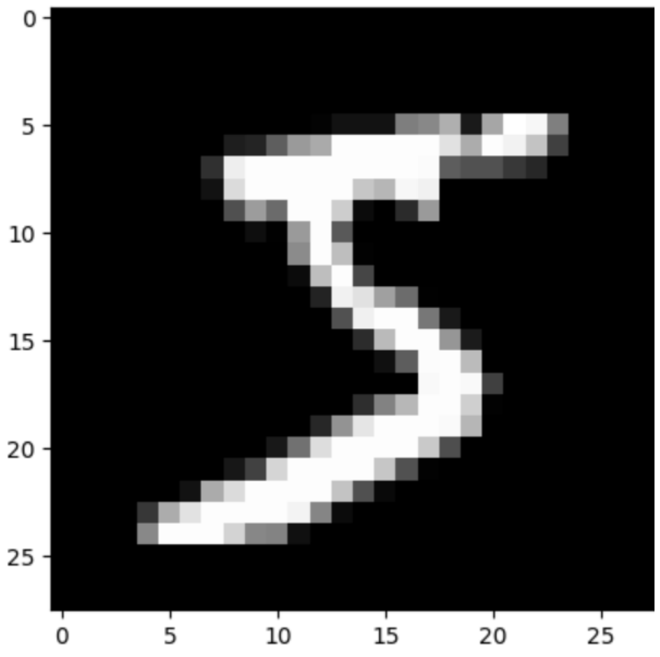
In [1]:
```
import cv2
import numpy as np
from keras.datasets import mnist
from keras.layers import Dense, Flatten, MaxPooling2D, Dropout
from keras.layers.convolutional import Conv2D
from keras.models import Sequential
from keras.utils import to_categorical
import matplotlib.pyplot as plt
```

In [2]:
```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

In [3]:
```
plt.imshow(X_train[0], cmap="gray")
plt.show()
print (y_train[0])
```



In [4]:
```
print ("Shape of X_train: {}".format(X_train.shape))
print ("Shape of y_train: {}".format(y_train.shape))
print ("Shape of X_test: {}".format(X_test.shape))
print ("Shape of y_test: {}".format(y_test.shape))
```

```
Shape of X_train: (60000, 28, 28)
Shape of y_train: (60000,)
Shape of X_test: (10000, 28, 28)
Shape of y_test: (10000,)
```

In [5]:
```
# Reshaping so as to convert images for our model
X_train = X_train.reshape(60000, 28, 28, 1)
X_test = X_test.reshape(10000, 28, 28, 1)
```

In [6]:
```
print ("Shape of X_train: {}".format(X_train.shape))
print ("Shape of y_train: {}".format(y_train.shape))
print ("Shape of X_test: {}".format(X_test.shape))
print ("Shape of y_test: {}".format(y_test.shape))
```

```
Shape of X_train: (60000, 28, 28, 1)
Shape of y_train: (60000,)
Shape of X_test: (10000, 28, 28, 1)
Shape of y_test: (10000,)
```

In [7]:
```
#one hot encoding
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

```
In [8]:
model = Sequential()

## Declare the layers
layer_1 = Conv2D(64, kernel_size=3, activation='relu', input_shape=(28, 28, 1))
layer_2 = MaxPooling2D(pool_size=2)
layer_3 = Conv2D(32, kernel_size=3, activation='relu')
layer_4 = MaxPooling2D(pool_size=2)
layer_5 = Dropout(0.5)
layer_6 = Flatten()
layer_7 = Dense(128, activation="relu")
layer_8 = Dropout(0.5)
layer_9 = Dense(10, activation='softmax')

## Add the layers to the model
model.add(layer_1)
model.add(layer_2)
model.add(layer_3)
model.add(layer_4)
model.add(layer_5)
model.add(layer_6)
model.add(layer_7)
model.add(layer_8)
model.add(layer_9)
```

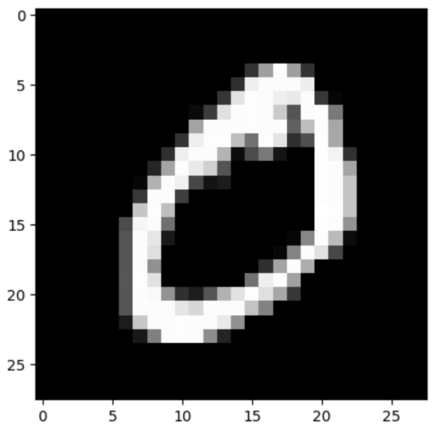
```
In [9]:
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
In [10]:
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=3)
```

```
Epoch 1/3
1875/1875 [==============================] - 45s 22ms/step - loss: 0.8765 - accuracy: 0.7776 - val_loss: 0.1128 - val_accuracy: 0.9665
Epoch 2/3
1875/1875 [==============================] - 41s 22ms/step - loss: 0.2648 - accuracy: 0.9222 - val_loss: 0.0837 - val_accuracy: 0.9730
Epoch 3/3
1875/1875 [==============================] - 41s 22ms/step - loss: 0.2032 - accuracy: 0.9403 - val_loss: 0.0576 - val_accuracy: 0.9805
Out[10]:
```

```
In [11]:
example = X_train[1]
prediction = model.predict(example.reshape(1, 28, 28, 1))
print ("Prediction (Softmax) from the neural network:\n\n {}".format(prediction))
hard_maxed_prediction = np.zeros(prediction.shape)
hard_maxed_prediction[0][np.argmax(prediction)] = 1
print ("\n\nHard-maxed form of the prediction: \n\n {}".format(hard_maxed_prediction))

print ("\n\n--------- Prediction --------- \n\n")
plt.imshow(example.reshape(28, 28), cmap="gray")
plt.show()
print("\n\nFinal Output: {}".format(np.argmax(prediction)))
```

```
1/1 [==============================] - 0s 207ms/step
Prediction (Softmax) from the neural network:

 [[9.9999928e-01 4.9414063e-11 8.8367727e-08 1.0839282e-09 3.7435699e-10
  1.0014264e-09 1.3681128e-07 4.1335552e-10 3.7364279e-07 1.4932097e-07]]


Hard-maxed form of the prediction:

 [[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]


--------- Prediction ---------
```



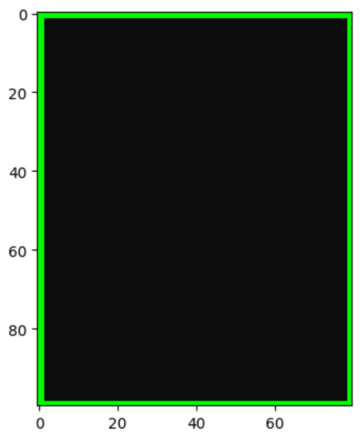
```
Final Output: 0
```

```
In [12]:
metrices=model.evaluate(X_test,y_test,verbose=0)
print("Metrices(test loss and Test Accuracy):")
print(metrices)
```

```
Metrices(test loss and Test Accuracy):
[0.057593006640672684, 0.9804999828338623]
```

In [13]:
```
image = cv2.imread('test_image.jpg')
image = np.full((100,80,3), 12, dtype = np.uint8)
grey = cv2.cvtColor(image.copy(), cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(grey.copy(), 75, 255, cv2.THRESH_BINARY_INV)
contours,hierarchy = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
preprocessed_digits = []

for c in contours:
    x,y,w,h = cv2.boundingRect(c)

    # Creating a rectangle around the digit in the original image (for displaying the digits fetched via contours)
    cv2.rectangle(image, (x,y), (x+w, y+h), color=(0, 255, 0), thickness=2)

    # Cropping out the digit from the image corresponding to the current contours in the for loop
    digit = thresh[y:y+h, x:x+w]

    # Resizing that digit to (18, 18)
    resized_digit = cv2.resize(digit, (18,18))

    # Padding the digit with 5 pixels of black color (zeros) in each side to finally produce the image of (28, 28)
    padded_digit = np.pad(resized_digit, ((5,5),(5,5)), "constant", constant_values=0)

    # Adding the preprocessed digit to the list of preprocessed digits
    preprocessed_digits.append(padded_digit)

print("\n\n\n----------------Contoured Image--------------------")
import os, types
import pandas as pd

def __iter__(self): return 0

print=("\n\n\n----------------Contoured Image--------------------")
plt.imshow(image, cmap="gray")
plt.show()

inp = np.array(preprocessed_digits)
```

```
----------------Contoured Image--------------------
```



In [14]:
```
model.save("models/mnistCNN.h5")
```