Report for Lab Assignment 2

Anurag Thantharate, Class ID - 24

I. Introduction

The Module-2 of CS5590/490 class provided a good introduction to the foundations of Deep Learning and helped us learn about how to build neural networks, learned about Convolutional networks, Recurrent Neural Networks, LSTM, different activation, optimizers and loss functions. We had an exposure to use and explore datasets from healthcare, real estate, image and text classification. I have used the learnt concept from Module-2 to develop my class Project and solutions for given problem in Lab and ICP's. The Lab 2 assignment focus on exploring and developing Deep learning models to analyze diverse range of datasets which includes image, text using Keras and Python.

II. Objectives

- Create a linear regression sequential model on a data set of our choice, display the results obtained using Tensor board analysis and document any changes impacted from variations in learning rate, batch size, optimizer and activation function.
- Implementation of logistic regression analysis on heart disease related data, normalize the data before feeding to the model, show graphical representation of loss associated to the model using tensor flow analysis and document any changes impacted from variations in learning rate, batch size, optimizer and activation function.
- Develop a CNN model to perform image classification on natural images data set.
- Perform a text classification leading to sentiment analysis on IMDB movie review data using CNN model along with evaluation of the difference in the results based on the different model approaches.
- Perform a text classification leading to sentiment analysis on IMDB movie review data using LSTM model and evaluate difference compared to CNN model.
- Perform autoencoding techniques using the MNIST data set, encode and decode images.

III. Datasets Used:

- Heat Disease and Death rate
- Natural Image classification
- IMDB movie reviews
- MNIIST dataset

IV. Tools/Software:

- PyCharm
- Python3 Interpreter
- Anaconda

V. Approach & Workflow for Problems

- Identify the specific data set to be used for the deep learning model for given problem.
- Load the appropriate data set as needed
- Process, filter and select only the required information from the data.
- Split the data set into training, validation and test data sets to be used on the model.
- Identify the deep learning models to be used for given problem.
- Identify the input, hidden (one or more as considered appropriate) and output layers to be applied to the model defined
- Compile the model using appropriate optimizer, loss and metrics functions
- Using the training data set, fit the model defined
- Evaluate the model by predicting the output for test data and cross-comparing the prediction with actual result of the test data.
- Identify the accuracy and loss associated with the model obtained using the test data
- Visualize the results (accuracy, loss) using graphical means.
- Define the tensor call back model using Tensor Board function
- Using the training data set, fit the model with tensor call back
- Evaluate the model by predicting the output for test data and cross-comparing the prediction with actual result of the test data.
- Identify the accuracy and loss associated with the model obtained from tensor flow
- Visualize the results (accuracy, loss) obtained from the tensor flow as graphical presentation

VI. Concepts:

- Linear regression sequential data analysis
- Logistic regression data analysis
- Image classification using CNN model
- CNN model-based text classification
- LSTM model-based text classification
- Autoencoding resulting in encoding and decoding of images

VII. Source & Results: https://github.com/adtmv7/CS5590-490-Python-Deep-Learning/tree/master/LAB2

==========

Question 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 Tensorboard

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

Workflow/Approaches:

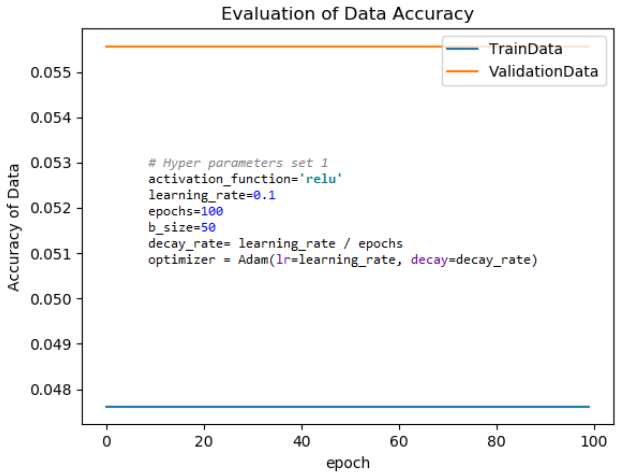
```
# Importing libraries
import pandas as pd
from keras.callbacks import TensorBoard
from keras.optimizers import Adam, SGD
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
from matplotlib import pyplot as plt

# Reading the dataset
deathrate = pd.read_csv('DeathRate.csv')

# Identifying features and predictor variables associated with the data
x = deathrate.iloc[:, 0:13]
y = deathrate.iloc[:, 13]

# Split the data set into training and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=100)
print(x_train.size)
print(x_train.shape)

# Hyper parameters set 1
# activation_function='relu'
# learning_rate=0.1
# epochs=100
# b_size=50
# decay_rate= learning_rate / epochs
# optimizer = Adam(lr=learning_rate, decay=decay_rate)

# Hyper parameters set 2
activation_function="tanh"
learning_rate=0.5
epochs=150
b_size=75
decay_rate= learning_rate / epochs
optimizer = SGD(lr=learning_rate, decay=decay_rate)
```

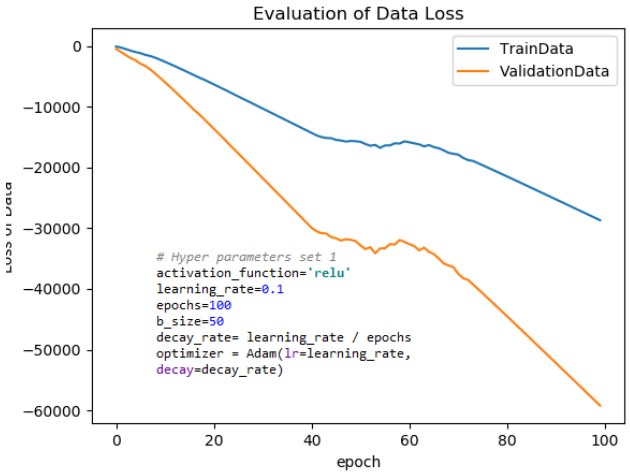
```
# Creating neural network to perform linear regression
# Model Creation
model = Sequential()
# Providing inputs to the hidden layer
model.add(Dense(100, input_dim=13, activation=activation_function))
# Adding hidden layers and activation functions
model.add(Dense(15, activation='relu'))
model.add(Dense(30, activation='relu'))
model.add(Dense(255, activation='tanh'))
# output layer
model.add(Dense(1, activation='sigmoid'))
# Compiling the defined model
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

# Fitting the defined model using the training data set
history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=100, batch_size=b_size, verbose=0, initial_epoch=0)

# Evaluation of the loss and accuracy associated to the test data set
[test_loss, test_acc] = model.evaluate(x_test, y_test)
print("Evaluation result on Test Data : Loss = {}, accuracy = {}".format(test_loss, test_acc))

# Listing components of data present in history
print('The components of data present in history are', history.history.keys())

# Graphical evaluation of accuracy associated with training and validation data
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Evaluation of Data Accuracy')
plt.xlabel('epoch')
plt.ylabel('Accuracy of Data')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()

# Graphical evaluation of loss associated with training and validation data
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('Loss of Data')
plt.title('Evaluation of Data Loss')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()
```

```
# Visualization of the model using tensor board
tbCallBack = TensorBoard(log_dir='./q1_linear_regression', histogram_freq=0, write_graph=True, write_images=True)

# Fitting the model defined using the training data along with validation using test data
history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=10, verbose=1, initial_epoch=0)

# Evaluation of the loss and accuracy associated to the test data set
[test_loss, test_acc] = model.evaluate(x_test, y_test)
print("Evaluation result on Test Data using Tensorflow : Loss = {}, accuracy = {}".format(test_loss, test_acc))

# Listing all the components of data present in history
print('The data components present in history using Tensorflow are', history.history.keys())

# Graphical evaluation of accuracy associated with training and validation data
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Evaluation of Data Accuracy using Tensorflow')
plt.xlabel('epoch')
plt.ylabel('Accuracy of Data')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()

# Graphical evaluation of loss associated with training and validation data
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('Loss of Data')
plt.title('Evaluation of Data Loss using Tensorflow')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()
```

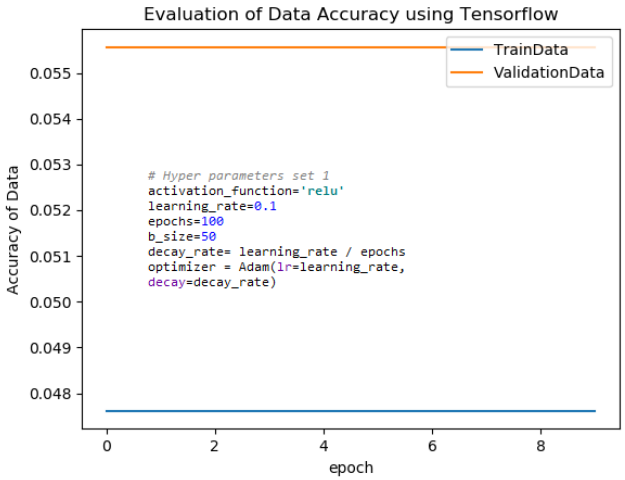
Output: For Hyperset#1

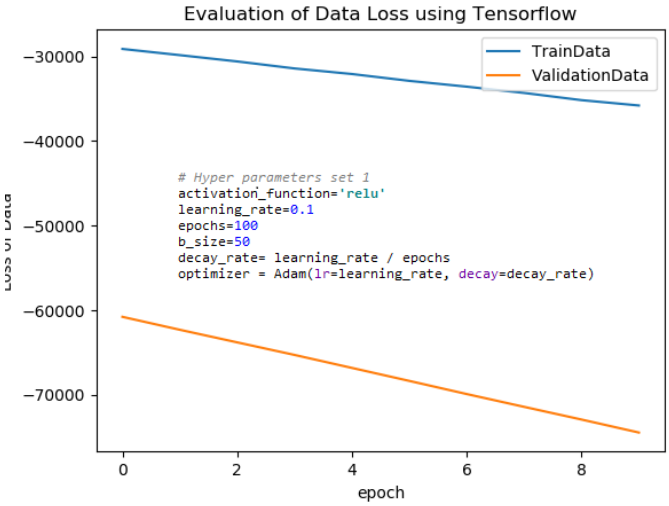
```
C:\ProgramData\Anaconda3\python.exe "C:/Users/1d630534/Documents/Anurag Projects/Spring 2020/Python with DL/La
Using TensorFlow backend.
546
(42, 13)
2020-05-10 19:22:27.145063: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructior
2020-05-10 19:22:27.147053: I tensorflow/core/common_runtime/process_util.cc:147] Creating new thread pool wit

18/18 [==============================] - 0s 55us/step
Evaluation result on Test Data : Loss = -59231.09765625, accuracy = 0.0555555559694767
The components of data present in history are dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
Train on 42 samples, validate on 18 samples
Epoch 1/10
```





