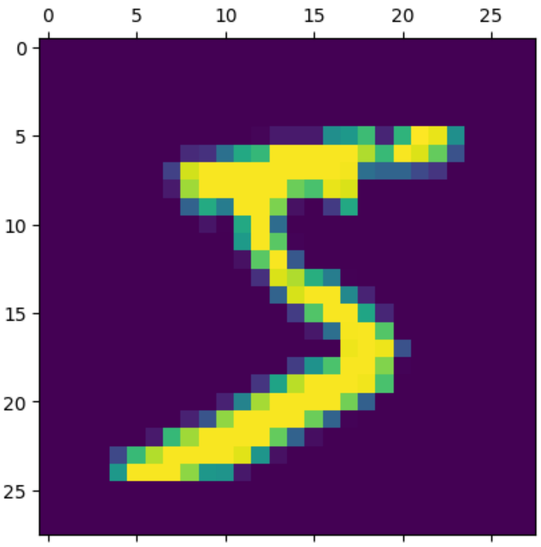
Problem Statement 2: To Implement Feed forward Neural Network with keras and tensorflow. Import the necessary packages Load the training and testing data(MNIST) Define the network architecture using keras Train the model using SGD Evaluate the network Plot the training loss and accuracy

```
#import 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
%matplotlib inline
```

```
#import dataset and split into train and test
mnist = tf.keras.datasets.mnist

(x_train,y_train), (x_test,y_test) = mnist.load_data()
```

```
plt.matshow(x_train[0])
```

```
<matplotlib.image.AxesImage at 0x7ab9a8dad4e0>
```




```
x_train = x_train/255
x_test = x_test/255
```

```
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.        , 0.01176471, 0.07058824, 0.07058824,
        0.07058824, 0.49411765, 0.53333333, 0.68627451, 0.10196078,
        0.65098039, 1.        , 0.96862745, 0.49803922, 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.11764706, 0.14117647,
        0.36862745, 0.60392157, 0.66666667, 0.99215686, 0.99215686,
        0.99215686, 0.99215686, 0.99215686, 0.88235294, 0.6745098 ,
        0.99215686, 0.94901961, 0.76470588, 0.25098039, 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.19215686, 0.93333333, 0.99215686,
        0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
        0.99215686, 0.99215686, 0.98431373, 0.36470588, 0.32156863,
        0.32156863, 0.21960784, 0.15294118, 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.07058824, 0.85882353, 0.99215686,
        0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.77647059,
        0.71372549, 0.96862745, 0.94509804, 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.31372549, 0.61176471,
        0.41960784, 0.99215686, 0.99215686, 0.80392157, 0.04313725,
        0.        , 0.16862745, 0.60392157, 0.        , 0.        ,
```

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

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_
  super().__init__(**kwargs)
```

```
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 784)	0
dense_3 (Dense)	(None, 128)	100,480
dense_4 (Dense)	(None, 10)	1,290

Total params: 101,770 (397.54 KB)
Trainable params: 101,770 (397.54 KB)
Non-trainable params: 0 (0.00 B)

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

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

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.7348 - loss: 1.0318 - val_accuracy: 0.9040 - val_loss: 0.3550
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9011 - loss: 0.3529 - val_accuracy: 0.9196 - val_loss: 0.2932
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9157 - loss: 0.3000 - val_accuracy: 0.9264 - val_loss: 0.2627
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9236 - loss: 0.2731 - val_accuracy: 0.9319 - val_loss: 0.2420
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9315 - loss: 0.2419 - val_accuracy: 0.9370 - val_loss: 0.2235
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9375 - loss: 0.2218 - val_accuracy: 0.9415 - val_loss: 0.2053
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.9424 - loss: 0.2030 - val_accuracy: 0.9447 - val_loss: 0.1937
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9453 - loss: 0.1895 - val_accuracy: 0.9482 - val_loss: 0.1813
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9493 - loss: 0.1787 - val_accuracy: 0.9521 - val_loss: 0.1705
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.9538 - loss: 0.1672 - val_accuracy: 0.9545 - val_loss: 0.1630
```

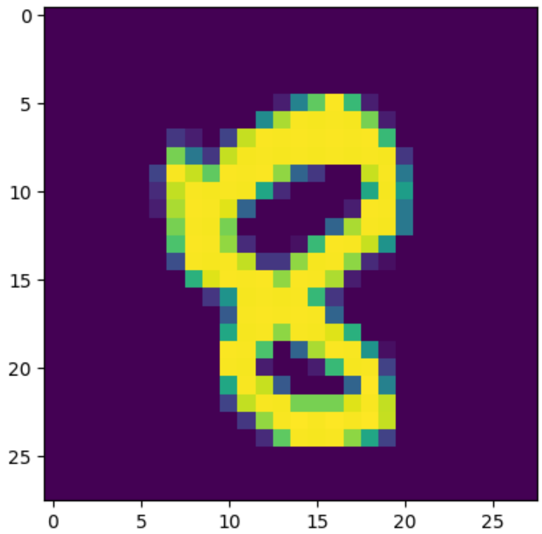
```
test_loss, test_acc=model.evaluate(x_test,y_test)

print("Loss-%.3f" %test_loss)

print("Accuracy=%.3f" %test_acc)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9455 - loss: 0.1924
Loss-0.163
Accuracy=0.955
```

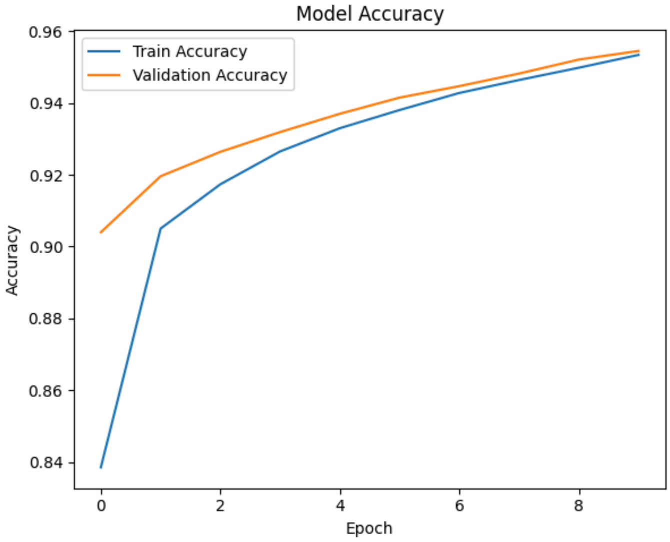
```
n=random.randint(0,9999)
plt.imshow(x_test[n])
plt.show
```

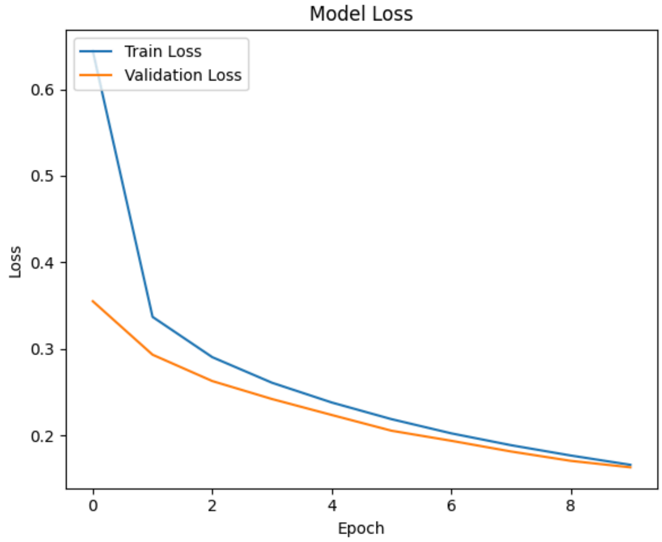
```
matplotlib.pyplot.show
def show(*args, **kwargs)
```

```
**Auto-show in jupyter notebooks**

The jupyter backends (activated via ``%matplotlib inline``,
``%matplotlib notebook``, or ``%matplotlib widget``), call ``show()`` at
the end of every cell by default. Thus, you usually don't have to call it
explicitly there.
```




```
import matplotlib.pyplot as plt

# Function to plot the training and validation accuracy/loss
def plot_accuracy_loss(history):
    # Set figure size
    plt.figure(figsize=(12, 5))

    # Plot training & validation accuracy values
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Model Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(loc='upper left')

    # Plot training & validation loss values
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Model Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(loc='upper left')

    # Show the plot
    plt.tight_layout()
    plt.show()

# Assuming 'history' object is available from model.fit()
plot_accuracy_loss(history)
```






```
test_predict = model.predict(x_test)

# Get the classification labels

test_predict_labels =np.argmax(test_predict, axis=1)

confusion_matrix = tf.math.confusion_matrix(labels=y_test, predictions=test_predict_labels)

print("Confusion matrix of the test set:\n", confusion_matrix)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Confusion matrix of the test set:
 tf.Tensor(
[[ 966    0    1    1    0    4    6    1    1    0]
 [    0 1117    2    2    0    1    3    2    8    0]
 [    6    2  984   10    4    2    5    7    8    4]
 [    0    0    7  968    1   10    0    9   11    4]
 [    1    0    6    0  940    0    7    3    4   21]
 [    8    1    0   19    4  832   10    2   10    6]
 [   10    3    2    1    7    9  922    1    3    0]
 [    0    8   18    5    4    2    0  969    2   20]
 [    7    2    2   17    7   10    9    8  905    7]
 [    9    7    1   10   25    5    1    7    2  942]], shape=(10, 10), dtype=int32)
```