**Importing the required libraries**

In [ ]:

**import** numpy

**import** tensorflow *#open source used for both ML and DL for computation*

**from** tensorflow.keras.datasets **import** mnist *#mnist dataset*

**from** tensorflow.keras.models **import** Sequential *#it is a plain stack of layers*

**from** tensorflow.keras **import** layers *#A Layer consists of a tensor- in tensor-out computat ion funct ion*

**from** tensorflow.keras.layers **import** Dense, Flatten *#Dense-Dense Layer is the regular deeply connected r*

*#faltten -used fot flattening the input or change the dimension*

**from** tensorflow.keras.layers **import** Conv2D *#onvoLutiona l Layer*

**from** keras.optimizers **import** Adam *#opt imizer*

**from** keras. utils **import** np\_utils *#used for one-hot encoding*

**import** matplotlib.pyplot **as** plt *#used for data visualization*

**load data**

In [2]:

(x\_train, y\_train), (x\_test, y\_test)**=**mnist**.**load\_data () *#splitting the mnist data into train and test*

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz

11490434/11490434 [==============================] - 3s 0us/step

In [3]:

print (x\_train**.**shape) *#shape is used for give the dimens ion values #60000-rows 28x28-pixels*

print (x\_test**.**shape)

(60000, 28, 28)

(10000, 28, 28)

**Understanding the data**

In [4]:

x\_train[0]

Out[4]:

array([[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 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]], dtype=uint8)

In [5]:

y\_train[0]

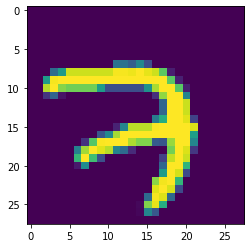
Out[5]:

5

In [6]:

plt**.**imshow(x\_train[5000]) *#ploting the index=image*

Out[6]:



**Reshaping The Data**

In [7]:

*#Reshaping to format which CNN expects (batch, height, width, channels)*

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**

In [8]:

number\_of\_classes **=** 10 *#storing the no of classes in a variable*

In [9]:

y\_train **=** np\_utils**.**to\_categorical (y\_train, number\_of\_classes) *#converts the output in binary format*

y\_test **=** np\_utils**.**to\_categorical (y\_test, number\_of\_classes)

In [10]:

y\_train[0] *#Printing the new label*

Out[10]:

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

**Add CNN Layers**

In [11]:

*#create 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'))

*#flatten the dimension of the image*

model**.**add(Flatten())

*#output layer with 10 neurons*

model**.**add(Dense(number\_of\_classes,activation **=** 'softmax'))

**Compiling The Model**

In [12]:

*#Compile model*

model**.**compile(loss**=** 'categorical\_crossentropy', optimizer**=**"Adam", metrics**=**['accuracy'])

**Train The Model**

In [13]:

*#fit the model*

model**.**fit(x\_train, y\_train, validation\_data**=**(x\_test, y\_test), epochs**=**5, batch\_size**=**32)

Epoch 1/5

1875/1875 [==============================] - 208s 110ms/step - loss: 0.2258 - accuracy: 0.9531 - val\_loss: 0.0743 - val\_accuracy: 0.9764

Epoch 2/5

1875/1875 [==============================] - 205s 109ms/step - loss: 0.0673 - accuracy: 0.9792 - val\_loss: 0.0742 - val\_accuracy: 0.9777

Epoch 3/5

1875/1875 [==============================] - 212s 113ms/step - loss: 0.0456 - accuracy: 0.9862 - val\_loss: 0.0857 - val\_accuracy: 0.9755

Epoch 4/5

1875/1875 [==============================] - 211s 112ms/step - loss: 0.0370 - accuracy: 0.9889 - val\_loss: 0.1036 - val\_accuracy: 0.9748

Epoch 5/5

1875/1875 [==============================] - 202s 108ms/step - loss: 0.0271 - accuracy: 0.9917 - val\_loss: 0.0972 - val\_accuracy: 0.9809

Out[13]:

**Observing The Metrics**

In [14]:

*# Final evaluation of the model*

metrics **=** model**.**evaluate(x\_test, y\_test, verbose**=**0)

print("Metrics (Test loss &Test Accuracy) : ")

print(metrics)

Metrics (Test loss &Test Accuracy) :

[0.09719487279653549, 0.98089998960495]

**Test The Model**

In [15]:

prediction**=**model**.**predict(x\_test[6000:6001])

print(prediction)

1/1 [==============================] - 1s 636ms/step

[[5.1234290e-12 2.9132625e-12 4.7156525e-11 1.9892369e-07 1.4272153e-02

4.1180425e-05 6.8472568e-15 7.2068692e-04 6.3077291e-06 9.8495942e-01]]

In [17]:

**import** numpy **as** np

print(np**.**argmax(prediction, axis**=**1)) *#printing our Labels from first 4 images*

np**.**argmax(y\_test[5000:5001]) *#printing the actual labels*

[9]

Out[17]:

3

**Save The Model**

In [18]:

*# Save the model*

model**.**save('models/project.h5')

In [ ]: