OASIS ML Group TRAINING 05

◆ ResNet, Data Augmentation, Pre-trained model

In this practice, you will need to analyze diabetic retinopathy. By doing this task, you suppose to learn several things as followed. First of all, you should try to know the past and future of ResNet, what's its benefit and drawbacks and implement ResNet18 and ResNet50 with PyTorch. Second, what's data augmentation and how to fulfill it is an important learning target. Last but not the least, the knowledge of learning why and how we use the pre-trained model will be your buddy in your master's life.

# This practice is referred to Lab04 of the DLP course at NCTUIOC.

Requirement：

Part I：

- Survey ResNet information
- Implement the **ResNet18**、**ResNet50** architecture with PyTorch.
- Implement your own custom DataLoader.
- Classify diabetic retinopathy grading via the ResNet architecture

Part II：

- Implement the ResNet18、ResNet50 architecture by **loading parameters from a pre-trained model**
- **Compare and visualize** the accuracy trend between the pre-trained model and without pretraining in the same architectures, you need to **plot each epoch accuracy** (not loss) during the training phase and testing phase.
- Calculate the **confusion matrix** and plotting.

Hint：

■ Diabetic retinopathy：

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 a large set of high-resolution retina images taken under a variety of imaging conditions. This Diabetic Retinopathy Detection dataset comes from Kaggle.

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

■ Prepare Data：

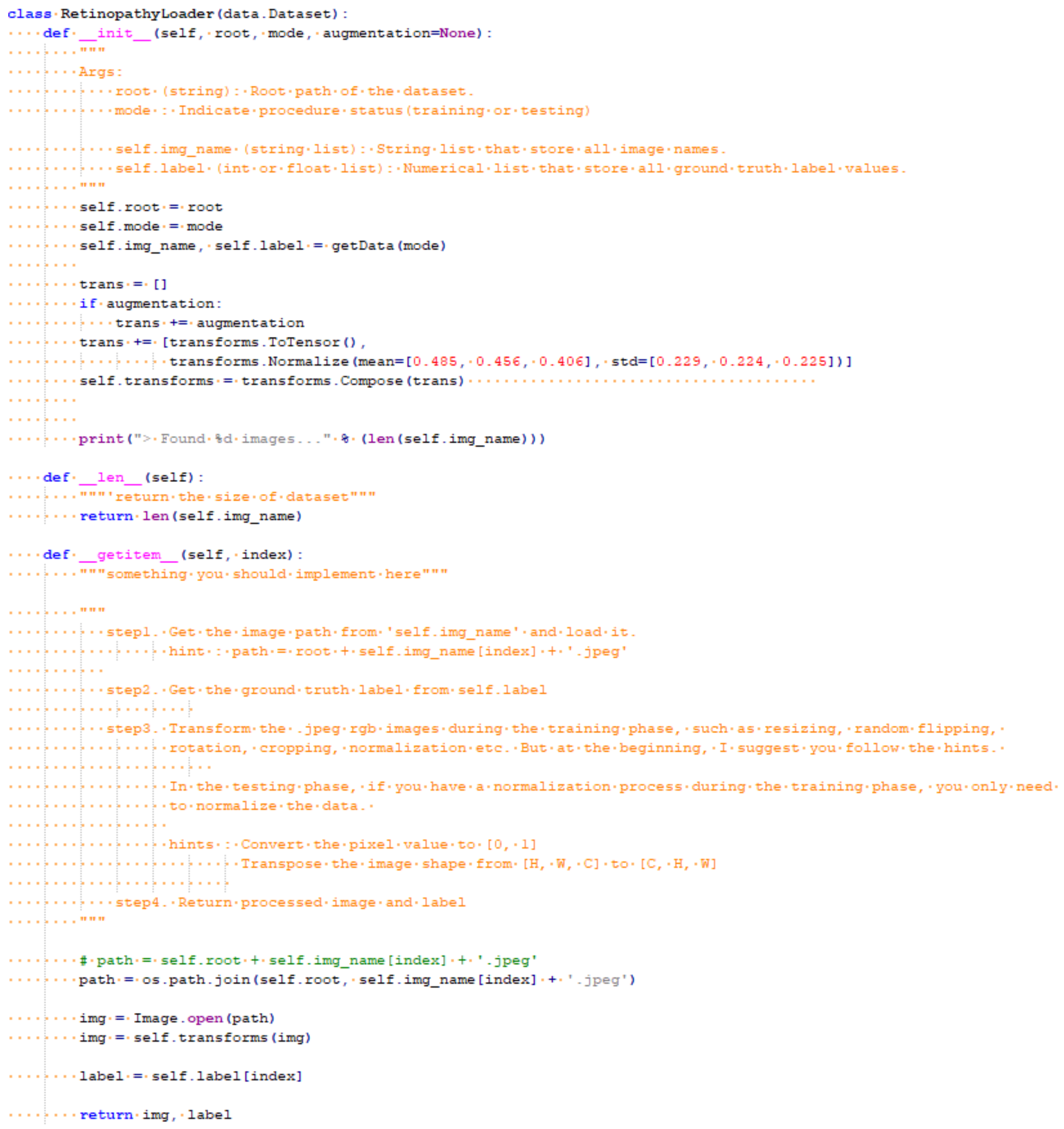
The dataset contains 35124 images, we divided the dataset into 28,099 training data and 7025 testing data. The image resolution is 512x512 and has been preprocessed.

