Lab3: Diabetic Retinopathy Detection

Author: 311553007 應耀德

1. Introduction

Lab Objective:

In this lab, you will need to analysis diabetic retinopathy (糖尿病所引發視網膜病變) in the following three steps. First, you need to write your own custom DataLoader through PyTorch framework. Second, you need to classify diabetic retinopathy grading via the ResNet architecture. Finally, you have to calculate the confusion matrix to evaluate the performance.

Requirements:

1. Implement the ResNet18, ResNet50 architecture and load parameters from a pretrained model.
2. Compare and visualize the accuracy trend between the pretrained model and without pretraining in same architectures, you need to plot each epoch accuracy (not loss) during training phase and testing phase.
3. Implement your own custom DataLoader.
4. Calculate the confusion matrix and plotting.

Data - Diabetic Retinopathy Detection (Kaggle):

Diabetic retinopathy is the leading cause of blindness in the working-age population of the developed world. It is estimated to affect over 93 million people. This dataset provided with a large set of high-resolution retina images taken under a variety of imaging conditions.

Image format: .jpeg

Image size: 512 x 512

Number of images: 25,124 images

Reference: https://www.kaggle.com/c/diabetic-retinopathy-detection#description

- Train data: train_img.csv, train_label.csv, 28,009 images
- Test data: test_img.csv, test_label.csv, 7,025 images

Loss function:

Cross-Entropy Loss

$$l(\theta) = -\sum_{i=1}^{n} \left(y_i \log \hat{y}_{\theta,i} + (1 - y_i) \log \left(1 - \hat{y}_{\theta,i}\right) \right)$$

Where

- n: $number\ of\ data\ points$
- i: $index\ of\ data\ points$
- θ: $weights$
- $\hat{y}$: $predict\ result$
- y: $ground\ truth$

Optimizer:

Stochastic Gradient Descent (SGD)

$$W \leftarrow W - \eta \frac{\partial L}{\partial W}$$

Where

- W: $weights$
- L: $loss\ function$
- η: $learning\ rate$

2. Experiment setups

Training settings:

- Batch Size: 128(ResNet18), 64(ResNet50)
- Epoch Size: 10(ResNet18), 5(ResNet50)
- Optimizer: SGD, learning rate = 1e-3, momentum = 0.9, weight decay = 5e-4
- Loss function: Cross-Entropy Loss

The details of your model (ResNet18 and ResNet50):

```python
class ResNet18(nn.Module):
    model: nn.Module
    def __init__(self, out_features, pretrained = True):
        super(ResNet18, self).__init__()

        self.model = torch_models.resnet18(weights = torch_models.ResNet18_Weights.IMAGENET1K_V1 if pretrained else None)
        # freeze pretrained layers
        if pretrained:
            for param in self.model.parameters():
                param.requires_grad = False

        self.model.fc = nn.Linear(self.model.fc.in_features, out_features)

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

class ResNet50(nn.Module):
    model: nn.Module
    def __init__(self, out_features, pretrained = True):
        super(ResNet50, self).__init__()

        self.model = torch_models.resnet50(weights = torch_models.ResNet50_Weights.IMAGENET1K_V1 if pretrained else None)
        # freeze pretrained layers
        if pretrained:
            for param in self.model.parameters():
                param.requires_grad = False

        self.model.fc = nn.Linear(self.model.fc.in_features, out_features)

    def forward(self, x):
        return self.model(x)
```

Figure 2. Implementation of ResNet18 and ResNet50

The details of your dataloader:

```python
class RetinopathyDataset(Dataset):
    _image_root_dir: str
    _image_names: np.ndarray
    _labels: np.ndarray
    _transforms: transforms.Compose

    def __init__(
        self, image_root_dir: str, image_csv: str, label_csv: str, transformation: bool = True, training: bool = True
    ):
        super(RetinopathyDataset, self).__init__()
        if not os.path.exists(image_csv):
            raise FileNotFoundError(f"The csv file {image_csv} does not exist.")
        if not os.path.exists(label_csv):
            raise FileNotFoundError(f"The csv file {label_csv} does not exist.")
        if not os.path.exists(image_root_dir):
            raise FileNotFoundError(f"The folder {image_root_dir} does not exist.")

        self._image_root_dir = image_root_dir
        self._image_names = np.squeeze(pd.read_csv(image_csv).values)
        self._labels = np.squeeze(pd.read_csv(label_csv).values)
        print("> Found %d images ... " % (len(self._image_names)))

        transform_operations = [
            transforms.Resize((224, 224))
        ]
        if transformation:
            if training:
                transform_operations += [
                    transforms.RandomInvert(p=0.5),
                ]

            transform_operations += [
                transforms.ToTensor(),
                transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
            ]
        else:
            transform_operations = [transforms.ToTensor()]
        self._transforms = transforms.Compose(transform_operations)

    def __len__(self):
        return self._image_names.shape[0]
```

```python
    def __getitem__(self, index: int):
        """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 resizing, random flipping,
                rotation, cropping, normalization etc. But at the beginning, I suggest you follow the hints.

                In the testing phase, if you have a normalization process during the training phase, you only need
                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 = self._image_root_dir + "/" + self._image_names[index] + ".jpeg"
        image = self._transforms(Image.open(path))
        return image, self._labels[index]
```

