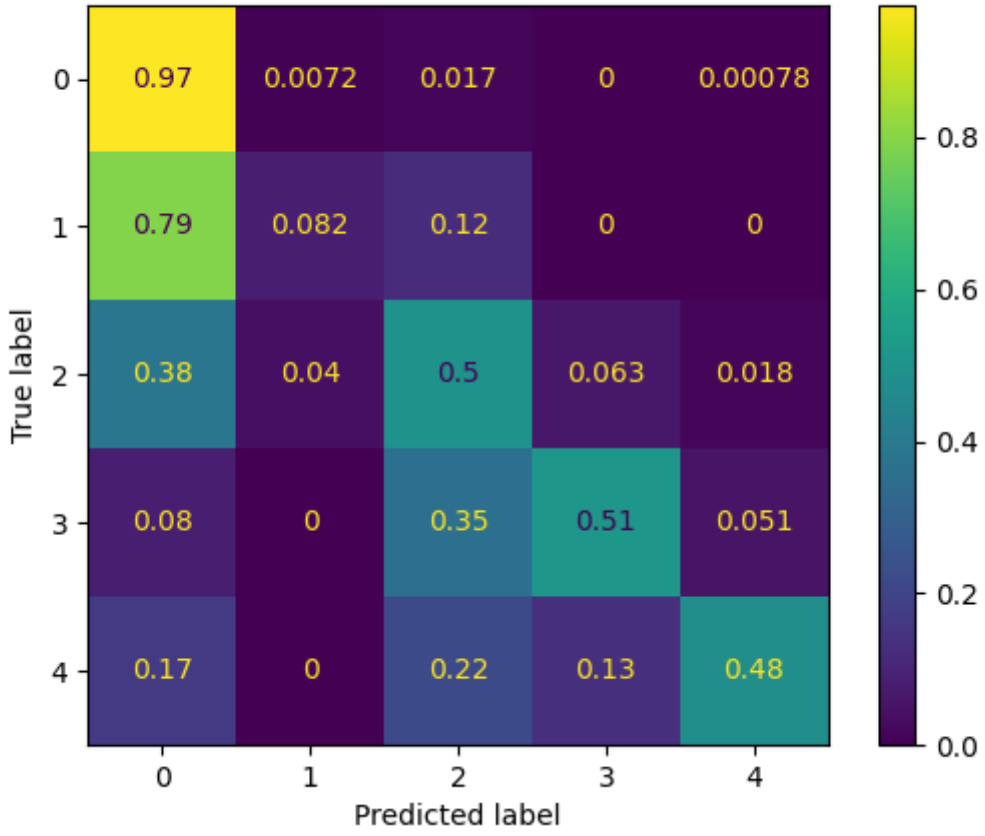
A. Anything you want to share

在實驗過程中有發現到 imbalanced data 對於 model 所帶來的影響，我也去了解該如何改善它，發現到可以使用以下幾種方法：

1. Oversampling
   把 minority class 補到跟 majority class 數量一樣多，可以透過 data aumentation 的方式，但有可能會造成 overfitting。
2. Undersampling
   把 majority class 砍到跟 minority class 數量一樣多，透過隨機刪除的方式，但有可能刪到一些重要的 feature，造成 underfitting。
3. Class-weighted loss
   對 minority class 的 loss 乘上一個較大的 weight，讓它對於 loss 的影響較多，因為我們的目標是要最小化 loss ，所以就會特別去學習 minority class，常見決定 class weight 的方式是
   - $$weight_{類別} = \frac{1}{類別數量}$$
   - $$weight_{類別} = 1 - \frac{該類別數量}{總資料量}$$
   - $$weight_{類別} = \frac{最多類別資料量}{該類別資料量}$$

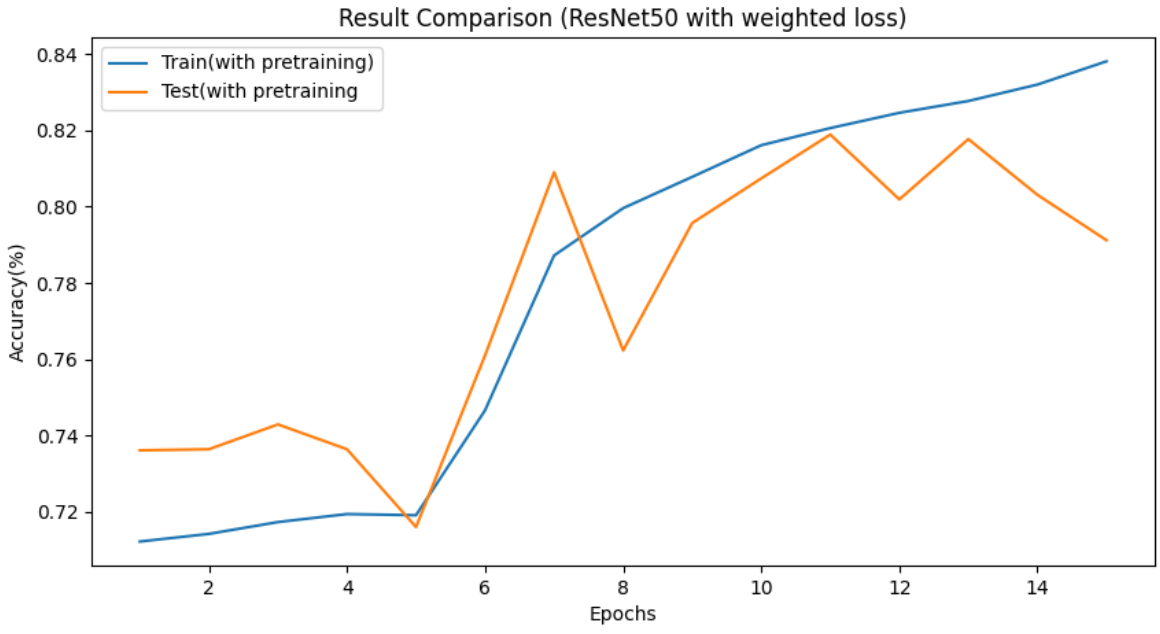
這邊我以上述第二種方式來決定 weight ， `weight = [0.2649, 0.9304, 0.8502, 0.9752, 0.9793]` ，並以此 weighted loss 再訓練一次 ResNet50，最高的 test accuracy 可以達到 **81.89%** ，並得到如以下結果。

- ResNet50 (with pretraining & weighted loss) Confusion Matrix



可以看到相較於沒有 weighted loss 時，Confusion Matrix 有改善一些，prediction 為 0 跟 2 的比例減少，而 1 跟 3 則增加，特別在 True label 為 3 的情況下，Predict label 為 3 的比例也大幅增加，不過在 True label 為 1 的情況下，Predict 為 1 的比例雖然有增加，但是只有增加了一點點。

- ResNet50 (with weighted loss) Comparison Figure



由上圖可以看到 Testt accuracy 是上下劇烈波動的，我想可能是因為當 model 去特別學習某些類

別資料時，會牽一髮而動全身，將原先大多數是 0 的 label，也預測成了該類別，而有這樣的結果。