Mathematical Foundations for Computer Vision and Machine Learning -Deep Learning application based on pytorch library using real data.

ID : 20131790 Name : So-Hyun Kwon

December 7, 2017

1 Problem Definition

- Make deep Learning application based on pytorch library using real data.
- Make digit recognition application with MNIST - Convolutional Neural Network.
- I used pytorch and openCV libraries.

2 Algorithm

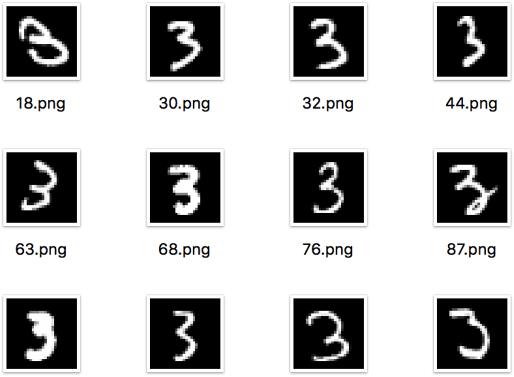
2.1 Definition

- Convolutional neural network (CNN) is a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery.
- The MNIST database is a large database of handwritten digits that is commonly used for training various image processing systems.
- Pytorch is an open source machine learning library, a scientific computing framework, and a script language based on the Lua programming language.
  It provides a wide range of algorithms for deep machine learning.

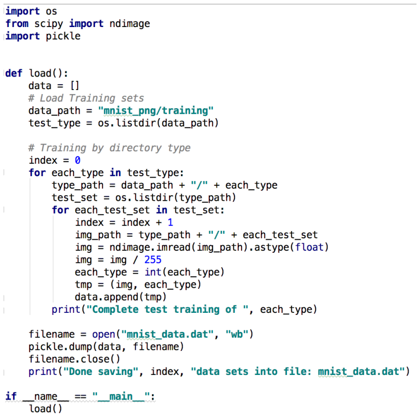
- OpenCV(Open source computer vision) is a library of programming functions mainly aimed at real-time computer vision. OpenCV supports the Deep learning framework Pytorch.

2.2 Process of algorithm and code description

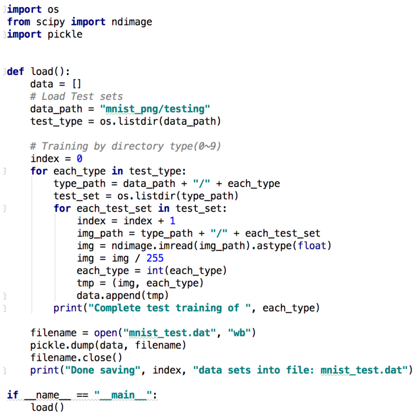
(i) Test data : MNIST
example :



(ii) Load train data and test data and make data file as name of *mnist_data.dat*, *mnist_test.dat*

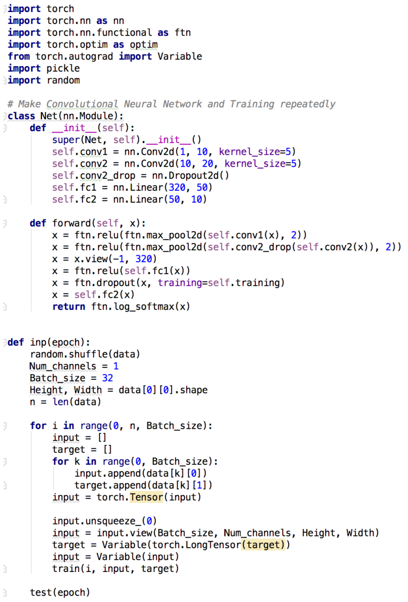


```python
import os
from scipy import ndimage
import pickle


def load():
    data = []
    # Load Training sets
    data_path = "mnist_png/training"
    test_type = os.listdir(data_path)

    # Training by directory type
    index = 0
    for each_type in test_type:
        type_path = data_path + "/" + each_type
        test_set = os.listdir(type_path)
        for each_test_set in test_set:
            index = index + 1
            img_path = type_path + "/" + each_test_set
            img = ndimage.imread(img_path).astype(float)
            img = img / 255
            each_type = int(each_type)
            tmp = (img, each_type)
            data.append(tmp)
        print("Complete test training of ", each_type)

    filename = open("mnist_data.dat", "wb")
    pickle.dump(data, filename)
    filename.close()
    print("Done saving", index, "data sets into file: mnist_data.dat")

if __name__ == "__main__":
    load()
```

