Name: Priyanshu Prakash, College:Vit Vellore

Name: Shreyas Saxena, College:KIIT

We are building the project on Mnist Dataset

About Mnist data: The MNIST database contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset.

Here we would we using ANN and CNN models to train and test the data sets

we are getting an test accuracy of 97.73%

In [ ]:
```
import torch
import torch.nn as nn
import torch.nn.functional as
```

In [2]:
```
from torch.utils.data import DataLoader
from torchvision import datasets,transforms
```

In [3]:
```
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
%matplotlib inline
```

In [4]:
```
#MNIST image to tensor
```

In [5]:
```
transform=transforms.ToTensor()
```

In [6]:
```
train_data=datasets.MNIST(root='/Users/priyanshuprakash/Desktop/4th Year/
```

In [7]:
```
test_data=datasets.MNIST(root='/Users/priyanshuprakash/Desktop/4th Year/I
```

In [8]:
```
train_data
```

Out[8]:
```
Dataset MNIST
    Number of datapoints: 60000
    Root location: /Users/priyanshuprakash/Desktop/4th Year/Image process
ing/ANN
    Split: Train
    StandardTransform
Transform: ToTensor()
```

In [9]:
```
test_data
```

Out[9]:
```
Dataset MNIST
    Number of datapoints: 10000
    Root location: /Users/priyanshuprakash/Desktop/4th Year/Image process
ing/ANN
    Split: Test
    StandardTransform
Transform: ToTensor()
```

In [10]:
```
type(train_data)
```

Out[10]:
```
torchvision.datasets.mnist.MNIST
```

In [11]:
```
train_data[0]
```

Out[11]:
```
(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
,
          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]]]),
 5)
```

In [12]:
```
image,label=train_data[0]
```

In [13]:
```
image.shape
```

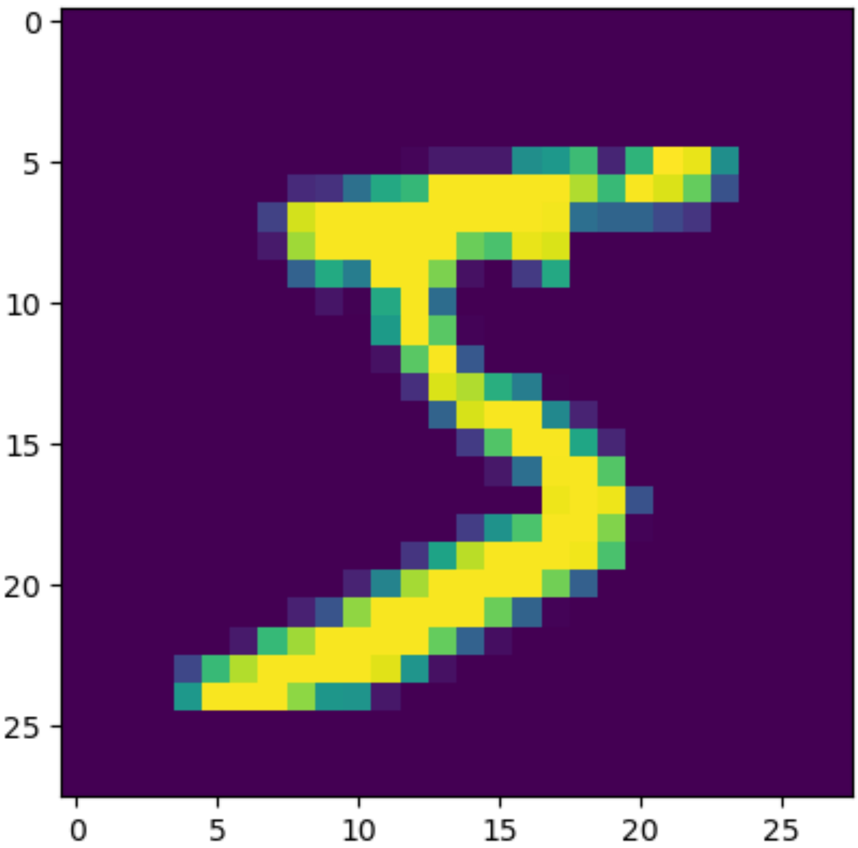
Out[13]:
```
torch.Size([1, 28, 28])
```

