Lab 4-2 - Diabetic Retinopathy Detection

Department of Computer Science, NCTU

TA 賴佑家

Important Rules

Important Date :

- Report Submission Deadline: 4/26 (Tue) 11:59 a.m.(中午11點59分)
- Demo date: 4/26 (Tue)

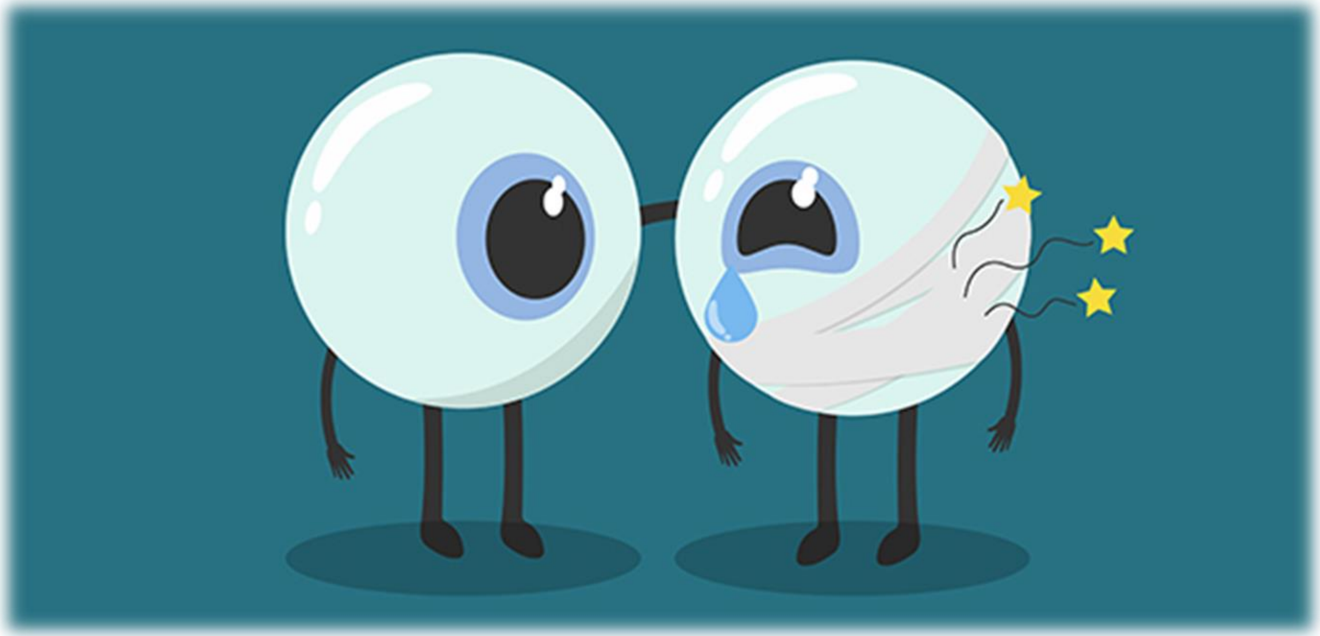
Turn in :

- Experiment Report (.pdf)
- Source code (.py)

Notice: zip all files in one file and name it like「DLP_LAB4-2_your studentID_name.zip」, ex: 「DLP_LAB4-2_309553005_賴佑家.zip」

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.
  Step 1. You need to write your own custom DataLoader through PyTorch framework.
  Step 2. You need to classify diabetic retinopathy grading via the ResNet [1].
  Step 3. You have to calculate the confusion matrix to evaluate the performance.



Source: http://www.commonhealth.com.tw/article/article.action?nid=66150

Requirements

- Implement the **ResNet18、ResNet50** architecture and load parameters from a pretrained model

- 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.