Figure 3. Implementation of dataset

```python
BATCH_SIZE: int = 128
train_dataloader = DataLoader(train_dataset, batch_size = BATCH_SIZE, shuffle = True)
test_dataloader = DataLoader(test_dataset, batch_size = BATCH_SIZE, shuffle = False)
```

Describing your evaluation through the confusion matrix:

x-axis 為 predicted label (代表 A)，y-axis 為 ground truth (代表 B)。斜對角為預測正確之比例。

從 row 來解讀，例如:屬於 B0 的 data 分布，普遍預測正確，部分資料被誤判為 A2，少部分被誤判為 A3, A4。

從 column 來解讀，例如:被預測為 A0 的 data，大部分預測正確，部分資料實際上是屬於其他 label。

合併解讀，例如:

從 A1 與 B1 可知，此 model 將 B1 皆誤判為其他 label，而預測為 A0 的最多，代表此 model 在預測 label 1 的 data 上沒有學習到任何關鍵辨別特徵。

也從 A0 可知，實際上還是有很多 data 被誤判為 A0，代表並沒有完全學習到辨別各 label 的特徵。

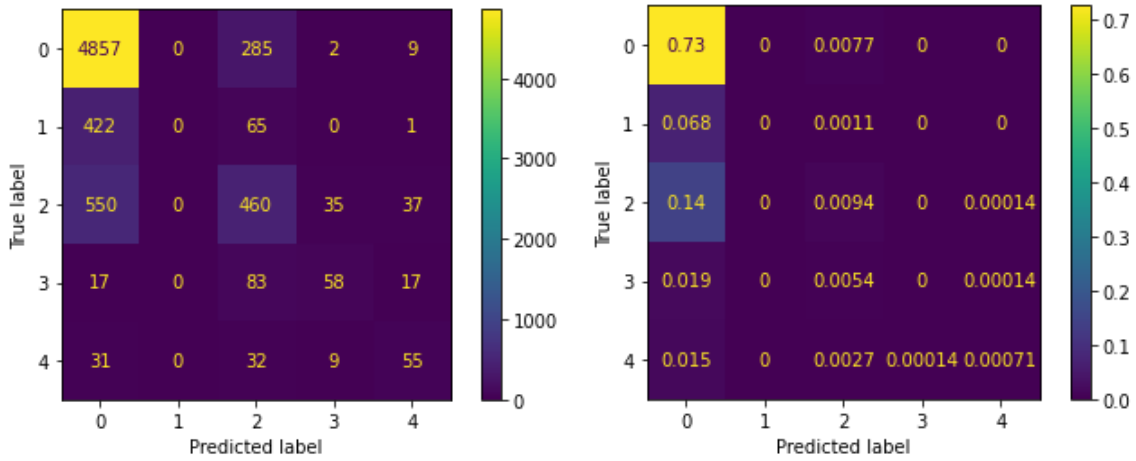
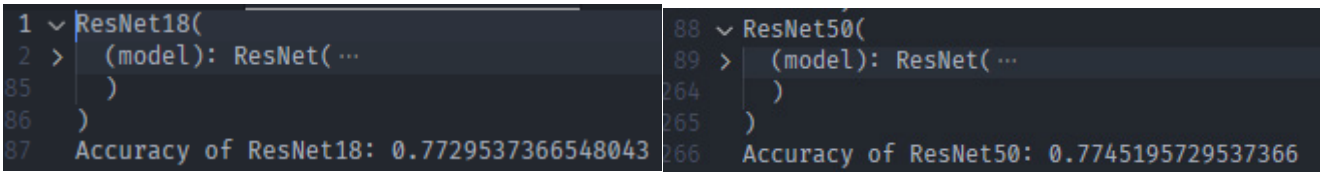


