MNIST Handwritten Digit Clasification using Deep Learning (Neural Network)

Importing the Dependencies

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
#from google.colab.patches import cv2_imshow
from PIL import Image
import tensorflow as tf
tf.random.set_seed(3)
from tensorflow import keras
from keras.datasets import mnist
from tensorflow.math import confusion_matrix
```

Loading the MNIST data from keras.datasets

In [3]:
```
(X_train, Y_train), (X_test, Y_test) =  mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz)
11490434/11490434 [==============================] - 3s 0us/step
```

In [4]:
```
type(X_train)
```

Out[4]:
```
numpy.ndarray
```

In [6]:
```
# shape of the numpy arrays
print(X_train.shape, Y_train.shape, X_test.shape, Y_test.shape)
```

```
(60000, 28, 28) (60000,) (10000, 28, 28) (10000,)
```

Training data = 60,000 Images

Test data = 10,000 Images

Image dimension --> 28 x 28

Grayscale Image --> 1 channel

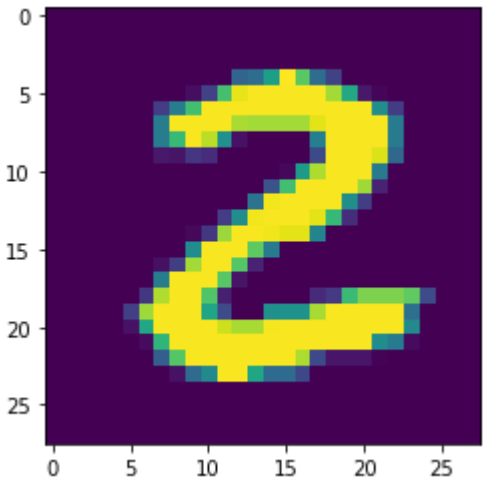
In [8]:
```
# printing the 10th image

print(X_train[9])
```

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0 189 190   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0 143 247 153   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0 136 247 242  86   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0 192 252 187   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  62 185  18   0   0   0
    0  89 236 217  47   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0 216 253  60   0   0   0
    0 212 255  81   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0 206 252  68   0   0   0
   48 242 253  89   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 131 251 212  21   0   0  11
  167 252 197   5   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  29 232 247  63   0   0   0 153
  252 226   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  45 219 252 143   0   0   0 116 249
  252 103   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   4  96 253 255 253 200 122   7  25 201 250
  158   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  92 252 252 253 217 252 252 200 227 252 231
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  87 251 247 231  65  48 189 252 252 253 252 251
  227  35   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0 190 221  98   0   0   0  42 196 252 253 252 252
  162   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0 111  29   0   0   0   0  62 239 252  86  42  42
   14   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  15 148 253 218   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 121 252 231  28   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  31 221 251 129   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 218 252 160   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 122 252  82   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   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 [ ]:
```
print(X_train[10].shape)
```

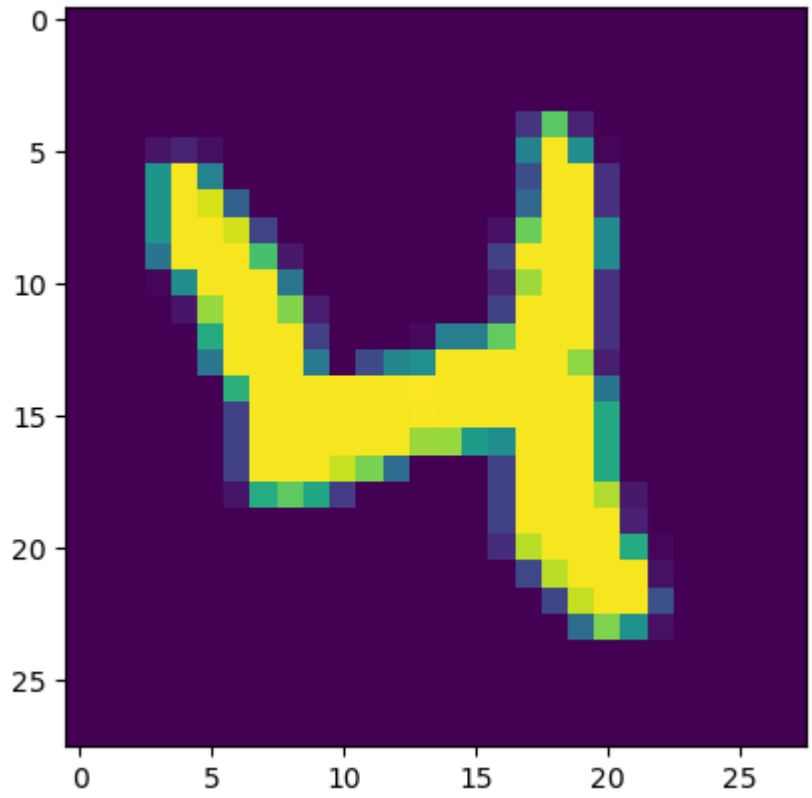
```
(28, 28)
```

In [ ]:
```
# displaying the image

