```
# Import the necessary packages


# Importing necessary Libraries
import tensorflow as tf
from tensorflow import keras


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
from sklearn.metrics import accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist
tf.keras.layers.serialize
tf.keras.utils.CustomObjectScope
tf.keras.utils.register_keras_serializable
```

```
<function keras.utils.generic_utils.register_keras_serializable(package='Custom',
name=None)>
```

+ Code + Text

```
# Load the training and testing data MNIST


# Import dataset & split into train and test data
mnist=tf.keras.datasets.mnist
(x_train,y_train),(x_test,y_test)=mnist.load_data()


# Length of the training dataset
len(x_train)
len(y_train)
```

```
60000
```

```
# Length of the testing dataset
len(x_test)
len(y_test)
```

```
10000
```

```
# Shape of the training dataset
x_train.shape
```

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

```
# Shape of the testing dataset
x_test.shape
```

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

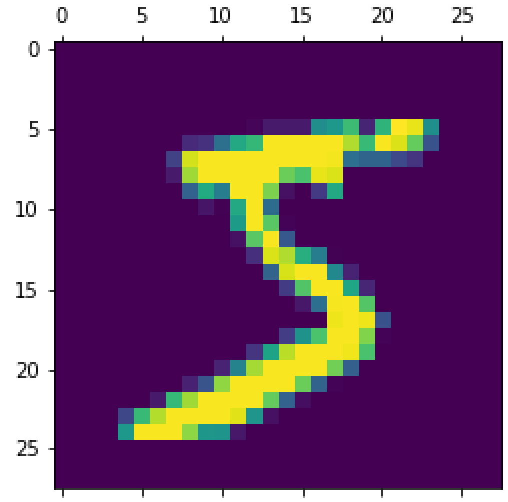
```
# See first Image Matrix
x_train[0]
```

```
          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, 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)
```

```
# See first image
plt.matshow(x_train[0])
```

```
    <matplotlib.image.AxesImage at 0x7fb2dbf72d90>
```



```
# Normalize the iamges by scaling pixel intensities to the range 0,1
x_train=x_train/255
x_test=x_test/255
```

```
# See first Naormalize Image Matrix
x_train[0]
```

```
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.18039216,
         0.50980392, 0.71764706, 0.99215686, 0.99215686, 0.81176471,
         0.00784314, 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.15294118, 0.58039216, 0.89803922,
         0.99215686, 0.99215686, 0.99215686, 0.98039216, 0.71372549,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.09411765, 0.44705882, 0.86666667, 0.99215686, 0.99215686,
         0.99215686, 0.99215686, 0.78823529, 0.30588235, 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.09019608, 0.25882353,
         0.83529412, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
         0.77647059, 0.31764706, 0.00784314, 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.07058824, 0.67058824, 0.85882353, 0.99215686,
         0.99215686, 0.99215686, 0.99215686, 0.76470588, 0.31372549,
         0.03529412, 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.21568627,
         0.6745098 , 0.88627451, 0.99215686, 0.99215686, 0.99215686,
         0.99215686, 0.95686275, 0.52156863, 0.04313725, 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.53333333,
         0.99215686, 0.99215686, 0.99215686, 0.83137255, 0.52941176,
         0.51764706, 0.0627451 , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
# Define the network architecture using Keras


model=keras.Sequential([
  # Input Layer
  keras.layers.Flatten(input_shape = (28,28)),
  # Hidden Layer
  keras.layers.Dense(128,activation ='relu'),
  # Output Layer
  keras.layers.Dense(10,activation = 'softmax')
])
```

```
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_5 (Flatten)         (None, 784)               0

 dense_10 (Dense)            (None, 128)               100480

 dense_11 (Dense)            (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

```
# Compile the Model
model.compile(loss='sparse_categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
```

```
#Train the model using SGD
```

```
history=model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.6548 - accuracy: 0.83
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3403 - accuracy: 0.90
Epoch 3/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2924 - accuracy: 0.91
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2623 - accuracy: 0.92
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2397 - accuracy: 0.93
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2208 - accuracy: 0.93
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2049 - accuracy: 0.94
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1911 - accuracy: 0.94
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1787 - accuracy: 0.95
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1678 - accuracy: 0.95
```

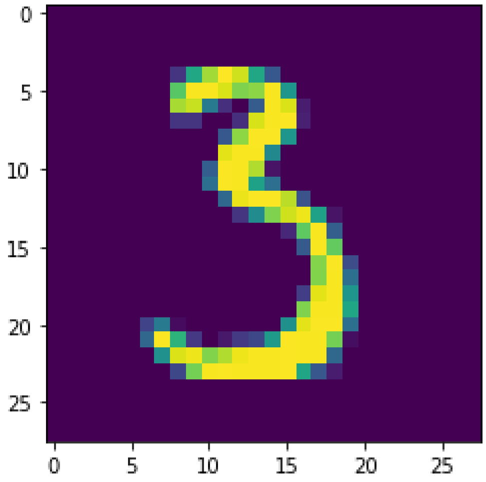
```
#Evaluate the network
```

```
test_loss,test_acc=model.evaluate(x_test,y_test)
print("Loss=%.3f" %test_loss)
print("Accuracy=%.3f" %test_acc)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.1640 - accuracy: 0.951
Loss=0.164
Accuracy=0.951
```

```
# Making Prediction on New Data
```

```
n=random.randint(0,9999)
plt.imshow(x_test[n])
plt.show()
```



```
predicted_value=model.predict(x_test)
print("Handwritten number is = %d" %np.argmax(predicted_value[n]))
```

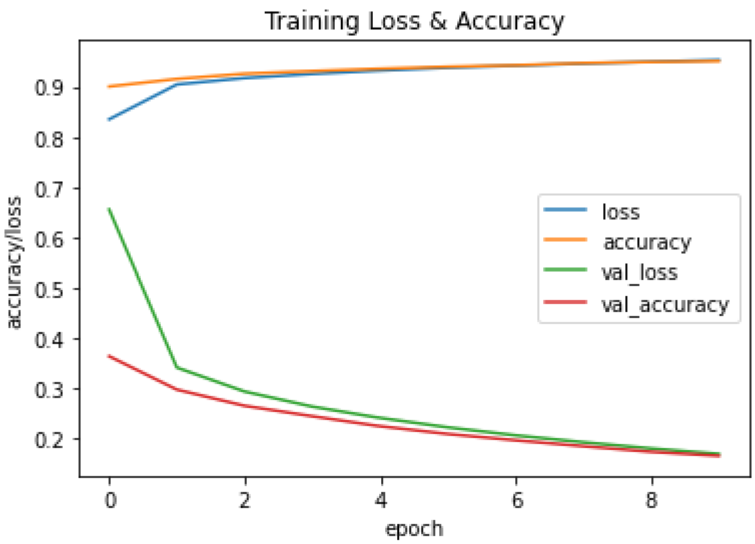
```
Handwritten number is = 3
```

```
# Plot the training loss and accuracy
```

```
history.history.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Training Loss & Accuracy')
plt.ylabel('accuracy/loss')
plt.xlabel('epoch')
plt.legend(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
plt.show()
```



✓ 0s completed at 12:32 PM