Load libraries and data

In [1]:
```
import os
import torch
import torch.nn as nn
from torch.autograd import Variable
import torchvision.datasets as dset
import torchvision.transforms as transforms
import torch.nn.functional as F
import torch.optim as optim
import PIL
import numpy as np

import matplotlib.pyplot as plt
## load mnist dataset
use_cuda = torch.cuda.is_available()

root = './data/'
if not os.path.exists(root):
    os.mkdir(root)

## added conversion for mnist to rgb
## cf. https://discuss.pytorch.org/t/grayscale-to-rgb-transform/18315/5
trans = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (1.0,)), transforms.Lambda(lambda x: x.repeat(3, 1, 1) )])

width = 28
height = 28

# if not exist, download mnist dataset
train_set = dset.MNIST(root=root, train=True, transform=trans, download=True)
test_set = dset.MNIST(root=root, train=False, transform=trans, download=True)

batch_size = 50

train_loader = torch.utils.data.DataLoader(
                 dataset=train_set,
                 batch_size=batch_size,
                 shuffle=True)
test_loader = torch.utils.data.DataLoader(
                dataset=test_set,
                batch_size=batch_size,
                shuffle=False,)

epochs = 15

print('==>>> total trainning batch number: {}'.format(len(train_loader)))
print('==>>> total testing batch number: {}'.format(len(test_loader)))
```

```
0.3%
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./data\MNIST\raw\train-images-idx3-ubyte.gz
100.0%
Extracting ./data\MNIST\raw\train-images-idx3-ubyte.gz to ./data\MNIST\raw
102.8%
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./data\MNIST\raw\train-labels-idx1-ubyte.gz
Extracting ./data\MNIST\raw\train-labels-idx1-ubyte.gz to ./data\MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
4.5%
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./data\MNIST\raw\t10k-images-idx3-ubyte.gz
100.0%
Extracting ./data\MNIST\raw\t10k-images-idx3-ubyte.gz to ./data\MNIST\raw
112.7%
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./data\MNIST\raw\t10k-labels-idx1-ubyte.gz
Extracting ./data\MNIST\raw\t10k-labels-idx1-ubyte.gz to ./data\MNIST\raw

==>>> total trainning batch number: 1200
==>>> total testing batch number: 200
```

In [2]:
```
print(use_cuda)
```

```
False
```

define lenet

In [3]:
```
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5, 1)
        self.conv2 = nn.Conv2d(20, 50, 5, 1)
        self.fc1 = nn.Linear(4*4*50, 500)
        self.fc2 = nn.Linear(500, 10)

    def forward(self, x):
        x = x.reshape(-1,3,28,28)
        x = x[:,0,:,:]
        x = x.reshape(-1,1,28,28)
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 4*4*50)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

    def name(self):
        return "LeNet"
```

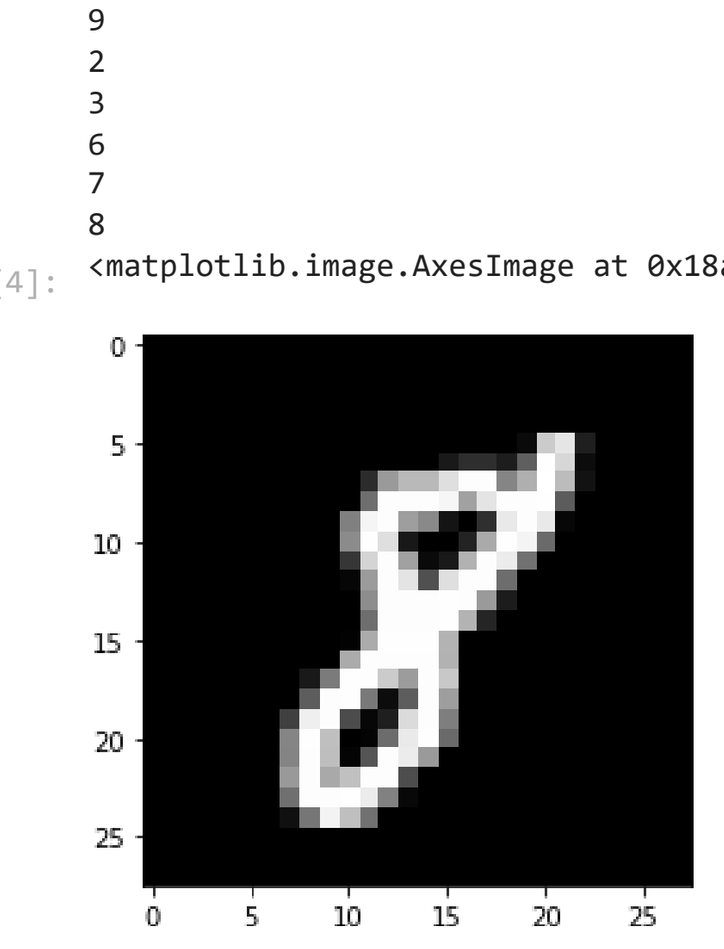
Export rgb images from the dataset for testing here and on khadas