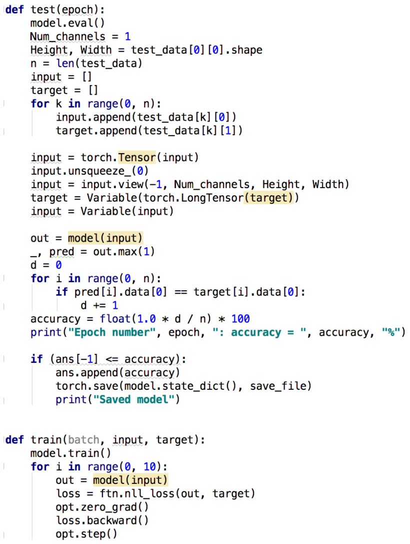


```python
import os
from scipy import ndimage
import pickle


def load():
    data = []
    # Load Test sets
    data_path = "mnist_png/testing"
    test_type = os.listdir(data_path)

    # Training by directory type(0~9)
    index = 0
    for each_type in test_type:
        type_path = data_path + "/" + each_type
        test_set = os.listdir(type_path)
        for each_test_set in test_set:
            index = index + 1
            img_path = type_path + "/" + each_test_set
            img = ndimage.imread(img_path).astype(float)
            img = img / 255
            each_type = int(each_type)
            tmp = (img, each_type)
            data.append(tmp)
        print("Complete test training of ", each_type)

    filename = open("mnist_test.dat", "wb")
    pickle.dump(data, filename)
    filename.close()
    print("Done saving", index, "data sets into file: mnist_test.dat")

if __name__ == "__main__":
    load()
```

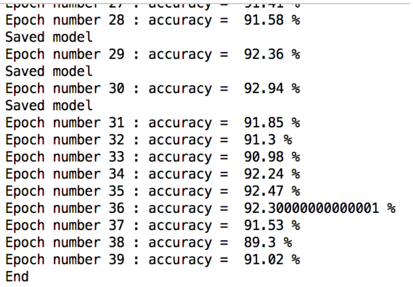
(iii) Make Convolutional Neural Network and Training   Testing repeatedly about 40 times.



```
import torch
import torch.nn as nn
import torch.nn.functional as ftn
import torch.optim as optim
from torch.autograd import Variable
import pickle
import random

# Make Convolutional Neural Network and Training repeatedly
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 = ftn.relu(ftn.max_pool2d(self.conv1(x), 2))
        x = ftn.relu(ftn.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = ftn.relu(self.fc1(x))
        x = ftn.dropout(x, training=self.training)
        x = self.fc2(x)
        return ftn.log_softmax(x)


def inp(epoch):
    random.shuffle(data)
    Num_channels = 1
    Batch_size = 32
    Height, Width = data[0][0].shape
    n = len(data)

    for i in range(0, n, Batch_size):
        input = []
        target = []
        for k in range(0, Batch_size):
            input.append(data[k][0])
            target.append(data[k][1])
        input = torch.Tensor(input)

        input.unsqueeze_(0)
        input = input.view(Batch_size, Num_channels, Height, Width)
        target = Variable(torch.LongTensor(target))
        input = Variable(input)
        train(i, input, target)

    test(epoch)
```

