THE REPORT

Of

SIGNAL & INFORMATION FINAL PROJECT

ADEO1 PENG KESHU

MO LI

LE Thu Huong

2018-06-07

PROJECT: CNN PROJECT-Handwritten Digits Recognition

Team member: ADEO1 PENG Keshu; MO LI; LE Thu Huong

1. INTRODUCTION

The aim of this project is to implement a CNN Lenet-5 to recognize digits(0-9).

This report presents our implementation of the CNN layer of 10*10 matrix combined with Pytorch(which provide our configuration environment), to recognize the numerical digits. The program was able to achieve an accuracy rate of 98%. This program also visualizes the results of recognition.

2. BACK GROUND AND MOTIVATION

Hand writing recognition, has great importance and uses in many domain: recognize zip code on mail, packet, deposit money into account in banking industry, all form required numeric entries in form filled up by hand...etc. There are different challenges faced while attempting to solve this problem. The hand writing digits are often not of the same size, thickness, orientation and position relative to the margins. Also, people write the same digit in many different forms. For images application, we often use convolutional neural network, abbreviated CNN. Image is the type of instructed data where we have to recognize what is in the image. In other word, these input data are not in well defined. For this particular problem, we will apply LeNet-5 the latest neural network to solve this issue. As we have known, the accuracy of the model depends on two components: (1) depends on your numbers of input (x) of output(y) and the number of hidden layer. According to Dominos [2] more data beats a clever algorithm, we are decided to use all the original MINIST data set [1]. However, we understand and accept the large scale of this application, to be require more time consuming and resources consuming it takes.

3. Introduction of CNN

Convolution Neural Networks are a special kind of multi-layer neural networks used to recognize visual patterns with extreme variability of handwritten digits, and with robustness to distortions and simple geometric transformations. LeNet -5 has its architecture diagram as Fig.1 [2]. LeNet-5 comprises 7 layers, all of which contain weights as trailable parameters. The values of the input pixels are normalized to make the mean input roughly 0, and the variance roughly 1, which accelerates learning. Layer C1, C3 and C5 are convolutional layers with 6, 16 and 120 feature maps respectively. Each unit in each feature maps are successively connected to the neighboring 5*5 feature maps, assisting training by localization. The layers S2 and S4 are subsampling layers with 6 feature maps of size 14*14 and 16 feature maps of size 28*28 respectively.

