mnist-bharat-number-recognisition

November 8, 2023

1 Recognising Hand written numbers

Importing Libraries

[32]:
```
import tensorflow
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense,Flatten
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
```

downloading the dataset by dividing into train&test

[33]:
```
(x_train,y_train), (x_test,y_test)= keras.datasets.mnist.load_data()
```

[34]:
```
x_train.shape
```

[34]:
```
(60000, 28, 28)
```

[35]:
```
x_train
```

[35]:
```
array([[[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]],

       [[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]],

       [[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]],

       …,

       [[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]],

       [[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]],

       [[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]]], dtype=uint8)
```

[36]:
```
x_test
```

[36]:
```
array([[[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]],

       [[0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0],
        …,
        [0, 0, 0, …, 0, 0, 0],
```

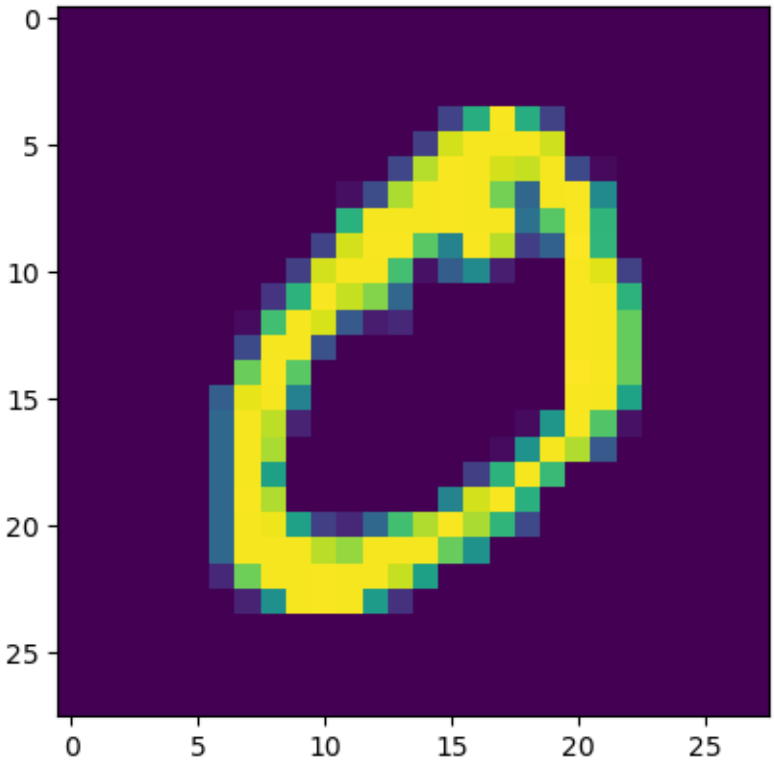
```
        [0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0]],

       [[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]],

       …,

       [[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]],

       [[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]],

       [[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]]], dtype=uint8)
```