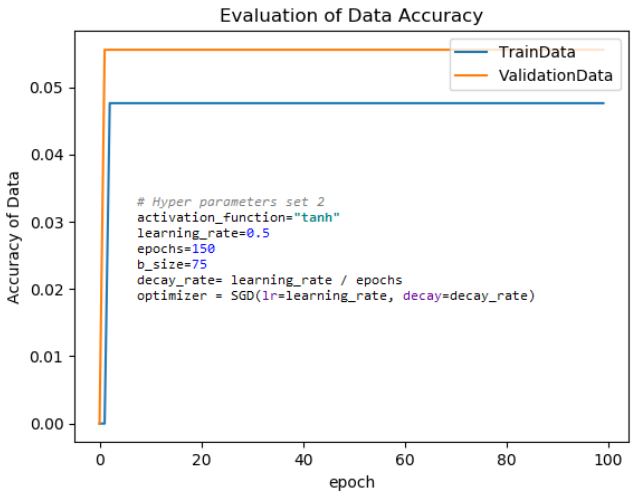

```
32/42 [=====================>........] - ETA: 0s - loss: -41080.7852 - accuracy: 0.0312
42/42 [==============================] - 0s 261us/step - loss: -35811.7617 - accuracy: 0.0476 - val_loss: -74447.2891 - val_accuracy: 0.0556

18/18 [==============================] - 0s 55us/step
Evaluation result on Test Data using Tensorflow : Loss = -74447.2890625, accuracy = 0.0555555559694767
The data components present in history using Tensorflow are dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
```

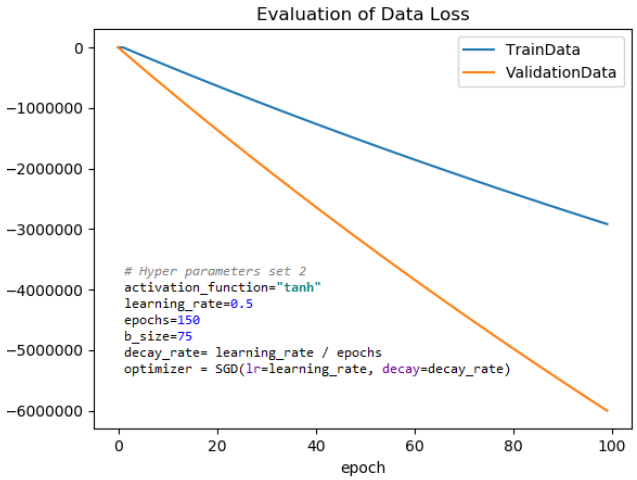




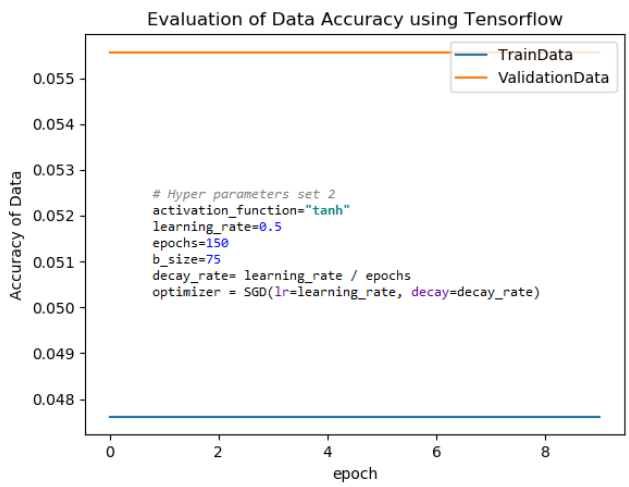
For Hyperset#2:

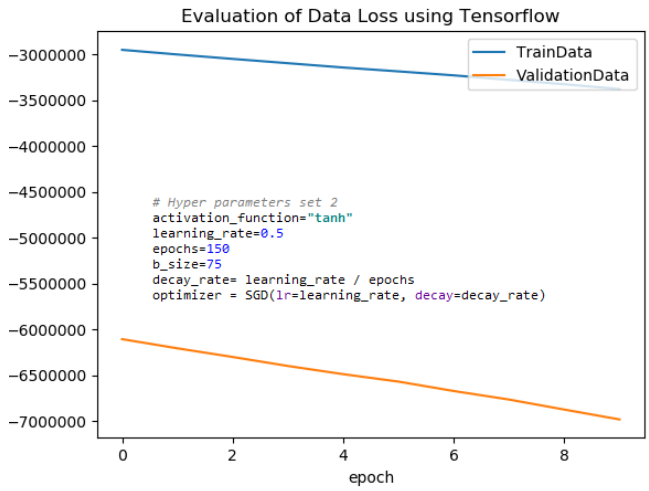
```
C:\ProgramData\Anaconda3\python.exe "C:/Users/1d630534/Documents/Anurag Projects/Spring 2020/Python with DL/I
Using TensorFlow backend.
546
(42, 13)
2020-05-10 19:30:28.713797: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructi
2020-05-10 19:30:28.715742: I tensorflow/core/common_runtime/process_util.cc:147] Creating new thread pool wi

18/18 [==============================] - 0s 55us/step
Evaluation result on Test Data : Loss = -5999840.5, accuracy = 0.0555555559694767
The components of data present in history are dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
Train on 42 samples, validate on 18 samples
```






```
18/18 [==============================] - 0s 55us/step
Evaluation result on Test Data using Tensorflow : Loss = -6979495.0, accuracy = 0.0555555559694767
The data components present in history using Tensorflow are dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
```





Loss decreases by changing parameters on Hyperset#2 for my dataset and model.

Question 2: Implement ***Logistic Regression** on 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

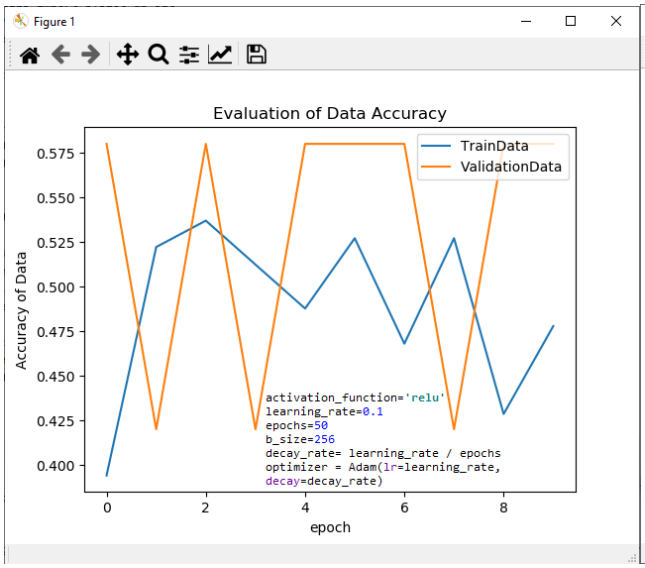
```
# Importing libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.optimizers import Adam, SGD
import numpy as np
from keras.layers import Dense
from matplotlib import pyplot as plt
from sklearn.preprocessing import StandardScaler
from keras.callbacks import TensorBoard

# Reading data
heartData = pd.read_csv('heart.csv')

# Identifying features and predictor associated with the heart data set
x = heartData.iloc[:, 0:13]
y = heartData.iloc[:, 13]

# Split the data set into training and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=42, test_size=.33)

# Normalization of training data set
sc = StandardScaler()
scData = sc.fit(x_train, x_test)
print(scData)

# Hyper parameters set 1
# activation_function='relu'
# learning_rate=0.1
# epochs=50
# b_size=256
# decay_rate= learning_rate / epochs
# optimizer = Adam(lr=learning_rate, decay=decay_rate)

# Hyper parameters set 2
# activation_function='tanh'
# learning_rate=0.3
# epochs=100
# b_size=32
# decay_rate= learning_rate / epochs
# optimizer = SGD(lr=learning_rate, decay=decay_rate)
```

