OASIS ML Group TRAINING 07

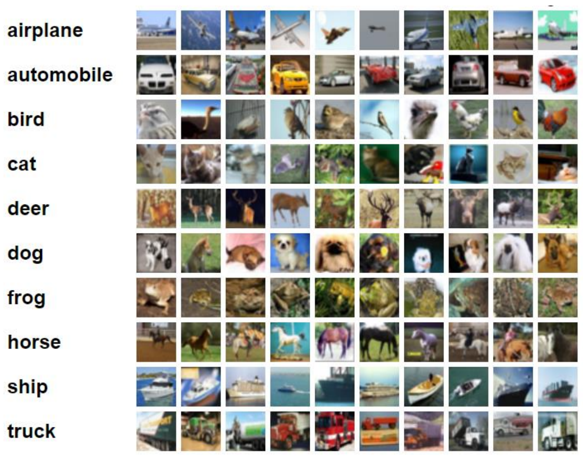
◆ CIFAR-10

In this practice, you are going to build a LeNet model by using PyTorch. This practice is quite similar with Training 05. Use the **CIFAR** datasets which is a collection of images that are commonly used to train ML and computer vision as training datasets. In this practice, you will learn how to **train an image classifier** and skill of **improve the testing accuracy with simple data augmentation**. Follow hints below to finish it.

▫ Practice：

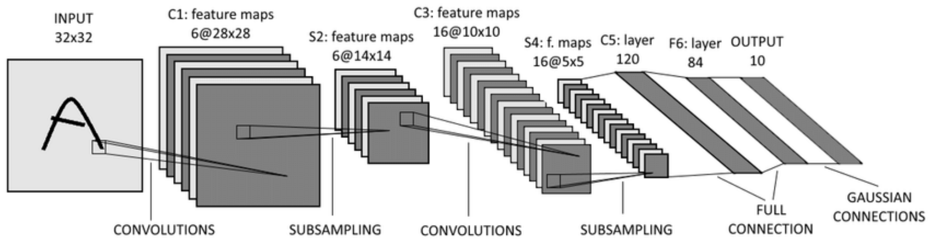
■ Analysis datasets：

CIFAR-10 is one of the most widely used datasets for machine learning research. The CIFAR-10 dataset contains **60,000 32x32** color images in **10 different classes**. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images of each class. Since the images in CIFAR-10 are low-resolution (32x32), this dataset can allow researchers to quickly try different algorithms to see what works.



■ LeNet：

You are asked to construct a classical CNN model called LeNet. LeNet is a convolutional neural network structure proposed by Yann LeCun et al. in 1989. Since the success of AlexNet in 2012, CNN has become the best choice for computer vision applications and many different types of CNN has been raised. Nowadays, CNN models are quite different from Lenet, but they are all developed on the basis of LeNet.



■ PyTorch-torchvision：

Specifically for vision, PyTorch have created a package called torchvision, that has data loaders for common datasets such as Imagenet, CIFAR10, MNIST, etc. and data transformers for images viz., torchvision.datasets and torch.utils.data.DataLoader.

■ Image Classifier：

Do the following steps in order:

1. Load and normalize the CIFAR10 training and test datasets using torchvision
2. Define a Convolutional Neural Network
3. Define a loss function.
4. Train the network on the training data.
5. Test the network on the test data.

Hint 01：Import library you needed.

```python
import torch
import torchvision
from torch.autograd import Variable
import torch.nn as nn
import torch.optim as optim
import torch.utils.data as Data
import torchvision.transforms as transforms
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import time
import sys
```

Hint 02：Check and set GPU.

```python
# check GPU
device = 'cuda:0' if torch.cuda.is_available() else 'cpu'
print('GPU state:', device)
```

Hint 03：Load the CIFAR-10 datasets. Here i provide two methods to load the datasets. One is get from torchvision.datasets.CIFAR10, another is load from pre-saved datasets by using unpickle function.

[Please see the sample code in reference part!]

Hint 04：Do the **basic** image pre-processing.

```python
# Image pre-processing
transform = transforms.Compose([
    # ToTensor是指把PIL.Image(RGB) 或者numpy.ndarray(H x W x C)
    # 從0到255的值對映到0到1的範圍內，並轉化成Tensor格式
    transforms.ToTensor(),
    transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5))
    ]
)

def target_transform(label):
    label  = np.array(label)
    target = torch.from_numpy(label).long()
    return target
```

Hint 05： Construct the training and testing **DataLoader**.

```
train_data = Cifar10_Dataset(True, transform, target_transform)
print('size of train_data:{}'.format(train_data.__len__()))

test_data = Cifar10_Dataset(False,transform, target_transform)
print('size of test_data:{}'.format(test_data.__len__()))

train_loader = Data.DataLoader(dataset=train_data, batch_size = BATCH_SIZE, shuffle=True)
test_loader  = Data.DataLoader(dataset=test_data, batch_size = 100, shuffle=True)
```

terminal：

```
size of train_data:50000
size of test_data:10000
```

Hint 06： construct LeNet by yourself.

```
#Lenet Model structure define

class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        '''
        Code here
        '''

    def forward(self,x):
        '''
        Code here
        '''
        return x

model = LeNet().to(device)
from torchsummary import summary
summary(model.cuda(), (3, 32, 32))
```

Hint 07： Define Hyper parameters by yourself.

Recommend initial parameters：

- learning_rate = 0.001
- batch_size = 128
- EPOCH：30~50
- Optimizer：Adam
- Loss function：CrossEntrpyLoss()

(You can tune by yourself.)

Hint 8： Define function you need then start training. At the last epoch, please show the classified result and highest testing accuracy as shown below.

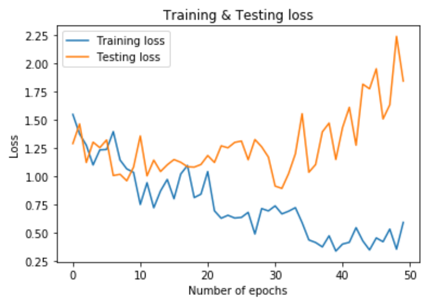
```
◆Epoch: [50/50]

 Progress:[*************************************************************************************************** ] 100%, 21.75s
Train --> Epoch：50, Loss: 0.5931550860404968, Acc: 86.417999%
--TESTING!!--
 Test result --> Acc: 61.169998%
 Come on, keep going!
 Accuracy of each classes
 Test Accuracy of plane: 63.1%, [631/1000]
 Test Accuracy of car: 74.5%, [745/1000]
 Test Accuracy of bird: 54.3%, [543/1000]
 Test Accuracy of cat: 42.6%, [426/1000]
 Test Accuracy of deer: 58.3%, [583/1000]
 Test Accuracy of dog: 48.1%, [481/1000]
 Test Accuracy of frog: 68.2%, [682/1000]
 Test Accuracy of horse: 67.3%, [673/1000]
 Test Accuracy of ship: 68.5%, [685/1000]
 Test Accuracy of truck: 66.8%, [668/1000]
 --> Highest testing value so far： 64.47000122070312
```

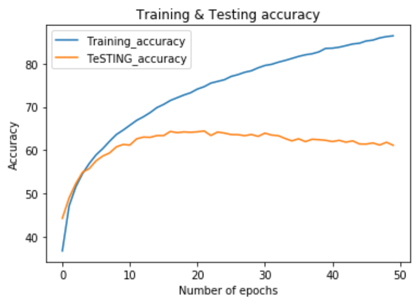
Hint 9： Please show the whole execution time.

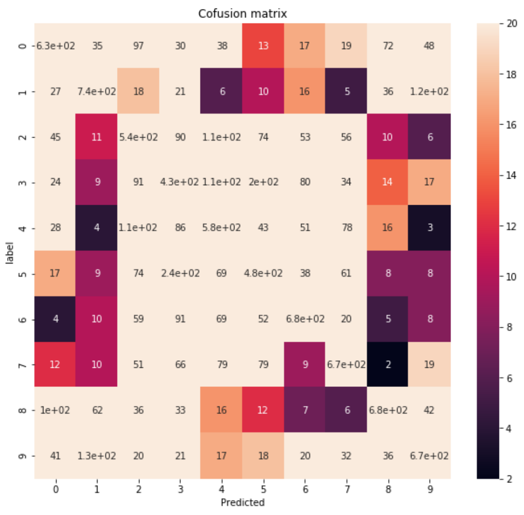
```
Excution time : 1333.270322 sec
```

Hint 10： Plot the **loss-epoch** curve and **accuracy-epoch** curve (*both training and testing*). Also show the **confusion matrix**.

