Python and Deep Learning Programming

(Spring 2020)

Lab – 2

Team:

Vishnu Vardhan Manne (15)

Sai Jaswanth Gattidi (07)

Vandith Thotla (25)

Department of Computer Science

University of Missouri, Kansas City

Introduction:

This Lab assignment majorly focuses to make us familiar with the Keras Library, and also helps us to get hands-on with concepts such as CNN (Convolution Neutral Network), Word Embedding, LSTM, Auto encoders, Optimizers, Image Dimension Reduction, etc. This lab work also involves the visualization over Tensor Board and the usage of tensorflow graphs, modules and sessions.

Objectives:

Objectives for this Lab are as follows:

1. Build a Sequential model using keras to implement **Linear Regression** with any data set of your choice except the datasets being discussed in the class or used before
    a. Show the graph on Tensor Board
    b. Plot the loss and then change the below parameter and report your view how the result changes in each case
        a. learning rate
        b. batch size
        c. optimizer
        d. activation function

2. Implement the **Logistic Regression** on the following dataset *https://www.kaggle.com/ronitf/heart-disease-uci.*
    a. Normalize the data before feeding it to the model
    b. Show the Loss on TensorBoard
    c. Change three hyperparameter and report how the accuracy changes

3. Implement the image classification with CNN model on anyone of the following datasets
https://www.kaggle.com/slothkong/10-monkey-species
https://www.kaggle.com/prasunroy/natural-images

4. Implement the text classification with CNN model on the following movie reviews dataset
https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews/data

5. Implement the text classification with LSTM model on the following movie reviews dataset.
https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews/data

6. Compare the results of CNN and LSTM models, for the text classification and describe, which model is best for the text classification based on your results. Tune the hyperparameters to attain good accuracy and show the results.

7. Apply Autoencoders on MNIST dataset and show the encoding and decoding on a particular image. Make sure you document each and every line of the code.

Datasets used:

1. Breast Cancer

(https://www.kaggle.com/merishnasuwal/breast-cancer-prediction-dataset)

2. Heart Disease UCI

(https://www.kaggle.com/ronitf/heart-disease-uci)

3. Natural Images

(https://www.kaggle.com/prasunroy/natural-images)

4. Movie Reviews

(https://www.kaggle.com/nltkdata/movie-review)

GitHub Link:

Code:

https://github.com/vishnuvardhanmanne/CS5590-Python-DL

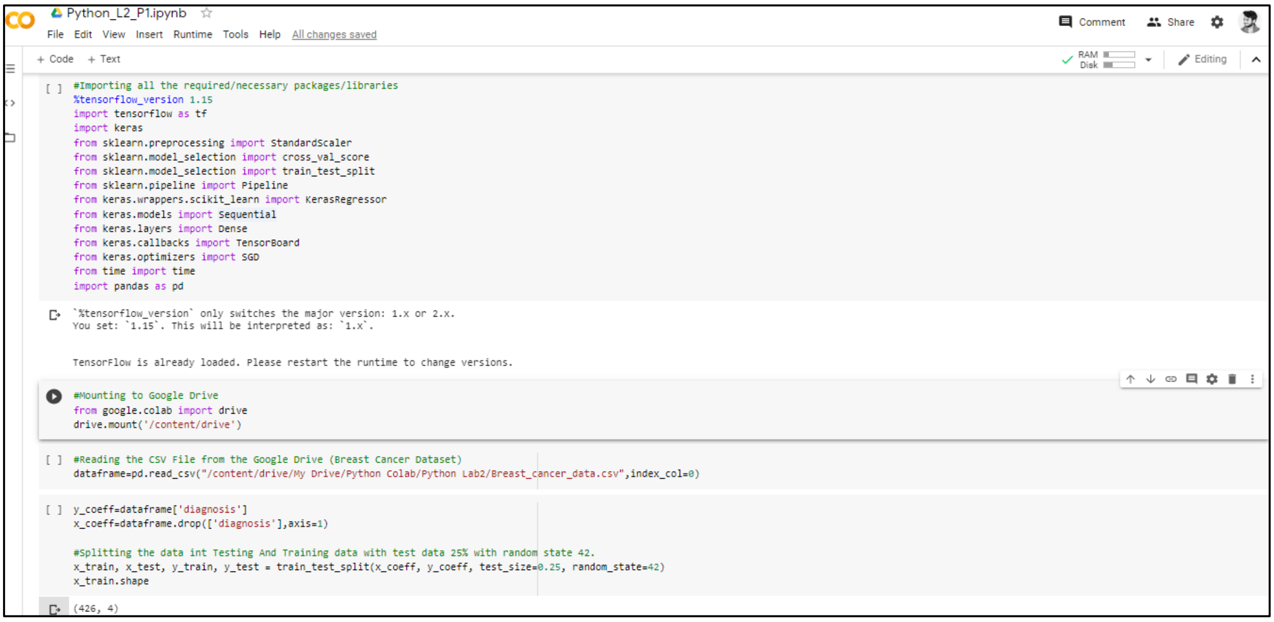
Wiki:

https://github.com/vishnuvardhanmanne/CS5590-Python-DL/wiki/Python-Lab_2-Assignment

Approach/ Methods:

1.

Code and Outputs:



```
#Importing all the required/necessary packages/libraries
%tensorflow_version 1.15
import tensorflow as tf
import keras
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from keras.wrappers.scikit_learn import KerasRegressor
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import TensorBoard
from keras.optimizers import SGD
from time import time
import pandas as pd
```

```
`%tensorflow_version` only switches the major version: 1.x or 2.x.
You set: `1.15`. This will be interpreted as: `1.x`.


TensorFlow is already loaded. Please restart the runtime to change versions.
```

```
#Mounting to Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
#Reading the CSV File from the Google Drive (Breast Cancer Dataset)
dataframe=pd.read_csv("/content/drive/My Drive/Python Colab/Python Lab2/Breast_cancer_data.csv",index_col=0)
```

```
y_coeff=dataframe['diagnosis']
x_coeff=dataframe.drop(['diagnosis'],axis=1)

#Splitting the data int Testing And Training data with test data 25% with random state 42.
x_train, x_test, y_train, y_test = train_test_split(x_coeff, y_coeff, test_size=0.25, random_state=42)
x_train.shape
```

```
(426, 4)
```

```
#Optimiser ADAM with learning rate 0.01
optm = keras.optimizers.Adam(learning_rate=0.01)
```

```
#Creating a Sequential Model Function
def modelfunction():
    mdl=Sequential()
    mdl.add(Dense(16,input_dim=4,init='normal',activation='relu'))
    mdl.add(Dense(32,init='normal',activation='relu'))
    mdl.add(Dense(1))
    mdl.compile(loss='mean_squared_error',optimizer=optm, metrics=['accuracy'])
    return mdl
```

```
#calling the TensorBpoard from keras
tensorboard=TensorBoard(log_dir="p1/{}".format(time()),histogram_freq=0, write_graph=True, write_images=True)
```

```
#Implementing the SkLearn regressor interface
regressor=KerasRegressor(build_fn=modelfunction)
```

```
#Fitting the model with batch size 150 and total of 100 epochs
mdl_fit=regressor.fit(x_train,y_train,epochs= 100, batch_size= 150,callbacks=[tensorboard])
evalve= regressor.score(x_test,y_test)
print(evalve)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(16, input_dim=4, activation="relu", kernel_initializer="normal")`
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(32, activation="relu", kernel_initializer="normal")`
  after removing the cwd from sys.path.
Epoch 1/100
426/426 [==============================] - 0s 779us/step - loss: 74.5228 - accuracy: 0.0000e+00
Epoch 2/100
426/426 [==============================] - 0s 23us/step - loss: 8.1413 - accuracy: 0.1573
Epoch 3/100
426/426 [==============================] - 0s 23us/step - loss: 12.6792 - accuracy: 0.0798
Epoch 4/100
426/426 [==============================] - 0s 19us/step - loss: 1.8706 - accuracy: 0.3873
Epoch 5/100
426/426 [==============================] - 0s 16us/step - loss: 2.5639 - accuracy: 0.0047
```

Initially, to implement the Linear Regression we are creating a Sequential Model Function with loss function as Mean_squared_error and Adam Optimizer with a learning rate of 0.01

KerasRegressor, a Wrapper class, which is an interface of the Scikit-Learn Library which we are implementing for the given Sequential Function to find the loss and the error of the model.

```
#EValuating the model
mdl.evaluate(x_test,y_test)
```

```
143/143 [==============================] - 0s 74us/step
[5.758453729269388, 0.6223776340484619]
```

```
x_test.iloc[1]
```

```
mean_texture          21.31000
mean_perimeter       123.60000
mean_area           1130.00000
mean_smoothness        0.09009
Name: 18.94, dtype: float64
```

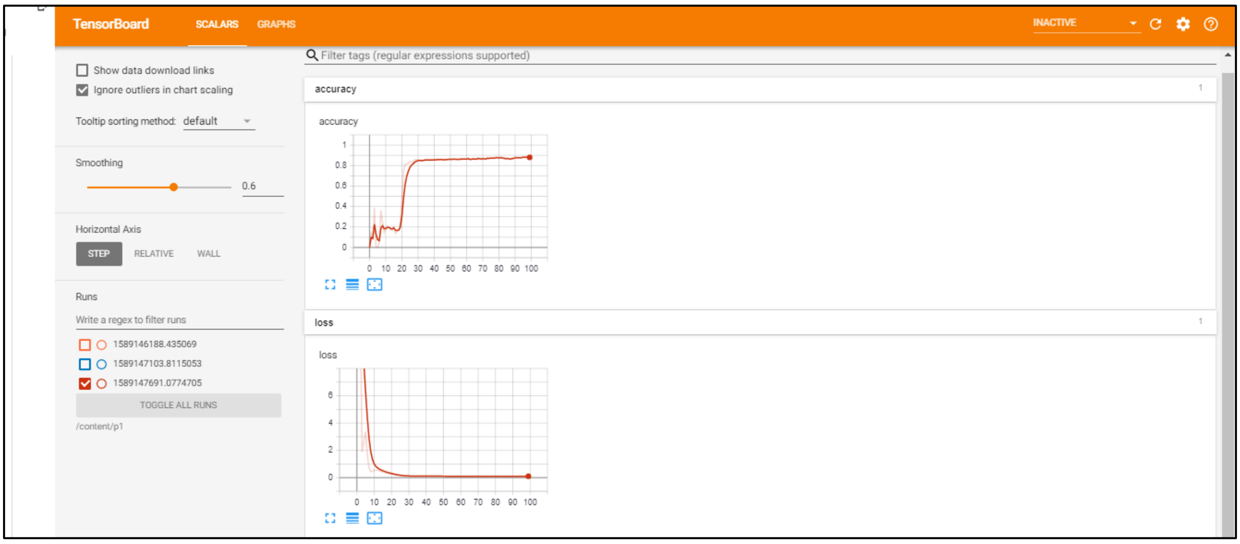
```
#Predicting using the model
y=mdl.predict_classes(x_test.iloc[1:])
```

```
#Loading the Tensor Board
%load_ext tensorboard
```

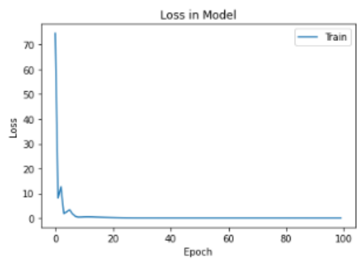
```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
#Getting started with the Tensor Board
%tensorboard --logdir /content/p1
```

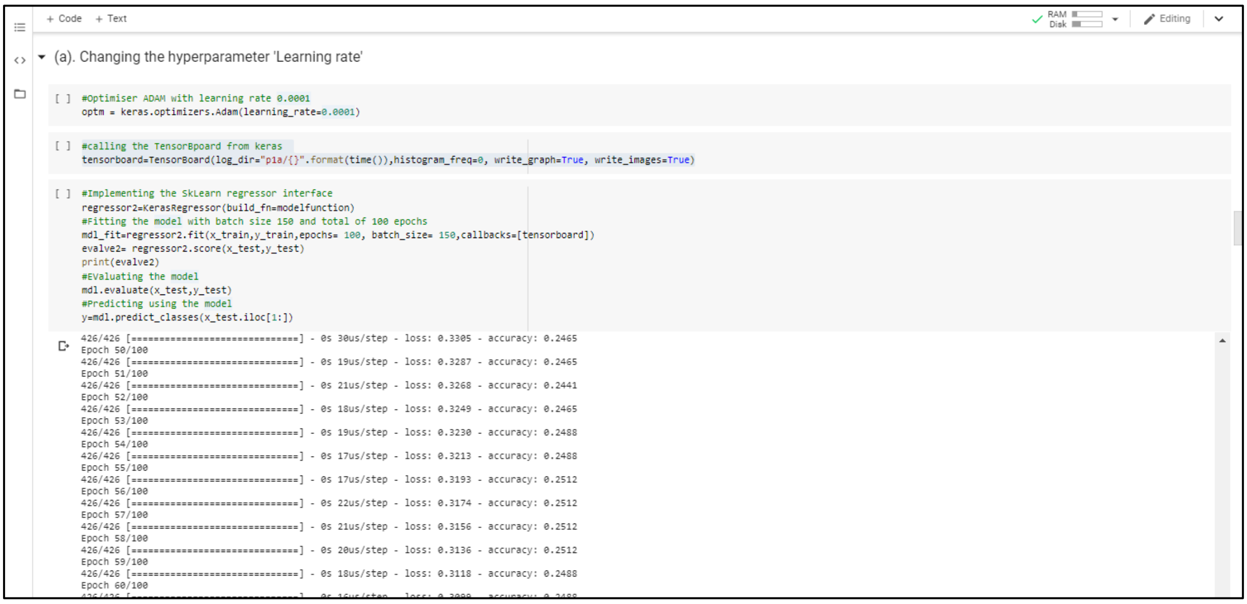
Plotting the accuracy and loss on Tensor Board by creating log files in the Google Colaboratory.