- Implement your own custom **DataLoader**

- Calculate the **confusion matrix** and plotting

Dataset - Diabetic Retinopathy Detection (kaggle)

- Diabetic retinopathy is the leading cause of blindness in the working-age population of the developed world.
- This dataset provided with a large set of high-resolution retina images taken under a variety of imaging conditions. **Format: .jpeg**

Class

0 - No DR

1 - Mild

2 - Moderate

3 - Severe

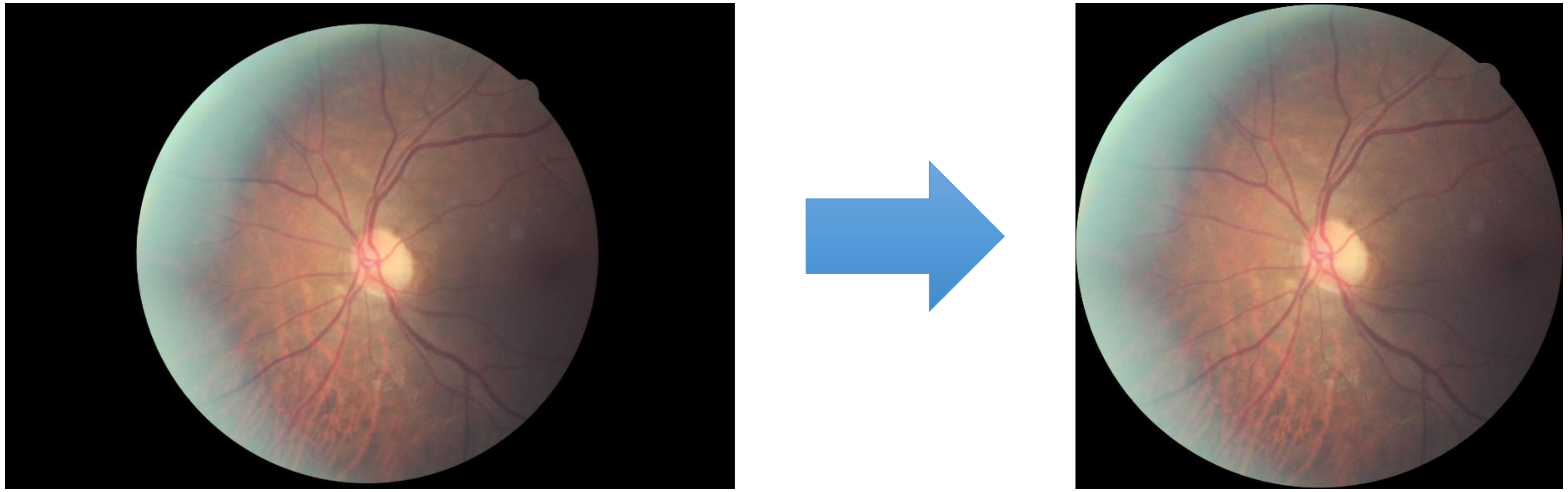
4 - Proliferative DR

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

Prepare Data

- **28,099** images for training
- **7025** for testing
- **Download link:** https://drive.google.com/open?id=1RTmrk7Qu9IBjQYLczaYKOvXaHWBS0o72
- The image resolution is 512x512 and has been preprocessed.

- **Input: [B, 3, 512, 512]** **Output: [B, 5]** **Ground truth: [B]**



4752 x 3168 **512 x 512**

Dataloader

- Implement your own custom DataLoader
- Below is the skeleton that you have to fill to have a custom dataset, refer to “dataloader.py”

```python
class RetinopathyLoader(data.Dataset):
    def __init__(self, root, mode):

    def __len__(self):
        """'return the size of dataset"""

    def __getitem__(self, index):
        """something you should implement here"""
```

Prepare Data

```python
def __init__(self, root, mode):
    """
    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.root = root
    self.img_name, self.label = getData(mode)
    self.mode = mode
    print("> Found %d images..." % (len(self.img_name)))

def __len__(self):
    """'return the size of dataset"""
    return len(self.img_name)
```