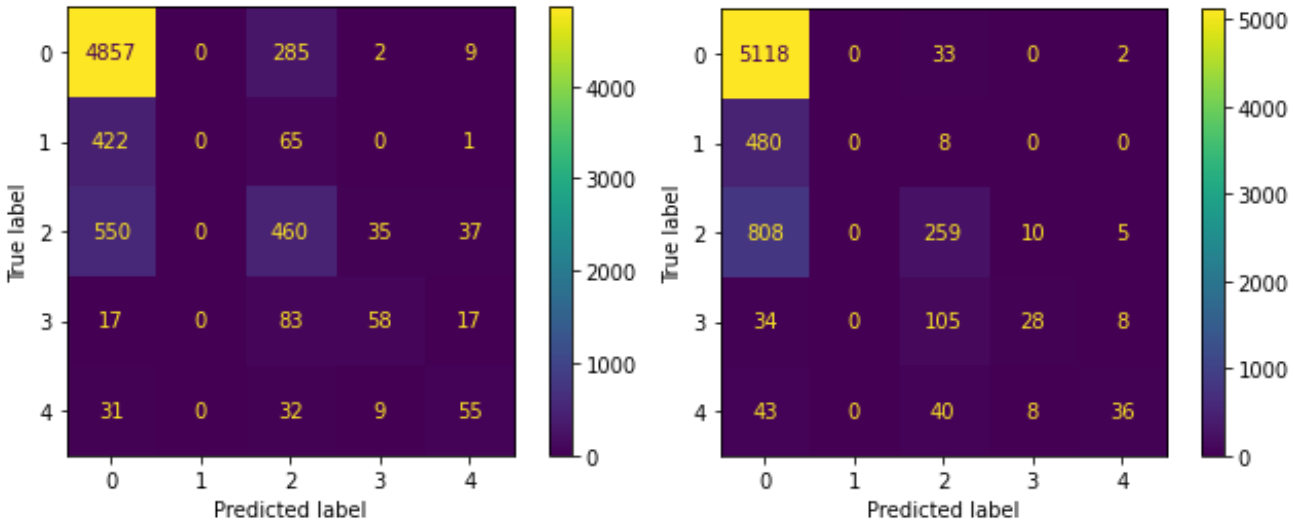
Figure 4. An example of confusion matrix (w/o normalization and with normalization)

3. Experimental results

The highest testing accuracy:

Network	pretrained	from scratch
ResNet18	**77.29%**	73.36%
ResNet50	**77.45%**	73.35%





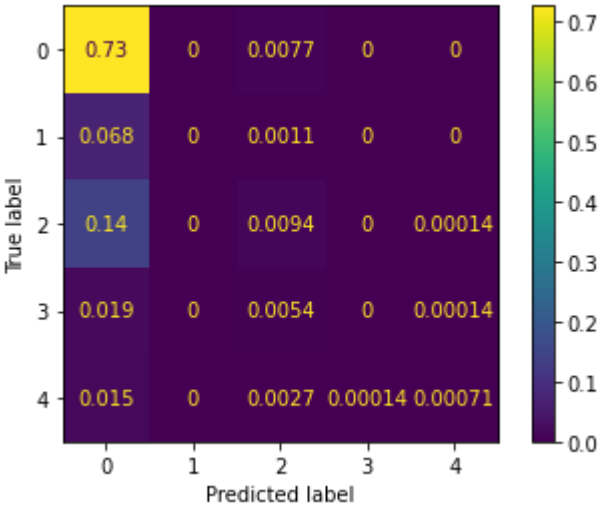


Figure 5. Confusion matrix of ResNet18(pretrained)