- **Download link：**

  https://drive.google.com/open?id=1RTmrk7Qu9IBjQYLczaYKOvXaHWBS0o72

■ 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, augmentation=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.root = root
        self.mode = mode
        self.img_name, self.label = getData(mode)
        
        trans = []
        if augmentation:
            trans += augmentation
        trans += [transforms.ToTensor(),
                  transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])]
        self.transforms = transforms.Compose(trans)
        
        
        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):
        """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.root + self.img_name[index] + '.jpeg'
        path = os.path.join(self.root, self.img_name[index] + '.jpeg')

        img = Image.open(path)
        img = self.transforms(img)

        label = self.label[index]

        return img, label
```

- The init function is where the initial logic happens like reading a csv, assigning transforms etc.

- The getitem function returns the data and labels and you can process the data like loading image, preprocessing, transforming image before returns.
- The index parameter where in the getitem function, it is the nth data/image you are going to return.
- **Reference**：

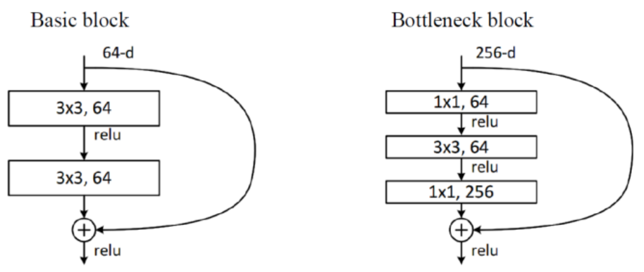
  https://pytorch.org/tutorials/beginner/data_loading_tutorial.html

  https://github.com/utkuozbulak/pytorch-custom-dataset-examples
- You can use getData function to read all images name and ground truth.

```
def getData(mode):
    if mode == 'train':
        img = pd.read_csv('train_img.csv', header=None)
        label = pd.read_csv('train_label.csv', header=None)
        return np.squeeze(img.values), np.squeeze(label.values)
    else:
        img = pd.read_csv('test_img.csv', header=None)
        label = pd.read_csv('test_label.csv', header=None)
        return np.squeeze(img.values), np.squeeze(label.values)
```

ResNet (Residual Network)：

- What is **Skip/Shortcut connection in ResNet?**
- Why ResNet can avoid vanishing gradient problem?
- Building residual basic block and bottleneck block



Data Augmentation ：

- Pytorch 提供之 torchvision data augmentation 技巧
- TRANSFORMING AND AUGMENTING IMAGES
- TORCHVISION.TRANSFORMS

Using pretrained model and reinitialize the specific layers：

- FINETUNING TORCHVISION MODELS
- TORCHVISION.MODELS
- Pretrained models for Pytorch

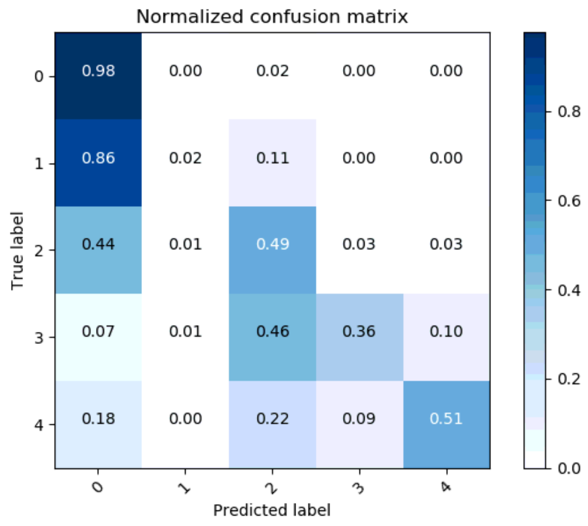
Confusion matrix：

A confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known. It is extremely useful for measuring Recall, Precision, Specificity, Accuracy, and most importantly AUC-ROC curve.

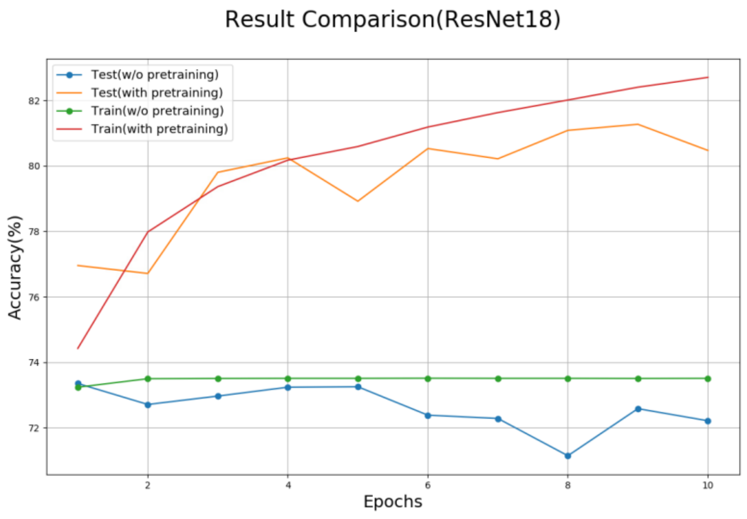
- **Sample code:**

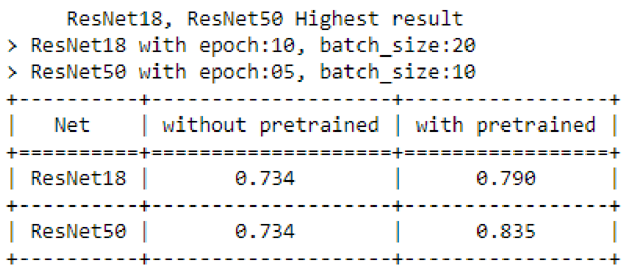
  You can use the example and library to conduct this section.

■ The confusion matrix will be similar as the following example:



Visualization figure of accuracy, comparison should like example as below.





```
     ResNet18, ResNet50 Highest result
> ResNet18 with epoch:10, batch_size:20
> ResNet50 with epoch:05, batch_size:10
+----------+--------------------+-----------------+
|   Net    | without pretrained | with pretrained |
+==========+====================+=================+
| ResNet18 |       0.734        |      0.790      |
+----------+--------------------+-----------------+
| ResNet50 |       0.734        |      0.835      |
+----------+--------------------+-----------------+
```