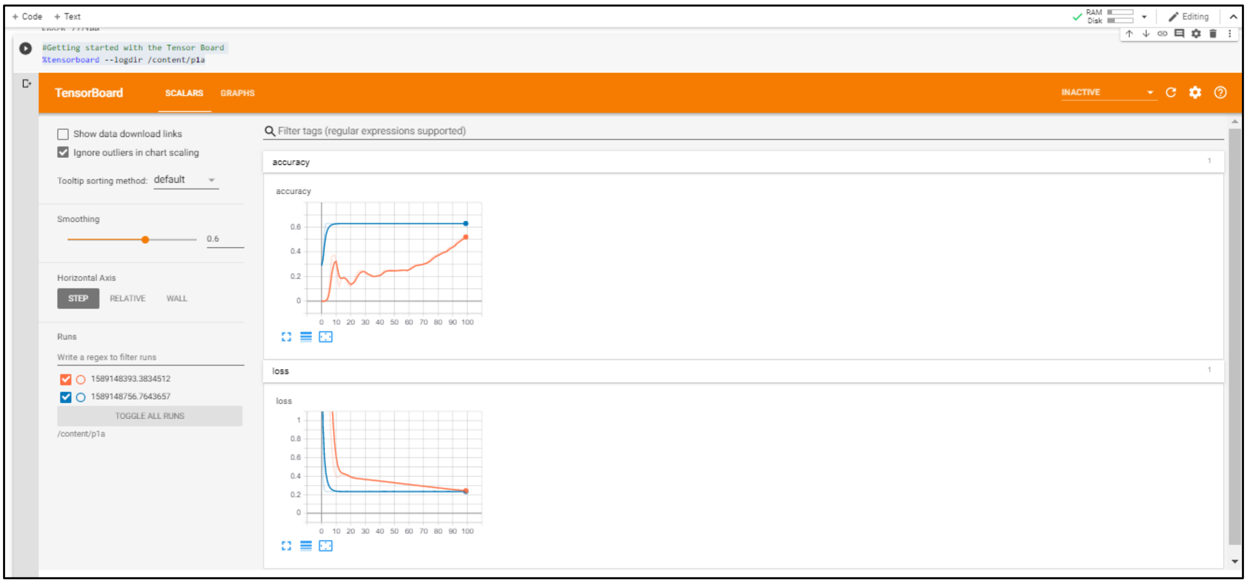



```
#Plotting the Model Loss
plt.plot(mdl_fit.history['loss'])
plt.title('Loss in Model')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'])
plt.show()
```



By changing the hyperparameter “Learning rate”, we observe that the model’s response in the training process defers, as we have lowered the learning rate the training process of the model to acquire utmost accuracy has been lowered too and left with an overall accuracy of 53.52%.



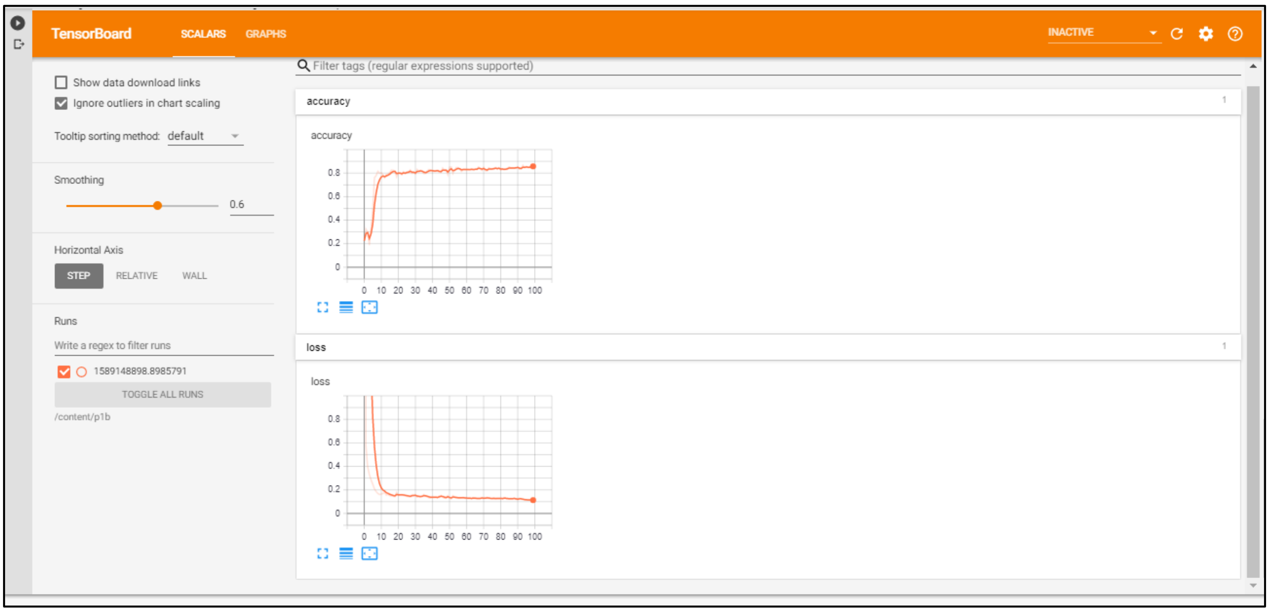


By changing the hyperparameter “Batch size”, we observe that the overall gtradients is not quite stable and is a bit noisy as we lower the batch size from 150 to 20.

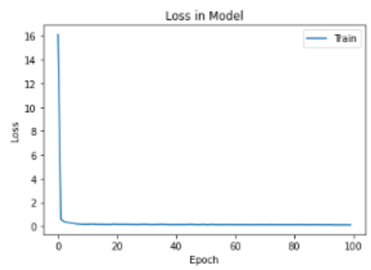
(b). Changing the hyperparameter 'Batch size'

```
[ ] #Optimiser ADAM with learning rate 0.01
    optm = keras.optimizers.Adam(learning_rate=0.01)
    #calling the TensorBpoard from keras
    tensorboard=TensorBoard(log_dir="p1b/{}".format(time()),histogram_freq=0, write_graph=True, write_images=True)
    #Implementing the SkLearn regressor interface
    regressor3=KerasRegressor(build_fn=modelfunction)
    #Fitting the model with batch size 150 and total of 100 epochs
    mdl_fit=regressor3.fit(x_train,y_train,epochs= 100, batch_size= 20,callbacks=[tensorboard]) #changing batch size to 20
    evalve3= regressor3.score(x_test,y_test)
    print(evalve3)
    #EValuating the model
    mdl.evaluate(x_test,y_test)
    #Predicting using the model
    y=mdl.predict_classes(x_test.iloc[1:])
    #Getting started with the Tensor Board
    %tensorboard --logdir /content/p1b
```

```
  after removing the cwd from sys.path.
Epoch 1/100
426/426 [==============================] - 0s 895us/step - loss: 16.1003 - accuracy: 0.2254
Epoch 2/100
426/426 [==============================] - 0s 90us/step - loss: 0.5747 - accuracy: 0.3239
Epoch 3/100
426/426 [==============================] - 0s 86us/step - loss: 0.3966 - accuracy: 0.3005
Epoch 4/100
426/426 [==============================] - 0s 91us/step - loss: 0.3309 - accuracy: 0.1878
Epoch 5/100
426/426 [==============================] - 0s 101us/step - loss: 0.2862 - accuracy: 0.3333
Epoch 6/100
426/426 [==============================] - 0s 89us/step - loss: 0.2485 - accuracy: 0.4836
Epoch 7/100
426/426 [==============================] - 0s 89us/step - loss: 0.2087 - accuracy: 0.7606
Epoch 8/100
426/426 [==============================] - 0s 96us/step - loss: 0.1865 - accuracy: 0.7793
Epoch 9/100
426/426 [==============================] - 0s 92us/step - loss: 0.1687 - accuracy: 0.8146
Epoch 10/100
426/426 [==============================] - 0s 88us/step - loss: 0.1615 - accuracy: 0.7981
Epoch 11/100
426/426 [==============================] - 0s 97us/step - loss: 0.1644 - accuracy: 0.8028
Epoch 12/100
```




```
[ ] #Plotting the Model Loss
    plt.plot(mdl_fit.history['loss'])
    plt.title('Loss in Model')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'])
    plt.show()
```

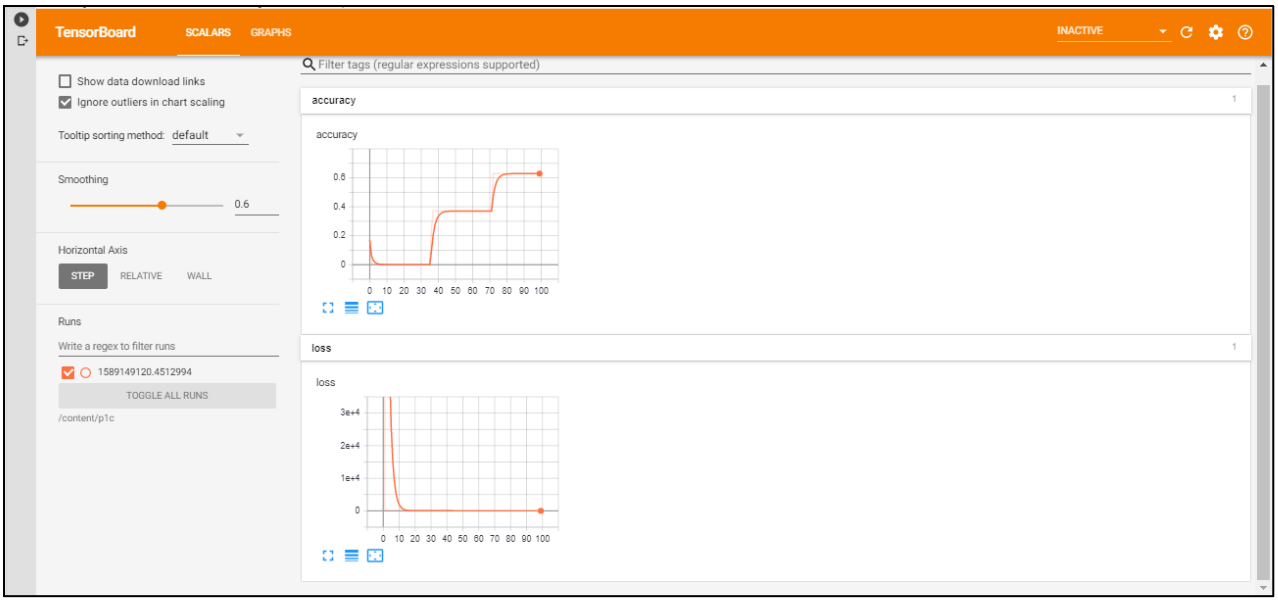


By changing the hyperparameter “Optimizer”, we observe that it doesn’t compute on the entire dataset as we have used SGD. SGD is a Gradient descent which operates on subsets of data.

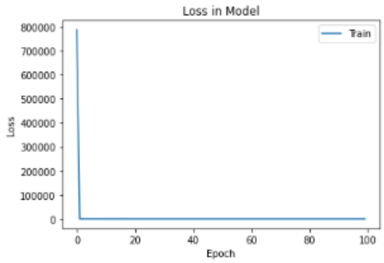
(c). Changing the hyperparameter 'Optimizer'

```
#Optimiser ADAM with learning rate 0.01
optm = keras.optimizers.SGD(learning_rate=0.01)  #Changing Optimizer to SGD
#calling the TensorBpoard from keras
tensorboard=TensorBoard(log_dir="p1c/{}".format(time()),histogram_freq=0, write_graph=True, write_images=True)
#Implementing the SkLearn regressor interface
regressor3=KerasRegressor(build_fn=modelfunction)
#Fitting the model with batch size 150 and total of 100 epochs
mdl_fit=regressor3.fit(x_train,y_train,epochs= 100, batch_size= 150,callbacks=[tensorboard])
evalve3= regressor3.score(x_test,y_test)
print(evalve3)
#EValuating the model
mdl.evaluate(x_test,y_test)
#Predicting using the model
y=mdl.predict_classes(x_test.iloc[1:])
#Getting started with the Tensor Board
%tensorboard --logdir /content/p1c
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(16, input_dim=4, activation="relu", kernel_initializer="normal")`
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(32, activation="relu", kernel_initializer="normal")`
  after removing the cwd from sys.path.
Epoch 1/100
426/426 [==============================] - 0s 799us/step - loss: 787294.3077 - accuracy: 0.1714
Epoch 2/100
426/426 [==============================] - 0s 18us/step - loss: 86.3060 - accuracy: 0.0000e+00
Epoch 3/100
426/426 [==============================] - 0s 18us/step - loss: 76.4797 - accuracy: 0.0000e+00
Epoch 4/100
426/426 [==============================] - 0s 19us/step - loss: 67.7742 - accuracy: 0.0000e+00
Epoch 5/100
426/426 [==============================] - 0s 16us/step - loss: 60.0630 - accuracy: 0.0000e+00
Epoch 6/100
426/426 [==============================] - 0s 19us/step - loss: 53.2315 - accuracy: 0.0000e+00
Epoch 7/100
426/426 [==============================] - 0s 18us/step - loss: 47.1794 - accuracy: 0.0000e+00
Epoch 8/100
426/426 [==============================] - 0s 22us/step - loss: 41.8184 - accuracy: 0.0000e+00
Epoch 9/100
426/426 [==============================] - 0s 17us/step - loss: 37.0692 - accuracy: 0.0000e+00
Epoch 10/100
426/426 [==============================] - 0s 19us/step - loss: 32.8641 - accuracy: 0.0000e+00
Epoch 11/100
```




```
#Plotting the Model Loss
plt.plot(mdl_fit.history['loss'])
plt.title('Loss in Model')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'])
plt.show()
```



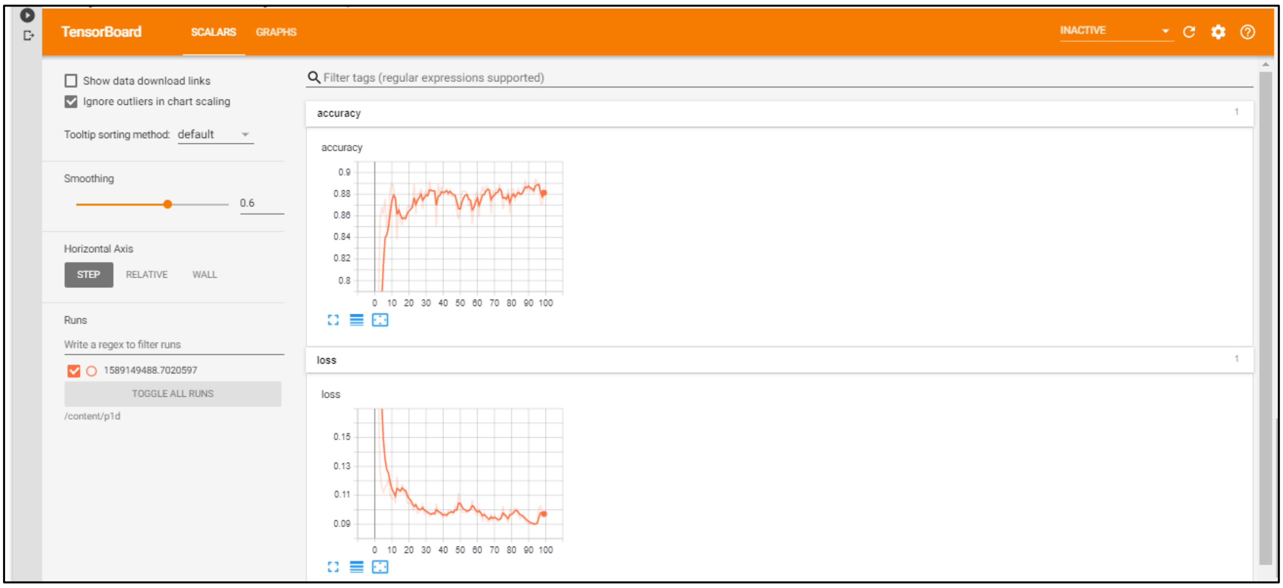
By changing the hyperparameter “Activation Function” from **Relu** to **tanh**, we see that there is not much of difference in the final accuracy of the model but the Scalars in the Tensor Board consists of much more noise than before.

(d). Changing the hyperparameter 'Activation Function'

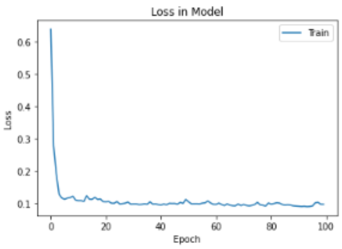
```
[ ] #Optimiser ADAM with learning rate 0.01
    optm = keras.optimizers.Adam(learning_rate=0.01)
    #Creating a Sequential Model Function
    def modelfunction1():
        mdl=Sequential()
        mdl.add(Dense(16,input_dim=4,init='normal',activation='relu'))
        mdl.add(Dense(32,init='normal',activation='tanh'))    #Changing relu to tanh
        mdl.add(Dense(1))
        mdl.compile(loss='mean_squared_error',optimizer=optm, metrics=['accuracy'])
        return mdl
    #calling the TensorBpoard from keras
    tensorboard=TensorBoard(log_dir="p1d/{}".format(time()),histogram_freq=0, write_graph=True, write_images=True)
    #Implementing the SkLearn regressor interface
    regressor4=KerasRegressor(build_fn=modelfunction1)
    #Fitting the model with batch size 150 and total of 100 epochs
    mdl_fit=regressor4.fit(x_train,y_train,epochs= 100, batch_size= 150,callbacks=[tensorboard])
    evalve4= regressor4.score(x_test,y_test)
    print(evalve4)
    #EValuating the model
    mdl.evaluate(x_test,y_test)
    #Predicting using the model
    y=mdl.predict_classes(x_test.iloc[1:])
    #Getting started with the Tensor Board
    %tensorboard --logdir /content/p1d
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(16, input_dim=4, activation="relu", kernel_initializer="normal")`
  """
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(32, activation="tanh", kernel_initializer="normal")`

Epoch 1/100
426/426 [==============================] - 0s 891us/step - loss: 0.6381 - accuracy: 0.3873
Epoch 2/100
426/426 [==============================] - 0s 20us/step - loss: 0.2786 - accuracy: 0.4507
Epoch 3/100
426/426 [==============================] - 0s 19us/step - loss: 0.1905 - accuracy: 0.7465
Epoch 4/100
426/426 [==============================] - 0s 17us/step - loss: 0.1275 - accuracy: 0.8545
Epoch 5/100
```




```
[ ] #Plotting the Model Loss
    plt.plot(mdl_fit.history['loss'])
    plt.title('Loss in Model')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'])
    plt.show()
```



2.

Code and Outputs:

```
#Importing all the required/necessary packages/libraries
%tensorflow_version 1.15
import tensorflow as tf
import keras
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from keras.wrappers.scikit_learn import KerasClassifier
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import TensorBoard
from keras import optimizers
from keras.optimizers import SGD
from keras.datasets import mnist
from keras.utils import np_utils
from time import time
import pandas as pd
import numpy as np
from __future__ import print_function
import datetime
import matplotlib.pyplot as plt
```

```
`%tensorflow_version` only switches the major version: 1.x or 2.x.
You set: `1.15`. This will be interpreted as: `1.x`.


TensorFlow is already loaded. Please restart the runtime to change versions.
```

```
#Mounting to the Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
#Reading the CSV File from the Google Drive (Heart Disease UCI Dataset)
dataframe=pd.read_csv("/content/drive/My Drive/Python Colab/Python Lab2/heart.csv",index_col=0)
df2 = dataframe.astype('float32')
```

```
# Normalizing the values to [0:1]
df2 /= df2.max()
```

```
#Optimiser ADAM with learning rate 0.01
optm = keras.optimizers.Adam(learning_rate=0.01)
```

```
##Splitting the data int Testing And Training data with test data 25% with random state 42.
y_coeff = df2['target']
x_coeff = df2.drop(['target'], axis = 1)
x_train, x_test, y_train, y_test = train_test_split(x_coeff, y_coeff,
                                                    test_size=0.25, random_state=42)
```

```
#Converting to one-hot vector
y_train1 = np_utils.to_categorical(y_train, 10)
y_test1 = np_utils.to_categorical(y_test, 10)
#Creating and Compiling a Sequential Model
mdl = Sequential()
mdl.add(Dense(output_dim=10, input_shape=(12,), init='normal', activation='softmax'))
mdl.compile(optimizer=optm, loss='categorical_crossentropy', metrics=['accuracy'])
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(input_shape=(12,), activation="softmax", units=10, kernel_initializer="normal")`
  """
```

```
#calling the TensorBoard from keras
tensorboard = TensorBoard(log_dir="p2/{}".format(time()),histogram_freq=0, write_graph=True, write_images=True)
```

```
##Fitting the model with batch size 50 and total of 20 epochs
mdl_fit=mdl.fit(x_train, y_train1, nb_epoch=15, batch_size=50,callbacks=[tensorboard])
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: UserWarning: The `nb_epoch` argument in `fit` has been renamed `epochs`.
  """Entry point for launching an IPython kernel.
Epoch 1/15
227/227 [==============================] - 0s 325us/step - loss: 2.1446 - accuracy: 0.3921
Epoch 2/15
227/227 [==============================] - 0s 41us/step - loss: 1.7287 - accuracy: 0.5419
Epoch 3/15
227/227 [==============================] - 0s 38us/step - loss: 1.3855 - accuracy: 0.5683
Epoch 4/15
227/227 [==============================] - 0s 39us/step - loss: 1.1410 - accuracy: 0.6079
Epoch 5/15
227/227 [==============================] - 0s 36us/step - loss: 0.9753 - accuracy: 0.7225
Epoch 6/15
227/227 [==============================] - 0s 38us/step - loss: 0.8657 - accuracy: 0.7489
Epoch 7/15
227/227 [==============================] - 0s 37us/step - loss: 0.7896 - accuracy: 0.7577
```

```
#predicting the accuracy of the model
score = mdl.evaluate(x_test, y_test1, verbose=1)
print('Loss: %.2f, Accuracy: %.2f' % (score[0], score[1]))
```

```
76/76 [==============================] - 0s 229us/step
Loss: 0.54, Accuracy: 0.84
```

```
y=mdl.predict_classes(x_test.iloc[1:])
```

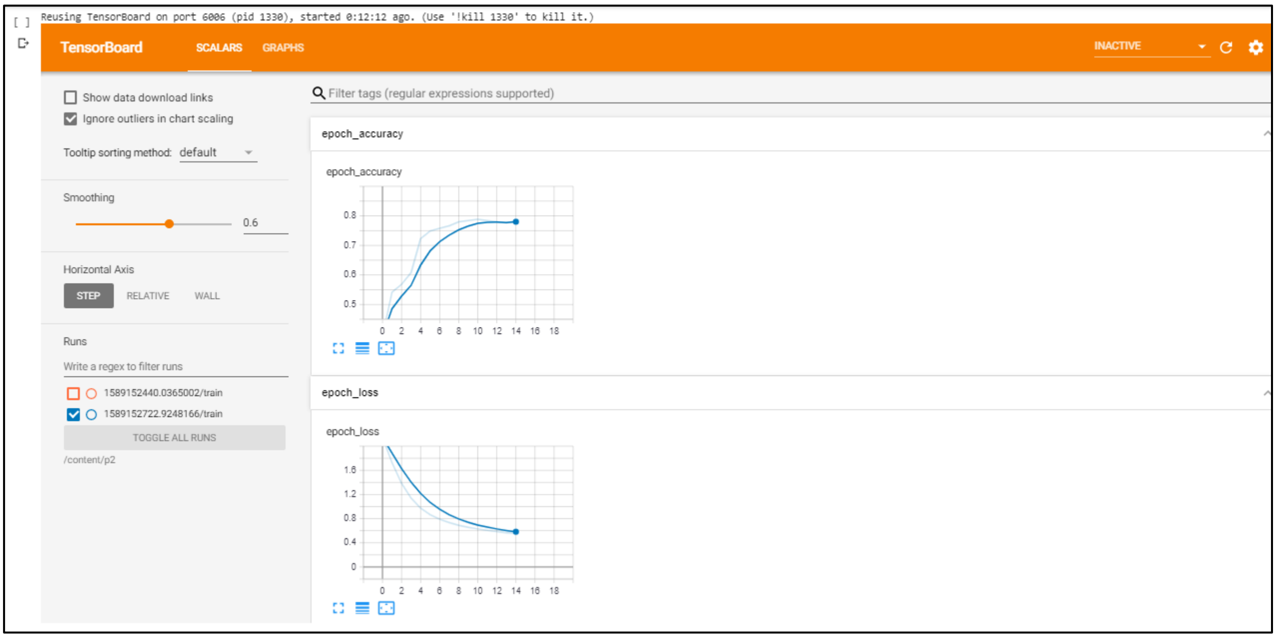
```
#Loading the Tensor Board
%load_ext tensorboard
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

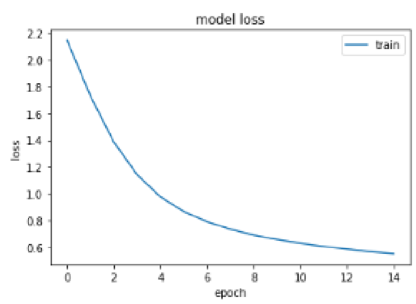
```
#Getting started with the Tensor Board
%tensorboard --logdir /content/p2
```

Initially, we are normalizing the dataset (normalizing all the dataset's features) and then to perform Logistic regression we are creating a Sequential Model using softmax activation with loss function as categorical_crossentropy and Adam Optimizer with a learning rate of 0.01

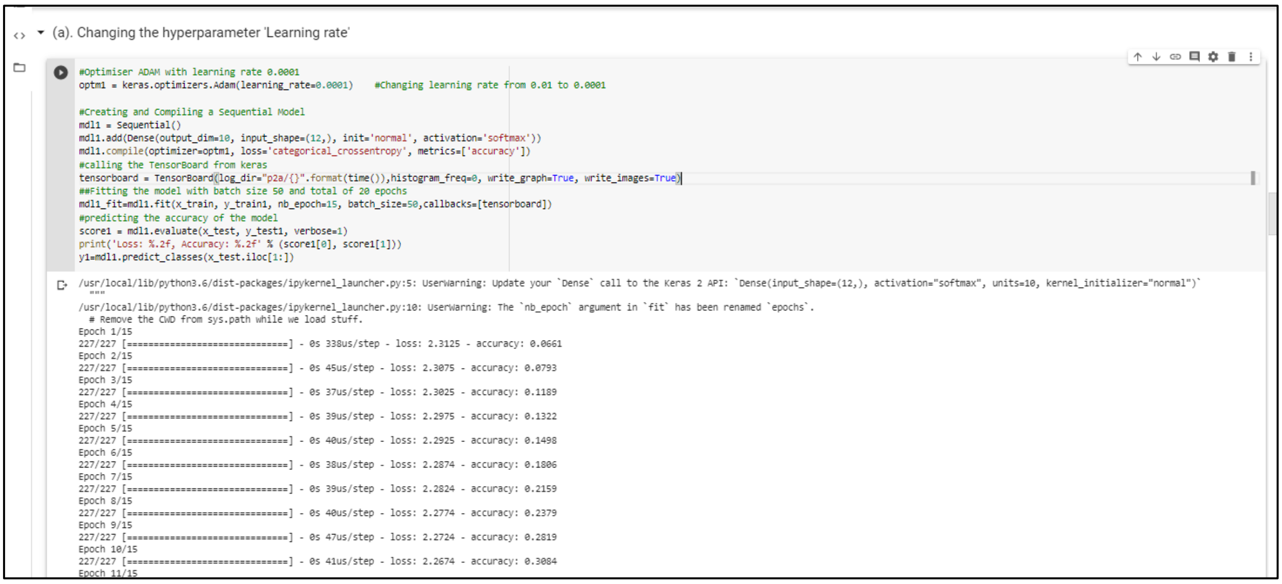
Plotting the accuracy and loss on Tensor Board by creating log files in the Google Colaboratory.




```
#plotting the loss
plt.plot(mdl_fit.history['loss'])
# plt.plot(history.history['test_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'])
plt.show()
```



By changing the hyperparameter “Learning rate”, we observe that the model’s response in the training process defers, as we have lowered the learning rate the training process of the model to acquire utmost accuracy has been lowered too and left with an overall accuracy of 39.65 %.

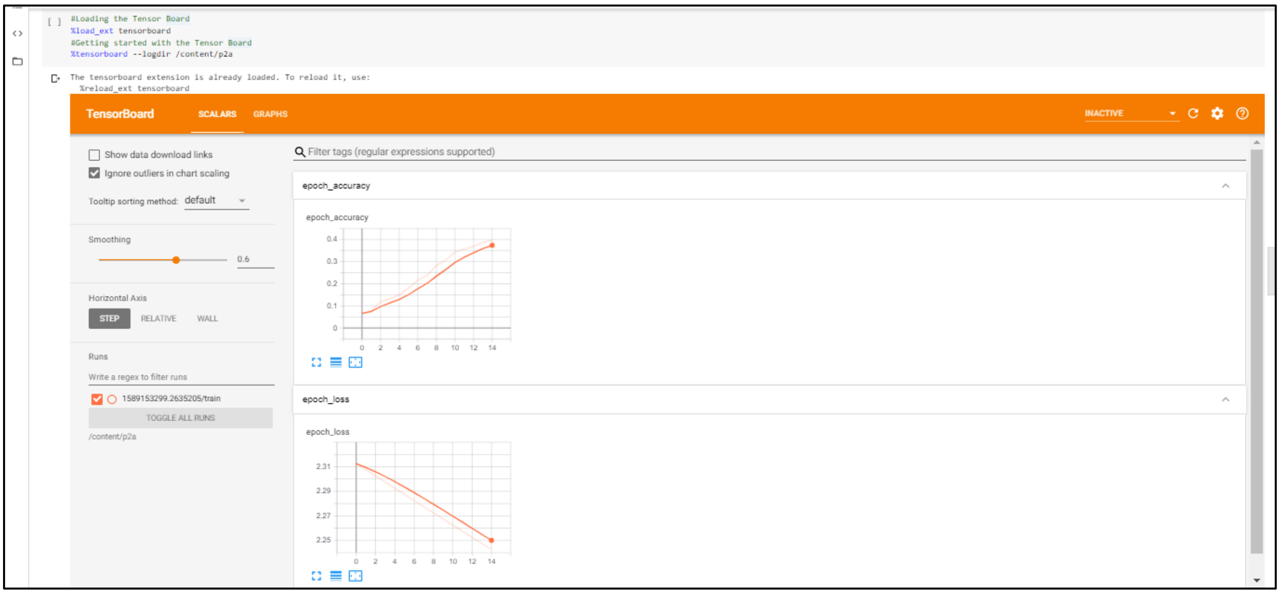


(a). Changing the hyperparameter 'Learning rate'

```
#Optimiser ADAM with learning rate 0.0001
optm1 = keras.optimizers.Adam(learning_rate=0.0001)    #Changing learning rate from 0.01 to 0.0001

#Creating and Compiling a Sequential Model
mdl1 = Sequential()
mdl1.add(Dense(output_dim=10, input_shape=(12,), init='normal', activation='softmax'))
mdl1.compile(optimizer=optm1, loss='categorical_crossentropy', metrics=['accuracy'])
#calling the TensorBoard from keras
tensorboard = TensorBoard(log_dir="p2a/{}".format(time()),histogram_freq=0, write_graph=True, write_images=True)
##Fitting the model with batch size 50 and total of 20 epochs
mdl1_fit=mdl1.fit(x_train, y_train1, nb_epoch=15, batch_size=50,callbacks=[tensorboard])
#predicting the accuracy of the model
score1 = mdl1.evaluate(x_test, y_test1, verbose=1)
print('Loss: %.2f, Accuracy: %.2f' % (score1[0], score1[1]))
y1=mdl1.predict_classes(x_test.iloc[1:])
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(input_shape=(12,), activation="softmax", units=10, kernel_initializer="normal")`
  """
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:10: UserWarning: The `nb_epoch` argument in `fit` has been renamed `epochs`.
  # Remove the CWD from sys.path while we load stuff.
Epoch 1/15
227/227 [==============================] - 0s 338us/step - loss: 2.3125 - accuracy: 0.0661
Epoch 2/15
227/227 [==============================] - 0s 45us/step - loss: 2.3075 - accuracy: 0.0793
Epoch 3/15
227/227 [==============================] - 0s 37us/step - loss: 2.3025 - accuracy: 0.1189
Epoch 4/15
227/227 [==============================] - 0s 39us/step - loss: 2.2975 - accuracy: 0.1322
Epoch 5/15
227/227 [==============================] - 0s 40us/step - loss: 2.2925 - accuracy: 0.1498
Epoch 6/15
227/227 [==============================] - 0s 38us/step - loss: 2.2874 - accuracy: 0.1806
Epoch 7/15
227/227 [==============================] - 0s 39us/step - loss: 2.2824 - accuracy: 0.2159
Epoch 8/15
227/227 [==============================] - 0s 40us/step - loss: 2.2774 - accuracy: 0.2379
Epoch 9/15
227/227 [==============================] - 0s 47us/step - loss: 2.2724 - accuracy: 0.2819
Epoch 10/15
227/227 [==============================] - 0s 41us/step - loss: 2.2674 - accuracy: 0.3084
Epoch 11/15
```

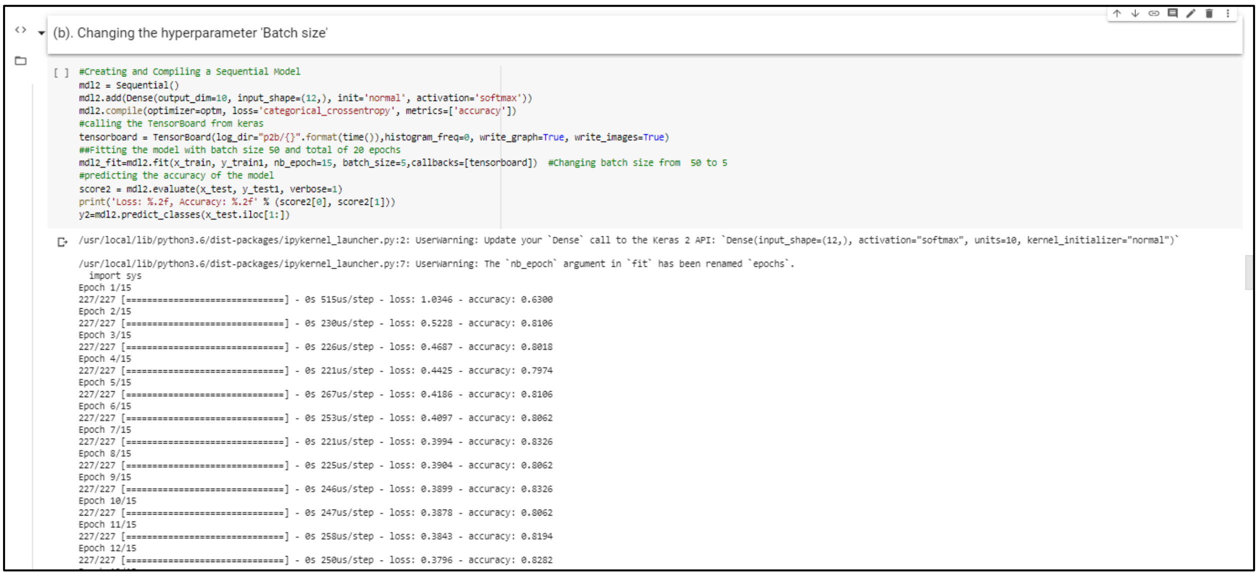


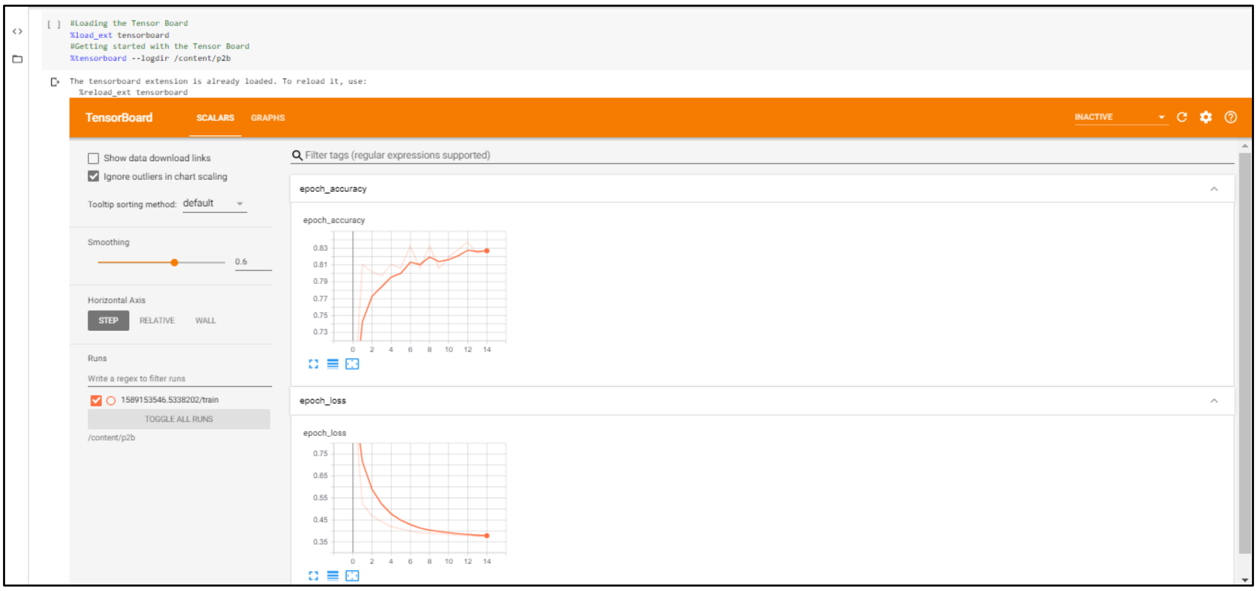
```
#Loading the Tensor Board
%load_ext tensorboard
#Getting started with the Tensor Board
%tensorboard --logdir /content/p2a
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

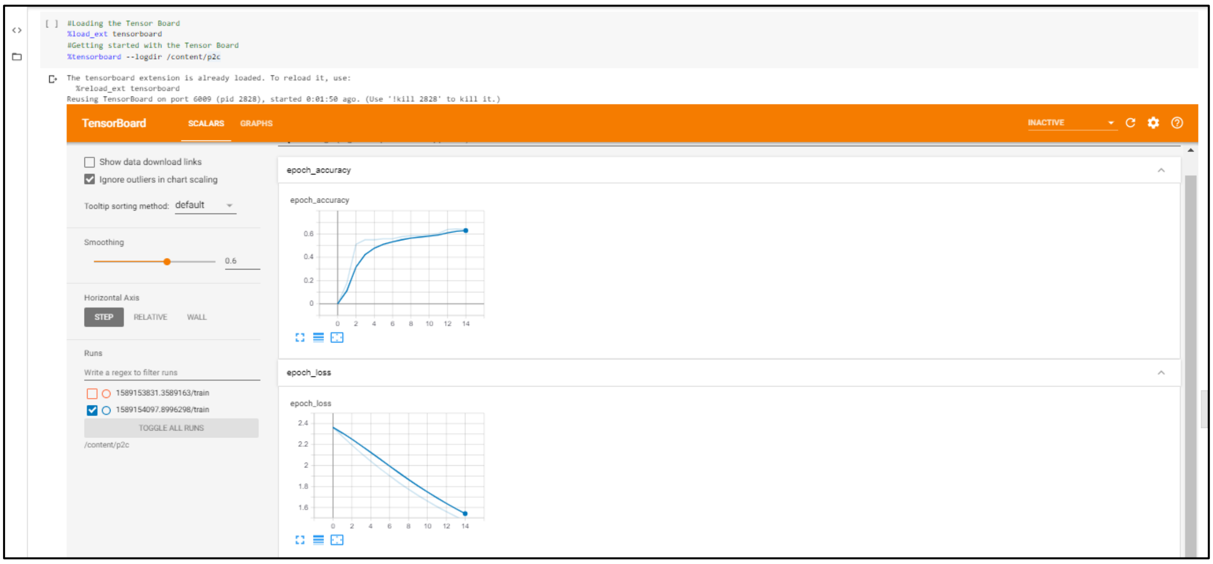

```
#plotting the loss
plt.plot(mdl1_fit.history['loss'])
# plt.plot(history.history['test_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'])
plt.show()
```

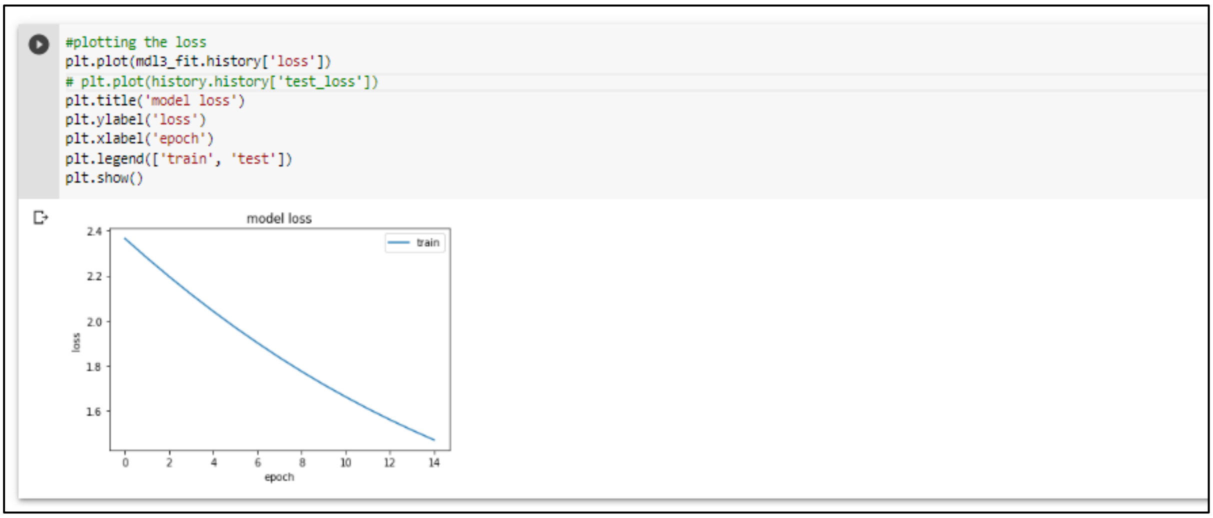
By changing the hyperparameter “Batch size”, we observe that the overall graph is a bit noisy and the execution time for each epoch also increased as we lower the batch size from 50 to 5.





By changing the hyperparameter "Optimizer", we observe that it doesn't compute on the entire dataset as we have used SGD. SGD is a Gradient descent which operates on subsets of data.





By changing the hyperparameter “Activation Function” from **Relu** to **tanh**, we see that there is not much of difference in the final accuracy of the model.

(d). Changing the hyperparameter 'Activation Function'

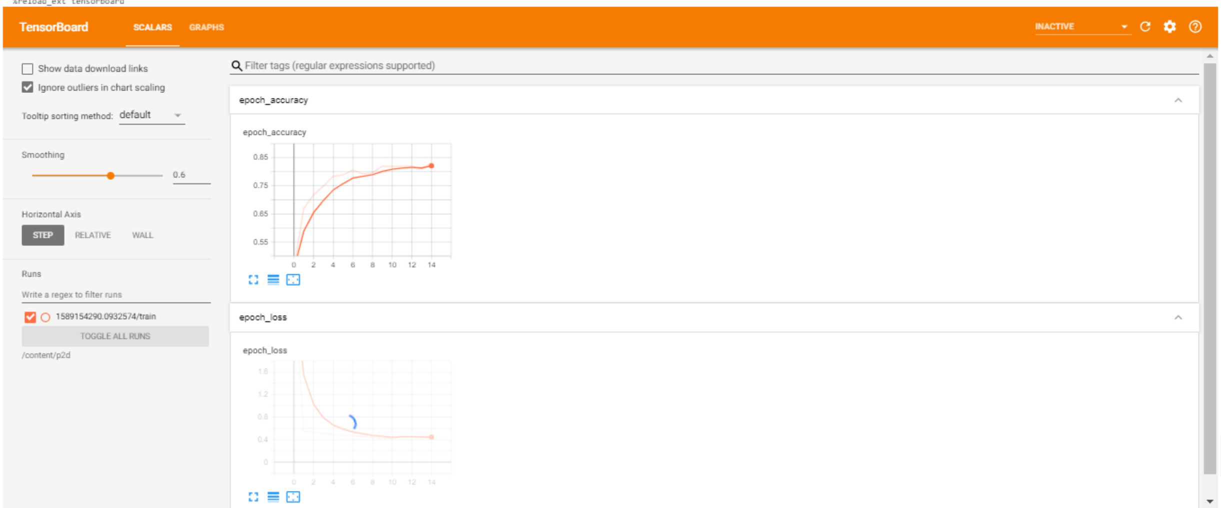
```
#Optimiser ADAM with learning rate 0.01
optm4 = keras.optimizers.Adam(learning_rate=0.01)

#Creating and Compiling a Sequential Model
mdl4 = Sequential()
mdl4.add(Dense(output_dim=10, input_shape=(12,), init='normal', activation='relu')) #changing activation from softmax to relu
mdl4.compile(optimizer=optm4, loss='categorical_crossentropy', metrics=['accuracy'])
#calling the TensorBoard from keras
tensorboard = TensorBoard(log_dir="p2d/{}".format(time()),histogram_freq=0, write_graph=True, write_images=True)
##Fitting the model with batch size 50 and total of 20 epochs
mdl4_fit=mdl4.fit(x_train, y_train1, nb_epoch=15, batch_size=50,callbacks=[tensorboard])
#predicting the accuracy of the model
score4 = mdl4.evaluate(x_test, y_test1, verbose=1)
print('Loss: %.2f, Accuracy: %.2f' % (score4[0], score4[1]))
y4=mdl4.predict_classes(x_test.iloc[1:])
```

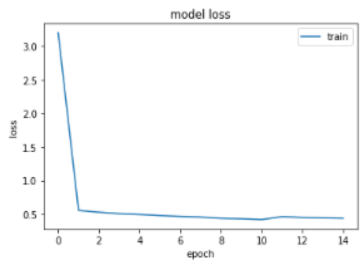
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(input_shape=(12,), activation="relu", units=10, kernel_initializer="normal")`
  """
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:10: UserWarning: The `nb_epoch` argument in `fit` has been renamed `epochs`.
  # Remove the CWD from sys.path while we load stuff.
Epoch 1/15
227/227 [==============================] - 0s 588us/step - loss: 3.1977 - accuracy: 0.4537
Epoch 2/15
227/227 [==============================] - 0s 50us/step - loss: 0.5539 - accuracy: 0.6696
Epoch 3/15
227/227 [==============================] - 0s 43us/step - loss: 0.5277 - accuracy: 0.7181
Epoch 4/15
227/227 [==============================] - 0s 38us/step - loss: 0.5063 - accuracy: 0.7489
Epoch 5/15
227/227 [==============================] - 0s 40us/step - loss: 0.4937 - accuracy: 0.7841
Epoch 6/15
227/227 [==============================] - 0s 38us/step - loss: 0.4781 - accuracy: 0.7885
Epoch 7/15
227/227 [==============================] - 0s 42us/step - loss: 0.4651 - accuracy: 0.8062
Epoch 8/15
227/227 [==============================] - 0s 39us/step - loss: 0.4547 - accuracy: 0.7930
Epoch 9/15
227/227 [==============================] - 0s 38us/step - loss: 0.4366 - accuracy: 0.7974
Epoch 10/15
227/227 [==============================] - 0s 37us/step - loss: 0.4302 - accuracy: 0.8194
Epoch 11/15
```

```
#Loading the Tensor Board
%load_ext tensorboard
#Getting started with the Tensor Board
%tensorboard --logdir /content/p2d
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```




```
#plotting the loss
plt.plot(mdl4_fit.history['loss'])
# plt.plot(history.history['test_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'])
plt.show()
```



3.

Code and Outputs:

Initially, we used Natural-images as our dataset.

We have imported the required libraries such as Tensorflow, numpy etc.

We have downloaded the data from the link provided in the above Objectives and uploaded the data to Google Drive to utilize it in the program.

```
%tensorflow_version 1.15
import tensorflow as tf
from keras import Sequential
import numpy as np
from keras.layers import Dense
from keras.utils import to_categorical
from keras.preprocessing.image import ImageDataGenerator, load_img
import os
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from keras.layers import Dense, Flatten, Dropout, Input
from keras.constraints import maxnorm
from keras.models import Model
from keras.optimizers import SGD, Adam
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.utils import np_utils
from keras import backend as K
K.common.image_dim_ordering()
```

```
`%tensorflow_version` only switches the major version: 1.x or 2.x.
You set: `1.15`. This will be interpreted as: `1.x`.


TensorFlow is already loaded. Please restart the runtime to change versions.
'tf'
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
label=['airplane','car','cat','dog','flower','fruit','motorbike','person']
```

```
import glob #airplane images retrieving
import cv2
train_images=[]
c=0
for filename in glob.glob('/content/drive/My Drive/pythonlab2/natural_images/airplane/*.jpg'):
    imagenormal = cv2.imread(filename)
    output = cv2.resize(imagenormal, (28,28))
    train_images.append([output,0])
```

```
for filename in glob.glob('/content/drive/My Drive/pythonlab2/natural_images/car/*.jpg'):#car images retrieving
    imagenormal = cv2.imread(filename)
    output1 = cv2.resize(imagenormal, (28,28))
    train_images.append([output1,1])
```

```
for filename in glob.glob('/content/drive/My Drive/pythonlab2/natural_images/cat/*.jpg'):#cat images retrieving
    imagenormal = cv2.imread(filename)
    output1 = cv2.resize(imagenormal, (28,28))
    train_images.append([output1,2])
```

```
for filename in glob.glob('/content/drive/My Drive/pythonlab2/natural_images/dog/*.jpg'):#dog images retrieving
    imagenormal = cv2.imread(filename)
    output1 = cv2.resize(imagenormal, (28,28))
    train_images.append([output1,3])
```

```
for filename in glob.glob('/content/drive/My Drive/pythonlab2/natural_images/flower/*.jpg'):#flower images retrieving
    imagenormal = cv2.imread(filename)
    output1 = cv2.resize(imagenormal, (28,28))
    train_images.append([output1,4])
```

```
for filename in glob.glob('/content/drive/My Drive/pythonlab2/natural_images/fruit/*.jpg'):#fruit images retrieving
    imagenormal = cv2.imread(filename)
    output1 = cv2.resize(imagenormal, (28,28))
    train_images.append([output1,5])
```

```
for filename in glob.glob('/content/drive/My Drive/pythonlab2/natural_images/motorbike/*.jpg'):#motorbike images retrieving
    imagenormal = cv2.imread(filename)
    output1 = cv2.resize(imagenormal, (28,28))
    train_images.append([output1,6])
```

```
for filename in glob.glob('/content/drive/My Drive/pythonlab2/natural_images/person/*.jpg'):#person images retrieving
    imagenormal = cv2.imread(filename)
    output1 = cv2.resize(imagenormal, (28,28))
    train_images.append([output1,7])
```

```
x=[]
y=[]
for im,label in train_images:
  x.append(im)
  y.append(label)
```

```
x=np.array(x).reshape(-1,28,28,3) #reshape the size
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2, random_state=3)
```

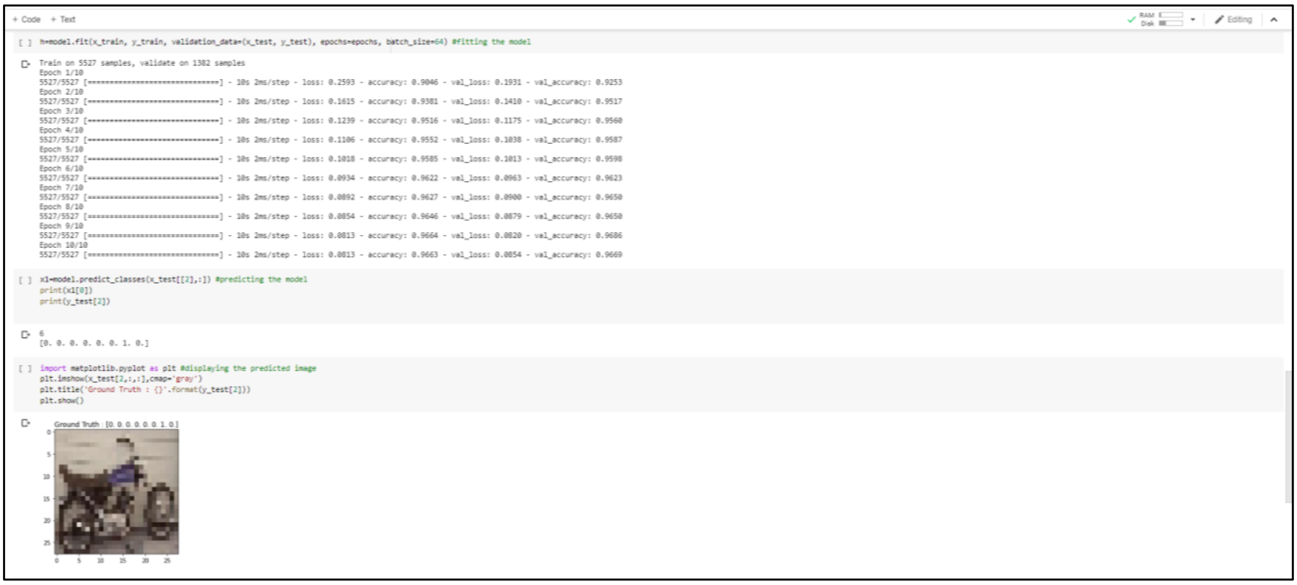
The data is then split into different lists x & y

After splitting, we have reshaped the image data into the shape (28, 28, 3)

The image is then plotted using Normalization of each pixel by 255.0 for an easier computation.

In this image classification, we used Sequential Model with layers i.e., one Con2D layers and one Maxpooling layer and then softmax as the activation function (because we have more than 2 classes)

We then compiled this model, by using binary_crossentropy and then fitted the model.

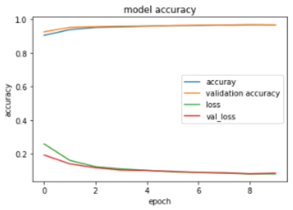


We fit the model for 10 epochs and then predicted the image for test data as shown above.

Finally, we plotted the graph for various parameters like accuracy, validation accuracy, loss and validation loss.

```
[ ] model.save("jaswanth.h5") #saving the modxel

[ ] import matplotlib.pyplot as plt #plotting the model accuracy
    plt.plot(h.history['accuracy'])
    plt.plot(h.history['val_accuracy'])
    plt.plot(h.history['loss'])
    plt.plot(h.history['val_loss'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['accuray', 'validation accuracy','loss','val_loss'])
    plt.show()
```



4.

Code and Outputs:

```
[ ] # Connecting to Google Drive
    from google.colab import drive
    drive.mount('/content/drive/')

    Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pf

    Enter your authorization code:
    ··········
    Mounted at /content/drive/

    # Importing the required libraries
    import pandas as pa
    from keras.preprocessing.text import Tokenizer
    from keras.preprocessing.sequence import pad_sequences
    from keras.models import Sequential
    from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D, Dropout, Conv1D, GlobalMaxPooling1D
    from sklearn.model_selection import train_test_split
    from keras.utils.np_utils import to_categorical
    import re
    from sklearn.preprocessing import LabelEncoder
    import matplotlib.pyplot as plt
    from keras.optimizers import adam

[ ] # Reading the train.tsv into train_data1
    train_data1 = pa.read_csv("/content/drive/My Drive/Lab2/Datasets/train.tsv",sep="\t")
    # Reading the test.tsv into test.tsv
    test_data1 = pa.read_csv("/content/drive/My Drive/Lab2/Datasets/test.tsv",sep="\t")
    # Printing the shape of the datasets
    print(train_data1.shape)
    train_data1.head
    print(test_data1.shape)
    test_data1.head
    # Dropping the unwanted columns
    train_data1 = train_data1.drop(columns=['PhraseId', 'SentenceId'])
    # Removing the non-alphabetic characters
```

First, we have connected the Google Colab with Google drive and then we have imported all the required libraries.

Then, we have read the ‘train.tsv’, ‘test.tsv’ files present in the google drive into train_data1 and test_data1 variable’s respectively.

Then we have printed the shape and head of both the train and test files.

```
[ ] # Removing the non-alphabetic characters
    train_data1['Phrase'] = train_data1['Phrase'].apply(lambda x: re.sub('[^a-zA-z0-9\s]', '', x.lower()))
    test_data1 = test_data1.drop(columns=['PhraseId', 'SentenceId'])
    test_data1['Phrase'] = test_data1['Phrase'].apply(lambda x: re.sub('[^a-zA-z0-9\s]', '', x.lower()))

    (156060, 4)
    (66292, 3)

[ ] # Taking the target column and deopping it from the training data
    label1=train_data1[['Sentiment']]
    train_data1=train_data1.drop(columns=['Sentiment'])

[ ] # Tokenization on train data
    max_feature1 = 4000
    tokenizer = Tokenizer(num_words=max_feature1, split=' ')
    tokenizer.fit_on_texts(train_data1['Phrase'].values)
    X_train1 = tokenizer.texts_to_sequences(train_data1['Phrase'].values)
    X_train1 = pad_sequences(X_train1)

[ ] # Tokenization on test data
    max_feature2 = 2000
    tokenizer = Tokenizer(num_words=max_feature2, split=' ')
    tokenizer.fit_on_texts(test_data1['Phrase'].values)
    X_test1 = tokenizer.texts_to_sequences(test_data1['Phrase'].values)
    X_test1 = pad_sequences(X_test1)

[ ] X_train1.shape

    (156060, 46)

[ ] X_test1.shape

    (66292, 46)
```

Then, we have converted the content in 'Phrase' column to lower case by using the lambda function and dropped the unnecessary columns(PhraseId , SentenceId) and then dropped the target column Phrase from the training data and stored it into label1 variable.

Then we have applied tokenization on training and test data for converting the text into words and performed padding in order to obtain strings of equal length for tokens.

After performing padding operation we have printed the shape of train and test data.

```
[ ] # Performing train test and split
    label_encoder = LabelEncoder()
    integer_encoded = label_encoder.fit_transform(label1)
    Y_train1 = to_categorical(integer_encoded)
    X_train, X_test, Y_train, Y_test = train_test_split(X_train1, Y_train1, test_size=0.2, random_state=10)
    print(X_train.shape,Y_train.shape)
    print(X_test.shape,Y_test.shape)

    (124848, 46) (124848, 5)
    (31212, 46) (31212, 5)
    /usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/_label.py:251: DataConversionWarning: A col
      y = column_or_1d(y, warn=True)

[ ] # Creating a CNN Model
    num_classes = Y_train1.shape[1]
    max_words= X_train1.shape[1]
    model1= Sequential()
    model1.add(Embedding(max_features,100,input_length=max_words))
    # Dropout 0.2% data while training
    model1.add(Dropout(0.2))
    # Adding a convolution layer to the model
    model1.add(Conv1D(64,kernel_size=3,padding='same',activation='relu',strides=1))
    # Performing Maxpool to reduce size of spatial representation
    model1.add(GlobalMaxPooling1D())
    # Adding another input layer
    model1.add(Dense(64,activation='relu'))
    # Dropout 0.2% data while training
    model1.add(Dropout(0.2))
    model1.add(Dense(num_classes,activation='softmax'))
    # Compiling the model
    model1.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

We have used the label encoder for normalization and splitted the data using the train-test-split- method.

Then we have created a CNN model and also applied max_pooling (to reduce size of spatial representation), dropout(dropout rate of 0.2) functions on the model.

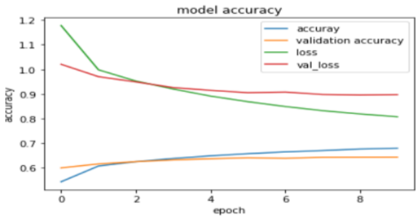
A few more layers are added then we have compiled the model with loss function = ‘categorical-crossentropy’ and optimizer = ‘adam’

```
[ ] # Fitting the model
    history1=model1.fit(X_train, Y_train, validation_data=(X_test, Y_test),epochs=10, batch_size=512, verbose=1)

    /usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slices.py:434: UserWarning: Converting sparse IndexedSlices to
      "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
    Train on 124848 samples, validate on 31212 samples
    Epoch 1/10
    124848/124848 [==============================] - 2s 17us/step - loss: 1.1778 - accuracy: 0.5440 - val_loss: 1.0206 - val_accuracy: 0.6002
    Epoch 2/10
    124848/124848 [==============================] - 2s 16us/step - loss: 0.9981 - accuracy: 0.6078 - val_loss: 0.9703 - val_accuracy: 0.6163
    Epoch 3/10
    124848/124848 [==============================] - 2s 15us/step - loss: 0.9527 - accuracy: 0.6254 - val_loss: 0.9491 - val_accuracy: 0.6254
    Epoch 4/10
    124848/124848 [==============================] - 2s 14us/step - loss: 0.9202 - accuracy: 0.6384 - val_loss: 0.9261 - val_accuracy: 0.6325
    Epoch 5/10
    124848/124848 [==============================] - 2s 16us/step - loss: 0.8915 - accuracy: 0.6493 - val_loss: 0.9149 - val_accuracy: 0.6374
    Epoch 6/10
    124848/124848 [==============================] - 2s 15us/step - loss: 0.8688 - accuracy: 0.6579 - val_loss: 0.9053 - val_accuracy: 0.6410
    Epoch 7/10
    124848/124848 [==============================] - 2s 15us/step - loss: 0.8492 - accuracy: 0.6651 - val_loss: 0.9078 - val_accuracy: 0.6390
    Epoch 8/10
    124848/124848 [==============================] - 2s 15us/step - loss: 0.8324 - accuracy: 0.6704 - val_loss: 0.8980 - val_accuracy: 0.6433
    Epoch 9/10
    124848/124848 [==============================] - 2s 15us/step - loss: 0.8190 - accuracy: 0.6768 - val_loss: 0.8960 - val_accuracy: 0.6438
    Epoch 10/10
    124848/124848 [==============================] - 2s 15us/step - loss: 0.8077 - accuracy: 0.6797 - val_loss: 0.8975 - val_accuracy: 0.6440

[ ] # Plotting acc,val_acc,loss,val_loss
    import matplotlib.pyplot as plt
    plt.plot(history1.history['accuracy'])
    plt.plot(history1.history['val_accuracy'])
    plt.plot(history1.history['loss'])
    plt.plot(history1.history['val_loss'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['accuray', 'validation accuracy','loss','val_loss'])
    plt.show()
```

The accuracy for this model is 67.97% and the validation_accuracy is 64.40%. Then, we have plotted a graph for accuracy, validation_Accuracy, loss, validation_loss.




```
[196] # Prediction
      y_predicted=model1.predict_classes(X_test1[:1])
      print(y_predicted[0]," PREDICTED LABEL")

      2  PREDICTED LABEL

[197] # Reading the file from drive
      file = pa.read_csv('/content/drive/My Drive/Lab2/Datasets/sampleSubmission.csv',sep=',')
      print(file['Sentiment'].iloc[0]," ACTUAL LABEL")

      2  ACTUAL LABEL
```

Then we have predicted the sentiment for a sentence using the model created and got the label predicted as 2

Then we have tested it by reading the data from samplesubmission.csv and found that the predicted label and the actual label are equal

5.

Code and Outputs:

```
[ ] # Connecting to Google Drive
    from google.colab import drive
    drive.mount('/content/drive/')

    Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8q

    Enter your authorization code:
    ..........
    Mounted at /content/drive/

[78] # Importing the required libraries
     import pandas as pa
     from keras.preprocessing.text import Tokenizer
     from keras.preprocessing.sequence import pad_sequences
     from keras.models import Sequential
     from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D, Dropout, Conv1D, GlobalMaxPooling1D
     from sklearn.model_selection import train_test_split
     from keras.utils.np_utils import to_categorical
     import re
     from sklearn.preprocessing import LabelEncoder
     import matplotlib.pyplot as plt
     from keras.optimizers import adam

[79] # Reading the train.tsv into train_data1
     train_data1 = pa.read_csv("/content/drive/My Drive/Lab2/Datasets/train.tsv",sep="\t")
     # Reading the test.tsv into test.tsv
     test_data1 = pa.read_csv("/content/drive/My Drive/Lab2/Datasets/test.tsv",sep="\t")
     # Printing the shape of the datasets
     print(train_data1.shape)
     train_data1.head
     print(test_data1.shape)
     test_data1.head
     # Dropping the unwanted columns
     train_data1 = train_data1.drop(columns=['PhraseId', 'SentenceId'])
     # Removing the non-alphabetic characters
```

First, we have connected the Google Colab with Google drive and then we have imported all the required libraries.

Then, we have read the 'train.tsv', 'test.tsv' files present in the google drive into train_data1 and test_data1 variable's respectively.

Then we have printed the shape and head of both the train and test files.

```
[79] # Removing the non-alphabetic characters
     train_data1['Phrase'] = train_data1['Phrase'].apply(lambda x: re.sub('[^a-zA-z0-9\s]', '', x.lower()))
     test_data1 = test_data1.drop(columns=['PhraseId', 'SentenceId'])
     test_data1['Phrase'] = test_data1['Phrase'].apply(lambda x: re.sub('[^a-zA-z0-9\s]', '', x.lower()))

     (156060, 4)
     (66292, 3)

[80] # Taking the target column and deopping it from the training data
     label1=train_data1[['Sentiment']]
     train_data1=train_data1.drop(columns=['Sentiment'])

[122] # Tokenization on train data
      max_feature1 = 4000
      tokenizer = Tokenizer(num_words=max_feature1, split=' ')
      tokenizer.fit_on_texts(train_data1['Phrase'].values)
      X_train1 = tokenizer.texts_to_sequences(train_data1['Phrase'].values)
      X_train1 = pad_sequences(X_train1)

[123] # Tokenization on test data
      max_feature2 = 2000
      tokenizer = Tokenizer(num_words=max_feature2, split=' ')
      tokenizer.fit_on_texts(test_data1['Phrase'].values)
      X_test1 = tokenizer.texts_to_sequences(test_data1['Phrase'].values)
      X_test1 = pad_sequences(X_test1)

[124] X_train1.shape

      (156060, 46)

[125] X_test1.shape

      (66292, 46)
```

Then, we have converted the content in 'Phrase' column to lower case by using the lambda function and dropped the unnecessary columns(PhraseId , SentenceId) and then dropped the target column Phrase from the training data and stored it into label1 variable.

Then we have applied tokenization on training and test data for converting the text into words and performed padding in order to obtain strings of equal length for tokens.

After performing padding operation we have printed the shape of train and test data.

```
[126] # Performing train test and split
      label_encoder = LabelEncoder()
      integer_encoded = label_encoder.fit_transform(label1)
      Y_train1 = to_categorical(integer_encoded)
      X_train, X_test, Y_train, Y_test = train_test_split(X_train1, Y_train1, test_size=0.2, random_state=10)
      print(X_train.shape,Y_train.shape)
      print(X_test.shape,Y_test.shape)

      (124848, 46) (124848, 5)
      (31212, 46) (31212, 5)
      /usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/_label.py:251: DataConversionWarning: A colu
        y = column_or_1d(y, warn=True)

[127] # Creating a LSTM Model
      embed_dim = 64
      lstm_out = 32
      model1 = Sequential()
      model1.add(Embedding(13734, embed_dim, input_length = X_train1.shape[1]))
      model1.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
      model1.add(Dense(Y_train1.shape[1],activation='softmax'))
      model1.compile(loss = 'categorical_crossentropy', optimizer='adam',metrics = ['accuracy'])
      print(model1.summary())

      Model: "sequential_16"
      _________________________________________________________________
      Layer (type)                 Output Shape              Param #
      =================================================================
      embedding_16 (Embedding)     (None, 46, 64)            878976
      _________________________________________________________________
      lstm_16 (LSTM)               (None, 32)                12416
      _________________________________________________________________
      dense_15 (Dense)             (None, 5)                 165
      =================================================================
      Total params: 891,557
      Trainable params: 891,557
      Non-trainable params: 0
      _________________________________________________________________
      None
```

We have used the label encoder for normalization and splitted the data using the train-test-split- method.

Then we have created a LSTM model and added an embedding layer.

Then we have compiled the model with loss function = ‘categorical-crossentropy’ and optimizer = ‘adam’

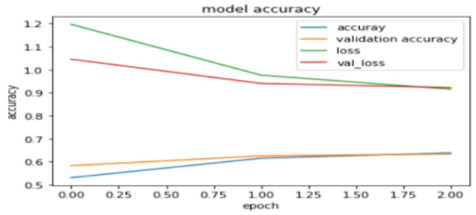
```
[128] # Fitting the model
      history1=model1.fit(X_train, Y_train, validation_data=(X_test, Y_test),epochs=3, batch_size=512, verbose=1)

      /usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slices.py:434: UserWarning: Converting sparse IndexedSlices to a d
        "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
      Train on 124848 samples, validate on 31212 samples
      Epoch 1/3
      124848/124848 [==============================] - 25s 201us/step - loss: 1.1971 - accuracy: 0.5311 - val_loss: 1.0457 - val_accuracy: 0.5834
      Epoch 2/3
      124848/124848 [==============================] - 25s 197us/step - loss: 0.9766 - accuracy: 0.6156 - val_loss: 0.9406 - val_accuracy: 0.6258
      Epoch 3/3
      124848/124848 [==============================] - 24s 196us/step - loss: 0.9153 - accuracy: 0.6388 - val_loss: 0.9226 - val_accuracy: 0.6340

[129] # Plotting acc,val_acc,loss,val_loss
      import matplotlib.pyplot as plt
      plt.plot(history1.history['accuracy'])
      plt.plot(history1.history['val_accuracy'])
      plt.plot(history1.history['loss'])
      plt.plot(history1.history['val_loss'])
      plt.title('model accuracy')
      plt.ylabel('accuracy')
      plt.xlabel('epoch')
      plt.legend(['accuray', 'validation accuracy','loss','val_loss'])
      plt.show()
```

The accuracy for this model is 63.88% and the validation_accuracy is 63.40%. Then, we have plotted a graph for accuracy, validation_Accuracy, loss, validation_loss.

[129]




```
[130] y_predicted=model1.predict_classes(X_test1[:1])
      print(y_predicted[0]," PREDICTED LABEL")
```

```
2  PREDICTED LABEL
```

```
[131] file = pa.read_csv('/content/drive/My Drive/Lab2/Datasets/sampleSubmission.csv',sep=',')
      print(file['Sentiment'].iloc[0]," ACTUAL LABEL")
```

```
2  ACTUAL LABEL
```

6.

Code and Outputs:

We can observe from the models in Question4(CNN) and Question5(LSTM) that the accuracy of CNN model is little more when compared with the LSTM model

(6) Tuning the parameters to achieve good accuracy for CNN Model

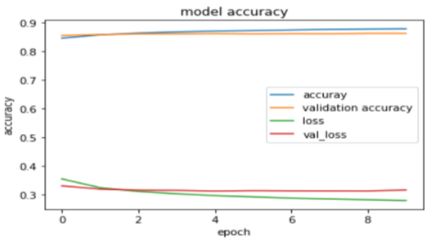
```
[ ] # Creating a CNN Model with learning rate of 0.01
    model2= Sequential()
    model2.add(Embedding(max_features,100,input_length=max_words))
    # Dropout 0.2% data while training
    model2.add(Dropout(0.2))
    # Adding a convolution layer to the model
    model2.add(Conv1D(64,kernel_size=3,padding='same',activation='relu',strides=1))
    # Performing Maxpool to reduce size of spatial representation
    model2.add(GlobalMaxPooling1D())
    # Adding another input layer
    model2.add(Dense(128,activation='relu'))
    # Dropout 0.2% data while training
    model2.add(Dropout(0.2))
    model2.add(Dense(num_classes,activation='softmax'))
    # Compiling the model
    model2.compile(loss='binary_crossentropy',optimizer=adam(lr=0.001),metrics=['accuracy'])
```

```
[ ] # Fitting the model
    history2=model2.fit(X_train, Y_train, validation_data=(X_test, Y_test),epochs=10, batch_size=50, verbose=1)
```

```
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slices.py:434: UserWarning: Converting sparse IndexedSlices to a 
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
Train on 124848 samples, validate on 31212 samples
Epoch 1/10
124848/124848 [==============================] - 16s 124us/step - loss: 0.3538 - accuracy: 0.8449 - val_loss: 0.3296 - val_accuracy: 0.8541
Epoch 2/10
124848/124848 [==============================] - 15s 119us/step - loss: 0.3232 - accuracy: 0.8566 - val_loss: 0.3183 - val_accuracy: 0.8573
Epoch 3/10
124848/124848 [==============================] - 15s 119us/step - loss: 0.3103 - accuracy: 0.8626 - val_loss: 0.3146 - val_accuracy: 0.8598
Epoch 4/10
124848/124848 [==============================] - 15s 122us/step - loss: 0.3020 - accuracy: 0.8667 - val_loss: 0.3141 - val_accuracy: 0.8595
Epoch 5/10
124848/124848 [==============================] - 15s 121us/step - loss: 0.2956 - accuracy: 0.8695 - val_loss: 0.3116 - val_accuracy: 0.8608
```

```
[ ] 124848/124848 [==============================] - 15s 122us/step - loss: 0.2911 - accuracy: 0.8713 - val_loss: 0.3130 - val_accuracy: 0.8598
    Epoch 7/10
    124848/124848 [==============================] - 15s 120us/step - loss: 0.2868 - accuracy: 0.8735 - val_loss: 0.3123 - val_accuracy: 0.8607
    Epoch 8/10
    124848/124848 [==============================] - 15s 118us/step - loss: 0.2838 - accuracy: 0.8757 - val_loss: 0.3122 - val_accuracy: 0.8603
    Epoch 9/10
    124848/124848 [==============================] - 15s 119us/step - loss: 0.2812 - accuracy: 0.8767 - val_loss: 0.3119 - val_accuracy: 0.8615
    Epoch 10/10
    124848/124848 [==============================] - 15s 120us/step - loss: 0.2784 - accuracy: 0.8776 - val_loss: 0.3153 - val_accuracy: 0.8616
```

```
[ ] # Plotting acc,val_acc,loss,val_loss
    import matplotlib.pyplot as plt
    plt.plot(history2.history['accuracy'])
    plt.plot(history2.history['val_accuracy'])
    plt.plot(history2.history['loss'])
    plt.plot(history2.history['val_loss'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['accuray', 'validation accuracy','loss','val_loss'])
    plt.show()
```



We have taken the CNN model created in Question4 and changed the hyper parameters inorder to get higher accuracy.

We have changed the loss function from categorical-crossentropy to binary-crossentropy and gave learning rate as 0.001 and changed the batch size to 50.

After making this change's to the model we can observer that validation accuracy has been increased from 64.40 % to 86.16 %.

(6) Tuning the parameters to achieve good accuracy for LSTM Model

```
[132] # Creating a LSTM Model
      embed_dim = 128
      lstm_out = 64
      model2 = Sequential()
      model2.add(Embedding(13734, embed_dim, input_length = X_train1.shape[1]))
      model2.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
      model2.add(Dense(Y_train1.shape[1],activation='softmax'))
      model2.compile(loss = 'binary_crossentropy', optimizer=adam(lr=0.01),metrics = ['accuracy'])
      print(model2.summary())
```

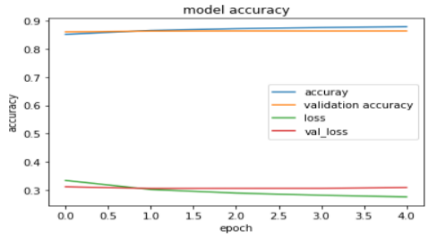
```
Model: "sequential_17"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_17 (Embedding)     (None, 46, 128)           1757952
_________________________________________________________________
lstm_17 (LSTM)               (None, 64)                49408
_________________________________________________________________
dense_16 (Dense)             (None, 5)                 325
=================================================================
Total params: 1,807,685
Trainable params: 1,807,685
Non-trainable params: 0
_________________________________________________________________
None
```

```
[133] # Fitting the model
      history2=model2.fit(X_train, Y_train, validation_data=(X_test, Y_test),epochs=5, batch_size=256, verbose=1)
```

```
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slices.py:434: UserWarning: Converting sparse IndexedSlices to a
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
Train on 124848 samples, validate on 31212 samples
Epoch 1/5
124848/124848 [==============================] - 50s 399us/step - loss: 0.3341 - accuracy: 0.8512 - val_loss: 0.3116 - val_accuracy: 0.8600
```

```
[133] Epoch 2/5
124848/124848 [==============================] - 49s 396us/step - loss: 0.3020 - accuracy: 0.8650 - val_loss: 0.3057 - val_accuracy: 0.8628
Epoch 3/5
124848/124848 [==============================] - 49s 396us/step - loss: 0.2893 - accuracy: 0.8714 - val_loss: 0.3060 - val_accuracy: 0.8632
Epoch 4/5
124848/124848 [==============================] - 49s 390us/step - loss: 0.2815 - accuracy: 0.8754 - val_loss: 0.3061 - val_accuracy: 0.8629
Epoch 5/5
124848/124848 [==============================] - 49s 389us/step - loss: 0.2759 - accuracy: 0.8780 - val_loss: 0.3091 - val_accuracy: 0.8631
```

```python
import matplotlib.pyplot as plt
plt.plot(history2.history['accuracy'])
plt.plot(history2.history['val_accuracy'])
plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['accuray', 'validation accuracy','loss','val_loss'])
plt.show()
```



We have taken the LSTM model created in Question5 and changed the hyper parameters inorder to get higher accuracy.

We have changed the loss function from categorical-crossentropy to binary-crossentropy and gave learning rate as 0.01 and changed the batch size to 256.

After making this change's to the model we can observer that validation accuracy has been increased from 63.40 % to 86.31 %.

7.

Code and Outputs:

Initially, we have downloaded the MNIST dataset and by using AutoEncoders we have encoded and decoded the images.

We then imported the required libraries and we loaded the data from keras.dataset library.

Using Adadelta as optimizer and binary_crossentrophy for loss we have compiled the model then to represent the input we use encoding and for reconstruction we use decoding.

```
+ Code  + Text

[ ] from keras.layers import Input, Dense
    from keras.models import Model
    from keras.callbacks import TensorBoard
    from keras.datasets import fashion_mnist
    import numpy as np
    from keras.datasets import mnist
    import matplotlib.pyplot as plt

    Using TensorFlow backend.

[ ] # encoded representation size
    encoding_dimmensions = 32

[ ] # input image placeholder
    input_image = Input(shape=(784,))
    # Encoded representation of the input
    encoded = Dense(encoding_dim, activation='relu')(input_image)
    # Loss reconstruction of the input
    decoded = Dense(784, activation='sigmoid')(encoded)
    # Maps an input to its reconstruction
    autoencoder = Model(input_image, decoded)

[ ] # Maps an input to its encoded representation
    encoder = Model(input_image, encoded)
    # create a image placeholder for an encoded input
    encoded_input = Input(shape=(encoding_dimmensions,))
    # retrieve the last layer of the autoencoder
    decoder_layer = autoencoder.layers[-1]
    # The decoder model
    decoder = Model(encoded_input, decoder_layer(encoded_input))

[ ] autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy', metrics=['accuracy'])

[ ] (x_train, _), (x_test, _) = mnist.load_data()
    x_train = x_train.astype('float32') / 255.
    x_test = x_test.astype('float32') / 255.
    x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
    x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

[ ] # Noise
    noise_factor = 0.5
    x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.shape)
    x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)
```

We have then fitted the model with 20 epochs and batch size of 256 and acquired an accuracy of 81.04%

```
+ Code  + Text

[ ] tensorboard = TensorBoard(log_dir='2', histogram_freq=0, write_graph=True, write_images=False)
    history = autoencoder.fit(x_train_noisy, x_train,
                              epochs=20,
                              batch_size=256,
                              shuffle=True,
                              validation_data=(x_test_noisy, x_test_noisy), callbacks=[tensorboard])

    Train on 60000 samples, validate on 10000 samples
    Epoch 1/20
    60000/60000 [==============================] - 3s 51us/step - loss: 0.1391 - accuracy: 0.8088 - val_loss: 0.0892 - val_accuracy: 0.0000e+00
    Epoch 2/20
    60000/60000 [==============================] - 3s 47us/step - loss: 0.1382 - accuracy: 0.8090 - val_loss: 0.0873 - val_accuracy: 0.0000e+00
    Epoch 3/20
    60000/60000 [==============================] - 3s 48us/step - loss: 0.1374 - accuracy: 0.8091 - val_loss: 0.0855 - val_accuracy: 0.0000e+00
    Epoch 4/20
    60000/60000 [==============================] - 3s 51us/step - loss: 0.1366 - accuracy: 0.8092 - val_loss: 0.0839 - val_accuracy: 0.0000e+00
    Epoch 5/20
    60000/60000 [==============================] - 3s 50us/step - loss: 0.1359 - accuracy: 0.8094 - val_loss: 0.0823 - val_accuracy: 0.0000e+00
    Epoch 6/20
    60000/60000 [==============================] - 3s 50us/step - loss: 0.1353 - accuracy: 0.8095 - val_loss: 0.0808 - val_accuracy: 0.0000e+00
    Epoch 7/20
    60000/60000 [==============================] - 3s 50us/step - loss: 0.1347 - accuracy: 0.8096 - val_loss: 0.0793 - val_accuracy: 0.0000e+00
    Epoch 8/20
    60000/60000 [==============================] - 3s 50us/step - loss: 0.1341 - accuracy: 0.8097 - val_loss: 0.0780 - val_accuracy: 0.0000e+00
    Epoch 9/20
    60000/60000 [==============================] - 3s 50us/step - loss: 0.1335 - accuracy: 0.8097 - val_loss: 0.0767 - val_accuracy: 0.0000e+00
    Epoch 10/20
    60000/60000 [==============================] - 3s 50us/step - loss: 0.1330 - accuracy: 0.8098 - val_loss: 0.0753 - val_accuracy: 0.0000e+00
    Epoch 11/20
    60000/60000 [==============================] - 3s 50us/step - loss: 0.1326 - accuracy: 0.8099 - val_loss: 0.0742 - val_accuracy: 0.0000e+00
    Epoch 12/20
    60000/60000 [==============================] - 3s 50us/step - loss: 0.1321 - accuracy: 0.8100 - val_loss: 0.0730 - val_accuracy: 0.0000e+00
    Epoch 13/20
    60000/60000 [==============================] - 3s 50us/step - loss: 0.1317 - accuracy: 0.8100 - val_loss: 0.0719 - val_accuracy: 0.0000e+00
    Epoch 14/20
    60000/60000 [==============================] - 3s 51us/step - loss: 0.1313 - accuracy: 0.8101 - val_loss: 0.0709 - val_accuracy: 0.0000e+00
    Epoch 15/20
    60000/60000 [==============================] - 3s 49us/step - loss: 0.1310 - accuracy: 0.8101 - val_loss: 0.0699 - val_accuracy: 0.0000e+00
    Epoch 16/20
    60000/60000 [==============================] - 3s 50us/step - loss: 0.1306 - accuracy: 0.8102 - val_loss: 0.0689 - val_accuracy: 0.0000e+00
    Epoch 17/20
    60000/60000 [==============================] - 3s 53us/step - loss: 0.1303 - accuracy: 0.8102 - val_loss: 0.0680 - val_accuracy: 0.0000e+00
    Epoch 18/20
    60000/60000 [==============================] - 3s 50us/step - loss: 0.1300 - accuracy: 0.8103 - val_loss: 0.0672 - val_accuracy: 0.0000e+00
    Epoch 19/20
    60000/60000 [==============================] - 3s 50us/step - loss: 0.1297 - accuracy: 0.8103 - val_loss: 0.0663 - val_accuracy: 0.0000e+00
    Epoch 20/20
    60000/60000 [==============================] - 3s 48us/step - loss: 0.1294 - accuracy: 0.8104 - val_loss: 0.0655 - val_accuracy: 0.0000e+00

[ ] # encode and decode
    encoded_images = encoder.predict(x_test)
    decoded_images = decoder.predict(encoded_images)
```

Adding noise to the encoded data and then reconstructing the original data.

<u>Evaluation and Discussion</u>:

1. We have successfully implemented linear regression over a sequential model, displayed the graphs on the Tensor Board also plotted them, changed the given hyperparameters and mentioned a brief comment about the changes.

2. We have successfully implemented logistic regression over a sequential model, displayed the graphs on the Tensor Board also plotted them, changed the given hyperparameters and mentioned a brief comment about the changes.

3. Using the Convolution Neural Network (CNN) model we have performed the Image Classification.

4. Using the Convolution Neural Network (CNN) model we have performed the Text Classification.

5. Using the LSTM model we have performed the Text Classification.

6. For the Text Classification, we have provided the best of the above two models.

7. Using Autoencoders, we have performed Encoding and then Decoded the MNIST dataset on list of digits.

Conclusion:

According to the above mentioned objectives, we have performed all the specific programs.