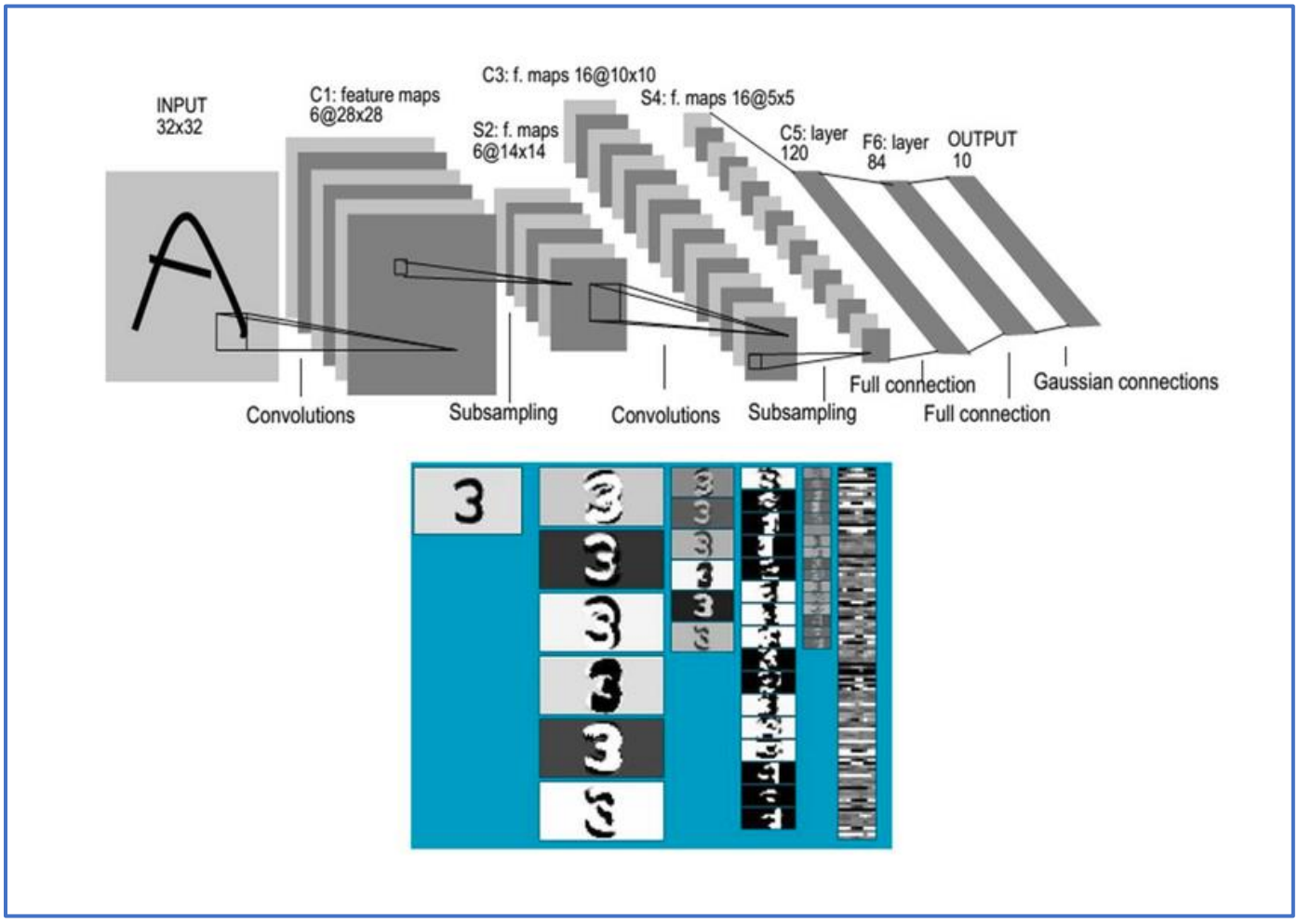


Fig.1 : Architecture of LeNet-5, a convolutional neural network, for digits recognition.

Given the high accuracy achieved by this technique, this was our initial proposed algorithm. However, the implementation of this algorithm is quite complex, it perform more efficiently for an extensively large database of digits as MINIST.

The reason we choosing python to develop this application is because of its extensively library available. More importantly, we have the ability to change the algorithm easily and quickly, faster computation and executing algorithm, to be train the model so much faster.

4. Program implementation

4.1 Operation System:

Linux

4.2 Software Requirement:

*) Pytorch

*) Terminal

*) Python: for program development only.

*) OpenCV

4.3 Data set

We are using the MINIST data set of images and 60000 testing images and 10000 testing images of the MINIST data set[1] . Each image is a 28x28 grayscale (0-255) labeled representation of an individual digit.

Fig.1 represents the sample images taken from the MINIST database[1]

4.4 Install software

1.0 Download Pytorch at : http://download.pytorch.org/whl/cpu/torch-0.4.0-cp27-cp27mu-linux_x86_64.whl

1.1 on Terminal: sudo pip install torch-0.4.0-cp27-cp27mu-linux_x86_64.whl

1.3 on Terminal: sudo pip install torchvision

1.4 on Terminal: sudo pip install opencv-python

At this step, you have successfully install all required software

4.5 DEVELOP PROGRAM

- **Step 0 Import the package**

```
from __future__ import print_function
import torch
import torch.utils.data as Data
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.autograd import Variable
```