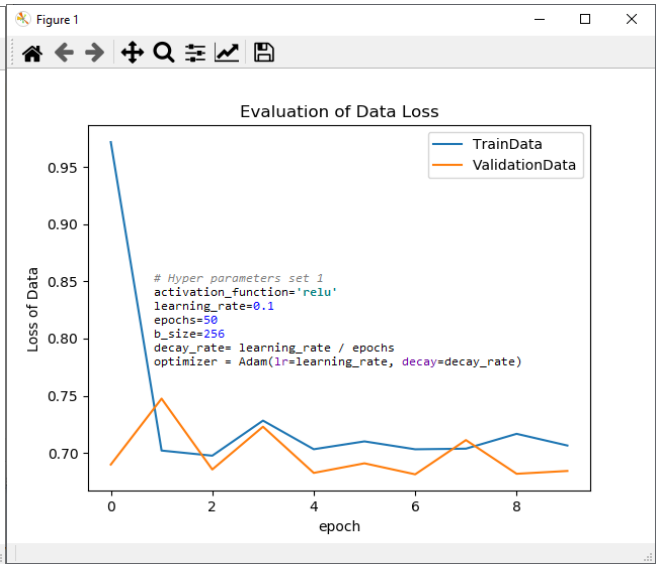
```
# Hyper parameters set 3
activation_function='relu'
learning_rate=0.5
epochs=150
b_size=60
decay_rate= learning_rate / epochs
optimizer = SGD(lr=learning_rate, decay=decay_rate)

# Creating neural network model for evaluation
# Define the model used
nnHeart = Sequential()
# Provide input and neurons for first hidden dense layer
nnHeart.add(Dense(60, input_dim=13, activation=activation_function))
# Adding multiple hidden layers
nnHeart.add(Dense(30, activation='relu'))
nnHeart.add(Dense(15, activation='tanh'))
# Define the output neuron
nnHeart.add(Dense(1, activation='sigmoid'))
# Fitting the neural network model on the training data set
nnHeart.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])

# Fitting the model defined using the training data along with validation using test data
history = nnHeart.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=10, verbose=0, initial_epoch=0)

# Evaluation of the loss and accuracy associated to the test data set
[test_loss, test_acc] = nnHeart.evaluate(x_test, y_test)
print("Evaluation result on Test Data : Loss = {}, accuracy = {}".format(test_loss, test_acc))

# Listing all the components of data present in history
print('The components of data present in history are', history.history.keys())
```

```
# Graphical evaluation of loss associated with training and validation data
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('Loss of Data')
plt.title('Evaluation of Data Loss')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()

# Visualization of the model using tensor board
tbCallBack = TensorBoard(log_dir='./2_logistic_regression', histogram_freq=0, write_graph=True, write_images=True)

# Fitting the model defined using the training data along with validation using test data
history = nnHeart.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=10, verbose=1, initial_epoch=0)

# Evaluation of the loss and accuracy associated to the test data set
[test_loss, test_acc] = nnHeart.evaluate(x_test, y_test)
print("Evaluation result on Test Data using Tensorflow : Loss = {}, accuracy = {}".format(test_loss, test_acc))

# Listing all the components of data present in history
print('The data components present in history using Tensorflow are', history.history.keys())

# Graphical evaluation of accuracy associated with training and validation data
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Evaluation of Data Accuracy using Tensorflow')
plt.xlabel('epoch')
plt.ylabel('Accuracy of Data')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()

# Graphical evaluation of loss associated with training and validation data
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('Loss of Data')
plt.title('Evaluation of Data Loss using Tensorflow')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()
```

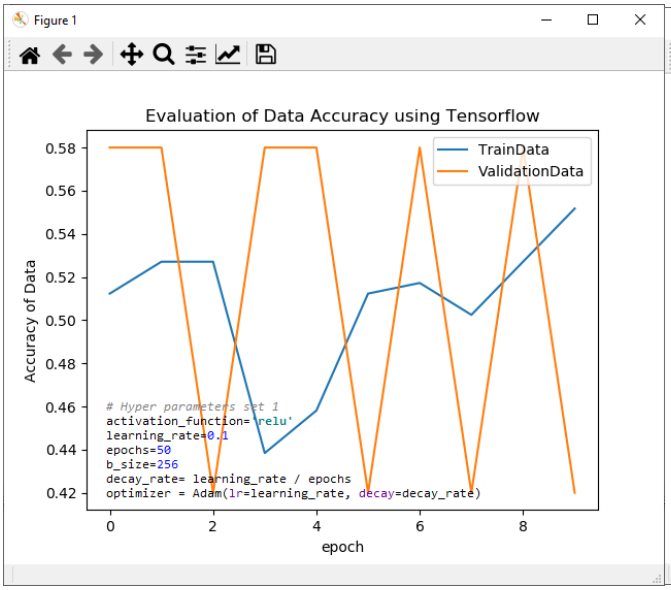
Output:

```
2_logistic_regression
C:\ProgramData\Anaconda3\python.exe "C:/Users/1d630534/Documents/Anurag Projects/Spring 2020/Python with DL/Lab2/2_logistic_regression.py"
Using TensorFlow backend.
StandardScaler(copy=True, with_mean=True, with_std=True)
2020-05-03 13:52:11.621290: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX AVX2
2020-05-03 13:52:11.622790: I tensorflow/core/common_runtime/process_util.cc:147] Creating new thread pool with default inter op setting: 2. Tune using inter_op_parallelism_threads for best performance.

 32/100 [========>.....................] - ETA: 0s
100/100 [==============================] - 0s 30us/step
Evaluation result on Test Data : Loss = 0.6841667342185974, accuracy = 0.5799999833106995
The components of data present in history are dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
Train on 203 samples, validate on 100 samples
Epoch 1/10
```



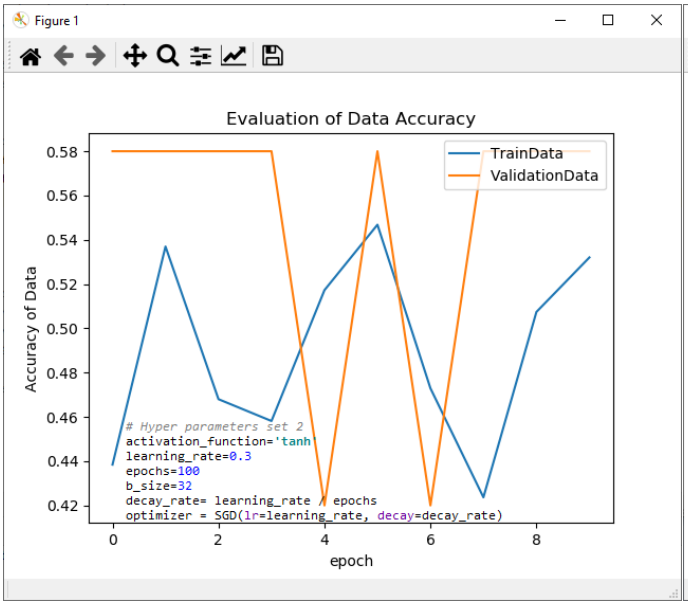



```
 32/100 [========>.....................] - ETA: 0s
100/100 [==============================] - 0s 30us/step
Evaluation result on Test Data using Tensorflow : Loss = 0.7039557027816773, accuracy = 0.41999998688697815
The data components present in history using Tensorflow are dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])

Process finished with exit code 0
```

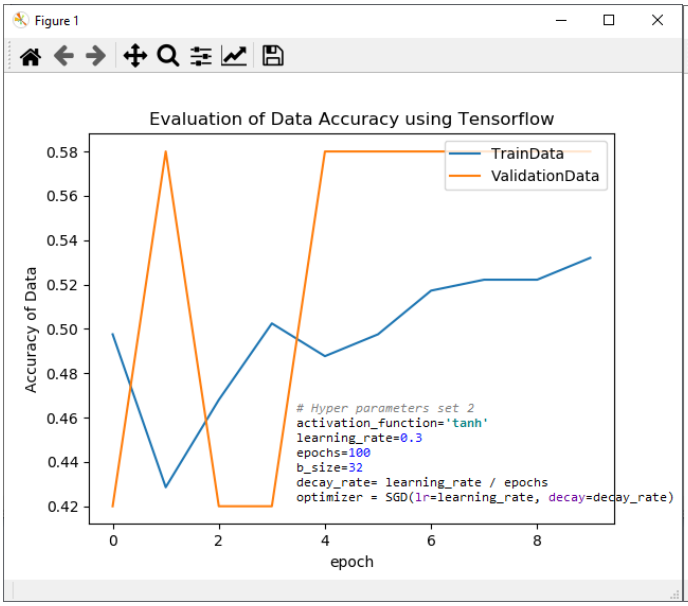


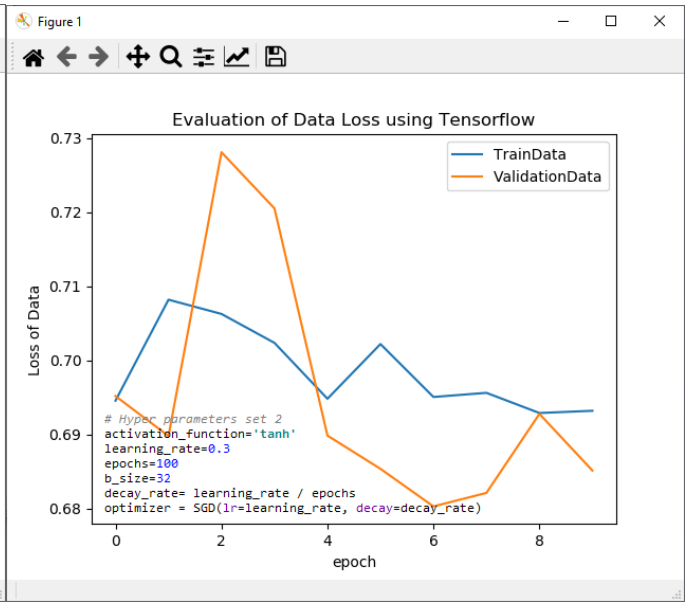
Hyperset#2:

```
Using TensorFlow backend.
StandardScaler(copy=True, with_mean=True, with_std=True)
2020-05-03 13:59:00.418097: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this Tens
2020-05-03 13:59:00.419509: I tensorflow/core/common_runtime/process_util.cc:147] Creating new thread pool with default inter

 32/100 [========>.....................] - ETA: 0s
100/100 [==============================] - 0s 20us/step
Evaluation result on Test Data : Loss = 0.6864550232887268, accuracy = 0.5799999833106995
The components of data present in history are dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
Train on 203 samples, validate on 100 samples
```



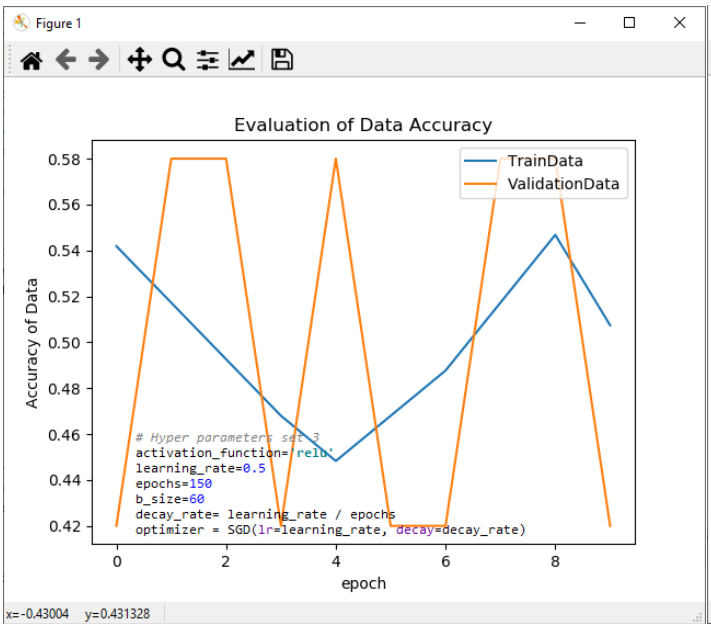

```
 32/100 [========>.....................] - ETA: 0s
100/100 [==============================] - 0s 30us/step
Evaluation result on Test Data using Tensorflow : Loss = 0.6851576972007751, accuracy = 0.5799999833106995
The data components present in history using Tensorflow are dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
```



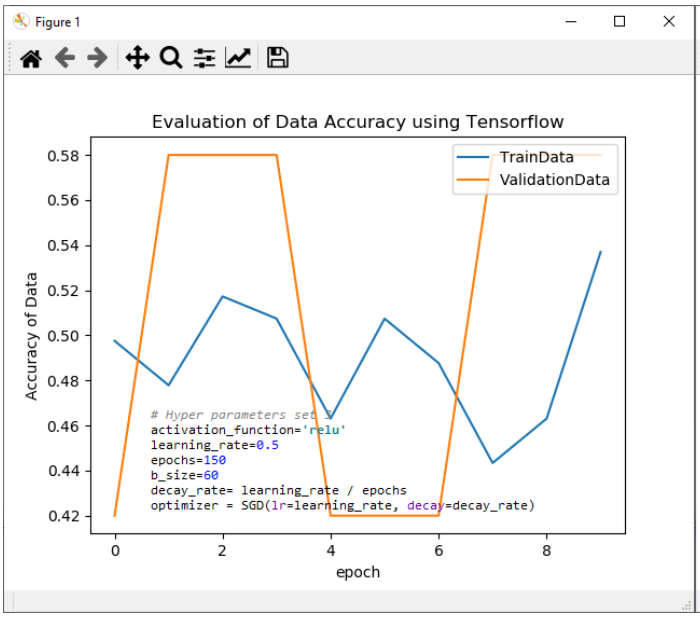


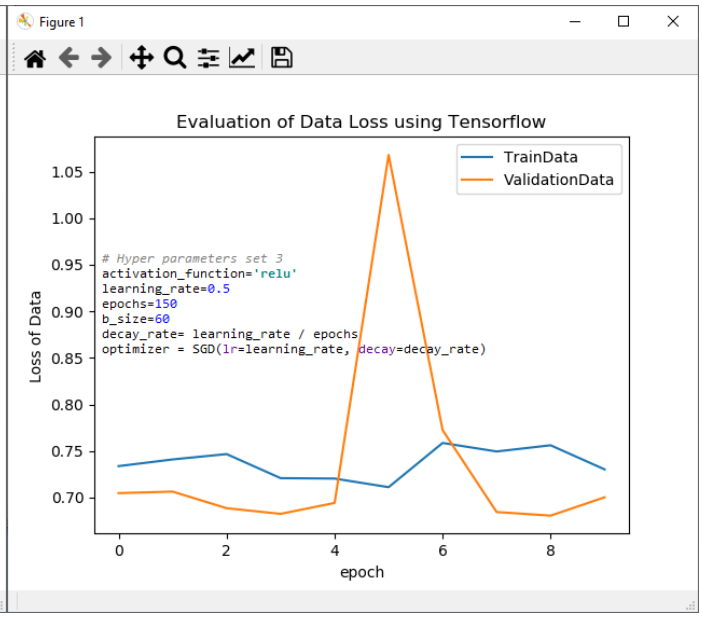
Hyperset#3:

```
C:\ProgramData\Anaconda3\python.exe "C:/Users/ld630534/Documents/Anurag Projects/Spring 2020/Python with DL/Lab2/2_logistic_regression.py"
Using TensorFlow backend.
StandardScaler(copy=True, with_mean=True, with_std=True)
2020-05-03 14:03:53.242834: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not compiled to
2020-05-03 14:03:53.244800: I tensorflow/core/common_runtime/process_util.cc:147] Creating new thread pool with default inter op setting: 2. Tune using inter_op

 32/100 [========>.....................] - ETA: 0s
100/100 [==============================] - 0s 30us/step
Evaluation result on Test Data : Loss = 0.9435591888427735, accuracy = 0.41999998688697815
The components of data present in history are dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
```




```
 32/203 [===>..........................] - ETA: 0s - loss: 0.6638 - accuracy: 0.6250
203/203 [==============================] - 0s 84us/step - loss: 0.7303 - accuracy: 0.5369 - val_loss: 0.7003 - val_accuracy: 0.5800

 32/100 [========>.....................] - ETA: 0s
100/100 [==============================] - 0s 40us/step
Evaluation result on Test Data using Tensorflow : Loss = 0.700319983959198, accuracy = 0.5799999833106995
The data components present in history using Tensorflow are dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
```





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

Output:

```
# Import appropriate libraries
import numpy as np
import os
import cv2
import matplotlib.pyplot as plt
from keras.callbacks import TensorBoard
from keras.models import Sequential
from keras.layers import Dense, MaxPool2D
from keras.layers import Dropout
from keras.layers import Flatten
from keras.constraints import maxnorm
from keras.optimizers import SGD
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils, to_categorical
from keras import backend as K
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Load data of natural images
labels = os.listdir('natural_images/')
x = [] # Feature predictor variables array
y = [] # Target variables array

for label in labels:
    pics = os.listdir('natural_images/{}/'.format(label))
    for pic in pics:
        image = cv2.imread('natural_images/{}/{}'.format(label, pic))
        image_resized = cv2.resize(image, (32, 32))
        x.append(np.array(image_resized))
        y.append(label)

x = np.array(x)
y = np.array(y)

x = x.astype('float32') / 255
enc = LabelEncoder().fit(y)
y_encoded = enc.transform(y)
y = to_categorical(y_encoded)

# Splitting data set into training and test data set
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33)

# Build the neural network model
# Define the model being built
model = Sequential()
# Convolutional layer
model.add(Conv2D(filters=65, kernel_size=(5, 5), activation='relu', input_shape=x_train.shape[1:]))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters=84, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
# Flatten layer
model.add(Flatten())
model.add(Dense(150, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(20, activation='softmax'))
# Compile the model defined
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

```
# Fit the model defined on the training data set
history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=5, batch_size=2000)

# Final evaluation of the model using the test data set
scores = model.evaluate(x_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))

[test_loss, test_acc] = model.evaluate(x_test, y_test)
print("Evaluation result on Test Data : Loss = {}, accuracy = {}".format(test_loss, test_acc))

# Listing all the components of data present in history
print('The data components present in history are', history.history.keys())

# Graphical evaluation of accuracy associated with training and validation data
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Evaluation of Data Accuracy')
plt.xlabel('epoch')
plt.ylabel('Accuracy of Data')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()

# Graphical evaluation of loss associated with training and validation data
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('Loss of Data')
plt.title('Evaluation of Data Loss')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()


# Visualization of the model using tensor board
tbCallBack = TensorBoard(log_dir='./lab2_1', histogram_freq=0, write_graph=True, write_images=True)

# Fitting the model defined using the training data along with validation using test data
history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=10, verbose=0, initial_epoch=0)

# Evaluation of the loss and accuracy associated to the test data set
[test_loss, test_acc] = model.evaluate(x_test, y_test)
print("Evaluation result on Test Data using Tensorflow : Loss = {}, accuracy = {}".format(test_loss, test_acc))

# Listing all the components of data present in history
print('The data components present in history using Tensorflow are', history.history.keys())

# Graphical evaluation of accuracy associated with training and validation data
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Evaluation of Data Accuracy using Tensorflow')
plt.xlabel('epoch')
plt.ylabel('Accuracy of Data')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()

# Graphical evaluation of loss associated with training and validation data
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('Loss of Data')
plt.title('Evaluation of Data Loss using Tensorflow')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()
```

```
C:\ProgramData\Anaconda3\python.exe "C:/Users/1d630534/Documents/Anurag Projects/Spring 2020/Python with DL/Lab2/3_image_classification.p
Using TensorFlow backend.
2020-05-03 20:13:42.638963: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow bina
2020-05-03 20:13:42.640553: I tensorflow/core/common_runtime/process_util.cc:147] Creating new thread pool with default inter op setting:
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        2432
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 12, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               590080
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 2056
=================================================================
Total params: 613,064
Trainable params: 613,064
Non-trainable params: 0
_________________________________________________________________
None
Train on 4622 samples, validate on 2277 samples
Epoch 1/5

Epoch 5/5

 2000/4622 [===========>..................] - ETA: 1s - loss: 1.2218 - accuracy: 0.5850
 4000/4622 [========================>.....] - ETA: 0s - loss: 1.1911 - accuracy: 0.5918
 4622/4622 [==============================] - 3s 662us/step - loss: 1.1800 - accuracy: 0.5954 - val_loss: 0.9829 - val_accuracy: 0.6649
Accuracy: 66.49%

   32/2277 [..............................] - ETA: 0s
  192/2277 [=>............................] - ETA: 0s
  416/2277 [====>.........................] - ETA: 0s
  608/2277 [=======>......................] - ETA: 0s
  832/2277 [=========>....................] - ETA: 0s
 1056/2277 [============>.................] - ETA: 0s
 1248/2277 [===============>..............] - ETA: 0s
 1440/2277 [=================>............] - ETA: 0s
 1632/2277 [====================>.........] - ETA: 0s
 1824/2277 [========================>.....] - ETA: 0s
 2016/2277 [=========================>....] - ETA: 0s
 2208/2277 [============================>.] - ETA: 0s
 2277/2277 [==============================] - 1s 294us/step
 Evaluation result on Test Data : Loss = 0.9828505001415744, accuracy = 0.6649099588394165
 The data components present in history are dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])