Prepare Data

- test_img.csv
- test_label.csv
- train_img.csv
- train_label.csv

```python
def getData(mode):
    if mode == 'train':
        img = pd.read_csv('train_img.csv')
        label = pd.read_csv('train_label.csv')
        return np.squeeze(img.values), np.squeeze(label.values)
    else:
        img = pd.read_csv('test_img.csv')
        label = pd.read_csv('test_label.csv')
        return np.squeeze(img.values), np.squeeze(label.values)
```

3798_left	0
9317_right	0
1991_right	0
2086_left	0
34952_left	0
18072_right	0
9958_left	0
32121_left	0
29612_left	0
21978_left	1
26746_left	0
21469_right	2
40812_right	0
22575_right	2

Image Format: .jpeg

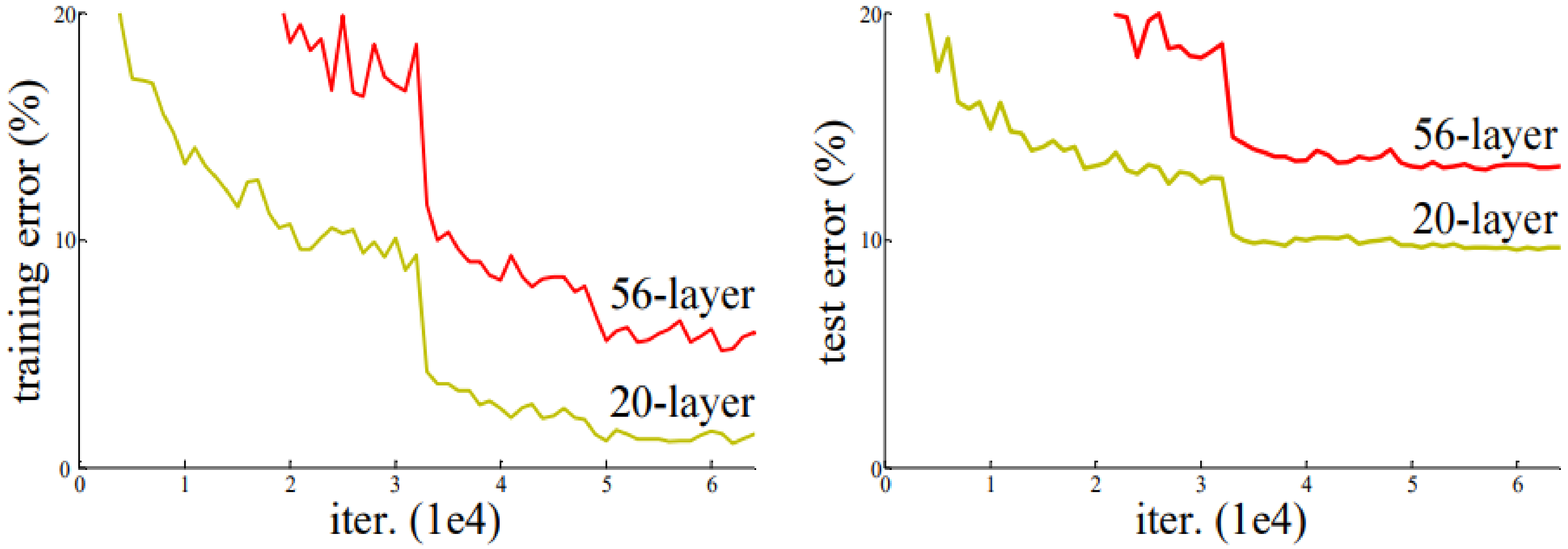
Please do not sort !!!