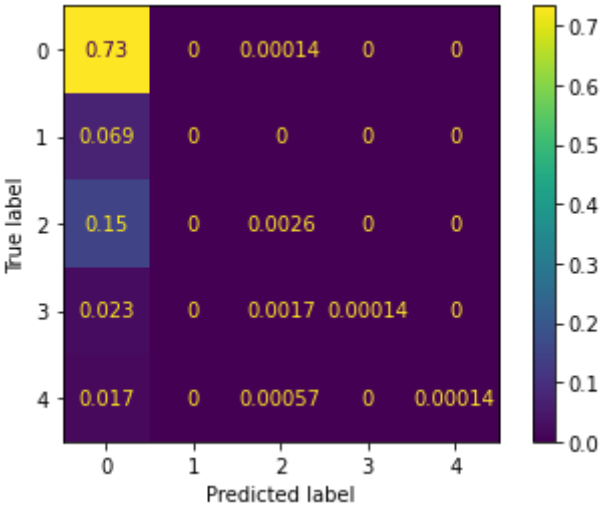
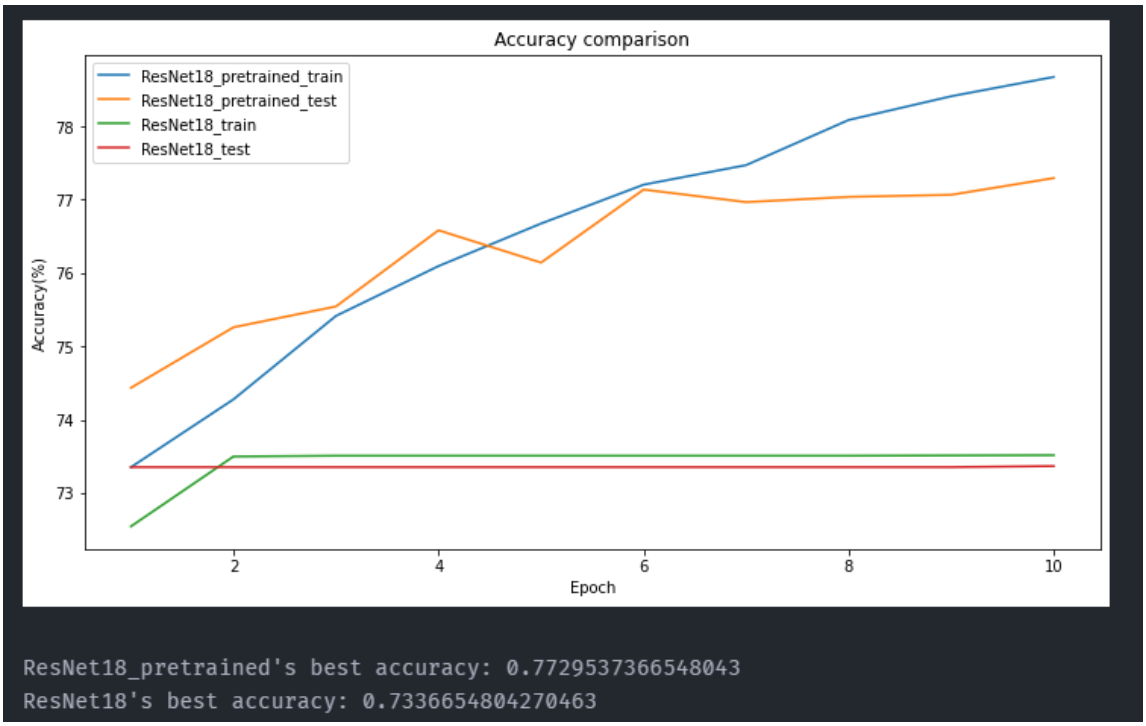
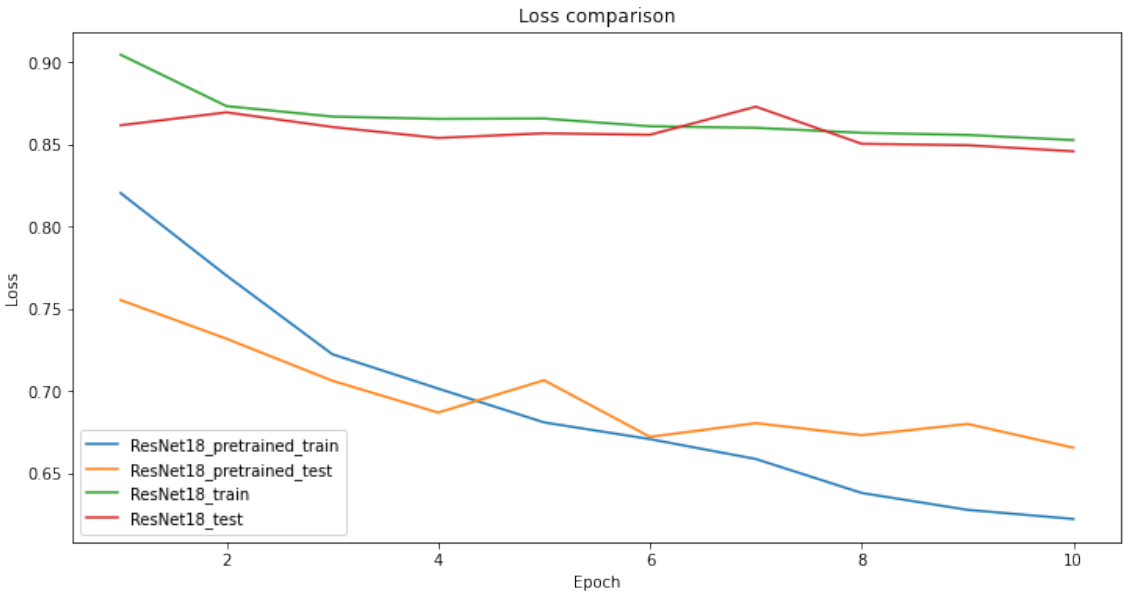