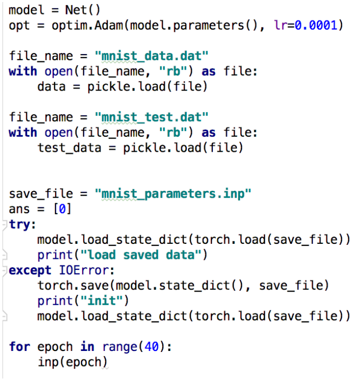


```
def test(epoch):
    model.eval()
    Num_channels = 1
    Height, Width = test_data[0][0].shape
    n = len(test_data)
    input = []
    target = []
    for k in range(0, n):
        input.append(test_data[k][0])
        target.append(test_data[k][1])

    input = torch.Tensor(input)
    input.unsqueeze_(0)
    input = input.view(-1, Num_channels, Height, Width)
    target = Variable(torch.LongTensor(target))
    input = Variable(input)

    out = model(input)
    _, pred = out.max(1)
    d = 0
    for i in range(0, n):
        if pred[i].data[0] == target[i].data[0]:
            d += 1
    accuracy = float(1.0 * d / n) * 100
    print("Epoch number", epoch, ": accuracy = ", accuracy, "%")

    if (ans[-1] <= accuracy):
        ans.append(accuracy)
        torch.save(model.state_dict(), save_file)
        print("Saved model")


def train(batch, input, target):
    model.train()
    for i in range(0, 10):
        out = model(input)
        loss = ftn.nll_loss(out, target)
        opt.zero_grad()
        loss.backward()
        opt.step()
```



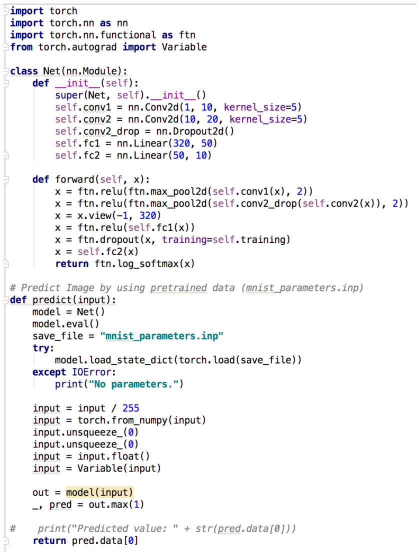
```
Epoch number 27 : accuracy =  91.41 %
Epoch number 28 : accuracy =  91.58 %
Saved model
Epoch number 29 : accuracy =  92.36 %
Saved model
Epoch number 30 : accuracy =  92.94 %
Saved model
Epoch number 31 : accuracy =  91.85 %
Epoch number 32 : accuracy =  91.3 %
Epoch number 33 : accuracy =  90.98 %
Epoch number 34 : accuracy =  92.24 %
Epoch number 35 : accuracy =  92.47 %
Epoch number 36 : accuracy =  92.30000000000001 %
Epoch number 37 : accuracy =  91.53 %
Epoch number 38 : accuracy =  89.3 %
Epoch number 39 : accuracy =  91.02 %
End
```

(iv) Make trained result as file (mnist_parameters.inp)



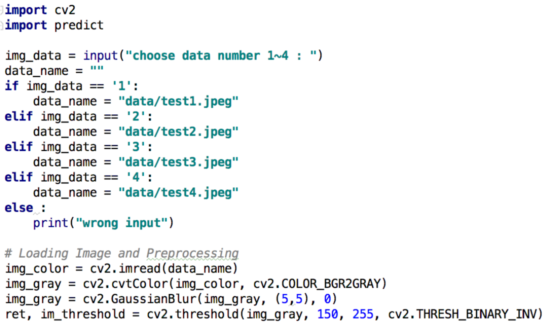
```
model = Net()
opt = optim.Adam(model.parameters(), lr=0.0001)

file_name = "mnist_data.dat"
with open(file_name, "rb") as file:
    data = pickle.load(file)

file_name = "mnist_test.dat"
with open(file_name, "rb") as file:
    test_data = pickle.load(file)


save_file = "mnist_parameters.inp"
ans = [0]
try:
    model.load_state_dict(torch.load(save_file))
    print("load saved data")
except IOError:
    torch.save(model.state_dict(), save_file)
    print("init")
    model.load_state_dict(torch.load(save_file))

for epoch in range(40):
    inp(epoch)
```

(v) Make python file which can predict image by using pretrained data (mnist_parameters.inp)