Prepare Data

```
def __getitem__(self, index):
    """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
    """

    return img, label
```

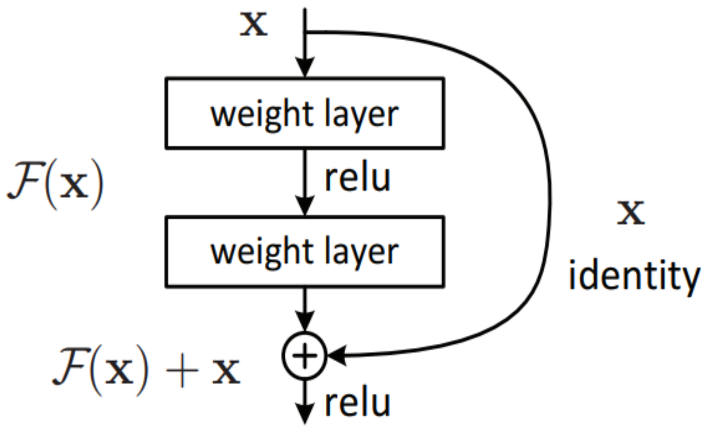
ResNet

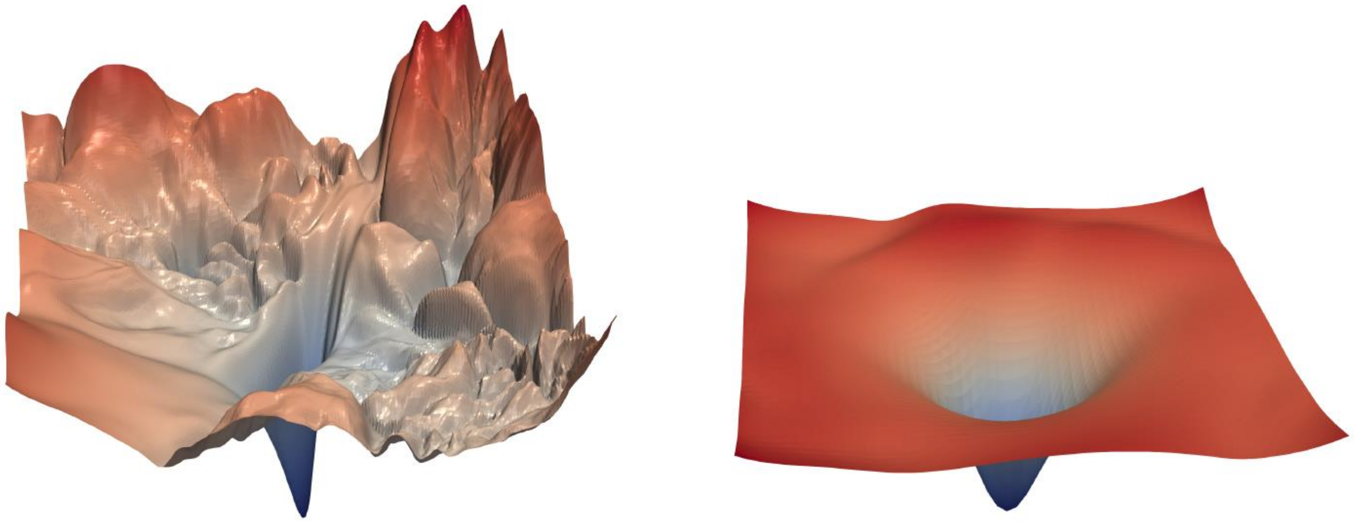
- ResNet (Residual Network) is the Winner of ILSVRC 2015 in image classification, detection, and localization, as well as Winner of MS COCO 2015 detection, and segmentation



ResNet

- To solve the problem of vanishing/exploding gradients, a skip / shortcut connection is added to add the input x to the output after few weight layers as below





(a) without skip connections

(b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

Source: Li, Hao, et al. "Visualizing the loss landscape of neural nets." Advances in Neural Information Processing Systems. 2018.

