Import Library

In [1]:

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
```

Loading dataset

In [2]:

```
mnist=tf.keras.datasets.mnist
print(mnist)
```

```
<module 'keras.api._v2.keras.datasets.mnist' from 'C:\\Users\\yD\\Anaconda3\\lib\\si
te-packages\\keras\\api\\_v2\\keras\\datasets\\mnist\\__init__.py'>
```

After loading the Dataset, Divide them into Train and Test datasets

In [3]:

```
(x_train,y_train),(x_test,y_test)=mnist.load_data()
```

x_train,x_test is represent the data

y_train,y_test is represent the label

Image size 28x28

In [4]:

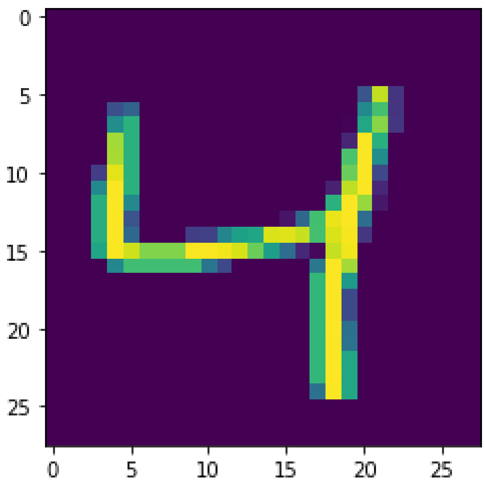
```
x_train.shape,y_train.shape,x_test.shape,y_test.shape
```

Out[4]:

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

In [5]:

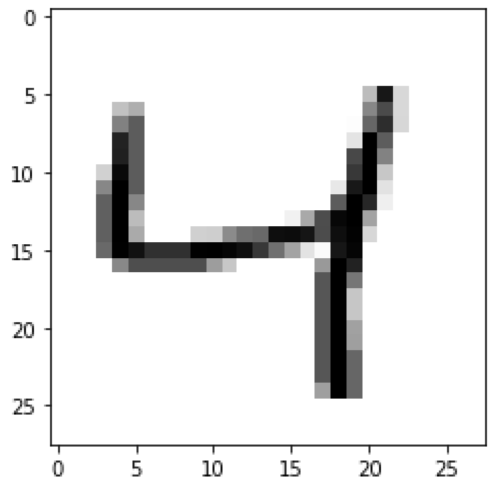
```
plt.imshow(x_train[2])
plt.show()
```



In [6]:

```
plt.imshow(x_train[2], cmap=plt.cm.binary)
```

```python
plt.show()
```



Before Normalization

In [7]:

```python
#Before Normalization all values between 0 to 255
print(x_train[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   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0  67 232  39   0   0   0   0   0]
 [  0   0   0   0  62  81   0   0   0   0   0   0   0   0   0   0   0   0
    0   0 120 180  39   0   0   0   0   0]
 [  0   0   0   0 126 163   0   0   0   0   0   0   0   0   0   0   0   0
    0   2 153 210  40   0   0   0   0   0]
 [  0   0   0   0 220 163   0   0   0   0   0   0   0   0   0   0   0   0
    0  27 254 162   0   0   0   0   0   0]
 [  0   0   0   0 222 163   0   0   0   0   0   0   0   0   0   0   0   0
    0 183 254 125   0   0   0   0   0   0]
 [  0   0   0  46 245 163   0   0   0   0   0   0   0   0   0   0   0   0
    0 198 254  56   0   0   0   0   0   0]
 [  0   0   0 120 254 163   0   0   0   0   0   0   0   0   0   0   0   0
   23 231 254  29   0   0   0   0   0   0]
 [  0   0   0 159 254 120   0   0   0   0   0   0   0   0   0   0   0   0
  163 254 216  16   0   0   0   0   0   0]
 [  0   0   0 159 254  67   0   0   0   0   0   0   0   0   0  14  86 178
  248 254  91   0   0   0   0   0   0   0]
 [  0   0   0 159 254  85   0   0   0  47  49 116 144 150 241 243 234 179
  241 252  40   0   0   0   0   0   0   0]
 [  0   0   0 150 253 237 207 207 207 253 254 250 240 198 143  91  28   5
  233 250   0   0   0   0   0   0   0   0]
 [  0   0   0   0 119 177 177 177 177 177  98  56   0   0   0   0   0 102
  254 220   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 169
  254 137   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 169
```

```
 254  57   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 169
 254  57   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 169
 255  94   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 169
 254  96   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 169
 254 153   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 169
 255 153   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  96
 254 153   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]]
```

Normalization

In [8]:

```
# Normalization is a pre-processing technique used to standardize data.

# In other words, having different sources of data inside the same range.

# Not normalizing the data before training can cause problems in our network.

# Making it drastically harder to train and decrease its learning speed.

# Normalization can also be done by x_train/255 and x_test/255.
```

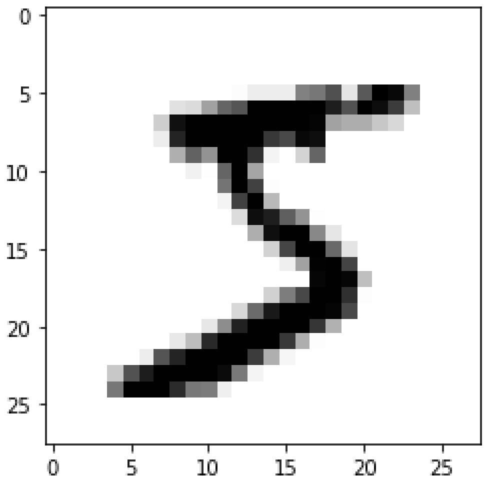
In [9]:

```
X_Train=tf.keras.utils.normalize(x_train)

X_Test=tf.keras.utils.normalize(x_test)

plt.imshow(x_train[0], cmap= plt.cm.binary)
```

Out[9]: <matplotlib.image.AxesImage at 0x1bd9b895820>



After Normalization

In [10]:
```
# After Normalization all values between 0 to 1
```

In [11]:
```
print(X_Train[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.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.27390552 0.94844896 0.15943754 0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.25584473 0.33424876
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.49518335 0.74277503 0.16093459 0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.37724213 0.48801958
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.00598797 0.45807973 0.62873688 0.11975941 0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.5392254  0.399517
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.06617766 0.62256024 0.39706598 0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.50999727 0.37445745
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.42040315 0.58351039 0.28716062 0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.10401864 0.5540123  0.36858778
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.44773239 0.57436377 0.12663138 0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.25313301 0.53579821 0.34383901
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
```

```
 0.04851716 0.48728105 0.53579821 0.06117381 0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.32308399 0.51612159 0.24383697
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.33121189 0.51612159 0.43890655 0.0325116  0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.30721383 0.49076926 0.12945488
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.02705027 0.16616597 0.34392491
 0.47917629 0.49076926 0.17582678 0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.22820814 0.36455892 0.12199806
 0.         0.         0.         0.06745775 0.0703283  0.16649147
 0.20667907 0.2152907  0.34590039 0.34877093 0.33585349 0.25691357
 0.34590039 0.36168838 0.05741085 0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.17859902 0.30123701 0.28218645
 0.24646665 0.24646665 0.24646665 0.30123701 0.30242767 0.29766503
 0.28575843 0.23575071 0.1702644  0.10835007 0.03333848 0.0059533
 0.27742381 0.29766503 0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.21481894 0.31952061
 0.31952061 0.31952061 0.31952061 0.31952061 0.17690972 0.10109127
 0.         0.         0.         0.         0.         0.18413052
 0.45852111 0.39714426 0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.50533162
 0.7594925  0.40964753 0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.54452139
 0.81839309 0.18365514 0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.54452139
 0.81839309 0.18365514 0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.52806828
 0.79678941 0.29371845 0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.52840114
 0.79416503 0.30015686 0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.49516488
 0.74421231 0.44828537 0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.49408658
 0.74551525 0.44730915 0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
```

```
  0.         0.         0.         0.         0.         0.30801364
  0.81495275 0.49089674 0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]]
```

Resizing the image

In [12]:

```
#Reshaping the array to 4-dims so that it can work with the Keras API(greyscale imag

# Size of image is 28 x 28

# -1 is used to increase by 1 dimension

# Numpy is used to reshape



X_Train=np.array(x_train).reshape(-1,28,28,1)
X_Test=np.array(x_test).reshape(-1,28,28,1)
```

Create Deep Neural Networks

Training 60,000 samples of handwritten dataset

In [13]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout,Activation,Flatten,Conv2D,MaxPooli
```

In [14]:

```
# Start neural network by using Sequential()

model= Sequential()
```

In [15]:

```
# First Convolutional Layear

model.add(Conv2D(64,(3,3),input_shape=X_Train.shape[1:]))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
```

In [16]:

```
# Second Convolutional Layear
model.add(Conv2D(64,(3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
```

In [17]:

```
# Third Convolutional Layear
model.add(Conv2D(64,(3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
```

In [18]:

```
# Fully Connected Layear 1

model.add(Flatten())
model.add(Dense(64))
model.add(Activation("relu"))
```

In [19]:

```
# Fully Connected Layear 2
model.add(Dense(32))
model.add(Activation("relu"))
```

In [20]:

```
# Fully Connected Layear 3

model.add(Dense(16))
model.add(Activation("relu"))
```

In [21]:

```
# Last Fully Connected Layear

# The softmax function is used as the activation function in the output layer of neu
# That is, softmax is used as the activation function for multi-class classification

model.add(Dense(10))
model.add(Activation("softmax"))
```

In [22]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 64)        640

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

 max_pooling2d (MaxPooling2D  (None, 13, 13, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        36928

 activation_1 (Activation)   (None, 11, 11, 64)        0

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 3, 3, 64)          36928

 activation_2 (Activation)   (None, 3, 3, 64)          0

 max_pooling2d_2 (MaxPooling  (None, 1, 1, 64)         0
 2D)
```

```
 flatten (Flatten)           (None, 64)                0

 dense (Dense)               (None, 64)                4160

 activation_3 (Activation)   (None, 64)                0

 dense_1 (Dense)             (None, 32)                2080

 activation_4 (Activation)   (None, 32)                0

 dense_2 (Dense)             (None, 16)                528

 activation_5 (Activation)   (None, 16)                0

 dense_3 (Dense)             (None, 10)                170

 activation_6 (Activation)   (None, 10)                0

=================================================================
Total params: 81,434
Trainable params: 81,434
Non-trainable params: 0
_________________________________________________________________
```

In [23]:

```
model.compile(loss="sparse_categorical_crossentropy",optimizer="adam", metrics="accu
```

Training Model

In [24]:

```
model.fit(X_Train,y_train,epochs=10,validation_split=0.3)
```

```
Epoch 1/10
1313/1313 [==============================] - 32s 24ms/step - loss: 0.5993 - accurac
y: 0.8171 - val_loss: 0.1592 - val_accuracy: 0.9576
Epoch 2/10
1313/1313 [==============================] - 34s 26ms/step - loss: 0.1369 - accurac
y: 0.9614 - val_loss: 0.1082 - val_accuracy: 0.9702
Epoch 3/10
1313/1313 [==============================] - 35s 27ms/step - loss: 0.0927 - accurac
y: 0.9740 - val_loss: 0.1067 - val_accuracy: 0.9694
Epoch 4/10
1313/1313 [==============================] - 35s 26ms/step - loss: 0.0755 - accurac
y: 0.9785 - val_loss: 0.0983 - val_accuracy: 0.9728
Epoch 5/10
1313/1313 [==============================] - 36s 28ms/step - loss: 0.0611 - accurac
y: 0.9828 - val_loss: 0.0801 - val_accuracy: 0.9793
Epoch 6/10
1313/1313 [==============================] - 40s 31ms/step - loss: 0.0496 - accurac
y: 0.9851 - val_loss: 0.1055 - val_accuracy: 0.9748
Epoch 7/10
1313/1313 [==============================] - 34s 26ms/step - loss: 0.0484 - accurac
y: 0.9856 - val_loss: 0.0748 - val_accuracy: 0.9821
Epoch 8/10
1313/1313 [==============================] - 35s 27ms/step - loss: 0.0392 - accurac
y: 0.9884 - val_loss: 0.0741 - val_accuracy: 0.9819
Epoch 9/10
1313/1313 [==============================] - 34s 26ms/step - loss: 0.0331 - accurac
y: 0.9905 - val_loss: 0.0912 - val_accuracy: 0.9805
Epoch 10/10
1313/1313 [==============================] - 36s 28ms/step - loss: 0.0309 - accurac
y: 0.9910 - val_loss: 0.0795 - val_accuracy: 0.9817
```

Out[24]: <keras.callbacks.History at 0x1bd9c27ecd0>

In [25]:
```
# Evaluating on testing data set MNIT
test_loss,test_accuarcy=model.evaluate(X_Test,y_test)
print("Test loss on 10,000 test samples",test_loss)

print("Validation accuracy on 10,000 test samples",test_accuarcy)
```

```
313/313 [==============================] - 2s 8ms/step - loss: 0.0751 - accuracy: 0.
9815
Test loss on 10,000 test samples 0.07512281835079193
Validation accuracy on 10,000 test samples 0.9815000295639038
```

In [26]:
```
prediction=model.predict(X_Test)
```

In [27]:
```
print(prediction)
```

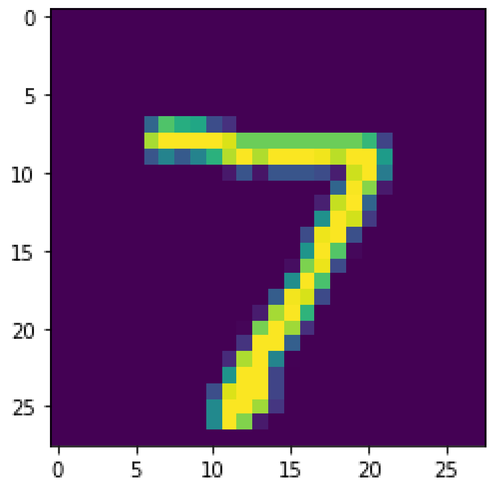
```
[[4.2913801e-13 9.3200230e-09 8.1251283e-06 ... 9.9999189e-01
  2.3413513e-08 2.1857410e-08]
 [5.0873826e-11 1.0185290e-07 9.9989414e-01 ... 8.4514875e-05
  1.2566670e-06 2.5037794e-16]
 [1.9400438e-12 9.9999702e-01 2.0900698e-10 ... 1.1003044e-09
  1.8166629e-07 3.2849112e-11]
 ...
 [3.5160285e-22 8.9691912e-35 7.6634379e-23 ... 0.0000000e+00
  1.5196358e-18 2.5844592e-30]
 [5.1401481e-25 2.5663874e-15 9.0965473e-16 ... 1.0140422e-23
  1.3378981e-11 1.6683961e-06]
 [1.4972705e-09 3.6003169e-21 7.1032251e-14 ... 3.5213394e-32
  9.5809680e-12 2.8880821e-15]]
```

In [28]:
```
print(np.argmax(prediction[0]))
```

```
7
```

In [29]:
```
# Check value is true or not
plt.imshow(X_Test[0])
```

Out[29]: <matplotlib.image.AxesImage at 0x1bd9d7bbeb0>



Now try to check on our image

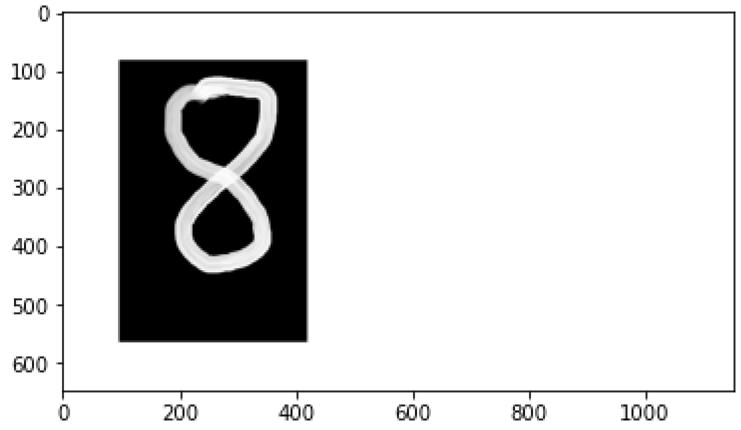
In [43]:
```python
# Try to check wheather it will able to predict on our image or not
import cv2
```

In [45]:
```python
# read image

img=cv2.imread("C:/Users/yD/Desktop/Eight.jpg")

plt.imshow(img)

plt.show()
```



In [46]:
```python
# check shape of your image
img.shape
```

Out[46]: (648, 1152, 3)

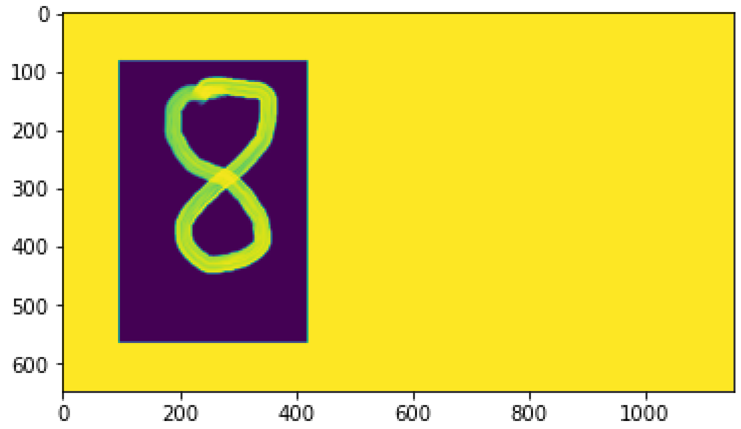
In [55]:
```python
# image have 3 channel
# First convert into grey image then resize in 28x28

grey=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

plt.imshow(grey)

grey.shape
```

Out[55]: (648, 1152)



In [57]:
```python
# Resize the image

resizedImg=cv2.resize(grey,(28,28),interpolation=cv2.INTER_AREA)
```

```
resizedImg.shape
```

Out[57]:
```
(28, 28)
```

In [58]:
```
# Now normalize the image

img1=tf.keras.utils.normalize(resizedImg)
```

In [64]:
```
# Change dimension of an image
img1=np.array(img1).reshape(-1,28,28,1)
img1.shape
```

Out[64]:
```
(1, 28, 28, 1)
```

In [65]:
```
# Select the Model
prediction=model.predict(img1)
```

In [67]:
```
# Final prediction
print(np.argmax(prediction))
```

```
8
```

In [ ]:
```

```