CNN

Import MNIST Images - Deep Learning with PyTorch 14

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
from torchvision.utils import make_grid

import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

%matplotlib inline
```

```
# Convert MNIST Image Files into a Tensor of 4-Dimensions (# of images, Height, Width, Color Channel)
transform = transforms.ToTensor()
```

```
# Train Data
train_data = datasets.MNIST(root='/cnn_data', train=True, download=True, transform=transform)
```

```
# test Data
test_data = datasets.MNIST(root='/cnn_data', train=False, download=True, transform=transform)
```

```
train_data
```

```
Dataset MNIST
    Number of datapoints: 60000
    Root location: /cnn_data
    Split: Train
    StandardTransform
Transform: ToTensor()
```

```
test_data
```

```
Dataset MNIST
    Number of datapoints: 10000
    Root location: /cnn_data
    Split: Test
    StandardTransform
Transform: ToTensor()
```

```
pwd
```

```
'/content'
```

```
ls
```

```
sample_data/
```

```
cd ../
```

```
/
```

```
ls -al
```

```
total 116
drwxr-xr-x   1 root root  4096 Oct 17 19:12 ./
drwxr-xr-x   1 root root  4096 Oct 17 19:12 ../
lrwxrwxrwx   1 root root     7 Jun  5 14:02 bin -> usr/bin/
drwxr-xr-x   2 root root  4096 Apr 18  2022 boot/
drwxr-xr-x   3 root root  4096 Oct 17 19:12 cnn_data/
drwxr-xr-x   1 root root  4096 Oct 16 13:23 content/
-rw-r--r--   1 root root  4332 Jun 21 00:40 cuda-keyring_1.0-1_all.deb
drwxr-xr-x   1 root root  4096 Oct 16 13:52 datalab/
drwxr-xr-x   5 root root   360 Oct 17 19:11 dev/
-rwxr-xr-x   1 root root     0 Oct 17 19:11 .dockerenv*
drwxr-xr-x   1 root root  4096 Oct 17 19:11 etc/
drwxr-xr-x   2 root root  4096 Apr 18  2022 home/
lrwxrwxrwx   1 root root     7 Jun  5 14:02 lib -> usr/lib/
lrwxrwxrwx   1 root root     9 Jun  5 14:02 lib32 -> usr/lib32/
lrwxrwxrwx   1 root root     9 Jun  5 14:02 lib64 -> usr/lib64/
lrwxrwxrwx   1 root root    10 Jun  5 14:02 libx32 -> usr/libx32/
drwxr-xr-x   2 root root  4096 Jun  5 14:02 media/
drwxr-xr-x   2 root root  4096 Jun  5 14:02 mnt/
-rw-r--r--   1 root root 17294 Jun 21 00:39 NGC-DL-CONTAINER-LICENSE
drwxr-xr-x   1 root root  4096 Oct 16 13:53 opt/
dr-xr-xr-x 175 root root     0 Oct 17 19:11 proc/
drwxr-xr-x  15 root root  4096 Oct 16 13:20 python-apt/
drwx------   1 root root  4096 Oct 16 13:53 root/
drwxr-xr-x   1 root root  4096 Oct 16 13:15 run/
lrwxrwxrwx   1 root root     8 Jun  5 14:02 sbin -> usr/sbin/
drwxr-xr-x   2 root root  4096 Jun  5 14:02 srv/
dr-xr-xr-x  13 root root     0 Oct 17 19:11 sys/
drwxrwxrwt   1 root root  4096 Oct 17 19:11 tmp/
drwxr-xr-x   1 root root  4096 Oct 16 13:39 tools/
drwxr-xr-x   1 root root  4096 Oct 16 13:53 usr/
drwxr-xr-x   1 root root  4096 Oct 16 13:52 var/
```

```
cd cnn_data
```

```
/cnn_data
```

```
ls -l
```

```
total 4
drwxr-xr-x 3 root root 4096 Oct 17 19:12 MNIST/
```

```
cd /
```

```
/
```

```
ls -l
```

```
total 108
lrwxrwxrwx   1 root root     7 Jun  5 14:02 bin -> usr/bin/
drwxr-xr-x   2 root root  4096 Apr 18  2022 boot/
drwxr-xr-x   3 root root  4096 Oct 17 19:12 cnn_data/
drwxr-xr-x   1 root root  4096 Oct 16 13:23 content/
-rw-r--r--   1 root root  4332 Jun 21 00:40 cuda-keyring_1.0-1_all.deb
drwxr-xr-x   1 root root  4096 Oct 16 13:52 datalab/
```