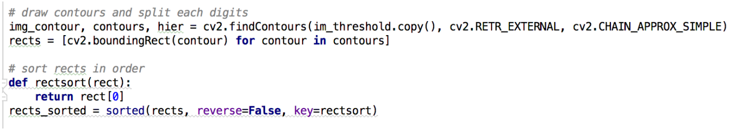
```python
import torch
import torch.nn as nn
import torch.nn.functional as ftn
from torch.autograd import Variable

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

# Predict Image by using pretrained data (mnist_parameters.inp)
def predict(input):
    model = Net()
    model.eval()
    save_file = "mnist_parameters.inp"
    try:
        model.load_state_dict(torch.load(save_file))
    except IOError:
        print("No parameters.")

    input = input / 255
    input = torch.from_numpy(input)
    input.unsqueeze_(0)
    input.unsqueeze_(0)
    input = input.float()
    input = Variable(input)

    out = model(input)
    _, pred = out.max(1)

#    print("Predicted value: " + str(pred.data[0]))
    return pred.data[0]
```

(vi) Load test image files and Preprocess image files (by using gaussianBlur and making binary image)



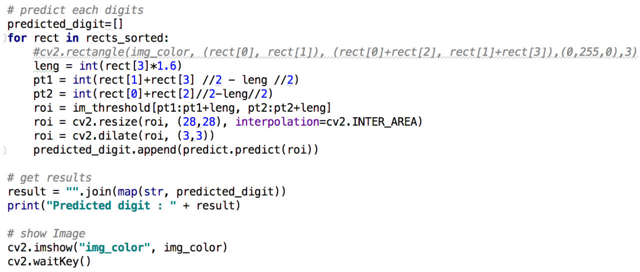
```python
import cv2
import predict

img_data = input("choose data number 1~4 : ")
data_name = ""
if img_data == '1':
    data_name = "data/test1.jpeg"
elif img_data == '2':
    data_name = "data/test2.jpeg"
elif img_data == '3':
    data_name = "data/test3.jpeg"
elif img_data == '4':
    data_name = "data/test4.jpeg"
else :
    print("wrong input")

# Loading Image and Preprocessing
img_color = cv2.imread(data_name)
img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)
img_gray = cv2.GaussianBlur(img_gray, (5,5), 0)
ret, im_threshold = cv2.threshold(img_gray, 150, 255, cv2.THRESH_BINARY_INV)
```

Draw contours and split each digits and sort each digits in order (point of x)



```python
# draw contours and split each digits
img_contour, contours, hier = cv2.findContours(im_threshold.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
rects = [cv2.boundingRect(contour) for contour in contours]

# sort rects in order
def rectsort(rect):
    return rect[0]
rects_sorted = sorted(rects, reverse=False, key=rectsort)
```

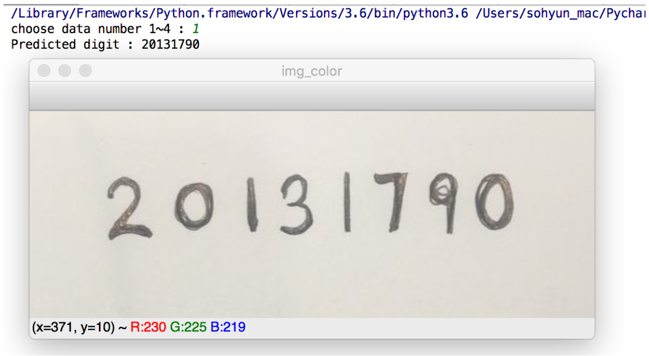
(vii) Predict each digits and Get results (+ show original Image)



```
# predict each digits
predicted_digit=[]
for rect in rects_sorted:
    #cv2.rectangle(img_color, (rect[0], rect[1]), (rect[0]+rect[2], rect[1]+rect[3]),(0,255,0),3)
    leng = int(rect[3]*1.6)
    pt1 = int(rect[1]+rect[3] //2 - leng //2)
    pt2 = int(rect[0]+rect[2]//2-leng//2)
    roi = im_threshold[pt1:pt1+leng, pt2:pt2+leng]
    roi = cv2.resize(roi, (28,28), interpolation=cv2.INTER_AREA)
    roi = cv2.dilate(roi, (3,3))
    predicted_digit.append(predict.predict(roi))

# get results
result = "".join(map(str, predicted_digit))
print("Predicted digit : " + result)

# show Image
cv2.imshow("img_color", img_color)
cv2.waitKey()
```