In [14]:
```
label
```

Out[14]:
```
5
```

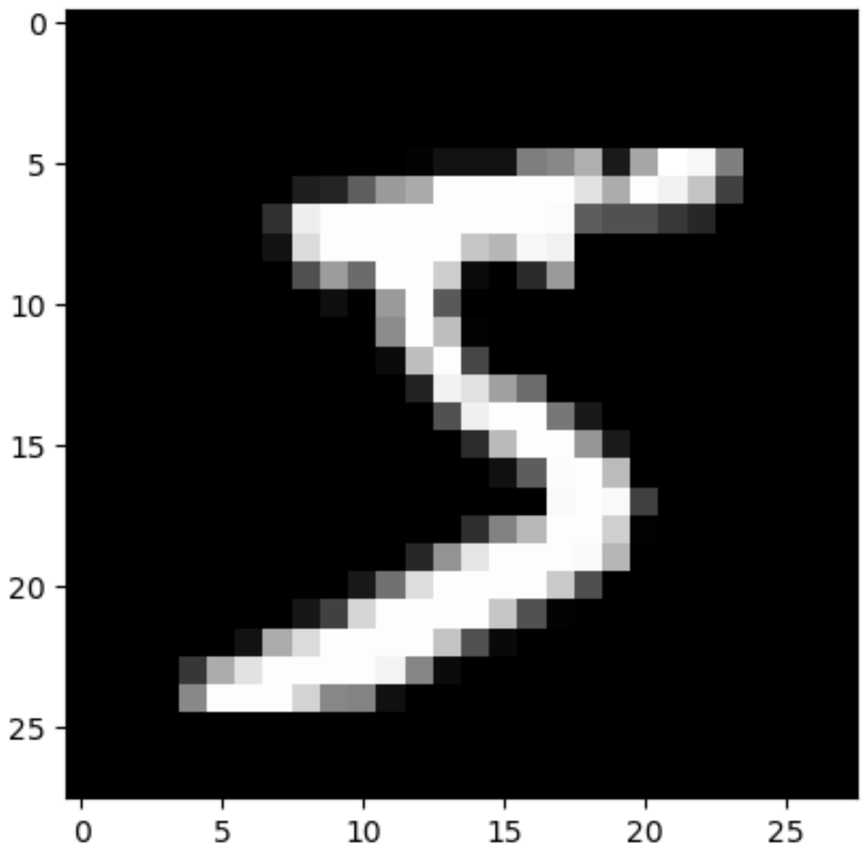
In [15]:
```
plt.imshow(image.reshape((28,28)))
```

Out[15]: <matplotlib.image.AxesImage at 0x163a81ed0>



In [16]:
```
plt.imshow(image.reshape((28,28)),cmap='gray')
```

Out[16]: <matplotlib.image.AxesImage at 0x163ae6710>



In [17]:
```
torch.manual_seed(101)

train_loader=DataLoader(train_data,batch_size=100,shuffle=True)
test_loader=DataLoader(test_data,batch_size=500,shuffle=False)
```

In [18]:
```
from torchvision.utils import make_grid
np.set_printoptions(formatter=dict(int=lambda x:f'{x:4}')) #FORMATTING
```

In [19]:
```
#First Batch
for images,labels in train_loader:
    break
```

In [20]:
```
images.shape   # 1->Means it is a grey scale image,(28,28)->width,length o
```

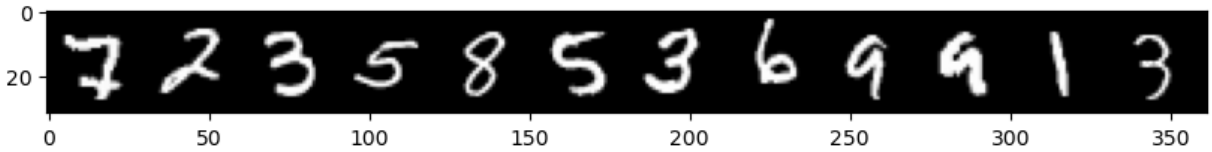
Out[20]:
```
torch.Size([100, 1, 28, 28])
```

In [21]:
```
labels.shape
```

Out[21]:
```
torch.Size([100])
```

In [22]:
```
#Print first 12 labels
print('Labels:',labels[:12].numpy())

#Print the first 12 images
im=make_grid(images[:12],nrow=12)#default nrow is 8
plt.figure(figsize=(10,4))

plt.imshow(np.transpose(im.numpy(),(1,2,0)));
```

```
Labels: [    7     2     3     5     8     5     3     6     9     9     1     3]
```



In [23]:
```
class MultilayerPerceptron(nn.Module):
    def __init__(self,in_sz=784,out_sz=10,layers=[120,84]):
        super().__init__()

        self.fc1=nn.Linear(in_sz,layers[0])
        self.fc2=nn.Linear(layers[0],layers[1])
        self.fc3=nn.Linear(layers[1],out_sz)

    def forward(self,x): #x->data features
        x=F.relu(self.fc1(x))
        x=F.relu(self.fc2(x))
        x=self.fc3(x)

        return F.log_softmax(x,dim=1) #Multi-class classification
```

In [24]:
```
torch.manual_seed(101)
model=MultilayerPerceptron()
model
```

Out[24]:
```
MultilayerPerceptron(
  (fc1): Linear(in_features=784, 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)
)
```

In [25]:
```
criterion=nn.CrossEntropyLoss()
optimizer=torch.optim.Adam(model.parameters(),lr=0.001)
```

In [26]:
```
images.shape
```

Out[26]:
```
torch.Size([100, 1, 28, 28])
```

In [27]:
```
images.view(100,-1)
```

Out[27]:
```
tensor([[0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        ...,
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.]])
```

In [28]:
```
import time
start_time = time.time()

epochs = 10
train_losses = []
test_losses = []
train_correct = []
test_correct = []

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.view(100, -1))  # Here we flatten X_train
        loss = criterion(y_pred, y_train)

        # Tally the number of correct predictions
        predicted = torch.max(y_pred.data, 1)[1]
        batch_corr = (predicted == y_train).sum()
        trn_corr += batch_corr

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

        # Print interim results
        if b%200 == 0:
            print(f'epoch: {i:2}  batch: {b:4} [{100*b:6}/60000]  loss: {
accuracy: {trn_corr.item()*100/(100*b):7.3f}%')

    # Update train loss & accuracy for the epoch
    train_losses.append(loss.item())
    train_correct.append(trn_corr.item())

    # 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.view(500, -1))  # Here we flatten X_test

            # Tally the number of correct predictions
            predicted = torch.max(y_val.data, 1)[1]
```

```
            tst_corr += (predicted == y_test).sum()

    # Update test loss & accuracy for the epoch
    loss = criterion(y_val, y_test)
    test_losses.append(loss)
    test_correct.append(tst_corr)

print(f'\nDuration: {time.time() - start_time:.0f} seconds') # print the




#total_time=time.time()-start_time
#print(f'Duration:{Total time/60}mins')
```

```
epoch:  0  batch:  200 [ 20000/60000]  loss: 0.23562382  accuracy:  83.24
5%
epoch:  0  batch:  400 [ 40000/60000]  loss: 0.35330707  accuracy:  87.53
2%
epoch:  0  batch:  600 [ 60000/60000]  loss: 0.13765770  accuracy:  89.58
2%
epoch:  1  batch:  200 [ 20000/60000]  loss: 0.24507998  accuracy:  94.88
0%
epoch:  1  batch:  400 [ 40000/60000]  loss: 0.14064841  accuracy:  95.11
0%
epoch:  1  batch:  600 [ 60000/60000]  loss: 0.07430533  accuracy:  95.37
7%
epoch:  2  batch:  200 [ 20000/60000]  loss: 0.08479684  accuracy:  96.57
0%
epoch:  2  batch:  400 [ 40000/60000]  loss: 0.06338982  accuracy:  96.67
2%
epoch:  2  batch:  600 [ 60000/60000]  loss: 0.06284785  accuracy:  96.73
7%
epoch:  3  batch:  200 [ 20000/60000]  loss: 0.11593810  accuracy:  97.65
0%
epoch:  3  batch:  400 [ 40000/60000]  loss: 0.05100821  accuracy:  97.46
8%
epoch:  3  batch:  600 [ 60000/60000]  loss: 0.03086828  accuracy:  97.49
3%
epoch:  4  batch:  200 [ 20000/60000]  loss: 0.11303577  accuracy:  98.17
0%
epoch:  4  batch:  400 [ 40000/60000]  loss: 0.04967898  accuracy:  98.01
3%
epoch:  4  batch:  600 [ 60000/60000]  loss: 0.05145194  accuracy:  98.01
0%
epoch:  5  batch:  200 [ 20000/60000]  loss: 0.00721604  accuracy:  98.41
0%
epoch:  5  batch:  400 [ 40000/60000]  loss: 0.03383062  accuracy:  98.46
8%
epoch:  5  batch:  600 [ 60000/60000]  loss: 0.05648751  accuracy:  98.38
0%
epoch:  6  batch:  200 [ 20000/60000]  loss: 0.02997145  accuracy:  98.82
0%
epoch:  6  batch:  400 [ 40000/60000]  loss: 0.06433750  accuracy:  98.69
0%
epoch:  6  batch:  600 [ 60000/60000]  loss: 0.08998419  accuracy:  98.67
```

```
0%
epoch:  7  batch:  200 [ 20000/60000]  loss: 0.07536934  accuracy:  99.10
5%
epoch:  7  batch:  400 [ 40000/60000]  loss: 0.09858016  accuracy:  98.91
0%
epoch:  7  batch:  600 [ 60000/60000]  loss: 0.03994036  accuracy:  98.86
5%
epoch:  8  batch:  200 [ 20000/60000]  loss: 0.00415698  accuracy:  99.22
0%
epoch:  8  batch:  400 [ 40000/60000]  loss: 0.00953338  accuracy:  99.13
5%
epoch:  8  batch:  600 [ 60000/60000]  loss: 0.08832055  accuracy:  99.11
0%
epoch:  9  batch:  200 [ 20000/60000]  loss: 0.00536544  accuracy:  99.34
0%
epoch:  9  batch:  400 [ 40000/60000]  loss: 0.01200775  accuracy:  99.26
5%
epoch:  9  batch:  600 [ 60000/60000]  loss: 0.02104353  accuracy:  99.24
2%

Duration: 39 seconds
```