Figure 6. Confusion matrix of ResNet50(pretrained)

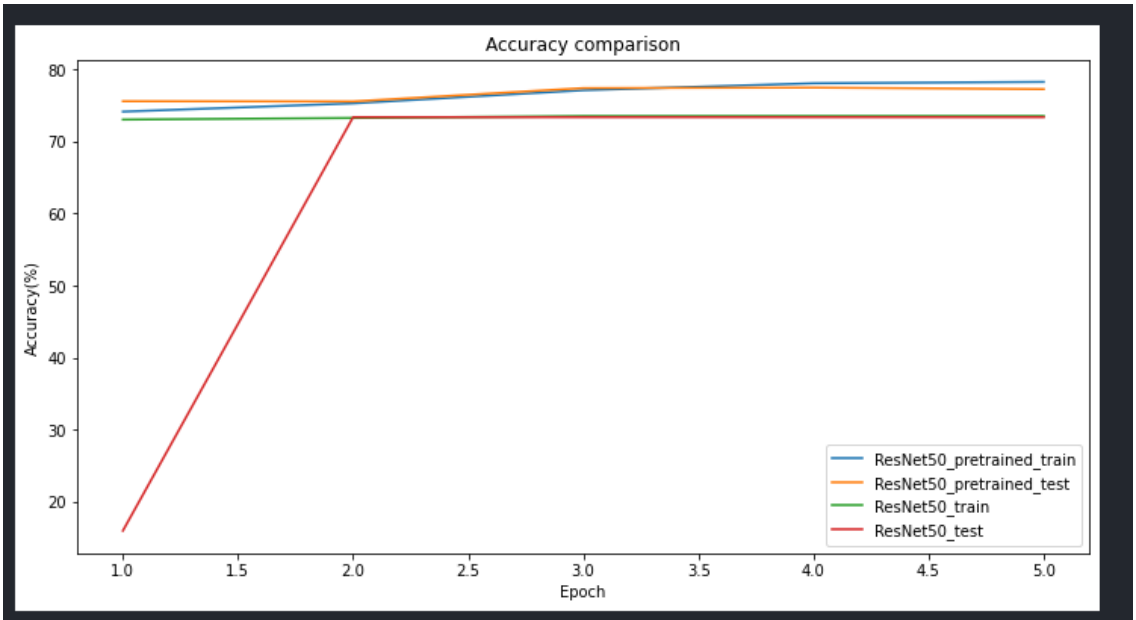
Comparison figures:

