```
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


### name: Sana Fathima H
### REGISTER NUMBER : 212223240145


transform = transforms.ToTensor()


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

```
100%|██████████| 9.91M/9.91M [00:00<00:00, 34.5MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 888kB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 8.59MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 947kB/s]
```

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

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

```
test_data
```

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

```
train_loader = DataLoader(train_data, batch_size=10, shuffle=True)
test_loader = DataLoader(test_data, batch_size=10, shuffle=False)


# 1 colour channel, 6 filter(output channel) 3 x 3 kernal , stride = 1
conv1 = nn.Conv2d(1,6,3,1) # ---> 6 filters ---> pooling ---> conv2
# 6 input filters conv1, 16 filters, 3 x 3 kernal, stride = 1
conv2 = nn.Conv2d(6,16,3,1)


for i, (X_train, y_train) in enumerate(train_data):
    break


### name: Sana Fathima H
### REGISTER NUMBER : 212223240145
X_train
```

```
          0.0902, 0.2588, 0.8353, 0.9922, 0.9922, 0.9922, 0.9922, 0.7765,
          0.3176, 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, 0.0706, 0.6706,
          0.8588, 0.9922, 0.9922, 0.9922, 0.9922, 0.7647, 0.3137, 0.0353,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.2157, 0.6745, 0.8863, 0.9922,
          0.9922, 0.9922, 0.9922, 0.9569, 0.5216, 0.0431, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.5333, 0.9922, 0.9922, 0.9922,
          0.8314, 0.5294, 0.5176, 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, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000]]])
```

```
X_train.shape
```

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

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

```
x
```

```
           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,
           0.9922, 0.4667, 0.0980, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.1765, 0.7294,
           0.9922, 0.9922, 0.5882, 0.1059, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0627,
           0.3647, 0.9882, 0.9922, 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, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.9765, 0.9922, 0.9765, 0.2510, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.1804, 0.5098,
           0.7176, 0.9922, 0.9922, 0.8118, 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, 0.1529, 0.5804, 0.8980, 0.9922,
           0.9922, 0.9922, 0.9804, 0.7137, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0941, 0.4471, 0.8667, 0.9922, 0.9922, 0.9922,
           0.9922, 0.7882, 0.3059, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0902, 0.2588, 0.8353, 0.9922, 0.9922, 0.9922, 0.9922, 0.7765,
           0.3176, 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, 0.0706, 0.6706,
           0.8588, 0.9922, 0.9922, 0.9922, 0.9922, 0.7647, 0.3137, 0.0353,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, 0.0000, 0.2157, 0.6745, 0.8863, 0.9922,
           0.9922, 0.9922, 0.9922, 0.9569, 0.5216, 0.0431, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, 0.0000, 0.5333, 0.9922, 0.9922, 0.9922,
           0.8314, 0.5294, 0.5176, 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, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000]]]])
```

```
### name: Sana Fathima H
### REGISTER NUMBER : 212223240145
```

```
class ConvolutionalNetwork(nn.Module):

    def __init__(self):
```

```
            super().__init__()
            self.conv1 = nn.Conv2d(1,6,3,1)
            self.conv2 = nn.Conv2d(6,16,3,1)
            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)
            X = F.relu(self.conv2(X))
            X = F.max_pool2d(X, 2, 2)
            X = X.view(-1, 5*5*16)
            X = F.relu(self.fc1(X))
            X = F.relu(self.fc2(X))
            X = self.fc3(X)
            return F.log_softmax(X, dim=1)


    torch.manual_seed(42)
    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)
)
```

```
    for param in model.parameters():
        print(param.numel())
```

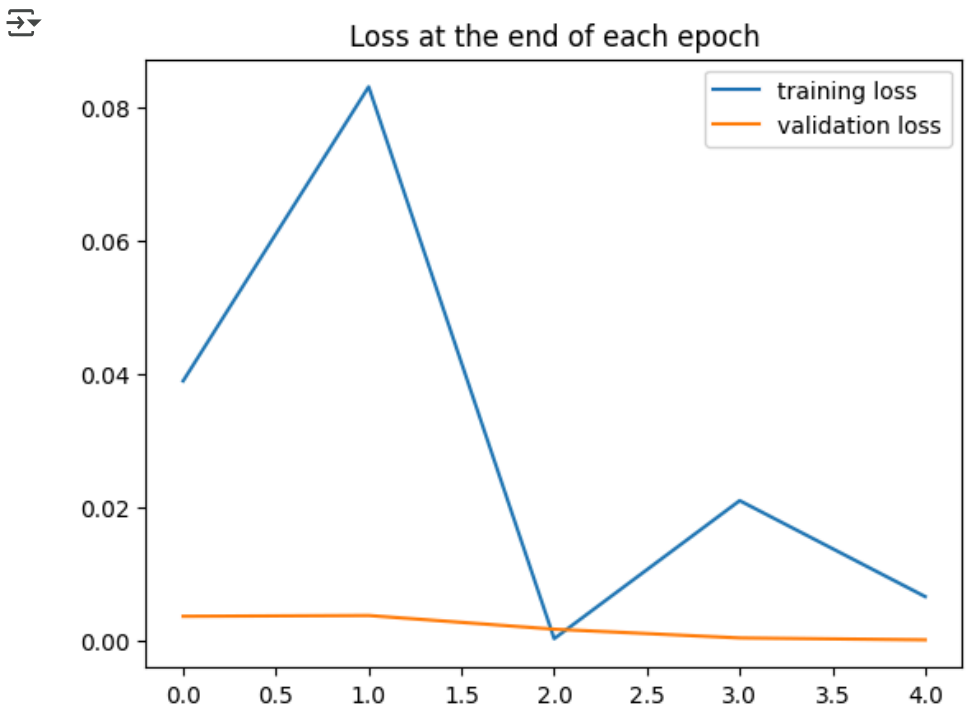
```
54
6
864
16
48000
120
10080
84
840
10
```

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)


import time
start_time = time.time()

# Variables ( Trackers)
epochs = 5
train_losses = []
test_losses = []
train_correct = []
test_correct = []

# for loop epochs
for i in range(epochs):

    trn_corr = 0
    tst_corr = 0

    # Run the training batches
    for b, (X_train, y_train) in enumerate(train_loader):
        b+=1

        # Apply the model
        y_pred = model(X_train)  # we not flatten X-train here
        loss = criterion(y_pred, y_train)


        predicted = torch.max(y_pred.data, 1)[1]
        batch_corr = (predicted == y_train).sum()  # Trure 1 / False 0 sum()
        trn_corr += batch_corr
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Print interim results
        if b%600 == 0:
            print(f'epoch: {i}  batch: {b} loss: {loss.item()}')

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

    # Run the testing batches
    with torch.no_grad():
        for b, (X_test, y_test) in enumerate(test_loader):

            # Apply the model
            y_val = model(X_test)

            # Tally the number of correct predictions
```