- **Step 1 Data Pre-processing**

This step we need to convert training data into a DataLoader that Torch can use. This makes it easier to use batches for training.

```
# prepare the data for training and testing
batch_size = 10
test_batch_size = 10

train_loader = Data.DataLoader(
    datasets.MNIST('./data', train=True, download=True,
                   transform=transforms.Compose([transforms.ToTensor(),
                                                 transforms.Normalize((0.1307,), (0.3081,))])),
    batch_size=10, shuffle=True
)

test_loader = Data.DataLoader(
    datasets.MNIST('./data', train=False, download=True,
                   transform=transforms.Compose([transforms.ToTensor(),
                                                 transforms.Normalize((0.1307,), (0.3081,))])),
    batch_size=10, shuffle=True
)

```

- **Step 2 Define the network structure**

There are four points:

1) Class CNN need to inherit from Module;

2) They need to call the constructor of the parent class: super(CNN, self).__init__();

3) To activate the function Relu in Pytorch is also a kind of layer;

4) The forward() method needs to be implemented for the forward propagation of the network, while the backward propagation only needs to call Variable.backward()method.

```
# define the whole network and do the process for pics
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x)

model = Net()
if torch.cuda.is_available():
    print('E')
    model.cuda()
```

- **Step 3 Training**

We took 60,000 samples here.

```
def train(epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        if torch.cuda.is_available():
            data, target = data.cuda(), target.cuda()

        data, target = Variable(data), Variable(target)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 50 == 0:
            print('Train Epoch: {} [{}/{} ({:.1f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx*len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.data[0]
            ))

```

- **Step 4 Testing**

We took 10,000 samples for testing as well.

```
def val():
    model.eval()
    test_loss = 0
    correct = 0
    for data, target in test_loader:
        data, target = Variable(data, volatile=True), Variable(target)
        output = model(data)
        test_loss = test_loss + F.nll_loss(output, target, size_average=False).data[0]
        pred = output.data.max(1, keepdim=True)[1]
        correct += pred.eq(target.data.view_as(pred)).cpu().sum()
    test_loss /= len(test_loader.dataset)
    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.3f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)
    ))
```

●Step 5 Visualize the Testing Result

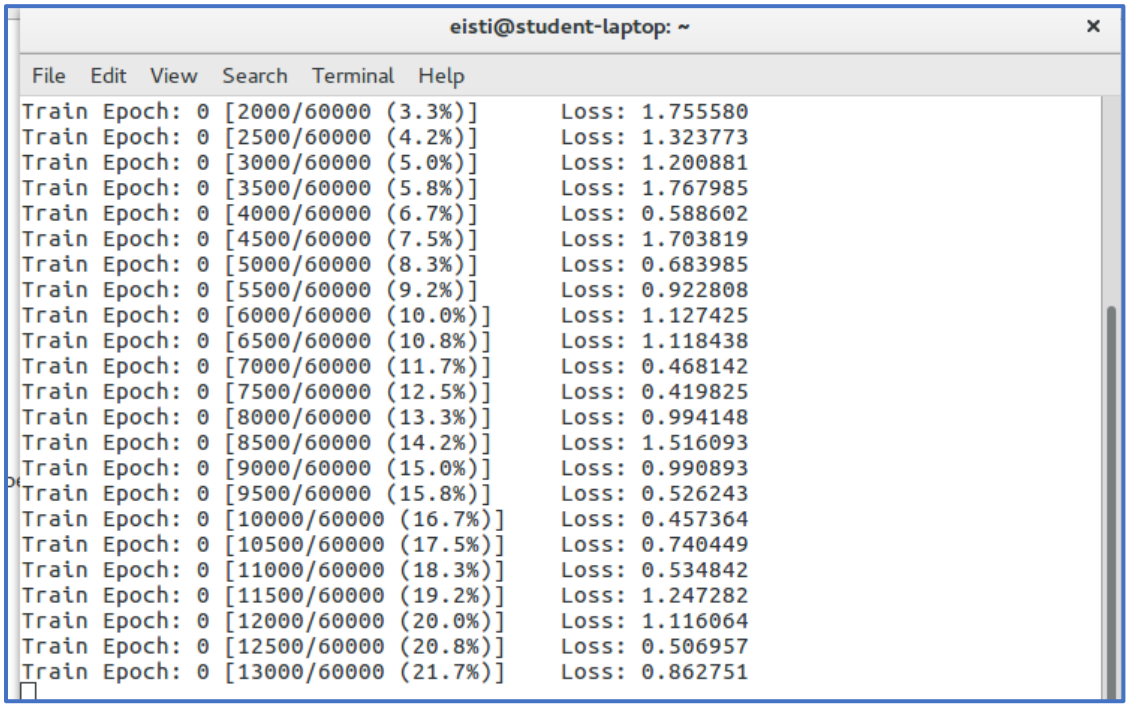
In the last step, we can have those data to be trained and tested. However, we can only know how many data we are training, how many data we lost in the training process and the accuracy of test result. So we create a new python file to visualize the test results.