   32/2277 [..............................] - ETA: 0s
  224/2277 [=>............................] - ETA: 0s
  480/2277 [=====>........................] - ETA: 0s
  672/2277 [=======>......................] - ETA: 0s
  896/2277 [==========>...................] - ETA: 0s
 1088/2277 [=============>................] - ETA: 0s
 1344/2277 [================>.............] - ETA: 0s
 1568/2277 [===================>..........] - ETA: 0s
 1792/2277 [======================>.......] - ETA: 0s
 2016/2277 [=========================>....] - ETA: 0s
 2277/2277 [==============================] - 1s 250us/step
 Evaluation result on Test Data using Tensorflow : Loss = 0.24367504618279914, accuracy = 0.9082125425338745
 The data components present in history using Tensorflow are dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
```

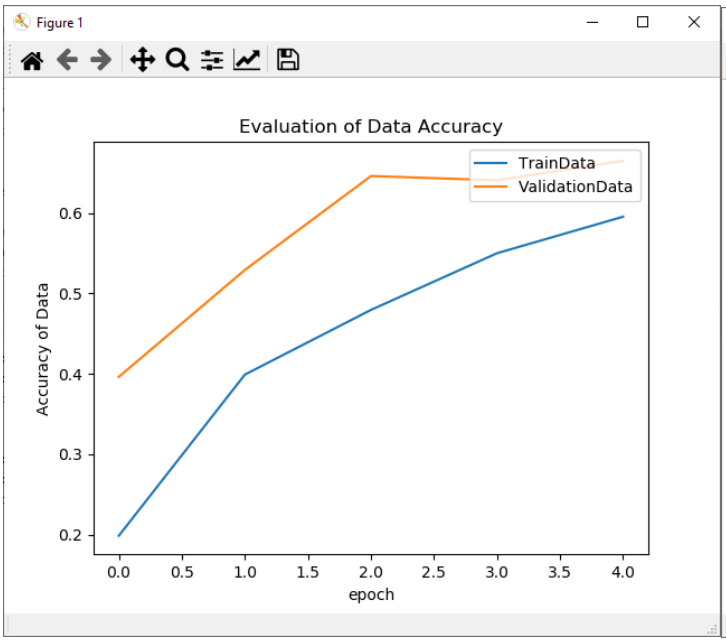


Figure 1
Evaluation of Data Accuracy
TrainData
ValidationData
Accuracy of Data
epoch


Figure 1
Evaluation of Data Loss
TrainData
ValidationData
Loss of Data
epoch

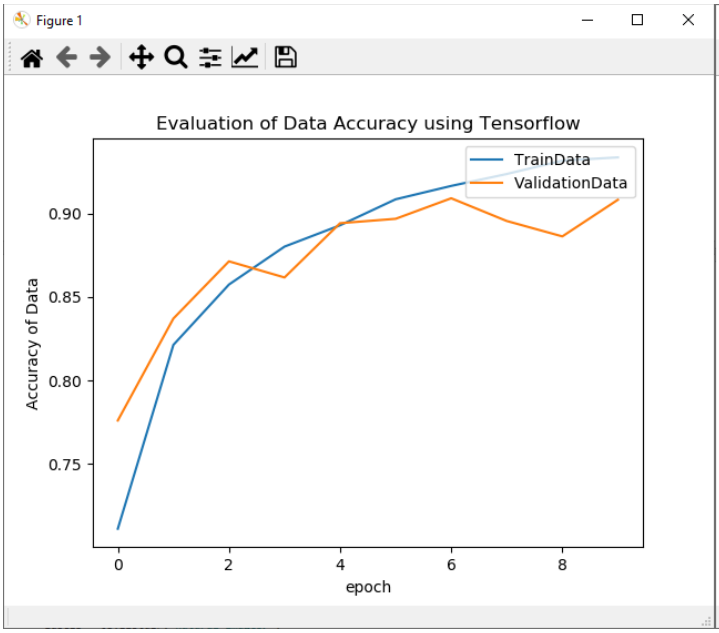


Figure 1
Evaluation of Data Accuracy using Tensorflow
TrainData
ValidationData
Accuracy of Data
epoch

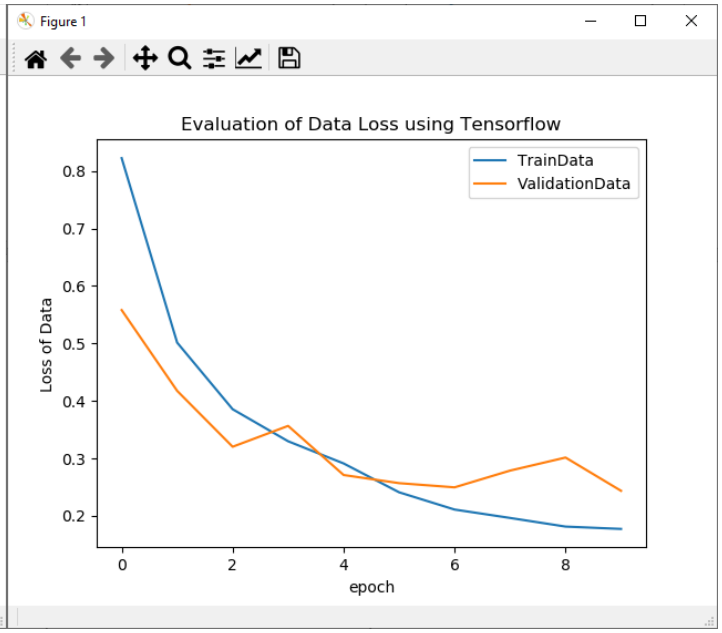


Figure 1
Evaluation of Data Loss using Tensorflow
TrainData
ValidationData
Loss of Data
epoch

Question 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.

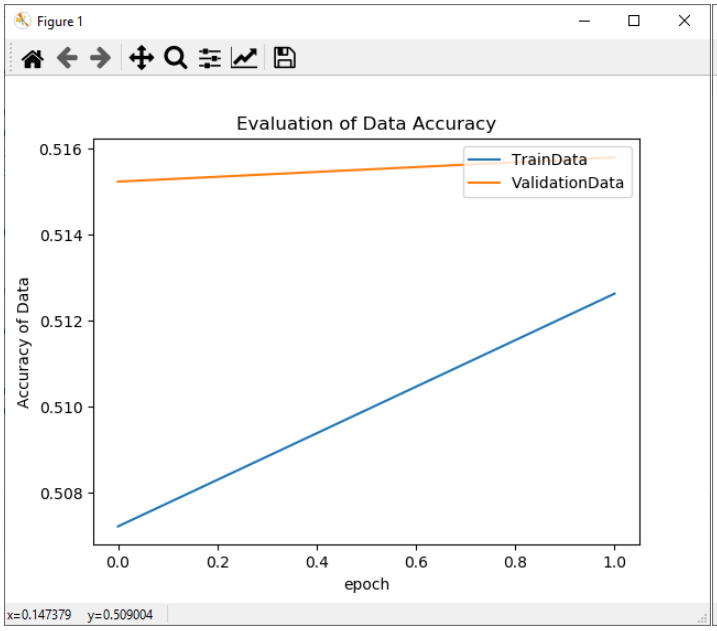
```
# Import libraries
import re
import pandas as pd
from keras import Sequential
from keras.callbacks import TensorBoard
from keras.constraints import maxnorm
from keras.layers import Embedding, Conv1D, Dropout, MaxPooling1D, Flatten, Dense
from keras.optimizers import SGD
from keras.utils import to_categorical
from keras_preprocessing.sequence import pad_sequences
from keras_preprocessing.text import Tokenizer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt

# Loading movie review sentiments data
testMovie_df = pd.read_csv('test.tsv', delimiter='\t', encoding='utf-8')
trainMovie_df = pd.read_csv('train.tsv', delimiter='\t', encoding='utf-8')

# Keeping only the necessary columns - cleaning the data set
trainMovie_df = trainMovie_df.drop(columns=['PhraseId', 'SentenceId'])
testMovie_df = testMovie_df.drop(columns=['PhraseId', 'SentenceId'])

trainMovie_df['Phrase'] = trainMovie_df['Phrase'].apply(lambda x: re.sub('[^a-zA-z0-9\s]', '', x.lower()))
testMovie_df['Phrase'] = testMovie_df['Phrase'].apply(lambda x: re.sub('[^a-zA-z0-9\s]', '', x.lower()))

max_features = 2000
tokenizer = Tokenizer(num_words=max_features, split=' ')
tokenizer.fit_on_texts(trainMovie_df['Phrase'].values)
X_train = tokenizer.texts_to_sequences(trainMovie_df['Phrase'].values)
X_train = pad_sequences(X_train)
```

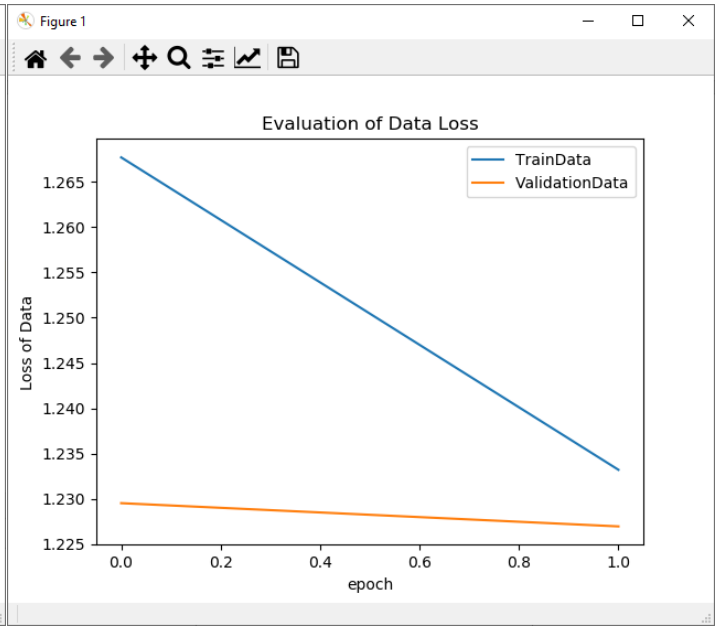
```
tokenizer.fit_on_texts(testMovie_df['Phrase'].values)
X_test = tokenizer.texts_to_sequences(testMovie_df['Phrase'].values)
X_test = pad_sequences(X_train)
print("handing data")

# Creating the model
embed_dim = 128
lstm_out = 196

# Design the model using classification
# Model defined
model = Sequential()
# Input layer of the model for processing
model.add(Embedding(max_features, embed_dim, input_length=X_train.shape[1]))
# Convolutional layer
model.add(Conv1D(128, (5), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(MaxPooling1D(5))
# Flatten layer
model.add(Flatten())
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.5))
# Output layer
model.add(Dense(5, activation='softmax'))
# Compile the model identified
sgd = SGD(lr=0.01, momentum=0.9, decay=0.01 / 15, nesterov=False)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
```

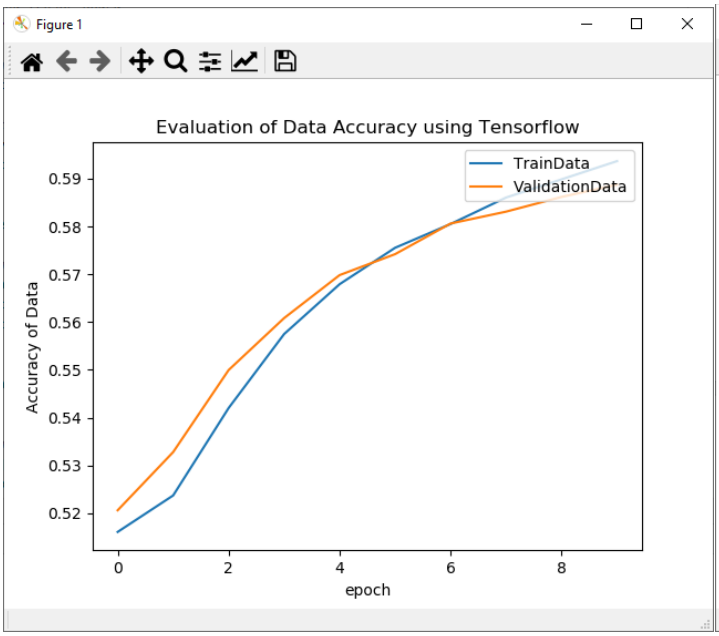
```
# Identify the data into training and test sets
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(trainMovie_df['Sentiment'])
Y_train = to_categorical(integer_encoded)
x_train, x_test, y_train, y_test = train_test_split(X_train, Y_train, test_size=0.25, random_state=30)

# Fitting the model identified using the training data set
history = model.fit(x_train, y_train, epochs=2, batch_size=500, validation_data=(x_test, y_test))

# Evaluation of the results of the model obtained using the test data set
[test_loss, test_acc] = model.evaluate(x_test, y_test)
print("Evaluation result on Test Data : Loss = {}, accuracy = {}".format(test_loss, test_acc))

# Listing all the components of data present in history
print('The data components present in history are', history.history.keys())

# Graphical evaluation of accuracy associated with training and validation data
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Evaluation of Data Accuracy')
plt.xlabel('epoch')
plt.ylabel('Accuracy of Data')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()


# Graphical evaluation of loss associated with training and validation data
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('Loss of Data')
plt.title('Evaluation of Data Loss')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()

# Visualization of the model using tensor board
tbCallBack = TensorBoard(log_dir='./lab2_1', histogram_freq=0, write_graph=True, write_images=True)

# Fitting the model defined using the training data along with validation using test data
history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=10, verbose=0, initial_epoch=0)

# Evaluation of the loss and accuracy associated to the test data set
[test_loss, test_acc] = model.evaluate(x_test, y_test)
print("Evaluation result on Test Data using Tensorflow : Loss = {}, accuracy = {}".format(test_loss, test_acc))

# Listing all the components of data present in history
print('The data components present in history using Tensorflow are', history.history.keys())

# Graphical evaluation of accuracy associated with training and validation data
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Evaluation of Data Accuracy using Tensorflow')
plt.xlabel('epoch')
plt.ylabel('Accuracy of Data')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()
```

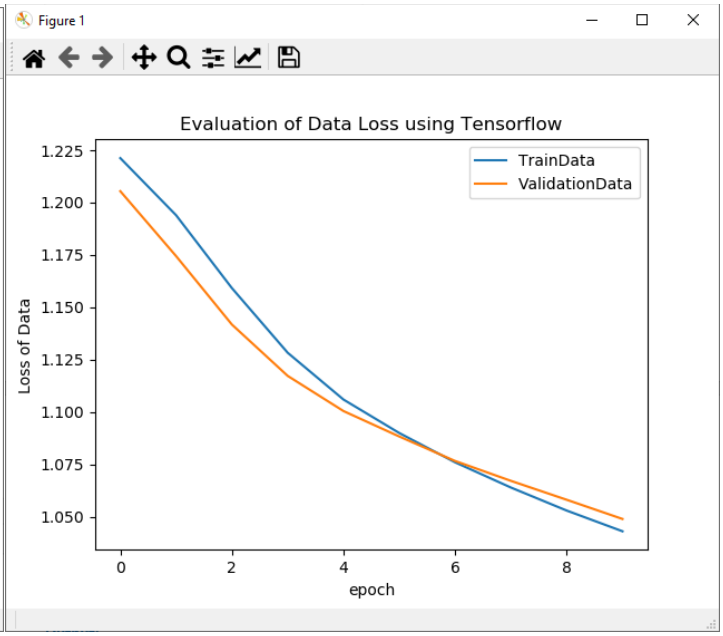
Output:

```
113500/117045 [============================>.] - ETA: 2s - loss: 1.2332 - accuracy: 0.5125
114000/117045 [============================>.] - ETA: 1s - loss: 1.2332 - accuracy: 0.5125
114500/117045 [============================>.] - ETA: 1s - loss: 1.2333 - accuracy: 0.5125
115000/117045 [============================>.] - ETA: 1s - loss: 1.2333 - accuracy: 0.5125
115500/117045 [============================>.] - ETA: 0s - loss: 1.2333 - accuracy: 0.5124
116000/117045 [============================>.] - ETA: 0s - loss: 1.2335 - accuracy: 0.5123
116500/117045 [============================>.] - ETA: 0s - loss: 1.2335 - accuracy: 0.5124
117000/117045 [============================>.] - ETA: 0s - loss: 1.2335 - accuracy: 0.5124
117045/117045 [==============================] - 80s 687us/step - loss: 1.2335 - accuracy: 0.5124 - val_loss: 1.2270 - val_accuracy: 0.5165


38176/39015 [============================>.] - ETA: 0s
38400/39015 [============================>.] - ETA: 0s
38624/39015 [============================>.] - ETA: 0s
38880/39015 [============================>.] - ETA: 0s
39015/39015 [==============================] - 9s 232us/step
Evaluation result on Test Data : Loss = 1.2269671072871902, accuracy = 0.51651930809021
The data components present in history are dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])

   32/39015 [..............................] - ETA: 8s
  288/39015 [..............................] - ETA: 8s

37984/39015 [============================>.] - ETA: 0s
38240/39015 [============================>.] - ETA: 0s
38496/39015 [============================>.] - ETA: 0s
38752/39015 [============================>.] - ETA: 0s
39008/39015 [============================>.] - ETA: 0s
39015/39015 [==============================] - 9s 221us/step
Evaluation result on Test Data using Tensorflow : Loss = 1.0488779652764915, accuracy = 0.5887479186058044
The data components present in history using Tensorflow are dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])

Process finished with exit code 0
```









Question 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

```
# Import libraries
import re
import pandas as pd
from keras import Sequential
from keras.callbacks import TensorBoard
from keras.constraints import maxnorm
from keras.layers import Embedding, Conv1D, Dropout, MaxPooling1D, Flatten, Dense, LSTM, SpatialDropout1D
from keras.optimizers import SGD
from keras.utils import to_categorical
from keras_preprocessing.sequence import pad_sequences
from keras_preprocessing.text import Tokenizer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt

# Load movie review data set
train_movie_df = pd.read_csv('train.tsv', delimiter='\t', encoding='utf-8')
test_movie_df = pd.read_csv('test.tsv', delimiter='\t', encoding='utf-8')

# cleaning the data set to identify features that are important
train_movie_df = train_movie_df.drop(columns=['PhraseId', 'SentenceId'])
test_movie_df = test_movie_df.drop(columns=['PhraseId', 'SentenceId'])

train_movie_df['Phrase'] = train_movie_df['Phrase'].apply(lambda x: re.sub('[^a-zA-z0-9\s]', '', x.lower()))
test_movie_df['Phrase'] = test_movie_df['Phrase'].apply(lambda x: re.sub('[^a-zA-z0-9\s]', '', x.lower()))

max_features = 2000
tokenizer = Tokenizer(num_words=max_features, split=' ')
tokenizer.fit_on_texts(train_movie_df['Phrase'].values)
X_train = tokenizer.texts_to_sequences(train_movie_df['Phrase'].values)
X_train = pad_sequences(X_train)

tokenizer.fit_on_texts(test_movie_df['Phrase'].values)
X_test = tokenizer.texts_to_sequences(test_movie_df['Phrase'].values)
X_test = pad_sequences(X_train)

# Creating the model
embed_dim = 128
lstm_out = 196
```

```
# Design the CNN using LSTM classification
# Model defined
model = Sequential()
# Input layer of the model for processing
model.add(Embedding(max_features, embed_dim, input_length=X_train.shape[1]))
# Hidden layer
model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
model.add(Dropout(0.5))
# Output layer
model.add(Dense(5, activation='softmax'))
# Compile the model identified
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Identify the data into training and test sets
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(train_movie_df['Sentiment'])
Y_train = to_categorical(integer_encoded)
x_train, x_test, y_train, y_test = train_test_split(X_train, Y_train, test_size=0.25, random_state=30)

# Fitting the model identified using the training data set
history = model.fit(x_train, y_train, epochs=2, batch_size=500, validation_data=(x_test, y_test))

# Evaluation of the results of the model obtained using the test data set
[test_loss, test_acc] = model.evaluate(x_test, y_test)
print("Evaluation result on Test Data : Loss = {}, accuracy = {}".format(test_loss, test_acc))

# Listing all the components of data present in history
print('The data components present in history are', history.history.keys())

# Graphical evaluation of accuracy associated with training and validation data
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Evaluation of Data Accuracy')
plt.xlabel('epoch')
plt.ylabel('Accuracy of Data')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()
```

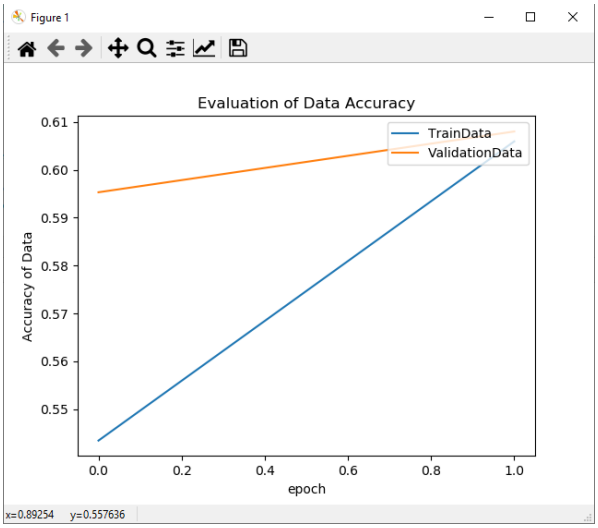
```
# Graphical evaluation of accuracy associated with training and validation data
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Evaluation of Data Accuracy')
plt.xlabel('epoch')
plt.ylabel('Accuracy of Data')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()

# Graphical evaluation of loss associated with training and validation data
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('Loss of Data')
plt.title('Evaluation of Data Loss')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()

# Visualization of the model using tensor board
tbCallBack = TensorBoard(log_dir='./lab2_1', histogram_freq=0, write_graph=True, write_images=True)

# Fitting the model defined using the training data along with validation using test data
history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=2, verbose=1, initial_epoch=0)

# Evaluation of the loss and accuracy associated to the test data set
[test_loss, test_acc] = model.evaluate(x_test, y_test)
print("Evaluation result on Test Data using Tensorflow : Loss = {}, accuracy = {}".format(test_loss, test_acc))

# Listing all the components of data present in history
print('The data components present in history using Tensorflow are', history.history.keys())

# Graphical evaluation of accuracy associated with training and validation data
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Evaluation of Data Accuracy using Tensorflow')
plt.xlabel('epoch')
plt.ylabel('Accuracy of Data')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()
```

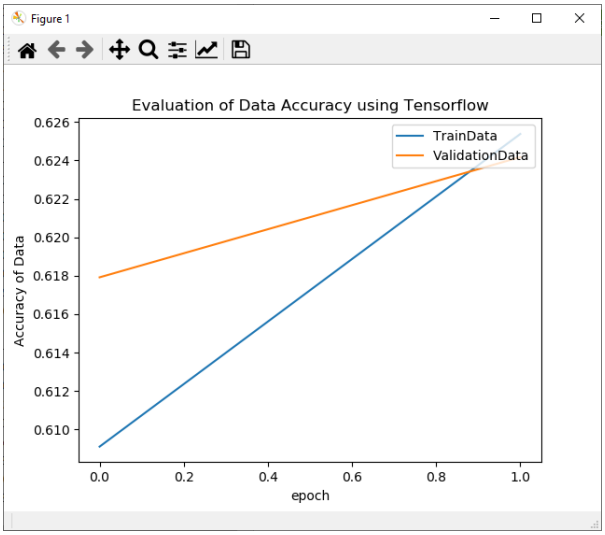
```
# Graphical evaluation of loss associated with training and validation data
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('Loss of Data')
plt.title('Evaluation of Data Loss using Tensorflow')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()
```

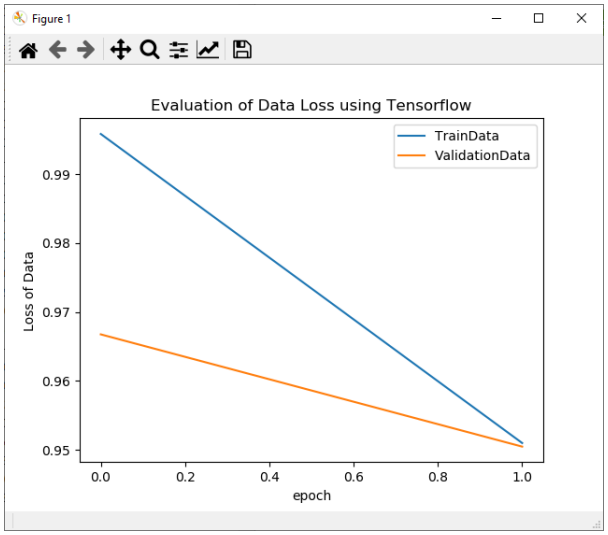
Output:

```
38720/39015 [============================>.] - ETA: 0s
38816/39015 [============================>.] - ETA: 0s
38912/39015 [============================>.] - ETA: 0s
39008/39015 [============================>.] - ETA: 0s
39015/39015 [==============================] - 24s 609us/step
Evaluation result on Test Data : Loss = 0.9855180190256677, accuracy = 0.6080482006072998
The data components present in history are dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
```

```
38720/39015 [============================>.] - ETA: 0s
38816/39015 [============================>.] - ETA: 0s
38912/39015 [============================>.] - ETA: 0s
38976/39015 [============================>.] - ETA: 0s
39015/39015 [==============================] - 24s 612us/step
Evaluation result on Test Data using Tensorflow : Loss = 0.9504551504042673, accuracy = 0.624170184135437
The data components present in history using Tensorflow are dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
```





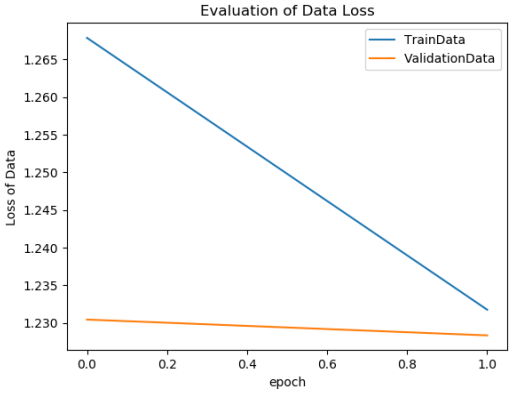


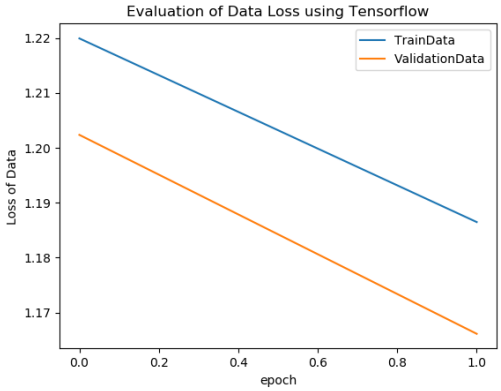
Question 6: Compare the results of CNN and LSTM models, for the text classification and describe, which model is best for the text classification based on your results. Tune the hyperparameters to attain good accuracy and show the results.

To compare results, I have modified the model with same epochs, loss and activation functions. LSTM performs better than CNN for provided dataset.

CNN:

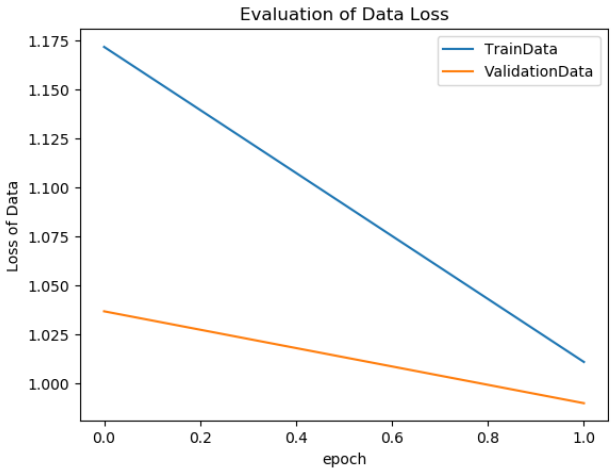
```
46818/46818 [==============================] - 36s 778us/step
Evaluation result on Test Data : Loss = 1.2283497115488038, accuracy = 0.5144388675689697
The data components present in history are dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
```

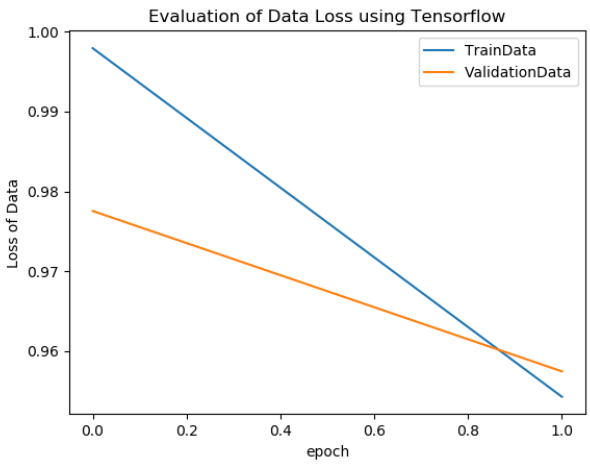




LSTM:

```
46688/46818 [==============================>.] - ETA: 0s
46784/46818 [==============================>.] - ETA: 0s
46818/46818 [==============================] - 37s 786us/step
Evaluation result on Test Data : Loss = 0.9898965473849086, accuracy = 0.6136314868927002
The data components present in history are dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
```





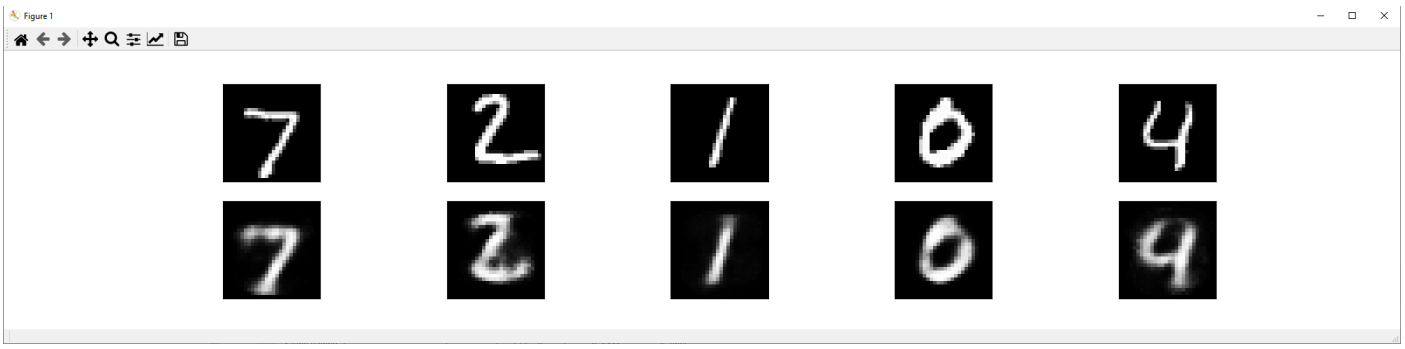
Question 7: Apply Autoencoders on MNIST dataset and show the encoding and decoding on a particular image. Make sure you document each and every line of the code.

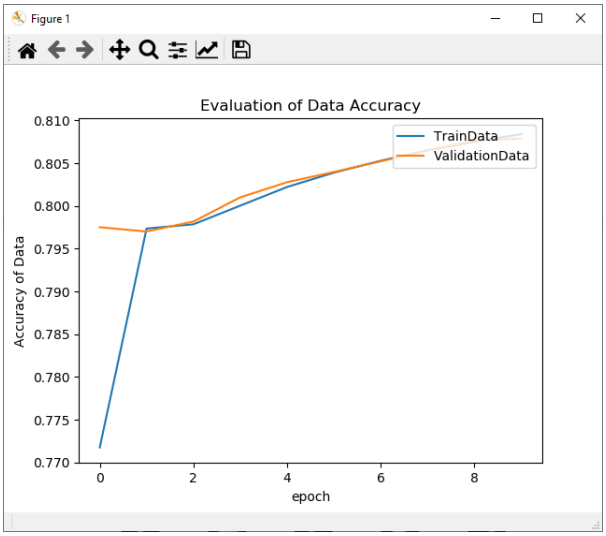
```
# Import libraries
from keras.callbacks import TensorBoard
from keras.layers import Input, Dense
from keras.models import Model

# Size of our encoded representations
encoding_dim = 32  # 32 floats -> compression of factor 24.5, assuming the input is 784 floats

# Input placeholder
input_img = Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = Dense(784, activation='sigmoid')(encoded)

# Mapping input to its reconstruction
autoencoder = Model(input_img, decoded)

# Model mapping an input to its encoded representation
encoder = Model(input_img, encoded)

# Decoder model representation
# Creating a placeholder for an encoded (32-dimensional) input
encoded_input = Input(shape=(encoding_dim,))
# Retrieving the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
# Creating the decoder model
decoder = Model(encoded_input, decoder_layer(encoded_input))

# Compiling the model defined
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy', metrics=['acc'])

# Loading input data set
from keras.datasets import mnist
import numpy as np
(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:])))

# Fitting the model defined on training data set
history = autoencoder.fit(x_train, x_train, epochs=10, batch_size=150, shuffle=True, validation_data=(x_test, x_test))

# Encode and decode some digits
# Note that we take them from the *test* set
encoded_imgs = encoder.predict(x_test)
decoded_imgs = autoencoder.predict(x_test)

# Visualization of reconstructed inputs and decoded representations
# Library imported to be used
from matplotlib import pyplot as plt
# Identify the number of digits to be displayed
n = 5  # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original input
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()

# Graphical evaluation of accuracy associated with training and validation data
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Evaluation of Data Accuracy')
plt.xlabel('epoch')
plt.ylabel('Accuracy of Data')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()
```

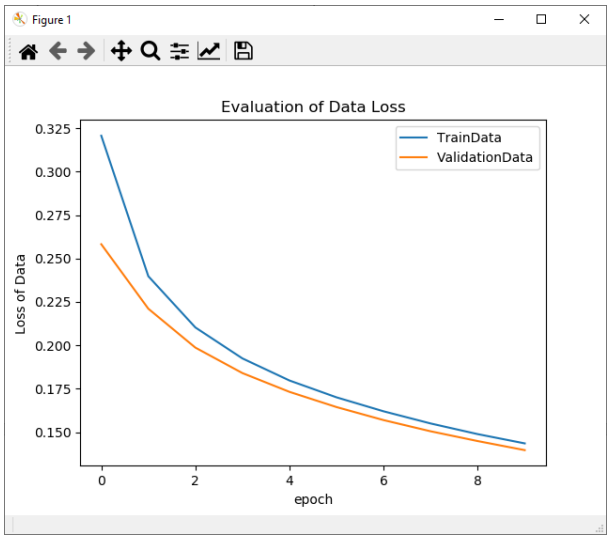
```
# Graphical evaluation of loss associated with training and validation data
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('Loss of Data')
plt.title('Evaluation of Data Loss')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()

# Visualization of the model using tensor board
tbCallBack = TensorBoard(log_dir='./7_autoencoder', histogram_freq=0, write_graph=True, write_images=True)

# Fitting the model defined using the training data along with validation using test data
history = autoencoder.fit(x_train, x_train, epochs=10, batch_size=256, shuffle=True, validation_data=(x_test, x_test))

# Listing all the components of data present in history
print('The data components present in history using Tensorflow are', history.history.keys())

# Graphical evaluation of accuracy associated with training and validation data
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Evaluation of Data Accuracy using Tensorflow')
plt.xlabel('epoch')
plt.ylabel('Accuracy of Data')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()

# Graphical evaluation of loss associated with training and validation data
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('Loss of Data')
plt.title('Evaluation of Data Loss using Tensorflow')
plt.legend(['TrainData', 'ValidationData'], loc='upper right')
plt.show()
```

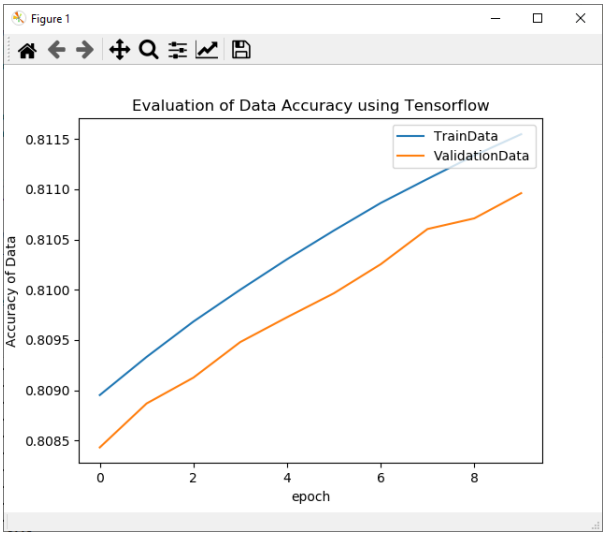
Output:

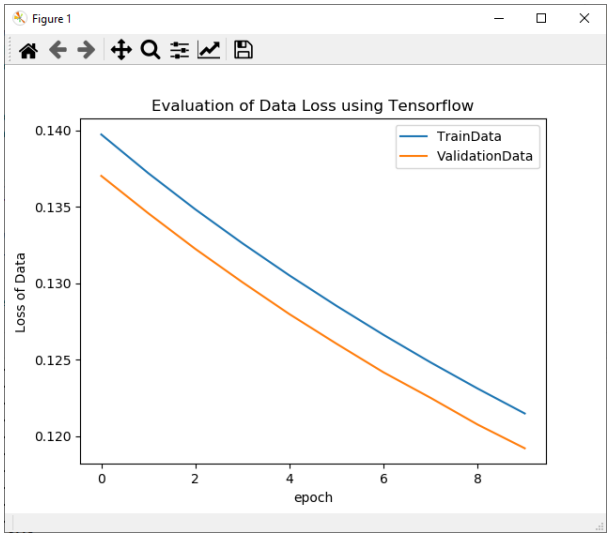
```
58050/60000 [============================>.] - ETA: 0s - loss: 0.1436 - acc: 0.8083
58950/60000 [============================>.] - ETA: 0s - loss: 0.1435 - acc: 0.8084
60000/60000 [==============================] - 3s 58us/step - loss: 0.1435 - acc: 0.8084 - val_loss: 0.1396 - val_acc: 0.8079
Train on 60000 samples, validate on 10000 samples

57088/60000 [============================>..] - ETA: 0s - loss: 0.1215 - acc: 0.8116
58368/60000 [============================>.] - ETA: 0s - loss: 0.1215 - acc: 0.8115
59648/60000 [============================>.] - ETA: 0s - loss: 0.1215 - acc: 0.8115
60000/60000 [==============================] - 3s 53us/step - loss: 0.1215 - acc: 0.8115 - val_loss: 0.1192 - val_acc: 0.8110
The data components present in history using Tensorflow are dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])

Process finished with exit code 0
```











VIII. Video Link. https://www.loom.com/share/b2a6db69fefe471391c74bca35c640a8

IX. References:

- www.google.com
- www.https://towardsdatascience.com/
- https://www.analyticsvidhya.com/
- https://machinelearningmastery.com/