q3

April 3, 2020

0.1 Question 3 - MNIST Classsification using PyTorch

[0]:
```
import torch
import numpy as np
```

Load Data Sets

[2]:
```
from torchvision import datasets
import torchvision.transforms as transforms

# number of subprocesses to use for data loading
num_workers = 0
# how many samples per batch to load
batch_size = 20

# convert data to torch.FloatTensor
transform = transforms.ToTensor()

# choose the training and test datasets
train_data = datasets.MNIST(root='data', train=True,
                                 download=True, transform=transform)
test_data = datasets.MNIST(root='data', train=False,
                                download=True, transform=transform)

# prepare data loaders
train_loader = torch.utils.data.DataLoader(train_data, batch_size=batch_size,
    num_workers=num_workers)
test_loader = torch.utils.data.DataLoader(test_data, batch_size=batch_size,
    num_workers=num_workers)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to
data/MNIST/raw/train-images-idx3-ubyte.gz
```

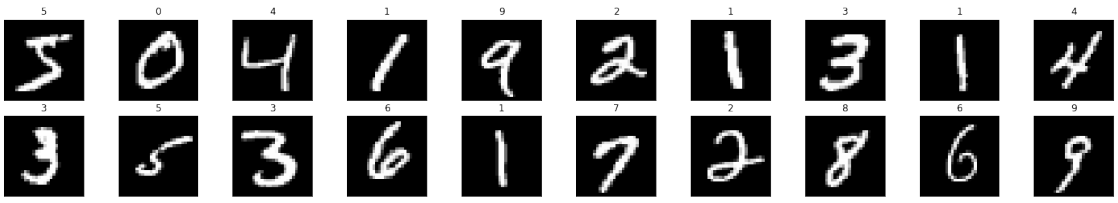
```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

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

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


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 to
data/MNIST/raw/t10k-images-idx3-ubyte.gz

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


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

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Extracting data/MNIST/raw/t10k-labels-idx1-ubyte.gz to data/MNIST/raw
Processing…
Done!
```

Visualize a batch of Training Set

[3]:
```
import matplotlib.pyplot as plt
%matplotlib inline

# obtain one batch of training images
dataiter = iter(train_loader)
images, labels = dataiter.next()
images = images.numpy()

# plot the images in the batch, along with the corresponding labels
fig = plt.figure(figsize=(25, 4))
for idx in np.arange(20):
    ax = fig.add_subplot(2, 20/2, idx+1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(images[idx]), cmap='gray')
    # print out the correct label for each image
    # .item() gets the value contained in a Tensor
    ax.set_title(str(labels[idx].item()))
```



Define the Network Architecture

[0]:
```
import torch.nn as nn
import torch.nn.functional as F

## Define the NN architecture
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 512)
        # linear layer (n_hidden -> hidden_2)
        self.fc2 = nn.Linear(512, 512)
        # linear layer (n_hidden -> 10)
        self.fc3 = nn.Linear(512, 10)
        # dropout layer (p=0.2)
        # dropout prevents overfitting of data
        self.dropout = nn.Dropout(0.2)

    def forward(self, x):
        # flatten image input
        x = x.view(-1, 28 * 28)
        # add hidden layer, with relu activation function
        x = F.relu(self.fc1(x))
        return x
```

[5]:
```
# initialize the NN
model = Net()
print(model)
```

```
Net(
  (fc1): Linear(in_features=784, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=512, bias=True)
  (fc3): Linear(in_features=512, out_features=10, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

Specify Loss Function and Optimizer

[0]:
```
criterion = nn.CrossEntropyLoss()

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

Train the Network

[7]:
```
n_epochs = 30  # suggest training between 20-50 epochs

model.train() # prep model for training

for epoch in range(n_epochs):
    # monitor training loss
    train_loss = 0.0

    ###################
    # train the model #
    ###################
    for data, target in train_loader:
        # clear the gradients of all optimized variables
        optimizer.zero_grad()
        # forward pass: compute predicted outputs by passing inputs to the model
        output = model(data)
        # calculate the loss
        loss = criterion(output, target)
        # backward pass: compute gradient of the loss with respect to model␣
↪parameters
        loss.backward()
        # perform a single optimization step (parameter update)
        optimizer.step()
        # update running training loss
        train_loss += loss.item()*data.size(0)

    # print training statistics
    # calculate average loss over an epoch
    train_loss = train_loss/len(train_loader.dataset)

    print('Epoch: {} \tTraining Loss: {:.6f}'.format(
        epoch+1,
        train_loss
        ))
```

```
Epoch: 1        Training Loss: 0.804222
Epoch: 2        Training Loss: 0.411707
Epoch: 3        Training Loss: 0.370065
Epoch: 4        Training Loss: 0.348959
Epoch: 5        Training Loss: 0.335502
Epoch: 6        Training Loss: 0.325918
Epoch: 7        Training Loss: 0.318626
Epoch: 8        Training Loss: 0.312826
Epoch: 9        Training Loss: 0.308064
Epoch: 10       Training Loss: 0.304058
Epoch: 11       Training Loss: 0.300624
Epoch: 12       Training Loss: 0.297635
```

```
Epoch: 13       Training Loss: 0.294999
Epoch: 14       Training Loss: 0.292651
Epoch: 15       Training Loss: 0.290539
Epoch: 16       Training Loss: 0.288626
Epoch: 17       Training Loss: 0.286881
Epoch: 18       Training Loss: 0.285279
Epoch: 19       Training Loss: 0.283802
Epoch: 20       Training Loss: 0.282434
Epoch: 21       Training Loss: 0.281160
Epoch: 22       Training Loss: 0.279970
Epoch: 23       Training Loss: 0.278856
Epoch: 24       Training Loss: 0.277808
Epoch: 25       Training Loss: 0.276820
Epoch: 26       Training Loss: 0.275887
Epoch: 27       Training Loss: 0.275002
Epoch: 28       Training Loss: 0.274163
Epoch: 29       Training Loss: 0.273365
Epoch: 30       Training Loss: 0.272604
```

Test the Trained Network¶

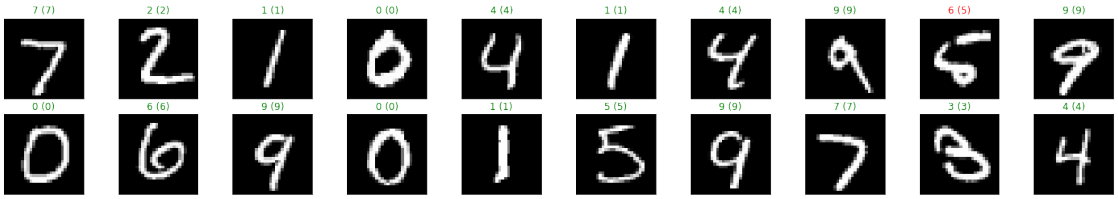
[8]:
```
test_loss = 0.0
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))

model.eval() # prep model for *evaluation*

for data, target in test_loader:
    # forward pass: compute predicted outputs by passing inputs to the model
    output = model(data)
    # calculate the loss
    loss = criterion(output, target)
    # update test loss
    test_loss += loss.item()*data.size(0)
    # convert output probabilities to predicted class
    _, pred = torch.max(output, 1)
    # compare predictions to true label
    correct = np.squeeze(pred.eq(target.data.view_as(pred)))
    # calculate test accuracy for each object class
    for i in range(batch_size):
        label = target.data[i]
        class_correct[label] += correct[i].item()
        class_total[label] += 1

# calculate and print avg test loss
test_loss = test_loss/len(test_loader.dataset)
print('Test Loss: {:.6f}\n'.format(test_loss))
```

```
for i in range(10):
    if class_total[i] > 0:
        print('Test Accuracy of %5s: %2d%% (%2d/%2d)' % (
            str(i), 100 * class_correct[i] / class_total[i],
            np.sum(class_correct[i]), np.sum(class_total[i])))
    else:
        print('Test Accuracy of %5s: N/A (no training examples)' % (classes[i]))

print('\nTest Accuracy (Overall): %2d%% (%2d/%2d)' % (
    100. * np.sum(class_correct) / np.sum(class_total),
    np.sum(class_correct), np.sum(class_total)))
```

```
Test Loss: 0.274007

Test Accuracy of     0: 98% (962/980)
Test Accuracy of     1: 97% (1109/1135)
Test Accuracy of     2: 88% (918/1032)
Test Accuracy of     3: 90% (914/1010)
Test Accuracy of     4: 92% (909/982)
Test Accuracy of     5: 87% (781/892)
Test Accuracy of     6: 94% (910/958)
Test Accuracy of     7: 92% (946/1028)
Test Accuracy of     8: 88% (864/974)
Test Accuracy of     9: 90% (916/1009)

Test Accuracy (Overall): 92% (9229/10000)
```

Visualize Sample Test Results

[9]:
```
dataiter = iter(test_loader)
images, labels = dataiter.next()

# get sample outputs
output = model(images)
# convert output probabilities to predicted class
_, preds = torch.max(output, 1)
# prep images for display
images = images.numpy()

# plot the images in the batch, along with predicted and true labels
fig = plt.figure(figsize=(25, 4))
for idx in np.arange(20):
    ax = fig.add_subplot(2, 20/2, idx+1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(images[idx]), cmap='gray')
    ax.set_title("{} ({})".format(str(preds[idx].item()), str(labels[idx].
 ↪item())),
                 color=("green" if preds[idx]==labels[idx] else "red"))
```



0.2 MNIST Classification using CNN

[0]:
```
import numpy as np # to handle matrix and data operation
import pandas as pd # to read csv and handle dataframe

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.utils.data
from torch.autograd import Variable

from sklearn.model_selection import train_test_split
```

[0]:
```
from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

[0]:
```
x_train = x_train.reshape(60000,784)
x_test = x_test.reshape(10000,784)
```

[0]:
```
BATCH_SIZE = 32
```

[0]:
```
torch_X_train = torch.from_numpy(x_train).type(torch.LongTensor)
torch_y_train = torch.from_numpy(y_train).type(torch.LongTensor)
```

[0]:
```
torch_X_test = torch.from_numpy(x_test).type(torch.LongTensor)
torch_y_test = torch.from_numpy(y_test).type(torch.LongTensor)
```

[0]:
```
train = torch.utils.data.TensorDataset(torch_X_train,torch_y_train)
test = torch.utils.data.TensorDataset(torch_X_test,torch_y_test)
```

[0]:
```
train_loader = torch.utils.data.DataLoader(train, batch_size = BATCH_SIZE,␣
↪shuffle = False)
test_loader = torch.utils.data.DataLoader(test, batch_size = BATCH_SIZE,␣
↪shuffle = False)
```

[59]:
```
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.linear1 = nn.Linear(784,250)
        self.linear2 = nn.Linear(250,100)
        self.linear3 = nn.Linear(100,10)

    def forward(self,X):
        X = F.relu(self.linear1(X))
        X = F.relu(self.linear2(X))
        X = self.linear3(X)
        return F.log_softmax(X, dim=1)

mlp = MLP()
print(mlp)
```

```
MLP(
  (linear1): Linear(in_features=784, out_features=250, bias=True)
  (linear2): Linear(in_features=250, out_features=100, bias=True)
  (linear3): Linear(in_features=100, out_features=10, bias=True)
)
```

[0]:
```
def fit(model, train_loader):
    optimizer = torch.optim.Adam(model.parameters())#,lr=0.001, betas=(0.9,0.
 ↪999))
    error = nn.CrossEntropyLoss()
    EPOCHS = 5
    model.train()
    for epoch in range(EPOCHS):
        correct = 0
        for batch_idx, (X_batch, y_batch) in enumerate(train_loader):
            var_X_batch = Variable(X_batch).float()
            var_y_batch = Variable(y_batch)
            optimizer.zero_grad()
            output = model(var_X_batch)
            loss = error(output, var_y_batch)
            loss.backward()
            optimizer.step()

            # Total correct predictions
            predicted = torch.max(output.data, 1)[1]
            correct += (predicted == var_y_batch).sum()
            #print(correct)
            if batch_idx % 50 == 0:
                print('Epoch : {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}\t Accuracy:{:.
 ↪3f}%'.format(
```

```
                    epoch, batch_idx*len(X_batch), len(train_loader.dataset),␣
↪100.*batch_idx / len(train_loader), (loss.data), float(correct*100) /␣
↪float(BATCH_SIZE*(batch_idx+1))))
```

[67]:
```
fit(mlp, train_loader)
```

```
Epoch : 0 [0/60000 (0%)]        Loss: 0.028779   Accuracy:100.000%
Epoch : 0 [1600/60000 (3%)]     Loss: 0.135546   Accuracy:96.875%
Epoch : 0 [3200/60000 (5%)]     Loss: 0.161492   Accuracy:97.061%
Epoch : 0 [4800/60000 (8%)]     Loss: 0.029837   Accuracy:97.144%
Epoch : 0 [6400/60000 (11%)]    Loss: 0.251311   Accuracy:97.233%
Epoch : 0 [8000/60000 (13%)]    Loss: 0.288707   Accuracy:96.987%
Epoch : 0 [9600/60000 (16%)]    Loss: 0.001341   Accuracy:96.564%
Epoch : 0 [11200/60000 (19%)]   Loss: 0.147621   Accuracy:96.270%
Epoch : 0 [12800/60000 (21%)]   Loss: 0.162637   Accuracy:96.213%
Epoch : 0 [14400/60000 (24%)]   Loss: 0.040754   Accuracy:96.182%
Epoch : 0 [16000/60000 (27%)]   Loss: 0.053642   Accuracy:96.214%
Epoch : 0 [17600/60000 (29%)]   Loss: 0.021809   Accuracy:96.257%
Epoch : 0 [19200/60000 (32%)]   Loss: 0.007951   Accuracy:96.339%
Epoch : 0 [20800/60000 (35%)]   Loss: 0.117186   Accuracy:96.376%
Epoch : 0 [22400/60000 (37%)]   Loss: 0.001032   Accuracy:96.438%
Epoch : 0 [24000/60000 (40%)]   Loss: 0.240103   Accuracy:96.451%
Epoch : 0 [25600/60000 (43%)]   Loss: 0.097887   Accuracy:96.497%
Epoch : 0 [27200/60000 (45%)]   Loss: 0.023727   Accuracy:96.500%
Epoch : 0 [28800/60000 (48%)]   Loss: 0.015119   Accuracy:96.528%
Epoch : 0 [30400/60000 (51%)]   Loss: 0.070183   Accuracy:96.537%
Epoch : 0 [32000/60000 (53%)]   Loss: 0.012500   Accuracy:96.544%
Epoch : 0 [33600/60000 (56%)]   Loss: 0.110607   Accuracy:96.530%
Epoch : 0 [35200/60000 (59%)]   Loss: 0.357193   Accuracy:96.540%
Epoch : 0 [36800/60000 (61%)]   Loss: 0.010914   Accuracy:96.549%
Epoch : 0 [38400/60000 (64%)]   Loss: 0.176304   Accuracy:96.555%
Epoch : 0 [40000/60000 (67%)]   Loss: 0.098055   Accuracy:96.530%
Epoch : 0 [41600/60000 (69%)]   Loss: 0.044533   Accuracy:96.539%
Epoch : 0 [43200/60000 (72%)]   Loss: 0.099969   Accuracy:96.551%
Epoch : 0 [44800/60000 (75%)]   Loss: 0.024746   Accuracy:96.576%
Epoch : 0 [46400/60000 (77%)]   Loss: 0.333618   Accuracy:96.550%
Epoch : 0 [48000/60000 (80%)]   Loss: 0.118122   Accuracy:96.565%
Epoch : 0 [49600/60000 (83%)]   Loss: 0.100130   Accuracy:96.555%
Epoch : 0 [51200/60000 (85%)]   Loss: 0.101556   Accuracy:96.549%
Epoch : 0 [52800/60000 (88%)]   Loss: 0.163493   Accuracy:96.555%
Epoch : 0 [54400/60000 (91%)]   Loss: 0.001940   Accuracy:96.550%
Epoch : 0 [56000/60000 (93%)]   Loss: 0.151699   Accuracy:96.552%
Epoch : 0 [57600/60000 (96%)]   Loss: 0.085508   Accuracy:96.577%
Epoch : 0 [59200/60000 (99%)]   Loss: 0.006413   Accuracy:96.607%
Epoch : 1 [0/60000 (0%)]        Loss: 0.066838   Accuracy:96.875%
Epoch : 1 [1600/60000 (3%)]     Loss: 0.038393   Accuracy:95.895%
Epoch : 1 [3200/60000 (5%)]     Loss: 0.088629   Accuracy:96.566%
```

```
Epoch : 1 [4800/60000 (8%)]     Loss: 0.073845   Accuracy:96.978%
Epoch : 1 [6400/60000 (11%)]    Loss: 0.323887   Accuracy:97.015%
Epoch : 1 [8000/60000 (13%)]    Loss: 0.080831   Accuracy:97.049%
Epoch : 1 [9600/60000 (16%)]    Loss: 0.028484   Accuracy:97.103%
Epoch : 1 [11200/60000 (19%)]   Loss: 0.217653   Accuracy:97.044%
Epoch : 1 [12800/60000 (21%)]   Loss: 0.015726   Accuracy:97.015%
Epoch : 1 [14400/60000 (24%)]   Loss: 0.031854   Accuracy:96.993%
Epoch : 1 [16000/60000 (27%)]   Loss: 0.102694   Accuracy:97.050%
Epoch : 1 [17600/60000 (29%)]   Loss: 0.314560   Accuracy:97.164%
Epoch : 1 [19200/60000 (32%)]   Loss: 0.043562   Accuracy:97.192%
Epoch : 1 [20800/60000 (35%)]   Loss: 0.078116   Accuracy:97.211%
Epoch : 1 [22400/60000 (37%)]   Loss: 0.010395   Accuracy:97.214%
Epoch : 1 [24000/60000 (40%)]   Loss: 0.008545   Accuracy:97.241%
Epoch : 1 [25600/60000 (43%)]   Loss: 0.087680   Accuracy:97.285%
Epoch : 1 [27200/60000 (45%)]   Loss: 0.016307   Accuracy:97.264%
Epoch : 1 [28800/60000 (48%)]   Loss: 0.070700   Accuracy:97.295%
Epoch : 1 [30400/60000 (51%)]   Loss: 0.283281   Accuracy:97.315%
Epoch : 1 [32000/60000 (53%)]   Loss: 0.124514   Accuracy:97.331%
Epoch : 1 [33600/60000 (56%)]   Loss: 0.077984   Accuracy:97.265%
Epoch : 1 [35200/60000 (59%)]   Loss: 0.018130   Accuracy:97.272%
Epoch : 1 [36800/60000 (61%)]   Loss: 0.063739   Accuracy:97.277%
Epoch : 1 [38400/60000 (64%)]   Loss: 0.262766   Accuracy:97.250%
Epoch : 1 [40000/60000 (67%)]   Loss: 0.040512   Accuracy:97.252%
Epoch : 1 [41600/60000 (69%)]   Loss: 0.088825   Accuracy:97.235%
Epoch : 1 [43200/60000 (72%)]   Loss: 0.161225   Accuracy:97.213%
Epoch : 1 [44800/60000 (75%)]   Loss: 0.050391   Accuracy:97.232%
Epoch : 1 [46400/60000 (77%)]   Loss: 0.577136   Accuracy:97.194%
Epoch : 1 [48000/60000 (80%)]   Loss: 0.154128   Accuracy:97.183%
Epoch : 1 [49600/60000 (83%)]   Loss: 0.199059   Accuracy:97.155%
Epoch : 1 [51200/60000 (85%)]   Loss: 0.163394   Accuracy:97.146%
Epoch : 1 [52800/60000 (88%)]   Loss: 0.163188   Accuracy:97.161%
Epoch : 1 [54400/60000 (91%)]   Loss: 0.007887   Accuracy:97.163%
Epoch : 1 [56000/60000 (93%)]   Loss: 0.123494   Accuracy:97.150%
Epoch : 1 [57600/60000 (96%)]   Loss: 0.084630   Accuracy:97.167%
Epoch : 1 [59200/60000 (99%)]   Loss: 0.041318   Accuracy:97.197%
Epoch : 2 [0/60000 (0%)]        Loss: 0.101924   Accuracy:93.750%
Epoch : 2 [1600/60000 (3%)]     Loss: 0.192111   Accuracy:96.078%
Epoch : 2 [3200/60000 (5%)]     Loss: 0.070167   Accuracy:97.123%
Epoch : 2 [4800/60000 (8%)]     Loss: 0.023145   Accuracy:97.413%
Epoch : 2 [6400/60000 (11%)]    Loss: 0.164375   Accuracy:97.217%
Epoch : 2 [8000/60000 (13%)]    Loss: 0.206731   Accuracy:97.261%
Epoch : 2 [9600/60000 (16%)]    Loss: 0.013814   Accuracy:97.228%
Epoch : 2 [11200/60000 (19%)]   Loss: 0.189980   Accuracy:97.302%
Epoch : 2 [12800/60000 (21%)]   Loss: 0.094958   Accuracy:97.319%
Epoch : 2 [14400/60000 (24%)]   Loss: 0.015213   Accuracy:97.367%
Epoch : 2 [16000/60000 (27%)]   Loss: 0.093988   Accuracy:97.287%
Epoch : 2 [17600/60000 (29%)]   Loss: 0.621408   Accuracy:97.221%
Epoch : 2 [19200/60000 (32%)]   Loss: 0.108640   Accuracy:97.223%
```

```
Epoch : 2 [20800/60000 (35%)]   Loss: 0.021036   Accuracy:97.216%
Epoch : 2 [22400/60000 (37%)]   Loss: 0.001247   Accuracy:97.209%
Epoch : 2 [24000/60000 (40%)]   Loss: 0.015268   Accuracy:97.233%
Epoch : 2 [25600/60000 (43%)]   Loss: 0.101386   Accuracy:97.261%
Epoch : 2 [27200/60000 (45%)]   Loss: 0.131324   Accuracy:97.239%
Epoch : 2 [28800/60000 (48%)]   Loss: 0.069168   Accuracy:97.243%
Epoch : 2 [30400/60000 (51%)]   Loss: 0.025075   Accuracy:97.282%
Epoch : 2 [32000/60000 (53%)]   Loss: 0.016453   Accuracy:97.300%
Epoch : 2 [33600/60000 (56%)]   Loss: 0.020328   Accuracy:97.273%
Epoch : 2 [35200/60000 (59%)]   Loss: 0.872610   Accuracy:97.298%
Epoch : 2 [36800/60000 (61%)]   Loss: 0.101088   Accuracy:97.296%
Epoch : 2 [38400/60000 (64%)]   Loss: 0.088246   Accuracy:97.294%
Epoch : 2 [40000/60000 (67%)]   Loss: 0.119826   Accuracy:97.302%
Epoch : 2 [41600/60000 (69%)]   Loss: 0.052212   Accuracy:97.324%
Epoch : 2 [43200/60000 (72%)]   Loss: 0.325394   Accuracy:97.305%
Epoch : 2 [44800/60000 (75%)]   Loss: 0.062563   Accuracy:97.310%
Epoch : 2 [46400/60000 (77%)]   Loss: 0.133759   Accuracy:97.273%
Epoch : 2 [48000/60000 (80%)]   Loss: 0.219807   Accuracy:97.271%
Epoch : 2 [49600/60000 (83%)]   Loss: 0.019319   Accuracy:97.288%
Epoch : 2 [51200/60000 (85%)]   Loss: 0.045573   Accuracy:97.273%
Epoch : 2 [52800/60000 (88%)]   Loss: 0.129453   Accuracy:97.265%
Epoch : 2 [54400/60000 (91%)]   Loss: 0.037412   Accuracy:97.270%
Epoch : 2 [56000/60000 (93%)]   Loss: 0.061451   Accuracy:97.275%
Epoch : 2 [57600/60000 (96%)]   Loss: 0.268131   Accuracy:97.278%
Epoch : 2 [59200/60000 (99%)]   Loss: 0.029599   Accuracy:97.316%
Epoch : 3 [0/60000 (0%)]        Loss: 0.151179   Accuracy:93.750%
Epoch : 3 [1600/60000 (3%)]     Loss: 0.128003   Accuracy:96.446%
Epoch : 3 [3200/60000 (5%)]     Loss: 0.006657   Accuracy:97.308%
Epoch : 3 [4800/60000 (8%)]     Loss: 0.015271   Accuracy:97.392%
Epoch : 3 [6400/60000 (11%)]    Loss: 0.127014   Accuracy:97.404%
Epoch : 3 [8000/60000 (13%)]    Loss: 0.137492   Accuracy:97.323%
Epoch : 3 [9600/60000 (16%)]    Loss: 0.011746   Accuracy:97.290%
Epoch : 3 [11200/60000 (19%)]   Loss: 0.193034   Accuracy:97.391%
Epoch : 3 [12800/60000 (21%)]   Loss: 0.151988   Accuracy:97.506%
Epoch : 3 [14400/60000 (24%)]   Loss: 0.001826   Accuracy:97.561%
Epoch : 3 [16000/60000 (27%)]   Loss: 0.038850   Accuracy:97.493%
Epoch : 3 [17600/60000 (29%)]   Loss: 0.005036   Accuracy:97.533%
Epoch : 3 [19200/60000 (32%)]   Loss: 0.041748   Accuracy:97.577%
Epoch : 3 [20800/60000 (35%)]   Loss: 0.061989   Accuracy:97.552%
Epoch : 3 [22400/60000 (37%)]   Loss: 0.000083   Accuracy:97.562%
Epoch : 3 [24000/60000 (40%)]   Loss: 0.010684   Accuracy:97.562%
Epoch : 3 [25600/60000 (43%)]   Loss: 0.035633   Accuracy:97.554%
Epoch : 3 [27200/60000 (45%)]   Loss: 0.003687   Accuracy:97.536%
Epoch : 3 [28800/60000 (48%)]   Loss: 0.059841   Accuracy:97.562%
Epoch : 3 [30400/60000 (51%)]   Loss: 0.055981   Accuracy:97.581%
Epoch : 3 [32000/60000 (53%)]   Loss: 0.057697   Accuracy:97.596%
Epoch : 3 [33600/60000 (56%)]   Loss: 0.099345   Accuracy:97.589%
Epoch : 3 [35200/60000 (59%)]   Loss: 0.004503   Accuracy:97.593%
```