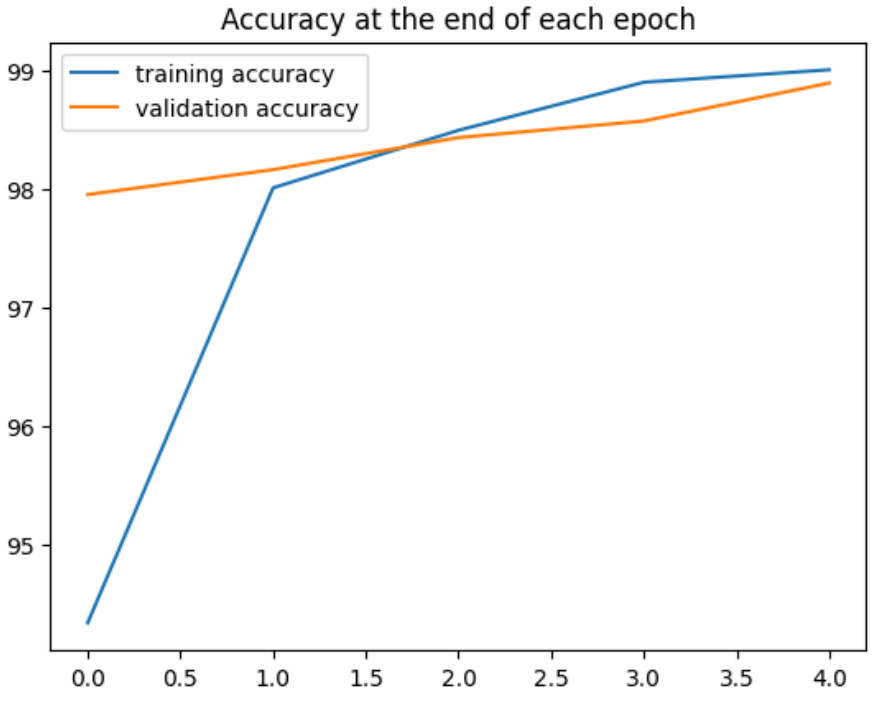
```
            predicted = torch.max(y_val.data, 1)[1]
            tst_corr += (predicted == y_test).sum()

        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: 0.040556274354457855
epoch: 0  batch: 1200 loss: 0.08253474533557892
epoch: 0  batch: 1800 loss: 0.3647049069404602
epoch: 0  batch: 2400 loss: 0.018250251188874245
epoch: 0  batch: 3000 loss: 0.008067040704190731
epoch: 0  batch: 3600 loss: 0.001166942878626287
epoch: 0  batch: 4200 loss: 0.5255253911018372
epoch: 0  batch: 4800 loss: 0.03260819613933563
epoch: 0  batch: 5400 loss: 0.007468158844858408
epoch: 0  batch: 6000 loss: 0.03889675810933113
epoch: 1  batch: 600 loss: 0.032828204333782196
epoch: 1  batch: 1200 loss: 0.04554177075624466
epoch: 1  batch: 1800 loss: 0.005784796085208654
epoch: 1  batch: 2400 loss: 0.02235613949596882
epoch: 1  batch: 3000 loss: 0.21643038094043732
epoch: 1  batch: 3600 loss: 0.00501451687887311
epoch: 1  batch: 4200 loss: 0.00045869071618653834
epoch: 1  batch: 4800 loss: 0.0019295118981972337
epoch: 1  batch: 5400 loss: 0.0008596166153438389
epoch: 1  batch: 6000 loss: 0.08304359018802643
epoch: 2  batch: 600 loss: 0.0006373372743837535
epoch: 2  batch: 1200 loss: 0.0015393418725579977
epoch: 2  batch: 1800 loss: 0.0012801657430827618
epoch: 2  batch: 2400 loss: 0.001396776526235044
epoch: 2  batch: 3000 loss: 0.3044474124908447
epoch: 2  batch: 3600 loss: 0.014451900497078896
epoch: 2  batch: 4200 loss: 0.021982822567224503
epoch: 2  batch: 4800 loss: 0.0007802899926900864
epoch: 2  batch: 5400 loss: 0.0016833205008879304
epoch: 2  batch: 6000 loss: 0.0002076365490211174
epoch: 3  batch: 600 loss: 0.0007947428966872394
epoch: 3  batch: 1200 loss: 0.002038671402260661
epoch: 3  batch: 1800 loss: 0.0004689941997639835
epoch: 3  batch: 2400 loss: 0.00021815943182446063
epoch: 3  batch: 3000 loss: 0.031423646956682205
epoch: 3  batch: 3600 loss: 0.0073494575917720795
epoch: 3  batch: 4200 loss: 0.0006103587802499533
epoch: 3  batch: 4800 loss: 0.13828447461128235
epoch: 3  batch: 5400 loss: 0.0007458419422619045
epoch: 3  batch: 6000 loss: 0.02092968113720417
epoch: 4  batch: 600 loss: 0.0009378452086821198
epoch: 4  batch: 1200 loss: 0.19402171671390533
epoch: 4  batch: 1800 loss: 0.0006758190575055778
epoch: 4  batch: 2400 loss: 0.00019682350102812052
epoch: 4  batch: 3000 loss: 0.005403806921094656
```

```
train_losses = [t.detach().numpy() for t in train_losses]
test_losses = [t.detach().numpy() for t in test_losses]

plt.plot(train_losses, label='training loss')
plt.plot(test_losses, label='validation loss')
plt.title('Loss at the end of each epoch')
plt.legend();
plt.show()
```




```
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();
plt.show()
```




```
test_load_all = DataLoader(test_data, batch_size=10000, shuffle=False)


with torch.no_grad():
    correct = 0
    for X_test, y_test in test_load_all:
        y_val = model(X_test)  # we don't flatten the data this time
        predicted = torch.max(y_val,1)[1]
        correct += (predicted == y_test).sum()


correct.item()
```

```
9889
```