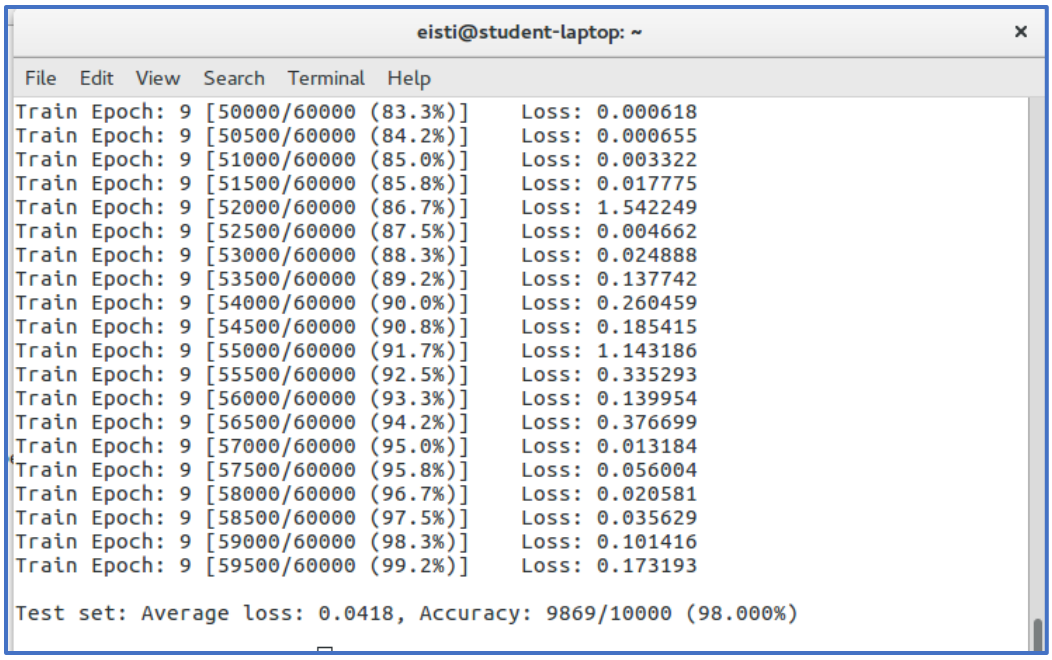
```python
def my_show_test(my_model):

    for data, target in my_test_loader:
        # print('data.shape', data.numpy().shape)
        temp = data
        data, target = Variable(data, volatile=True), Variable(target)
        output = my_model(data)
        out = output.data[0].numpy()
        # print(out)
        print('perdict number is  : ', np.where(out == np.max(out))[0])
        im = cv2.resize((temp.numpy())[0].transpose([1, 2, 0]),(280,280),interpolation=cv2.INTER_CUBIC)
        cv2.imshow('lenet', im)
        cv2.waitKey(0)


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x)

my_model = Net()
state_dict = torch.load('./lenet.pkl')['state_dict']
# from collections import OrderedDict
# new_state_dict = OrderedDict()
# for k, v in state_dict.items():
#     name = k[7:]
#     new_state_dict[name] = v
# my_model.load_state_dict(new_state_dict)
my_model.load_state_dict(state_dict)

my_show_test(my_model)
```

5. TEST RESULTS

We run the training and test file.