In [4]:
```
numbers = []
for i in range(len(train_set)):
    image, label = train_set[i]
    if label not in numbers:
        print(label)
        # denormalize
        imag=(np.array(image.tolist())+0.5) * 255
        #print(imag.shape)
        # shape image from CHW -> HWC
        imag = np.ascontiguousarray( imag.transpose((1,2,0)), dtype=np.uint8)
        #print(imag.shape)
        #print(imag.astype(np.uint8))
        pil_image = PIL.Image.frombytes('RGB',(28,28), imag)
        pil_image.save(str(label)+".bmp")
        numbers.append(label)


# visualize the last image as example
plt.imshow(imag)
```

```
5
0
4
1
9
2
3
6
7
8
```

Out[4]:
```
<matplotlib.image.AxesImage at 0x18a482253d0>
```



Train lenet

In [5]:
```
## training
model = LeNet()

if use_cuda:
    model = model.cuda()

optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

criterion = nn.CrossEntropyLoss()

for epoch in range(epochs):
    # trainning
    ave_loss = 0
    for batch_idx, (x, target) in enumerate(train_loader):
        optimizer.zero_grad()

        if use_cuda:
            x, target = x.cuda(), target.cuda()
        else:
            x, target = Variable(x), Variable(target)

        out = model(x)
        loss = criterion(out, target)
        ave_loss = ave_loss * 0.9 + loss.item() * 0.1
        loss.backward()
        optimizer.step()
        if (batch_idx+1) % 100 == 0 or (batch_idx+1) == len(train_loader):
            print('==>>> epoch: {}, batch index: {}, train loss: {:.6f}'.format(
                epoch, batch_idx+1, ave_loss))
    # testing
    correct_cnt, ave_loss = 0, 0
    total_cnt = 0
    for batch_idx, (x, target) in enumerate(test_loader):
        with torch.no_grad():
            if use_cuda:
                x, target = x.cuda(), target.cuda()
            else:
                x, target = Variable(x), Variable(target)
        out = model(x)
        loss = criterion(out, target)
        _, pred_label = torch.max(out.data, 1)
        total_cnt += x.data.size()[0]
        correct_cnt += (pred_label == target.data).sum()
        # smooth average
        ave_loss = ave_loss * 0.9 + loss.item() * 0.1

        if(batch_idx+1) % 100 == 0 or (batch_idx+1) == len(test_loader):
            print('==>>> epoch: {}, batch index: {}, test loss: {:.6f}, acc: {:.3f}'.format(
                epoch, batch_idx+1, ave_loss, correct_cnt.item() * 1.0 / total_cnt))
```

```
==>>> epoch: 0, batch index: 100, train loss: 0.692832
==>>> epoch: 0, batch index: 200, train loss: 0.244985
==>>> epoch: 0, batch index: 300, train loss: 0.176380
==>>> epoch: 0, batch index: 400, train loss: 0.169317
==>>> epoch: 0, batch index: 500, train loss: 0.108723
==>>> epoch: 0, batch index: 600, train loss: 0.109910
==>>> epoch: 0, batch index: 700, train loss: 0.102145
==>>> epoch: 0, batch index: 800, train loss: 0.075479
==>>> epoch: 0, batch index: 900, train loss: 0.065534
==>>> epoch: 0, batch index: 1000, train loss: 0.105046
==>>> epoch: 0, batch index: 1100, train loss: 0.076107
==>>> epoch: 0, batch index: 1200, train loss: 0.049476
==>>> epoch: 0, batch index: 100, test loss: 0.080764, acc: 0.975
==>>> epoch: 0, batch index: 200, test loss: 0.067783, acc: 0.982
==>>> epoch: 1, batch index: 100, train loss: 0.044717
==>>> epoch: 1, batch index: 200, train loss: 0.061548
==>>> epoch: 1, batch index: 300, train loss: 0.037922
==>>> epoch: 1, batch index: 400, train loss: 0.061890
==>>> epoch: 1, batch index: 500, train loss: 0.068256
==>>> epoch: 1, batch index: 600, train loss: 0.045414
==>>> epoch: 1, batch index: 700, train loss: 0.052347
==>>> epoch: 1, batch index: 800, train loss: 0.071089
==>>> epoch: 1, batch index: 900, train loss: 0.056285
==>>> epoch: 1, batch index: 1000, train loss: 0.063602
==>>> epoch: 1, batch index: 1100, train loss: 0.050540
==>>> epoch: 1, batch index: 1200, train loss: 0.050442
==>>> epoch: 1, batch index: 100, test loss: 0.041789, acc: 0.986
==>>> epoch: 1, batch index: 200, test loss: 0.034787, acc: 0.990
==>>> epoch: 2, batch index: 100, train loss: 0.027779
==>>> epoch: 2, batch index: 200, train loss: 0.033519
==>>> epoch: 2, batch index: 300, train loss: 0.071477
==>>> epoch: 2, batch index: 400, train loss: 0.026952
==>>> epoch: 2, batch index: 500, train loss: 0.031233
==>>> epoch: 2, batch index: 600, train loss: 0.027247
==>>> epoch: 2, batch index: 700, train loss: 0.034233
==>>> epoch: 2, batch index: 800, train loss: 0.047066
==>>> epoch: 2, batch index: 900, train loss: 0.035486
==>>> epoch: 2, batch index: 1000, train loss: 0.029059
==>>> epoch: 2, batch index: 1100, train loss: 0.031663
==>>> epoch: 2, batch index: 1200, train loss: 0.038137
==>>> epoch: 2, batch index: 100, test loss: 0.035306, acc: 0.983
==>>> epoch: 2, batch index: 200, test loss: 0.043023, acc: 0.988
==>>> epoch: 3, batch index: 100, train loss: 0.016539
==>>> epoch: 3, batch index: 200, train loss: 0.020007
==>>> epoch: 3, batch index: 300, train loss: 0.012577
==>>> epoch: 3, batch index: 400, train loss: 0.037696
==>>> epoch: 3, batch index: 500, train loss: 0.022093
==>>> epoch: 3, batch index: 600, train loss: 0.028688
==>>> epoch: 3, batch index: 700, train loss: 0.026935
==>>> epoch: 3, batch index: 800, train loss: 0.023693
==>>> epoch: 3, batch index: 900, train loss: 0.013840
==>>> epoch: 3, batch index: 1000, train loss: 0.042766
==>>> epoch: 3, batch index: 1100, train loss: 0.032594
==>>> epoch: 3, batch index: 1200, train loss: 0.033977
==>>> epoch: 3, batch index: 100, test loss: 0.030136, acc: 0.988
==>>> epoch: 3, batch index: 200, test loss: 0.026095, acc: 0.992
==>>> epoch: 4, batch index: 100, train loss: 0.030209
==>>> epoch: 4, batch index: 200, train loss: 0.016743
==>>> epoch: 4, batch index: 300, train loss: 0.023052
==>>> epoch: 4, batch index: 400, train loss: 0.023972
==>>> epoch: 4, batch index: 500, train loss: 0.018334
==>>> epoch: 4, batch index: 600, train loss: 0.027314
==>>> epoch: 4, batch index: 700, train loss: 0.020631
==>>> epoch: 4, batch index: 800, train loss: 0.018747
==>>> epoch: 4, batch index: 900, train loss: 0.012305
==>>> epoch: 4, batch index: 1000, train loss: 0.019247
==>>> epoch: 4, batch index: 1100, train loss: 0.025061
==>>> epoch: 4, batch index: 1200, train loss: 0.026796
==>>> epoch: 4, batch index: 100, test loss: 0.036315, acc: 0.985
==>>> epoch: 4, batch index: 200, test loss: 0.022533, acc: 0.991
==>>> epoch: 5, batch index: 100, train loss: 0.009922
==>>> epoch: 5, batch index: 200, train loss: 0.016705
==>>> epoch: 5, batch index: 300, train loss: 0.014992
==>>> epoch: 5, batch index: 400, train loss: 0.012316
==>>> epoch: 5, batch index: 500, train loss: 0.023518
==>>> epoch: 5, batch index: 600, train loss: 0.018592
==>>> epoch: 5, batch index: 700, train loss: 0.022243
==>>> epoch: 5, batch index: 800, train loss: 0.025662
==>>> epoch: 5, batch index: 900, train loss: 0.012951
==>>> epoch: 5, batch index: 1000, train loss: 0.016556
==>>> epoch: 5, batch index: 1100, train loss: 0.018877
==>>> epoch: 5, batch index: 1200, train loss: 0.036122
==>>> epoch: 5, batch index: 100, test loss: 0.021460, acc: 0.989
==>>> epoch: 5, batch index: 200, test loss: 0.015569, acc: 0.993
==>>> epoch: 6, batch index: 100, train loss: 0.014586