ResNet

- ResNet can avoid vanishing gradient problem

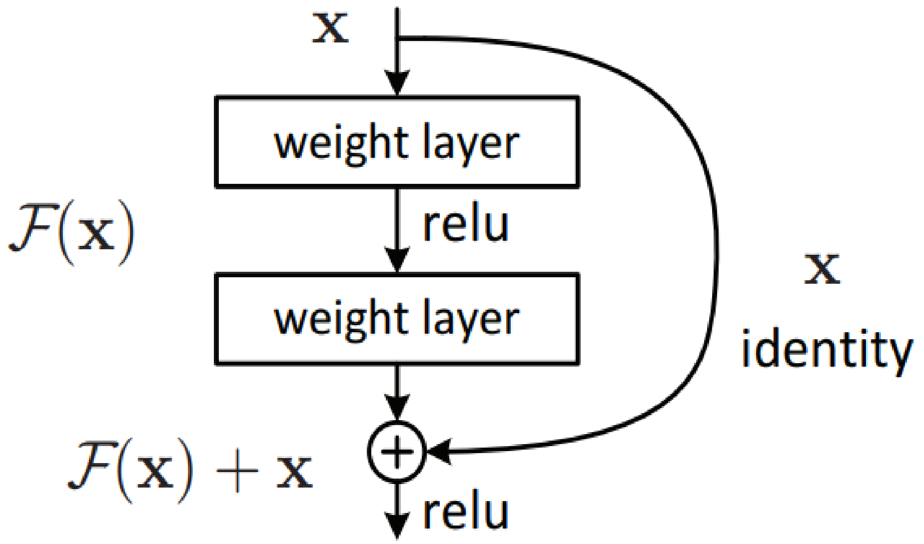
$$x \rightarrow w_1 \underset{y_1}{\rightarrow} w_2 \underset{y_2}{\rightarrow} w_3 \underset{y_3}{\rightarrow} w_4 \underset{y_4}{\rightarrow} Loss$$

$$\frac{\partial Loss}{\partial w_1} = \frac{\partial Loss}{\partial y_4}\frac{\partial y_4}{\partial z_4}\frac{\partial z_4}{\partial y_3}\frac{\partial y_3}{\partial z_3}\frac{\partial z_3}{\partial y_2}\frac{\partial y_2}{\partial z_2}\frac{\partial z_2}{\partial y_1}\frac{\partial y_1}{\partial z_1}\frac{\partial z_1}{\partial w_1}$$

$$= \frac{\partial Loss}{\partial y_4}\sigma'(z_4)w_4\sigma'(z_3)w_3\sigma'(z_2)w_2\sigma'(z_1)x_1$$

ResNet

- ResNet can avoid vanishing gradient problem



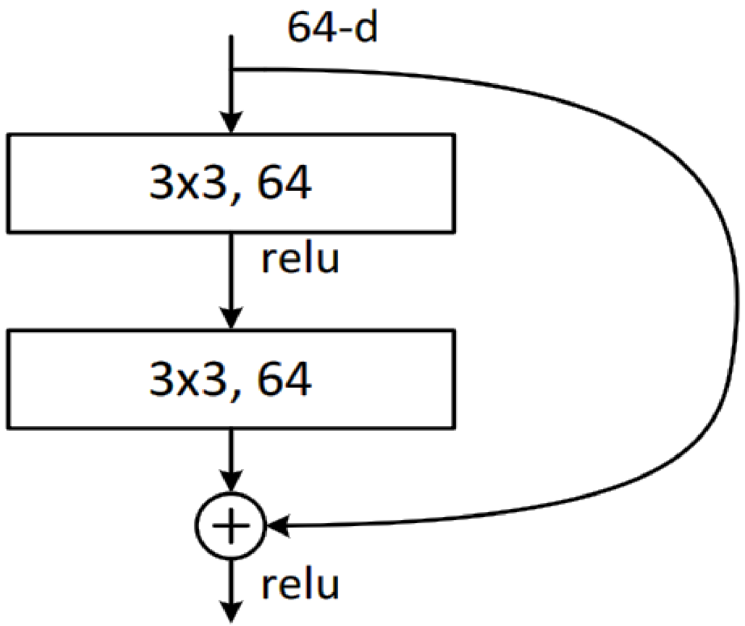
$$\mathbf{x}_L = \mathbf{x}_l + \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathcal{W}_i),$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L} \frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L} \left(1 + \frac{\partial}{\partial \mathbf{x}_l} \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathcal{W}_i)\right).$$