```
Epoch : 3 [36800/60000 (61%)]   Loss: 0.132729   Accuracy:97.597%
Epoch : 3 [38400/60000 (64%)]   Loss: 0.078958   Accuracy:97.580%
Epoch : 3 [40000/60000 (67%)]   Loss: 0.049452   Accuracy:97.577%
Epoch : 3 [41600/60000 (69%)]   Loss: 0.001719   Accuracy:97.574%
Epoch : 3 [43200/60000 (72%)]   Loss: 0.175857   Accuracy:97.580%
Epoch : 3 [44800/60000 (75%)]   Loss: 0.034069   Accuracy:97.589%
Epoch : 3 [46400/60000 (77%)]   Loss: 0.021272   Accuracy:97.599%
Epoch : 3 [48000/60000 (80%)]   Loss: 0.064088   Accuracy:97.620%
Epoch : 3 [49600/60000 (83%)]   Loss: 0.003137   Accuracy:97.625%
Epoch : 3 [51200/60000 (85%)]   Loss: 0.014169   Accuracy:97.628%
Epoch : 3 [52800/60000 (88%)]   Loss: 0.221221   Accuracy:97.625%
Epoch : 3 [54400/60000 (91%)]   Loss: 0.001268   Accuracy:97.612%
Epoch : 3 [56000/60000 (93%)]   Loss: 0.063965   Accuracy:97.596%
Epoch : 3 [57600/60000 (96%)]   Loss: 0.254212   Accuracy:97.604%
Epoch : 3 [59200/60000 (99%)]   Loss: 0.000153   Accuracy:97.633%
Epoch : 4 [0/60000 (0%)]        Loss: 0.077204   Accuracy:96.875%
Epoch : 4 [1600/60000 (3%)]     Loss: 0.160978   Accuracy:97.059%
Epoch : 4 [3200/60000 (5%)]     Loss: 0.022128   Accuracy:97.463%
Epoch : 4 [4800/60000 (8%)]     Loss: 0.087849   Accuracy:97.641%
Epoch : 4 [6400/60000 (11%)]    Loss: 0.282174   Accuracy:97.512%
Epoch : 4 [8000/60000 (13%)]    Loss: 0.022924   Accuracy:97.572%
Epoch : 4 [9600/60000 (16%)]    Loss: 0.057789   Accuracy:97.456%
Epoch : 4 [11200/60000 (19%)]   Loss: 0.173873   Accuracy:97.480%
Epoch : 4 [12800/60000 (21%)]   Loss: 0.177891   Accuracy:97.498%
Epoch : 4 [14400/60000 (24%)]   Loss: 0.058999   Accuracy:97.568%
Epoch : 4 [16000/60000 (27%)]   Loss: 0.087360   Accuracy:97.493%
Epoch : 4 [17600/60000 (29%)]   Loss: 0.002993   Accuracy:97.544%
Epoch : 4 [19200/60000 (32%)]   Loss: 0.007460   Accuracy:97.577%
Epoch : 4 [20800/60000 (35%)]   Loss: 0.172938   Accuracy:97.576%
Epoch : 4 [22400/60000 (37%)]   Loss: 0.003647   Accuracy:97.597%
Epoch : 4 [24000/60000 (40%)]   Loss: 0.003558   Accuracy:97.616%
Epoch : 4 [25600/60000 (43%)]   Loss: 0.123521   Accuracy:97.651%
Epoch : 4 [27200/60000 (45%)]   Loss: 0.062108   Accuracy:97.650%
Epoch : 4 [28800/60000 (48%)]   Loss: 0.079932   Accuracy:97.683%
Epoch : 4 [30400/60000 (51%)]   Loss: 0.002716   Accuracy:97.703%
Epoch : 4 [32000/60000 (53%)]   Loss: 0.218044   Accuracy:97.687%
Epoch : 4 [33600/60000 (56%)]   Loss: 0.009926   Accuracy:97.693%
Epoch : 4 [35200/60000 (59%)]   Loss: 0.001960   Accuracy:97.707%
Epoch : 4 [36800/60000 (61%)]   Loss: 0.003491   Accuracy:97.706%
Epoch : 4 [38400/60000 (64%)]   Loss: 0.193213   Accuracy:97.684%
Epoch : 4 [40000/60000 (67%)]   Loss: 0.121122   Accuracy:97.677%
Epoch : 4 [41600/60000 (69%)]   Loss: 0.058635   Accuracy:97.672%
Epoch : 4 [43200/60000 (72%)]   Loss: 0.121910   Accuracy:97.668%
Epoch : 4 [44800/60000 (75%)]   Loss: 0.020887   Accuracy:97.711%
Epoch : 4 [46400/60000 (77%)]   Loss: 0.099990   Accuracy:97.676%
Epoch : 4 [48000/60000 (80%)]   Loss: 0.167490   Accuracy:97.677%
Epoch : 4 [49600/60000 (83%)]   Loss: 0.096079   Accuracy:97.671%
Epoch : 4 [51200/60000 (85%)]   Loss: 0.009618   Accuracy:97.687%
```

```
Epoch : 4 [52800/60000 (88%)]   Loss: 0.322359   Accuracy:97.714%
Epoch : 4 [54400/60000 (91%)]   Loss: 0.001315   Accuracy:97.702%
Epoch : 4 [56000/60000 (93%)]   Loss: 0.065532   Accuracy:97.717%
Epoch : 4 [57600/60000 (96%)]   Loss: 0.120293   Accuracy:97.729%
Epoch : 4 [59200/60000 (99%)]   Loss: 0.000217   Accuracy:97.758%
```

[68]:
```
def evaluate(model):
#model = mlp
    correct = 0
    for test_imgs, test_labels in test_loader:
        #print(test_imgs.shape)
        test_imgs = Variable(test_imgs).float()
        output = model(test_imgs)
        predicted = torch.max(output,1)[1]
        correct += (predicted == test_labels).sum()
    print("Test accuracy:{:.3f}% ".format( float(correct) /␣
 ↪(len(test_loader)*BATCH_SIZE)))
evaluate(mlp)
```

```
Test accuracy:0.971%
```

##Since a CNN needs a image shape as input let's reshape our flatten images to real image

[69]:
```
torch_X_train = torch_X_train.view(-1, 1,28,28).float()
torch_X_test = torch_X_test.view(-1,1,28,28).float()
print(torch_X_train.shape)
print(torch_X_test.shape)

# Pytorch train and test sets
train = torch.utils.data.TensorDataset(torch_X_train,torch_y_train)
test = torch.utils.data.TensorDataset(torch_X_test,torch_y_test)

# data loader
train_loader = torch.utils.data.DataLoader(train, batch_size = BATCH_SIZE,␣
 ↪shuffle = False)
test_loader = torch.utils.data.DataLoader(test, batch_size = BATCH_SIZE,␣
 ↪shuffle = False)MB
```

```
torch.Size([60000, 1, 28, 28])
torch.Size([10000, 1, 28, 28])
```

[70]:
```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=5)
        self.conv2 = nn.Conv2d(32, 32, kernel_size=5)
        self.conv3 = nn.Conv2d(32,64, kernel_size=5)
        self.fc1 = nn.Linear(3*3*64, 256)
```

```
        self.fc2 = nn.Linear(256, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        #x = F.dropout(x, p=0.5, training=self.training)
        x = F.relu(F.max_pool2d(self.conv2(x), 2))
        x = F.dropout(x, p=0.5, training=self.training)
        x = F.relu(F.max_pool2d(self.conv3(x),2))
        x = F.dropout(x, p=0.5, training=self.training)
        x = x.view(-1,3*3*64 )
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)

cnn = CNN()
print(cnn)

it = iter(train_loader)
X_batch, y_batch = next(it)
print(cnn.forward(X_batch).shape)
```

```
CNN(
  (conv1): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(32, 32, kernel_size=(5, 5), stride=(1, 1))
  (conv3): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=576, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=10, bias=True)
)
torch.Size([32, 10])
```

[71]:
```
fit(cnn,train_loader)
```