==>>> epoch: 6, batch index: 200, train loss: 0.017695
==>>> epoch: 6, batch index: 300, train loss: 0.010129
==>>> epoch: 6, batch index: 400, train loss: 0.011107
==>>> epoch: 6, batch index: 500, train loss: 0.013350
==>>> epoch: 6, batch index: 600, train loss: 0.020766
==>>> epoch: 6, batch index: 700, train loss: 0.007682
==>>> epoch: 6, batch index: 800, train loss: 0.010922
==>>> epoch: 6, batch index: 900, train loss: 0.005340
==>>> epoch: 6, batch index: 1000, train loss: 0.010726
==>>> epoch: 6, batch index: 1100, train loss: 0.018290
==>>> epoch: 6, batch index: 1200, train loss: 0.009744
==>>> epoch: 6, batch index: 100, test loss: 0.018391, acc: 0.990
==>>> epoch: 6, batch index: 200, test loss: 0.023691, acc: 0.992
==>>> epoch: 7, batch index: 100, train loss: 0.005870
==>>> epoch: 7, batch index: 200, train loss: 0.004290
==>>> epoch: 7, batch index: 300, train loss: 0.003639
==>>> epoch: 7, batch index: 400, train loss: 0.015330
==>>> epoch: 7, batch index: 500, train loss: 0.021944
==>>> epoch: 7, batch index: 600, train loss: 0.007665
==>>> epoch: 7, batch index: 700, train loss: 0.017832
==>>> epoch: 7, batch index: 800, train loss: 0.012149
==>>> epoch: 7, batch index: 900, train loss: 0.013120
==>>> epoch: 7, batch index: 1000, train loss: 0.010922
==>>> epoch: 7, batch index: 1100, train loss: 0.016828
==>>> epoch: 7, batch index: 1200, train loss: 0.016702
==>>> epoch: 7, batch index: 100, test loss: 0.029370, acc: 0.987
==>>> epoch: 7, batch index: 200, test loss: 0.017990, acc: 0.992
==>>> epoch: 8, batch index: 100, train loss: 0.007684
==>>> epoch: 8, batch index: 200, train loss: 0.006316
==>>> epoch: 8, batch index: 300, train loss: 0.003383
==>>> epoch: 8, batch index: 400, train loss: 0.004378
==>>> epoch: 8, batch index: 500, train loss: 0.004669
==>>> epoch: 8, batch index: 600, train loss: 0.005213
==>>> epoch: 8, batch index: 700, train loss: 0.007496
==>>> epoch: 8, batch index: 800, train loss: 0.009835
==>>> epoch: 8, batch index: 900, train loss: 0.005950
==>>> epoch: 8, batch index: 1000, train loss: 0.014239
==>>> epoch: 8, batch index: 1100, train loss: 0.012707
==>>> epoch: 8, batch index: 1200, train loss: 0.004230
==>>> epoch: 8, batch index: 100, test loss: 0.021762, acc: 0.988
==>>> epoch: 8, batch index: 200, test loss: 0.021383, acc: 0.992
==>>> epoch: 9, batch index: 100, train loss: 0.023725
==>>> epoch: 9, batch index: 200, train loss: 0.003537
==>>> epoch: 9, batch index: 300, train loss: 0.004308
==>>> epoch: 9, batch index: 400, train loss: 0.002626
==>>> epoch: 9, batch index: 500, train loss: 0.004971
==>>> epoch: 9, batch index: 600, train loss: 0.006806
==>>> epoch: 9, batch index: 700, train loss: 0.003093
==>>> epoch: 9, batch index: 800, train loss: 0.002064
==>>> epoch: 9, batch index: 900, train loss: 0.001606
==>>> epoch: 9, batch index: 1000, train loss: 0.018593
==>>> epoch: 9, batch index: 1100, train loss: 0.015799
==>>> epoch: 9, batch index: 1200, train loss: 0.006154
==>>> epoch: 9, batch index: 100, test loss: 0.032166, acc: 0.988
==>>> epoch: 9, batch index: 200, test loss: 0.018751, acc: 0.992
==>>> epoch: 10, batch index: 100, train loss: 0.005070
==>>> epoch: 10, batch index: 200, train loss: 0.006071
==>>> epoch: 10, batch index: 300, train loss: 0.003327
==>>> epoch: 10, batch index: 400, train loss: 0.002622
==>>> epoch: 10, batch index: 500, train loss: 0.005446
==>>> epoch: 10, batch index: 600, train loss: 0.004339
==>>> epoch: 10, batch index: 700, train loss: 0.006930
==>>> epoch: 10, batch index: 800, train loss: 0.003691