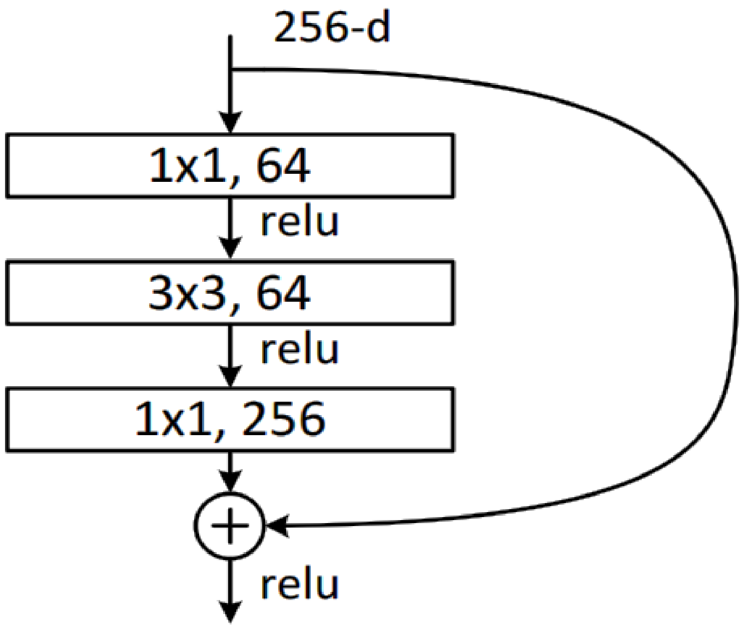
ResNet

- ResNe18(Basic block), ResNet50(Bottleneck block)

Basic block



Bottleneck block



Using Pretrained Model

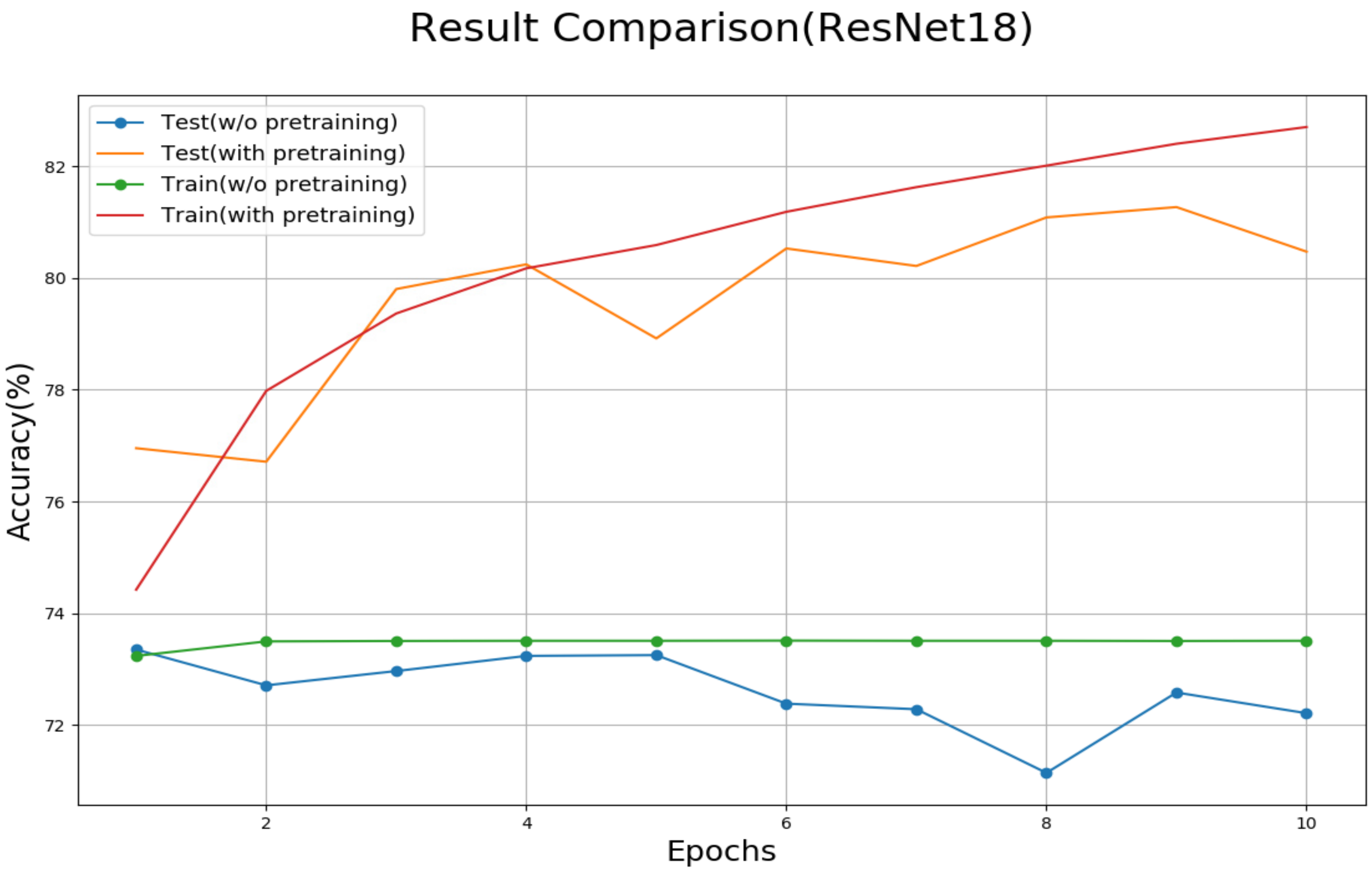
- Using pretrained model by torchvision module

