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/
```

```
1 cd ../
```

```
/
```

```
1 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/
```

```
1 cd cnn_data
```

```
/cnn_data
```

```
1 ls -l
```

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

```
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 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

```
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)
```

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

```
1 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 # 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 g
```