(Remember to save the highest testing accuracy and the best model with code provided below)







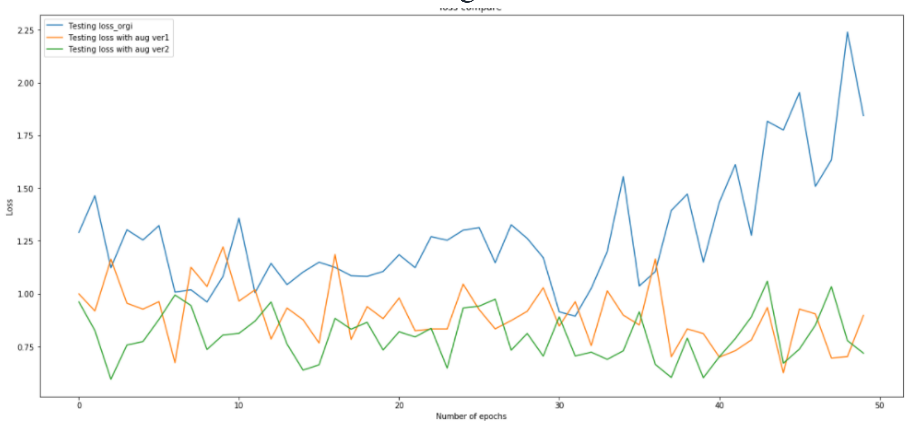
```python
torch.save(model.state_dict(), './model/'+ model_name)
print(" -> Save the best model & value so far ~")
```

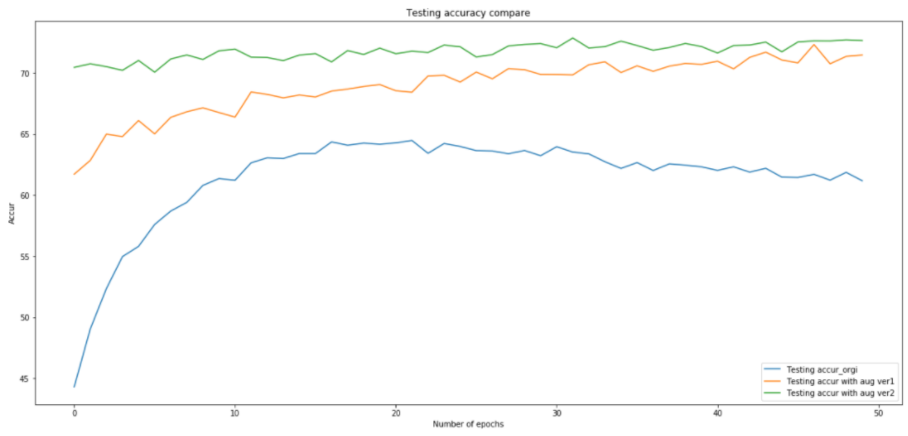
Hint 11： Do more image pre-processing with code below to improve the performance.

```python
transform_train_ver1 = transforms.Compose([
    transforms.RandomCrop(32, padding=4),

    # hori-flep with prob=0.5
    transforms.RandomHorizontalFlip(),

    transforms.ToTensor(),
    transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])])
```

```python
transform_train_aug2 = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])

# Normalize the test set same as training set without augmentation
transform_test_aug2 = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])
```

Hint 12：Please plot the figure and table of comparing original LeNet, LeNet with augmentation version1 and LeNet with augmentation version2.





```
Compare Reult
+------------------+--------+-----------------------+-----------------------+
|      Version     | LeNet  |       LeNet with      |       LeNet with      |
|                  |        |   augmentation ver1   |   augmentation ver2   |
+==================+========+=======================+=======================+
| Best Accuracy (%)| 64.470 |        72.320         |        72.860         |
+------------------+--------+-----------------------+-----------------------+
```

▫ Reference：

- PyTorch document [Link]
- PyTorch document *(Chinese)* [Link]
- The CIFAR-10 dataset [Link]
- LeNet-5 [Link] [Github]
- Sample code [Link]

▫ Reference paper：

- Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791. ***[Link] [中文分享]***