Code: (Referred from Geeks for Geeks Website and modified accordingly)

```
#Importing all the necessary libraries and modules
import numpy as np
import tensorflow as tf
import keras
from keras.datasets import mnist
from keras.models import Model
from keras.layers import Dense, Input
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten
from keras import backend as k
import cv2
from google.colab import drive
import os

drive.mount('/content/drive')

#Loading the actual MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()


img_rows, img_cols=28, 28
#Converting all the inputs into one dimensional array
if k.image_data_format() == 'channels_first':
   x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
   x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
   inpx = (1, img_rows, img_cols)

else:
   x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
   x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
   inpx = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255 #normalizing the data
x_test /= 255

#Since output is not binary and will be from 0-9
y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)

#Defining the internal Layers of the network
inpx = Input(shape=inpx)
layer1 = Conv2D(32, kernel_size=(3, 3), activation='relu')(inpx)
layer2 = Conv2D(64, (3, 3), activation='relu')(layer1)
layer3 = MaxPooling2D(pool_size=(3, 3))(layer2)
```

```
layer4 = Dropout(0.5)(layer3)
layer5 = Flatten()(layer4)
layer6 = Dense(250, activation='sigmoid')(layer5)
layer7 = Dense(10, activation='softmax')(layer6)

#Parameters requrired to train the model, using SGD and a learning rate
 of 0.05
model = Model([inpx], layer7)
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.05),
              loss=keras.losses.categorical_crossentropy,
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=10, batch_size=50)
model.save("Xmodel")

#Evaluating the trained model
score = model.evaluate(x_test, y_test, verbose=0)
print('loss=', score[0])
print('accuracy=', score[1])

#pre-processing our own data
def img(data_folder,dim):
  im_data = []
  for filepath in os.listdir(data_folder):
    # print(filepath)
    x = cv2.imread(data_folder+'{0}'.format(filepath) , cv2.IMREAD_GRAY
SCALE)
    im_data.append(cv2.resize(x,dim))
  return np.array(im_data)

path = '/content/drive/MyDrive/Colab Notebooks/My_Integers/'
dim =(28,28)
X =img(path,dim)
X1=(X.reshape(X.shape[0], img_rows, img_cols, 1)/255).astype('float32')
ypred = model.predict(X1)

import matplotlib.pyplot as plt

for i in range (len(ypred)):
  print('predicted handwritten digit is: '+str(np.argmax(ypred[i])))
  plt.imshow(X[i])
  plt.show()
```

Outputs:

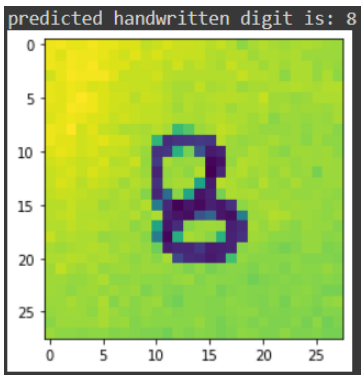
1. Training:

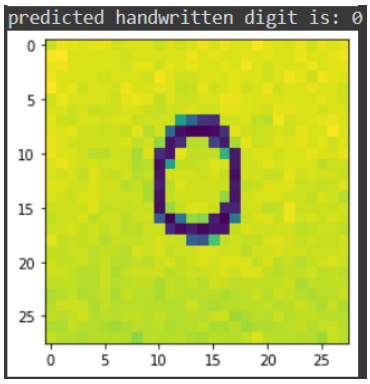
```
Epoch 1/10
1200/1200 [==============================] - 5s 4ms/step - loss: 0.0905 - accuracy: 0.9732
Epoch 2/10
1200/1200 [==============================] - 5s 4ms/step - loss: 0.0785 - accuracy: 0.9771
Epoch 3/10
1200/1200 [==============================] - 5s 4ms/step - loss: 0.0699 - accuracy: 0.9789
Epoch 4/10
1200/1200 [==============================] - 5s 4ms/step - loss: 0.0642 - accuracy: 0.9806
Epoch 5/10
1200/1200 [==============================] - 5s 4ms/step - loss: 0.0570 - accuracy: 0.9831
Epoch 6/10
1200/1200 [==============================] - 5s 4ms/step - loss: 0.0529 - accuracy: 0.9841
Epoch 7/10
1200/1200 [==============================] - 5s 4ms/step - loss: 0.0494 - accuracy: 0.9850
Epoch 8/10
1200/1200 [==============================] - 5s 4ms/step - loss: 0.0446 - accuracy: 0.9872
Epoch 9/10
1200/1200 [==============================] - 5s 4ms/step - loss: 0.0416 - accuracy: 0.9875
Epoch 10/10
1200/1200 [==============================] - 5s 4ms/step - loss: 0.0395 - accuracy: 0.9880
```

2. Accuracy:

```
loss= 0.030958453193306923
accuracy= 0.9894999861717224
```

3. Predicted Outputs on Handwritten Digits:





4. Actual digits: