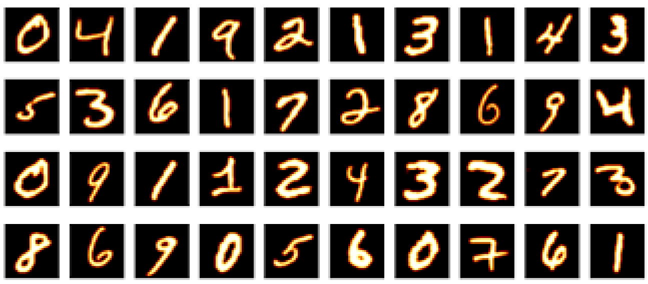
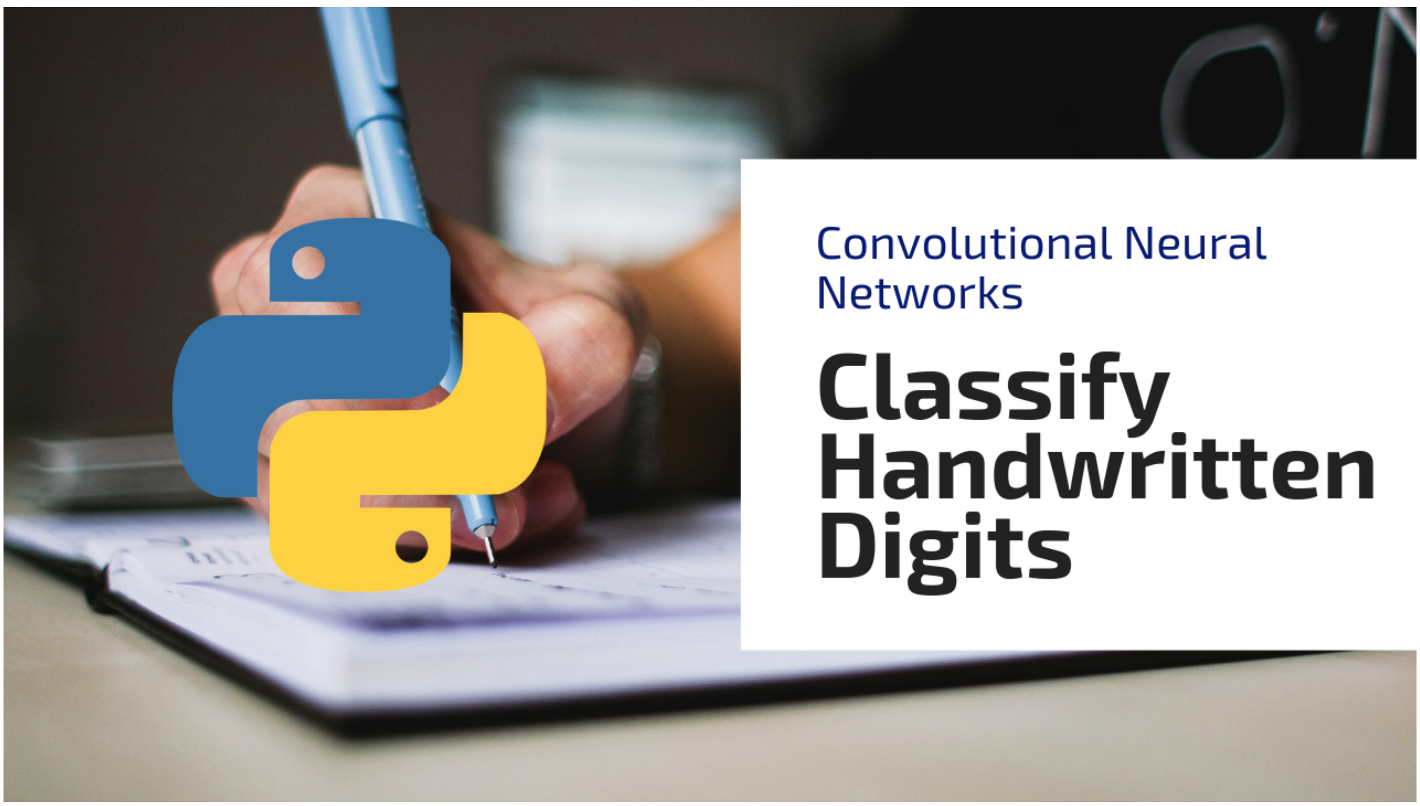
```
import·numpy·as·np
import·matplotlib.pyplot·as·plt
import·pandas·as·pd
import·os
import·tensorflow·as·tf
import·seaborn·as·sns
```






```
#·Go·to·tensorflow·documentation·->·tf.keras·->·dataset·->·mnist·()
mnist·=·tf.keras.datasets.mnist
```

```
mnist
```

```
<module 'keras.api._v2.keras.datasets.mnist' from '/usr/local/lib/python3.8/dist-packages/keras/api/_v2/keras/datasets/mnist/__init__.py'>
```

```
# These is the format from Keras documentation to load the dataset and assign the variable

(X_train_full, y_train_full), (X_test, y_test) = mnist.load_data()
```

```
(X_train_full,y_train_full),·(X_test,·y_test)·=·mnist.load_data()
```