```
Epoch : 0 [0/60000 (0%)]        Loss: 24.703955  Accuracy:6.250%
Epoch : 0 [1600/60000 (3%)]     Loss: 1.980315   Accuracy:17.279%
Epoch : 0 [3200/60000 (5%)]     Loss: 1.460487   Accuracy:29.115%
Epoch : 0 [4800/60000 (8%)]     Loss: 0.748759   Accuracy:38.949%
Epoch : 0 [6400/60000 (11%)]    Loss: 1.238455   Accuracy:45.553%
Epoch : 0 [8000/60000 (13%)]    Loss: 0.865389   Accuracy:50.461%
Epoch : 0 [9600/60000 (16%)]    Loss: 0.764462   Accuracy:54.485%
Epoch : 0 [11200/60000 (19%)]   Loss: 0.602340   Accuracy:58.191%
Epoch : 0 [12800/60000 (21%)]   Loss: 0.548422   Accuracy:60.793%
Epoch : 0 [14400/60000 (24%)]   Loss: 0.491586   Accuracy:63.089%
Epoch : 0 [16000/60000 (27%)]   Loss: 0.702923   Accuracy:64.814%
Epoch : 0 [17600/60000 (29%)]   Loss: 0.591373   Accuracy:66.640%
Epoch : 0 [19200/60000 (32%)]   Loss: 0.427941   Accuracy:68.225%
Epoch : 0 [20800/60000 (35%)]   Loss: 0.508253   Accuracy:69.772%
Epoch : 0 [22400/60000 (37%)]   Loss: 0.220711   Accuracy:71.122%
```

```
Epoch : 0 [24000/60000 (40%)]   Loss: 0.182511   Accuracy:72.121%
Epoch : 0 [25600/60000 (43%)]   Loss: 0.578731   Accuracy:73.174%
Epoch : 0 [27200/60000 (45%)]   Loss: 0.405716   Accuracy:74.104%
Epoch : 0 [28800/60000 (48%)]   Loss: 0.335528   Accuracy:75.035%
Epoch : 0 [30400/60000 (51%)]   Loss: 0.524058   Accuracy:75.710%
Epoch : 0 [32000/60000 (53%)]   Loss: 0.522807   Accuracy:76.346%
Epoch : 0 [33600/60000 (56%)]   Loss: 0.353773   Accuracy:77.019%
Epoch : 0 [35200/60000 (59%)]   Loss: 0.348661   Accuracy:77.677%
Epoch : 0 [36800/60000 (61%)]   Loss: 0.335008   Accuracy:78.307%
Epoch : 0 [38400/60000 (64%)]   Loss: 0.295484   Accuracy:78.841%
Epoch : 0 [40000/60000 (67%)]   Loss: 0.330925   Accuracy:79.359%
Epoch : 0 [41600/60000 (69%)]   Loss: 0.761555   Accuracy:79.790%
Epoch : 0 [43200/60000 (72%)]   Loss: 0.308168   Accuracy:80.204%
Epoch : 0 [44800/60000 (75%)]   Loss: 0.039641   Accuracy:80.661%
Epoch : 0 [46400/60000 (77%)]   Loss: 0.347005   Accuracy:81.032%
Epoch : 0 [48000/60000 (80%)]   Loss: 0.695292   Accuracy:81.404%
Epoch : 0 [49600/60000 (83%)]   Loss: 0.664559   Accuracy:81.742%
Epoch : 0 [51200/60000 (85%)]   Loss: 0.267460   Accuracy:82.083%
Epoch : 0 [52800/60000 (88%)]   Loss: 0.457143   Accuracy:82.418%
Epoch : 0 [54400/60000 (91%)]   Loss: 0.123103   Accuracy:82.742%
Epoch : 0 [56000/60000 (93%)]   Loss: 0.171987   Accuracy:83.040%
Epoch : 0 [57600/60000 (96%)]   Loss: 0.293099   Accuracy:83.320%
Epoch : 0 [59200/60000 (99%)]   Loss: 0.139004   Accuracy:83.656%
Epoch : 1 [0/60000 (0%)]        Loss: 0.197437   Accuracy:93.750%
Epoch : 1 [1600/60000 (3%)]     Loss: 0.182057   Accuracy:93.137%
Epoch : 1 [3200/60000 (5%)]     Loss: 0.449420   Accuracy:94.028%
Epoch : 1 [4800/60000 (8%)]     Loss: 0.116307   Accuracy:93.729%
Epoch : 1 [6400/60000 (11%)]    Loss: 0.029951   Accuracy:93.797%
Epoch : 1 [8000/60000 (13%)]    Loss: 0.101050   Accuracy:93.875%
Epoch : 1 [9600/60000 (16%)]    Loss: 0.333615   Accuracy:93.625%
Epoch : 1 [11200/60000 (19%)]   Loss: 0.426801   Accuracy:93.643%
Epoch : 1 [12800/60000 (21%)]   Loss: 0.298322   Accuracy:93.462%
Epoch : 1 [14400/60000 (24%)]   Loss: 0.108020   Accuracy:93.528%
Epoch : 1 [16000/60000 (27%)]   Loss: 0.474114   Accuracy:93.469%
Epoch : 1 [17600/60000 (29%)]   Loss: 0.277469   Accuracy:93.455%
Epoch : 1 [19200/60000 (32%)]   Loss: 0.163615   Accuracy:93.506%
Epoch : 1 [20800/60000 (35%)]   Loss: 0.227395   Accuracy:93.568%
Epoch : 1 [22400/60000 (37%)]   Loss: 0.010190   Accuracy:93.630%
Epoch : 1 [24000/60000 (40%)]   Loss: 0.085058   Accuracy:93.633%
Epoch : 1 [25600/60000 (43%)]   Loss: 0.092341   Accuracy:93.606%
Epoch : 1 [27200/60000 (45%)]   Loss: 0.161767   Accuracy:93.548%
Epoch : 1 [28800/60000 (48%)]   Loss: 0.029576   Accuracy:93.573%
Epoch : 1 [30400/60000 (51%)]   Loss: 0.205116   Accuracy:93.589%
Epoch : 1 [32000/60000 (53%)]   Loss: 0.252389   Accuracy:93.581%
Epoch : 1 [33600/60000 (56%)]   Loss: 0.318719   Accuracy:93.583%
Epoch : 1 [35200/60000 (59%)]   Loss: 0.124187   Accuracy:93.583%
Epoch : 1 [36800/60000 (61%)]   Loss: 0.110199   Accuracy:93.636%
Epoch : 1 [38400/60000 (64%)]   Loss: 0.135114   Accuracy:93.633%
```

```
Epoch : 1 [40000/60000 (67%)]   Loss: 0.337480   Accuracy:93.608%
Epoch : 1 [41600/60000 (69%)]   Loss: 0.151904   Accuracy:93.637%
Epoch : 1 [43200/60000 (72%)]   Loss: 0.201784   Accuracy:93.651%
Epoch : 1 [44800/60000 (75%)]   Loss: 0.120788   Accuracy:93.670%
Epoch : 1 [46400/60000 (77%)]   Loss: 0.631979   Accuracy:93.638%
Epoch : 1 [48000/60000 (80%)]   Loss: 0.330198   Accuracy:93.644%
Epoch : 1 [49600/60000 (83%)]   Loss: 0.092436   Accuracy:93.667%
Epoch : 1 [51200/60000 (85%)]   Loss: 0.046334   Accuracy:93.693%
Epoch : 1 [52800/60000 (88%)]   Loss: 0.316572   Accuracy:93.701%
Epoch : 1 [54400/60000 (91%)]   Loss: 0.114109   Accuracy:93.695%
Epoch : 1 [56000/60000 (93%)]   Loss: 0.394695   Accuracy:93.727%
Epoch : 1 [57600/60000 (96%)]   Loss: 0.083526   Accuracy:93.792%
Epoch : 1 [59200/60000 (99%)]   Loss: 0.015199   Accuracy:93.860%
Epoch : 2 [0/60000 (0%)]        Loss: 0.217455   Accuracy:90.625%
Epoch : 2 [1600/60000 (3%)]     Loss: 0.406698   Accuracy:94.056%
Epoch : 2 [3200/60000 (5%)]     Loss: 0.423212   Accuracy:94.864%
Epoch : 2 [4800/60000 (8%)]     Loss: 0.065729   Accuracy:94.454%
Epoch : 2 [6400/60000 (11%)]    Loss: 0.006236   Accuracy:94.325%
Epoch : 2 [8000/60000 (13%)]    Loss: 0.204239   Accuracy:94.410%
Epoch : 2 [9600/60000 (16%)]    Loss: 0.303900   Accuracy:94.248%
Epoch : 2 [11200/60000 (19%)]   Loss: 0.255757   Accuracy:94.436%
Epoch : 2 [12800/60000 (21%)]   Loss: 0.152741   Accuracy:94.459%
Epoch : 2 [14400/60000 (24%)]   Loss: 0.082564   Accuracy:94.471%
Epoch : 2 [16000/60000 (27%)]   Loss: 0.283941   Accuracy:94.374%
Epoch : 2 [17600/60000 (29%)]   Loss: 0.014378   Accuracy:94.380%
Epoch : 2 [19200/60000 (32%)]   Loss: 0.083600   Accuracy:94.452%
Epoch : 2 [20800/60000 (35%)]   Loss: 0.360928   Accuracy:94.556%
Epoch : 2 [22400/60000 (37%)]   Loss: 0.102351   Accuracy:94.593%
Epoch : 2 [24000/60000 (40%)]   Loss: 0.109202   Accuracy:94.653%
Epoch : 2 [25600/60000 (43%)]   Loss: 0.092279   Accuracy:94.714%
Epoch : 2 [27200/60000 (45%)]   Loss: 0.241476   Accuracy:94.679%
Epoch : 2 [28800/60000 (48%)]   Loss: 0.107590   Accuracy:94.704%
Epoch : 2 [30400/60000 (51%)]   Loss: 0.188392   Accuracy:94.759%
Epoch : 2 [32000/60000 (53%)]   Loss: 0.298663   Accuracy:94.699%
Epoch : 2 [33600/60000 (56%)]   Loss: 0.165397   Accuracy:94.690%
Epoch : 2 [35200/60000 (59%)]   Loss: 0.109414   Accuracy:94.738%
Epoch : 2 [36800/60000 (61%)]   Loss: 0.111880   Accuracy:94.790%
Epoch : 2 [38400/60000 (64%)]   Loss: 0.247883   Accuracy:94.786%
Epoch : 2 [40000/60000 (67%)]   Loss: 0.241609   Accuracy:94.799%
Epoch : 2 [41600/60000 (69%)]   Loss: 0.274725   Accuracy:94.809%
Epoch : 2 [43200/60000 (72%)]   Loss: 0.335197   Accuracy:94.828%
Epoch : 2 [44800/60000 (75%)]   Loss: 0.005174   Accuracy:94.872%
Epoch : 2 [46400/60000 (77%)]   Loss: 1.024677   Accuracy:94.889%
Epoch : 2 [48000/60000 (80%)]   Loss: 0.283337   Accuracy:94.860%
Epoch : 2 [49600/60000 (83%)]   Loss: 0.264172   Accuracy:94.854%
Epoch : 2 [51200/60000 (85%)]   Loss: 0.145659   Accuracy:94.857%
Epoch : 2 [52800/60000 (88%)]   Loss: 0.662844   Accuracy:94.893%
Epoch : 2 [54400/60000 (91%)]   Loss: 0.017660   Accuracy:94.922%
```

```
Epoch : 2 [56000/60000 (93%)]   Loss: 0.118975   Accuracy:94.960%
Epoch : 2 [57600/60000 (96%)]   Loss: 0.390172   Accuracy:94.961%
Epoch : 2 [59200/60000 (99%)]   Loss: 0.000941   Accuracy:95.016%
Epoch : 3 [0/60000 (0%)]        Loss: 0.039304   Accuracy:96.875%
Epoch : 3 [1600/60000 (3%)]     Loss: 0.274808   Accuracy:94.424%
Epoch : 3 [3200/60000 (5%)]     Loss: 0.082404   Accuracy:95.142%
Epoch : 3 [4800/60000 (8%)]     Loss: 0.180459   Accuracy:95.281%
Epoch : 3 [6400/60000 (11%)]    Loss: 0.055648   Accuracy:95.134%
Epoch : 3 [8000/60000 (13%)]    Loss: 0.032606   Accuracy:95.219%
Epoch : 3 [9600/60000 (16%)]    Loss: 0.057986   Accuracy:94.975%
Epoch : 3 [11200/60000 (19%)]   Loss: 0.275520   Accuracy:95.103%
Epoch : 3 [12800/60000 (21%)]   Loss: 0.070716   Accuracy:94.950%
Epoch : 3 [14400/60000 (24%)]   Loss: 0.027030   Accuracy:94.928%
Epoch : 3 [16000/60000 (27%)]   Loss: 0.066866   Accuracy:94.941%
Epoch : 3 [17600/60000 (29%)]   Loss: 0.028506   Accuracy:94.941%
Epoch : 3 [19200/60000 (32%)]   Loss: 0.219603   Accuracy:94.998%
Epoch : 3 [20800/60000 (35%)]   Loss: 0.051435   Accuracy:94.993%
Epoch : 3 [22400/60000 (37%)]   Loss: 0.011864   Accuracy:95.047%
Epoch : 3 [24000/60000 (40%)]   Loss: 0.177396   Accuracy:95.073%
Epoch : 3 [25600/60000 (43%)]   Loss: 0.050125   Accuracy:95.154%
Epoch : 3 [27200/60000 (45%)]   Loss: 0.215653   Accuracy:95.131%
Epoch : 3 [28800/60000 (48%)]   Loss: 0.357761   Accuracy:95.106%
Epoch : 3 [30400/60000 (51%)]   Loss: 0.303260   Accuracy:95.127%
Epoch : 3 [32000/60000 (53%)]   Loss: 0.435719   Accuracy:95.042%
Epoch : 3 [33600/60000 (56%)]   Loss: 0.038899   Accuracy:95.008%
Epoch : 3 [35200/60000 (59%)]   Loss: 0.157955   Accuracy:95.084%
Epoch : 3 [36800/60000 (61%)]   Loss: 0.065332   Accuracy:95.127%
Epoch : 3 [38400/60000 (64%)]   Loss: 0.087521   Accuracy:95.145%
Epoch : 3 [40000/60000 (67%)]   Loss: 0.119639   Accuracy:95.131%
Epoch : 3 [41600/60000 (69%)]   Loss: 0.089203   Accuracy:95.143%
Epoch : 3 [43200/60000 (72%)]   Loss: 0.412446   Accuracy:95.152%
Epoch : 3 [44800/60000 (75%)]   Loss: 0.103714   Accuracy:95.211%
Epoch : 3 [46400/60000 (77%)]   Loss: 0.355738   Accuracy:95.202%
Epoch : 3 [48000/60000 (80%)]   Loss: 0.139295   Accuracy:95.191%
Epoch : 3 [49600/60000 (83%)]   Loss: 0.194295   Accuracy:95.189%
Epoch : 3 [51200/60000 (85%)]   Loss: 0.083338   Accuracy:95.181%
Epoch : 3 [52800/60000 (88%)]   Loss: 0.779976   Accuracy:95.192%
Epoch : 3 [54400/60000 (91%)]   Loss: 0.033422   Accuracy:95.196%
Epoch : 3 [56000/60000 (93%)]   Loss: 0.117889   Accuracy:95.208%
Epoch : 3 [57600/60000 (96%)]   Loss: 0.146757   Accuracy:95.232%
Epoch : 3 [59200/60000 (99%)]   Loss: 0.007108   Accuracy:95.286%
Epoch : 4 [0/60000 (0%)]        Loss: 0.037586   Accuracy:96.875%
Epoch : 4 [1600/60000 (3%)]     Loss: 0.532983   Accuracy:95.282%
Epoch : 4 [3200/60000 (5%)]     Loss: 0.141879   Accuracy:95.575%
Epoch : 4 [4800/60000 (8%)]     Loss: 0.068667   Accuracy:95.882%
Epoch : 4 [6400/60000 (11%)]    Loss: 0.088112   Accuracy:95.740%
Epoch : 4 [8000/60000 (13%)]    Loss: 0.112370   Accuracy:95.754%
Epoch : 4 [9600/60000 (16%)]    Loss: 0.098440   Accuracy:95.650%
```