==>>> epoch: 10, batch index: 900, train loss: 0.013034
==>>> epoch: 10, batch index: 1000, train loss: 0.005052
==>>> epoch: 10, batch index: 1100, train loss: 0.004451
==>>> epoch: 10, batch index: 1200, train loss: 0.003483
==>>> epoch: 10, batch index: 100, test loss: 0.017463, acc: 0.988
==>>> epoch: 10, batch index: 200, test loss: 0.022378, acc: 0.992
==>>> epoch: 11, batch index: 100, train loss: 0.002103
==>>> epoch: 11, batch index: 200, train loss: 0.008607
==>>> epoch: 11, batch index: 300, train loss: 0.001407
==>>> epoch: 11, batch index: 400, train loss: 0.006857
==>>> epoch: 11, batch index: 500, train loss: 0.001883
==>>> epoch: 11, batch index: 600, train loss: 0.006338
==>>> epoch: 11, batch index: 700, train loss: 0.005154
==>>> epoch: 11, batch index: 800, train loss: 0.007621
==>>> epoch: 11, batch index: 900, train loss: 0.002136
==>>> epoch: 11, batch index: 1000, train loss: 0.001887
==>>> epoch: 11, batch index: 1100, train loss: 0.002976
==>>> epoch: 11, batch index: 1200, train loss: 0.003714
==>>> epoch: 11, batch index: 100, test loss: 0.022718, acc: 0.988
==>>> epoch: 11, batch index: 200, test loss: 0.023327, acc: 0.992
==>>> epoch: 12, batch index: 100, train loss: 0.001266
==>>> epoch: 12, batch index: 200, train loss: 0.001311
==>>> epoch: 12, batch index: 300, train loss: 0.002648
==>>> epoch: 12, batch index: 400, train loss: 0.000990
==>>> epoch: 12, batch index: 500, train loss: 0.003779
==>>> epoch: 12, batch index: 600, train loss: 0.001753
==>>> epoch: 12, batch index: 700, train loss: 0.000731
==>>> epoch: 12, batch index: 800, train loss: 0.001258
==>>> epoch: 12, batch index: 900, train loss: 0.004582
==>>> epoch: 12, batch index: 1000, train loss: 0.003587
==>>> epoch: 12, batch index: 1100, train loss: 0.000962
==>>> epoch: 12, batch index: 1200, train loss: 0.002078
==>>> epoch: 12, batch index: 100, test loss: 0.028338, acc: 0.988
==>>> epoch: 12, batch index: 200, test loss: 0.018723, acc: 0.992
==>>> epoch: 13, batch index: 100, train loss: 0.000867
==>>> epoch: 13, batch index: 200, train loss: 0.000851
==>>> epoch: 13, batch index: 300, train loss: 0.000608
==>>> epoch: 13, batch index: 400, train loss: 0.003210
==>>> epoch: 13, batch index: 500, train loss: 0.001383
==>>> epoch: 13, batch index: 600, train loss: 0.003188
==>>> epoch: 13, batch index: 700, train loss: 0.001968
==>>> epoch: 13, batch index: 800, train loss: 0.002006
==>>> epoch: 13, batch index: 900, train loss: 0.003140
==>>> epoch: 13, batch index: 1000, train loss: 0.002594
==>>> epoch: 13, batch index: 1100, train loss: 0.001831
==>>> epoch: 13, batch index: 1200, train loss: 0.000941
==>>> epoch: 13, batch index: 100, test loss: 0.024532, acc: 0.988
==>>> epoch: 13, batch index: 200, test loss: 0.016675, acc: 0.993
==>>> epoch: 14, batch index: 100, train loss: 0.006181
==>>> epoch: 14, batch index: 200, train loss: 0.002198
==>>> epoch: 14, batch index: 300, train loss: 0.002840
==>>> epoch: 14, batch index: 400, train loss: 0.000727
==>>> epoch: 14, batch index: 500, train loss: 0.001912
==>>> epoch: 14, batch index: 600, train loss: 0.006814
==>>> epoch: 14, batch index: 700, train loss: 0.001083
==>>> epoch: 14, batch index: 800, train loss: 0.001439
==>>> epoch: 14, batch index: 900, train loss: 0.000572
==>>> epoch: 14, batch index: 1000, train loss: 0.000980
==>>> epoch: 14, batch index: 1100, train loss: 0.001047
==>>> epoch: 14, batch index: 1200, train loss: 0.001458
==>>> epoch: 14, batch index: 100, test loss: 0.022703, acc: 0.988
==>>> epoch: 14, batch index: 200, test loss: 0.012982, acc: 0.993
```

