2. Implementing Feedforward neural networks with Keras and TensorFlow a. Import the necessary packages b. Load the training and testing data (MNIST/CIFAR10) c. Define the network architecture using Keras d. Train the model using SGD e. Evaluate the network f. Plot the training loss and accuracy

```
#a. Import the necessary packages
#imoporting 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
```

```
#b. Load the training and testing data (MNIST/CIFAR10)
#import dataset and split into train and test data
mnist = tf.keras.datasets.mnist
(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
```

```
#c. Define the network architecture using Keras
#to see length of training dataset
len(x_train)
```

```
60000
```

```
##to see length of testing dataset
len(x_test)
```

```
10000
```

```
#d. Train the model using SGD
#shape of training dataset  60,000 images having 28*28 size
x_train.shape
```

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

```
#shape of training dataset  60,000 images having 28*28 size
x_train.shape
```

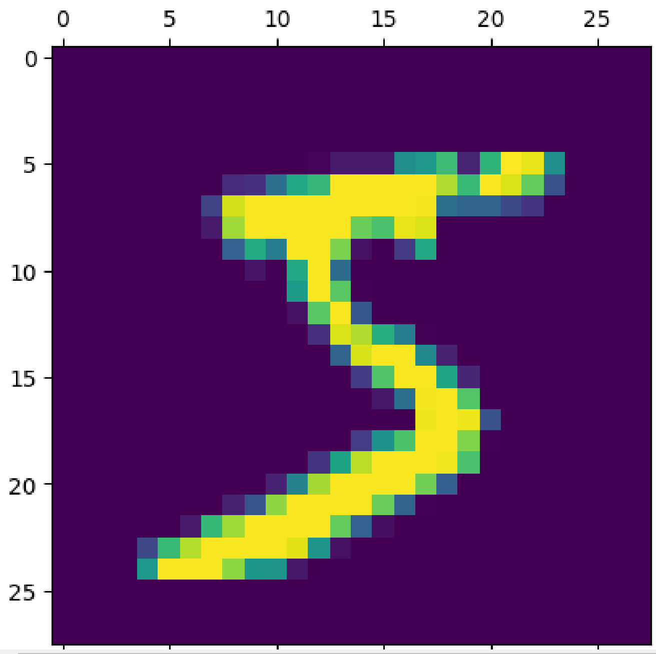
```
(60000, 28, 28)
```

```
x_train[10]
```

ndarray (28, 28) show data

```
#to see how first image look
plt.matshow(x_train[0])
```

```
<matplotlib.image.AxesImage at 0x7d09206ac2b0>
```




```
#normalize the images by scaling pixel intensities to the range 0,1

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.        ,
```

```
#e. Evaluate the network
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 a
  super().__init__(**kwargs)
```

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 128)	100,480
dense_1 (Dense)	(None, 10)	1,290

Total params: 101,770 (397.54 KB)
Trainable params: 101,770 (397.54 KB)

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

Train the model

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

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.7264 - loss: 1.0487 - val_accuracy: 0.9062 - val_lo
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9039 - loss: 0.3459 - val_accuracy: 0.9211 - val_lo
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9194 - loss: 0.2848 - val_accuracy: 0.9299 - val_lo
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - accuracy: 0.9276 - loss: 0.2566 - val_accuracy: 0.9345 - val_lo
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9343 - loss: 0.2357 - val_accuracy: 0.9393 - val_lo
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9412 - loss: 0.2115 - val_accuracy: 0.9431 - val_lo
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9460 - loss: 0.1929 - val_accuracy: 0.9478 - val_lo
```

```
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9478 - loss: 0.1861 - val_accuracy: 0.9500 - val_lo
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9511 - loss: 0.1718 - val_accuracy: 0.9519 - val_lo
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.9541 - loss: 0.1654 - val_accuracy: 0.9536 - val_l
```

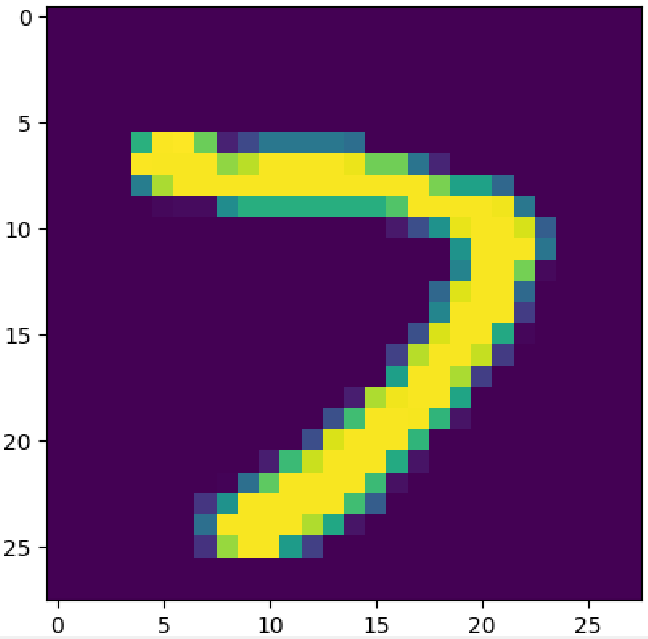
Evaluate the model

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

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9462 - loss: 0.1847
Loss=0.159
Accuracy=0.954
```

Making Prediction on New Data

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




```
#f. Plot the training loss and accuracy
#we use predict() on new data
predicted_value=model.predict(x_test)
print("Handwritten number in the image is= %d" %np.argmax(predicted_value[n]))
```

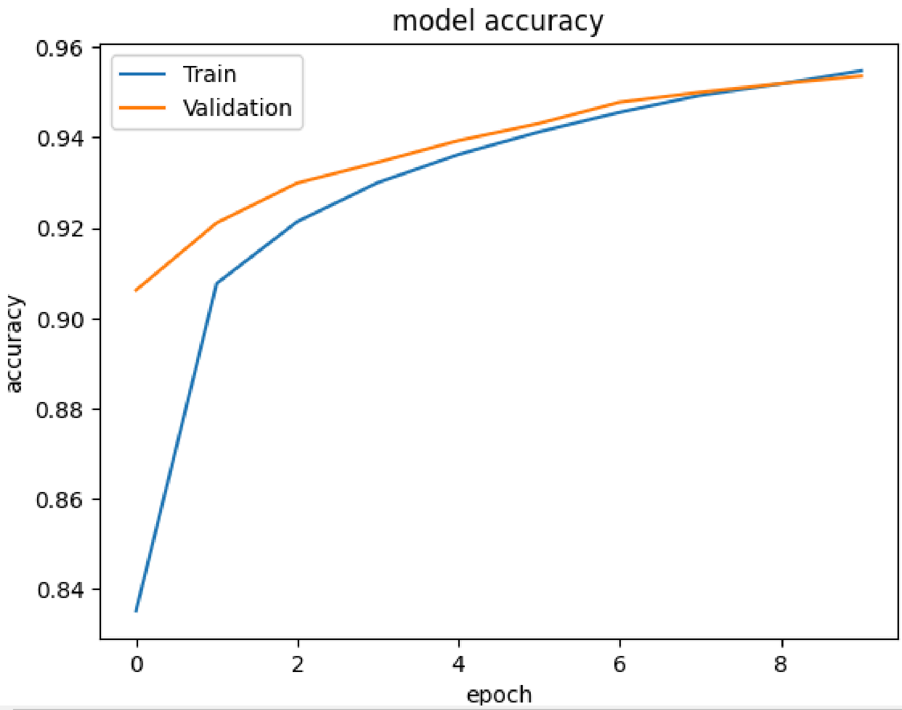
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Handwritten number in the image is= 7
```

Plot graph for Accuracy and Loss

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

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

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

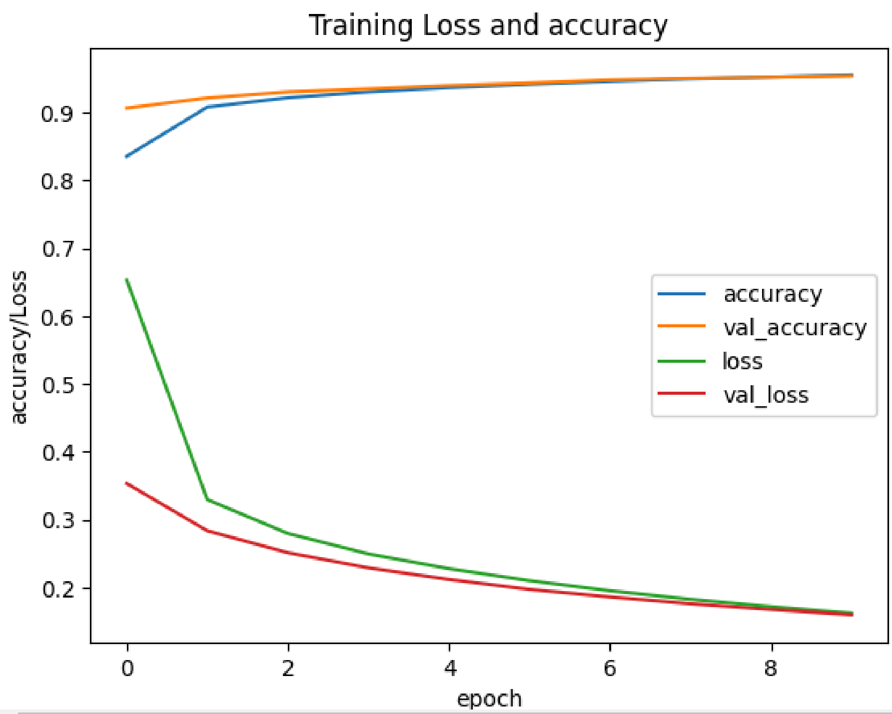


graph representing the model’s accuracy

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

graph represents the model’s loss

```
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 and accuracy')
plt.ylabel('accuracy/Loss')
plt.xlabel('epoch')
plt.legend(['accuracy', 'val_accuracy','loss','val_loss'])
plt.show()
```



Start coding or generate with AI.