- ResNet18

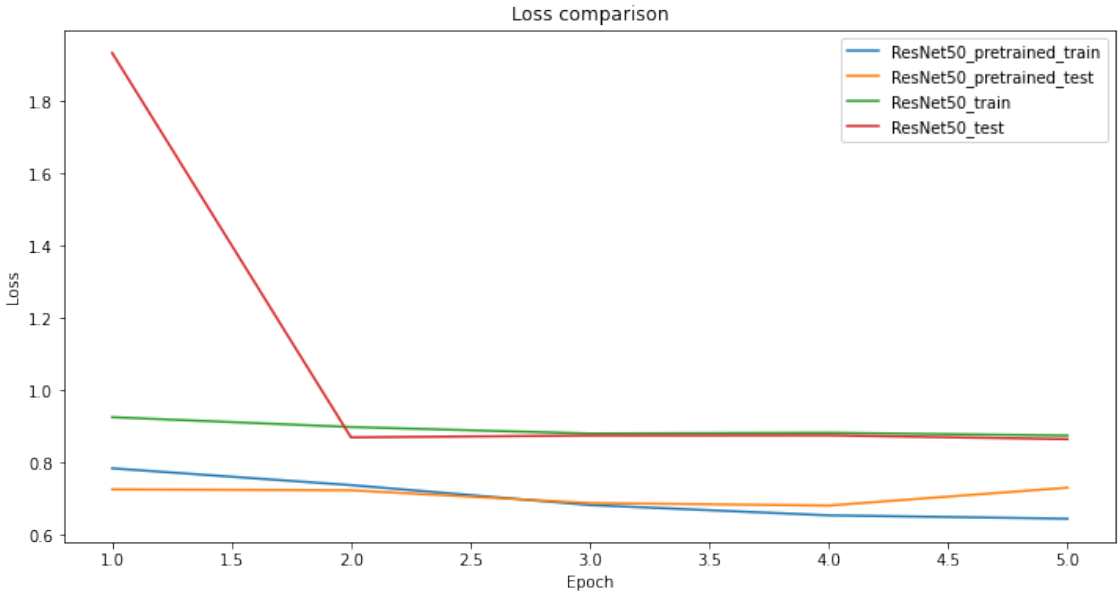




- ResNet50




```
ResNet50_pretrained's best accuracy: 0.7745195729537366
ResNet50's best accuracy: 0.7335231316725979
```



4. Discussion

Why is the highest accuracy always around 77%?

Possible problems

- From the confusion matrix result and the data distribution, the dataset is like long-tailed dataset. The data is imbalanced. From the left to the right entry, it is in the order of labels.

```
training data counts: [20655  1955  4210   698   581]
testing data counts: [5153  488 1082  175  127]
```

- Use OpenCV to analyze images (training data). About 15% images are too dark (brightness under 25%). Need more data augmentations to learn the contours of image.

```
Number of dark images in training data: 0.15829744830776896
Number of dark images in testing data: 0.16298932384341638
```

What's the performance between models with transforms and baseline model?

實驗配置皆為前面設定，model 使用 ResNet18 with pretrained weights，除了 transforms 有更動，baseline model 的配置為 Resize 與 ToTensor。

Finetuning 的感想:當時在進行實驗時，並沒有與 baseline model 比較、accuracy 提升程度、confusion matrix 預測情況，花費大部分的時間在依照 data 特性組合多個 transforms 與 training 上面。雖然有分析 data distribution 與圖片亮度占比，就以結果來說，許多 transforms 配置並沒有改善 accuracy，反而還比 baseline model 還差。下圖為測試過的組合，後面括號為 best accuracy

Without normalization

1. random choice (flips), invert (77.4)
2. random choice (flips, rotation), invert (77.6)
3. random choice (invert, equal), random choice (flips) (77.8)
4. invert, random choice (flips, rotation) (77.4) x
5. resize, random choice (flips, rotation), invert (77.5)
6. resize, invert (77.4)
7. resize, invert, brightness 0.1-1.0 (76.9)
8. resize, random choice (invert, equal) (77.7)
9. resize, random choice (invert, equal), RandomAutocontras (77.4)
10. resize, random choice (invert, RandomOrder(equal, RandomAutocontrast)) (77.5)