In [29]:
```
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();
```

Evaluating test data

In [30]:
```
print(test_correct) # contains the results of all 10 epochs
print()
print(f'Test accuracy: {test_correct[-1].item()*100/10000:.3f}%') # print
```

```
[tensor(9437), tensor(9579), tensor(9691), tensor(9706), tensor(9746), te
nsor(9759), tensor(9777), tensor(9756), tensor(9751), tensor(9773)]

Test accuracy: 97.730%
```

In [31]:
```
# Extract the data all at once, not in batches
test_load_all = DataLoader(test_data, batch_size=10000, shuffle=False)
```

In [32]:
```
with torch.no_grad():
    correct = 0
    for X_test, y_test in test_load_all:
        y_val = model(X_test.view(len(X_test), -1))  # pass in a flattene
        predicted = torch.max(y_val,1)[1]
        correct += (predicted == y_test).sum()
print(f'Test accuracy: {correct.item()}/{len(test_data)} = {correct.item(
```

```
Test accuracy: 9773/10000 =  97.730%
```

Confusion Matrix

In [33]:
```
# print a row of values for reference
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]]

[[ 970    0    4    0    1    2    6    3    5    3]
 [   0 1124    4    0    0    0    3    7    0    2]
 [   1    3 1002    2    1    0    2   11    2    0]
 [   3    3    5  991    0   10    1    3    4    2]
 [   1    0    2    0  961    3    4    2    2    5]
 [   0    0    0    5    0  865    3    1    2    2]
 [   1    1    2    0    6    2  938    0    1    0]
 [   1    0    3    4    1    0    0  978    3    0]
 [   2    4   10    2    1    8    1    2  952    3]
 [   1    0    0    6   11    2    0   21    3  992]]
```

Using CNN for "Mnist" Dataset

In [34]:
```
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
```

In [36]:
```
transform = transforms.ToTensor()

train_data = datasets.MNIST(root='/Users/priyanshuprakash/Desktop/4th Yea
test_data = datasets.MNIST(root='/Users/priyanshuprakash/Desktop/4th Year
```

In [37]:
```
train_data
```

Out[37]:
```
Dataset MNIST
    Number of datapoints: 60000
    Root location: /Users/priyanshuprakash/Desktop/4th Year/Image process
ing/ANN
    Split: Train
    StandardTransform
Transform: ToTensor()
```

In [38]:
```
test_data
```

Out[38]:
```
Dataset MNIST
    Number of datapoints: 10000
    Root location: /Users/priyanshuprakash/Desktop/4th Year/Image process
ing/ANN
    Split: Test
    StandardTransform
Transform: ToTensor()
```

In [39]:
```
train_loader = DataLoader(train_data, batch_size=10, shuffle=True)
test_loader = DataLoader(test_data, batch_size=10, shuffle=False)
```

In [40]:
```
#Defining CNN
```

In [41]:
```
conv1 = nn.Conv2d(1, 6, 3, 1)
conv2 = nn.Conv2d(6, 16, 3, 1)
```

In [42]:
```
# Grab the first MNIST record
for i, (X_train, y_train) in enumerate(train_data):
    break
```

In [43]:
```
# Create a rank-4 tensor to be passed into the model
# (train_loader will have done this already)
x = X_train.view(1,1,28,28)
print(x.shape)
```

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

In [44]:
```
# Perform the first convolution/activation
x = F.relu(conv1(x))
print(x.shape)
```

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

In [45]:
```
# Run the first pooling layer
x = F.max_pool2d(x, 2, 2)
print(x.shape)
```

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

In [46]:
```
# Perform the second convolution/activation
x = F.relu(conv2(x))
print(x.shape)
```

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

In [48]:
```
# Run the second pooling layer
x = F.max_pool2d(x, 2, 2)
print(x.shape)
```

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

In [49]:
```
# Flatten the data
x = x.view(-1, 5*5*16)
print(x.shape)
```

```
torch.Size([1, 400])
```

In [50]:
```
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)
```

In [51]:
```
torch.manual_seed(42)
model = ConvolutionalNetwork()
model
```

Out[51]:
```
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)
)
```

In [52]:
```
#Loss function
```

In [53]:
```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

In [54]:
```
#Train the model
import time
start_time = time.time()

epochs = 5
train_losses = []
test_losses = []
train_correct = []
test_correct = []

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 don't flatten X-train here
        loss = criterion(y_pred, y_train)

        # Tally the number of correct predictions
        predicted = torch.max(y_pred.data, 1)[1]
        batch_corr = (predicted == y_train).sum()
        trn_corr += batch_corr

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

        # Print interim results
        if b%600 == 0:
            print(f'epoch: {i:2}  batch: {b:4} [{10*b:6}/60000]  loss: {l
accuracy: {trn_corr.item()*100/(10*b):7.3f}%')

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

    # 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)

print(f'\nDuration: {time.time() - start_time:.0f} seconds') # print the
```

```
epoch:  0  batch:  600 [  6000/60000]  loss: 0.04055630  accuracy:  78.41
7%
epoch:  0  batch: 1200 [ 12000/60000]  loss: 0.08253471  accuracy:  85.80
0%
epoch:  0  batch: 1800 [ 18000/60000]  loss: 0.36470532  accuracy:  88.68
9%
epoch:  0  batch: 2400 [ 24000/60000]  loss: 0.01825019  accuracy:  90.52
5%
epoch:  0  batch: 3000 [ 30000/60000]  loss: 0.00806712  accuracy:  91.65
0%
epoch:  0  batch: 3600 [ 36000/60000]  loss: 0.00097706  accuracy:  92.49
2%
epoch:  0  batch: 4200 [ 42000/60000]  loss: 0.44326892  accuracy:  93.13
3%
epoch:  0  batch: 4800 [ 48000/60000]  loss: 0.03169333  accuracy:  93.61
5%
epoch:  0  batch: 5400 [ 54000/60000]  loss: 0.01946524  accuracy:  94.03
1%
epoch:  0  batch: 6000 [ 60000/60000]  loss: 0.02709320  accuracy:  94.33
3%
epoch:  1  batch:  600 [  6000/60000]  loss: 0.01472266  accuracy:  97.75
0%
epoch:  1  batch: 1200 [ 12000/60000]  loss: 0.04359249  accuracy:  97.87
5%
epoch:  1  batch: 1800 [ 18000/60000]  loss: 0.00124075  accuracy:  97.90
0%
epoch:  1  batch: 2400 [ 24000/60000]  loss: 0.03912879  accuracy:  97.85
4%
epoch:  1  batch: 3000 [ 30000/60000]  loss: 0.14564939  accuracy:  97.86
7%
epoch:  1  batch: 3600 [ 36000/60000]  loss: 0.00049980  accuracy:  97.87
8%
epoch:  1  batch: 4200 [ 42000/60000]  loss: 0.00076085  accuracy:  97.91
4%
epoch:  1  batch: 4800 [ 48000/60000]  loss: 0.00105086  accuracy:  97.91
2%
epoch:  1  batch: 5400 [ 54000/60000]  loss: 0.00745581  accuracy:  97.94
1%
epoch:  1  batch: 6000 [ 60000/60000]  loss: 0.13721663  accuracy:  97.94
3%
epoch:  2  batch:  600 [  6000/60000]  loss: 0.00099742  accuracy:  98.68
3%
epoch:  2  batch: 1200 [ 12000/60000]  loss: 0.00254112  accuracy:  98.65
0%
epoch:  2  batch: 1800 [ 18000/60000]  loss: 0.00188525  accuracy:  98.53
9%
epoch:  2  batch: 2400 [ 24000/60000]  loss: 0.00276521  accuracy:  98.59
6%