[37]:
```
y_train
```

[37]:
```
array([5, 0, 4, …, 5, 6, 8], dtype=uint8)
```

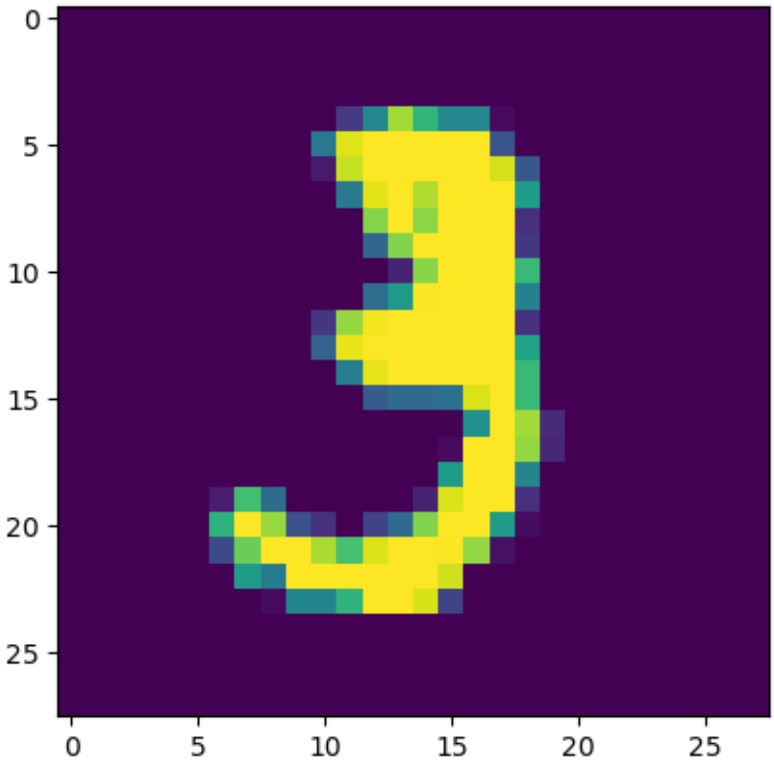
[38]:
```
plt.imshow(x_train[1])
```

[38]:
```
<matplotlib.image.AxesImage at 0x245295819a0>
```



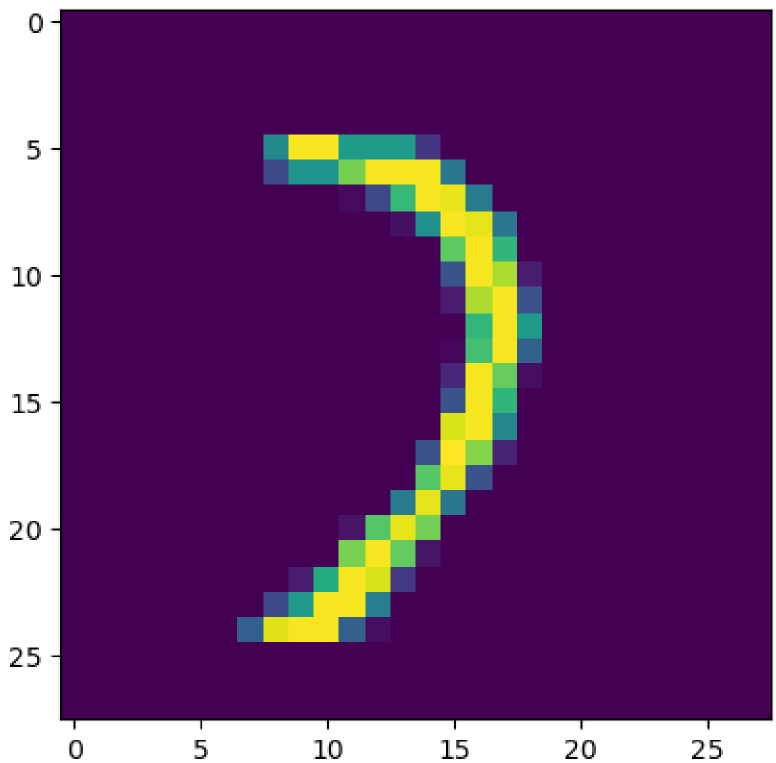
[39]:
```
plt.imshow(x_train[10])
```

[39]: <matplotlib.image.AxesImage at 0x2452ad62d30>



[70]:
```
plt.imshow(x_train[140])
```

[70]:
```
<matplotlib.image.AxesImage at 0x24517923730>
```



[41]:
```
x_train[1] #It shows pixel values of first image. Generally its ranges from
↪0-255 so we need to normalize to 0-1 and train model
```

[41]:
```
array([[  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,   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,   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,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,  51, 159, 253, 159,  50,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  48, 238, 252, 252, 252, 237,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
  54, 227, 253, 252, 239, 233, 252,  57,   6,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  10,  60,
 224, 252, 253, 252, 202,  84, 252, 253, 122,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 163, 252,
 252, 252, 253, 252, 252,  96, 189, 253, 167,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  51, 238, 253,
 253, 190, 114, 253, 228,  47,  79, 255, 168,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,  48, 238, 252, 252,
 179,  12,  75, 121,  21,   0,   0, 253, 243,  50,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,  38, 165, 253, 233, 208,
  84,   0,   0,   0,   0,   0,   0, 253, 252, 165,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   7, 178, 252, 240,  71,  19,
  28,   0,   0,   0,   0,   0,   0, 253, 252, 195,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,  57, 252, 252,  63,   0,   0,
   0,   0,   0,   0,   0,   0,   0, 253, 252, 195,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0, 198, 253, 190,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0, 255, 253, 196,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,  76, 246, 252, 112,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0, 253, 252, 148,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,  85, 252, 230,  25,   0,   0,   0,
   0,   0,   0,   0,   0,   7, 135, 253, 186,  12,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,  85, 252, 223,   0,   0,   0,   0,
   0,   0,   0,   0,   7, 131, 252, 225,  71,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,  85, 252, 145,   0,   0,   0,   0,
   0,   0,   0,  48, 165, 252, 173,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,  86, 253, 225,   0,   0,   0,   0,
   0,   0, 114, 238, 253, 162,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,  85, 252, 249, 146,  48,  29,  85,
 178, 225, 253, 223, 167,  56,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,  85, 252, 252, 252, 229, 215, 252,
 252, 252, 196, 130,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
```

```
       [  0,   0,   0,   0,   0,   0,  28, 199, 252, 252, 253, 252, 252,
        233, 145,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  25, 128, 252, 253, 252, 141,
         37,   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,   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,   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,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0]], dtype=uint8)
```

