# ICP8

## Answer1)

**Program Description**:  A program to get accuracy and loss based on basic sequential Neural Network Model. Then comparing accuracy and loss with adding more dense layer to that model.

**Code Description:**
Start with setting up environment and importing necessary libraries.
**Keras**: Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. In this Sequential model is a linear stack of layers.

```
import warnings
warnings.filterwarnings('ignore')
# Imported the necessary libraries and created our environment
# Keras is a high-level library.
# Sequential model is a linear stack of layers.
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from sklearn.model_selection import train_test_split
# # Importing pandas library as we will be doing dataframe manipulation,
import pandas as pd
import numpy as np
# matplotlib is a plotting library which we will use for our graph
import matplotlib.pyplot as plt
```

Next is importing ‘diabetes.csv’ file into dataframe and then creating training and testing datasets.
By analyzing input we can get that we have 8 useful features.

```
# Next is importing ‘diabetes.csv’ file into dataframe.
# We are using pd.read_csv() method of Pandas for that. It creates a Dataframe from a csv file.
dataset = pd.read_csv("diabetes.csv", header=None).values
# Printing the Shape of dataset
print('Shape of Dataset: ', dataset.shape, '\n')

# Splitting the data into test and train
# In this test_size = o.25 means 25% od data for testing and 75% of data for training
X_train, X_test, Y_train, Y_test = train_test_split(dataset[:,0:8], dataset[:,8],
                                                    test_size=0.25, random_state=87)
```

Using np.random.seed function that sets the random seed to 155.

```
np.random.seed(155)
```

For comparison purpose we will create two separate models.
The first model (first_nn) we are creating is just the basic sequential Neural Network with input and output layers. We are using **Dense Layer** which is one of the type of layer and all of its nodes/neurons are fully connected.
For defining first i.e. input layer we are mentioning 20 hidden units are used by model.
Model needs to know how many input features or shape of input to expect so we are using parameter **input_dim= 8**.
Next thing we are mentioning is for model to use **'relu'** activation.

```
# creating first Model (first_nn)
# First Model: It has 8 input size and dense layer of 20 units and activation= relu
# Code which was provided in the class
first_nn = Sequential()
# hidden layer
first_nn.add(Dense(20, input_dim=8, activation='relu'))
```

Next we will define output layer.
We have defined 1 hidden unit means our model will have single output only. For output layer we are using activation **'sigmoid'.**

```
# output layer
# We took 1 here because it is a binary classification
first_nn.add(Dense(1, activation='sigmoid'))
```

Let's compile our model. In compiling we are configuring the model with **adam** optimizer and **binary_crossentropy** loss function. To monitor the accuracy we are passing ['accuracy'] to metrics argument.

```
# Compiling the first model
first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
```

Lets fit the model with train data. We are training model over 100 epochs (iteration) over all the samples of X_train and Y_Train with default batches of sample train data.

```
# fitting the model with train data
first_nn_fitted = first_nn.fit(X_train, Y_train, epochs=100,
                               initial_epoch=0)
```

Final action on this model is to get summary and evaluate it.

```
# Printing the Summary of first Model
print(first_nn.summary())
# Evaluating the first Model
print(first_nn.evaluate(X_test, Y_test))

# Evaluating testing data on first Model
first_evl = first_nn.evaluate(X_test, Y_test)

print('Epoch for first model ends here************************************************************************', '\n')
```

**OUTPUT:**

```
 32/576 [>.............................] - ETA: 0s - loss: 0.5864 - acc: 0.7188
576/576 [==============================] - 0s 35us/step - loss: 0.5435 - acc: 0.7396
Epoch 100/100

 32/576 [>.............................] - ETA: 0s - loss: 0.4768 - acc: 0.7812
576/576 [==============================] - 0s 35us/step - loss: 0.5396 - acc: 0.7413
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 20)                180
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 21
=================================================================
Total params: 201
Trainable params: 201
Non-trainable params: 0
_________________________________________________________________
None
```

**ADDING DENSE LAYERS**

----------------------------------------

Let’s create second model now. Here input layer will be identical to our first model.

```
# Starting 2nd model ------------------------------------------------------------------

# creating second model(second_nn)
# Second Model: In this we are adding dense layers(30, 45, 50) and activation= relu
# To check the changes in accuracy and loss.
second_nn = Sequential()
# hidden layer ( It has 8 input size and dense layer of 20 units)
second_nn.add(Dense(20, input_dim=8, activation='relu'))
```

The main thing that we are doing difference here is we will be adding multiple hidden layers here for model to give more flexibility for data processing and training.
We are adding total three hidden layers with 30, 45 and 50 hidden dense units respectively.

```
# Adding additional dense layer of 30 units
second_nn.add(Dense(30));
# Adding additional dense layer of 45 units
second_nn.add(Dense(45));
# Adding additional dense layer of 50 units
second_nn.add(Dense(50));
```

Output layer is also same as first one.

```
# output layer
# We took 1 here because it is a binary classification
second_nn.add(Dense(1, activation='sigmoid'))
```

Let’s, compile and fit the model.

```
# Compiling the second Model
second_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
# fitting the data
second_nn_fitted = second_nn.fit(X_train, Y_train, epochs=100,
                                 initial_epoch=0)
```

Now next step for this model is to evaluate it.

```
# Evaluating testing data on Second Model
second_evl = first_nn.evaluate(X_test, Y_test)

print('*************Second model statistics********************************************************',' \n')
# Printing the summary Second Model
print(second_nn.summary())
# Printing the Evaluation of Second Model
print(second_nn.evaluate(X_test, Y_test))
```

Plotting a graph of first model and second model to compare the accuracy and loss.

Plotting history to check the accuracy.

```
# Plotting history for accuracy
# first_nn_fitted.history for accuracy
plt.plot(first_nn_fitted.history['acc'])
plt.plot(second_nn_fitted.history['acc'])
# printing title
plt.title("Accuracy on training data")
# y label as accuracy
plt.ylabel('Accuracy')
# x label as epoch
plt.xlabel('Epoch')
# Placing a legend on Model1 and Model2
plt.legend(['Model1', 'Model2'], loc= 'upper left')
# To show the graph
plt.show()
```

Plotting history to check the loss.

```python
# Plotting History for loss
# first_nn_fitted.history for loss
loss = first_nn_fitted.history['loss']
plt.plot(first_nn_fitted.history['loss'])
plt.plot(second_nn_fitted.history['loss'])
# Printing Title
plt.title('Loss on training data')
# y label as loss
plt.ylabel('Loss')
# x label as epoch
plt.xlabel('Epoch')
# Placing a legend on Model1 and Model2
plt.legend(['Model1', 'Model2'], loc= 'upper left')
# To show the graph
plt.show()
```

**OUTPUT:**

```
 32/576 [>.............................] - ETA: 0s - loss: 0.4109 - acc: 0.8125
576/576 [==============================] - 0s 55us/step - loss: 0.5185 - acc: 0.7569

 32/192 [====>.........................] - ETA: 0s
192/192 [==============================] - 0s 19us/step
*************Second model statistics*********************************************************

Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 20)                180
_________________________________________________________________
dense_4 (Dense)              (None, 30)                630
_________________________________________________________________
dense_5 (Dense)              (None, 45)                1395
_________________________________________________________________
dense_6 (Dense)              (None, 50)                2300
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 51
=================================================================
Total params: 4,556
Trainable params: 4,556
Non-trainable params: 0
_________________________________________________________________
None
```

**COMPARISION**

-----------------------

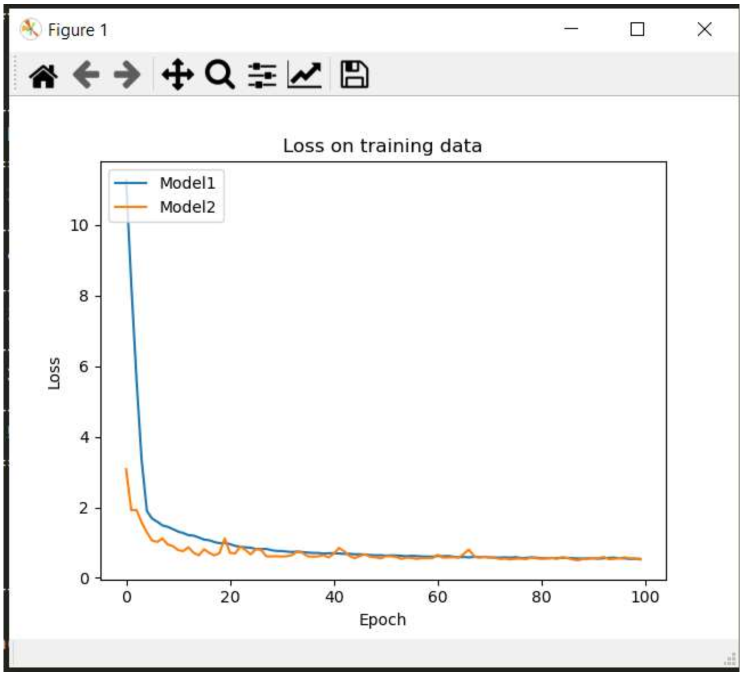
Lets compare accuracy and loss of both the models.

```
# Compare accuracy and loss of both the models
print('************************************************************************************************','\n')
print('First models Accuracy: ',first_evl[1],'Second models Accuracy: ',second_evl[1])
print('First models Loss: ',first_evl[0],'Second models Loss: ',second_evl[0])
```

OUTPUT:

```
First models Accuracy:  0.6614583134651184 Second models Accuracy:  0.6875
First models Loss:  0.6456884145736694 Second models Loss:  0.5734400103489558
```

Graphical comparison of Accuracy and loss of both the models.




**CONCLUSION:**

**------------------------**

**As we can see from evaluation values of both the models and above graph, accuracy of second model is greater while loss is less than first model. We can conclude that with the increase of hidden layers to certain extent the better model can be built.**

## Answer2)

**Program Description**: A program to change the data source to Breast Cancer dataset and perform the required changes.
Breast Cancer dataset is designated to predict if a patient has Malignant (M) or Benign = Bcancer.

**Code Description:**
Start with setting up environment and importing necessary libraries.
Pandas library as we will be doing dataframe manipulation.
Matplotlib is a plotting library which we will use for our graph

```
# Imported the necessary libraries and created our environment
# Keras is a high-level library.
# Sequential model is a linear stack of layers.
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from sklearn.model_selection import train_test_split
# Importing pandas library as we will be doing dataframe manipulation
import pandas as pd
# matplotlib is a plotting library which we will use for our graph
import matplotlib.pyplot as plt
```

Importing ‘CC.csv’ file into dataframe.
We are using pd.read_csv() method of Pandas for that. It creates a Dataframe from a csv file.
Changed the data source to Breast Cancerdataset.

```
# Next is importing ‘CC.csv’ file into dataframe.
# We are using pd.read_csv() method of Pandas for that. It creates a Dataframe from a csv file.
# Change the data source to Breast Cancerdataset.
dataset = pd.read_csv('breastcancer.csv')
# Printing 3 samples dataset
print(dataset.sample(3))
```

**OUTPUT:**

```
         id diagnosis  ...  fractal_dimension_worst  Unnamed: 32
528  918192         B  ...                  0.07253          NaN
392  903507         M  ...                  0.10190          NaN
149  869931         B  ...                  0.07014          NaN
```

Output shows the 3 sample data from our dataset.
As we can see from the above data set first column is id and last column is showing Nan values, so we will drop first and last column from dataset.

```
# Droping the first and last column fromm dataset
dataset.drop(dataset.columns[[0,32]], axis=1, inplace= True)
print(dataset.sample(3))

# Removing the null values from the dataset
dataset.isnull().sum()
```

**OUTPUT:**

```
[3 rows x 33 columns]
    diagnosis  radius_mean  ...  symmetry_worst  fractal_dimension_worst
4           M        20.29  ...          0.2364                  0.07678
320         B        10.25  ...          0.2608                  0.09702
481         B        13.90  ...          0.2356                  0.07603
```

From the above dataset we can see the first column (id) and last column of null values are dropped.

Separating feature values and target values from our data set. Into x and y

```
# Separating features and target values
x = dataset.iloc[:, 1:]
y = dataset.iloc[:,0]
```

Mapping target column into numerical values (0 and 1) because Breast Cancerdataset is designated to predict if a patient has Malignant (M) or Benign = Bcancer.
b= 0(Benign) and M=1(Malignant)

```
# Mapping target column into numerical values
# b= 0(Benign) and M=1(Malignant)
y = y.map({'B':0,'M' :1})
print(y)
```

**OUTPUT:**

```
[3 rows x 31 columns]
0      1
1      1
2      1
3      1
4      1
      ..
564    1
565    1
566    1
567    1
568    0
Name: diagnosis, Length: 569, dtype: int64
```

In above output we mapped our target column into numerical values(0 and 1).

Next step is to split the data into test and train.
In this test_size = 0.25 means 25% of data for testing and 75% of data for training

```
# Splitting the data into test and train
# In our dataset 1st column is our target value and other columns are features
X_train, X_test, Y_train, Y_test = train_test_split(x,y,test_size=0.25, random_state=87)
```

Creating the model it has 30 input size and dense layer of 40 units and activation= relu. (hidden layer)
Next is output layer.
We took 1 here because it is a binary classification.
Then compiling our first model and fitting the data.

```
# Creating Model
first_nn = Sequential()
# hidden layer
# It has 30 input size and dense layer of 40 units and activation= relu
first_nn.add(Dense(40, input_dim=30, activation='relu'))
# output layer
# We took 1 here because it is a binary classification
first_nn.add(Dense(1, activation='sigmoid'))
# Compiling the first Model
first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
# Fitting the data
my_first_nn_fitted = first_nn.fit(X_train, Y_train, epochs=100, initial_epoch=0)
```

**OUTPUT:**

We got the below output for 100 epochs(accuracy, loss).

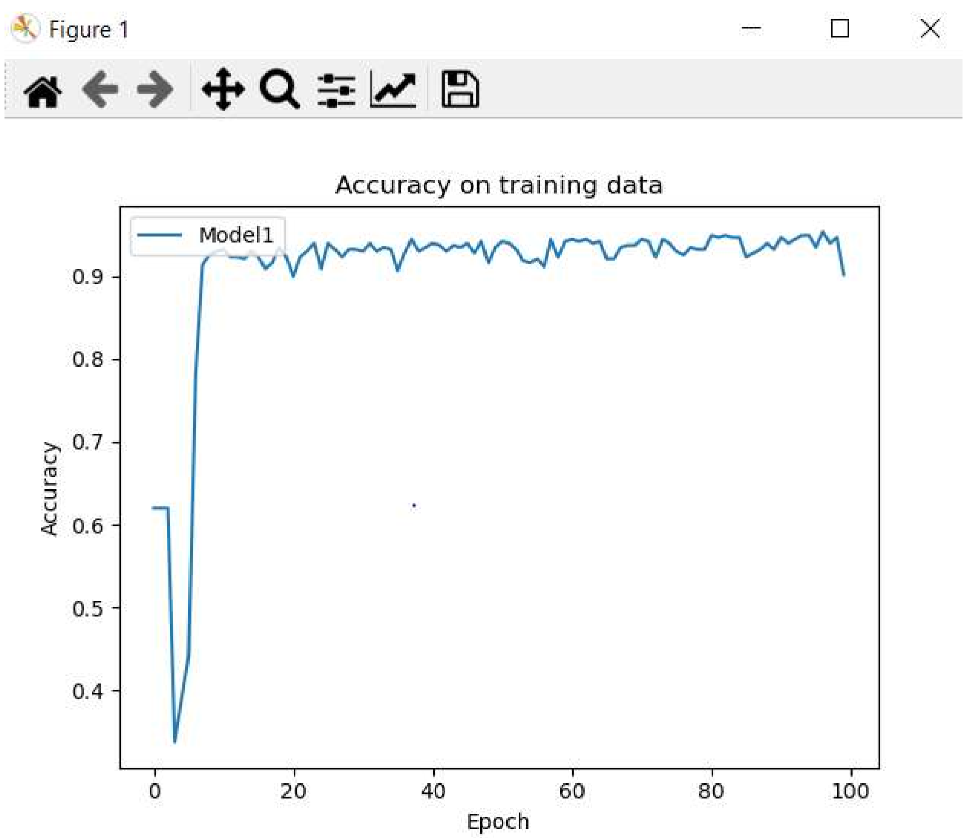
```
 32/426 [=>...........................] - ETA: 0s - loss: 0.0435 - acc: 1.0000
426/426 [==============================] - 0s 44us/step - loss: 0.1272 - acc: 0.9507
Epoch 96/100

 32/426 [=>...........................] - ETA: 0s - loss: 0.2143 - acc: 0.8750
426/426 [==============================] - 0s 44us/step - loss: 0.1596 - acc: 0.9413
Epoch 97/100

 32/426 [=>...........................] - ETA: 0s - loss: 0.1246 - acc: 0.9062
426/426 [==============================] - 0s 40us/step - loss: 0.1411 - acc: 0.9390
Epoch 98/100

 32/426 [=>...........................] - ETA: 0s - loss: 0.1925 - acc: 0.9375
426/426 [==============================] - 0s 42us/step - loss: 0.1620 - acc: 0.9366
Epoch 99/100

 32/426 [=>...........................] - ETA: 0s - loss: 0.3469 - acc: 0.9062
426/426 [==============================] - 0s 35us/step - loss: 0.1683 - acc: 0.9343
Epoch 100/100
```

Next step is plotting history to check the accuracy of our first model.
first_nn_fitted.history for accuracy in these we are printing title putting y label as accuracy and x label as "epoch" Placing a legend on Model1 and loc at upper left and showing our graph.

```python
# Plotting history for accuracy
# first_nn_fitted.history for accuracy
plt.plot(my_first_nn_fitted.history['acc'])
# printing title
plt.title("Accuracy on training data")
# y label as accuracy
plt.ylabel('Accuracy')
# x label as epoch
plt.xlabel('Epoch')
# Placing a legend on Model1 and loc at upper left
plt.legend(['Model1'], loc= 'upper left')
# To show the Graph
plt.show()
```

**OUTPUT:**

The below output shows the accuracy graph of our first model.

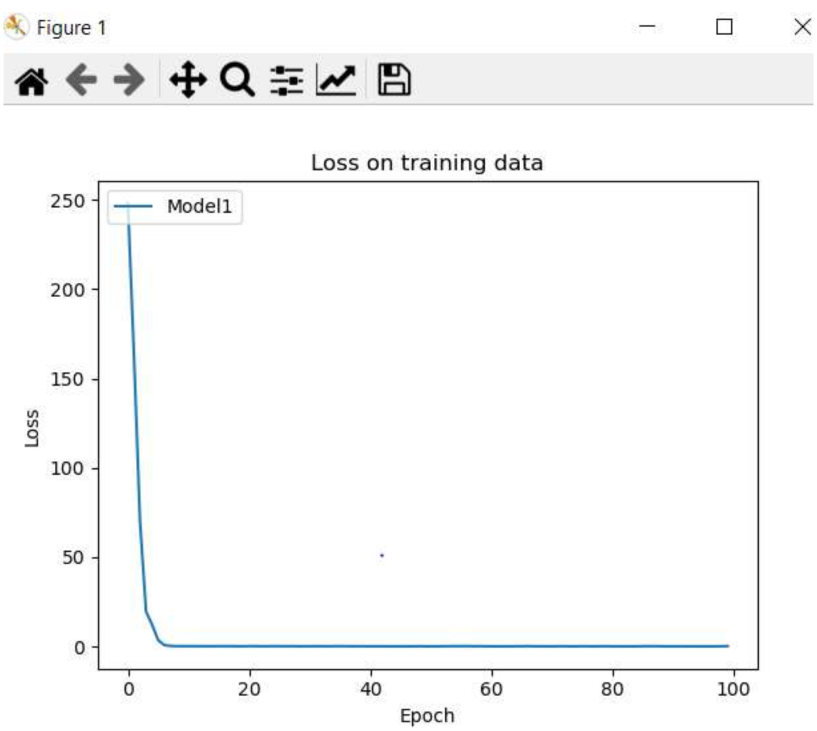


Now Plotting history to check the loss of our first model.

first_nn_fitted.history for loss. In which we are printing title, putting y label as accuracy and x label as "epoch"  Placing a legend on Model1 and loc at upper left and showing our graph.

```
# Plotting history for loss
# first_nn_fitted.history for loss
plt.plot(my_first_nn_fitted.history['loss'])
# Printing Title
plt.title('Loss on training data')
# Printing y label
plt.ylabel('Loss')
# Printing X label
plt.xlabel('Epoch')
# Placing a legend on Model1 and loc at upper left
plt.legend(['Model1'], loc= 'upper left')
# To show the Graph
plt.show()
```

**OUTPUT:**



In the above output, the graph of loss on train data. Which is increasing corresponded with epoch and loss.

**Answer3)**

**Program description**: A program to Normalize the data before feeding.
The Data to the model and check how the normalization change your accuracy.
fromsklearn.preprocessing importStandardScalersc =StandardScaler()
Breast Cancerdataset is designated to predict if a patient has Malignant (M) or Benign = Bcancer

**Code description:**

**We have created first_model of our dataset( in above part), now we create second_model and scale our dataset to check if our accuracy and loss improves.**

Scaling our data.

```
# Scaling the data
# StandardScaler will transform your data such that its distribution will have a mean value 0 and standard deviation of 1
sc = StandardScaler()
sc.fit(x)
x_scale = sc.fit_transform(x)
```

Splitting the data into test and train
In our dataset 1st column is our target value and other columns are features.
We will scale our x and y.

```
# Splitting the data into test and train
# In our dataset 1st column is our target value and other columns are features
X_train, X_test, Y_train, Y_test = train_test_split(x,y,test_size=0.25, random_state=87)
X_scaledtrain, X_scaledtest, Y_train2, Y_test2 = train_test_split(x_scale,y,test_size=0.25, random_state=87)
```

Now we will create our second model:
In which hidden layer
It has 30 input size and dense layer of 40 units and activation= relu
output layer
We took 1 here because it is a binary classification and then we will fit our data.

```python
# creating model
second_nn = Sequential()
# hidden layer
# It has 30 input size and dense layer of 40 units and activation= relu
second_nn.add(Dense(40, input_dim=30, activation='relu'))
# output layer
# We took 1 here because it is a binary classification
second_nn.add(Dense(1, activation='sigmoid'))
# Compiling the second Model
second_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
# Fitting the data
second_nn_fitted = second_nn.fit(X_train, Y_train, epochs=100, initial_epoch=0)
second_nn_fittedscale = second_nn.fit(X_scaledtrain, Y_train2, epochs=100, initial_epoch=0)
```

**OUTPUT:**

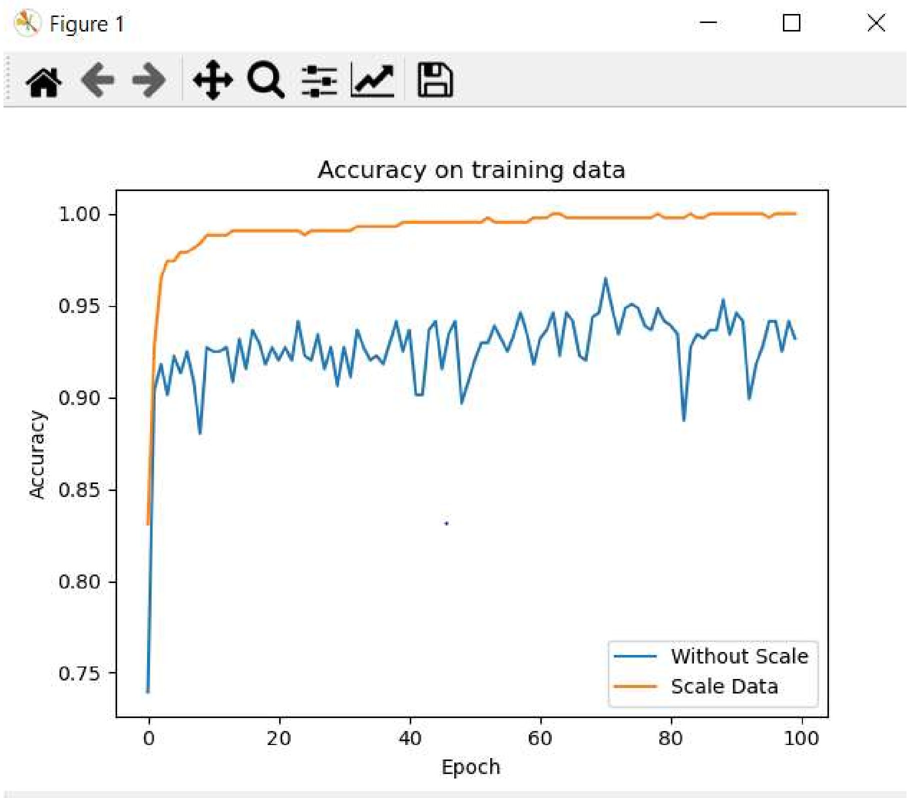
We got the below output for 100 epochs(accuracy, loss).

```
 32/426 [=>...........................] - ETA: 0s - loss: 0.0120 - acc: 1.0000
426/426 [==============================] - 0s 61us/step - loss: 0.0087 - acc: 0.9977
Epoch 96/100

 32/426 [=>...........................] - ETA: 0s - loss: 0.0173 - acc: 1.0000
426/426 [==============================] - 0s 35us/step - loss: 0.0085 - acc: 0.9977
Epoch 97/100

 32/426 [=>...........................] - ETA: 0s - loss: 0.0032 - acc: 1.0000
426/426 [==============================] - 0s 35us/step - loss: 0.0084 - acc: 0.9977
Epoch 98/100

 32/426 [=>...........................] - ETA: 0s - loss: 5.7728e-04 - acc: 1.0000
426/426 [==============================] - 0s 35us/step - loss: 0.0082 - acc: 0.9977
Epoch 99/100

 32/426 [=>...........................] - ETA: 0s - loss: 0.0038 - acc: 1.0000
426/426 [==============================] - 0s 35us/step - loss: 0.0081 - acc: 1.0000
Epoch 100/100
```

Next step is plotting graph of second model's accuracy and compare it with first model's accuracy.

```python
# Plotting history for accuracy
# first_nn_fitted.history for accuracy
plt.plot(second_nn_fitted.history['acc'])
plt.plot(second_nn_fittedscale.history['acc'])
# printing title
plt.title("Accuracy on training data")
# y label as accuracy
plt.ylabel('Accuracy')
# x label as epoch
plt.xlabel('Epoch')
# Placing a legend on without scale and scale data and loc at lower left
plt.legend(['Without Scale', 'Scale Data'], loc= 'lower right')
# To show the Graph
plt.show()
```

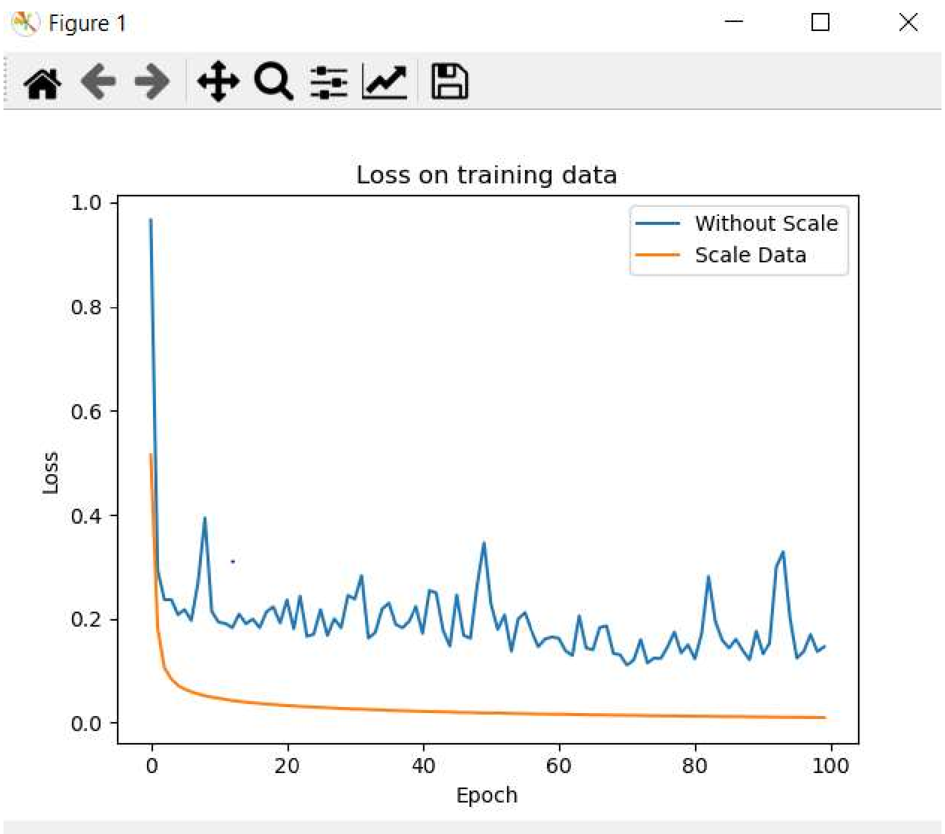
**OUTPUT:**




Plotting graph for second model's loss and comparing it with first model's loss.

```
# Plotting history for loss
# second_nn_fitted.history for loss
plt.plot(second_nn_fitted.history['loss'])
plt.plot(second_nn_fittedscale.history['loss'])
# printing title
plt.title('Loss on training data')
# y label as accuracy
plt.ylabel('Loss')
# x label as epoch
plt.xlabel('Epoch')
# Placing a legend on without scale and scale data and loc at upper right
plt.legend(['Without Scale', 'Scale Data'], loc= 'upper right')
# To show the Graph
plt.show()
```

**OUTPUT:**




**CONCLUSION:**

**------------------------**

**As we can see from above graphs, accuracy of second model which used scaled data is greater while loss is less compared to first model. We can conclude that with use of scaling on input data, a better model can be built.**