epoch:  2  batch: 3000 [ 30000/60000]  loss: 0.24948892  accuracy:  98.55
3%
epoch:  2  batch: 3600 [ 36000/60000]  loss: 0.03415573  accuracy:  98.54
4%
epoch:  2  batch: 4200 [ 42000/60000]  loss: 0.02346098  accuracy:  98.52
4%
epoch:  2  batch: 4800 [ 48000/60000]  loss: 0.01159347  accuracy:  98.54
6%
epoch:  2  batch: 5400 [ 54000/60000]  loss: 0.00048137  accuracy:  98.53
```

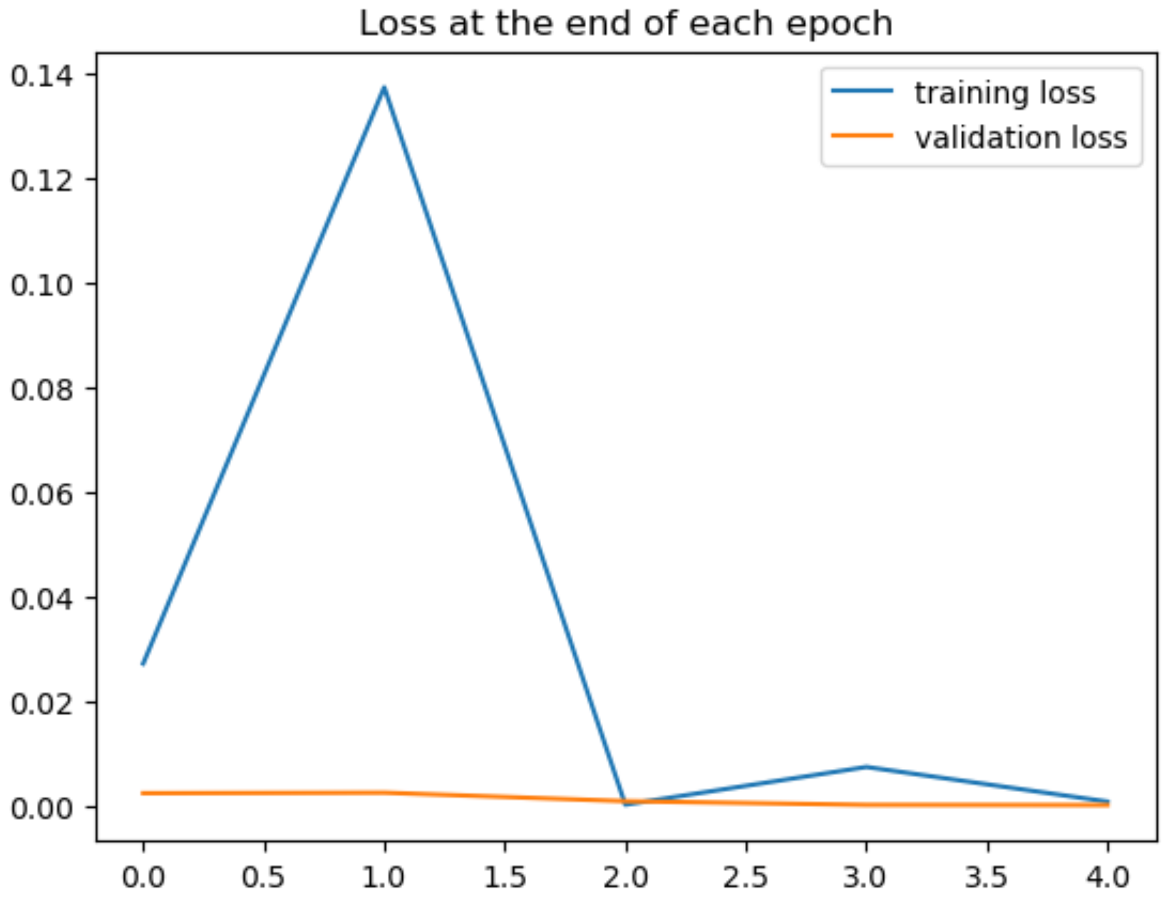
```
1%
epoch:  2  batch: 6000 [ 60000/60000]  loss: 0.00013920  accuracy:  98.53
5%
epoch:  3  batch:  600 [  6000/60000]  loss: 0.00057079  accuracy:  98.90
0%
epoch:  3  batch: 1200 [ 12000/60000]  loss: 0.00067071  accuracy:  98.80
0%
epoch:  3  batch: 1800 [ 18000/60000]  loss: 0.00059610  accuracy:  98.86
1%
epoch:  3  batch: 2400 [ 24000/60000]  loss: 0.00048164  accuracy:  98.81
7%
epoch:  3  batch: 3000 [ 30000/60000]  loss: 0.12320199  accuracy:  98.83
3%
epoch:  3  batch: 3600 [ 36000/60000]  loss: 0.00446792  accuracy:  98.83
3%
epoch:  3  batch: 4200 [ 42000/60000]  loss: 0.00079797  accuracy:  98.84
8%
epoch:  3  batch: 4800 [ 48000/60000]  loss: 0.12382223  accuracy:  98.84
2%
epoch:  3  batch: 5400 [ 54000/60000]  loss: 0.00634904  accuracy:  98.83
3%
epoch:  3  batch: 6000 [ 60000/60000]  loss: 0.00734646  accuracy:  98.85
3%
epoch:  4  batch:  600 [  6000/60000]  loss: 0.00278111  accuracy:  99.08
3%
epoch:  4  batch: 1200 [ 12000/60000]  loss: 0.17682204  accuracy:  99.06
7%
epoch:  4  batch: 1800 [ 18000/60000]  loss: 0.00004966  accuracy:  99.08
3%
epoch:  4  batch: 2400 [ 24000/60000]  loss: 0.00025236  accuracy:  99.06
7%
epoch:  4  batch: 3000 [ 30000/60000]  loss: 0.00490724  accuracy:  99.04
0%
epoch:  4  batch: 3600 [ 36000/60000]  loss: 0.05752410  accuracy:  99.04
4%
epoch:  4  batch: 4200 [ 42000/60000]  loss: 0.01335440  accuracy:  99.07
1%
epoch:  4  batch: 4800 [ 48000/60000]  loss: 0.00020917  accuracy:  99.06
5%
epoch:  4  batch: 5400 [ 54000/60000]  loss: 0.00006763  accuracy:  99.05
0%
epoch:  4  batch: 6000 [ 60000/60000]  loss: 0.00072345  accuracy:  99.05
7%

Duration: 74 seconds
```

In [55]:

```
## Plot the loss and accuracy comparisons
```

In [56]:

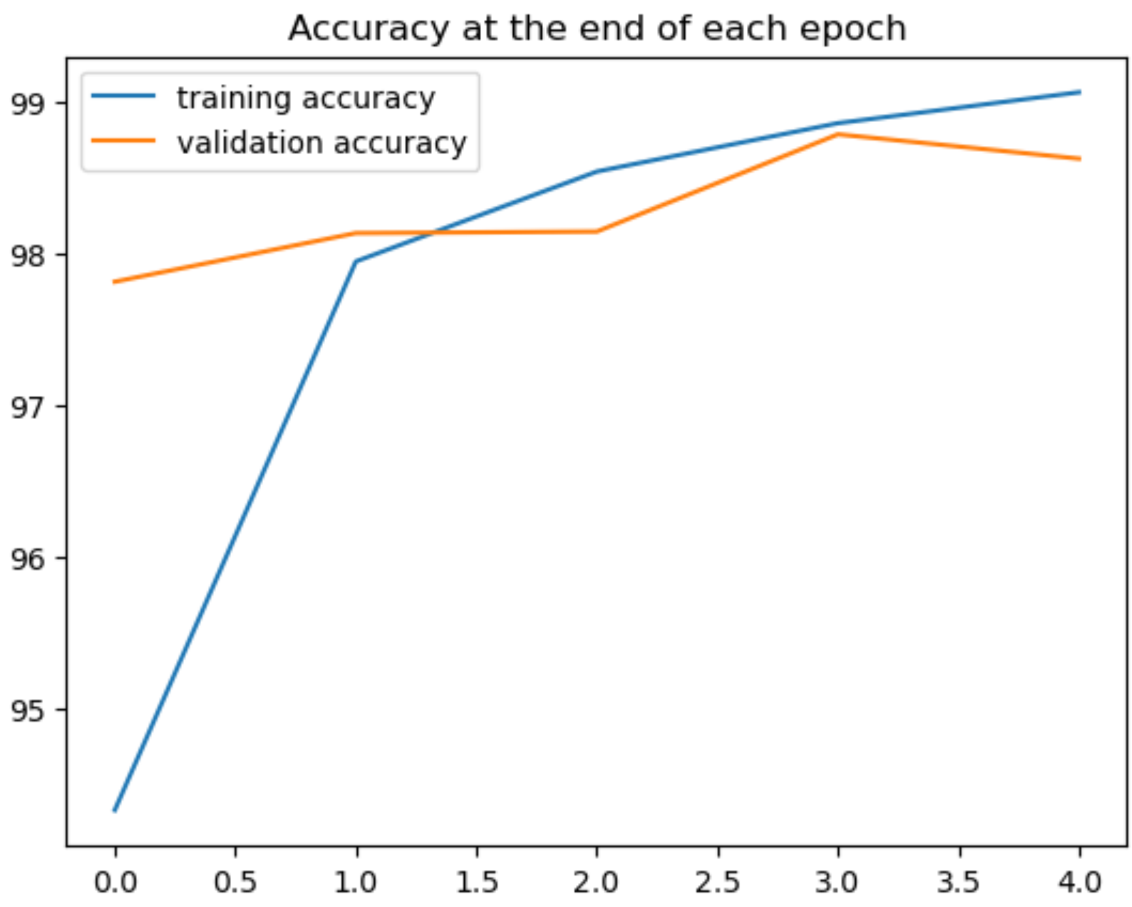
```
plt.plot(train_losses, label='training loss')
plt.plot(test_losses, label='validation loss')
plt.title('Loss at the end of each epoch')
plt.legend();
```



In [57]:
```
test_losses
```

Out[57]:
```
[tensor(0.0023),
 tensor(0.0024),
 tensor(0.0008),
 tensor(0.0001),
 tensor(6.7299e-05)]
```

In [58]:
```
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();
```



In [59]:
```
#Evaluate Test data
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()
print(f'Test accuracy: {correct.item()}/{len(test_data)} = {correct.item(
```

```
Test accuracy: 9862/10000 =  98.620%
```

In [64]:
```
#Confusion Matrix
# print a row of values for reference
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)))
```

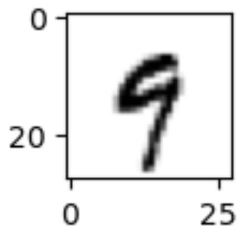
```
[[    0    1    2    3    4    5    6    7    8    9]]

[[ 975    0    0    0    0    3    4    0    4    0]
 [    0 1134    4    1    1    0    4    5    0    3]
 [    2    0 1019    1    0    0    0    5    2    0]
 [    0    1    0 1002    0   19    0    0    1    0]
 [    0    0    1    0  978    0    2    0    1   16]
 [    0    0    0    2    0  858    0    0    0    2]
 [    1    0    0    0    2    2  947    0    0    1]
 [    1    0    6    2    0    0    0 1015    3    6]
 [    1    0    2    2    0    3    1    1  962    9]
 [    0    0    0    0    1    7    0    2    1  972]]
```

Run a new image through the model

We can also pass a single image through the model to obtain a prediction. Pick a number from 0 to 9999, assign it to "x", and we'll use that value to select a number from the MNIST test set.

In [61]:

```
x = 2019
plt.figure(figsize=(1,1))
plt.imshow(test_data[x][0].reshape((28,28)), cmap="gist_yarg");
```



In [62]:

```
model.eval()
with torch.no_grad():
    new_pred = model(test_data[x][0].view(1,1,28,28)).argmax()
print("Predicted value:",new_pred.item())
```

```
Predicted value: 9
```