# Part 2 Experiment with my dataset

## Prepare the model and load my dataset

In [0]:
```
#install torch in google colab
# http://pytorch.org/
from os.path import exists
from wheel.pep425tags import get_abbr_impl, get_impl_ver, get_abi_tag
platform = '{}{}-{}'.format(get_abbr_impl(), get_impl_ver(), get_abi_tag())
cuda_output = !ldconfig -p|grep cudart.so|sed -e 's/.*\.\([0-9]*\)\.\([0-9]*\)$/cu\1\2/'
accelerator = cuda_output[0] if exists('/dev/nvidia0') else 'cpu'

!pip install -q http://download.pytorch.org/whl/{accelerator}/torch-0.4.1-{platform}-linux_x86_64.whl torchvision
```

In [0]:
```
import torch
print(torch.__version__)
print(torch.cuda.is_available())
```

```
0.4.1
False
```

In [0]:
```
#initialization
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
import torchvision.datasets as datasets

# Device configuration
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

# Hyper parameters
num_epochs = 5
num_classes = 10
batch_size = 100
learning_rate = 0.001

# MNIST dataset
mnist_trainset = datasets.MNIST(root='./data', train=True, download=True, transform=None)
len(mnist_trainset)
print(mnist_trainset)

train_dataset = torchvision.datasets.MNIST(root='./data/',
                                           train=True,
                                           transform=transforms.ToTensor(),
                                           download=True)

test_dataset = torchvision.datasets.MNIST(root='./data/',
                                          train=False,
                                          transform=transforms.ToTensor())
```

```
Dataset MNIST
    Number of datapoints: 60000
    Split: train
    Root Location: ./data
    Transforms (if any): None
    Target Transforms (if any): None
```

In [0]:
```
# Mounte files to gdrive
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee649
lhc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fww
w.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.
com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
·········
Mounted at /content/gdrive
```

In [0]:
```
# loade test myDateSet

import pandas as pd
from matplotlib.pyplot import imshow
import numpy as np
from PIL import Image
from torch.utils.data.dataset import Dataset
from torchvision import transforms

class SimpleDataset(Dataset):
    def __init__(self, data_path, csv_name, transform = None ):
        """
        Args:
            data_path (string): path to the folder where images and csv files are located
            csv_name (string): name of the csv label file
            transform: pytorch transforms for transforms and tensor conversion
        """
        # Set path
        self.data_path = data_path
        # Transforms
        self.transform = transform
        # Read the csv file
        self.data_info = pd.read_csv(data_path + csv_name, header=None)
        # First column contains the image paths
        self.image_arr = np.asarray(self.data_info.iloc[:, 0])
        # Second column is the labels
        self.label_arr = np.asarray(self.data_info.iloc[:, 1])
        # Calculate len
        self.data_len = len(self.data_info.index)


    def __getitem__(self, index):
        # Get image name from the pandas df
        single_image_name = self.image_arr[index]
```

```
        # Open image
        img_as_img = Image.open(self.data_path + single_image_name)
        if self.transform is not None:
              img_as_img = self.transform(img_as_img)

        # Get label(class) of the image based on the cropped pandas column
        single_image_label = self.label_arr[index]
        #convert to tensor to be consistent with MNIST dataset
        single_image_label = torch.LongTensor( [ single_image_label ] )[0]
        return (img_as_img, single_image_label)

    def __len__(self):
        return self.data_len

#mydata = SimpleDataset( "gdrive/My Drive/myDataSet/", "label.csv")
mydataT = SimpleDataset( "gdrive/My Drive/myDataSet/", "label.csv", transform=transforms.ToTensor())

#testc = mydataT[1]
#testT = testc[0]
#imgt = Image.fromarray((testT.numpy()[0] * 255).astype("uint8"))
#display(imgt)
```

In [0]:
```
# loade My test set to make my test loader
my_test_loader = torch.utils.data.DataLoader(dataset=mydataT,
                                             batch_size=batch_size,
                                             shuffle=False)
```

In [0]:
```
# Convolutional neural network (two convolutional layers)
class ConvNet(nn.Module):
    def __init__(self, num_classes=10):
        super(ConvNet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.fc = nn.Linear(7*7*32, num_classes)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.reshape(out.size(0), -1)
        out = self.fc(out)
        return out

model = ConvNet(num_classes).to(device)
```

In [0]:
```
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

## 1.1 experiment on 100 size trainning set

In [0]:
```
# Hyper parameters
num_epochs = 5
num_classes = 10
batch_size = 50
learning_rate = 0.001

