#Importing libraries

```
#for working with arrays
import numpy as np
#open source used for both ML and DL for computation
import tensorflow
#mnist dataset
from tensorflow.keras.datasets import mnist
#it is a plain stack of layers
from tensorflow.keras.models import Sequential
#A Layer consists of a tensor- in tensor-out computat ion function
from tensorflow.keras import layers
#Dense-Dense Layer is the regular deeply connected layers
#faltten -used fot flattening the input or change the dimension
from tensorflow.keras.layers import Dense, Flatten
#Convolutional Layer
from tensorflow.keras.layers import Conv2D
#Optimizer
from keras.optimizers import Adam
#Used for one-hot encoding
from keras. utils import np_utils
#for data visualization
import matplotlib.pyplot as plt
```

#LOADING DATA

```
#splitting the mnist data into train and test
(x_train, y_train), (x_test, y_test)=mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
#shape is used for give the dimension values #60000-rows 28x28-pixels
print(x_train.shape)
print(x_test.shape)
```

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

```
x_train[0]
```

```
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,
```

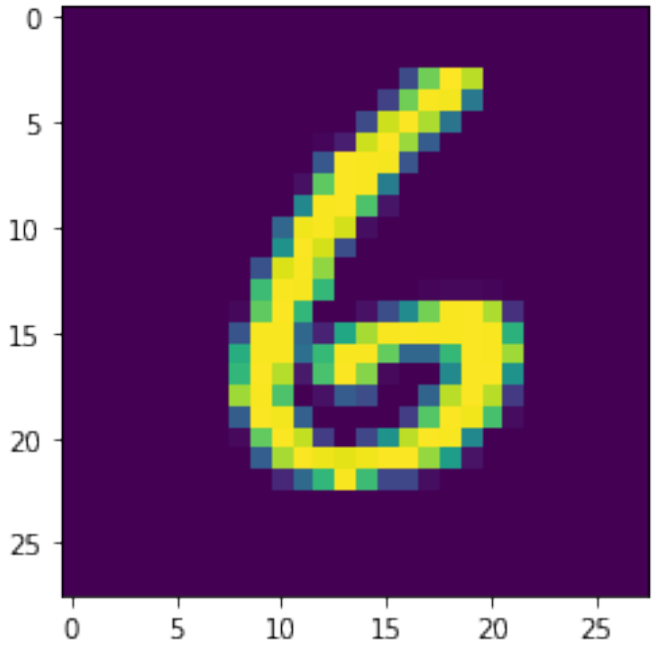
```
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,
          0,   0,   0,   0,   0,   0,   0,   0,   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)
```

```
#Plotting the image
plt.imshow(x_train[6000])
```

```
<matplotlib.image.AxesImage at 0x7f07a227f490>
```




```
np.argmax(y_train[6000])
```

```
0
```

#Reshaping dataset

```
#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')

#Storing number of classes in a variable
number_of_classes = 10

#converts the output in binary format
y_train = np_utils.to_categorical (y_train, number_of_classes)
y_test = np_utils.to_categorical (y_test, number_of_classes)
```

#Add CNN Layers

```
#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

```
#Compile model
model.compile(loss= 'categorical_crossentropy', optimizer="Adam",
metrics=['accuracy'])
x_train = np.asarray(x_train)
y_train = np.asarray(y_train)
```

#Training the model

```
#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 [==============================] - 189s 101ms/step - loss:
0.0247 - accuracy: 0.9923 - val_loss: 0.1239 - val_accuracy: 0.9763
Epoch 2/5
1875/1875 [==============================] - 185s 99ms/step - loss:
0.0227 - accuracy: 0.9931 - val_loss: 0.1384 - val_accuracy: 0.9766
Epoch 3/5
1875/1875 [==============================] - 184s 98ms/step - loss:
0.0192 - accuracy: 0.9944 - val_loss: 0.1439 - val_accuracy: 0.9734
Epoch 4/5
1875/1875 [==============================] - 186s 99ms/step - loss:
0.0167 - accuracy: 0.9949 - val_loss: 0.1475 - val_accuracy: 0.9786
Epoch 5/5
1875/1875 [==============================] - 187s 100ms/step - loss:
0.0153 - accuracy: 0.9956 - val_loss: 0.2663 - val_accuracy: 0.9695

<keras.callbacks.History at 0x7f079d8e3a50>
```

```
# 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) :
[0.2662925720214844, 0.9695000052452087]
```

```
prediction=model.predict(x_test[6000:6001])
print(prediction)
```

```
1/1 [==============================] - 0s 21ms/step
[[1.90929128e-22 3.87971029e-33 1.21386323e-17 4.28218216e-09
  1.16075036e-07 1.51229324e-11 4.69814482e-31 3.22366759e-07
  1.32163915e-08 9.99999523e-01]]
```

```
#printing our Labels from first 4 images
import numpy as np
print(np.argmax(prediction, axis=1))
```

```
[9]
```

```
#Printing the actual labels
np.argmax(y_test[6000:6001])
```

```
9
```

#Save the model

```
# Save the model
model.save('handwritten/mnistCNN.h5')
```