3 ScreenShots of result

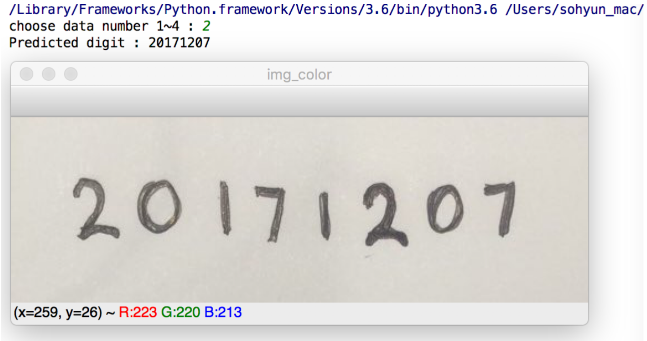
When I run this code with *test*1*.jpeg*, *test*2*.jpeg*, *test*3*.jpeg*, *test*4*.jpeg* the result looks like below.



```
/Library/Frameworks/Python.framework/Versions/3.6/bin/python3.6 /Users/sohyun_mac/Pychar
choose data number 1~4 : 1
Predicted digit : 20131790
```

(x=371, y=10) ~ R:230 G:225 B:219

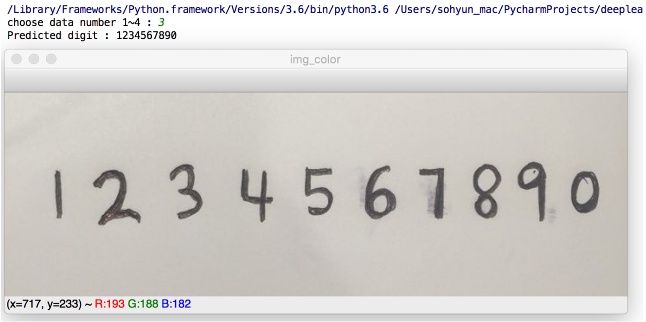
Figure 1: test1.jpeg : my student ID



```
/Library/Frameworks/Python.framework/Versions/3.6/bin/python3.6 /Users/sohyun_mac/
choose data number 1~4 : 2
Predicted digit : 20171207
```

(x=259, y=26) ~ R:223 G:220 B:213

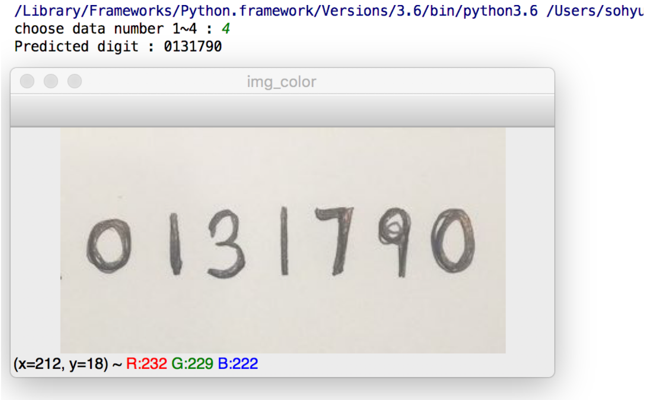
Figure 2: test2.jpeg : today's date



```
/Library/Frameworks/Python.framework/Versions/3.6/bin/python3.6 /Users/sohyun_mac/PycharmProjects/deeplea
choose data number 1~4 : 3
Predicted digit : 1234567890
```

(x=717, y=233) ~ R:193 G:188 B:182

Figure 3: test3.jpeg : all numbers



```
/Library/Frameworks/Python.framework/Versions/3.6/bin/python3.6 /Users/sohyu
choose data number 1~4 : 4
Predicted digit : 0131790
```

(x=212, y=18) ~ R:232 G:229 B:222

Figure 4: test4.jpeg : when 0 comes first