Importing Libraries

```
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

```
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
type(X_train)
```

```
numpy.ndarray
```

```
# 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,)
```

Above output shows that

Training data = 60,000 Images

Test data = 10,000 Images

Image dimension ---> 28x28

Grayscale Image ---> 1 Channel

```
# Pinting the 10th Image
print(X_train[10])
```

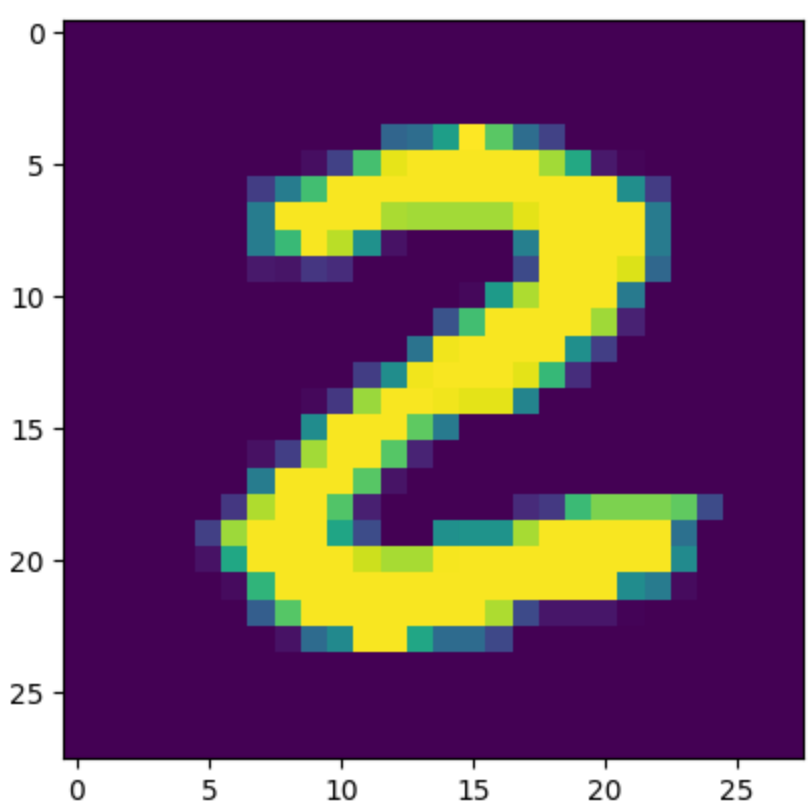
```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   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   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  42 118 219 166 118 118   6
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 103 242 254 254 254 254 254  66
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  18 232 254 254 254 254 254 238
   70   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 104 244 254 224 254 254 254
  141   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0 207 254 210 254 254 254
   34   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  84 206 254 254 254 254
   41   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  24 209 254 254 254
  171   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  91 137 253 254 254 254
  112   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  40 214 250 254 254 254 254 254
   34   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  81 247 254 254 254 254 254 254
  146   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 110 246 254 254 254 254 254
  171   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  73  89  89  93 240 254
  171   0   0   0   0   0   0   0   0   0]
```

```
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1 128 254
 219  31   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   7 254 254
 214  28   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 138 254 254
 116   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0  19 177  90   0   0   0   0   0  25 240 254 254
  34   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0 164 254 215  63  36   0  51  89 206 254 254 139
   8   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0  57 197 254 254 222 180 241 254 254 253 213  11
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0 140 105 254 254 254 254 254 254 236   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   7 117 117 165 254 254 239  50   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]]
```

```
print(X_train[10].shape)
```

```
(28, 28)
```

```
# Displaying the image
plt.imshow(X_train[25])
plt.show()
# printing the corresponding image
print(Y_train[50])
```




```
3
```

Image Lables

```
print(Y_train.shape, Y_test.shape)
```

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

```
# Unique Value in Y_train
print(np.unique(Y_train))