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

```
X_train_full
```

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

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

```
# The dataset contains 60000 images and array size as 28X28

X_train_full.shape
```

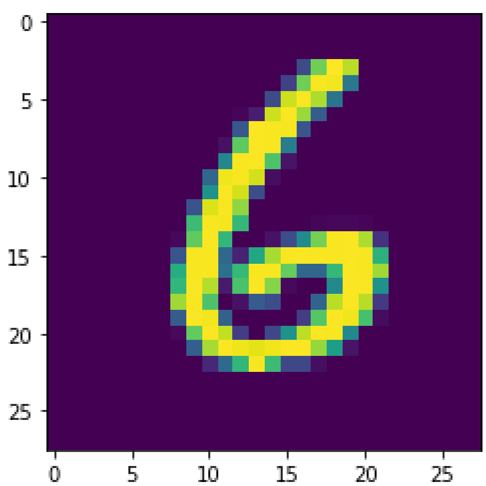
```
(60000, 28, 28)
```

```
X_train_full[5000]
```

```
          0,   0],
       [  0,   0, 140, 251, 254, 254, 254, 254, 254, 254, 254, 254, 254,
        254, 254, 254, 254, 254, 189,  23,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0, 226, 254, 208, 199, 199, 199, 199, 139,  61,  61,  61,
         61,  61, 128, 222, 254, 254, 189,  21,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,  38,  82,  13,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,  34, 213, 254, 254, 115,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,  84, 254, 254, 234,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,  84, 254, 254, 234,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0, 106, 157, 254, 254, 243,  51,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  25, 117,
        228, 228, 228, 253, 254, 254, 254, 254, 240,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  68, 119, 220, 254,
        254, 254, 254, 254, 254, 254, 254, 254, 142,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  37, 187, 253, 254, 254, 254,
        223, 206, 206,  75,  68, 215, 254, 254, 117,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0, 113, 219, 254, 242, 227, 115,  89,
         31,   0,   0,   0,   0, 200, 254, 241,  41,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0, 169, 254, 176,  62,   0,   0,   0,
          0,   0,   0,   0,  48, 231, 254, 234,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,  18, 124,   0,   0,   0,   0,   0,
          0,   0,   0,   0,  84, 254, 254, 166,   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, 254, 238,  57,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0, 210, 250, 254, 168,   0,   0,   0,   0,   0,   0,
          0,   0]
```

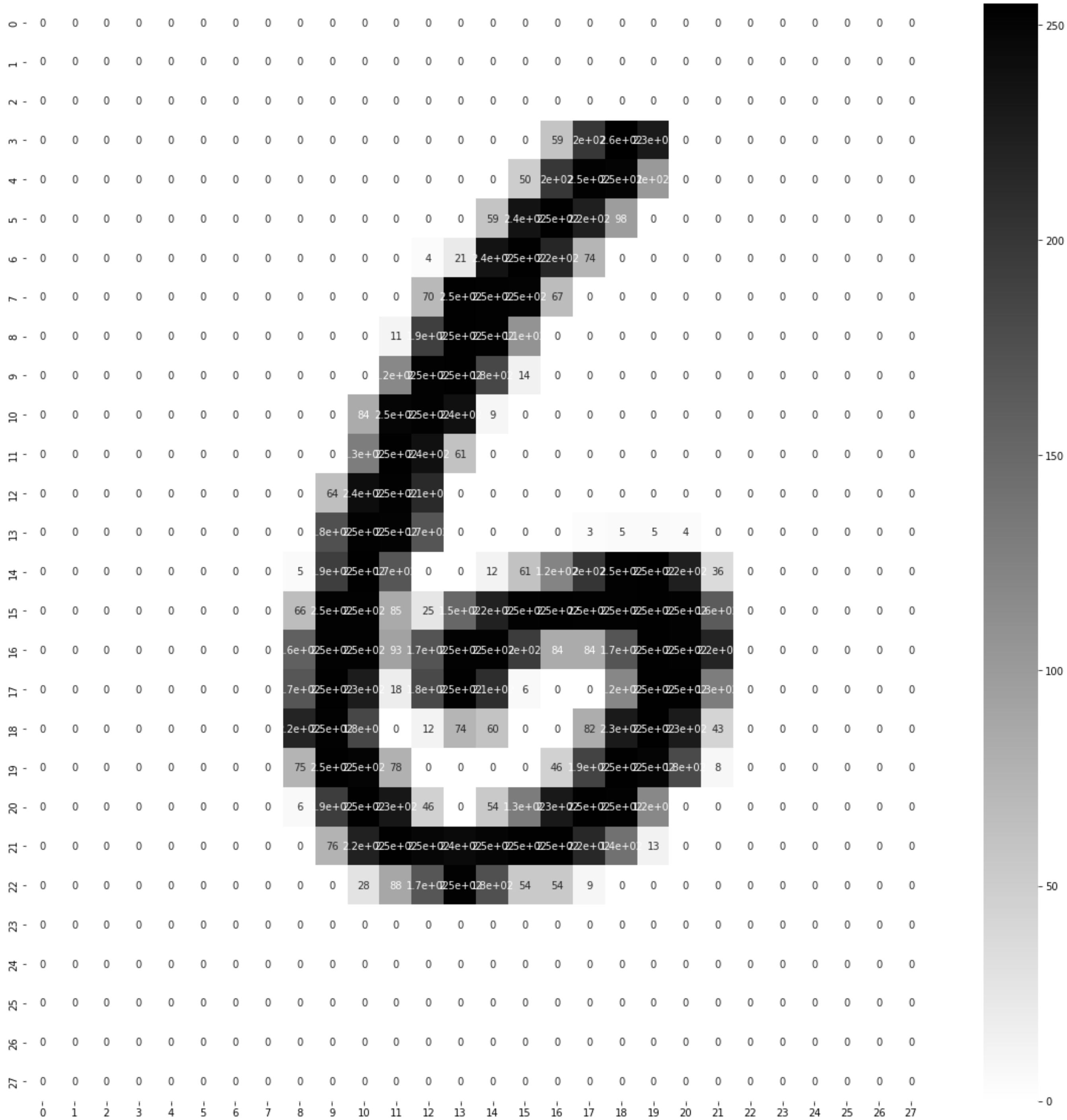
```
        0,   0],
      [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0, 242, 254, 239,  57,   0,   0,   0,   0,   0,   0,
         0,   0],
      [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,  89, 251, 241,  86,   0,   0,   0,   0,   0,   0,   0,
         0,   0],
      [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   5, 206, 246, 157,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0],
      [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   4, 117,  69,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0],
      [  0,   0,   0,   0,   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)
```

```
# Let us plot to see the image 

plt.imshow(X_train_full[6000])#, cmap = "binary")
#plt.axis("off")
plt.show()
```

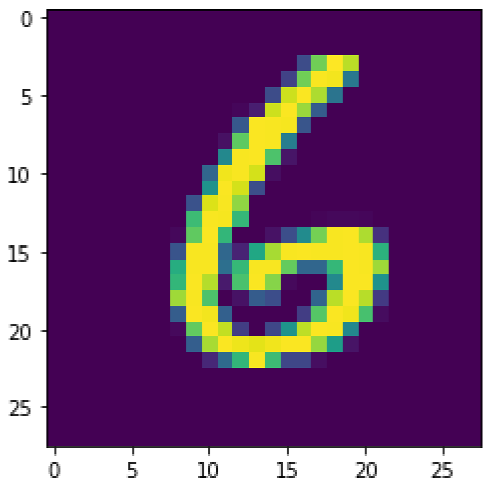



```
# To zoom the image, let us plot the heatmap. Image is collection of pixels between 0 to 256 
```

```
plt.figure(figsize = (20, 20))
sns.heatmap(X_train_full[6000], annot = True, cmap = "binary")
plt.show()
```


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
250
200
150
100
50
0

```
plt.imshow(X_train_full[6000])#, cmap = "binary")
#plt.axis("off")
plt.show()
```




```
#Scale the data between 0 and 1 using unit scaling by dividing it by 255

X_valid, X_train = X_train_full[:5000]/255., X_train_full[5000:]/255.
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]


# Scale the test data also  dividing it by 255

X_test  =  X_test/255.


print(X_valid.shape)
print(X_train.shape)
print(y_valid.shape)
print(y_train.shape)
print(X_test.shape)
```

```
(5000, 28, 28)
(55000, 28, 28)
(5000,)
(55000,)
(10000, 28, 28)
```

```
# Let us build the ANN Image classification model. Step 1 : Assigning the parameters for forw

layers = [tf.keras.layers.Flatten(input_shape = [28,28], name = "inputlayer"),
          tf.keras.layers.Dense(300, activation = "relu", name = "hiddenlayer1"),
          tf.keras.layers.Dense(100, activation = "relu", name = "hiddenlayer2"),
          tf.keras.layers.Dense(10, activation = "softmax", name = "outputlayer")]



# Instantiating the Sequential model and parsing the parameters variable as declared above
```

```
model_clf = tf.keras.models.Sequential(layers)
```

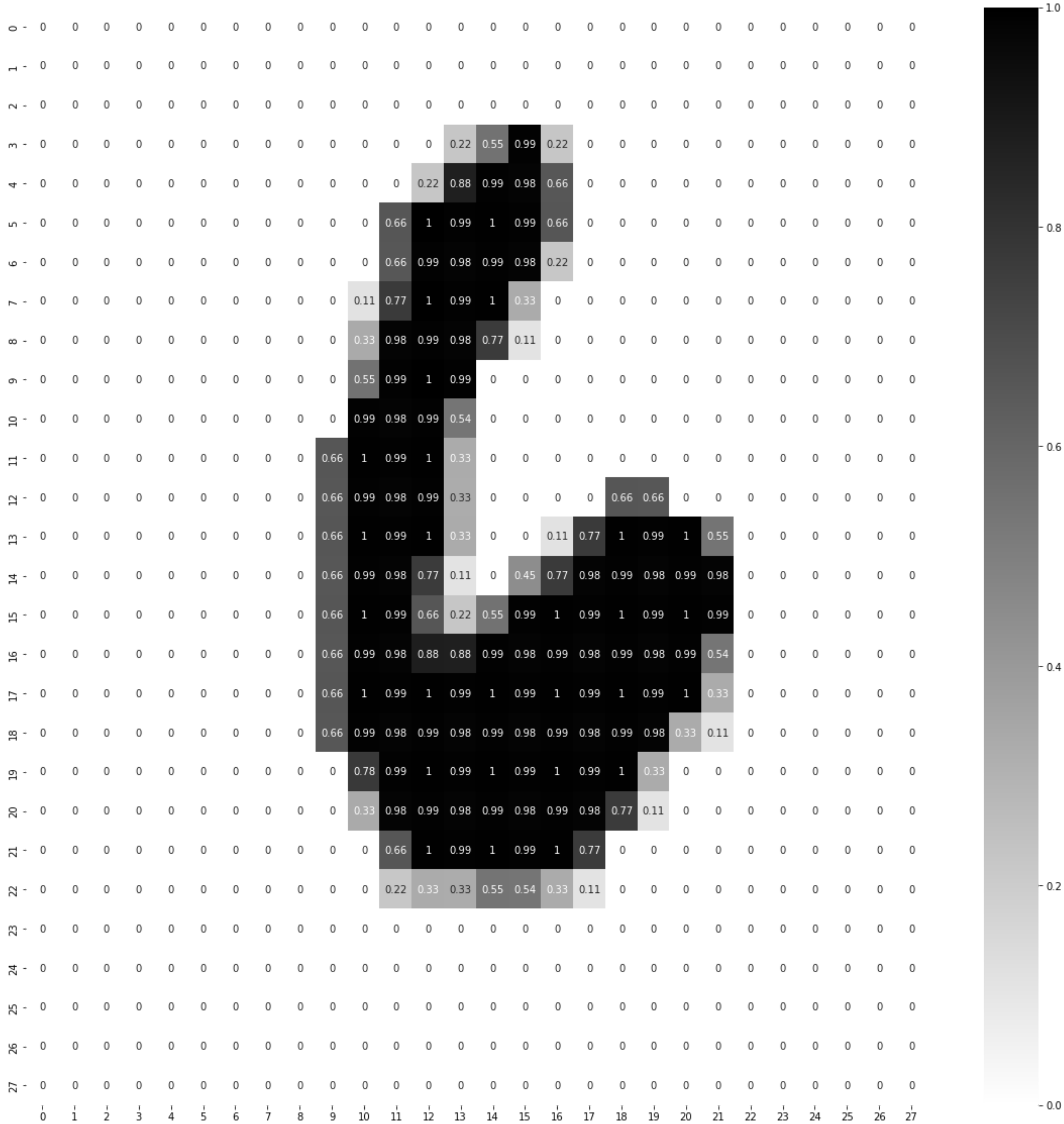
```
# Finding the summary of the model

model_clf.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inputlayer (Flatten)        (None, 784)               0

 hiddenlayer1 (Dense)        (None, 300)               235500

 hiddenlayer2 (Dense)        (None, 100)               30100

 outputlayer (Dense)         (None, 10)                1010

=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
_________________________________________________________________
```

```
# Assigning the parameters for back propogation to reduce the loss

LOSS_FUNCTION = "sparse_categorical_crossentropy"
OPTIMIZER = "ADAM"
METRICS = "accuracy"
```

```
# Compiling the model and parsing the assigned parameters for back propogation

model_clf.compile(loss = LOSS_FUNCTION, optimizer = OPTIMIZER, metrics =METRICS )
```

```
# To zoom the image, let us plot the heatmap. Image is collection of pixels between 0 to 256 

plt.figure(figsize = (20, 20))
sns.heatmap(X_train[6000], annot = True, cmap = "binary")
plt.show()
```


1.0
0.8
0.6
0.4
0.2
0.0

```
# To find the weights and bias assigned to this model

model_clf.layers[1].name
```

```
'hiddenlayer1'
```

```
# To find the weights and bias assigned to this model

model_clf.layers[1]
```

```
<keras.layers.core.dense.Dense at 0x7f2064e31ca0>
```

```
# To find the weights and bias assigned to this model

hidden1 = model_clf.layers[1]


# To find the weights and bias assigned to this model using get_weights() function

weights,biases = hidden1.get_weights()
```

```
#·Train·the·model.·Assign·the·parameters·for·running·epochs,·validation_set.·Fit·the·model·an

EPOCHS·=·5
VALIDATION_SET·=·(X_valid,·y_valid)
history·=·model_clf.fit(X_train,y_train,·epochs·=·EPOCHS,·validation_data·=·VALIDATION_SET)
```

```
Epoch 1/5
1719/1719 [==============================] - 8s 3ms/step - loss: 0.2062 - accuracy: 0.93
Epoch 2/5
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0854 - accuracy: 0.97
Epoch 3/5
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0595 - accuracy: 0.98
Epoch 4/5
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0436 - accuracy: 0.98
Epoch 5/5
1719/1719 [==============================] - 9s 5ms/step - loss: 0.0345 - accuracy: 0.98
```

```
# Function to Save the model

import time
import os

def SavedModel_Path(model_dir = "/content/Handwritten_image_classification_model/"):
  os.makedirs(model_dir, exist_ok= True)
  filename = time.strftime("Model_%y_%m_%D_%H_%M_%S_.h5")
  model_path = os.path.join(model_dir, filename )
  print(f"your model will be saved at the following location\n{model_path}")
  return model_path
```

```
#·Calling·the·function·to·the·save·the·model

model_clf.save(SavedModel_Path())
```

```
your model will be saved at the following location
/content/Handwritten_image_classification_model/Model_23_01_01/30/23_07_44_17_.h5
```

```
history.params
```

```
{'verbose': 1, 'epochs': 5, 'steps': 1719}
```

```
history.history
```

```
{'loss': [0.20617744326591492,
  0.0853799432516098,
  0.05947514995932579,
  0.04361899569630623,
  0.03446190804243088],
 'accuracy': [0.9383999705314636,
  0.973800003528595,
  0.9810000061988831,
  0.985836386680603,
  0.9887818098068237],
 'val_loss': [0.0920095443725586,
  0.09643342345952988,
  0.07115127891302109,
  0.07957673072814941,
  0.08478434383869171],
 'val_accuracy': [0.97079998254776,
  0.9710000157356262,
  0.9783999919891357,
  0.9787999987602234,
  0.9769999980926514]}
```

```
#·Finding·the·history·of·the·model·and·saving·it·in·a·DataFrame

pd.DataFrame(history.history)
```

	loss	accuracy	val_loss	val_accuracy
0	0.206177	0.938400	0.092010	0.9708
1	0.085380	0.973800	0.096433	0.9710
2	0.059475	0.981000	0.071151	0.9784
3	0.043619	0.985836	0.079577	0.9788
4	0.034462	0.988782	0.084784	0.9770

```
#·Let·us·find·the·accuracy·for·the·blind·dataset·which·is·X_test·and·y_test·dataset
```

```
model_clf.evaluate(X_test,·y_test)
```

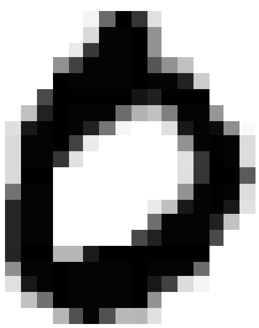
```
313/313 [==============================] - 1s 3ms/step - loss: 0.0802 - accuracy: 0.9772
[0.08019960671663284, 0.9771999716758728]
```

```
# To find the single image from the blind dataset called X_test
X_test[3]
```

```
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.85098039, 0.98431373, 0.98431373,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.71372549, 0.86666667, 0.98431373, 0.98431373, 0.98431373,
         0.70588235, 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.85490196, 0.99215686, 0.99215686,
         0.28627451, 0.28627451, 0.89411765, 0.99215686, 0.99215686,
         1.        , 0.99215686, 0.99215686, 0.99215686, 0.99215686,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.44313725, 0.98431373, 0.98431373,
         0.99215686, 0.98431373, 0.98431373, 0.98431373, 0.98431373,
         0.99215686, 0.98431373, 0.98431373, 0.98431373, 0.57647059,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.12156863, 0.90196078, 0.98431373,
         0.99215686, 0.98431373, 0.98431373, 0.98431373, 0.98431373,
         0.99215686, 0.90196078, 0.74117647, 0.1372549 , 0.03921569,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.24313725, 0.55686275,
         0.99215686, 0.98431373, 0.98431373, 0.98431373, 0.98431373,
         0.99215686, 0.41960784, 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.28235294, 0.68235294, 0.98431373, 0.67843137, 0.27843137,
         0.28235294, 0.11764706, 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
```

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

```
# Now, let us plot to see the image

plt.imshow(X_test[3], cmap = "binary")
plt.axis("off")
plt.show()
```

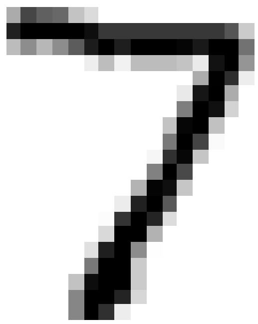


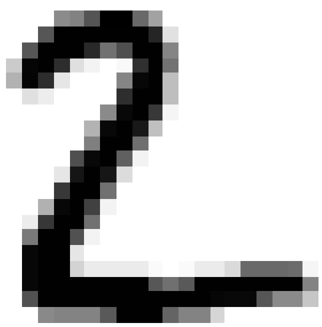
```
# Now, let us find the prediction

model_clf.predict(X_test[:3])
```

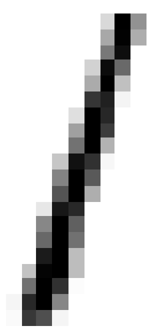
```
1/1 [==============================] - 0s 24ms/step
array([[1.83626656e-10, 5.37727196e-08, 5.14762746e-07, 2.84614562e-05,
        3.83316079e-12, 2.02787561e-07, 6.60341356e-13, 9.99970555e-01,
        1.03015402e-07, 3.64948853e-08],
       [2.52475350e-11, 3.83083687e-07, 9.99999642e-01, 1.34662965e-08,
        2.32179820e-15, 1.43215551e-10, 6.09612100e-12, 7.63226114e-12,
        2.31220665e-09, 1.37647026e-15],
       [2.15630379e-07, 9.99894023e-01, 6.40114592e-07, 7.75008779e-10,
        8.82117638e-06, 7.76971820e-09, 6.15818237e-07, 9.51907859e-05,
        4.74777693e-07, 5.43441847e-10]], dtype=float32)
```

```
# Now, let us plot to see the multiple images

for i in range(0,3):
```

```
    plt.imshow(X_test[i], cmap = "binary")
    plt.axis("off")
    plt.show()
```







```
# Now, let us find the prediction

prediction = model_clf.predict(X_test[:3])
```

```
1/1 [==============================] - 0s 18ms/step
```

```
prediction[0]
```

```
array([1.8362666e-10, 5.3772720e-08, 5.1476275e-07, 2.8461456e-05,
       3.8331608e-12, 2.0278756e-07, 6.6034136e-13, 9.9997056e-01,
       1.0301540e-07, 3.6494885e-08], dtype=float32)
```

```
# Now, lets find the maximum probablity value from the prediction

y_prob = prediction[0]
```

```
# We need to take the highest probablity value in this case is 1 and the position of 1 is 7.

y_prob.round(3)
```

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

```
# We can find the maximum probality value from the array using argmax function

np.argmax(y_prob.round(3))
```

```
7
```

```
# To load the model

from tensorflow import keras
```

```
#·Load·your·trained·model
model·=·keras.models.load_model('/content/Handwritten_image_classification_model/Model_23_01_
```

```
model
```

```
<keras.engine.sequential.Sequential at 0x7f1fd538ac10>
```

```
model.predict(X_test[:3])
```

```
1/1 [==============================] - 0s 14ms/step
array([[1.83626656e-10, 5.37727196e-08, 5.14762746e-07, 2.84614562e-05,
        3.83316079e-12, 2.02787561e-07, 6.60341356e-13, 9.99970555e-01,
        1.03015402e-07, 3.64948853e-08],
       [2.52475350e-11, 3.83083687e-07, 9.99999642e-01, 1.34662965e-08,
        2.32179820e-15, 1.43215551e-10, 6.09612100e-12, 7.63226114e-12,
        2.31220665e-09, 1.37647026e-15],
       [2.15630379e-07, 9.99894023e-01, 6.40114592e-07, 7.75008779e-10,
        8.82117638e-06, 7.76971820e-09, 6.15818237e-07, 9.51907859e-05,
        4.74777693e-07, 5.43441847e-10]], dtype=float32)
```

✓ 0s completed at 1:43 PM