```
Epoch : 4 [11200/60000 (19%)]   Loss: 0.284854   Accuracy:95.593%
Epoch : 4 [12800/60000 (21%)]   Loss: 0.082128   Accuracy:95.605%
Epoch : 4 [14400/60000 (24%)]   Loss: 0.066085   Accuracy:95.468%
Epoch : 4 [16000/60000 (27%)]   Loss: 0.137398   Accuracy:95.422%
Epoch : 4 [17600/60000 (29%)]   Loss: 0.083179   Accuracy:95.389%
Epoch : 4 [19200/60000 (32%)]   Loss: 0.296649   Accuracy:95.372%
Epoch : 4 [20800/60000 (35%)]   Loss: 0.079538   Accuracy:95.387%
Epoch : 4 [22400/60000 (37%)]   Loss: 0.021219   Accuracy:95.399%
Epoch : 4 [24000/60000 (40%)]   Loss: 0.045635   Accuracy:95.460%
Epoch : 4 [25600/60000 (43%)]   Loss: 0.025185   Accuracy:95.431%
Epoch : 4 [27200/60000 (45%)]   Loss: 0.313549   Accuracy:95.384%
Epoch : 4 [28800/60000 (48%)]   Loss: 0.358900   Accuracy:95.384%
Epoch : 4 [30400/60000 (51%)]   Loss: 0.290802   Accuracy:95.354%
Epoch : 4 [32000/60000 (53%)]   Loss: 0.031017   Accuracy:95.367%
Epoch : 4 [33600/60000 (56%)]   Loss: 0.085758   Accuracy:95.362%
Epoch : 4 [35200/60000 (59%)]   Loss: 0.152539   Accuracy:95.416%
Epoch : 4 [36800/60000 (61%)]   Loss: 0.162986   Accuracy:95.441%
Epoch : 4 [38400/60000 (64%)]   Loss: 0.174990   Accuracy:95.444%
Epoch : 4 [40000/60000 (67%)]   Loss: 0.521610   Accuracy:95.456%
Epoch : 4 [41600/60000 (69%)]   Loss: 0.122210   Accuracy:95.484%
Epoch : 4 [43200/60000 (72%)]   Loss: 0.318961   Accuracy:95.503%
Epoch : 4 [44800/60000 (75%)]   Loss: 0.035795   Accuracy:95.550%
Epoch : 4 [46400/60000 (77%)]   Loss: 0.384703   Accuracy:95.535%
Epoch : 4 [48000/60000 (80%)]   Loss: 0.491851   Accuracy:95.561%
Epoch : 4 [49600/60000 (83%)]   Loss: 0.055732   Accuracy:95.561%
Epoch : 4 [51200/60000 (85%)]   Loss: 0.087599   Accuracy:95.583%
Epoch : 4 [52800/60000 (88%)]   Loss: 0.285666   Accuracy:95.597%
Epoch : 4 [54400/60000 (91%)]   Loss: 0.043655   Accuracy:95.602%
Epoch : 4 [56000/60000 (93%)]   Loss: 0.363469   Accuracy:95.604%
Epoch : 4 [57600/60000 (96%)]   Loss: 0.103343   Accuracy:95.624%
Epoch : 4 [59200/60000 (99%)]   Loss: 0.018067   Accuracy:95.676%
```

0.3 SVM Classifer for MNIST DATA SET

[0]:
```
from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

[0]:
```
x_train = x_train.reshape(60000,784)
x_test = x_test.reshape(10000,784)
```

[0]:
```
## Applying HOG feature extraction
```

[0]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from skimage.feature import hog
from sklearn import preprocessing
```

```
from collections import Counter
```

[38]:
```
list_hog_train = []
for feature in x_train:
    fd = hog(feature.reshape((28,28)), orientations=10,␣
 ↪pixels_per_cell=(7,7),cells_per_block=(1,1),visualize=False )
    list_hog_train.append(fd)
hog_features = np.array(list_hog_train, 'float64')
preProcess = preprocessing.MaxAbsScaler().fit(hog_features)
hog_features_transformed_train = preProcess.transform(hog_features)
print(hog_features_transformed_train.shape)
```

```
(60000, 160)
```

[0]:
```
## Extracting hog feature for test data
```

[41]:
```
list_hog_test = []
for feature in x_test:
    fd = hog(feature.reshape((28,28)), orientations=10,␣
 ↪pixels_per_cell=(7,7),cells_per_block=(1,1),visualize=False )
    list_hog_test.append(fd)
hog_features_test = np.array(list_hog_test, 'float64')
preProcess = preprocessing.MaxAbsScaler().fit(hog_features_test)
hog_features_transformed_test = preProcess.transform(hog_features_test)
print(hog_features_transformed_test.shape)
```

```
(10000, 160)
```

[0]:
```
model = SVC()
model.fit(hog_features_transformed_train,y_train)
y_pred = model.predict(hog_features_transformed_test)
```

Fitting the model with best parameters

[43]:
```
print(accuracy_score(y_test, y_pred))
```

```
0.9713
```