Normalising the pixel values to 0-1

[42]:
```
X_train= x_train/255
X_test= x_test/255
```

Building a simple neural network

[43]:
```
model=Sequential()
model.add(Flatten(input_shape=(28,28)))
model.add(Dense(128,activation='relu'))
model.add(Dense(10,activation='softmax'))
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 784)               0

 dense_2 (Dense)             (None, 128)               100480

 dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

[44]:
```
model.
  ↪compile(loss='sparse_categorical_crossentropy',optimizer='Adam',metrics=['accuracy'])
model.fit(X_train,y_train,epochs=10,validation_split=0.2)
```

```
Epoch 1/10
1500/1500 [==============================] - 10s 6ms/step - loss: 0.2927 -
accuracy: 0.9176 - val_loss: 0.1595 - val_accuracy: 0.9543
Epoch 2/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.1327 -
accuracy: 0.9614 - val_loss: 0.1294 - val_accuracy: 0.9634
Epoch 3/10
1500/1500 [==============================] - 8s 5ms/step - loss: 0.0899 -
accuracy: 0.9736 - val_loss: 0.1099 - val_accuracy: 0.9669
Epoch 4/10
1500/1500 [==============================] - 7s 5ms/step - loss: 0.0661 -
accuracy: 0.9803 - val_loss: 0.0966 - val_accuracy: 0.9706
Epoch 5/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0512 -
accuracy: 0.9845 - val_loss: 0.0927 - val_accuracy: 0.9727
Epoch 6/10
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0397 -
accuracy: 0.9881 - val_loss: 0.0888 - val_accuracy: 0.9743
Epoch 7/10
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0321 -
accuracy: 0.9902 - val_loss: 0.1006 - val_accuracy: 0.9719
Epoch 8/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0250 -
accuracy: 0.9924 - val_loss: 0.0882 - val_accuracy: 0.9756
Epoch 9/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0208 -
accuracy: 0.9940 - val_loss: 0.0930 - val_accuracy: 0.9744
Epoch 10/10
1500/1500 [==============================] - 8s 5ms/step - loss: 0.0169 -
accuracy: 0.9953 - val_loss: 0.0957 - val_accuracy: 0.9747
```

[44]:
```
<keras.callbacks.History at 0x2452ade18b0>
```

[45]:
```
model.predict(X_test)
```

```
313/313 [==============================] - 1s 3ms/step
```

[45]:
```
array([[8.5839508e-10, 1.8917118e-12, 7.5510505e-09, …, 9.9999893e-01,
        2.4837241e-10, 2.8400832e-08],
       [1.0145028e-09, 1.0243253e-06, 9.9999893e-01, …, 4.0688998e-15,
        9.2062621e-11, 2.5614173e-14],
       [3.4333041e-04, 9.9575078e-01, 1.8154395e-04, …, 2.6668913e-03,
        6.5478130e-04, 5.6096642e-06],
```

```
       …,
       [7.9992448e-17, 1.2983661e-12, 1.4181497e-14, …, 3.6875809e-07,
        2.3453754e-09, 1.0009860e-06],
       [2.4981875e-15, 1.5609898e-15, 1.0015844e-15, …, 2.4089571e-12,
        4.8702386e-08, 4.5870964e-14],
       [5.7340820e-11, 2.1233560e-16, 4.0711426e-10, …, 3.9116966e-16,
        4.8753010e-13, 3.9655894e-16]], dtype=float32)
```

[48]:
```
y_prob=model.predict(X_test)
```

```
313/313 [==============================] - 1s 3ms/step
```

[52]:
```
y_prob
```

[52]:
```
array([[8.5839508e-10, 1.8917118e-12, 7.5510505e-09, …, 9.9999893e-01,
        2.4837241e-10, 2.8400832e-08],
       [1.0145028e-09, 1.0243253e-06, 9.9999893e-01, …, 4.0688998e-15,
        9.2062621e-11, 2.5614173e-14],
       [3.4333041e-04, 9.9575078e-01, 1.8154395e-04, …, 2.6668913e-03,
        6.5478130e-04, 5.6096642e-06],
       …,
       [7.9992448e-17, 1.2983661e-12, 1.4181497e-14, …, 3.6875809e-07,
        2.3453754e-09, 1.0009860e-06],
       [2.4981875e-15, 1.5609898e-15, 1.0015844e-15, …, 2.4089571e-12,
        4.8702386e-08, 4.5870964e-14],
       [5.7340820e-11, 2.1233560e-16, 4.0711426e-10, …, 3.9116966e-16,
        4.8753010e-13, 3.9655894e-16]], dtype=float32)
```

[53]:
```
y_prob.argmax(axis=1)
```

[53]:
```
array([7, 2, 1, …, 4, 5, 6], dtype=int64)
```

[54]:
```
y_pred = y_prob.argmax(axis=1)
```

[55]:
```
accuracy_score(y_test,y_pred)
```

[55]:
```
0.9756
```

[56]:
```
history = model.fit(X_train,y_train,epochs=10,validation_split=0.2)
```

```
Epoch 1/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0138 -
accuracy: 0.9960 - val_loss: 0.0904 - val_accuracy: 0.9768
Epoch 2/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0112 -
accuracy: 0.9966 - val_loss: 0.0936 - val_accuracy: 0.9767
Epoch 3/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0114 -
```

```
accuracy: 0.9964 - val_loss: 0.0945 - val_accuracy: 0.9779
Epoch 4/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0079 -
accuracy: 0.9980 - val_loss: 0.0942 - val_accuracy: 0.9793
Epoch 5/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0091 -
accuracy: 0.9974 - val_loss: 0.1125 - val_accuracy: 0.9743
Epoch 6/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0062 -
accuracy: 0.9983 - val_loss: 0.1046 - val_accuracy: 0.9769
Epoch 7/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0065 -
accuracy: 0.9980 - val_loss: 0.1281 - val_accuracy: 0.9738
Epoch 8/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0067 -
accuracy: 0.9980 - val_loss: 0.1024 - val_accuracy: 0.9792
Epoch 9/10
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0048 -
accuracy: 0.9986 - val_loss: 0.1163 - val_accuracy: 0.9786
Epoch 10/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0055 -
accuracy: 0.9981 - val_loss: 0.1307 - val_accuracy: 0.9749
```

[57]:
```
y_prob = model.predict(X_test)
```

