Let's Grow More

Name:Rashmi Ranjan Nayak

Task #2

Develop A Neural Network That Can Read Handwriting:

In [1]:

```
import tensorflow as tf
```

Loading MNIST dataset from tensorflow datasets

In [2]:

```
mnist = tf.keras.datasets.mnist
```

train-test splitting

In [3]:

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

In [4]:

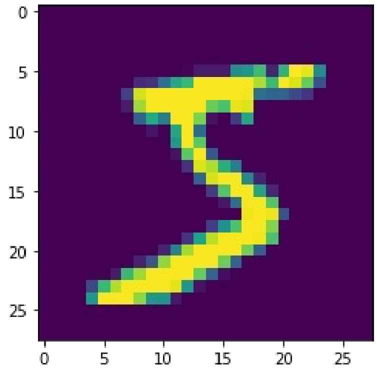
```
x_train.shape
```

Out[4]:

(60000, 28, 28)

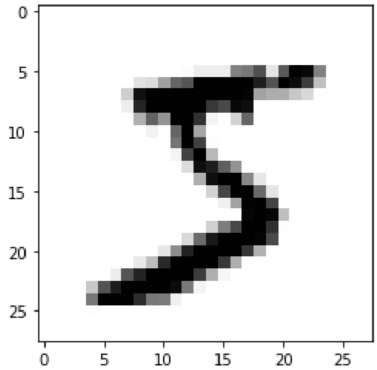
In [5]:

```
import matplotlib.pyplot as plt
plt.imshow(x_train[0])
plt.show()
plt.imshow(x_train[0], cmap=plt.cm.binary)
```



Out[5]:

<matplotlib.image.AxesImage at 0x21d04c68bc8>



Checking the values of each pixel before Normalization:

In [6]:

```
print(x_train[0])
```

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

As image are in Gray level(1 channel ==> 0to 255),not colored (RGB)

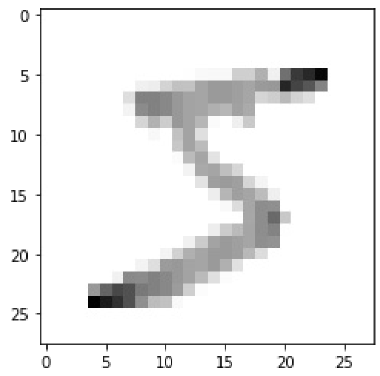
Normalizing the data | Pre-processing step

In [7]:

```
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[7]:

```
<matplotlib.image.AxesImage at 0x21d00009f88>
```



After Normalization:

In [8]:

```
print(x_train[0])
```

```
[[0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         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.         0.05780486 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.         0.12250613 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.         0.04500225 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.         0.1541463  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.         0.0253731  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.         0.20500962
 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.         0.01622378
 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.
 0.04586451 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.         0.10498298 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 [9]:

```
print(y_train[0])
```

```
5
```

Resizing image to make it suitable for apply Convolution operation:

In [10]:

```
import numpy as np
IMG_SIZE = 28
x_trainr = np.array(x_train).reshape(-1,IMG_SIZE,IMG_SIZE,1) ## Increasing one dimension for kernel operation
x_testr = np.array(x_test).reshape(-1,IMG_SIZE,IMG_SIZE,1)  ## Increasing one dimension for kernel operation
print("Training Sample demension",x_trainr.shape)
print("Testing Sample demension",x_testr.shape)
```

```
Training Sample demension (60000, 28, 28, 1)
Testing Sample demension (10000, 28, 28, 1)
```

Creating a Deep Neural Network:

Training on 60,000 of MNIST handwritten dataset

In [11]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D
```

In [12]:

```
#### Creating a neural network now
model = Sequential()

##### First Convolution Layer 0 1 2 3 (60000,28,28,1)
model.add(Conv2D(64, (3,3), input_shape = x_trainr.shape[1:]))### Only for first convolution Layer to mention inp
ut Layer size
model.add(Activation("relu"))## activation function to make it non-linear, <0, remove, >0
model.add(MaxPooling2D(pool_size=(2,2)))## Maxpooling single maximum value of 2x2