Save trained lenet as pyTorch native format

In [6]:
```
torch.save(model, "lenet.pt")
```

Predict using the stored model and previously exported example images

In [7]:
```
import cv2 as cv

## conv function represent different ways to implement the de-normalization
def conv_fn1(data):
    r_data = ((np.array(data) / 255.) - 0.5) / 1.0
    return r_data


def conv_fn2(data):
    r_data = ((np.array(data) - 127.5) / 255.)
    return r_data



def test_acc(model, conv_fn, verb=False, transpose=True):
    for i in range(0,9):
        im = cv.imread("%s.bmp"%i, cv.IMREAD_COLOR)
        img = conv_fn(im)
        if transpose: # HWC -> CHW
            img = img.transpose((2,0,1))
        img_tensor = torch.from_numpy(img).reshape(1,3,28,28).float()
        if use_cuda:
            model = model.cuda()
            img_tensor = img_tensor.cuda()
        out = model(img_tensor)
        res = np.argmax( out.tolist()[0] )
        print("Image is classified as %i -> should be %s"%(res, i))
        if verb:
            print(np.array(out.tolist()[0]) * 127.5 + 127.5)



# predict using test image using opencv (like we would do on khadas)
# Load model
l_model = torch.load("lenet.pt")

test_acc(l_model, conv_fn1, verb=False)
```

```
Image is classified as 0 -> should be 0
Image is classified as 1 -> should be 1
Image is classified as 2 -> should be 2
Image is classified as 3 -> should be 3
Image is classified as 4 -> should be 4
Image is classified as 5 -> should be 5
Image is classified as 6 -> should be 6
Image is classified as 7 -> should be 7
Image is classified as 8 -> should be 8
```

export lmodel to onnx -> check output of export very carefully; dont miss any error

In [8]:
```
def export_model_to_onnx(model):
    input_dimension = torch.randn(1, 3, 28, 28)
    if use_cuda:
        input_dimension = input_dimension.cuda()
    # very important or must leave out - not sure need to test again...
    traced = torch.jit.trace(model, input_dimension)

    torch.onnx.export(
                    traced,
                    input_dimension,
                    "lenet.onnx",
                    opset_version=7,
                    verbose=True,
                    export_params=True,
                    input_names=['input'],
                    output_names=['output'],
                    dynamic_axes=None
                    #dynamic_axes={'input': {0: 'batch', 2: 'height', 3: 'width'},  # shape(1,1,28,28)
                    #              'output': {0: 'batch', 1: 'classes'} }  # shape(1,10)
                    #example_outputs = torch_out
    )

export_model_to_onnx(l_model)
```

```
D:\Downloads\novelsense\example-network\1_create_and_export_network\venv\lib\site-packages\torch\onnx\utils.py:359: UserWarning: Model has no forward function
  warnings.warn("Model has no forward function")
graph(%input : Float(1, 3, 28, 28, strides=[2352, 784, 28, 1], requires_grad=0, device=cpu),
      %conv1.bias : Float(20, strides=[1], requires_grad=0, device=cpu),
      %conv1.weight : Float(20, 1, 5, 5, strides=[25, 25, 5, 1], requires_grad=0, device=cpu),
      %conv2.bias : Float(50, strides=[1], requires_grad=0, device=cpu),
      %conv2.weight : Float(50, 20, 5, 5, strides=[500, 25, 5, 1], requires_grad=0, device=cpu),
      %fc1.bias : Float(500, strides=[1], requires_grad=0, device=cpu),
      %fc1.weight : Float(500, 800, strides=[800, 1], requires_grad=0, device=cpu),
      %fc2.bias : Float(10, strides=[1], requires_grad=0, device=cpu),
      %fc2.weight : Float(10, 500, strides=[500, 1], requires_grad=0, device=cpu)):
  %9 : Long(4, strides=[1], device=cpu) = onnx::Constant[value= -1   3  28  28 [ CPUDoubleType{4} ]]() # C:\Users\Sajjad\AppData\Local\Temp\ipykernel_1088\836
706537.py:10:0
  %26 : Long(4, strides=[1], requires_grad=0, device=cpu) = onnx::Cast[to=7](%9) # C:\Users\Sajjad\AppData\Local\Temp\ipykernel_1088\836706537.py:10:0
  %x : Float(1, 3, 28, 28, strides=[2352, 784, 28, 1], device=cpu) = onnx::Reshape(%input, %26) # C:\Users\Sajjad\AppData\Local\Temp\ipykernel_1088\836706537.
py:10:0
  %11 : Long(device=cpu) = onnx::Constant[value={0}]() # C:\Users\Sajjad\AppData\Local\Temp\ipykernel_1088\836706537.py:11:0
  %27 : Long(requires_grad=0, device=cpu) = onnx::Cast[to=7](%11) # C:\Users\Sajjad\AppData\Local\Temp\ipykernel_1088\836706537.py:11:0
  %12 : Float(1, 28, 28, strides=[784, 28, 1], device=cpu) = onnx::Gather[axis=1](%x, %27) # C:\Users\Sajjad\AppData\Local\Temp\ipykernel_1088\836706537.py:1
1:0
  %13 : Long(4, strides=[1], device=cpu) = onnx::Constant[value= -1   1  28  28 [ CPUDoubleType{4} ]]() # C:\Users\Sajjad\AppData\Local\Temp\ipykernel_1088\83
6706537.py:12:0
  %28 : Long(4, strides=[1], requires_grad=0, device=cpu) = onnx::Cast[to=7](%13) # C:\Users\Sajjad\AppData\Local\Temp\ipykernel_1088\836706537.py:12:0
  %input.1 : Float(1, 1, 28, 28, strides=[784, 784, 28, 1], device=cpu) = onnx::Reshape(%12, %28) # C:\Users\Sajjad\AppData\Local\Temp\ipykernel_1088\83670653
7.py:12:0
  %input.3 : Float(1, 20, 24, 24, strides=[11520, 576, 24, 1], device=cpu) = onnx::Conv[dilations=[1, 1], group=1, kernel_shape=[5, 5], pads=[0, 0, 0, 0], str
ides=[1, 1]](%input.1, %conv1.weight, %conv1.bias), scope: __module.conv1 # D:\Downloads\novelsense\example-network\1_create_and_export_network\venv\lib\site-
packages\torch\nn\modules\conv.py:443:0
  %input.7 : Float(1, 20, 24, 24, strides=[11520, 576, 24, 1], device=cpu) = onnx::Relu(%input.3) # D:\Downloads\novelsense\example-network\1_create_and_expor
t_network\venv\lib\site-packages\torch\nn\functional.py:1442:0
  %input.11 : Float(1, 20, 12, 12, strides=[2880, 144, 12, 1], device=cpu) = onnx::MaxPool[kernel_shape=[2, 2], pads=[0, 0, 0, 0], strides=[2, 2]](%input.7) #
D:\Downloads\novelsense\example-network\1_create_and_export_network\venv\lib\site-packages\torch\nn\functional.py:797:0
  %input.15 : Float(1, 50, 8, 8, strides=[3200, 64, 8, 1], device=cpu) = onnx::Conv[dilations=[1, 1], group=1, kernel_shape=[5, 5], pads=[0, 0, 0, 0], strides
=[1, 1]](%input.11, %conv2.weight, %conv2.bias), scope: __module.conv2 # D:\Downloads\novelsense\example-network\1_create_and_export_network\venv\lib\site-pac
kages\torch\nn\modules\conv.py:443:0
  %input.19 : Float(1, 50, 8, 8, strides=[3200, 64, 8, 1], device=cpu) = onnx::Relu(%input.15) # D:\Downloads\novelsense\example-network\1_create_and_export_n
etwork\venv\lib\site-packages\torch\nn\functional.py:1442:0
  %x.4 : Float(1, 50, 4, 4, strides=[800, 16, 4, 1], device=cpu) = onnx::MaxPool[kernel_shape=[2, 2], pads=[0, 0, 0, 0], strides=[2, 2]](%input.19) # D:\Downl
oads\novelsense\example-network\1_create_and_export_network\venv\lib\site-packages\torch\nn\functional.py:797:0
  %21 : Long(2, strides=[1], device=cpu) = onnx::Constant[value=  -1  800 [ CPUDoubleType{2} ]]() # C:\Users\Sajjad\AppData\Local\Temp\ipykernel_1088\83670653
7.py:17:0
  %29 : Long(2, strides=[1], requires_grad=0, device=cpu) = onnx::Cast[to=7](%21) # C:\Users\Sajjad\AppData\Local\Temp\ipykernel_1088\836706537.py:17:0
  %input.23 : Float(1, 800, strides=[800, 1], device=cpu) = onnx::Reshape(%x.4, %29) # C:\Users\Sajjad\AppData\Local\Temp\ipykernel_1088\836706537.py:17:0
  %input.27 : Float(1, 500, strides=[500, 1], device=cpu) = onnx::Gemm[alpha=1., beta=1., transB=1](%input.23, %fc1.weight, %fc1.bias), scope: __module.fc1 #
D:\Downloads\novelsense\example-network\1_create_and_export_network\venv\lib\site-packages\torch\nn\modules\linear.py:103:0
  %input.31 : Float(1, 500, strides=[500, 1], device=cpu) = onnx::Relu(%input.27) # D:\Downloads\novelsense\example-network\1_create_and_export_network\venv\l
ib\site-packages\torch\nn\functional.py:1442:0
  %output : Float(1, 10, strides=[10, 1], requires_grad=1, device=cpu) = onnx::Gemm[alpha=1., beta=1., transB=1](%input.31, %fc2.weight, %fc2.bias), scope: __
module.fc2 # D:\Downloads\novelsense\example-network\1_create_and_export_network\venv\lib\site-packages\torch\nn\modules\linear.py:103:0
  return (%output)
```

check onnx

In [9]:
```
import onnx

# Load the ONNX model
onnx_model = onnx.load("lenet.onnx")

# Check that the IR is well formed
onnx.checker.check_model(onnx_model)

# Print a Human readable representation of the graph
print( onnx.helper.printable_graph(onnx_model.graph) )
```

```
graph torch-jit-export (
  %input[FLOAT, 1x3x28x28]
) optional inputs with matching initializers (
  %conv1.bias[FLOAT, 20]
  %conv1.weight[FLOAT, 20x1x5x5]
  %conv2.bias[FLOAT, 50]
  %conv2.weight[FLOAT, 50x20x5x5]
  %fc1.bias[FLOAT, 500]
  %fc1.weight[FLOAT, 500x800]
  %fc2.bias[FLOAT, 10]
  %fc2.weight[FLOAT, 10x500]
) {
  %onnx::Cast_9 = Constant[value = <Tensor>]()
  %onnx::Reshape_26 = Cast[to = 7](%onnx::Cast_9)
  %x = Reshape(%input, %onnx::Reshape_26)
  %onnx::Cast_11 = Constant[value = <Scalar Tensor []>]()
  %onnx::Gather_27 = Cast[to = 7](%onnx::Cast_11)
  %onnx::Reshape_12 = Gather[axis = 1](%x, %onnx::Gather_27)
  %onnx::Cast_13 = Constant[value = <Tensor>]()
  %onnx::Reshape_28 = Cast[to = 7](%onnx::Cast_13)
  %input.1 = Reshape(%onnx::Reshape_12, %onnx::Reshape_28)
  %input.3 = Conv[dilations = [1, 1], group = 1, kernel_shape = [5, 5], pads = [0, 0, 0, 0], strides = [1, 1]](%input.1, %conv1.weight, %conv1.bias)
  %input.7 = Relu(%input.3)
  %input.11 = MaxPool[kernel_shape = [2, 2], pads = [0, 0, 0, 0], strides = [2, 2]](%input.7)
  %input.15 = Conv[dilations = [1, 1], group = 1, kernel_shape = [5, 5], pads = [0, 0, 0, 0], strides = [1, 1]](%input.11, %conv2.weight, %conv2.bias)
  %input.19 = Relu(%input.15)
  %x.4 = MaxPool[kernel_shape = [2, 2], pads = [0, 0, 0, 0], strides = [2, 2]](%input.19)
  %onnx::Cast_21 = Constant[value = <Tensor>]()
  %onnx::Reshape_29 = Cast[to = 7](%onnx::Cast_21)
  %input.23 = Reshape(%x.4, %onnx::Reshape_29)
  %input.27 = Gemm[alpha = 1, beta = 1, transB = 1](%input.23, %fc1.weight, %fc1.bias)
  %input.31 = Relu(%input.27)
  %output = Gemm[alpha = 1, beta = 1, transB = 1](%input.31, %fc2.weight, %fc2.bias)
  return %output
}
```