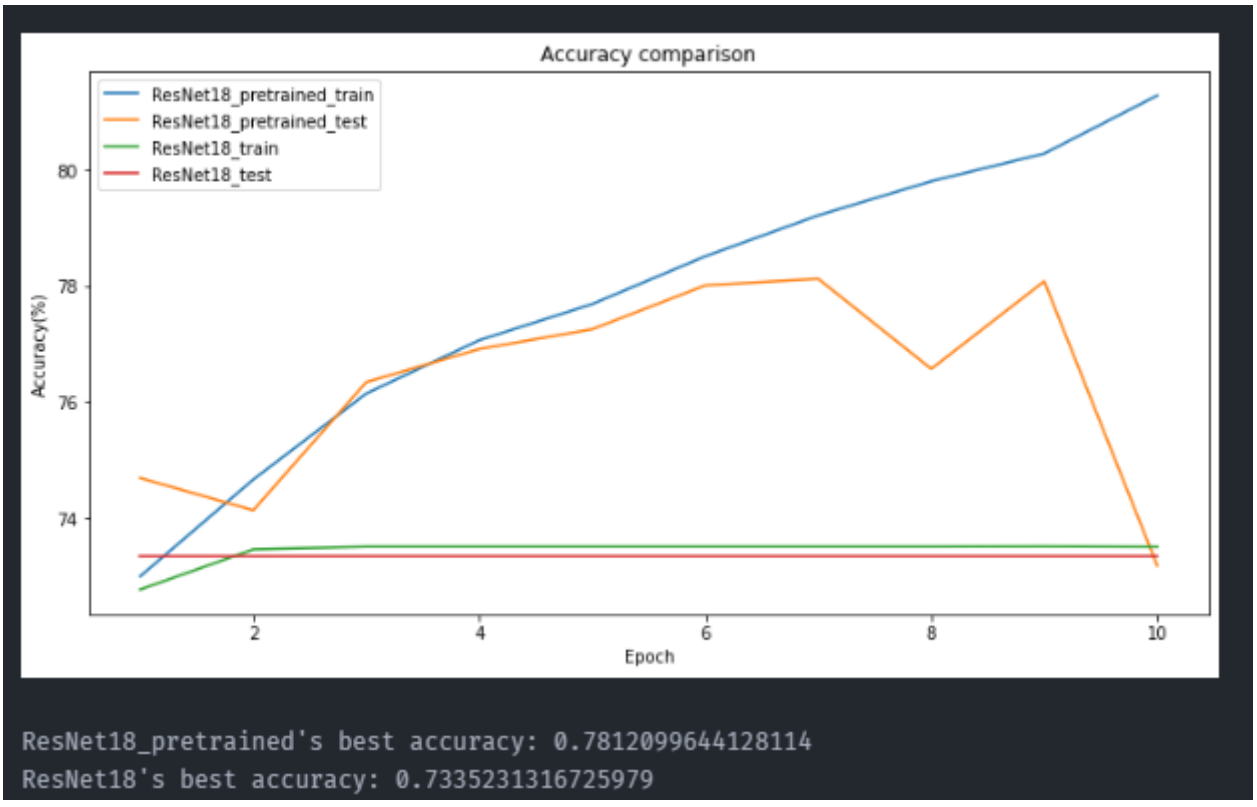


Figure 7. Accuracy comparison of baseline ResNet18(pretrained)

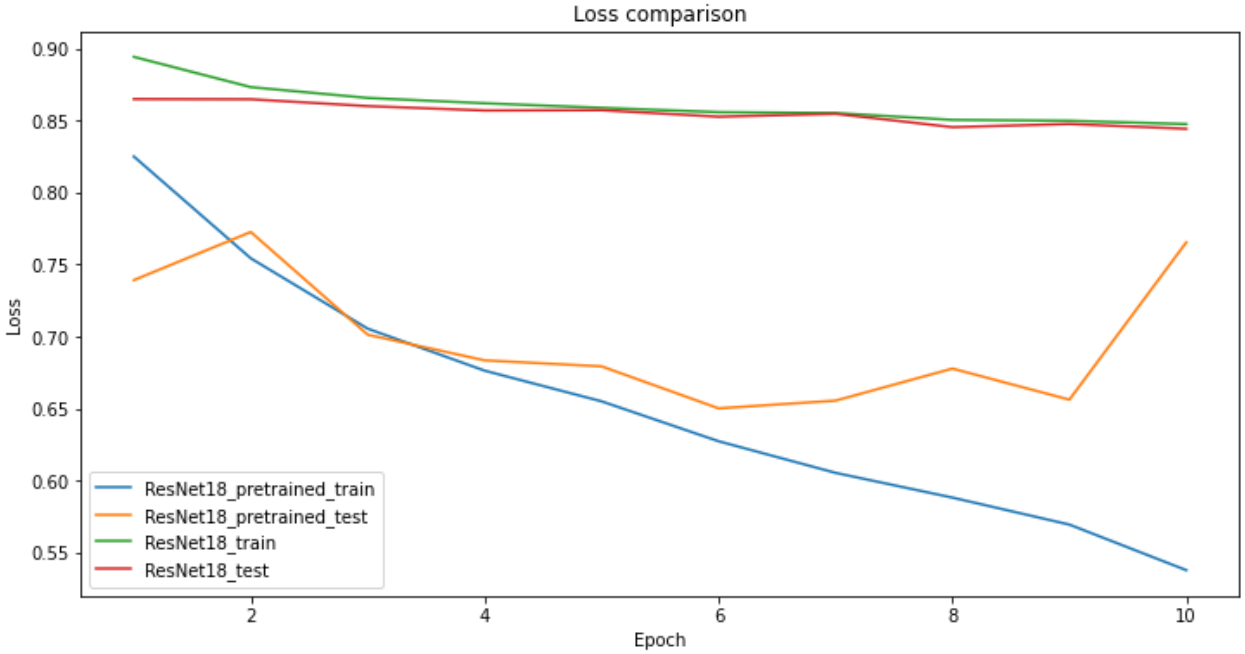


Figure 8. Loss comparison of baseline ResNet18(pretrained)

The data is imbalanced. How can I solve it?