##### 2nd Convolutional Layer
model.add(Conv2D(64, (3,3))) ## 2nd Convolutional Layer
model.add(Activation("relu")) ## activation function
model.add(MaxPooling2D(pool_size=(2,2))) ## Maxpooling

##### 3rd Convolutional 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 Layer, needvto be flatten so that 2D to 1D
model.add(Dense(64))
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)) ## this  Last dense Layer must be equal to 10
model.add(Activation("softmax")) ### activation function is changed to softmax (class probabiliities)
```

Model summery:

In [13]:

```
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 (MaxPooling2 (None, 5, 5, 64)          0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928     
_________________________________________________________________
activation_2 (Activation)    (None, 3, 3, 64)          0         
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 1, 1, 64)          0         
_________________________________________________________________
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, 10)                330       
_________________________________________________________________
activation_5 (Activation)    (None, 10)                0         
=================================================================
Total params: 81,066
Trainable params: 81,066
Non-trainable params: 0
_________________________________________________________________
```

In [14]:

```
print("Total Training Samples = ",len(x_trainr))
```

```
Total Training Samples =  60000
```

In [15]:

```
model.compile(loss ="sparse_categorical_crossentropy", optimizer = "adam", metrics=["accuracy"])
```

In [16]:

```
model.fit(x_trainr, y_train,epochs=5, validation_split = 0.3, batch_size=1) ## Training Model
```

```
Epoch 1/5
42000/42000 [==============================] - 108s 3ms/step - loss: 0.2393 - accuracy: 0.9268 - val
_loss: 0.1264 - val_accuracy: 0.9658
Epoch 2/5
42000/42000 [==============================] - 112s 3ms/step - loss: 0.1181 - accuracy: 0.9677 - val
_loss: 0.1131 - val_accuracy: 0.9698
Epoch 3/5
42000/42000 [==============================] - 114s 3ms/step - loss: 0.1029 - accuracy: 0.9739 - val
_loss: 0.0966 - val_accuracy: 0.9754
Epoch 4/5
42000/42000 [==============================] - 116s 3ms/step - loss: 0.0985 - accuracy: 0.9750 - val
_loss: 0.0987 - val_accuracy: 0.9776
Epoch 5/5
42000/42000 [==============================] - 115s 3ms/step - loss: 0.0981 - accuracy: 0.9756 - val
_loss: 0.1092 - val_accuracy: 0.9729
```

Out[16]:

```
<keras.callbacks.History at 0x21d03bf72c8>
```

In [17]:

```
test_loss, test_acc = model.evaluate(x_testr, y_test, batch_size=1)
print("Test loss on 10,000 test samples", test_loss)
print("Validation Accuracy on 10,000 test samples", test_acc)
```

```
10000/10000 [==============================] - 11s 1ms/step - loss: 0.0881 - accuracy: 0.9779
Test loss on 10,000 test samples 0.0880742147564888
Validation Accuracy on 10,000 test samples 0.9779000282287598
```

In [18]:

```
predictions = model.predict([x_testr])
```

In [19]:

```
print(predictions)
```

```
[[3.98236994e-20 1.76070799e-07 1.46776614e-07 ... 9.99994159e-01
  2.34744686e-08 4.24170821e-06]
 [8.63195769e-03 4.09690023e-04 9.87197697e-01 ... 3.71094840e-03
  1.63410950e-05 3.28977694e-06]
 [1.40158605e-15 1.00000000e+00 2.51318161e-10 ... 3.70012976e-09
  1.03811404e-10 4.27631715e-12]
 ...
 [1.55509511e-21 8.60572028e-16 9.18513549e-15 ... 5.83426502e-11
  1.72200036e-13 2.81773049e-09]
 [3.62762184e-24 5.50039849e-19 3.21641406e-21 ... 5.49855232e-19
  4.79376033e-13 2.08339550e-08]
 [2.40092987e-07 2.90620167e-10 1.04741502e-11 ... 4.20690799e-13
  2.54795696e-09 4.18601388e-07]]
```

In [20]:

```
## in order to understand, convert the predictions from one hot encoding, we need to use numpy for that
print(np.argmax(predictions[0])) ### so actually argmax will return the maximum value indexx and find the value o
f it
```

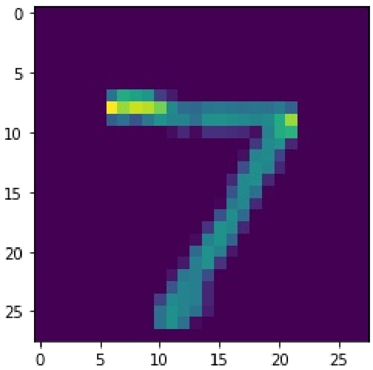
```
7
```

In [21]:

```
### now to check that is our answer is true or not
plt.imshow(x_test[0])
```

Out[21]:

```
<matplotlib.image.AxesImage at 0x21d02134b48>
```



In [22]:

```
print(np.argmax(predictions[128]))
```

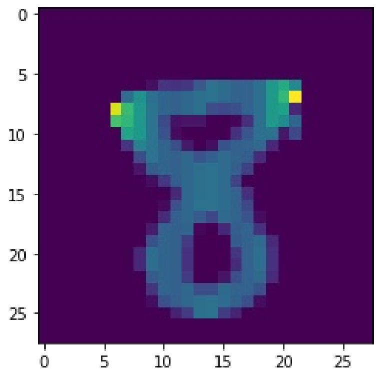
```
8
```

In [23]:

```
plt.imshow(x_test[128])
```

Out[23]:

<matplotlib.image.AxesImage at 0x21d02227ac8>



In [24]:

```
import cv2
```

In [84]:

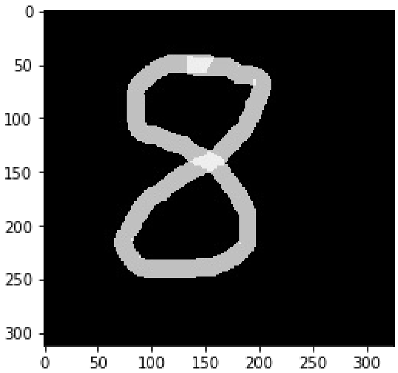
```
img = cv2.imread("8test.png")
```

In [85]:

```
plt.imshow(img)
```

Out[85]:

<matplotlib.image.AxesImage at 0x21d0272c348>



In [86]:

```
img.shape
```

Out[86]:

(312, 326, 3)

In [87]:

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

In [88]:

```
gray.shape
```

Out[88]:

(312, 326)

In [89]:

```
resized = cv2.resize(gray, (28,28), interpolation = cv2.INTER_AREA)
```

In [90]:

```
resized.shape
```

Out[90]:

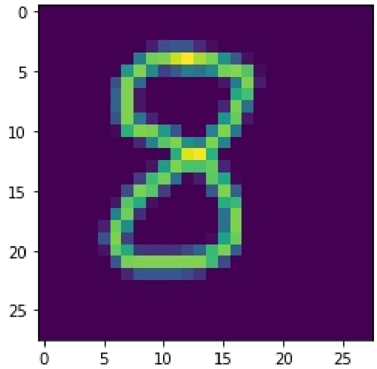
```
(28, 28)
```

In [91]:

```
plt.imshow(resized)
```

Out[91]:

```
&lt;matplotlib.image.AxesImage at 0x21d0276a2c8&gt;
```



In [92]:

```
newimg = tf.keras.utils.normalize (resized, axis =1) ## 0 to 1 scaling
```

In [93]:

```
newimg = np.array(newimg).reshape(-1,IMG_SIZE, IMG_SIZE, 1) ## kernel operation of convolutional layer
```

In [94]:

```
newimg.shape
```

Out[94]:

```
(1, 28, 28, 1)
```

In [95]:

```
predictions = model.predict(newimg)
```

In [96]:

```
print(np.argmax(predictions))
```

```
8
```

In [ ]:

```

```