plt.imshow(X_train[25])
plt.show()

# print the corresponding label
print(Y_train[25])
```




```
2
```

In [9]:
```
# displaying the image

plt.imshow(X_train[20])
plt.show()

# print the corresponding label
print(Y_train[20])
```




```
4
```

Image Lables

In [13]:
```
print(Y_train.shape, Y_test.shape)
```

```
(60000,) (10000,)
```

In [12]:
```
# unique values in Y_train
print(np.unique(Y_train))

# unique values in Y_test
print(np.unique(Y_test))
```

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

We can use these labels as such or we can also apply One Hot Encoding

```
All the images have the same dimensions in this dataset, If not, we have to resize
all the images to a common dimension
```

In [10]:
```
# scaling the values

X_train = X_train/255
X_test = X_test/255
```

In [11]:
```
# printing the 10th image

print(X_train[10])
```

```
[[0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
```

Building the Neural Network

In [20]:
```
# setting up the layers of the Neural  Network

model = keras.Sequential([
                          keras.layers.Flatten(input_shape=(28,28)),
                          keras.layers.Dense(64, activation='relu'),
                          keras.layers.Dense(128, activation='relu'),
                          keras.layers.Dense(10, activation='sigmoid')
])
```

In [21]:
```
# compiling the Neural Network

model.compile(optimizer='adam',
              loss = 'sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [22]:
```
# training the Neural Network

model.fit(X_train, Y_train, epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.2612 - accuracy:
0.9233
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1177 - accuracy:
0.9639
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0835 - accuracy:
0.9737
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0656 - accuracy:
0.9792
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0541 - accuracy:
0.9830
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0445 - accuracy:
0.9851
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0374 - accuracy:
0.9875
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0323 - accuracy:
0.9891
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0306 - accuracy:
0.9900
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0252 - accuracy:
0.9917
```

Out[22]:
```
<keras.src.callbacks.History at 0x25a6df89490>
```

Training data accuracy = 99.17%

Accuracy on Test data:

In [23]:
```
loss, accuracy = model.evaluate(X_test, Y_test)
print(accuracy)
```

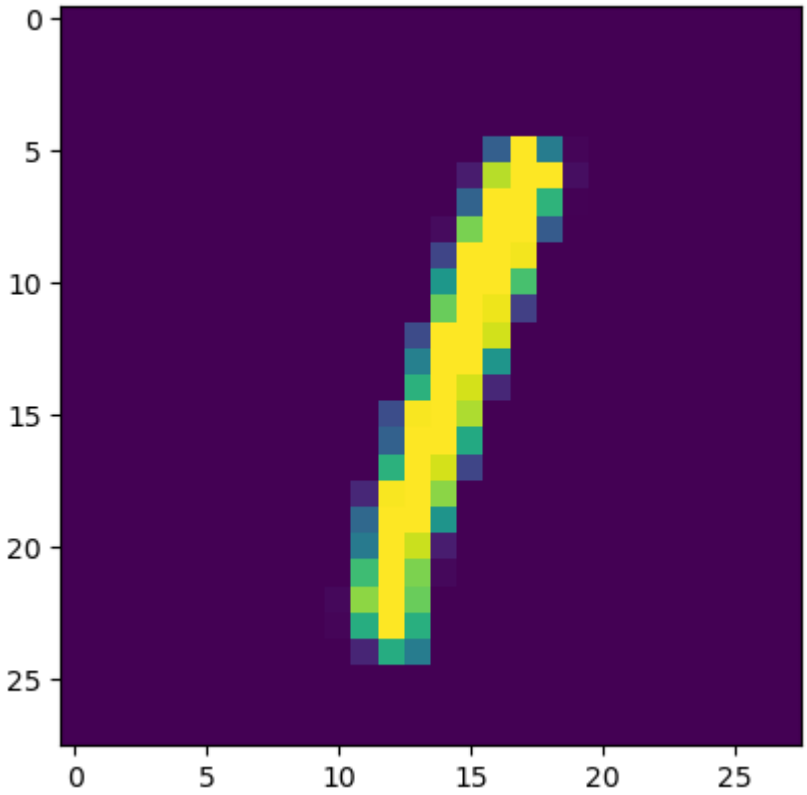
```
313/313 [==============================] - 1s 2ms/step - loss: 0.1005 - accuracy: 0.
9757
0.9757000207901001
```

Test data accuracy = 97.57%

In [24]:
```
print(X_test.shape)
```

```
(10000, 28, 28)
```

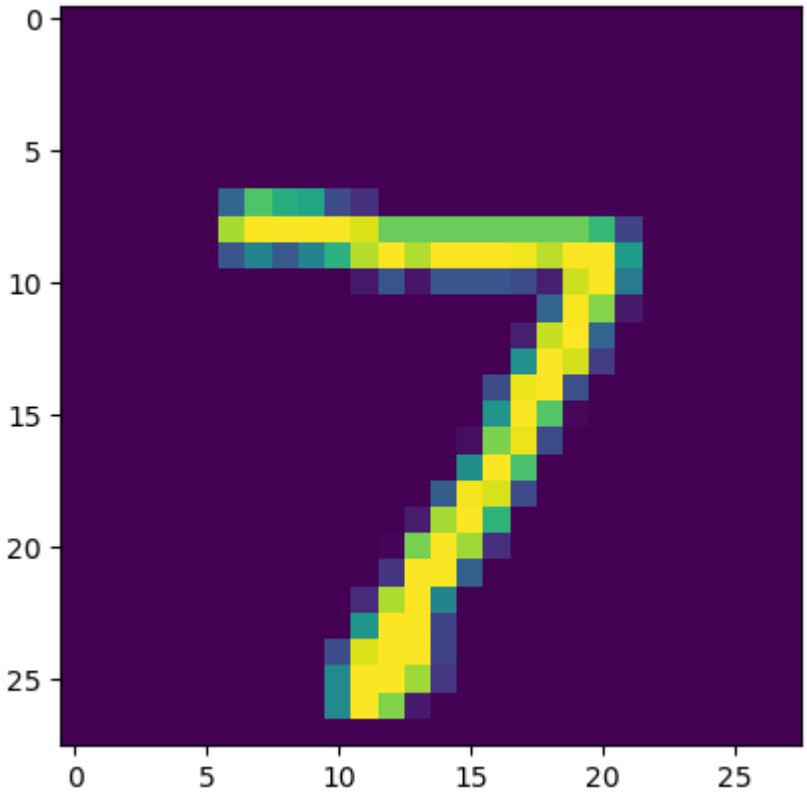
In [28]:
```
plt.imshow(X_test[5])
plt.show()
```



In [30]:
```
print(Y_test[5])
```

```
1
```

In [25]:
```
# first data point in X_test
plt.imshow(X_test[0])
plt.show()
```



In [32]:
```
print(Y_test[0])
```

```
7
```

In [33]:
```
Y_pred = model.predict(X_test)
```

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

In [34]:
```
print(Y_pred.shape)
```

```
(10000, 10)
```

In [35]:
```
print(Y_pred[0])
```

```
[2.0773856e-05 3.2481041e-03 4.6754088e-02 7.4509448e-01 1.0255403e-06
 7.3025115e-03 1.1680031e-09 9.9999982e-01 2.8102559e-03 4.0035740e-02]
```

model.predict() gives the prediction probability of each class for that data point

In [ ]:
```
# converting the prediction probabilities to class label

label_for_first_test_image = np.argmax(Y_pred[0])
print(label_for_first_test_image)
```

```
7
```

In [ ]:
```

```

In [ ]:
```
# converting the prediction probabilities to class label for all test data points
Y_pred_labels = [np.argmax(i) for i in Y_pred]
print(Y_pred_labels)
```

```
8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 8, 4, 5, 6, 7, 8, 9, 8, 6, 5, 0, 6,
8, 9, 4, 1, 9, 3, 8, 0, 4, 8, 9, 1, 4, 0, 5, 5, 2, 1, 5, 4, 0, 7, 6, 0, 1, 7, 0,
6, 8, 9, 5, 1, 7, 9, 8, 6, 0, 8, 1, 7, 7, 1, 3, 2, 9, 1, 4, 2, 0, 0, 7, 8, 4, 6,
4, 9, 9, 8, 4, 7, 6, 5, 6, 5, 6, 9, 6, 3, 0, 2, 4, 6, 9, 0, 2, 5, 5, 1, 9, 3, 9,
7, 8, 7, 2, 2, 5, 7, 9, 8, 2, 1, 9, 1, 3, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2,
3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 6, 5, 3, 0, 7, 0, 4, 1,
4, 3, 6, 7, 2, 3, 1, 2, 1, 2, 9, 6, 0, 1, 3, 0, 2, 7, 5, 7, 6, 2, 9, 1, 9, 0, 6,
0, 6, 0, 2, 0, 6, 1, 5, 8, 4, 3, 0, 1, 5, 4, 4, 8, 5, 7, 5, 7, 8, 3, 4, 8, 8, 5,
2, 9, 7, 1, 3, 8, 1, 0, 7, 5, 3, 6, 9, 4, 7, 7, 9, 9, 3, 4, 4, 3, 8, 6, 2, 0, 1,
2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8,
9, 0, 8, 3, 9, 5, 5, 2, 6, 8, 4, 9, 1, 7, 1, 2, 3, 5, 9, 6, 9, 1, 1, 1, 2, 9, 5,
6, 8, 1, 2, 0, 7, 7, 5, 8, 2, 9, 8, 9, 0, 4, 6, 7, 1, 3, 4, 5, 6, 0, 3, 6, 8, 7,
0, 4, 2, 7, 4, 7, 5, 4, 3, 4, 2, 8, 1, 5, 1, 2, 0, 2, 5, 6, 4, 3, 0, 0, 0, 3, 3,
5, 7, 0, 6, 4, 8, 8, 6, 3, 4, 6, 9, 9, 8, 2, 7, 7, 1, 0, 1, 2, 3, 4, 5, 6, 7, 8,
9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 0, 1, 2, 3, 4, 5, 6, 7, 8, 2, 1, 7, 2, 5, 0, 8, 0,
2, 7, 8, 8, 3, 6, 0, 2, 7, 6, 6, 1, 2, 8, 8, 7, 7, 4, 7, 7, 3, 7, 4, 5, 4, 3, 3,
8, 4, 1, 1, 9, 7, 4, 3, 7, 3, 3, 0, 2, 5, 5, 6, 6, 3, 5, 2, 5, 9, 9, 8, 4, 1, 0,
6, 0, 9, 6, 8, 8, 5, 6, 1, 1, 9, 8, 9, 2, 3, 5, 5, 9, 4, 2, 1, 9, 2, 9, 2, 0, 6,
0, 4, 0, 0, 1, 2, 3, 4, 7, 8, 9, 0, 1, 2, 3, 7, 8, 9, 0, 1, 2, 3, 4, 7, 8, 9, 7,
```

Y_test --> True labels

Y_pred_labels --> Predicted Labels

Confusin Matrix

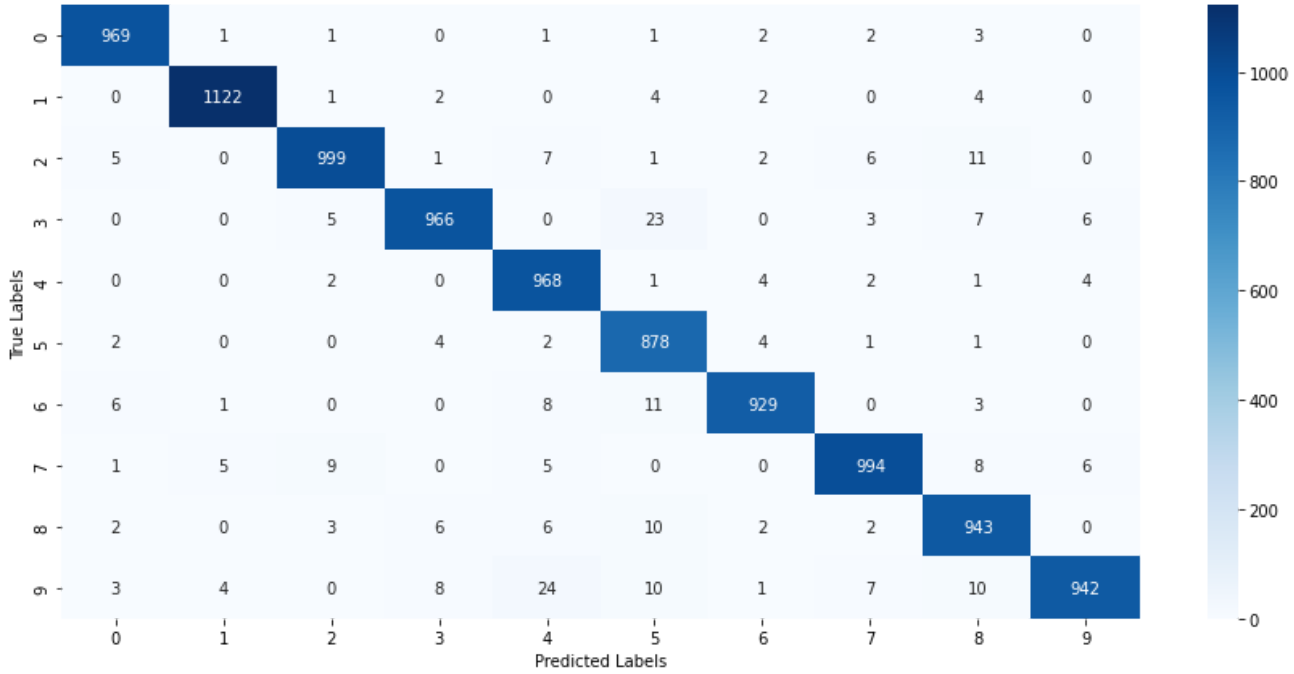
In [ ]:
```
conf_mat = confusion_matrix(Y_test, Y_pred_labels)
```

In [ ]:
```
print(conf_mat)
```

```
tf.Tensor(
[[ 969    1    1    0    1    1    2    2    3    0]
 [   0 1122    1    2    0    4    2    0    4    0]
 [   5    0  999    1    7    1    2    6   11    0]
 [   0    0    5  966    0   23    0    3    7    6]
 [   0    0    2    0  968    1    4    2    1    4]
 [   2    0    0    4    2  878    4    1    1    0]
 [   6    1    0    0    8   11  929    0    3    0]
 [   1    5    9    0    5    0    0  994    8    6]
 [   2    0    3    6    6   10    2    2  943    0]
 [   3    4    0    8   24   10    1    7   10  942]], shape=(10, 10), dtype=int32)
```

In [ ]:
```
plt.figure(figsize=(15,7))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues')
plt.ylabel('True Labels')
plt.xlabel('Predicted Labels')
```

Out[30]:
```
Text(0.5, 42.0, 'Predicted Labels')
```



Building a Predictive System

Prediction image link:
https://camo.githubusercontent.com/3d9666a8f0c5658667292b74ca19295827c2b22a0e903db283998ae2
(https://camo.githubusercontent.com/3d9666a8f0c5658667292b74ca19295827c2b22a0e903db283998ae2

In [68]:
```
input_image = cv2.imread('MNIST_digit.png')
```

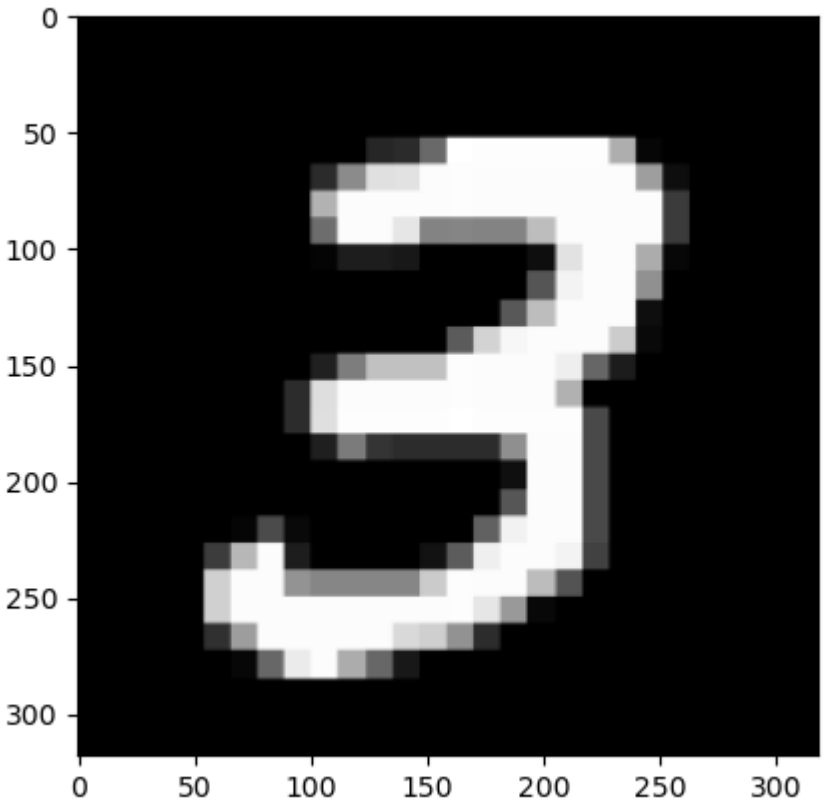
In [69]:
```
type(input_image)
```

Out[69]:
```
numpy.ndarray
```

In [70]:
```
print(input_image)
```

```
[[[0 0 0]
  [0 0 0]
  [0 0 0]
  ...
  [0 0 0]
  [0 0 0]
  [0 0 0]]

 [[0 0 0]
  [0 0 0]
  [0 0 0]
  ...
  [0 0 0]
  [0 0 0]
  [0 0 0]]

 [[0 0 0]
  [0 0 0]
  [0 0 0]
  ...
  [0 0 0]
  [0 0 0]
  [0 0 0]]

 ...

 [[0 0 0]
  [0 0 0]
  [0 0 0]
  ...
  [0 0 0]
  [0 0 0]
  [0 0 0]]

 [[0 0 0]
  [0 0 0]
  [0 0 0]
  ...
  [0 0 0]
  [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 [87]:
```
plt.imshow(input_image)
```



Out[87]:
```
<matplotlib.image.AxesImage at 0x25a1a2c1d50>
```

In [74]:
```
input_image.shape
```

Out[74]:
```
(318, 318, 3)
```

In [75]:
```
grayscale = cv2.cvtColor(input_image, cv2.COLOR_RGB2GRAY)
```

In [76]:
```
grayscale.shape
```

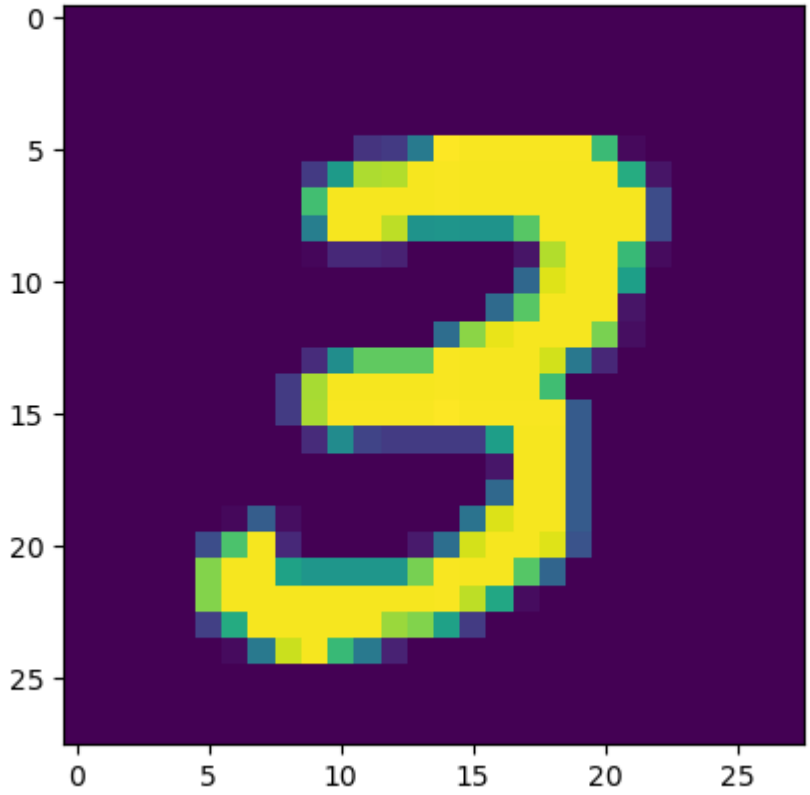
Out[76]:
```
(318, 318)
```

In [77]:
```
input_image_resize = cv2.resize(grayscale, (28, 28))
```

In [78]:
```
input_image_resize.shape
```

Out[78]:
```
(28, 28)
```

In [88]:
```
plt.imshow(input_image_resize)
```



Out[88]:
```
<matplotlib.image.AxesImage at 0x25a1a3534d0>
```

In [81]:
```
input_image_resize = input_image_resize/255
```

In [82]:
```
type(input_image_resize)
```

Out[82]:
```
numpy.ndarray
```

In [83]:
```
image_reshaped = np.reshape(input_image_resize, [1,28,28])
```

In [84]:
```
input_prediction = model.predict(image_reshaped)
print(input_prediction)
```

```
1/1 [==============================] - 0s 266ms/step
[[1.4969674e-17 6.4147948e-10 3.2105504e-06 1.0000000e+00 2.6870534e-15
  3.6363667e-06 5.8934504e-23 1.8358632e-04 5.4213418e-05 3.0160829e-01]]
```

In [85]:
```
input_pred_label = np.argmax(input_prediction)
```

In [86]:
```
print(input_pred_label)
```

```
3
```

Predictive System

In [ ]:
```
input_image_path = input('Path of the image to be predicted: ')

input_image = cv2.imread(input_image_path)

cv2_imshow(input_image)

grayscale = cv2.cvtColor(input_image, cv2.COLOR_RGB2GRAY)

input_image_resize = cv2.resize(grayscale, (28, 28))

input_image_resize = input_image_resize/255

image_reshaped = np.reshape(input_image_resize, [1,28,28])

input_prediction = model.predict(image_reshaped)

input_pred_label = np.argmax(input_prediction)

print('The Handwritten Digit is recognised as ', input_pred_label)
```

```
Path of the image to be predicted: /content/MNIST_digit.png
```

```
The Handwritten Digit is recognised as  3
```

In [ ]: