In [1]:
```
import tensorflow as tf
```

Loading - "MNIST Data Set"

Contains Training samples = 60,000 and Testing samples = 10,000

from tensorflow

In [2]:
```
mnist = tf.keras.datasets.mnist #handwritten characters
```

Divide into train and test datasets

In [3]:
```
#unpacking the dataset into train and test datasets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

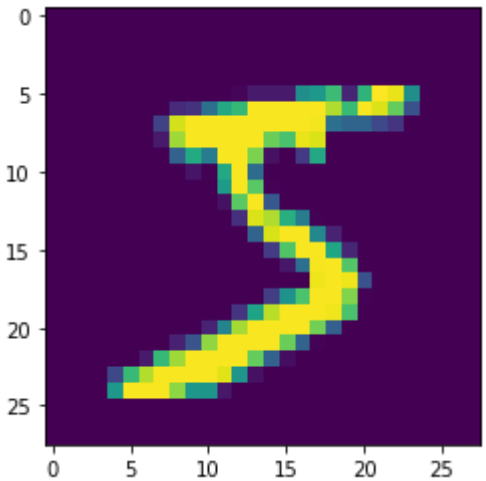
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 7s 1us/step
11501568/11490434 [==============================] - 7s 1us/step
```

In [4]:
```
x_train.shape
```

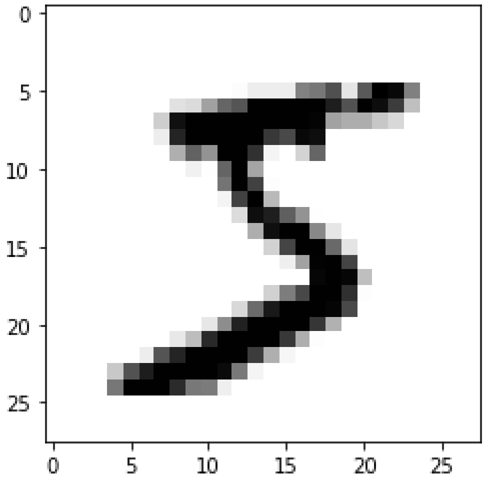
Out[4]:
```
(60000, 28, 28)
```

In [6]:
```
import matplotlib.pyplot as plt
```

In [7]:
```
#checking how data looks like
plt.imshow(x_train[0])
plt.show() #executing the graph
#we donno whether it's color image or binary images
#so inorder to plot it, change the config
plt.imshow(x_train[0], cmap = plt.cm.binary)
```



Out[7]:
```
<matplotlib.image.AxesImage at 0x1b8437be520>
```



checking the values of each pixel

Before Normalization

In [8]:

```
print(x_train[0]) #before normalization
```

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136
  175  26 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253
  225 172 253 242 195  64   0   0   0   0]
 [  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251
   93  82  82  56  39   0   0   0   0   0]
 [  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119
   25   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253
  150  27   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252
  253 187   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249
  253 249  64   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
  253 207   2   0   0   0   0   0   0   0]
```

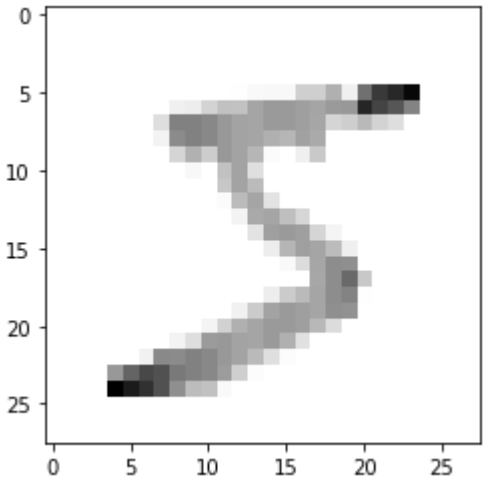
```
[  0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253
 250 182   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201
  78   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]]
```

Images are in Gray level (1 pixel == 0 to 255), not (RGB)

Normalizing the data | Pre-Processing Step

In [11]:
```
x_train = tf.keras.utils.normalize (x_train, axis=1)
x_test = tf.keras.utils.normalize(x_test, axis=1)
plt.imshow(x_train[0], cmap = plt.cm.binary)
```

Out[11]:
```
<matplotlib.image.AxesImage at 0x1b8437462b0>
```



After Normalization

In [12]:
```
print(x_train[0]) #values are normalized (0 to 1)
```

```
[[0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
```

```
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.00393124 0.02332955 0.02620568 0.02625207 0.17420356 0.17566281
 0.28629534 0.05664824 0.51877786 0.71632322 0.77892406 0.89301644
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.05780486 0.06524513 0.16128198 0.22713296
 0.22277047 0.32790981 0.36833534 0.3689874  0.34978968 0.32678448
 0.368094   0.3747499  0.79066747 0.67980478 0.61494005 0.45002403
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.12250613 0.45858525 0.45852825 0.43408872 0.37314701
 0.33153488 0.32790981 0.36833534 0.3689874  0.34978968 0.32420121
 0.15214552 0.17865984 0.25626376 0.1573102  0.12298801 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.04500225 0.4219755  0.45852825 0.43408872 0.37314701
 0.33153488 0.32790981 0.28826244 0.26543758 0.34149427 0.31128482
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.1541463  0.28272888 0.18358693 0.37314701
 0.33153488 0.26569767 0.01601458 0.         0.05945042 0.19891229
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.0253731  0.00171577 0.22713296
 0.33153488 0.11664776 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.20500962
 0.33153488 0.24625638 0.00291174 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.01622378
 0.24897876 0.32790981 0.10191096 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.04586451 0.31235677 0.32757096 0.23335172 0.14931733 0.00129164
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.10498298 0.34940902 0.3689874  0.34978968 0.15370495
 0.04089933 0.         0.         0.         0.         0.
```

```
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.06551419 0.27127137 0.34978968 0.32678448
 0.245396   0.05882702 0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.02333517 0.12857881 0.32549285
 0.41390126 0.40743158 0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.32161793
 0.41390126 0.54251585 0.20001074 0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.06697006 0.18959827 0.25300993 0.32678448
 0.41390126 0.45100715 0.00625034 0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.05110617 0.19182076 0.33339444 0.3689874  0.34978968 0.32678448
 0.40899334 0.39653769 0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.04117838 0.16813739
 0.28960162 0.32790981 0.36833534 0.3689874  0.34978968 0.25961929
 0.12760592 0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.04431706 0.11961607 0.36545809 0.37314701
 0.33153488 0.32790981 0.36833534 0.28877275 0.111988   0.00258328
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.05298497 0.42752138 0.4219755  0.45852825 0.43408872 0.37314701
 0.33153488 0.25273681 0.11646967 0.01312603 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.37491383 0.56222061
 0.66525569 0.63253163 0.48748768 0.45852825 0.43408872 0.359873
 0.17428513 0.01425695 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.92705966 0.82698729
 0.74473314 0.63253163 0.4084877  0.24466922 0.22648107 0.02359823
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         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 [13]:
```
print(y_train[0]) #labels
```

```
5
```

Resizing image to make it suitable for applying Convolution operation

In [15]:
```
import numpy as np
IMG_SIZE = 28
x_trainr = np.array(x_train).reshape(-1, IMG_SIZE, IMG_SIZE,1) ### increasing 1 dime
x_testr = np.array(x_test).reshape(-1, IMG_SIZE, IMG_SIZE,1) ### increasing 1 dimens
print("Training Samples dimension", x_trainr.shape)
print("Testing Samples dimensions",x_testr.shape)
```

```
Training Samples dimension (60000, 28, 28, 1)
Testing Samples dimensions (10000, 28, 28, 1)
```

Creating a Deep Neural Network

Training on 60,000 samples of MNIST handwritten dataset

In [17]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, Max
```

In [20]:
```
### creating a neural network
model = Sequential()

#### First Layer
model.add(Conv2D(64, (3,3), input_shape = x_trainr.shape[1:])) ### mention input lay
model.add(Activation('relu')) ## Activation function, to make it non-linear
model.add(MaxPooling2D(pool_size = (2,2))) ## Maxpooling, Single max value of 2x2 ma

#### Second Layer
model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))

#### Third Layer
model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))

#### Fully Connected Layer #1
model.add(Flatten()) ### before using fully connected, it need to be flatten (2D to
model.add(Dense(64)) # neural network
model.add(Activation('relu'))

#### Fully Connected Layer #2
model.add(Dense(32))
model.add(Activation('relu'))

#### Last Fully Connected Layer, output must be equal to number of classes, 10(0-9)
model.add(Dense(10)) ## last dense layer must be equal to 10
model.add(Activation('softmax')) ## changed to sotmax (class probabilites)
```

In [22]:
```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 26, 26, 64)        640

 activation_2 (Activation)   (None, 26, 26, 64)        0

 max_pooling2d_1 (MaxPooling  (None, 13, 13, 64)       0
 2D)

 conv2d_3 (Conv2D)           (None, 11, 11, 64)        36928

 activation_3 (Activation)   (None, 11, 11, 64)        0

 max_pooling2d_2 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 conv2d_4 (Conv2D)           (None, 3, 3, 64)          36928

 activation_4 (Activation)   (None, 3, 3, 64)          0

 max_pooling2d_3 (MaxPooling  (None, 1, 1, 64)         0
 2D)

 flatten (Flatten)           (None, 64)                0

 dense (Dense)               (None, 64)                4160

 activation_5 (Activation)   (None, 64)                0

 dense_1 (Dense)             (None, 32)                2080

 activation_6 (Activation)   (None, 32)                0

 dense_2 (Dense)             (None, 10)                330

 activation_7 (Activation)   (None, 10)                0

=================================================================
Total params: 81,066
Trainable params: 81,066
Non-trainable params: 0
_________________________________________________________________
```

In [23]:
```
print("Total Training Samples = ", len(x_trainr))
```

```
Total Training Samples =  60000
```

In [24]:
```
model.compile(loss = "sparse_categorical_crossentropy", optimizer = "adam", metrics=
```

In [25]:
```
model.fit(x_trainr, y_train, epochs=5, validation_split = 0.3) ## Training my model
```

```
Epoch 1/5
1313/1313 [==============================] - 26s 19ms/step - loss: 0.3518 - accurac
y: 0.8880 - val_loss: 0.1345 - val_accuracy: 0.9591
Epoch 2/5
```

```
1313/1313 [==============================] - 24s 19ms/step - loss: 0.1119 - accurac
y: 0.9655 - val_loss: 0.1261 - val_accuracy: 0.9623
Epoch 3/5
1313/1313 [==============================] - 26s 20ms/step - loss: 0.0807 - accurac
y: 0.9749 - val_loss: 0.0788 - val_accuracy: 0.9749
Epoch 4/5
1313/1313 [==============================] - 26s 20ms/step - loss: 0.0622 - accurac
y: 0.9800 - val_loss: 0.0675 - val_accuracy: 0.9793
Epoch 5/5
1313/1313 [==============================] - 26s 20ms/step - loss: 0.0503 - accurac
y: 0.9837 - val_loss: 0.0705 - val_accuracy: 0.9788
```

Out[25]:
```
<keras.callbacks.History at 0x1b84bc786d0>
```

In [26]:
```
### Evaluating on testing dataset MNIST
test_loss, test_acc = model.evaluate(x_testr, y_test)
print("Test loss on 10,000 test samples ",test_loss)
print("Validation Accuracy on 10,000 test samples ",test_acc)
```

```
313/313 [==============================] - 2s 5ms/step - loss: 0.0673 - accuracy: 0.
9796
Test loss on 10,000 test samples  0.06731598824262619
Validation Accuracy on 10,000 test samples  0.9796000123023987
```

In [27]:
```
predictions = model.predict([x_testr])
```

In [28]:
```
print(predictions)
```

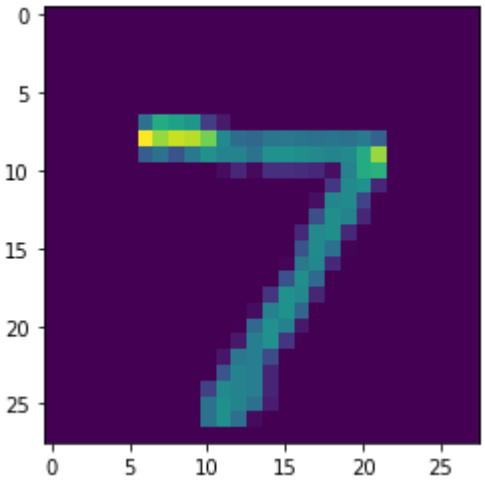
```
[[1.0075364e-08 4.1938529e-07 7.2846314e-07 ... 9.9998963e-01
  3.7899053e-08 8.1203061e-06]
 [2.2520733e-03 8.1896333e-06 9.9697220e-01 ... 1.1457161e-05
  7.6291493e-05 8.9106429e-08]
 [2.7473826e-07 9.9988329e-01 1.0038377e-06 ... 1.7928201e-05
  5.5844421e-06 4.0396847e-07]
 ...
 [1.0216693e-10 4.6228013e-07 3.9864737e-10 ... 1.5555089e-07
  1.0085337e-06 1.5766143e-06]
 [1.5050702e-04 2.5392904e-07 8.8294797e-08 ... 1.0789056e-06
  1.2867320e-03 1.0875900e-05]
 [8.5594296e-04 2.5891336e-07 4.8824097e-04 ... 8.7840294e-07
  1.0400967e-04 2.8484084e-05]]
```

In [29]:
```
print(np.argmax(predictions[0]))
```

```
7
```

In [30]:
```
plt.imshow(x_test[0])
```

Out[30]:
```
<matplotlib.image.AxesImage at 0x1b84b671f70>
```

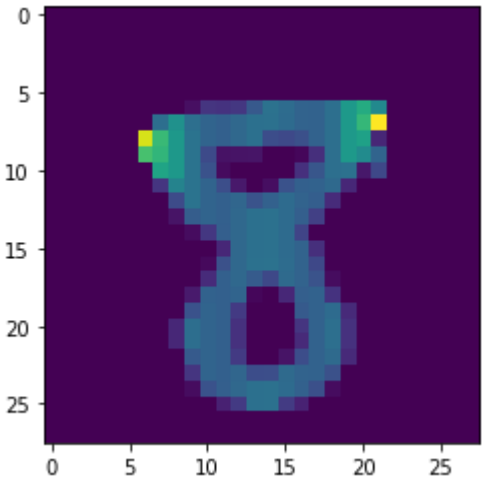


In [31]:
```
print(np.argmax(predictions[128]))
```

```
8
```

In [32]:
```
plt.imshow(x_test[128])
```

Out[32]:
```
<matplotlib.image.AxesImage at 0x1b83e991ee0>
```

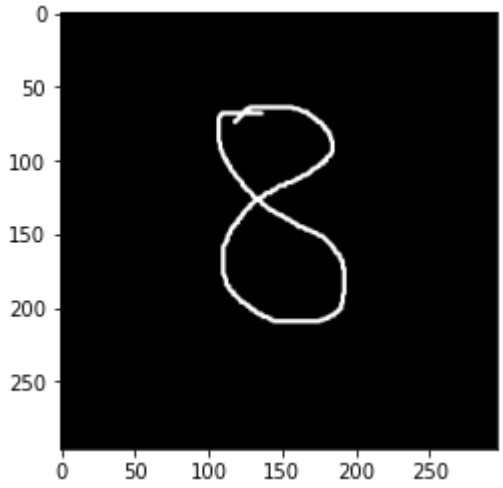


In [33]:
```
import cv2 #importing own image to check the predictions
```

In [36]:
```
img = cv2.imread('eight.png')
```

In [37]:
```
plt.imshow(img)
```

Out[37]:
```
<matplotlib.image.AxesImage at 0x1b84124ea00>
```



In [39]:
```
img.shape
```

Out[39]:
```
(296, 296, 3)
```

In [40]:
```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray.shape
```

Out[40]:
```
(296, 296)
```

In [48]:
```
resized = cv2.resize(gray, (28,28), interpolation = cv2.INTER_AREA)
```

In [49]:
```
resized.shape
```

Out[49]:
```
(28, 28)
```

In [50]:
```
newing = tf.keras.utils.normalize (resized, axis=1)
```

In [51]:
```
newing = np.array(newing).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
```

In [52]:
```
newing.shape
```

Out[52]:
```
(1, 28, 28, 1)
```

In [53]:
```
predictions = model.predict(newing)
print(np.argmax(predictions))
```

```
8
```