SOURCE CODE

1st Solution:

```
#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

#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
#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)
#EValuating the model
mdl.evaluate(x_test,y_test)
x_test.iloc[1]
#Predicting using the model
y=mdl.predict_classes(x_test.iloc[1:])

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

#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()

"""### (a). Changing the hyperparameter 'Learning rate'"""

#Optimiser ADAM with learning rate 0.0001
optm = keras.optimizers.Adam(learning_rate=0.0001)
#calling the TensorBpoard from keras
tensorboard=TensorBoard(log_dir="p1a/{}".format(time()),histogram_freq=0,
write_graph=True, write_images=True)
#Implementing the SkLearn regressor interface
regressor2=KerasRegressor(build_fn=modelfunction)
#Fitting the model with batch size 150 and total of 100 epochs
mdl_fit=regressor2.fit(x_train,y_train,epochs= 100, batch_size=
150,callbacks=[tensorboard])
evalve2= regressor2.score(x_test,y_test)
print(evalve2)
#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/p1a

#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()

"""### (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

#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()

"""### (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

#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()

"""### (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

#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()
```

2nd Solution:

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

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

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

"""###(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:])

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

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

"""### (b). Changing the hyperparameter 'Batch size' """

#Creating and Compiling a Sequential Model
mdl2 = Sequential()
mdl2.add(Dense(output_dim=10, input_shape=(12,), init='normal', activation='softmax'))
mdl2.compile(optimizer=optm, loss='categorical_crossentropy', metrics=['accuracy'])
#calling the TensorBoard from keras
tensorboard = TensorBoard(log_dir="p2b/{}".format(time()),histogram_freq=0,
write_graph=True, write_images=True)
##Fitting the model with batch size 50 and total of 20 epochs
mdl2_fit=mdl2.fit(x_train, y_train1, nb_epoch=15, batch_size=5,callbacks=[tensorboard])
#Changing batch size from  50 to 5
#predicting the accuracy of the model
score2 = mdl2.evaluate(x_test, y_test1, verbose=1)
print('Loss: %.2f, Accuracy: %.2f' % (score2[0], score2[1]))
y2=mdl2.predict_classes(x_test.iloc[1:])

#Loading the Tensor Board
%load_ext tensorboard
#Getting started with the Tensor Board
%tensorboard --logdir /content/p2b

#plotting the loss
plt.plot(mdl2_fit.history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'])
plt.show()

"""### (c). Changing the hyperparameter 'Optimizer'"""
```

```
#Optimiser ADAM with learning rate 0.01
optm3 = keras.optimizers.SGD(learning_rate=0.01)   #Changing Optimizer from Adam to SGD
#Creating and Compiling a Sequential Model
mdl3 = Sequential()
mdl3.add(Dense(output_dim=10, input_shape=(12,), init='normal', activation='softmax'))
mdl3.compile(optimizer=optm3, loss='categorical_crossentropy', metrics=['accuracy'])
#calling the TensorBoard from keras
tensorboard = TensorBoard(log_dir="p2c/{}".format(time()),histogram_freq=0,
write_graph=True, write_images=True)
##Fitting the model with batch size 50 and total of 20 epochs
mdl3_fit=mdl3.fit(x_train, y_train1, nb_epoch=15,
batch_size=50,callbacks=[tensorboard])
#predicting the accuracy of the model
score3 = mdl3.evaluate(x_test, y_test1, verbose=1)
print('Loss: %.2f, Accuracy: %.2f' % (score3[0], score3[1]))
y3=mdl3.predict_classes(x_test.iloc[1:])

#Loading the Tensor Board
%load_ext tensorboard
#Getting started with the Tensor Board
%tensorboard --logdir /content/p2c

#plotting the loss
plt.plot(mdl3_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()

"""### (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:])

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

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

3rd Solution:

```
# %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()
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'):#cat1 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)

import matplotlib.pyplot as plt #displaying the image predicted
plt.imshow(x_train[10,:,:],cmap='gray')
plt.title('Ground Truth : {}'.format(y_train[10]))
plt.show()

x_test = x_test.astype('float32')
x_train = x_train.astype('float32')
x_train = x_train / 255.0
x_test = x_test / 255.0
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
model = Sequential() #creating the sequential model
model.add(Conv2D(64, (3, 3), input_shape=(x_train.shape[1:]), padding='same',
activation='relu'))
model.add(Dropout(0.5))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(64, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.3))
model.add(Dense(num_classes, activation='softmax'))
epochs = 10
lrate = 0.001
decay = lrate/epochs
sgd = Adam(lr=lrate)
model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
h=model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=epochs,
batch_size=64) #fitting the model
x1=model.predict_classes(x_test[[2],:]) #predicting the model
print(x1[0])
print(y_test[2])

import matplotlib.pyplot as plt #displaying the predicted image
plt.imshow(x_test[2,:,:],cmap='gray')
plt.title('Ground Truth : {}'.format(y_test[2]))
plt.show()

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()
```

4th Solution:

```
# 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
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()))
# 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
X_test1.shape
# 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)
# 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'
])
# Fitting the model
history1=model1.fit(X_train, Y_train, validation_data=(X_test, Y_test),epochs=10, b
atch_size=512, verbose=1)
# 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()
```

5th Solution:

```
# 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
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()))
# 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
X_test1.shape
# 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)
# Creating a LSTM Model
embed_dim = 40
lstm_out = 20
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())
# Fitting the model
```

```
history1=model1.fit(X_train, Y_train, validation_data=(X_test, Y_test),epochs=3, batch_size=256, verbose=1)
# 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()
```

6th Solution:

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)
# 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()
# Prediction
y_predicted=model2.predict_classes(X_test1[:1])
print(y_predicted[0]," PREDICTED LABEL")
# 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")
```

Tuning the parameters to achieve good accuracy for LSTM Model

```
# Creating a LSTM Model
embed_dim = 80
lstm_out = 40
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())
# Fitting the model
history2=model2.fit(X_train, Y_train, validation_data=(X_test, Y_test),epochs=5, batch_size=50, verbose=1)
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()
y_predicted=model1.predict_classes(X_test1[:1])
print(y_predicted[0]," PREDICTED LABEL")
file = pa.read_csv('/content/drive/My Drive/Lab2/Datasets/sampleSubmission.csv',sep=',')
print(file['Sentiment'].iloc[0]," ACTUAL LABEL")
```

7th Solution:

```
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

# 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)

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

# encode and decode
encoded_images = encoder.predict(x_test)
decoded_images = decoder.predict(encoded_images)
n = 50  # Number of digits that displays
plt.figure(figsize=(50, 4))
for i in range(n):
    #original data display
    ax = plt.subplot(3, n, i + 1 )
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(True)
    ax.get_yaxis().set_visible(True)

n = 50 # Number of digits that displays
plt.figure(figsize=(50, 4))
for i in range(n):
  # Noisy data
  ax = plt.subplot(3, n, i + 1 + n)
  plt.imshow(x_test_noisy[i].reshape(28, 28))
  plt.gray()
  ax.get_xaxis().set_visible(True)
  ax.get_yaxis().set_visible(True)

n = 50 # Number of digits that displays
plt.figure(figsize=(50, 4))
for i in range(n):
    # Reconstruction data
    ax = plt.subplot(3, n, i + 1 + n + n)
    plt.imshow(decoded_images[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(True)
    ax.get_yaxis().set_visible(True)
plt.show()

figure1 = plt.figure() # plotting the loss curve
plt.plot(auto_fit.history['loss'], 'r')
plt.plot(auto_fit.history['val_loss'], 'b')
plt.legend(['Training loss', 'Validation Loss'])
plt.xlabel('Epoch values ')
plt.ylabel('Loss')
plt.title('Loss Curve for training and validation : ')

figure2 = plt.figure() #plotting the accuracy curve
plt.plot(auto_fit.history['accuracy'], 'r')
plt.plot(auto_fit.history['val_accuracy'], 'b')
plt.legend(['Training acc', 'Validation acc'])
plt.xlabel('Epoch values ')
plt.ylabel('Accuracy')
plt.title('Accuracy Curve for training and validation : ')
```