# Data loader

# loade trainning MNIST set
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)

# Train the model
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        if i > 1:
            break
        images = images.to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if (i+1) % 1 == 0:
            print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                   .format(epoch+1, num_epochs, i+1, total_step, loss.item()))
```

```
Epoch [1/5], Step [1/1200], Loss: 0.0746
Epoch [1/5], Step [2/1200], Loss: 0.1215
Epoch [2/5], Step [1/1200], Loss: 0.0581
Epoch [2/5], Step [2/1200], Loss: 0.0184
Epoch [3/5], Step [1/1200], Loss: 0.1512
Epoch [3/5], Step [2/1200], Loss: 0.2745
Epoch [4/5], Step [1/1200], Loss: 0.0497
Epoch [4/5], Step [2/1200], Loss: 0.0699
Epoch [5/5], Step [1/1200], Loss: 0.3059
Epoch [5/5], Step [2/1200], Loss: 0.0232
```

In [0]:
```
# Test the model on myDataSet
model.eval()  # eval mode (batchnorm uses moving mean/variance instead of mini-batch mean/variance)
with torch.no_grad():
    correct = 0
```

```
    correct = 0
    total = 0
    for images, labels in my_test_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Test Accuracy of 100 images trained model on the my dataset: {} %'.format(100 * correct / total))
```

```
Test Accuracy of 100 images trained model on the my dataset: 66.66666666666667 %
```

### 1.2 experiment on 250 size trainning set

In [0]:
```
num_epochs = 5
num_classes = 10
batch_size = 50
learning_rate = 0.001

# Data loader

# loade trainning MNIST set
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)

# Train the model
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        if i > 4:
            break
        images = images.to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if (i+1) % 1 == 0:
            print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                   .format(epoch+1, num_epochs, i+1, total_step, loss.item()))
```

```
Epoch [1/5], Step [1/1200], Loss: 0.0706
Epoch [1/5], Step [2/1200], Loss: 0.1659
Epoch [1/5], Step [3/1200], Loss: 0.0552
Epoch [1/5], Step [4/1200], Loss: 0.2078
Epoch [1/5], Step [5/1200], Loss: 0.0696
Epoch [2/5], Step [1/1200], Loss: 0.0255
Epoch [2/5], Step [2/1200], Loss: 0.0501
Epoch [2/5], Step [3/1200], Loss: 0.1045
Epoch [2/5], Step [4/1200], Loss: 0.0424
Epoch [2/5], Step [5/1200], Loss: 0.0489
Epoch [3/5], Step [1/1200], Loss: 0.1225
Epoch [3/5], Step [2/1200], Loss: 0.0569
Epoch [3/5], Step [3/1200], Loss: 0.1091
Epoch [3/5], Step [4/1200], Loss: 0.1069
Epoch [3/5], Step [5/1200], Loss: 0.0500
Epoch [4/5], Step [1/1200], Loss: 0.0635
Epoch [4/5], Step [2/1200], Loss: 0.1186
Epoch [4/5], Step [3/1200], Loss: 0.0227
Epoch [4/5], Step [4/1200], Loss: 0.2184
Epoch [4/5], Step [5/1200], Loss: 0.1934
Epoch [5/5], Step [1/1200], Loss: 0.0969
Epoch [5/5], Step [2/1200], Loss: 0.1210
Epoch [5/5], Step [3/1200], Loss: 0.0166
Epoch [5/5], Step [4/1200], Loss: 0.0332
Epoch [5/5], Step [5/1200], Loss: 0.1518
```

In [0]:
```
# Test the model on MNIST dataset
model.eval()  # eval mode (batchnorm uses moving mean/variance instead of mini-batch mean/variance)
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in my_test_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Test Accuracy of 250 images trained model on the my dataset: {} %'.format(100 * correct / total))
```

```
Test Accuracy of 250 images trained model on the my dataset: 68.33333333333333 %
```

### 1.3 experiment on 500 size trainning set

In [0]:
```
# Hyper parameters
num_epochs = 5
num_classes = 10
batch_size = 50
learning_rate = 0.001

# Data loader

# loade trainning MNIST set
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)
```

```
# Train the model
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        if i > 9:
            break
        images = images.to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if (i+1) % 1 == 0:
            print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                   .format(epoch+1, num_epochs, i+1, total_step, loss.item()))
```

