```
Handwritten Digit Recognition Using MNIST Dataset With The Help Of Neural Network.
Dataset Used : MNIST Dataset
About Dataset :
MNIST is a commonly used dataset in machine learning and computer vision research, which consists of a set of 70,000 images of
handwritten digits (0-9), each of size 28x28 pixels. The dataset is split into two sets: a training set of 60,000 images and a
test set of 10,000 images. The training set is used to train a machine learning model, while the test set is used to evaluate the model's
```

```
#Importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from numpy import unique , argmax
```

+ Code + Text

```
#TensorFlow already contain MNIST data set which can be loaded using Keras
import tensorflow as tf # installing tenserflow
from tensorflow import keras
```

```
#To Load the MNIST dataset from the Keras API provided by TensorFlow.
mnist = tf.keras.datasets.mnist
```

```
The Above Code Reflects that the Dataset Contains :

An array of 60,000 images, each represented as a 28x28 NumPy array, with pixel values ranging from 0 to 255.
An array of 60,000 labels, each representing the correct digit (0-9) for the 1.
An array of 10,000 images, each represented as a 28x28 NumPy array, with pixel values ranging from 0 to 255.
An array of 10,000 labels, each representing the correct digit (0-9) for the 3.
```

```
(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
```

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

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

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

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

```
print(x_test)
```

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

 [[0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]
  ...
  [0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]]

 [[0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]
  ...
  [0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]]

 ...

 [[0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]
  ...
  [0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]]

 [[0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]
  ...
  [0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]]

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

```
#Reshaping the input Data which is used as a input in CNN in Tenserflow
#CNN takes the input Data in 4D Format with the shape (num_samples, image_height, image_width, num_channels)
#Here (num_channels) is set to 1 which means input image is Grayscale.

x_train = x_train.reshape((x_train.shape[0] , x_train.shape[1] , x_train.shape[2],1))
x_test = x_test.reshape((x_test.shape[0] , x_test.shape[1] , x_test.shape[2],1))
print(x_train.shape)
print(x_test.shape)
print(x_train.dtype)
print(x_test.dtype)
```

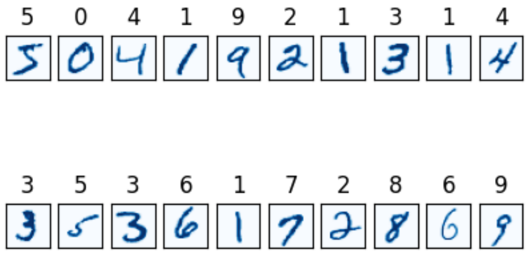
```
(60000, 28, 28, 1)
(10000, 28, 28, 1)
uint8
uint8
```

```
#Normalizing Pixel Values

x_train = x_train.astype('float32')/255.0
```

```
x_test = x_test.astype('float32')/255.0
print(x_train.dtype)
print(x_test.dtype)
```

```
float32
float32
```

```
#Visulaizing Subsets of images in MNIST Dataset along with coressponding labels.

fig=plt.figure(figsize=(5,3))
for i in range(20):
    ax =fig.add_subplot(2,10,i+1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(x_train[i]), cmap='Blues')
    ax.set_title(y_train[i])
```




```
#showing shape of single image
img_shape= x_train.shape[1:]
img_shape
```

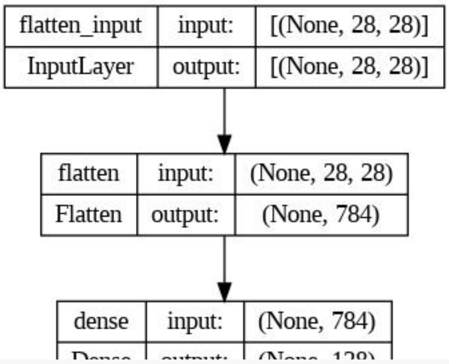
```
(28, 28, 1)
```

```
#BUILDING NEURAL NETWORK THAT CAN READ HANDWRITTEN DIGITS

#Creating aSequential Model in Keras
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10)
])
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten (Flatten)           (None, 784)               0         
                                                                 
 dense (Dense)               (None, 128)               100480    
                                                                 
 dropout (Dropout)           (None, 128)               0         
                                                                 
 dense_1 (Dense)             (None, 10)                1290      
                                                                 
=================================================================
Total params: 101770 (397.54 KB)
Trainable params: 101770 (397.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
#Displaying Neural Network Model
from tensorflow.keras.utils import plot_model
plot_model(model, 'model.jpg', show_shapes = True)
```




```
#Making Prediction on Model
prediction = model(x_train[:1]).numpy()
prediction
```

```
array([[ 0.54571486,  0.27524394,  0.31579307,  0.03075318, -0.01230033,
        -0.19809306,  0.3899559 ,  0.3062836 , -0.24998352, -0.6257482 ]],
      dtype=float32)
```

```
#Applying Softmax() Function to prediction array
#This convert an output vector of real numbers into a probability distribution over predicted classes
tf.nn.softmax(prediction).numpy()
```

```
array([[0.15135913, 0.11549006, 0.12026931, 0.09044063, 0.08662948,
        0.07194108, 0.1295279 , 0.11913104, 0.06830322, 0.04690819]],
      dtype=float32)
```

```
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
loss_fn(y_train[:1], prediction).numpy()
model.compile(optimizer='adam',loss=loss_fn,metrics=['accuracy'])
```

```
#Model fitting

#Training the Model
model.fit(x_train, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 7s 3ms/step - loss: 0.2963 - accuracy: 0.9136
Epoch 2/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1420 - accuracy: 0.9574
Epoch 3/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1079 - accuracy: 0.9669
Epoch 4/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0868 - accuracy: 0.9730
Epoch 5/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0757 - accuracy: 0.9759
<keras.src.callbacks.History at 0x7d00aa2db9d0>
```

```
#Evaluating the Model
model.evaluate(x_test, y_test, verbose=2)
```

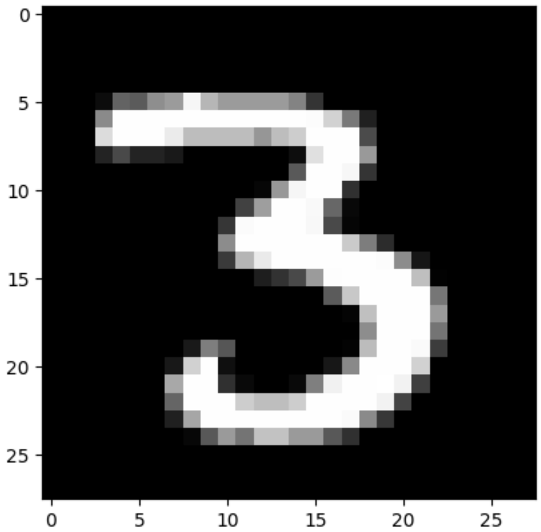
```
313/313 - 1s - loss: 0.0700 - accuracy: 0.9786 - 617ms/epoch - 2ms/step
[0.0699625238776207, 0.978600025177002]
```

```
#Creating a new sequential model which includes both previously trained model and softmax layer.
probability_model = tf.keras.Sequential([ model,tf.keras.layers.Softmax() ])
probability_model(x_test[:5])
```

```
<tf.Tensor: shape=(5, 10), dtype=float32, numpy=
array([[1.4491250e-09, 1.8083081e-09, 9.0282683e-06, 5.8593498e-05,
        1.7052123e-12, 2.4329015e-07, 4.4401718e-14, 9.9993181e-01,
        3.0611069e-08, 1.5394862e-07],
       [2.9584570e-09, 1.1868534e-03, 9.9881196e-01, 7.9176419e-07,
        1.6404112e-15, 1.7283705e-07, 2.0730835e-09, 7.0644107e-12,
        2.5894522e-07, 1.8599996e-15],
       [4.1466681e-08, 9.9966753e-01, 8.9438603e-05, 1.8448593e-06,
        4.9796054e-06, 6.9808234e-06, 2.1137846e-06, 8.6289961e-05,
        1.4021866e-04, 5.3885850e-07],
       [9.9986386e-01, 2.7712185e-10, 9.4905830e-05, 3.1549497e-08,
        2.5994165e-08, 3.4995121e-06, 3.1233110e-05, 4.1635089e-06,
        1.5309158e-08, 2.2872423e-06],
       [6.5349354e-07, 1.0509485e-08, 7.1089548e-06, 1.6578970e-07,
        9.9408686e-01, 5.0717163e-06, 2.4466556e-06, 1.2649639e-04,
        1.1078427e-05, 5.7599703e-03]], dtype=float32)>
```

```
#Displaying a Grayscale Image
img = x_train[12]
```

```
plt.imshow(np.squeeze(img) ,cmap='gray')
plt.show()
```




```
#Predicting the Result
img= img.reshape(1, img.shape[0],img.shape[1],img.shape[2])
p= model.predict([img])
print("predicted : {}".format(argmax(p)))
```

```
1/1 [==============================] - 0s 84ms/step
predicted : 3
```