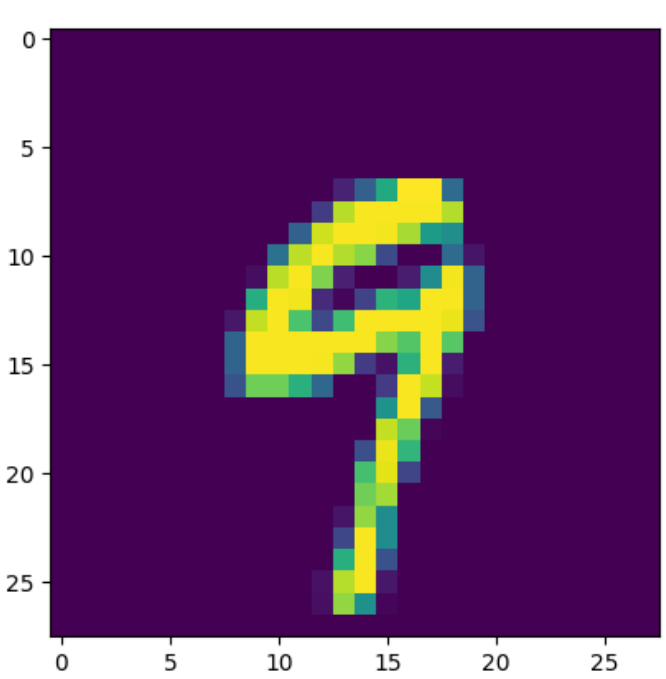
```
correct.item()/len(test_data)
```

```
0.9889
```

```
np.set_printoptions(formatter=dict(int=lambda x: f'{x:4}'))
print(np.arange(10).reshape(1,10))
print()

# print the confusion matrix
print(confusion_matrix(predicted.view(-1), y_test.view(-1)))
```

```
[[   0    1    2    3    4    5    6    7    8    9]]

    [[ 977    3    1    0    0    2    4    1    4    0]
     [   0 1130    2    0    0    0    1    4    0    3]
     [   0    1 1022    0    0    0    0    4    2    0]
     [   0    0    3 1007    0    7    0    2    2    2]
     [   0    0    1    0  971    0    1    0    0    5]
     [   0    0    0    1    0  879    7    0    1    5]
     [   1    1    0    0    4    2  944    0    1    0]
     [   1    0    3    0    0    0    0 1013    1    2]
     [   1    0    0    2    1    1    1    1  962    8]
     [   0    0    0    0    6    1    0    3    1  984]]
```

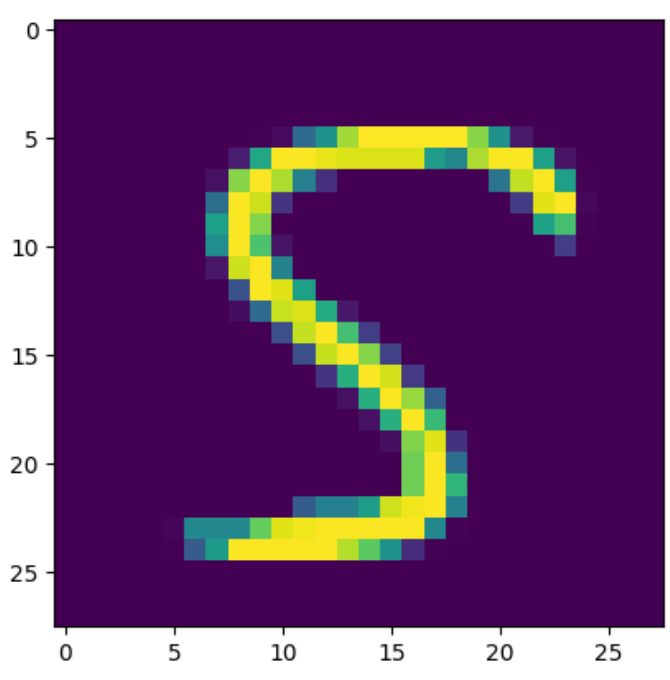
```
plt.imshow(test_data[2019][0].reshape(28,28))
plt.show()
```




```
model.eval()
with torch.no_grad():
    new_prediction = model(test_data[2019][0].view(1,1,28,28))


new_prediction.argmax()
```

```
tensor(9)
```

```
plt.imshow(test_data[333][0].reshape(28,28))
plt.show()
```




```
model.eval()
with torch.no_grad():
    new_prediction = model(test_data[333][0].view(1,1,28,28))
```

```
new_prediction.argmax()
```

```
tensor(5)
```

```
test_data[333][1]
```

```
5
```