Based on the experiment setups

- Test 1: Use cross-entropy loss with class weights

  此方法在 loss function 中加入各個 class 權重，預設每項 class 權重為 1。

  資料量越少的 class，該 class 權重越大 (loss 占比較多，gradient 較多)。

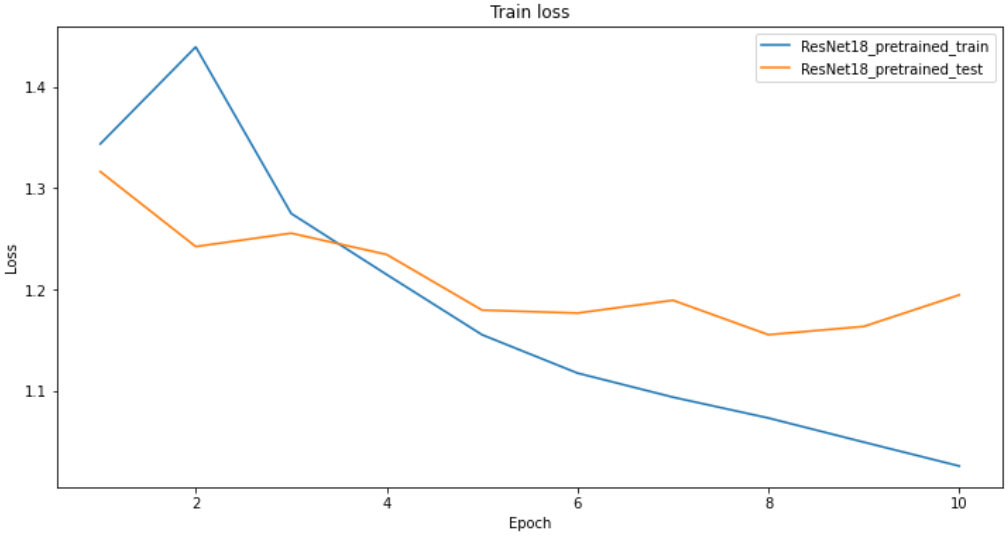
  使得模型也能較關注資料量較少的 class 上，避免都偏向學習資料量較多的 class。

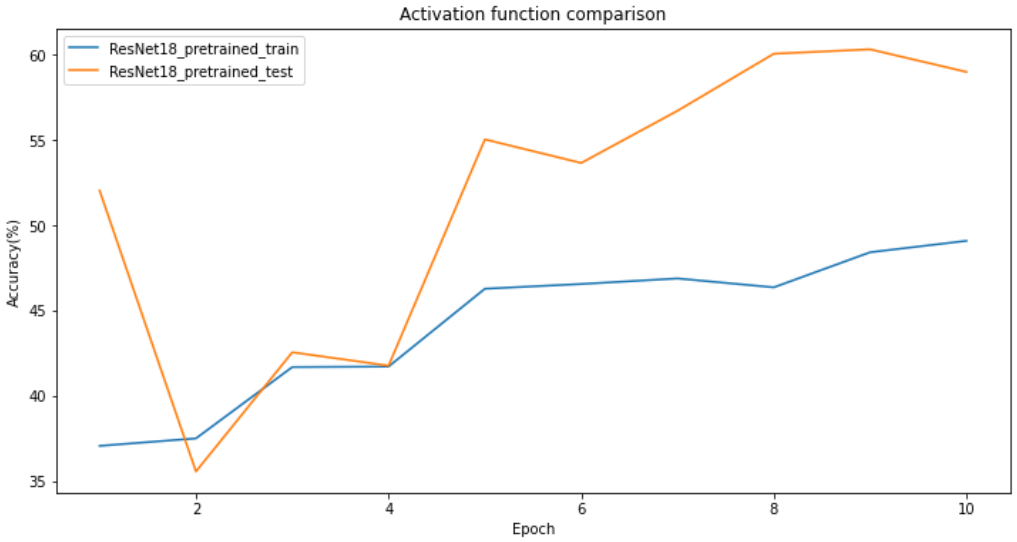
$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -\sum_{c=1}^{C} w_c \log \frac{\exp(x_{n,c})}{\sum_{i=1}^{C} \exp(x_{n,i})} y_{n,c}$$

Figure 3. The formula of cross-entropy loss function, where x: *input (output of the model)*, y: *ground truth*, ω: *class weight (constant)*, C: *number of classes*

a. ResNet18 with pretrained weights

   從 comparison loss 圖中可知(請忽略圖中標題)，training loss 持續收斂，但礙於訓練時間過長，只能先跑前面的訓練情況。





- Test 2:直接對 Data 做 augmentation，upsampling 較少的 label 類別資料
  WIP