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)
```

```
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 /cnn_data/MNIST/raw/train-imag
100%|██████████| 9912422/9912422 [00:00<00:00, 35955319.52it/s]
Extracting /cnn_data/MNIST/raw/train-images-idx3-ubyte.gz to /cnn_data/MNIST/raw

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 /cnn_data/MNIST/raw/train-labe
100%|██████████| 28881/28881 [00:00<00:00, 109724360.35it/s]
Extracting /cnn_data/MNIST/raw/train-labels-idx1-ubyte.gz to /cnn_data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to /cnn_data/MNIST/raw/t10k-images
100%|██████████| 1648877/1648877 [00:00<00:00, 29369586.10it/s]
Extracting /cnn_data/MNIST/raw/t10k-images-idx3-ubyte.gz to /cnn_data/MNIST/raw

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 /cnn_data/MNIST/raw/t10k-labels
100%|██████████| 4542/4542 [00:00<00:00, 24053697.94it/s]
Extracting /cnn_data/MNIST/raw/t10k-labels-idx1-ubyte.gz to /cnn_data/MNIST/raw
```

```
# 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()
```

```
1 pwd
```

```
'/content'
```

```
1 ls
```

```
sample_data/
```

```
1 cd ../
```

```
/
```

```
1 ls -al
```

```
total 116
drwxr-xr-x   1 root root  4096 Oct 18 11:36 ./
drwxr-xr-x   1 root root  4096 Oct 18 11:36 ../
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 18 11:36 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   6 root root   460 Oct 18 11:35 dev/
-rwxr-xr-x   1 root root     0 Oct 18 11:35 .dockerenv*
drwxr-xr-x   1 root root  4096 Oct 18 11:35 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 18 11:35 opt/
dr-xr-xr-x 202 root root     0 Oct 18 11:35 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 18 11:35 sys/
drwxrwxrwt   1 root root  4096 Oct 18 11:35 tmp/
drwxr-xr-x   1 root root  4096 Oct 16 13:39 tools/
drwxr-xr-x   1 root root  4096 Oct 18 11:35 usr/
drwxr-xr-x   1 root root  4096 Oct 16 13:52 var/
```

```
1 cd cnn_data
```

```
/cnn_data
```

```
1 ls -l
```

```
total 4
drwxr-xr-x 3 root root 4096 Oct 18 11:36 MNIST/
```

```
1 cd /
```

```
/
```

```
1 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 18 11:36 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   6 root root   460 Oct 18 11:35 dev/
drwxr-xr-x   1 root root  4096 Oct 18 11:35 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 18 11:35 opt/
dr-xr-xr-x 202 root root     0 Oct 18 11:35 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 18 11:35 sys/
drwxrwxrwt   1 root root  4096 Oct 18 11:35 tmp/
drwxr-xr-x   1 root root  4096 Oct 16 13:39 tools/
drwxr-xr-x   1 root root  4096 Oct 18 11:35 usr/
drwxr-xr-x   1 root root  4096 Oct 16 13:52 var/
```

```
1 cd content/
```

```
/content
```

```
1 ls -al
```

```
total 16
drwxr-xr-x 1 root root 4096 Oct 16 13:23 ./
drwxr-xr-x 1 root root 4096 Oct 18 11:36 ../
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

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

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

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

```
1 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([[[[0.0000, 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]],

         [[0.0920, 0.0920, 0.0920,  ..., 0.0920, 0.0920, 0.0920],
          [0.0920, 0.0920, 0.0920,  ..., 0.0920, 0.0920, 0.0920],
          [0.0920, 0.0920, 0.0920,  ..., 0.0920, 0.0920, 0.0920],
          ...,
          [0.0920, 0.0920, 0.1303,  ..., 0.0920, 0.0920, 0.0920],
          [0.0920, 0.0920, 0.0000,  ..., 0.0920, 0.0920, 0.0920],
          [0.0920, 0.0920, 0.0920,  ..., 0.0920, 0.0920, 0.0920]],

         [[0.1114, 0.1114, 0.1114,  ..., 0.1114, 0.1114, 0.1114],
          [0.1114, 0.1114, 0.1114,  ..., 0.1114, 0.1114, 0.1114],
          [0.1114, 0.1114, 0.1114,  ..., 0.1114, 0.1114, 0.1114],
          ...,
          [0.1114, 0.1114, 0.1431,  ..., 0.1114, 0.1114, 0.1114],
          [0.1114, 0.1114, 0.2564,  ..., 0.1114, 0.1114, 0.1114],
          [0.1114, 0.1114, 0.1114,  ..., 0.1114, 0.1114, 0.1114]],

         [[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]]]],
       grad_fn=<ReluBackward0>)
```

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

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

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

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

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

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

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

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

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

```
1 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])
```

```
1 (((28-2) / 2) -2) / 2
```

```
5.5
```

Convolutional Neural Network Model - Deep Learning with PyTorch 16

```
1 # Model Class
2 class ConvolutionalNetwork(nn.Module):
3   def __init__(self) -> None:
4     super().__init__()
5     self.conv1 = nn.Conv2d(1, 6, 3, 1)
6     self.conv2 = nn.Conv2d(6, 16, 3, 1)
7
8     # Fully Connected Layers
9     self.fc1 = nn.Linear(5*5*16, 120)
10    self.fc2 = nn.Linear(120, 84)
11    self.fc3 = nn.Linear(84, 10)
12
13  def forward(self, X):
14    X = F.relu(self.conv1(X))
15    X = F.max_pool2d(X, 2, 2) # 2x2 kernel and stride = 2
16    # Second pass
17    X = F.relu(self.conv2(X))
18    X = F.max_pool2d(X, 2, 2) # 2x2 kernel and stride = 2
19
20    # Re-View the data to flatten it out
21    X = X.view(-1, 16*5*5) # Negative one so the batch size can be varied
22
23    # Fully Connected Layers
24    X = F.relu(self.fc1(X))
25    X = F.relu(self.fc2(X))
26    X = self.fc3(X)
27
28    return F.log_softmax(X, dim=1)
29
30
```

```
1  # Create an Instance of the Model
2  torch.manual_seed(41)
3  model = ConvolutionalNetwork()
4  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)
)
```

```
1  # Loss Function Optimizer
2  criterion = nn.CrossEntropyLoss()
3  optimizer = torch.optim.Adam(model.parameters(), lr=0.001) # The smaller the learning rate, the longer it's going to take t
```

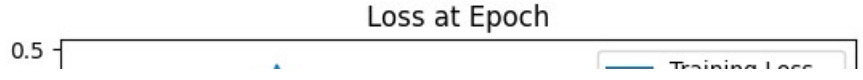
Train and Test CNN Model - Deep Learning with PyTorch 17

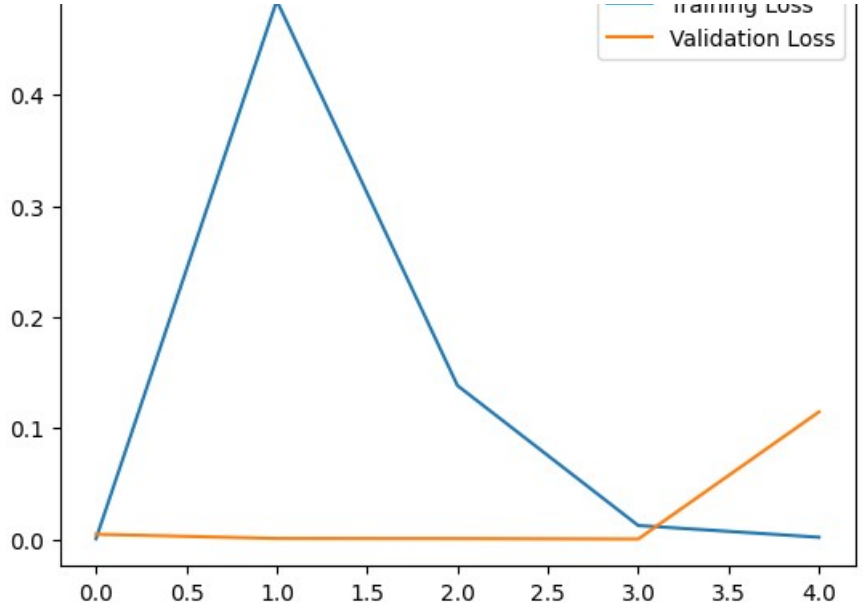
```
import time
start_time = time.time()

# 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_train

    predicted = torch.max(y_pred.data, 1)[1] # Add up the number of correct predictions. Indexed off the first point
    batch_correct = (predicted == y_train).sum() # How many we got correct from this specific batch. True=1, False=0, sum th
    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: 0.1623624861240387
Epoch: 0 Batch: 1200 Loss: 0.1641543209552765
Epoch: 0 Batch: 1800 Loss: 0.5098981857299805
Epoch: 0 Batch: 2400 Loss: 0.1306418627500534
Epoch: 0 Batch: 3000 Loss: 0.005703817121684551
Epoch: 0 Batch: 3600 Loss: 0.46332210302352905
Epoch: 0 Batch: 4200 Loss: 0.004197881557047367
Epoch: 0 Batch: 4800 Loss: 0.0018000779673457146
Epoch: 0 Batch: 5400 Loss: 0.07375213503837585
Epoch: 0 Batch: 6000 Loss: 0.0003859388525597751
Epoch: 1 Batch: 600 Loss: 0.004290326032787561
Epoch: 1 Batch: 1200 Loss: 0.2521086633205414
Epoch: 1 Batch: 1800 Loss: 0.002427826402708888
Epoch: 1 Batch: 2400 Loss: 0.0021776421926915646
Epoch: 1 Batch: 3000 Loss: 0.02223074808716774
Epoch: 1 Batch: 3600 Loss: 0.6111965179443359
Epoch: 1 Batch: 4200 Loss: 0.016707444563508034
Epoch: 1 Batch: 4800 Loss: 0.0006908098584972322
Epoch: 1 Batch: 5400 Loss: 0.0002799260546453297
Epoch: 1 Batch: 6000 Loss: 0.4848875403404236
Epoch: 2 Batch: 600 Loss: 0.03840283304452896
Epoch: 2 Batch: 1200 Loss: 0.005653898231685162
Epoch: 2 Batch: 1800 Loss: 0.0019390363013371825
Epoch: 2 Batch: 2400 Loss: 0.013562331907451153
Epoch: 2 Batch: 3000 Loss: 0.004443833604454994
Epoch: 2 Batch: 3600 Loss: 0.0005063370917923748
Epoch: 2 Batch: 4200 Loss: 0.04586367309093475
Epoch: 2 Batch: 4800 Loss: 0.006673174444586039
Epoch: 2 Batch: 5400 Loss: 0.02076614275574684
Epoch: 2 Batch: 6000 Loss: 0.13829907774925232
Epoch: 3 Batch: 600 Loss: 0.0021926886402070522
Epoch: 3 Batch: 1200 Loss: 0.10718250274658203
Epoch: 3 Batch: 1800 Loss: 0.0005934062064625323
Epoch: 3 Batch: 2400 Loss: 0.00016517048061359674
Epoch: 3 Batch: 3000 Loss: 0.0017503199633210897
Epoch: 3 Batch: 3600 Loss: 0.0007489544805139303
Epoch: 3 Batch: 4200 Loss: 0.01164452824741602
Epoch: 3 Batch: 4800 Loss: 7.926556281745434e-05
Epoch: 3 Batch: 5400 Loss: 0.039218269288539886
Epoch: 3 Batch: 6000 Loss: 0.012269356288015842
Epoch: 4 Batch: 600 Loss: 0.019970223307609558
Epoch: 4 Batch: 1200 Loss: 0.031921446323394775
Epoch: 4 Batch: 1800 Loss: 0.2468046396970749
Epoch: 4 Batch: 2400 Loss: 0.00011870403250213712
Epoch: 4 Batch: 3000 Loss: 0.0005112775252200663
Epoch: 4 Batch: 3600 Loss: 0.00011618930147960782
Epoch: 4 Batch: 4200 Loss: 0.00035029969876632094
Epoch: 4 Batch: 4800 Loss: 0.04948687180876732
Epoch: 4 Batch: 5400 Loss: 0.03161567822098732
Epoch: 4 Batch: 6000 Loss: 0.0017111159395426512
Training time: 2.9672022223472596 minutes!
178.03213334083557
```

```
1 # Graph the loss of Epoch
2 train_losses = [tl.item() for tl in train_losses]
3 plt.plot(train_losses, label='Training Loss')
4 plt.plot(test_losses, label='Validation Loss')
5 plt.title('Loss at Epoch')
6 plt.legend()
```

```
<matplotlib.legend.Legend at 0x7c7fdfc48400>
```






```
1
2   import time
3   start_time = time.time()
4
5   # Create Variables To Tracks Things
6   epochs = 5
7   train_losses = []
8   test_losses = []
9   train_correct = []
10  test_correct = []
11
12  # For Loop of Epochs
13  for i in range(epochs):
14    trn_corr = 0
15    tst_corr = 0
16
17
18    # Train
19    for b,(X_train, y_train) in enumerate(train_loader):
20      b+=1 # start our batches at 1
21      y_pred = model(X_train) # get predicted values from the training set. Not flattened 2D
22      loss = criterion(y_pred, y_train) # how off are we? Compare the predictions to correct answers in y_train
23
24      predicted = torch.max(y_pred.data, 1)[1] # add up the number of correct predictions. Indexed off the first point
25      batch_corr = (predicted == y_train).sum() # how many we got correct from this batch. True = 1, False=0, sum those up
26      trn_corr += batch_corr # keep track as we go along in training.
27
28      # Update our parameters
29      optimizer.zero_grad()
30      loss.backward()
31      optimizer.step()
32
33
34      # Print out some results
35      if b%600 == 0:
36        print(f'Epoch: {i}  Batch: {b}  Loss: {loss.item()}')
37
38    train_losses.append(loss)
39    train_correct.append(trn_corr)
40
41
42    # Test
43    with torch.no_grad(): #No gradient so we don't update our weights and biases with test data
44      for b,(X_test, y_test) in enumerate(test_loader):
45        y_val = model(X_test)
46        predicted = torch.max(y_val.data, 1)[1] # Adding up correct predictions
47        tst_corr += (predicted == y_test).sum() # T=1 F=0 and sum away
48
49
50    loss = criterion(y_val, y_test)
51    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: 0.1623624861240387
Epoch: 0  Batch: 1200  Loss: 0.1641543209552765
Epoch: 0  Batch: 1800  Loss: 0.5098981857299805
Epoch: 0  Batch: 2400  Loss: 0.1306418627500534
Epoch: 0  Batch: 3000  Loss: 0.005703817121684551
Epoch: 0  Batch: 3600  Loss: 0.46332210302352905
Epoch: 0  Batch: 4200  Loss: 0.004197881557047367
Epoch: 0  Batch: 4800  Loss: 0.0018000779673457146
Epoch: 0  Batch: 5400  Loss: 0.07375213503837585
Epoch: 0  Batch: 6000  Loss: 0.0003859388525597751
Epoch: 1  Batch: 600  Loss: 0.004290326032787561
Epoch: 1  Batch: 1200  Loss: 0.2521086633205414
Epoch: 1  Batch: 1800  Loss: 0.002427826402708888
Epoch: 1  Batch: 2400  Loss: 0.0021776421926915646
Epoch: 1  Batch: 3000  Loss: 0.02223074808716774
Epoch: 1  Batch: 3600  Loss: 0.6111965179443359
Epoch: 1  Batch: 4200  Loss: 0.016707444563508034
Epoch: 1  Batch: 4800  Loss: 0.0006908098584972322
Epoch: 1  Batch: 5400  Loss: 0.0002799260546453297
Epoch: 1  Batch: 6000  Loss: 0.4848875403404236
Epoch: 2  Batch: 600  Loss: 0.03840283304452896
Epoch: 2  Batch: 1200  Loss: 0.005653898231685162
Epoch: 2  Batch: 1800  Loss: 0.0019390363013371825
Epoch: 2  Batch: 2400  Loss: 0.013562331907451153
Epoch: 2  Batch: 3000  Loss: 0.004443833604454994
Epoch: 2  Batch: 3600  Loss: 0.0005063370917923748
Epoch: 2  Batch: 4200  Loss: 0.04586367309093475
Epoch: 2  Batch: 4800  Loss: 0.006673174444586039
Epoch: 2  Batch: 5400  Loss: 0.02076614275574684
Epoch: 2  Batch: 6000  Loss: 0.13829907774925232
Epoch: 3  Batch: 600  Loss: 0.0021926886402070522
Epoch: 3  Batch: 1200  Loss: 0.10718250274658203
Epoch: 3  Batch: 1800  Loss: 0.0005934062064625323
Epoch: 3  Batch: 2400  Loss: 0.00016517048061359674
Epoch: 3  Batch: 3000  Loss: 0.0017503199633210897
Epoch: 3  Batch: 3600  Loss: 0.0007489544805139303
Epoch: 3  Batch: 4200  Loss: 0.01164452824741602
Epoch: 3  Batch: 4800  Loss: 7.926556281745434e-05
Epoch: 3  Batch: 5400  Loss: 0.039218269288539886
Epoch: 3  Batch: 6000  Loss: 0.012269356288015842
Epoch: 4  Batch: 600  Loss: 0.019970223307609558
Epoch: 4  Batch: 1200  Loss: 0.031921446323394775
Epoch: 4  Batch: 1800  Loss: 0.2468046396970749
Epoch: 4  Batch: 2400  Loss: 0.00011870403250213712
Epoch: 4  Batch: 3000  Loss: 0.0005112775252200663
Epoch: 4  Batch: 3600  Loss: 0.00011618930147960782
Epoch: 4  Batch: 4200  Loss: 0.00035029969876632094
Epoch: 4  Batch: 4800  Loss: 0.04948687180876732
Epoch: 4  Batch: 5400  Loss: 0.03161567822098732
Epoch: 4  Batch: 6000  Loss: 0.0017111159395426512
Training Took: 2.6056419650713605 minutes!
```

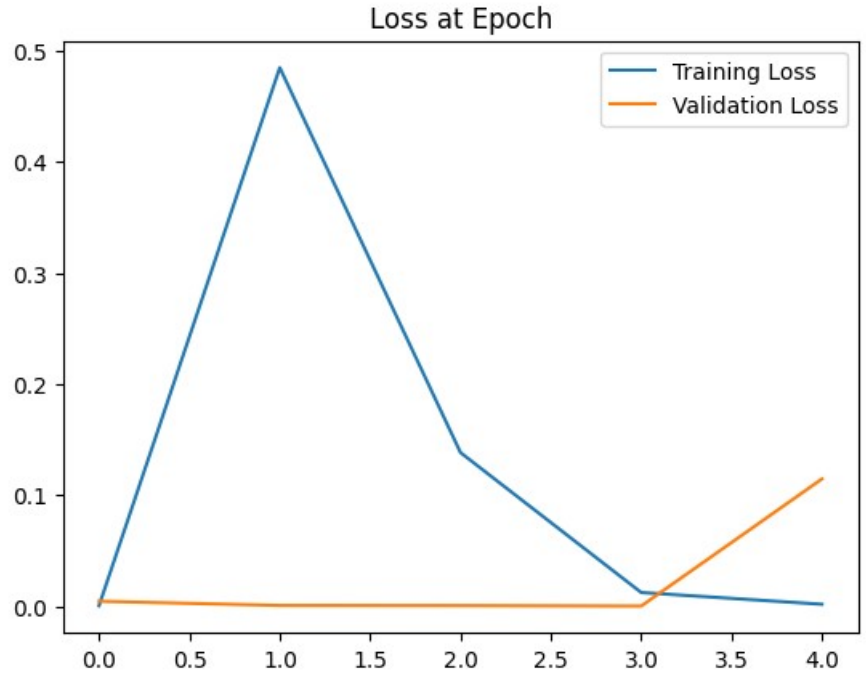
Double-cliquez (ou appuyez sur Entrée) pour modifier

Graph CNN Results - Deep Learning with PyTorch 18

```
# Graph the loss of Epoch
train_losses = [tl.item() for tl in train_losses]
plt.plot(train_losses, label='Training Loss')
plt.plot(test_losses, label='Validation Loss')
plt.title('Loss at Epoch')
plt.legend()
```

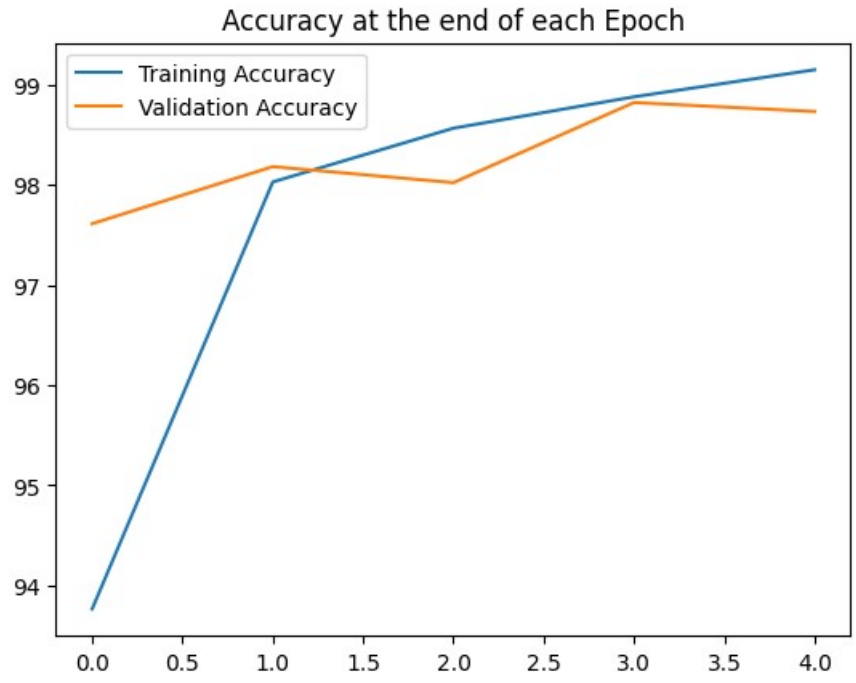
```
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7c7fdfa2cc10>
```




```
# Graph the accuracy at the end of each Epoch
plt.plot([t/600 for t in train_correct], label='Training Accuracy')
plt.plot([t/100 for t in test_correct], label='Validation Accuracy')
plt.title('Accuracy at the end of each Epoch')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7c7fde1e7a00>
```




```
test_load_everything = DataLoader(test_data, batch_size=10_000, shuffle=False)
```

```
with torch.no_grad():
  correct = 0

```

```
  for X_test, y_test in test_load_everything:
    y_val = model(X_test)
    predicted = torch.max(y_val.data, 1)[1]
    correct += (predicted == y_test).sum()

```

```
# Did for correct (out of 10000)
correct.item()
```

```
9873
```

```
# Did for correct (percentage of correct)
correct.item() / len(test_data) * 100
```

```
98.72999999999999
```

NSend New Image Thru The Model - Deep Learning with PyTorch 19

```
# Grab an Image
test_data[4143] # Tensor with an image in it... at the end, it shows the label

```

```
(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, 0.0000, 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, 0.1765, 0.9098, 0.8275, 0.4980, 0.0627,
           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, 0.0706, 0.7961, 0.9961, 0.9961, 0.9961, 0.8235,
           0.3765, 0.0667, 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, 0.4353, 0.9961, 0.9961, 0.4667, 0.8941, 0.9961,
           0.9961, 0.9412, 0.1490, 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, 0.8824, 0.9961, 0.7922, 0.0588, 0.1569, 0.8235,
           0.9961, 0.9961, 0.4431, 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, 0.2314, 0.9569, 0.9961, 0.2706, 0.0000, 0.0000, 0.8157,
 0.9961, 0.9961, 0.6902, 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, 0.7725, 0.9961, 0.9961, 0.1333, 0.0000, 0.2235, 0.9843,
 0.9961, 0.9961, 0.5725, 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, 0.9176, 0.9961, 0.6235, 0.0157, 0.0392, 0.7176, 0.9961,
 0.9961, 0.9961, 0.3843, 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, 0.9176, 0.9961, 0.1569, 0.0000, 0.6549, 0.9961, 0.9961,
 0.9961, 0.9961, 0.2353, 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, 0.7373, 0.9961, 0.2824, 0.6588, 0.9765, 0.9961, 0.9961,
```

```
1 # Grab just the data
2 test_data[4143][0]
```

```
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, 0.0000, 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, 0.1765, 0.9098, 0.8275, 0.4980, 0.0627,
          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, 0.0706, 0.7961, 0.9961, 0.9961, 0.9961, 0.8235,
          0.3765, 0.0667, 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, 0.4353, 0.9961, 0.9961, 0.4667, 0.8941, 0.9961,
          0.9961, 0.9412, 0.1490, 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, 0.8824, 0.9961, 0.7922, 0.0588, 0.1569, 0.8235,
          0.9961, 0.9961, 0.4431, 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, 0.2314, 0.9569, 0.9961, 0.2706, 0.0000, 0.0000, 0.8157,
          0.9961, 0.9961, 0.6902, 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, 0.7725, 0.9961, 0.9961, 0.1333, 0.0000, 0.2235, 0.9843,
          0.9961, 0.9961, 0.5725, 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, 0.9176, 0.9961, 0.6235, 0.0157, 0.0392, 0.7176, 0.9961,
          0.9961, 0.9961, 0.3843, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
```

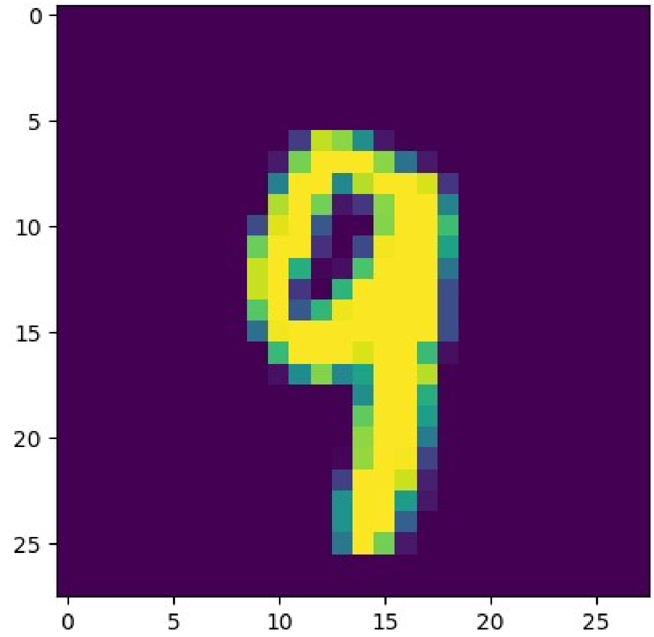
```
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.9176, 0.9961, 0.1569, 0.0000, 0.6549, 0.9961, 0.9961,
         0.9961, 0.9961, 0.2353, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.7373, 0.9961, 0.2824, 0.6588, 0.9765, 0.9961, 0.9961,
```

```
# Reshape it
test_data[4143][0].reshape(28, 28)
```

```
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, 0.0000, 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, 0.1765, 0.9098, 0.8275, 0.4980, 0.0627, 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, 0.0706, 0.7961, 0.9961, 0.9961, 0.9961, 0.8235, 0.3765, 0.0667,
         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, 0.4353, 0.9961, 0.9961, 0.4667, 0.8941, 0.9961, 0.9961, 0.9412,
         0.1490, 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, 0.8824, 0.9961, 0.7922, 0.0588, 0.1569, 0.8235, 0.9961, 0.9961,
         0.4431, 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,
         0.2314, 0.9569, 0.9961, 0.2706, 0.0000, 0.0000, 0.8157, 0.9961, 0.9961,
         0.6902, 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,
         0.7725, 0.9961, 0.9961, 0.1333, 0.0000, 0.2235, 0.9843, 0.9961, 0.9961,
         0.5725, 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,
         0.9176, 0.9961, 0.6235, 0.0157, 0.0392, 0.7176, 0.9961, 0.9961, 0.9961,
         0.3843, 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,
         0.9176, 0.9961, 0.1569, 0.0000, 0.6549, 0.9961, 0.9961, 0.9961, 0.9961,
         0.2353, 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,
         0.7373, 0.9961, 0.2824, 0.6588, 0.9765, 0.9961, 0.9961, 0.9961, 0.9961,
```

```
# Show the image
plt.imshow(test_data[4143][0].reshape(28, 28))

```

```
<matplotlib.image.AxesImage at 0x7c7fdd233f70>
```



```
# Pass the image through the model
model.eval()

with torch.no_grad():
  new_prediction = model(test_data[4143][0].view(1, 1, 28, 28)) # batch size=1, color channel=1, image size: 28x28

```

```
# Check the new prediction - get the probabilities
new_prediction
```

```
tensor([[-2.2904e+01, -1.6249e+01, -2.0912e+01, -1.5222e+01, -8.7263e+00,
         -2.0335e+01, -3.1855e+01, -1.3508e+01, -1.1665e+01, -1.7260e-04]])
```

```
# Highest probability
new_prediction.argmax()
```

```
tensor(9)
```

```
# Grab an Image
test_data[1978] # Tensor with an image in it... at the end, it shows the label
```

```
(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, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.4392, 0.6902, 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, 0.1529, 0.2314,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.7922, 0.9255, 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, 0.5490, 0.9843,
         0.1608, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0353, 0.8392, 0.9255, 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, 0.6196, 0.9961,
         0.1725, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.1412, 0.9961, 0.7333, 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, 0.8941, 0.9059,
         0.0824, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.3804, 0.9961, 0.5804, 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, 0.8941, 0.8235,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.8275, 0.9961, 0.4196, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.1098, 0.9412, 0.8235,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0314, 0.3098,
         0.7569, 0.7922, 0.9608, 0.9961, 0.2392, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.2667, 0.9961, 0.6784,
         0.0000, 0.0000, 0.0000, 0.0039, 0.0706, 0.6392, 0.8235, 0.9961,
         0.9961, 0.9961, 0.9961, 0.9294, 0.1412, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0118, 0.7176, 0.9961, 0.3804,
         0.2039, 0.2627, 0.3804, 0.4039, 0.9961, 1.0000, 0.9961, 0.9725,
         0.7686, 0.9451, 0.9961, 0.4196, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0863, 0.8314, 0.9961, 0.9961,
         0.9961, 0.9961, 0.9961, 0.9961, 0.9961, 0.9961, 0.5647, 0.1294,
```

```
1 # Grab just the data
2 test_data[1978][0]
```

```
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, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.4392, 0.6902, 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, 0.1529, 0.2314,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.7922, 0.9255, 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, 0.5490, 0.9843,
          0.1608, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0353, 0.8392, 0.9255, 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, 0.6196, 0.9961,
          0.1725, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.1412, 0.9961, 0.7333, 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, 0.8941, 0.9059,
          0.0824, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.3804, 0.9961, 0.5804, 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, 0.8941, 0.8235,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.8275, 0.9961, 0.4196, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.1098, 0.9412, 0.8235,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0314, 0.3098,
          0.7569, 0.7922, 0.9608, 0.9961, 0.2392, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.2667, 0.9961, 0.6784,
          0.0000, 0.0000, 0.0000, 0.0039, 0.0706, 0.6392, 0.8235, 0.9961,
          0.9961, 0.9961, 0.9961, 0.9294, 0.1412, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0118, 0.7176, 0.9961, 0.3804,
          0.2039, 0.2627, 0.3804, 0.4039, 0.9961, 1.0000, 0.9961, 0.9725,
          0.7686, 0.9451, 0.9961, 0.4196, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0863, 0.8314, 0.9961, 0.9961,
          0.9961, 0.9961, 0.9961, 0.9961, 0.9961, 0.9961, 0.5647, 0.1294,
```

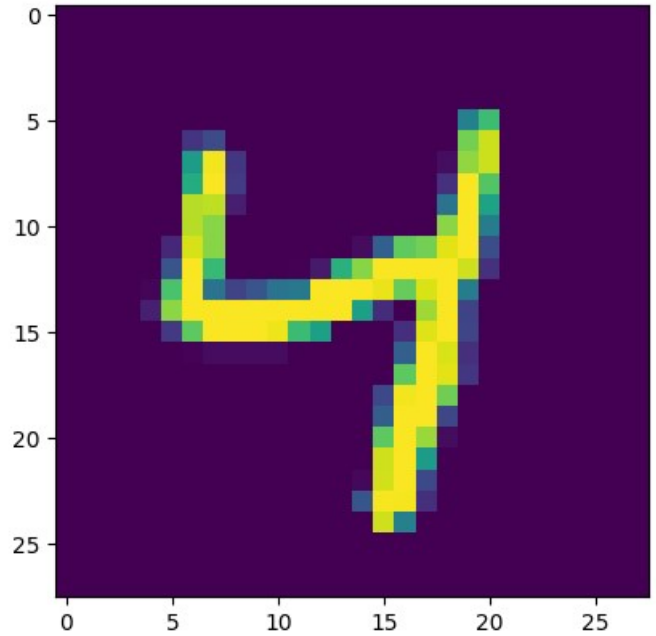
```
1 # Reshape it
2 test_data[1978][0].reshape(28, 28)
```

```
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, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.4392, 0.6902, 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, 0.1529, 0.2314, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.7922, 0.9255, 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, 0.5490, 0.9843, 0.1608,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0353, 0.8392, 0.9255, 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, 0.6196, 0.9961, 0.1725,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.1412, 0.9961, 0.7333, 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, 0.8941, 0.9059, 0.0824,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
```

```
 0.3804, 0.9961, 0.5804, 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, 0.8941, 0.8235, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.8275, 0.9961, 0.4196, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.1098, 0.9412, 0.8235, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0314, 0.3098, 0.7569, 0.7922,
 0.9608, 0.9961, 0.2392, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.2667, 0.9961, 0.6784, 0.0000,
 0.0000, 0.0000, 0.0039, 0.0706, 0.6392, 0.8235, 0.9961, 0.9961, 0.9961,
 0.9961, 0.9294, 0.1412, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0118, 0.7176, 0.9961, 0.3804, 0.2039,
 0.2627, 0.3804, 0.4039, 0.9961, 1.0000, 0.9961, 0.9725, 0.7686, 0.9451,
 0.9961, 0.4196, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0863, 0.8314, 0.9961, 0.9961, 0.9961,
 0.9961, 0.9961, 0.9961, 0.9961, 0.9961, 0.5647, 0.1294, 0.0000, 0.8588,
```

```
# Show the image
plt.imshow(test_data[1978][0].reshape(28, 28))

```

```
<matplotlib.image.AxesImage at 0x7c7fdd5df940>
```




```
# Pass the image through the model
model.eval()

with torch.no_grad():
  new_prediction = model(test_data[1978][0].view(1, 1, 28, 28)) # batch size=1, color channel=1, image size: 28x28

```

```
# Check the new prediction - get the probabilities
new_prediction
```

```
tensor([[-33.3065, -20.1489, -27.4578, -34.2620,   0.0000, -28.9177, -24.5439,
         -28.5581, -22.6274, -18.5013]])
```

```
# Highest probability
new_prediction.argmax()
```

```
tensor(4)
```