```
Epoch [1/5], Step [1/1200], Loss: 0.0568
Epoch [1/5], Step [2/1200], Loss: 0.1204
Epoch [1/5], Step [3/1200], Loss: 0.0863
Epoch [1/5], Step [4/1200], Loss: 0.0193
Epoch [1/5], Step [5/1200], Loss: 0.0751
Epoch [1/5], Step [6/1200], Loss: 0.0257
Epoch [1/5], Step [7/1200], Loss: 0.1827
Epoch [1/5], Step [8/1200], Loss: 0.1464
Epoch [1/5], Step [9/1200], Loss: 0.0615
Epoch [1/5], Step [10/1200], Loss: 0.0615
Epoch [2/5], Step [1/1200], Loss: 0.0896
Epoch [2/5], Step [2/1200], Loss: 0.0287
Epoch [2/5], Step [3/1200], Loss: 0.1033
Epoch [2/5], Step [4/1200], Loss: 0.0350
Epoch [2/5], Step [5/1200], Loss: 0.1826
Epoch [2/5], Step [6/1200], Loss: 0.0317
Epoch [2/5], Step [7/1200], Loss: 0.0823
Epoch [2/5], Step [8/1200], Loss: 0.1101
Epoch [2/5], Step [9/1200], Loss: 0.1387
Epoch [2/5], Step [10/1200], Loss: 0.1394
Epoch [3/5], Step [1/1200], Loss: 0.0379
Epoch [3/5], Step [2/1200], Loss: 0.0287
Epoch [3/5], Step [3/1200], Loss: 0.0618
Epoch [3/5], Step [4/1200], Loss: 0.1528
Epoch [3/5], Step [5/1200], Loss: 0.1078
Epoch [3/5], Step [6/1200], Loss: 0.3177
Epoch [3/5], Step [7/1200], Loss: 0.0551
Epoch [3/5], Step [8/1200], Loss: 0.1169
Epoch [3/5], Step [9/1200], Loss: 0.0379
Epoch [3/5], Step [10/1200], Loss: 0.0129
Epoch [4/5], Step [1/1200], Loss: 0.0753
Epoch [4/5], Step [2/1200], Loss: 0.1044
Epoch [4/5], Step [3/1200], Loss: 0.0712
Epoch [4/5], Step [4/1200], Loss: 0.1321
Epoch [4/5], Step [5/1200], Loss: 0.0460
Epoch [4/5], Step [6/1200], Loss: 0.0998
Epoch [4/5], Step [7/1200], Loss: 0.1236
Epoch [4/5], Step [8/1200], Loss: 0.2159
Epoch [4/5], Step [9/1200], Loss: 0.0520
Epoch [4/5], Step [10/1200], Loss: 0.0442
Epoch [5/5], Step [1/1200], Loss: 0.0647
Epoch [5/5], Step [2/1200], Loss: 0.1449
Epoch [5/5], Step [3/1200], Loss: 0.2781
Epoch [5/5], Step [4/1200], Loss: 0.0883
Epoch [5/5], Step [5/1200], Loss: 0.1453
Epoch [5/5], Step [6/1200], Loss: 0.0329
Epoch [5/5], Step [7/1200], Loss: 0.1503
Epoch [5/5], Step [8/1200], Loss: 0.0326
Epoch [5/5], Step [9/1200], Loss: 0.0184
Epoch [5/5], Step [10/1200], Loss: 0.0529
```

In [0]:
```
# Test the model on myDataSet

model.eval()  # eval mode (batchnorm uses moving mean/variance instead of mini-batch mean/variance)
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in my_test_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Test Accuracy of 500 images trained model on the my dataset: {} %'.format(100 * correct / total))
```

```
Test Accuracy of 500 images trained model on the my dataset: 70.0 %
```

## 1.4 experiment on 1000 size trainning set

In [28]:
```
# Hyper parameters
num_epochs = 5
num_classes = 10
batch_size = 50
learning_rate = 0.001

# Data loader

# loade trainning MNIST set
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)

# Train the model
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
```

```
        if i > 19:
            break
        images = images.to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if (i+1) % 1 == 0:
            print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                   .format(epoch+1, num_epochs, i+1, total_step, loss.item()))
```

```
Epoch [1/5], Step [1/1200], Loss: 0.0120
Epoch [1/5], Step [2/1200], Loss: 0.1557
Epoch [1/5], Step [3/1200], Loss: 0.0498
Epoch [1/5], Step [4/1200], Loss: 0.0168
Epoch [1/5], Step [5/1200], Loss: 0.0265
Epoch [1/5], Step [6/1200], Loss: 0.0753
Epoch [1/5], Step [7/1200], Loss: 0.0644
Epoch [1/5], Step [8/1200], Loss: 0.0198
Epoch [1/5], Step [9/1200], Loss: 0.1191
Epoch [1/5], Step [10/1200], Loss: 0.0466
Epoch [1/5], Step [11/1200], Loss: 0.3127
Epoch [1/5], Step [12/1200], Loss: 0.0728
Epoch [1/5], Step [13/1200], Loss: 0.1182
Epoch [1/5], Step [14/1200], Loss: 0.2562
Epoch [1/5], Step [15/1200], Loss: 0.0844
Epoch [1/5], Step [16/1200], Loss: 0.0859
Epoch [1/5], Step [17/1200], Loss: 0.1902
Epoch [1/5], Step [18/1200], Loss: 0.0768
Epoch [1/5], Step [19/1200], Loss: 0.0727
Epoch [1/5], Step [20/1200], Loss: 0.0614
Epoch [2/5], Step [1/1200], Loss: 0.0515
Epoch [2/5], Step [2/1200], Loss: 0.0375
Epoch [2/5], Step [3/1200], Loss: 0.0705
Epoch [2/5], Step [4/1200], Loss: 0.0562
Epoch [2/5], Step [5/1200], Loss: 0.0729
Epoch [2/5], Step [6/1200], Loss: 0.1545
Epoch [2/5], Step [7/1200], Loss: 0.0767
Epoch [2/5], Step [8/1200], Loss: 0.0307
Epoch [2/5], Step [9/1200], Loss: 0.0711
Epoch [2/5], Step [10/1200], Loss: 0.1464
Epoch [2/5], Step [11/1200], Loss: 0.0440
Epoch [2/5], Step [12/1200], Loss: 0.0433
Epoch [2/5], Step [13/1200], Loss: 0.0755
Epoch [2/5], Step [14/1200], Loss: 0.0639
Epoch [2/5], Step [15/1200], Loss: 0.0332
Epoch [2/5], Step [16/1200], Loss: 0.0135
Epoch [2/5], Step [17/1200], Loss: 0.1410
Epoch [2/5], Step [18/1200], Loss: 0.0319
Epoch [2/5], Step [19/1200], Loss: 0.0082
Epoch [2/5], Step [20/1200], Loss: 0.0108
Epoch [3/5], Step [1/1200], Loss: 0.1283
Epoch [3/5], Step [2/1200], Loss: 0.1578
Epoch [3/5], Step [3/1200], Loss: 0.0229
Epoch [3/5], Step [4/1200], Loss: 0.0231
Epoch [3/5], Step [5/1200], Loss: 0.0510
Epoch [3/5], Step [6/1200], Loss: 0.0162
Epoch [3/5], Step [7/1200], Loss: 0.1096
Epoch [3/5], Step [8/1200], Loss: 0.0789
Epoch [3/5], Step [9/1200], Loss: 0.0525
Epoch [3/5], Step [10/1200], Loss: 0.0163
Epoch [3/5], Step [11/1200], Loss: 0.0128
Epoch [3/5], Step [12/1200], Loss: 0.0892
Epoch [3/5], Step [13/1200], Loss: 0.0739
Epoch [3/5], Step [14/1200], Loss: 0.0578
Epoch [3/5], Step [15/1200], Loss: 0.0601
Epoch [3/5], Step [16/1200], Loss: 0.0671
Epoch [3/5], Step [17/1200], Loss: 0.0828
Epoch [3/5], Step [18/1200], Loss: 0.0077
Epoch [3/5], Step [19/1200], Loss: 0.0269
Epoch [3/5], Step [20/1200], Loss: 0.1492
Epoch [4/5], Step [1/1200], Loss: 0.0319
Epoch [4/5], Step [2/1200], Loss: 0.0223
Epoch [4/5], Step [3/1200], Loss: 0.0588
Epoch [4/5], Step [4/1200], Loss: 0.0755
Epoch [4/5], Step [5/1200], Loss: 0.0371
Epoch [4/5], Step [6/1200], Loss: 0.0809
Epoch [4/5], Step [7/1200], Loss: 0.0213
Epoch [4/5], Step [8/1200], Loss: 0.0880
Epoch [4/5], Step [9/1200], Loss: 0.0954
Epoch [4/5], Step [10/1200], Loss: 0.1407
Epoch [4/5], Step [11/1200], Loss: 0.0300
Epoch [4/5], Step [12/1200], Loss: 0.1095
Epoch [4/5], Step [13/1200], Loss: 0.0202
Epoch [4/5], Step [14/1200], Loss: 0.0151
Epoch [4/5], Step [15/1200], Loss: 0.0352
Epoch [4/5], Step [16/1200], Loss: 0.0646
Epoch [4/5], Step [17/1200], Loss: 0.1348
Epoch [4/5], Step [18/1200], Loss: 0.0856
Epoch [4/5], Step [19/1200], Loss: 0.0057
Epoch [4/5], Step [20/1200], Loss: 0.1028
Epoch [5/5], Step [1/1200], Loss: 0.0106
Epoch [5/5], Step [2/1200], Loss: 0.0162
Epoch [5/5], Step [3/1200], Loss: 0.0203
Epoch [5/5], Step [4/1200], Loss: 0.0509
Epoch [5/5], Step [5/1200], Loss: 0.0216
Epoch [5/5], Step [6/1200], Loss: 0.1055
Epoch [5/5], Step [7/1200], Loss: 0.0246
Epoch [5/5], Step [8/1200], Loss: 0.0530
Epoch [5/5], Step [9/1200], Loss: 0.1653
Epoch [5/5], Step [10/1200], Loss: 0.3039
Epoch [5/5], Step [11/1200], Loss: 0.0216
Epoch [5/5], Step [12/1200], Loss: 0.0419
Epoch [5/5], Step [13/1200], Loss: 0.0314
```

```
Epoch [5/5], Step [14/1200], Loss: 0.0686
Epoch [5/5], Step [15/1200], Loss: 0.0600
Epoch [5/5], Step [16/1200], Loss: 0.2127
Epoch [5/5], Step [17/1200], Loss: 0.0256
Epoch [5/5], Step [18/1200], Loss: 0.0451
Epoch [5/5], Step [19/1200], Loss: 0.1293
Epoch [5/5], Step [20/1200], Loss: 0.0513
```

In [29]:
```
# Test the model on myDataSet

model.eval()  # eval mode (batchnorm uses moving mean/variance instead of mini-batch mean/variance)
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in my_test_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Test Accuracy of 50 images trained model on the my dataset: {} %'.format(100 * correct / total))
```

```
Test Accuracy of 50 images trained model on the my dataset: 68.33333333333333 %
```

## 1.5 experiment on 60000 size trainning set

In [0]:
```
# Hyper parameters
num_epochs = 5
num_classes = 10
batch_size = 100
learning_rate = 0.001

# Data loader

# loade trainning MNIST set
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)
# Train the model
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if (i+1) % 100 == 0:
            print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                   .format(epoch+1, num_epochs, i+1, total_step, loss.item()))
```

```
Epoch [1/5], Step [100/600], Loss: 0.2975
Epoch [1/5], Step [200/600], Loss: 0.1002
Epoch [1/5], Step [300/600], Loss: 0.1049
Epoch [1/5], Step [400/600], Loss: 0.0836
Epoch [1/5], Step [500/600], Loss: 0.1222
Epoch [1/5], Step [600/600], Loss: 0.0575
Epoch [2/5], Step [100/600], Loss: 0.0124
Epoch [2/5], Step [200/600], Loss: 0.0734
Epoch [2/5], Step [300/600], Loss: 0.0132
Epoch [2/5], Step [400/600], Loss: 0.0182
Epoch [2/5], Step [500/600], Loss: 0.0949
Epoch [2/5], Step [600/600], Loss: 0.1136
Epoch [3/5], Step [100/600], Loss: 0.0144
Epoch [3/5], Step [200/600], Loss: 0.0502
Epoch [3/5], Step [300/600], Loss: 0.0173
Epoch [3/5], Step [400/600], Loss: 0.0579
Epoch [3/5], Step [500/600], Loss: 0.0267
Epoch [3/5], Step [600/600], Loss: 0.0393
Epoch [4/5], Step [100/600], Loss: 0.0037
Epoch [4/5], Step [200/600], Loss: 0.0056
Epoch [4/5], Step [300/600], Loss: 0.0556
Epoch [4/5], Step [400/600], Loss: 0.0237
Epoch [4/5], Step [500/600], Loss: 0.0268
Epoch [4/5], Step [600/600], Loss: 0.0188
Epoch [5/5], Step [100/600], Loss: 0.0127
Epoch [5/5], Step [200/600], Loss: 0.0079
Epoch [5/5], Step [300/600], Loss: 0.0035
Epoch [5/5], Step [400/600], Loss: 0.0219
Epoch [5/5], Step [500/600], Loss: 0.0375
Epoch [5/5], Step [600/600], Loss: 0.0393
```

In [0]:
```
# Test the model on myDataSet

model.eval()  # eval mode (batchnorm uses moving mean/variance instead of mini-batch mean/variance)
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in my_test_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Test Accuracy of 60000 images trained model on the my dataset: {} %'.format(100 * correct / total))
```

```
Test Accuracy of 60000 images trained model on the my dataset: 73.33333333333333 %
```