

Introduction

In this project, I will implement a neural network to classify handwritten digits using the MNIST dataset. The MNIST dataset comprises 28x28 grayscale images of handwritten digits ranging from 0 to 9. This project is a fundamental exercise in deep learning and image recognition, offering a solid understanding of how to handle image-based datasets. As a widely recognized benchmark problem, it serves as an ideal starting point for beginners in machine learning and neural networks.

Objective

The primary goal of this project is to build a neural network capable of classifying handwritten digits into their respective classes. The system will be trained on labeled image data, enabling it to learn patterns associated with each digit. Ultimately, I aim to develop a predictive system that can correctly identify the digit in any new handwritten image provided as input.

Approach

The project begins with data collection and preprocessing. Since the MNIST dataset is readily available in the Keras library, accessing it is straightforward. Preprocessing will include normalizing pixel values to a range of 0 to 1 and standardizing image dimensions. Labels for the images will be converted into one-hot encoded vectors.

Next, I will construct a neural network comprising fully connected layers. The architecture will include:

1. An **input layer** to accept the flattened image data.
2. **Hidden layers** with activation functions to capture non-linear patterns.
3. A final **output layer** with softmax activation for classifying digits.

The model will be trained using the training dataset, optimized with the Adam optimizer, and evaluated on a separate test dataset. Key metrics, such as accuracy, will assess the model's performance.

Finally, I will develop a predictive system. This system will take a new handwritten digit image as input and predict the corresponding digit. By completing this project, I aim to demonstrate the practical applications of neural networks in image recognition tasks and establish a robust foundation for tackling more complex deep learning challenges in the future.

step by Step Approach

- 1) importing the dataset
- 2) image processing
- 3) splitting the data into train_test split
- 4) building the neural network
- 5) training the neural network
- 6) giving new image to trained data
- 7) prediction is done on new image given to trained neural network.

IMPORTING THE NECESSARY LIBRARIES AND DEPENDENCIES

In [109]:

```
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 tensorflow.math import confusion_matrix
```

IMPORTING THE DATASET WHICH IS IN KERAS

In [110]:

```
from keras.datasets import mnist
```

so here in this dataset data is already split into train test data so we just have to load it (for reference you can visit [keras mnist dataset on google](#))

Where

Training data have 60,000 Images

& Test data has 10,000 Images

Image dimension = 28 x 28

Grayscale Image = 1 channel

In [111]:

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

In [112]:

```
type(X_train)
```

Out[112]:

numpy.ndarray

In [113]:

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

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

In [114]:

```
print(X_train[5])
```

```
[[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 13 25 100
 122  7  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  33 151 208 252 252
 252 146  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  40 152 244 252 253 224 211
 252 232 40  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  15 152 239 252 252 252 216  31  37
 252 252 60  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  96 252 252 252 252 217  29  0  37
 252 252 60  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0 181 252 252 220 167  30  0  0  77
 252 252 60  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0 26 128  58  22  0  0  0  0 100
 252 252 60  0  0  0  0  0  0  0]
```

```
[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 157
252 252 60 0 0 0 0 0 0 0 0]
[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 110 121 122 121 202
252 194 3 0 0 0 0 0 0 0 0]
[ 0 0 0 0 0 0 0 0 0 0 0 10 53 179 253 253 255 253 253
228 35 0 0 0 0 0 0 0 0]
[ 0 0 0 0 0 0 0 0 0 5 54 227 252 243 228 170 242 252 252
231 117 6 0 0 0 0 0 0 0]
[ 0 0 0 0 0 0 0 6 78 252 252 125 59 0 18 208 252 252
252 252 87 7 0 0 0 0 0]
[ 0 0 0 0 0 5 135 252 252 180 16 0 21 203 253 247 129
173 252 252 184 66 49 49 0 0]
[ 0 0 0 0 3 136 252 241 106 17 0 53 200 252 216 65 0
14 72 163 241 252 252 223 0 0]
[ 0 0 0 0 105 252 242 88 18 73 170 244 252 126 29 0 0
0 0 0 89 180 180 37 0 0]
[ 0 0 0 0 231 252 245 205 216 252 252 252 124 3 0 0 0
0 0 0 0 0 0 0 0]
[ 0 0 0 0 207 252 252 252 252 178 116 36 4 0 0 0 0
0 0 0 0 0 0 0]
[ 0 0 0 0 13 93 143 121 23 6 0 0 0 0 0 0 0
0 0 0 0 0 0 0]
[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0]
[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0]
[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0]
[ 0 0 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 [115]:

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

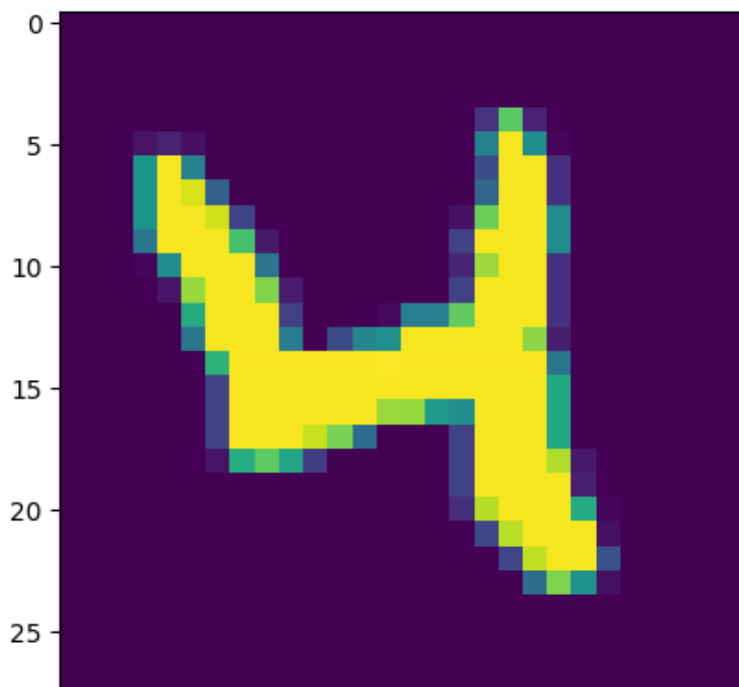
(28, 28)

In [116]:

```
#DISPLAYING THE IMAGE
```

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

```
print(Y_train[20])
```



0 5 10 15 20 25

(60000,) (10000,)

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

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

$$\bar{X}_{\text{test}} = \bar{X}_{\text{test}}/255$$

[0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
[0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
[0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
[0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
[0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.16470588
	0.4627451	0.85882353	0.65098039	0.4627451	0.4627451
	0.	0.	0.	0.	0.02352941
	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
[0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
	0.	0.	0.	0.40392157	0.94901961
	0.99607843	0.99607843	0.99607843	0.99607843	0.25882353
	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
[0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
	0.	0.	0.	0.07058824	0.90980392
	0.99607843	0.99607843	0.99607843	0.99607843	0.93333333
	0.2745098	0.	0.	0.	0.

0.	0.	0.	0.	0.	0.
[0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.40784314
0.95686275	0.99607843	0.87843137	0.99607843	0.99607843	0.99607843
0.55294118	0.	0.	0.	0.	0.
0.	0.	0.	0.]	
[0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.81176471	0.99607843	0.82352941	0.99607843	0.99607843	0.99607843
0.13333333	0.	0.	0.	0.	0.
0.	0.	0.	0.]	
[0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.32941176	0.80784314	0.99607843	0.99607843	0.99607843	0.99607843
0.16078431	0.	0.	0.	0.	0.
0.	0.	0.	0.]	
[0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.09411765	0.81960784	0.99607843	0.99607843	0.99607843
0.67058824	0.	0.	0.	0.	0.
0.	0.	0.	0.]	
[0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.35686275	0.5372549	0.99215686	0.99607843	0.99607843	0.99607843
0.43921569	0.	0.	0.	0.	0.
0.	0.	0.	0.]	
[0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.15686275	0.83921569
0.98039216	0.99607843	0.99607843	0.99607843	0.99607843	0.99607843
0.13333333	0.	0.	0.	0.	0.
0.	0.	0.	0.]	
[0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.31764706	0.96862745
0.99607843	0.99607843	0.99607843	0.99607843	0.99607843	0.99607843
0.57254902	0.	0.	0.	0.	0.
0.	0.	0.	0.]	
[0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.43137255
0.96470588	0.99607843	0.99607843	0.99607843	0.99607843	0.99607843
0.67058824	0.	0.	0.	0.	0.
0.	0.	0.	0.]	
[0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.28627451	0.34901961	0.34901961	0.36470588	0.94117647	0.99607843
0.67058824	0.	0.	0.	0.	0.
0.	0.	0.	0.]	
[0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.00392157	0.50196078	0.99607843
0.85882353	0.12156863	0.	0.	0.	0.
0.	0.	0.	0.]	
[0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.02745098	0.99607843	0.99607843
0.83921569	0.10980392	0.	0.	0.	0.
0.	0.	0.	0.]	
[0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.54117647	0.99607843	0.99607843
0.45490196	0.	0.	0.	0.	0.
0.	0.	0.	0.]	
[0.	0.	0.	0.	0.	0.
0.0745098	0.69411765	0.35294118	0.	0.	0.
0.	0.	0.09803922	0.94117647	0.99607843	0.99607843
0.13333333	0.	0.	0.	0.	0.
0.	0.	0.	0.]	
[0.	0.	0.	0.	0.	0.
0.64313725	0.99607843	0.84313725	0.24705882	0.14117647	0.
0.2	0.34901961	0.80784314	0.99607843	0.99607843	0.54509804
0.03137255	0.	0.	0.	0.	0.
0.	0.	0.	0.]	
[0.	0.	0.	0.	0.	0.

```

0.22352941 0.77254902 0.99607843 0.99607843 0.87058824 0.70588235
0.94509804 0.99607843 0.99607843 0.99215686 0.83529412 0.04313725
0. 0. 0. 0. 0. 0.
0. 0. 0. 0. ]
[0. 0. 0. 0. 0. 0.
0. 0.54901961 0.41176471 0.99607843 0.99607843 0.99607843
0.99607843 0.99607843 0.99607843 0.9254902 0. 0.
0. 0. 0. 0. 0. 0.
0. 0. 0. 0. ]
[0. 0. 0. 0. 0. 0.
0. 0. 0.02745098 0.45882353 0.45882353 0.64705882
0.99607843 0.99607843 0.9372549 0.19607843 0. 0.
0. 0. 0. 0. 0. 0.
0. 0. 0. 0. ]
[0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0.
0. 0. 0. 0. ]
[0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0.
0. 0. 0. 0. ]
[0. 0. 0. 0. 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 NEURAL NETWORK

In [121]:

```

model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28,28)),
    keras.layers.Dense(50, activation='relu'),
    keras.layers.Dense(50, activation='relu'),
    keras.layers.Dense(50, activation='relu'),
    keras.layers.Dense(10, activation='sigmoid')
])

```

/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

super().__init__(**kwargs)

NOW COMPIILING THE NN

In [122]:

```

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

```



NOW TRAINING THE NN

In [123]:

```

model.fit(X_train, Y_train, epochs=20)

```

Epoch 1/20
1875/1875  5s 2ms/step - accuracy: 0.8425 - loss: 0.5183
Epoch 2/20
1875/1875  6s 3ms/step - accuracy: 0.9560 - loss: 0.1468

```

Epoch 3/20
1875/1875 ————— 9s 2ms/step - accuracy: 0.9672 - loss: 0.1062
Epoch 4/20
1875/1875 ————— 5s 3ms/step - accuracy: 0.9754 - loss: 0.0824
Epoch 5/20
1875/1875 ————— 4s 2ms/step - accuracy: 0.9803 - loss: 0.0652
Epoch 6/20
1875/1875 ————— 6s 2ms/step - accuracy: 0.9829 - loss: 0.0560
Epoch 7/20
1875/1875 ————— 6s 3ms/step - accuracy: 0.9844 - loss: 0.0477
Epoch 8/20
1875/1875 ————— 4s 2ms/step - accuracy: 0.9869 - loss: 0.0415
Epoch 9/20
1875/1875 ————— 4s 2ms/step - accuracy: 0.9870 - loss: 0.0398
Epoch 10/20
1875/1875 ————— 6s 3ms/step - accuracy: 0.9876 - loss: 0.0360
Epoch 11/20
1875/1875 ————— 9s 2ms/step - accuracy: 0.9896 - loss: 0.0289
Epoch 12/20
1875/1875 ————— 6s 3ms/step - accuracy: 0.9893 - loss: 0.0322
Epoch 13/20
1875/1875 ————— 10s 2ms/step - accuracy: 0.9900 - loss: 0.0285
Epoch 14/20
1875/1875 ————— 5s 3ms/step - accuracy: 0.9907 - loss: 0.0265
Epoch 15/20
1875/1875 ————— 4s 2ms/step - accuracy: 0.9925 - loss: 0.0209
Epoch 16/20
1875/1875 ————— 4s 2ms/step - accuracy: 0.9921 - loss: 0.0248
Epoch 17/20
1875/1875 ————— 5s 3ms/step - accuracy: 0.9925 - loss: 0.0217
Epoch 18/20
1875/1875 ————— 4s 2ms/step - accuracy: 0.9925 - loss: 0.0218
Epoch 19/20
1875/1875 ————— 6s 2ms/step - accuracy: 0.9938 - loss: 0.0179
Epoch 20/20
1875/1875 ————— 5s 2ms/step - accuracy: 0.9940 - loss: 0.0175

```

Out[123]:

```
<keras.src.callbacks.history.History at 0x7af1d39b4f10>
```

accuracy: 99.40

NOR FOR ACCURACY ON TEST DATA

In [124]:

```

loss, accuracy = model.evaluate(X_test, Y_test)
print (accuracy)

```

```

313/313 ————— 1s 1ms/step - accuracy: 0.9663 - loss: 0.1840
0.9700999855995178

```

Accuracy :97.00

In [125]:

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

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

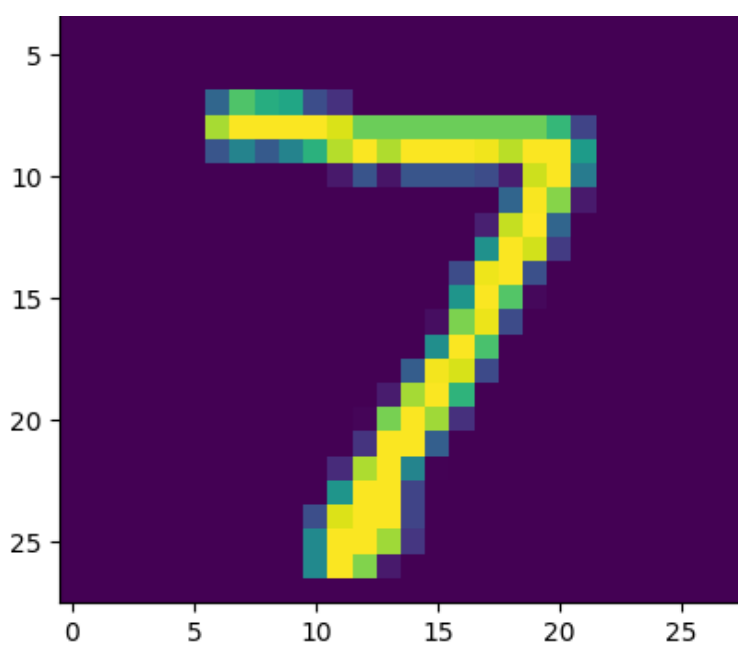
In [126]:

```

#NOW FOR FIRST DATA POINT OF X_test
plt.imshow(X_test[0])
plt.show()

```





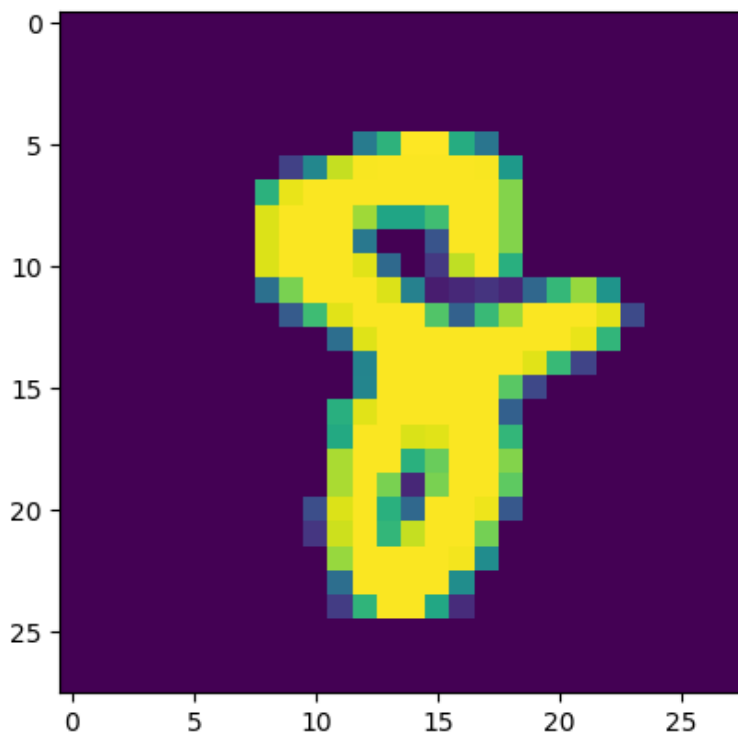
In [127]:

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

7

In [128]:

```
#NOW FOR ANY RANDOM NUMBER OF DATA POINT OF X_test
plt.imshow(X_test[277])
plt.show()
```



In [129]:

```
print(Y_test[277])
```

8

In [130]:

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

313/313 ————— 1s 2ms/step

$(10000, 10)$

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

[3.8084084e-12 7.2866434e-01 8.3955041e-05 3.5990405e-01 2.9628985e-08
2.1199497e-05 5.7384824e-25 1.0000000e+00 8.0411115e-07 8.5248584e-01]

In [133]:

```
#CONVERTING THE PREDICTION PROBABILITIES TO CLASS LABEL FOR FIRST DATA POINT.
label_for_first_test_image = np.argmax(Y_pred[0])
print(label_for_first_test_image)
```

7

In [134]:

```
#NOW FOR ALL DATA POINTS
Y_pred_labels = [np.argmax(i) for i in Y_pred]
print(Y_pred_labels)
```

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

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

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

WHERE

Y_test = True labels

Y_pred_labels = Predicted Labels

NOW CREATIN CONFUSION MATRIX

In [135]:

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

In [136]:

```
print(conf_mat)
```

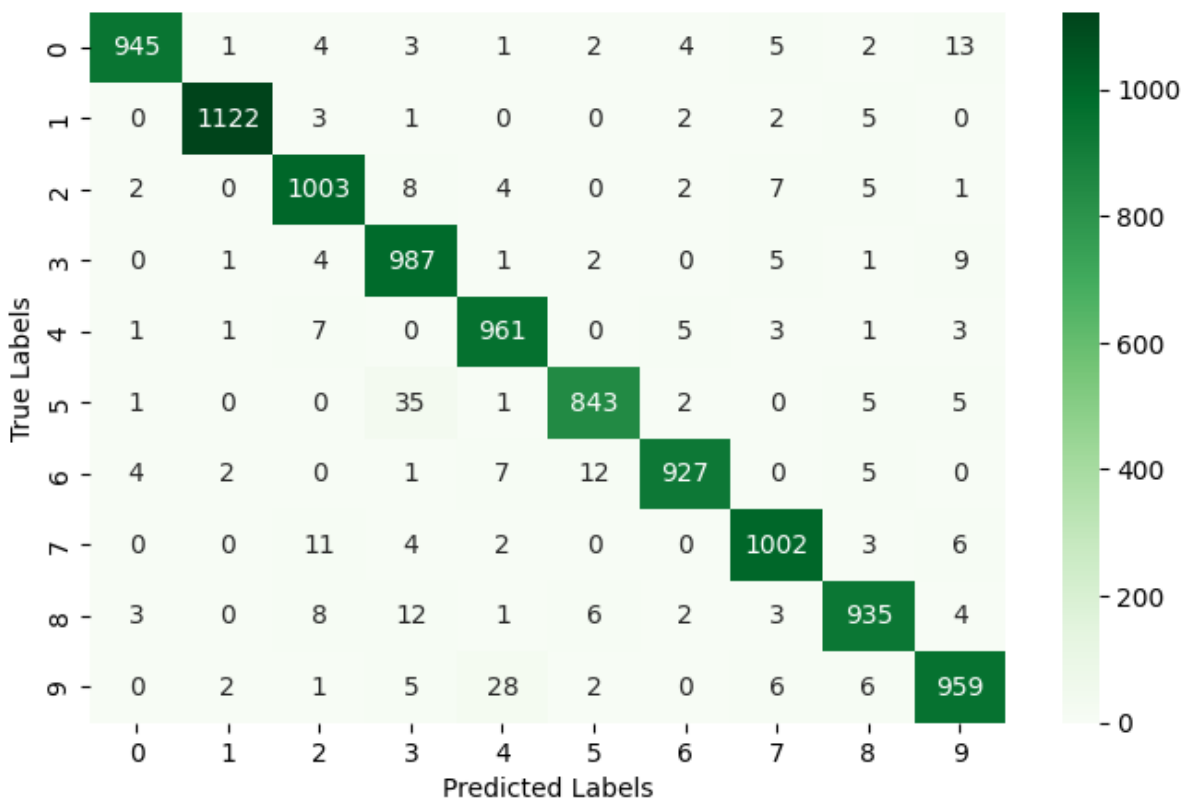
```
tf.Tensor(  
[[ 945     1     4     3     1     2     4     5     2    13]  
 [    0 1122     3     1     0     0     2     2     5     0]  
 [     2     0 1003     8     4     0     2     7     5     1]  
 [    0     1     4  987     1     2     0     5     1     9]  
 [     1     1     7     0  961     0     5     3     1     3]  
 [     1     0     0    35     1  843     2     0     5     5]  
 [     4     2     0     1     7    12   927     0     5     0]  
 [     0     0    11     4     2     0     0  1002     3     6]  
 [     3     0     8    12     1     6     2     3   935     4]  
 [     0     2     1     5    28     2     0     6     6   959]], shape=(10, 10), dtype=int32)
```

In [137]:

```
plt.figure(figsize=(8,5))  
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Greens')  
plt.ylabel('True Labels')  
plt.xlabel('Predicted Labels')
```

Out[137]:

Text(0.5, 25.72222222222214, 'Predicted Labels')



NOW BUILDING A PREDICTIVE SYSTEM

In [138]:

```
input_image_path = '/content/MNIST_digit.png'

input_image = cv2.imread(input_image_path)
```

In [139]:

```
type(input_image)
```

Out[139]:

numpy.ndarray

In [140]:

```
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 [141]:

```
import cv2
```

```

import numpy as np
import requests
from google.colab.patches import cv2_imshow

# Load the image from the URL
url = 'https://upload.wikimedia.org/wikipedia/commons/2/27/MnistExamples.png'
response = requests.get(url, stream=True).raw
input_image = np.asarray(bytearray(response.read()), dtype="uint8")
input_image = cv2.imdecode(input_image, cv2.IMREAD_GRAYSCALE)

# Check image dimensions
print(f"Image dimensions: {input_image.shape}")

# Crop a single digit (adjust coordinates based on the specific image layout)
# Example: Cropping the first digit (coordinates may vary)
single_digit = input_image[5:50, 5:50] # Replace with the coordinates of the desired digit

# Display the single digit
cv2_imshow(single_digit)

```

Image dimensions: (361, 594)



In [142]:

```
input_image.shape
```

Out[142]:

```
(361, 594)
```

In [143]:

```
grayscale = input_image # The image is already in grayscale
```

In [144]:

```
grayscale.shape
```

Out[144]:

```
(361, 594)
```

In [145]:

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

In [146]:

```
input_image_resize.shape
```

Out[146]:

```
(28, 28)
```

In [147]:

```
cv2_imshow(input_image_resize)
```



In [148]:

```
input_image_resize = input_image_resize/255
```

In [149]:

```
type(input_image_resize)
```


Out[149]:

numpy.ndarray

In [150]:

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

In [151]:

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

```
1/1 ————— 0s 19ms/step
[[2.6880250e-13 2.3996851e-08 3.2410774e-01 4.3018866e-02 2.2588846e-26
  9.9999845e-01 2.0655115e-19 1.5868343e-10 6.1476588e-01 9.9055147e-01]]
```

In [152]:

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

In [153]:

```
print(input_pred_label)
```

5

Predictive System

In [154]:

```
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_resized = np.reshape(input_image_resize, [1,28,28])
input_prediction = model.predict(image_resized)
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/digit_sample.png

A small, square image showing a handwritten digit '7' in black ink on a white background.

```
1/1 ————— 0s 20ms/step
The Handwritten Digit is recognised as  7
```