On terminal: python trainingData.py

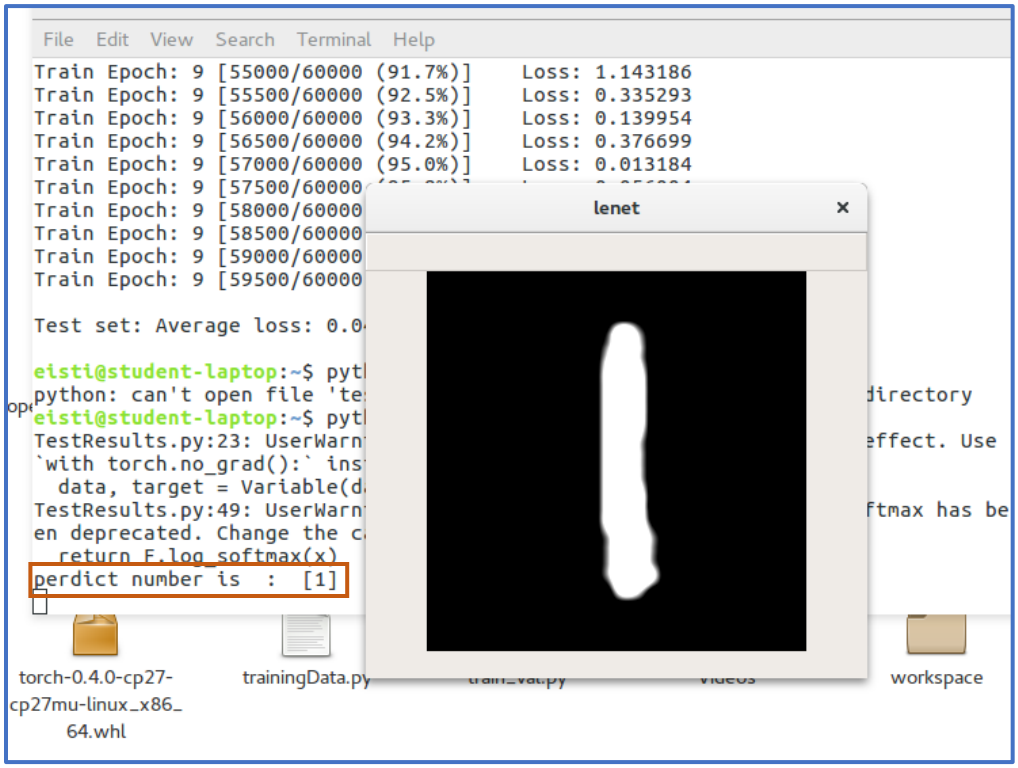




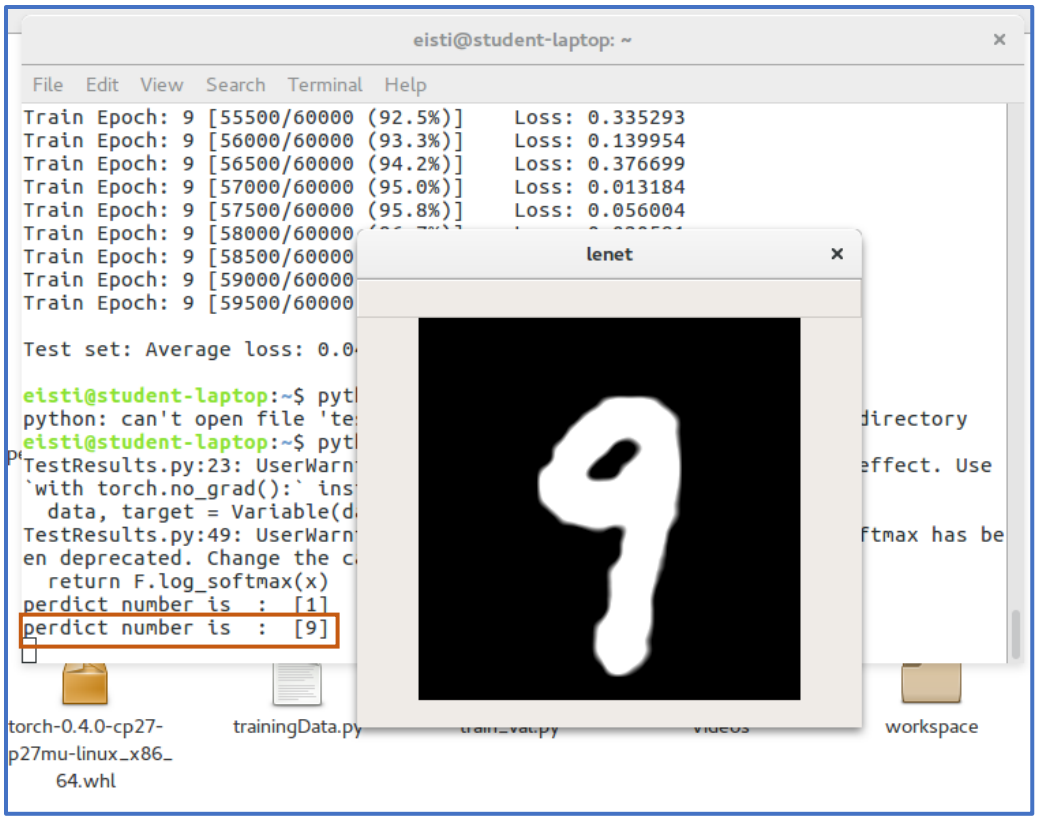
We can see that the accuracy can reach 98%.

Then we run the visualization file to see the test results.

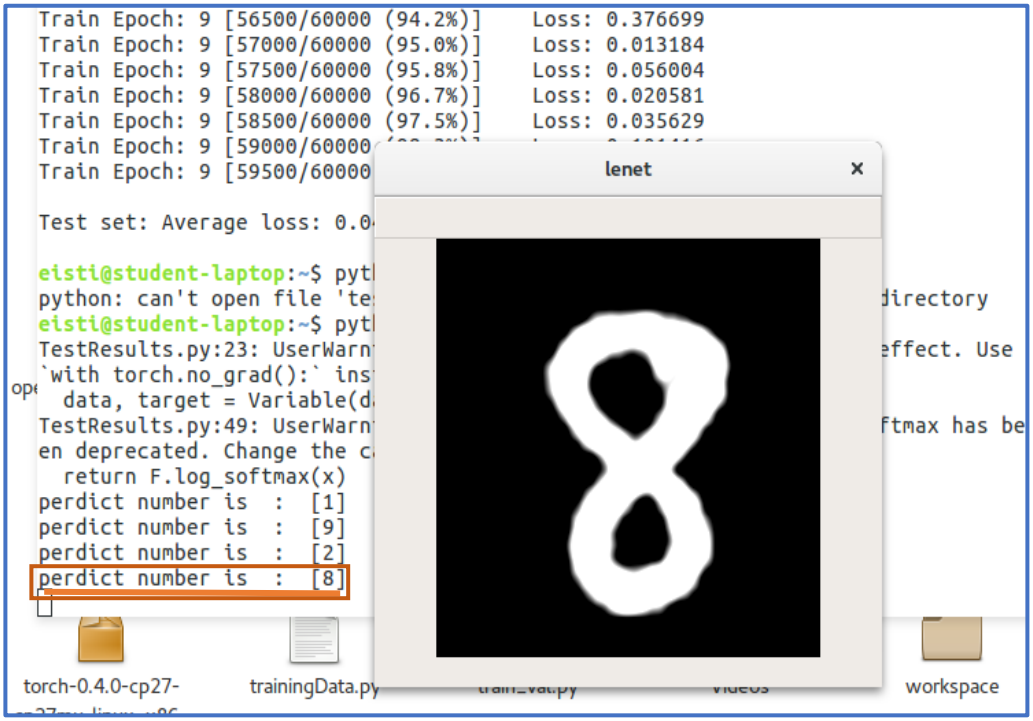
On terminal: python TestResults.py



The predict number is : 1



The predict number is : 9



The predict number is : 8

6. CONSLUSION

This project was considered as a study to approach knowledge and an examination to test my capability in developing neural network project in the future. Therefore, we are expecting to receive more advices that help to improve the code, and provide a better accuracy and less time consuming for the next application.

Thank you!

Best regards,

Team1 : Huong LE – Mo LI – Keshu PENG

REFERENCE

[1] MNISTDatabase of Handwritten digits: http://yann.lecun.com/exdb/mnist/

[2] A few useful things to know about machine learning
https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf

APPENDIX : SOURCE CODE

1. The file: trainingData.py

```
# coding=utf-8
from __future__ import print_function
import torch
import torch.utils.data as Data
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.autograd import Variable
```

```
# prepare the data for training and testing
batch_size = 10
test_batch_size = 10

train_loader = Data.DataLoader(
    datasets.MNIST('./data', train=True, download=True,
                   transform=transforms.Compose([transforms.ToTensor(),
                                                 transforms.Normalize((0.1307,),
(0.3081,))])),
    batch_size=10, shuffle=True
)

test_loader = Data.DataLoader(
    datasets.MNIST('./data', train=False, download=True,
                   transform=transforms.Compose([transforms.ToTensor(),
                                                 transforms.Normalize((0.1307,),
(0.3081,))])),
    batch_size=10, shuffle=True
)

# define the whole network and do the process for pics
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x)

model = Net()
if torch.cuda.is_available():
    print('E')
    model.cuda()

# optimization
```

```
optimizer = optim.SGD(model.parameters(), lr=0.01)

# traning data. we use 60000 data for training
def train(epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        if torch.cuda.is_available():
            data, target = data.cuda(), target.cuda()

        data, target = Variable(data), Variable(target)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 50 == 0:
            print('Train Epoch: {} [{}/{} ({:.1f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx*len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.data[0]
            ))

# testing data. we use 10000 data for testing
def val():
    model.eval()
    test_loss = 0
    correct = 0
    for data, target in test_loader:
        data, target = Variable(data, volatile=True), Variable(target)
        output = model(data)
        test_loss = test_loss + F.nll_loss(output, target, size_average=False).data[0]
        pred = output.data.max(1, keepdim=True)[1]
        correct += pred.eq(target.data.view_as(pred)).cpu().sum()
    test_loss /= len(test_loader.dataset)
    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.3f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)
    ))



for epoch in range(0, 10):
    train(epoch)
    val()

torch.save({'state_dict': model.state_dict(),}, 'lenet.pkl')
```

2. The file TestResults.py

```
# coding=utf-8
from __future__ import print_function
import torch
import torch.utils.data as Data
import torch.nn as nn
import torch.nn.functional as F
from torchvision import datasets, transforms
from torch.autograd import Variable
import cv2
import numpy as np

my_test_loader = Data.DataLoader(
    datasets.MNIST('./data', train=False, download=True,
                   transform=transforms.Compose([transforms.ToTensor(),
                                                 transforms.Normalize((0.1307,), (0.3081,))])),
    batch_size=1, shuffle=True
)
def my_show_test(my_model):

    for data, target in my_test_loader:
        # print('data.shape', data.numpy().shape)
        temp = data
        data, target = Variable(data, volatile=True), Variable(target)
        output = my_model(data)
        out = output.data[0].numpy()
        # print(out)
        print('perdict number is  : ', np.where(out == np.max(out))[0])
        im = cv2.resize((temp.numpy())[0].transpose([1, 2, 0]),(280,280),interpolation=cv2.IN-
TER_CUBIC)
        cv2.imshow('lenet', im)
        cv2.waitKey(0)


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
```

```
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x)

my_model = Net()
state_dict = torch.load('./lenet.pkl')['state_dict']
# from collections import OrderedDict
# new_state_dict = OrderedDict()
# for k, v in state_dict.items():
#     name = k[7:]
#     new_state_dict[name] = v
# my_model.load_state_dict(new_state_dict)
my_model.load_state_dict(state_dict)

my_show_test(my_model)
```