```
313/313 [==============================] - 1s 3ms/step
```

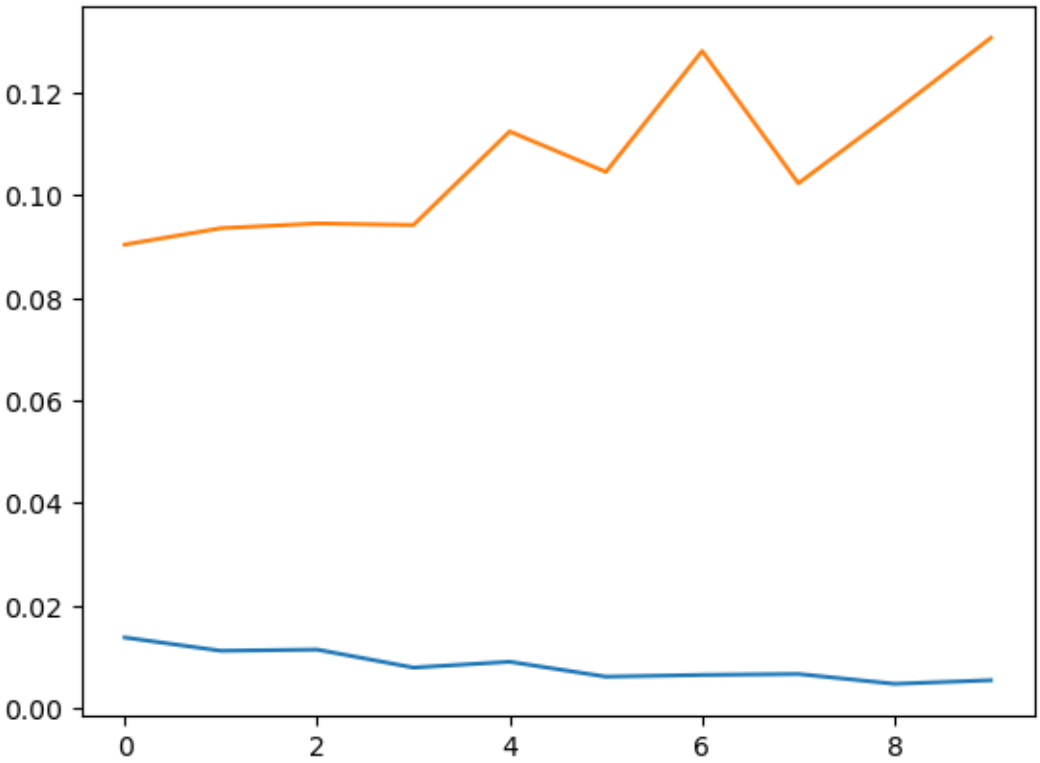
[58]:
```
y_pred = y_prob.argmax(axis=1)
```

[59]:
```
accuracy_score(y_test,y_pred)
```

[59]: 0.9757

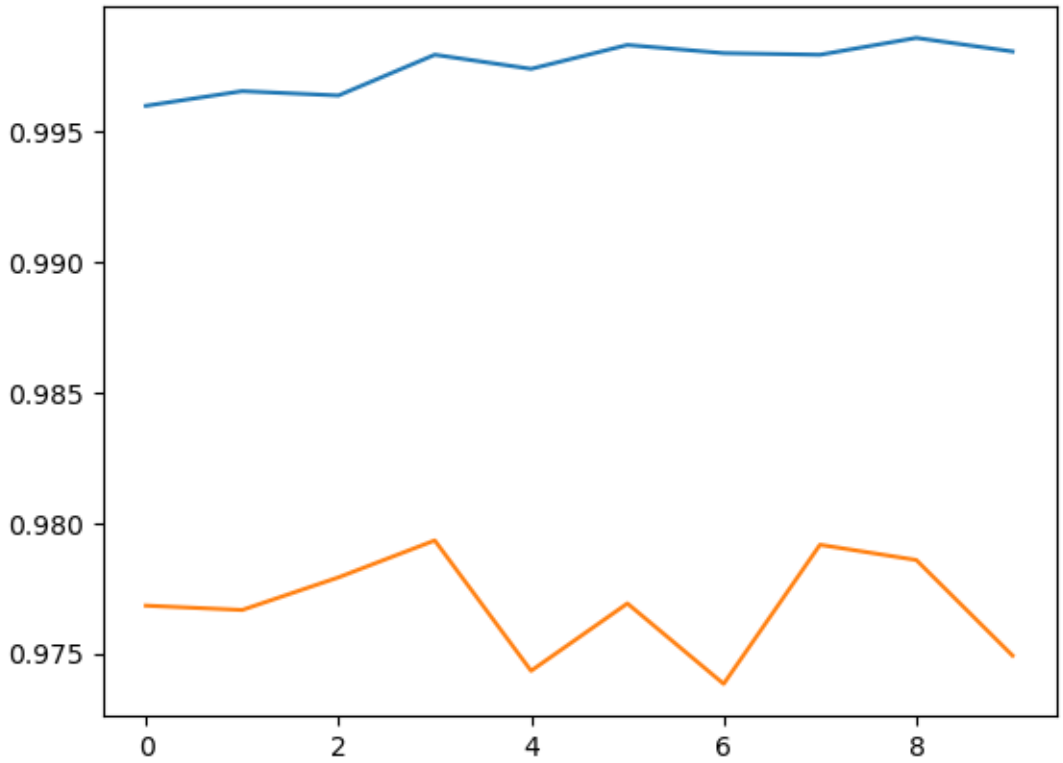
[60]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```

[60]: [<matplotlib.lines.Line2D at 0x24519603760>]



[61]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
```

[61]: [<matplotlib.lines.Line2D at 0x24519bf1fa0>]



[62]:
```
X_test
```

[62]:
```
array([[[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.]],

       [[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.]],

       [[0., 0., 0., …, 0., 0., 0.],
        [0., 0., 0., …, 0., 0., 0.],
        [0., 0., 0., …, 0., 0., 0.],
        …,
```

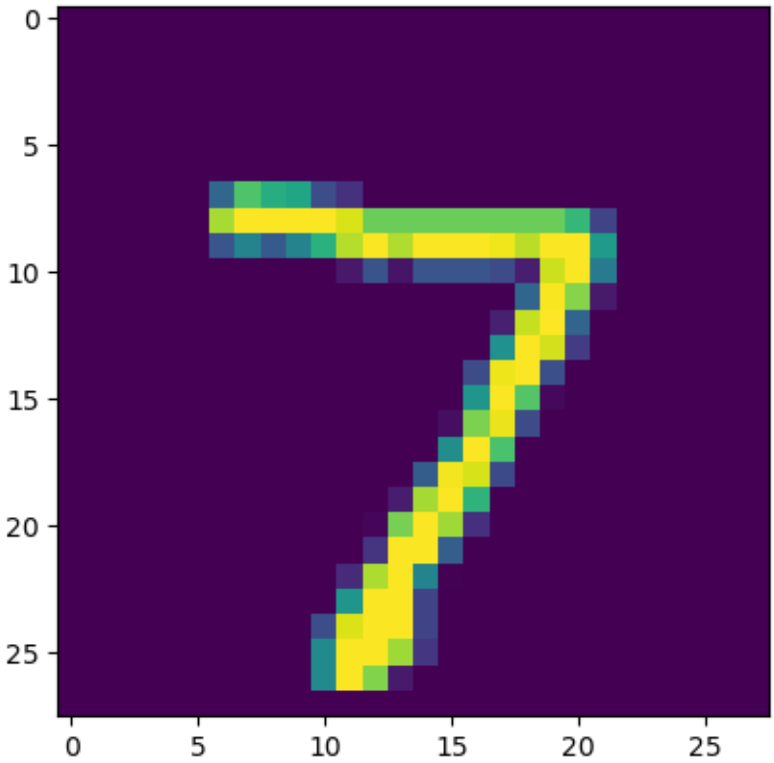
```
        [0., 0., 0., …, 0., 0., 0.],
        [0., 0., 0., …, 0., 0., 0.],
        [0., 0., 0., …, 0., 0., 0.]],

       …,

       [[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.]],

       [[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.]],

       [[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.]]])
```

[63]:
```
plt.imshow
```

[63]:
```
<function matplotlib.pyplot.imshow(X, cmap=None, norm=None, *, aspect=None,
interpolation=None, alpha=None, vmin=None, vmax=None, origin=None, extent=None,
interpolation_stage=None, filternorm=True, filterrad=4.0, resample=None,
url=None, data=None, **kwargs)>
```

[64]:
```
plt.imshow(X_test[0])
```

[64]:
```
<matplotlib.image.AxesImage at 0x24517839df0>
```



[65]:
```
model.predict(X_test[0].reshape(1,28,28))
```

```
1/1 [==============================] - 0s 58ms/step
```

[65]:
```
array([[3.7149667e-12, 2.4038387e-16, 1.2113532e-11, 6.8904434e-08,
        1.6952081e-22, 6.3673563e-12, 9.1037041e-18, 9.9999988e-01,
        9.5047105e-14, 1.2422003e-08]], dtype=float32)
```

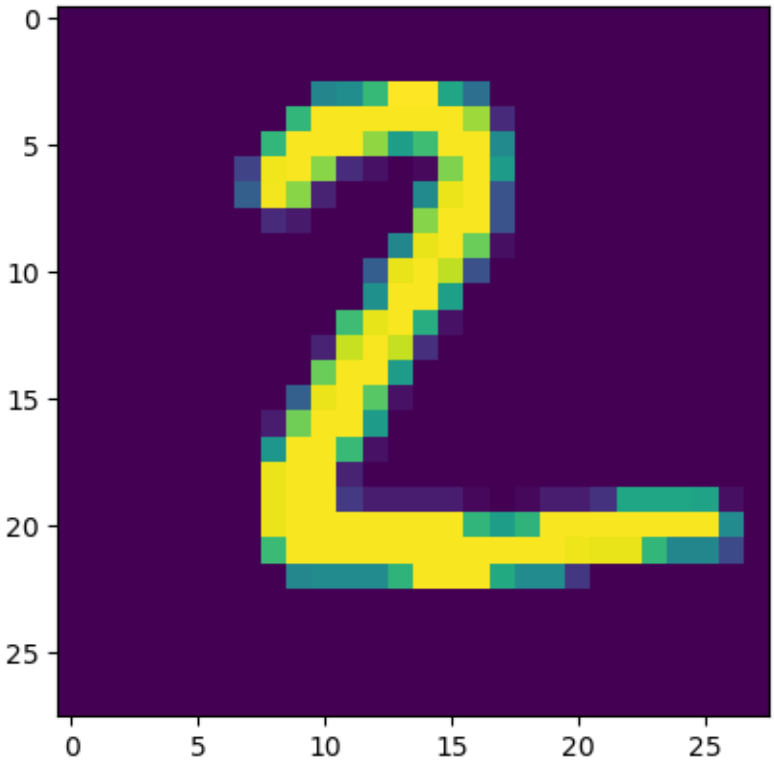
[67]:
```
model.predict(X_test[0].reshape(1,28,28)).argmax(axis=1)
```

```
1/1 [==============================] - 0s 41ms/step
```

[67]:
```
array([7], dtype=int64)
```

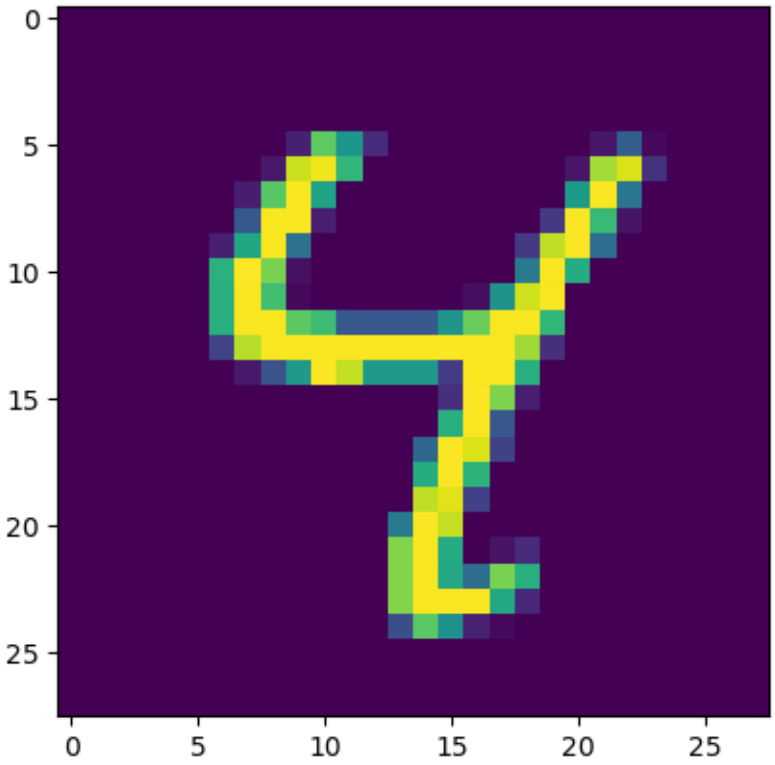
[68]:
```
plt.imshow(X_test[1])
```

[68]:
```
<matplotlib.image.AxesImage at 0x245178f12b0>
```



[71]:
```
plt.imshow(X_test[6])
```

[71]: <matplotlib.image.AxesImage at 0x2451798c1c0>



[72]:
```
model.predict(X_test[1].reshape(1,28,28))
```

```
1/1 [==============================] - 0s 50ms/step
```

[72]:
```
array([[9.0836498e-19, 2.2006896e-11, 1.0000000e+00, 1.1269086e-19,
        8.1333986e-28, 2.6898412e-24, 4.9036062e-20, 3.7776017e-24,
        9.2106045e-16, 8.8455555e-25]], dtype=float32)
```

[ ]:

[ ]: