Step 1: Import necessary libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from keras.models import Sequential
from keras.datasets import mnist
import keras
```

Step 2: Laod the Dataset

```
(x_train,y_train),(x_test,y_test)=mnist.load_data()
#normalize the data (max of 0-255 ,so divide by 255)
x_train=x_train/255
x_test=x_test/255
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 1s 0us/step
```

Step 3: Define the network Architecture

```
model=Sequential() #Sequential as a feed forward network
model.add(keras.layers.Flatten(input_shape=(28,28)))  #Flatten the input(i.e. convett to 1-D array) and provide the input size to the input la
model.add(keras.layers.Dense(256,activation='relu'))  #Define the hidden layer with 128 nodes (<than 28x28) with Relu as activation function
model.add(keras.layers.Dense(10,activation='softmax'))#Define the output layer with 10 nodes (number of classes=10) with softmax activation fu
```

Step 4: Train the Model

```
model.compile(optimizer='sgd',loss='sparse_categorical_crossentropy',metrics=['accuracy']) #Compile the model using the SGD optimizer and meth
H=model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=10) #Fit the model for the training set and also providing the validation s
```

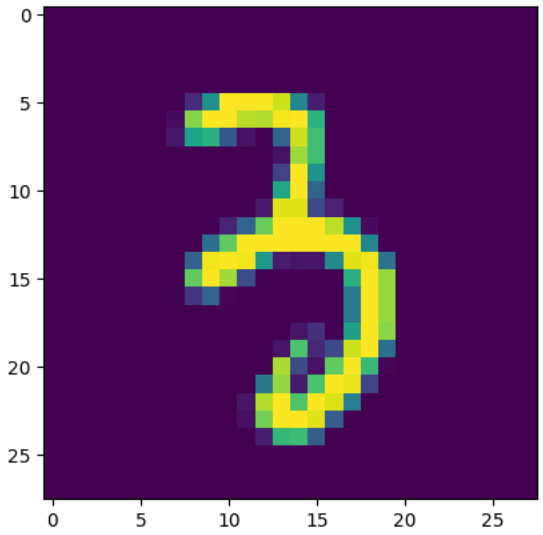
```
Epoch 1/10
1875/1875 [==============================] - 17s 5ms/step - loss: 0.6252 - accuracy: 0.8442 - val_loss: 0.3457 - val_accuracy: 0.9065
Epoch 2/10
1875/1875 [==============================] - 7s 3ms/step - loss: 0.3255 - accuracy: 0.9097 - val_loss: 0.2826 - val_accuracy: 0.9212
Epoch 3/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2765 - accuracy: 0.9224 - val_loss: 0.2504 - val_accuracy: 0.9309
Epoch 4/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2460 - accuracy: 0.9313 - val_loss: 0.2249 - val_accuracy: 0.9380
Epoch 5/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2233 - accuracy: 0.9378 - val_loss: 0.2109 - val_accuracy: 0.9424
Epoch 6/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2055 - accuracy: 0.9427 - val_loss: 0.1942 - val_accuracy: 0.9450
Epoch 7/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1900 - accuracy: 0.9470 - val_loss: 0.1823 - val_accuracy: 0.9497
Epoch 8/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1769 - accuracy: 0.9503 - val_loss: 0.1693 - val_accuracy: 0.9515
Epoch 9/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1655 - accuracy: 0.9536 - val_loss: 0.1609 - val_accuracy: 0.9544
Epoch 10/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1554 - accuracy: 0.9568 - val_loss: 0.1516 - val_accuracy: 0.9566
```

Step 5: Evaluate the model

```
test_loss,test_acc=model.evaluate(x_test,y_test) #evaluate the modelfor the test set
#Print the accuracy and the loss of the model
print("Loss=%.3f"%test_loss)
print("Accuracy=%.3f"%test_acc)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.1516 - accuracy: 0.9566
Loss=0.152
Accuracy=0.957
```

```
#Randomly choose any image from the test test and evaluate the predict of the model
import random
n=random.randint(0,9999) # choose random number between 0-9999
plt.imshow(x_test[n]) #display that image
plt.show()
pred=model.predict(x_test) #predict the output using the model
print("Digit is : " ,np.argmax(pred[n])) # print output
```

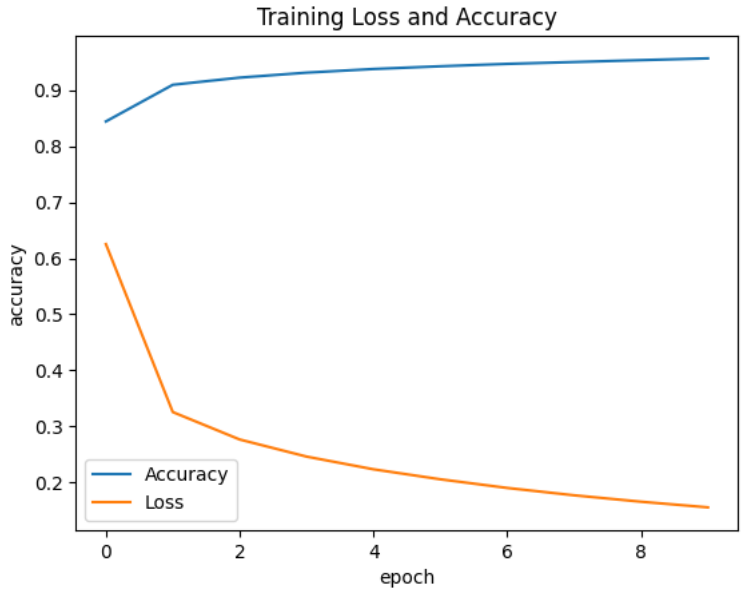


```
313/313 [==============================] - 1s 2ms/step
Digit is :  3
```

Step 6: Plot the accuracy and loss Graphs

```
plt.plot(H.history['accuracy']) #get the accuracy for the training set from the model's history attribute
plt.plot(H.history['loss'])   #get the loss for the training set from the model's history attribute
plt.title('Training Loss and Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Accuracy', 'Loss'])
```

```
<matplotlib.legend.Legend at 0x7f8d3c0b5630>
```




```
plt.plot(H.history['val_accuracy']) #get the accuracy for the validation set from the model's history attribute
plt.plot(H.history['val_loss']) #get the loss for the validation set from the model's history attribute
plt.title('Testing Loss and accuracy')
plt.ylabel('accuracy')
```

```
plt.xlabel('epoch')
plt.legend(['Accuracy', 'Loss'])
```

```
<matplotlib.legend.Legend at 0x7f8d3c17a110>
```