# Unique Value 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]
```

```
# Scaling the values
X_train = X_train/255
X_test = X_test/255
```

```
# Printing the 20th image
print(X_train[20])
```

```
  0.19215686 0.98823529 0.98823529 0.98823529 0.98823529 0.98823529
  0.98823529 0.85098039 0.84705882 0.55294118 0.49411765 0.98823529
  0.98823529 0.98823529 0.60784314 0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.19215686 0.98823529 0.98823529 0.98823529 0.91764706 0.8
  0.34901961 0.         0.         0.         0.19215686 0.98823529
  0.98823529 0.98823529 0.60784314 0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.05490196 0.61960784 0.75294118 0.59215686 0.17647059 0.
  0.         0.         0.         0.         0.19215686 0.98823529
  0.98823529 0.98823529 0.88235294 0.06666667 0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.19215686 0.98823529
  0.98823529 0.98823529 0.98823529 0.09019608 0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.12941176 0.89411765
  0.98823529 0.98823529 0.98823529 0.61568627 0.01568627 0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.21568627
  0.89803922 0.98823529 0.98823529 0.98823529 0.04313725 0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.20784314 0.90980392 0.98823529 0.98823529 0.24705882 0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.35294118 0.80784314 0.51372549 0.04313725 0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         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

```
# Setting up the layers of the Neural Network
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28,28)),
    keras.layers.Dense(50, activation='relu'),
    keras.layers.Dense(10, activation='sigmoid')
])
```

```
# Compiling the Neural Network
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
# Training the Neural Network
model.fit(X_train, Y_train, epochs=10)
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8478 - loss: 0.5343
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9539 - loss: 0.1591
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9671 - loss: 0.1123
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9744 - loss: 0.0884
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9791 - loss: 0.0729
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9824 - loss: 0.0620
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9848 - loss: 0.0532
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9869 - loss: 0.0461
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9887 - loss: 0.0401
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9902 - loss: 0.0349
<keras.src.callbacks.history.History at 0x7cad02c74ac0>
```

Above output shows that Training data accuracy = 99.02%

Accuracy on Test Data:

```
loss, accuraccy = model.evaluate(X_test, Y_test)
print(f"Test Loss: {loss}")
print(f"Test Accuracy: {accuraccy}")
```

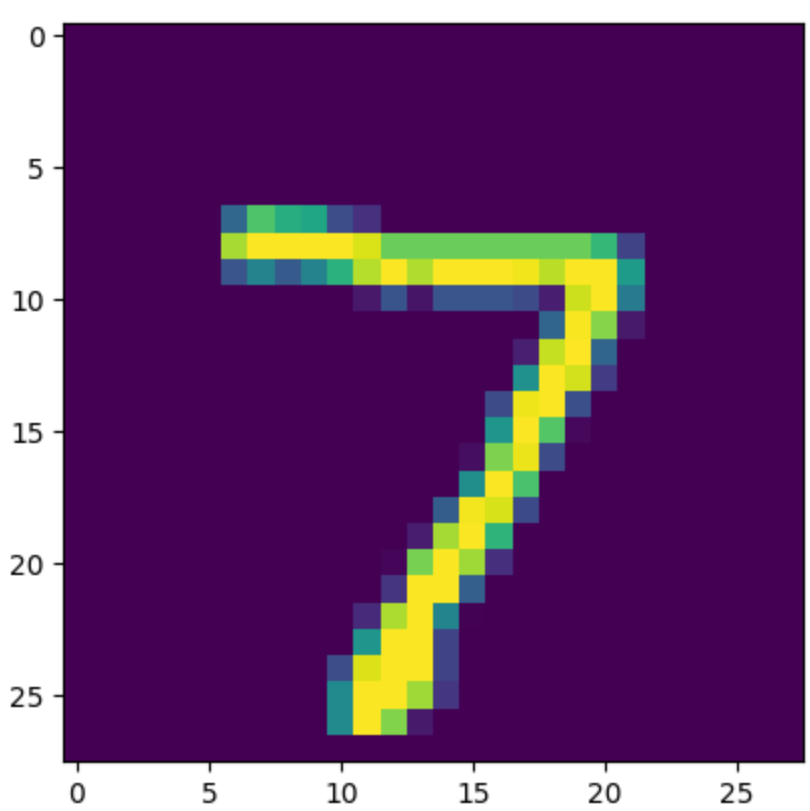
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9692 - loss: 0.1083
Test Loss: 0.09586985409259796
Test Accuracy: 0.9718999862670898
```

Test data accuracy = 97.18%

```
print(X_test.shape)
```

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

```
plt.imshow(X_test[0])
plt.show()
```




```
print(Y_test[0])
```

```
7
```

```
Y_pred = model.predict(X_test)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```

```
print(Y_pred.shape)
```

```
(10000, 10)
```

```
print(Y_pred[0])
```

```
[6.6705686e-03 3.1692503e-05 6.7617349e-02 9.5477194e-01 9.2853525e-10
 4.3258894e-02 3.2479630e-08 9.9999344e-01 2.7314574e-01 6.6098881e-01]
```

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

```
7
```

```
Y_pred_labels = [np.argmax(i) for i in Y_pred]
print(Y_pred_labels)
```

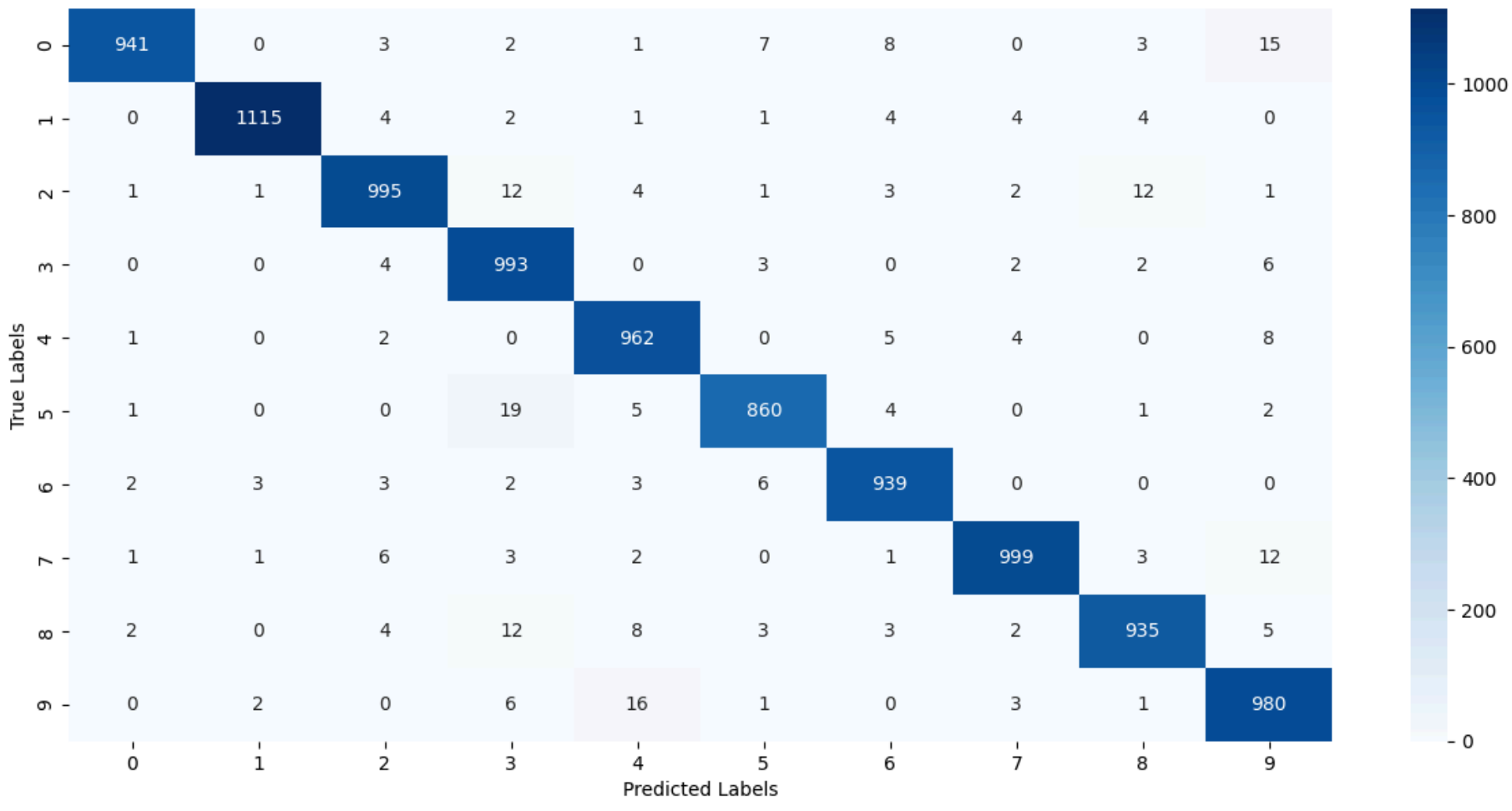
```
[7, 2, 1, 0, 4, 1, 4, 9, 5, 9, 0, 6, 9, 0, 1, 5, 9, 7, 3, 4, 9, 6, 6, 5, 4, 0, 7, 4, 0, 1, 3, 1, 3, 4, 7, 2, 7, 1, 2, 1
```

```
conf_mat = confusion_matrix(Y_test, Y_pred_labels)
print(conf_mat)
```

```
tf.Tensor(
[[ 941    0    3    2    1    7    8    0    3   15]
 [   0 1115    4    2    1    1    4    4    4    0]
 [   1    1  995   12    4    1    3    2   12    1]
 [   0    0    4  993    0    3    0    2    2    6]
 [   1    0    2    0  962    0    5    4    0    8]
 [   1    0    0   19    5  860    4    0    1    2]
 [   2    3    3    2    3    6  939    0    0    0]
 [   1    1    6    3    2    0    1  999    3   12]
 [   2    0    4   12    8    3    3    2  935    5]
 [   0    2    0    6   16    1    0    3    1  980]], shape=(10, 10), dtype=int32)
```

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

```
Text(0.5, 47.7222222222222, 'Predicted Labels')
```



Building Predictive System

```
input_image_path = '/content/MNIST_digit.png'
input_image = cv2.imread(input_image_path)
```

```
type(input_image)
```

```
numpy.ndarray
```

```
cv2_imshow(input_image)
```

```
input_image.shape
```

```
(318, 318, 3)
```

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

```
grayscale.shape
```

```
(318, 318)
```

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

```
input_image_resize.shape
```

```
(28, 28)
```

```
cv2_imshow(input_image_resize)
```

```
input_image_resize = input_image_resize/255
```

```
type(input_image_resize)
```

```
numpy.ndarray
```

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

```
input_prediction = model.predict(image_reshaped)
print(input_prediction)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
[[1.4789014e-09 3.3444320e-10 2.8190700e-06 1.0000000e+00 1.5516093e-12
  2.0472713e-01 1.1959650e-18 1.8005638e-05 9.7836589e-04 6.8385339e-01]]
```

```
input_pred_label = np.argmax(input_prediction)
```

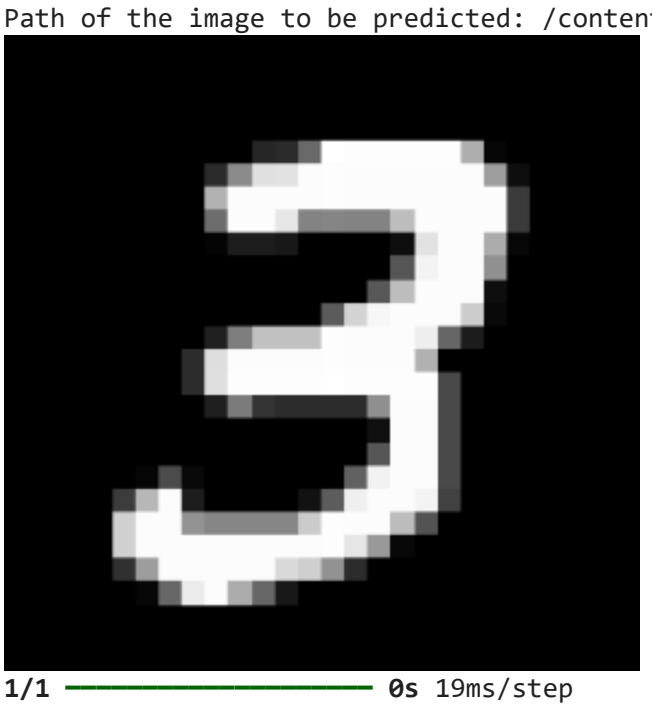
```
print(input_pred_label)
```

```
3
```

Predctive System

```
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
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
The Handwritten Digit is Recognised as:  3
```