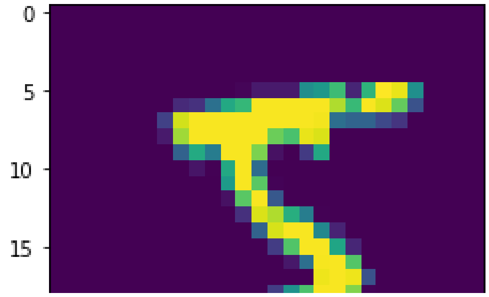
```
<matplotlib.image.AxesImage at 0x7f9465e88510>
```



Reshaping the data

```
X_train =X_train.reshape(60000,28,28,1).astype('float32')
X_test =X_test.reshape(10000,28,28,1).astype('float32')
```

One Hot Encoding

```
number_of_classes = 10
y_train = np_utils.to_categorical(y_train,number_of_classes)
y_test=np_utils.to_categorical(y_test,number_of_classes)
y_train[0]
```

```
array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.], dtype=float32)
```

MODEL BUILDING

ADD CNN LAYERS

```
# CREATING THE MODEL
model = Sequential()
#adding model layer
model.add(Conv2D(64,(3,3),input_shape=(28,28,1),activation='relu'))
model.add(Conv2D(32,(3,3),activation='relu'))
model.add(Flatten())
model.add(Dense(number_of_classes,activation='softmax'))
```

Compiling the model

```
model.compile(loss='categorical_crossentropy',optimizer='Adam',metrics=['accuracy'])
```

Train the model

```
model.fit(X_train,y_train,epochs= 2,validation_data=(X_test,y_test),batch_size=32)
```

```
Epoch 1/2
1875/1875 [==============================] - 183s 97ms/step - loss: 0.0511 - accuracy: 0
Epoch 2/2
1875/1875 [==============================] - 181s 97ms/step - loss: 0.0352 - accuracy: 0
<keras.callbacks.History at 0x7f945dd91190>
```

OBSERVING THE METRICS

```
metrics= model.evaluate(X_test,y_test,verbose=0)
print("Metrics(Test loss & Test Accuracy): ")
print(metrics)
```

```
Metrics(Test loss & Test Accuracy): 
[0.07943267375230789, 0.9800999760627747]
```

PREDICTING THE OUTPUT

```
prediction = model.predict(X_test[:4])
print(prediction)
```

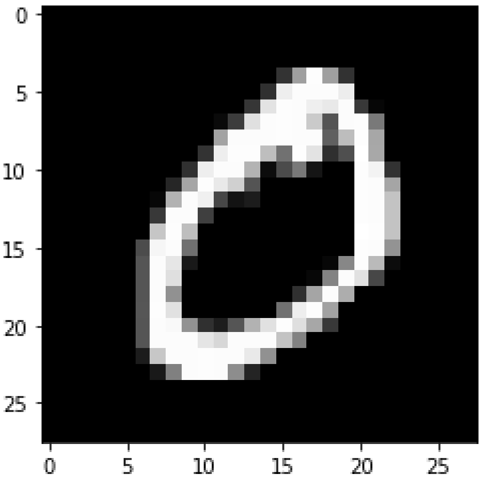
```
1/1 [==============================] - 0s 105ms/step
[[1.92560523e-09 4.01850529e-17 5.50588419e-09 2.96046232e-09
  8.29567738e-14 2.69424494e-14 2.47960087e-20 1.00000000e+00
  9.45855200e-12 3.30721561e-10]
 [2.61439848e-09 1.07959084e-11 1.00000000e+00 1.53540125e-10
  2.90862453e-15 1.13654680e-15 5.82739135e-11 7.71922828e-19
  2.21205207e-11 8.07522963e-17]
 [5.78494564e-06 9.94477868e-01 6.71379121e-06 2.65865587e-08
  1.06376270e-03 4.35107807e-03 6.03496301e-06 6.35672563e-08
  8.86625785e-05 3.47364748e-10]
 [9.99999762e-01 1.14116395e-16 1.04427467e-09 2.66038293e-14
  3.39294669e-11 3.35020171e-11 1.04318076e-07 1.11794697e-11
  4.95246011e-10 1.00263563e-07]]
```

```
import numpy as np
print(np.argmax(prediction,axis=1))
print(y_test[:4])
```

```
[7 2 1 0]
[[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

```
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 21ms/step
Prediction (Softmax) from the neural network:

 [[9.9999511e-01 9.9007680e-16 2.4157876e-08 8.5403615e-14 4.9131352e-11
   6.9746596e-13 4.8227434e-06 5.4842005e-14 3.2907513e-10 3.0204330e-09]]


Hard-maxed form of the prediction:

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


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



```
Final Output: 0
```

OBSERVING THE METRICS

```
metrics= model.evaluate(X_test,y_test,verbose=0)
print("Metrics(Test loss & Test Accuracy): ")
print(metrics)
```

```
Metrics(Test loss & Test Accuracy):
[0.07943267375230789, 0.9800999760627747]
```

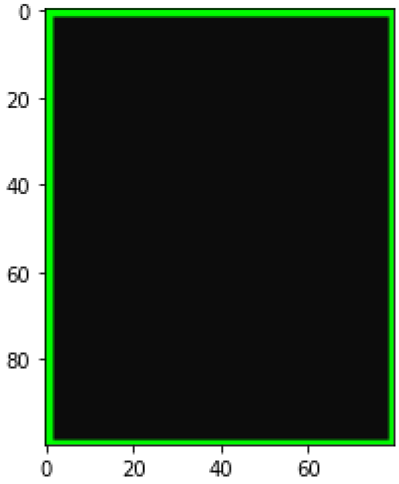
```
prediction = model.predict(X_test[:4])
print(prediction)
import numpy as np
print(np.argmax(prediction,axis=1))
print(y_test[:4])
```

```
1/1 [==============================] - 0s 26ms/step
[[1.92560523e-09 4.01850529e-17 5.50588419e-09 2.96046232e-09
  8.29567738e-14 2.69424494e-14 2.47960087e-20 1.00000000e+00
  9.45855200e-12 3.30721561e-10]
 [2.61439848e-09 1.07959084e-11 1.00000000e+00 1.53540125e-10
  2.90862453e-15 1.13654680e-15 5.82739135e-11 7.71922828e-19
  2.21205207e-11 8.07522963e-17]
 [5.78494564e-06 9.94477868e-01 6.71379121e-06 2.65865587e-08
  1.06376270e-03 4.35107807e-03 6.03496301e-06 6.35672563e-08
  8.86625785e-05 3.47364748e-10]
 [9.99999762e-01 1.14116395e-16 1.04427467e-09 2.66038293e-14
  3.39294669e-11 3.35020171e-11 1.04318076e-07 1.11794697e-11
  4.95246011e-10 1.00263563e-07]]
[7 2 1 0]
[[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

SAVE THE MODEL

```
import cv2
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_SIMP
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
    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
    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
    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--------------------
```



SAVE THE MODEL

```
model.save('model.h5')
```