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
                 :
  )
  (layer2): Sequential(
                 :
  )
  (layer3): Sequential(
                 :
  )
  (layer4): Sequential(
                 :
  )
  (avgpool): AvgPool2d(kernel_size=7, stride=1, padding=0)
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

You need to reinitialize the specific layers

Result Comparison

- 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.



Confusion Matrix

- A confusion matrix is a table that is often used to describe the performance of a classification model

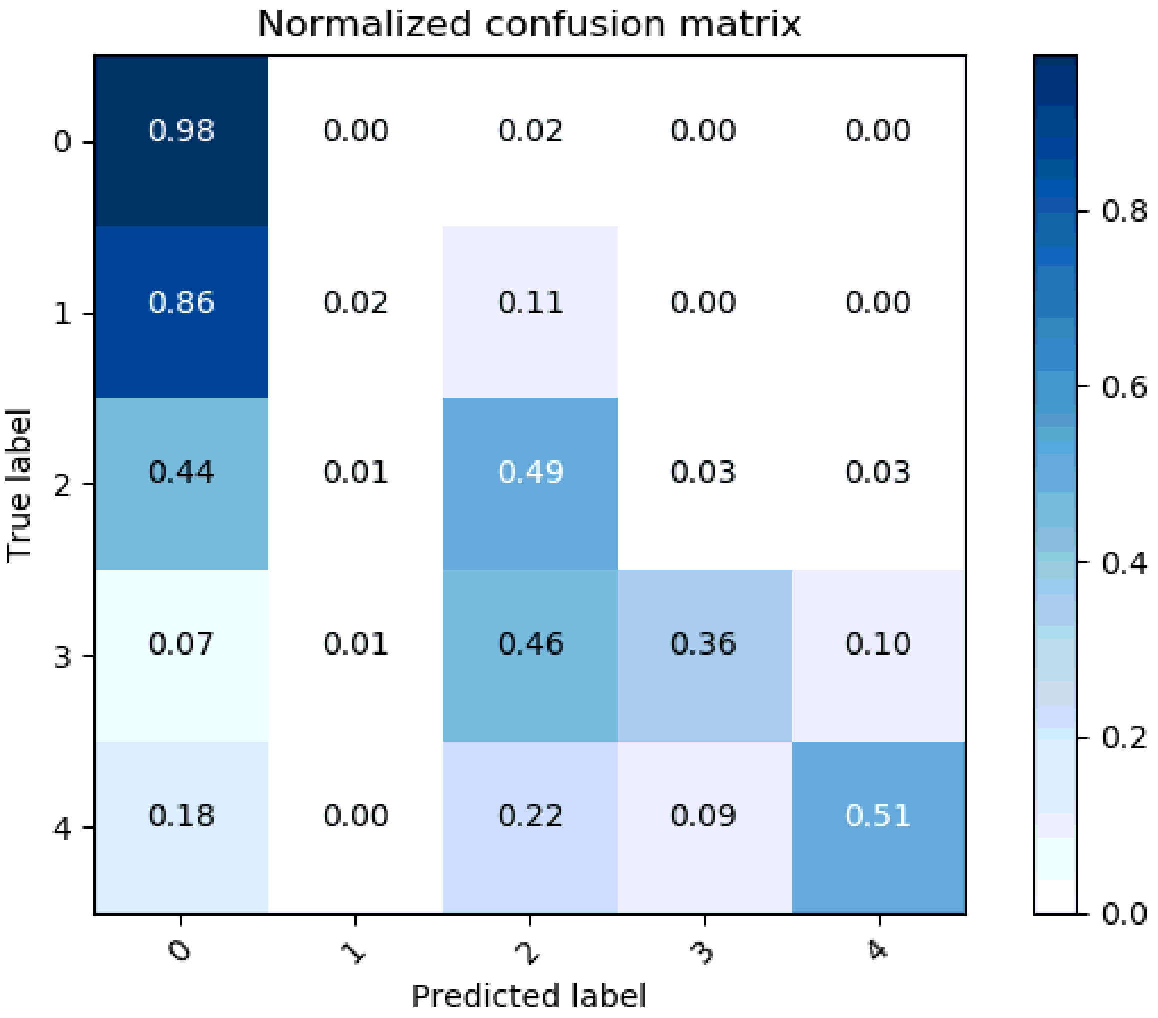
- y_true : ground truth label array
- y_pred: prediction array
- Classes: label name [‘0’, ‘1’, ‘2’, ‘3’, ‘4’]

```
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
```

Reference: https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py

Confusion Matrix

- Calculate the confusion matrix and plotting



Hyper Parameters

- Batch size= 4
- Learning rate = 1e-3
- Epochs = **10** (resnet18), **5** (resnet50)
- Optimizer: SGD Momentum = 0.9 Weight_decay = 5e-4
- Loss function: torch.nn.CrossEntropyLoss()

- **You can adjust the hyper-parameters according to your own ideas.**

- If you use "nn.CrossEntropyLoss", don't add softmax after final fc layer because this criterion combines LogSoftMax and NLLLoss in one single class.

Report Spec

Report Spec

1. Introduction (20%)
2. Experiment setups (30%)
   A. The details of your model (ResNet)
   B. The details of your Dataloader
   C. Describing your evaluation through the confusion matrix
3. Experimental results (30%)
   A. The highest testing accuracy
      - Screenshot
      - Anything you want to present
   B. Comparison figures
      - Plotting the comparison figures **(RseNet18/50, with/without pretraining)**
4. Discussion (20%)
   A. Anything you want to share

- ---- Criterion of result (40%) ----
- Accuracy > = 82% = 100 pts
- Accuracy 80~82% = 90 pts
- Accuracy 75~80% = 80 pts
- Accuracy < 75% = 70 pts

- **Score: 40% experimental results + 60% (report+ demo score)**
- **P.S If the zip file name or the report spec have format error, it will be penalty (-5).**

Reference

[1] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.