```
drwxr-xr-x   5 root root   360 Oct 17 19:11 dev/
drwxr-xr-x   1 root root  4096 Oct 17 19:11 etc/
drwxr-xr-x   2 root root  4096 Apr 18  2022 home/
lrwxrwxrwx   1 root root     7 Jun  5 14:02 lib -> usr/lib/
lrwxrwxrwx   1 root root     9 Jun  5 14:02 lib32 -> usr/lib32/
lrwxrwxrwx   1 root root     9 Jun  5 14:02 lib64 -> usr/lib64/
lrwxrwxrwx   1 root root    10 Jun  5 14:02 libx32 -> usr/libx32/
drwxr-xr-x   2 root root  4096 Jun  5 14:02 media/
drwxr-xr-x   2 root root  4096 Jun  5 14:02 mnt/
-rw-r--r--   1 root root 17294 Jun 21 00:39 NGC-DL-CONTAINER-LICENSE
drwxr-xr-x   1 root root  4096 Oct 16 13:53 opt/
dr-xr-xr-x 175 root root     0 Oct 17 19:11 proc/
drwxr-xr-x  15 root root  4096 Oct 16 13:20 python-apt/
drwx------   1 root root  4096 Oct 16 13:53 root/
drwxr-xr-x   1 root root  4096 Oct 16 13:15 run/
lrwxrwxrwx   1 root root     8 Jun  5 14:02 sbin -> usr/sbin/
drwxr-xr-x   2 root root  4096 Jun  5 14:02 srv/
dr-xr-xr-x  13 root root     0 Oct 17 19:11 sys/
drwxrwxrwt   1 root root  4096 Oct 17 19:11 tmp/
drwxr-xr-x   1 root root  4096 Oct 16 13:39 tools/
drwxr-xr-x   1 root root  4096 Oct 16 13:53 usr/
drwxr-xr-x   1 root root  4096 Oct 16 13:52 var/
```

```
cd content/
```

```
/content
```

```
ls -al
```

```
total 16
drwxr-xr-x 1 root root 4096 Oct 16 13:23 ./
drwxr-xr-x 1 root root 4096 Oct 17 19:12 ../
drwxr-xr-x 4 root root 4096 Oct 16 13:23 .config/
drwxr-xr-x 1 root root 4096 Oct 16 13:23 sample_data/
```

Convolutional and Pooling Layers - Deep Learning with PyTorch 15

Double-cliquez (ou appuyez sur Entrée) pour modifier

```
# Create a small batch size for images...  let's say 10
train_loader = DataLoader(train_data, batch_size=10, shuffle=True)
test_loader = DataLoader(test_data, batch_size=10, shuffle=False)
```

```
# Define the CNN Model
# Decribe the convolutional layer and what it's doing (2 convolutional layers)
# This is an example
conv1 = nn.Conv2d(1, 6, 3, 1)
conv2 = nn.Conv2d(in_channels=6, out_channels=16, kernel_size=3, stride=1)

```

```
# Grab 1 MNIST record/image
for i, (X_train, y_train) in enumerate(train_data):
  break
```

```
X_train
```

```
tensor([[[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0118, 0.0706, 0.0706, 0.0706,
          0.4941, 0.5333, 0.6863, 0.1020, 0.6510, 1.0000, 0.9686, 0.4980,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.1176, 0.1412, 0.3686, 0.6039, 0.6667, 0.9922, 0.9922, 0.9922,
          0.9922, 0.9922, 0.8824, 0.6745, 0.9922, 0.9490, 0.7647, 0.2510,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.1922,
          0.9333, 0.9922, 0.9922, 0.9922, 0.9922, 0.9922, 0.9922, 0.9922,
          0.9922, 0.9843, 0.3647, 0.3216, 0.3216, 0.2196, 0.1529, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0706,
          0.8588, 0.9922, 0.9922, 0.9922, 0.9922, 0.9922, 0.7765, 0.7137,
          0.9686, 0.9451, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.3137, 0.6118, 0.4196, 0.9922, 0.9922, 0.8039, 0.0431, 0.0000,
          0.1686, 0.6039, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0549, 0.0039, 0.6039, 0.9922, 0.3529, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.5451, 0.9922, 0.7451, 0.0078, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0431, 0.7451, 0.9922, 0.2745, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.1373, 0.9451, 0.8824, 0.6275,
          0.4235, 0.0039, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.3176, 0.9412, 0.9922,
```

```
X_train.shape
```

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

```
x = X_train.view(1,1, 28, 28)
```

```
# Perform the first convolution
x = F.relu(conv1(x)) # Rectified Linear Unit for the activation function
```

```
x
```

```
tensor([[[[0.1709, 0.1709, 0.1709,  ..., 0.1709, 0.1709, 0.1709],
          [0.1709, 0.1709, 0.1709,  ..., 0.1709, 0.1709, 0.1709],
          [0.1709, 0.1709, 0.1709,  ..., 0.1709, 0.1709, 0.1709],
          ...,
          [0.1709, 0.1709, 0.1158,  ..., 0.1709, 0.1709, 0.1709],
          [0.1709, 0.1709, 0.1537,  ..., 0.1709, 0.1709, 0.1709],
          [0.1709, 0.1709, 0.1709,  ..., 0.1709, 0.1709, 0.1709]],

         [[0.0006, 0.0006, 0.0006,  ..., 0.0006, 0.0006, 0.0006],
          [0.0006, 0.0006, 0.0006,  ..., 0.0006, 0.0006, 0.0006],
          [0.0006, 0.0006, 0.0006,  ..., 0.0006, 0.0006, 0.0006],
          ...,
          [0.0006, 0.0006, 0.1298,  ..., 0.0006, 0.0006, 0.0006],
          [0.0006, 0.0006, 0.0454,  ..., 0.0006, 0.0006, 0.0006],
          [0.0006, 0.0006, 0.0006,  ..., 0.0006, 0.0006, 0.0006]],

         [[0.1699, 0.1699, 0.1699,  ..., 0.1699, 0.1699, 0.1699],
          [0.1699, 0.1699, 0.1699,  ..., 0.1699, 0.1699, 0.1699],
          [0.1699, 0.1699, 0.1699,  ..., 0.1699, 0.1699, 0.1699],
          ...,
          [0.1699, 0.1699, 0.3766,  ..., 0.1699, 0.1699, 0.1699],
          [0.1699, 0.1699, 0.3260,  ..., 0.1699, 0.1699, 0.1699],
          [0.1699, 0.1699, 0.1699,  ..., 0.1699, 0.1699, 0.1699]],

         [[0.0000, 0.0000, 0.0000,  ..., 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000,  ..., 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000,  ..., 0.0000, 0.0000, 0.0000],
          ...,
          [0.0000, 0.0000, 0.0000,  ..., 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000,  ..., 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000,  ..., 0.0000, 0.0000, 0.0000]],

         [[0.0591, 0.0591, 0.0591,  ..., 0.0591, 0.0591, 0.0591],
          [0.0591, 0.0591, 0.0591,  ..., 0.0591, 0.0591, 0.0591],
          [0.0591, 0.0591, 0.0591,  ..., 0.0591, 0.0591, 0.0591],
          ...,
          [0.0591, 0.0591, 0.0000,  ..., 0.0591, 0.0591, 0.0591],
          [0.0591, 0.0591, 0.1010,  ..., 0.0591, 0.0591, 0.0591],
          [0.0591, 0.0591, 0.0591,  ..., 0.0591, 0.0591, 0.0591]],

         [[0.3034, 0.3034, 0.3034,  ..., 0.3034, 0.3034, 0.3034],
          [0.3034, 0.3034, 0.3034,  ..., 0.3034, 0.3034, 0.3034],
          [0.3034, 0.3034, 0.3034,  ..., 0.3034, 0.3034, 0.3034],
          ...,
          [0.3034, 0.3034, 0.4017,  ..., 0.3034, 0.3034, 0.3034],
          [0.3034, 0.3034, 0.4095,  ..., 0.3034, 0.3034, 0.3034],
          [0.3034, 0.3034, 0.3034,  ..., 0.3034, 0.3034, 0.3034]]]],
       grad_fn=<ReluBackward0>)
```

```
# 1 is the single image, 6 is the filters asked for, 26x26
x.shape
```

```
torch.Size([1, 6, 26, 26])
```

```
# Pass through the pooling layer
x = F.max_pool2d(x, 2, 2) # Kernel of 2 and stride of 2
```

```
x.shape # 26 / 2 = 13
```

```
torch.Size([1, 6, 13, 13])
```

```
# Do the second convolutional layer
x = F.relu(conv2(x))
```

```
x.shape # Again, no padding was specified so 2 pixels were lost around the outside of the image
```

```
torch.Size([1, 16, 11, 11])
```

```
# Pooling layer
x = F.max_pool2d(x, 2, 2)
```

```
x.shape # 11 / 2 = 5.5 but it is rounded down because no data can invented to round up
```

```
torch.Size([1, 16, 5, 5])
```

```
(((28-2) / 2) -2) / 2
```

```
5.5
```

Convolutional Neural Network Model - Deep Learning with PyTorch 16

```
# Model Class
class ConvolutionalNetwork(nn.Module):
  def __init__(self) -> None:
    super().__init__()
    self.conv1 = nn.Conv2d(1, 6, 3, 1)
    self.conv2 = nn.Conv2d(6, 16, 3, 1)

    # Fully Connected Layers
    self.fc1 = nn.Linear(5*5*16, 120)
    self.fc2 = nn.Linear(120, 84)
    self.fc3 = nn.Linear(84, 10)

  def forward(self, X):
    X = F.relu(self.conv1(X))
    X = F.max_pool2d(X, 2, 2) # 2x2 kernel and stride = 2
    # Second pass
    X = F.relu(self.conv2(X))
    X = F.max_pool2d(X, 2, 2) # 2x2 kernel and stride = 2

    # Re-View the data to flatten it out
    X = X.view(-1, 16*5*5) # Negative one so the batch size can be varied

    # Fully Connected Layers
    X = F.relu(self.fc1(X))
    X = F.relu(self.fc2(X))
    X = self.fc3(X)

    return F.log_softmax(X, dim=1)


```

```
# Create an Instance of the Model
torch.manual_seed(41)
model = ConvolutionalNetwork()
```

```
model
```

```
ConvolutionalNetwork(
  (conv1): Conv2d(1, 6, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(6, 16, kernel_size=(3, 3), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
```

```
# Loss Function Optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001) # The smaller the learning rate, the longer it's g
```

Train and Test CNN Model - Deep Learning with PyTorch 17

```
import time
start_time = time.time()

# Create Variables to track things
epochs = 5
train_losses = []
test_losses = []
train_correct = []
test_correct = []


# For Loop offor Epochs
for i in range(epochs):
  training_correct = 0
  testing_correct = 0

  # Train
  for b, (X_train, y_train) in enumerate(train_loader):
    b += 1 # Start the batches at 1

    y_pred = model(X_train) # Get the predicted values from the training set (data is 2d, not flattened.)
    loss = criterion(y_pred, y_train) # How off are we? Compare the predictions to the correct answers in y_t

    predicted = torch.max(y_pred.data, 1)[1] # Add up the number of correct predictions. Indexed off the firs
    batch_correct = (predicted == y_train).sum() # How many we got correct from this specific batch. True=1,
    training_correct += batch_correct # Keep track as we go along in training.

    # Update the parameters
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # Print out some results
    if b % 600 == 0 :
      print(f'Epoch: {i} Batch: {b} Loss: {loss.item()}')


  train_losses.append(loss)
  train_correct.append(training_correct)

  # Test
  with torch.no_grad(): # No gradient so the weights and the bias are not updated with test data
    for b , (X_test, y_test) in enumerate(test_loader):
      y_val = model(X_test)
      predicted = torch.max(y_val.data, 1)[1] # Adding up correct predictions
      testing_correct += (predicted == y_test).sum() # True=1, False=0, sum all

  loss = criterion(y_val, y_test)
  test_losses.append(loss)
  test_correct.append(testing_correct)

current_time = time.time()
total = current_time - start_time
print(f'Training time: {total/60} minutes!')
total
```

```
Epoch: 0 Batch: 600 Loss: 4.9471535021439195e-05
Epoch: 0 Batch: 1200 Loss: 1.8047001503873616e-05
Epoch: 0 Batch: 1800 Loss: 8.344646573732462e-08
Epoch: 0 Batch: 2400 Loss: 0.003162816632539034
Epoch: 0 Batch: 3000 Loss: 1.1622306374192704e-05
Epoch: 0 Batch: 3600 Loss: 0.0001164354252978228
Epoch: 0 Batch: 4200 Loss: 9.417489081897656e-07
Epoch: 0 Batch: 4800 Loss: 0.00047675552195869386
Epoch: 0 Batch: 5400 Loss: 0.0
Epoch: 0 Batch: 6000 Loss: 1.609314722372801e-06
Epoch: 1 Batch: 600 Loss: 2.777537929432583e-06
Epoch: 1 Batch: 1200 Loss: 1.5139510196604533e-06
Epoch: 1 Batch: 1800 Loss: 2.731903805397451e-05
Epoch: 1 Batch: 2400 Loss: 1.889325176307466e-05
Epoch: 1 Batch: 3000 Loss: 1.4351843674376141e-05
Epoch: 1 Batch: 3600 Loss: 9.142941962636542e-06
Epoch: 1 Batch: 4200 Loss: 8.943313878262416e-05
Epoch: 1 Batch: 4800 Loss: 0.00012190106644993648
Epoch: 1 Batch: 5400 Loss: 1.3207888514443766e-05
Epoch: 1 Batch: 6000 Loss: 0.0
Epoch: 2 Batch: 600 Loss: 2.1097897842992097e-05
Epoch: 2 Batch: 1200 Loss: 0.0
Epoch: 2 Batch: 1800 Loss: 8.022654583328404e-06
Epoch: 2 Batch: 2400 Loss: 0.0
Epoch: 2 Batch: 3000 Loss: 0.0
Epoch: 2 Batch: 3600 Loss: 0.0013853703858330846
Epoch: 2 Batch: 4200 Loss: 8.702245395397767e-07
Epoch: 2 Batch: 4800 Loss: 0.0
Epoch: 2 Batch: 5400 Loss: 5.555012648983393e-06
Epoch: 2 Batch: 6000 Loss: 3.576274423267023e-07
Epoch: 3 Batch: 600 Loss: 9.775114904186921e-07
Epoch: 3 Batch: 1200 Loss: 1.1514954167068936e-05
Epoch: 3 Batch: 1800 Loss: 7.748575399091351e-07
Epoch: 3 Batch: 2400 Loss: 2.2887920749781188e-06
Epoch: 3 Batch: 3000 Loss: 1.192092824453539e-08
Epoch: 3 Batch: 3600 Loss: 0.0
Epoch: 3 Batch: 4200 Loss: 0.002368538174778223
Epoch: 3 Batch: 4800 Loss: 0.0001560908422106877
Epoch: 3 Batch: 5400 Loss: 4.863749927608296e-05
Epoch: 3 Batch: 6000 Loss: 1.5114685993466992e-05
Epoch: 4 Batch: 600 Loss: 0.0
Epoch: 4 Batch: 1200 Loss: 5.960462701182223e-08
Epoch: 4 Batch: 1800 Loss: 2.384185648907078e-08
Epoch: 4 Batch: 2400 Loss: 1.2159273410361493e-06
Epoch: 4 Batch: 3000 Loss: 0.0002526980242691934
Epoch: 4 Batch: 3600 Loss: 0.0036067436449229717
Epoch: 4 Batch: 4200 Loss: 0.0
Epoch: 4 Batch: 4800 Loss: 3.6161560274194926e-05
Epoch: 4 Batch: 5400 Loss: 0.4160960614681244
Epoch: 4 Batch: 6000 Loss: 4.497986446949653e-05
Training time: 4.19429939587911 minutes!
251.65796375274658
```

```

import time
start_time = time.time()
```

```

# Create Variables To Tracks Things
epochs = 5
train_losses = []
test_losses = []
train_correct = []
test_correct = []

# For Loop of Epochs
for i in range(epochs):
  trn_corr = 0
  tst_corr = 0


  # Train
  for b,(X_train, y_train) in enumerate(train_loader):
    b+=1 # start our batches at 1
    y_pred = model(X_train) # get predicted values from the training set. Not flattened 2D
    loss = criterion(y_pred, y_train) # how off are we? Compare the predictions to correct answers in y_trair

    predicted = torch.max(y_pred.data, 1)[1] # add up the number of correct predictions. Indexed off the firs
    batch_corr = (predicted == y_train).sum() # how many we got correct from this batch. True = 1, False=0, s
    trn_corr += batch_corr # keep track as we go along in training.

    # Update our parameters
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()


    # Print out some results
    if b%600 == 0:
      print(f'Epoch: {i}  Batch: {b}  Loss: {loss.item()}')

  train_losses.append(loss)
  train_correct.append(trn_corr)


  # Test
  with torch.no_grad(): #No gradient so we don't update our weights and biases with test data
    for b,(X_test, y_test) in enumerate(test_loader):
      y_val = model(X_test)
      predicted = torch.max(y_val.data, 1)[1] # Adding up correct predictions
      tst_corr += (predicted == y_test).sum() # T=1 F=0 and sum away


  loss = criterion(y_val, y_test)
  test_losses.append(loss)
  test_correct.append(tst_corr)



current_time = time.time()
total = current_time - start_time
print(f'Training Took: {total/60} minutes!')
```

```
Epoch: 0  Batch: 600  Loss: 0.0027929155621677637
Epoch: 0  Batch: 1200  Loss: 3.493723488645628e-05
Epoch: 0  Batch: 1800  Loss: 0.0010003604693338275
Epoch: 0  Batch: 2400  Loss: 0.1295616179704666
Epoch: 0  Batch: 3000  Loss: 1.3505514289136045e-05
Epoch: 0  Batch: 3600  Loss: 0.00021176428708713502
Epoch: 0  Batch: 4200  Loss: 0.00011720164911821485
Epoch: 0  Batch: 4800  Loss: 0.0009881147416308522
Epoch: 0  Batch: 5400  Loss: 2.384185648907078e-08
Epoch: 0  Batch: 6000  Loss: 0.0
Epoch: 1  Batch: 600  Loss: 0.0008038681116886437
```

```
Epoch: 1  Batch: 1200  Loss: 4.76837058727142e-08
Epoch: 1  Batch: 1800  Loss: 3.4093500289600343e-06
Epoch: 1  Batch: 2400  Loss: 3.4710730687947944e-05
Epoch: 1  Batch: 3000  Loss: 0.0002807883720379323
Epoch: 1  Batch: 3600  Loss: 9.417489081897656e-07
Epoch: 1  Batch: 4200  Loss: 0.008015107363462448
Epoch: 1  Batch: 4800  Loss: 0.00021272152662277222
Epoch: 1  Batch: 5400  Loss: 7.617282335559139e-06
Epoch: 1  Batch: 6000  Loss: 7.709871715633199e-05
Epoch: 2  Batch: 600  Loss: 1.1241128959227353e-05
Epoch: 2  Batch: 1200  Loss: 9.536717584524013e-07
Epoch: 2  Batch: 1800  Loss: 0.0005367971025407314
Epoch: 2  Batch: 2400  Loss: 4.148400421399856e-06
Epoch: 2  Batch: 3000  Loss: 0.0
Epoch: 2  Batch: 3600  Loss: 0.07778234779834747
Epoch: 2  Batch: 4200  Loss: 1.001354917207209e-06
Epoch: 2  Batch: 4800  Loss: 0.042573653161525726
Epoch: 2  Batch: 5400  Loss: 0.0
Epoch: 2  Batch: 6000  Loss: 0.00151331617962569
Epoch: 3  Batch: 600  Loss: 1.4113868928689044e-05
Epoch: 3  Batch: 1200  Loss: 2.2530307433044072e-06
Epoch: 3  Batch: 1800  Loss: 2.384185471271394e-08
Epoch: 3  Batch: 2400  Loss: 3.576272717964457e-07
Epoch: 3  Batch: 3000  Loss: 2.384185648907078e-08
Epoch: 3  Batch: 3600  Loss: 0.054267413914203644
Epoch: 3  Batch: 4200  Loss: 0.680134654045105
Epoch: 3  Batch: 4800  Loss: 0.0008131394279189408
Epoch: 3  Batch: 5400  Loss: 1.3255626072350424e-05
Epoch: 3  Batch: 6000  Loss: 1.0728830091011332e-07
Epoch: 4  Batch: 600  Loss: 0.000250897224759683
Epoch: 4  Batch: 1200  Loss: 0.0004779839946422726
Epoch: 4  Batch: 1800  Loss: 0.0010529584251344204
Epoch: 4  Batch: 2400  Loss: 0.001856612740084529
Epoch: 4  Batch: 3000  Loss: 1.2707006135315169e-05
Epoch: 4  Batch: 3600  Loss: 0.0
Epoch: 4  Batch: 4200  Loss: 1.192092824453539e-08
Epoch: 4  Batch: 4800  Loss: 3.576278118089249e-08
Epoch: 4  Batch: 5400  Loss: 2.0265562739041343e-07
Epoch: 4  Batch: 6000  Loss: 0.0012134775752201676
Training Took: 4.140049517154694 minutes!
```