Experiment No. 1: Linear regression using deep neural network on Boston housing dataset.

```
from keras.callbacks import ModelCheckpoint

from keras.models import Sequential

from keras.layers import Dense, Activation, Flatten

from sklearn.model selection import train_test_split

from sklearn.ensenmble import RandomForestRegressor

from sklearn.metrics import mean_absolute_error

from matplotlib import pyplot as plt

import seaborn as sb

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import warnings

warnings.filterwarnings('ignore')

warnings.flterwarnings('ignore', category=DeprecationWarning)

from xgboost import XGBRegressor


#processing of Dataset

def get_data):

#get train data

train_data_path ='train.csv'

train = pd.read csv(train _data_path)


#get test data

test_data_path ='test.csv'
```

```
test = pd.read _csv(test_data_path)

return train, test


def get_combined_data ():


#reading train data

train, test = get_data()

target = train.SalePrice

train.drop (['SalePrice'],axis = 1, inplace = True)

combined=train.append(test)

combined.reset_index(inplace=True)

combined.drop([index', ‘Id’], inplace=True, axis=1)

return combined, target


#Load train and test data into pandas DataFrames


train_data, test_data=getdata()


#Combine train and test data to process them together

combined, target = get_combined_data()


# define a function to get the columns that don't have any missing values

def get_cols_with_no_nans(df,col_type):


Arguments :
```

```
df: The dataframe to process

col_type:

num : to only get numerical columns with no nans

no_num : to only get nun-numerical columns with no nans

all : to get any columns with no nans

‘’’

if (col_type =='num'):

predictors = df.select_dtypes(exclude=[‘object'])

elif (col_type =='no_num’):

predictors = df.select_dtypes(include=['object’])

elif (col_type =='all ):

predictors = df

else

print('Error : choose a type (num, no_num, all)')

return 0

cols_with_no_nans = []

for col in predictors.columns:

if not df[col].isnull().any():

cols_with_no_nans.append(col)

return cols_with_no_nans


# Get the columns that do not have any missing values.

num_cols = get_cols_with_no_nans(combined, 'num’)

cat_cols = get_cols_with_no_nans(combined, 'no_num’)

# Let's see how many columns we got

print (Number of numerical columns with no nan values:, len(num_cols))
```

print ('Number of nun-numerical columns with no nan values:, 'len(cat_cols))

[out]:

Number of numerical columns with no nan values : 25

Number of nun-numerical columns with no nan values : 20

#0ne Hot Encode The Categorical Features

def oneHotEncode(df,colNames):

for col in colNames:

if(df[col].dtype == np.dtype('object')):

dummies = pd.get_dummies(df[col],prefix=col)

df= pd.concat([df,dummies], axis=1)

#drop the encoded column

dfdrop([col],axis = 1, inplace=True)

return df

print("There were {} columns before encoding categorical

features'.format(combined.shape[1]))

combined = oneHotEncode(combined, cat_cols)

print('There are {} columns after encoding categorical features'.format(combined.shape[1]))

[out]:

There were 45 columns before encoding categorical features

There are 149 columns after encoding categorical features

```
#split back combined dataFrame to training data and test data

def split combined ():

global combined

train = combined[:1460]

test = combined [1460:]


return train, test

train, test = split_combined()


#Making the Deep Neural Network

NN_model = Sequential()


# The Input Layer :

NN_model.add(Dense(128, kernel Initlalizer='normal', input_dim = train.shape [1],

activation='relu'))


# The Hidden Layers:

NN model.add(Dense(256, kernel _Initiallzer='normal',activation=’relu'))

NN model.add(Dense(256, kernel Initlalizer='normal',activation='relu'))

NN model.add(Dense(256, kernel_Initialízer='normal',activatlon='relu'))


# The Output Layer:

NN_ model.add (Dense(1, kernel_initializer='normal',activation=’linear’))
```

Compile the network:

NN_model.compile (loss='mean_absolute_error', optimlzer='adam', metrics=['mean

_absolute_error')

NN_model.summary()

[Out]:

---Layer (type) Output Shape Param #

===

-- dense_1 (Dense) (None, 128) 19200

---dense_2 (Dense) (None, 256) 33024

---dense_3 (Derrse) (None, 256) 65792

---dense_4 (Dense) (None, 256) 65792

---dense_5 (Dense) (None, 1) 257

===

Total params: 184,065 Trainable params: 184,065 Non-trainable params: 0

#Define a checkpoint call back:

checkpoint_name = ''Weights-{epoch:03d}--(val_loss:.5f).hdf5'

checkpoint = ModelCheckpoint (checkpoint_name, monitor='val _loss', verbose = 1,

save_best_only= True, mode ='auto')

callbacks_list = [checkpoint]

#Train the model:

NN_model.fit(train, target, epochs=500, batch_size=32, validation_split = 0.2,

callbacks=callbacks_list)

[out]:

Train on 1168 samples, valildate on 292 samples

Epoch 1/500

1168/1168 [==========]-0s 266us/step - loss: 19251.8903 - mean_absolute_error:

19251.8903 – val_loss: 23041.8968 – val_mean_absolute_error: 23041.8968

Epoch 00001: val_loss did not improve from 21730.93555

Epoch 2/500

1168/1168 [=========]- 0s 268us/step- loss: 18180.4985- mean _absolute_error:

18180,4985- val_loss: 22197.7991 - val _mean_absolute_error: 22197.7991

Epoch 00002: val_loss did not improve from 21730.93555

Epoch 00500: val_loss did not improve from 18738.1983 1

Load wights file of the best model :

wights_file = "Weights-478--18738.19831. hdf5' # choose the best checkpoint

NN_model.load_weights (wights_file) # load it

NN_model.compile(loss='mean absolute error'.

optimizer='adam',metrics=['mean_absolute_error’])

```
#Test the model

def make_submission (prediction, sub_name):

my_submission = pd.Data Frame({'Id':pd.read_csv('test.csv'). Id, 'SalePrice':prediction})

my_submission.to_csv('{}.csv'.format(sub_name), index=False)

print('A submission file has been made')

predictions = NN_model.predict(test)
```

OUT:

0.14605

Experiment No. 2: Deep neural network on IMDB Dataset

1. The dataset is the Large Movie Review Dataset, often referred to as the IMDB dataset.

2. The IMDB dataset contains 50,000 highly polar movie reviews (good or bad) for training and the same amount again for testing. The problem is to determine whether a given movie review has a positive or negative sentiment.

```
#Load the IMDB Dataset with Keras

import numpy as np
from tensorflow.keras.datasets import imdb
import matplotlib.pyplot as plt

# load the dataset
(X_train, y_train), (X_test, y_test) = imdb.load_data()
X= np.concatenate((X_train, X_test), axis=0)
y= np.concatenate((y_train, y_test), axis=0)

# summarize size
print("Training data: ")
print(X.shape)
print(y.shape)
```

OUT:

Training data:

(50000,)

(50000,)

#Word Embedding

imdbload_data (nb_words=5000)

#truncate or pad the dataset to a length of 500 for each observation

X_train = sequence.pad_sequences(X_train, maxlen=500)

X_test = sequence.pad_sequences(X_test, maxlen=500)

Embedding(5000, 32, input_length=500)

#Simple Multi-Layer Perceptron Model for the IMDB Dataset

MLP for the IMDB problem

from tensorflow.keras.datasets import imdb

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import Flatten

from tensorflow.keras.layers import Embedding

from tensorflow.keras.preprocessing import sequence

load the dataset but only keep the top n words, zero the rest

top_words = 5000

(X_train, y_train),(X_test, y_test) = imdb.load_data (num_words=top_words)

```
#bound reviews at 500 words, truncating longer reviews and zero-padding shorter one

max_words =500
X_train = sequence.pad_sequences (X_train, maxlen=max_words)
X_ test = sequence.pad_sequences (X_test, maxlen=max_words)

# create the model
model = Sequential()
model.add (Embedding (top_words, 32, input length=max_words))
model.add (Flatten ())
model.add (Dense [250, activation='relu'))
model.add (Dense(1, activation='sigmoid'))
model.compile (loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()

# Fit the model
nodel.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=2, batch_size=128,
verbose=2)

# Final evaluation of the model
Scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

OUT:

Epoch ½

196/196-4s - loss: 0.5579 - accuracy: 0.6664 - val loss: 0.3060 - val_accuracy: 0.8700 -

4s/epoch - 2Oms/step

Epoch 2/2

196/196 - 4s - loss: 0.2108 - accuracy: 0.9165 - val_loss: 0.3006 - val_accuracy: 0.8731 -

4s/epoch - 19ms/step

Accuracy: 87.319%

Experiment No. 3 : Plant Disease prediction using CNN (PART 1)

```
import numpy as np

import pickle

import cv2

from os import listdir

from sklearn.preprocessing import LabelBinarizer

from keras.models import Sequential

from keras.layers.normalization import Batch Normalization

from keras.layers.convolutional import Conv2D

from keras.layers.convolutional import MaxPooling2D

from keras.layers.core import Activation, Flatten, Dropout, Dense

from keras import backend as K

from keras.preprocessing.image import ImageDataGenerator

from keras.optimizers import Adam

from keras.preprocessing import image

from keras.preprocessing.image import img_to_array

from sklearn.preprocessing import MultiLabelBinarizer

from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt

import tensorflow

EPOCHS = 25

INIT_LR= 1e-3

BS = 32

default image_size = tuple((256, 256))

image_size = 0
```

```
directory_root ='./input/plantvillage/'
width=256
height=256
depth=3
def convert_image_to_array(image_dir):
try:
image = cv2.imread(image_dir)
if image is not None:
image = cv2.resize (image, default_image_size)
return img_to_array(image)
else:
return np.array ([])
except Exception as e:
print(f"Error : {e}")


return None
image_list, label_list = [],[]
try:
print("[INFO] Loading images..")
root_dir = listdir (directory_root)


for directory in root_dir :
# remove .DS_Store from list
if directory == ".DS_Store":
root_dir.remove (directory)
```

```
for plant_folder in root_dir :

plant_disease_folder_list = listdir(f" {directory_root}/(plant_folder}”)


for disease_folder in plant_disease_folder list:

# remove .DS_Store from list

if disease folder == ".DS_Store" :

plant_disease _folder_list.remove (disease_folder)


for plant_disease_folder in plant_disease_folder_list:

print(f”[INFO] Processing {plant disease_folder}..")

plant_disease_image_list = listdir(f"{directory_root}/{plant_folder}/{plant_disease_folder} /")


for single_plant_disease_image in plant_disease_image_list :

if single_plant_disease_image == ".DS_ Store":

plant_disease_image_list.remove(single_plant_disease_image)


for image in plant_disease_image_list[:200]:

image_directory = f”{directory_root}/{plant, folder}/{plant_disease_folder}/{image}"

if image_directory.endswith (".jpg") == True or image_directory.endswith(".JPG") == True:

Image_list.append (convert_image_to_array(image_directory))

label_list.append (plant_disease_folder)

print("[INFO] Image loading completed")

except Exception as e:

print(f”Error:{e}")

image_size = len(image_list)
```

```
label_binarizer = LabelBinarizer()

image_labels = label_binarizer.fit_transform(label_list)

pickle.dump(label_binarizer,open('label_transform.pkl', 'wb'))

n_classes = len(label_binarizer.classes_)


print(label_binarizer.classes)


np_image_list = np.array (image_list, dtype=np.float16) / 225.0


print("[NFO] Spliting data to train, test")

X_train, x_test, y_train, y_test = train_test_split(np_image_list, image_labels, test_size=0.2,

random state = 42)


aug = ImageDataGenerator(

rotation_range=25, width_shift_range=0.1,

height_shift_range=0.1, shear_range=0.2,

zoom_range=0.2,horizontal_flip=True,fill mode="nearest")


model = Sequential()

inputShape = (height, width, depth)

    chanDim=-1

if K.image_data_format() == "channels_first":

inputShape = (depth, height, width)

      chan Dim=1

model.add (Conv2D(32, (3, 3), padding="same",input_shape=inputShape))

model.add(Activation("relu"'))
```

```
model.add (Batch Normalization(axis=chanDim))

model.add (MaxPooling2D(pool_size=(3, 3)))

model.add (Dropout(0.25))

model.add (Conv2D (64, (3, 3), padding="'same'"))

model.add (Activation(“relu"))

model.add(BatchNormalization(axis=chanDim))

model.add(Conv2D(64,(3, 3),padding="'same"))

model.add(Activation(“relu")

model.add (Batch Normalization(axis=chanDim))

model.add(MaxPooling2D(poo_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3),. padding="'same"))

model.add(Activation("relu"))

model.add(BatchNormalization(axis=chan Dim))

model.add(Conv2D (128, (3, 3),padding="same"))

model.add(Activation(“relu"))

model.add(BatchNormalization(axis=chanDim))

model.add(MaxPooling2D(poolsize=(2, 2))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(1024))

model.add(Activation("relu"))

model.add(BatchNormalization())

model.add(Dropout(0.5))

model.add (Dense(n_classes))

model.add (Activation("'softmax"))
```

```
opt= Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)

# distribution

model.compile(loss="binary_crossentropy", optimizer=opt,metrics=["accuracy"])

# train the network

print("[INFO] training network..")


history = model.fit _generator(

aug.flow(x_train, y_train, batch _size=BS),

validation_data=(x test, y_test),

steps_per_epoch=len(x_train) // BS,

epochs=EPOCHS, verbose=1

)

acc = history.history['acc']

val_acc = history. history['val_acc']

loss = history.history['loss']

val _loss = history.history ['val_loss']

epochs = range(1, len(acc) + 1)


#Train and validation accuracy

plt.plot(epochs, acc, 'b', label='Training accuracy')

plt.plot(epochs, val_acc, 'r', label='Validation accurarcy')

plt.title('Training and Validation accurarcy')

plt.legend()

plt.figure()

#Train and validation loss

plt.plot(epochs, loss, 'b', label='Training loss')
```

```
plt.plot(epochs, val_loss, 'r, label='Validation loss')

plt.title('Training and Validation loss')

plt.legend()

plt.show()


print("[INFO] Calculating model accuracy")

scores = model.evaluate(x_test, y_test)

print(f"Test Accuracy: {scores[1]*100}")


# save the model to disk

print("[INFO] Saving model..")

pickle.dump(model,open('cnn_model.pkl', 'wb')
```

Accuracy: 96.77%

[INFO] Calculating model accuracy

591/591[=====-===] - 2s 3ms/step

Test Accuracy : 96.773830807551 92

Experiment No. 3 : Classification of MNIST Fashion dataset using CNN(PART 2)

```
import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

from keras.utils import to_categorical

from matplotlib.pyplot import figure, show

import warnings

import seaborn as sns

warnings.flterwarnings(ignore')

import matplotlib.style as style
```

```
from sklearn.model_selection import train_test_split
from keras.layers import Input, Concatenate, concatenate, Dense, Embedding, Dropout,
Conv2D, MaxPooling2 D
from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau, History
from keras.layers import Dropout, Flatten, GlobalAveragePooling2D, Activation
from sklearn,preprocessing import LabelEncoder, OneHotEncoder
from keras.preprocessing.image import imageDataGenerator
from sklearn.model_selection import train_test_split
from keras.applications.resnet50 import ResNet50
from keras.callbacks import ReduceLROnPlateau
from keras.callbacks import ModelCheckpoint
from keras.applications.vggl6 import VGG16
from keras.utils import to_categorical
from sklearn.utils import class_weight
from keras.layers.normalization import BatchNormalization
from matplotlib import pyplot as plt
from keras import backend as K
from keras.optimizers import SGD
from keras.models import Model
import seaborn as sns
import numpy as np
import argparse
import time
import glob
import cv2
import numpy
import os
```

```
import glob

ímport sys

import os

import json

import pprint

import warnings

warnings.flterwarnings('ignore')


#Load Data

!curl -L -0 https://www.dropbox.com/s/heyqll2my8uwotq/fashionmaistzip

!unzip fashionmnist:zip


#Load training and test data using dataframes from Pandas.

train = pd. read_csv("fashion-mnist_train.csv")

test = pd.read csv("fashion-mnist_test.csv'")


Img_rows, img_cols = 28, 28

input_shape = (img_rows, img_cols, 1)

X= train.iloc[:,1:]

Y= traln.iloc[:,:1]

X_ test = test.iloc[:, 1:]

Y_test = test.iloc[:,: 1]


#Normalization

X= np.asarray(X).reshape (X.shape [0], img_rows,img_cols, 1)

X_test = np.asarray(X_ test).reshape(X_test.shape [0], img_rows,ímg_cols,1)
```

```
X= (255. - X) /255.

X_test = (255. - X_test) / 255.


#Number of classes

classes = len(Y['label'],value _counts())


print("Number of features: ", X.shape[1])

print("Number of train samples: ", Xshape [0])

print("Number of test samples: ", X test.shape [0])

OUT:

Number of features: 28

Number of train samples: 60000

Number of test samples: 10000


#Training

Y_test = to_categorical(Y_test)

Y= to_categorical (Y)

X_train, X_val, Y_train, Y_val = train_test_split(X, Y, stratify=Y, test_size=0.2,

random_state=66)

Irr = ReducelLROn Plateau (monitor='val_loss', factor=0.1, patience=2, verbose=1,

epsilon=1e-3, mode= ‘min')

early_stopping = EarlyStopping(monitor='val loss',patience=5,verbose=0, mode='auto’)

checkpoint = ModelCheckpoint("checkpoint.hdf5", monitor='val_acc', verbose=1,

save_best_only=True, mode='max’)

batch size = 64

epochs = 10
```

```
from sklearn.model_selection import GridSearchCV

from keras.wrappers.scikit_learn import KerasClassifier


# define the grid search parameters

batch_size = [16, 32, 64, 80]

epochs = [10, 25, 50]

param_grid = dict (batch_size=batch_size, epochs=epochs)

model = KerasClassifier(build_fn=model_basic, verbose=0)

Grid=GridSearchCV(estimator=model, param_grid=param_grid,n_jobs1, cv=3)

grid_result = grid.fit(X_train, Y_train)


# summarize results

print("Best: %using %s" % (grid_result.best_score, grid_result.bestparams_))

means = grid_result.cv_results ['mean_test_score']

stds = grid_result.cv_results ['std_test_score']

params =grid_result.cv_results ['params']

for mean, stdev, param in zip(means, stds, params):

print("%f (%f) with: %r" % (mean, stdev, param))


#Model

def model_basic(classes=classes,optimizer='adam'):

kernel_size = (3,3)

dropout = 0.25

pool_size = (2,2)

inputs = Input(shape=(img_rows, img_cols, 1))
```

```
y= Conv2D (filters=32, kernel_size=kernel_size,activation='relu',padding='same') (inputs)
y= MaxPooling2D(pool_size=pool_size,strides=(2,2)) (y)
y= Dropout(dropout)(y)

y= Flatten()(y)

y= Dense(256,activation='relu')(y)
y= BatchNormalization()(y)
y= Dropout(dropout)(y)
outputs = Dense(classes, activation='softmax')(y)

model = Model(inputs=inputs, outputs=outputs)
model.compile (optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'|)
return model
basic_model = model_basic()

import warnings
warnings.filterwarnings('ignore')
history = basic_model.fit(X_train, Y_train, batch size=batch_size, epochs=epochs,
verbose=1, validation_data=(X_val, Y_val)
```

OUT:

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
48000/48000 [=======]-82s Zms/step- loss: 0.4390 - acc:.8470 - val_loss: 0.4019val _acc:
0.8618
```

Epoch 2/10

48000/48000 [=========]-82s 2ms/step - loss: 0.3458 - acc: 0.8776 - val_loss: 0.3046

val_acc: 0.8932

Epoch 3/10

48000/48000 [========] -82s 2ms/step - loss: 0.3110 - acc: 0.8883 - val_loss: 0.2947 -

val_acc: 0.8953

Epoch 4/10

48000/48000 [===]-81s 2ms/step - loss: 0.2935 - acc: 0.8937 - val_loss: 0.2772 -

val_acc: 0.9024

Epoch 5/10

48000/48000 [=========]-85s 2ms/step- loss: 0.2710 - acc: 0.9030 - val_loss: 0.2855-

val_acc: 0.8952

Epoch 6/10[===========| -84s 2ms/step - loss: 0.2592 - acc: 0.9067 - val _loss: 0.2574-

val_acc: 0.9063

Epoch 7/10

48000/48000 [==========]-85s 2ms/step-loss: 0.2450 - acc: 0.9107 - val_loss: 0.2773 -

val acc: 0.8998

Epoch 8/10

48000/48000 [=======] - 84s 2ms/step - loss: 0.2338 - acc: 0.9147 – val_loss: 0.2833 -

val_acc: 0.8955

Epoch 9/10

48000/48000 [========]-82s 2ms/step - loss: 0.2239 - acc: 0.9174 – val_loss: 0.2553 -

val_acc: 0.9127

Epoch 10/10

48000/48000 [========]-84s 2ms/step - loss: 0.2153 - acc: 0.9207 - val loss: 0.2564 -

val_acc: 0.9123

score = basic_model.evaluate(X_test,Y_test, verbose=0)

```
print("Test loss:", score[0])

print("Test accuracy:', score[1])
```

OUT:

Test loss: 0.24634755521683693

Test accuracy: 0.9137

Experiment No. 4: Google Stock Price prediction using RNN

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

data = pd.read_csv(‘GOOG. csv', date_parser =True)

data_training = data[data[‘Date']<’2019-01-01'].copy()
data_test = data[data['Date']>='2019-01-01’].copy()

data_training = data_training.drop([' Date', 'Adj Close'], axis = 1)

scaler = MinMaxScaler()
data_training = scaler.fit_transform(data _training)
data_training

# create RNN with 60 timesteps, ie. look 60 previous time steps
X_ train =[]
y_train =[]

for i in range (60, data_training.shape [0]):
X_train.append(data_training[i-60:1])
y_train.append(data training[i, 0])

X_train, y_train = np.array(X_train), np.array(y _train)

X_train.shape
```

OUT:
(3557, 60, 5)

```
#Building LSTM

from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout

regressior = Sequential()

regressioradd (LSTM (units = 60, ‘activation = 'relu', return_sequences = 'True’,
input_shape = (X_train.shape[1], 5)
regressior.add (Dropout(0.2))

regressior.add (LSTM (units = 60, activation ='relu', return_sequences = True))
regressioradd (Dropout(0.2))

regresslor.add(LSTM (units = 80, activation='relu', return_sequences=True))
regressior.add(Dropout(0.2))

regressior.add(LSTM(units =120, activation = 'relu))
regressior.add(Dropout(0.2))

regressior.add(Dense (units = 1))

regressior.compile (optimizer='adam’, loss = 'mean squared_error’)

regressior.fit(X_train, y_train, epochs=50, batch_size=32)

OUT:
Epoch 1/50
3557/3557 [=====]-16s 5ms/sample - loss: 0.0137
Epoch 2/50
3557/3557 [====]-12s 3ms/sample - loss: 0.0022
Epoch 3/50
```

3557/3557 [======]12s 3ms/sample - loss: 0.0018
Epoch 4/50
3557/3557 [========]- 12s 3ms/sample - loss: 0.0016
Epoch 5/50
3557/3557 [=====]- 12s 3ms/sample - loss: 0.0016

Epoch 45/50
3557/3557 [=====]-13s 4ms/sample -loss: 6.5112e-04
Epoch 46/50
3557/3557 [=====] -13s 4ms/sample - loss: 6.0908e-04
Epoch 47/50
3557/3557 [===]- 15s 4ms/sample - loss: 6.663 2 e-04
Epoch 48/50
3557/3557 [=======]-15s 4ms/sample - loss: 6.9701e-04
Epoch 49/50
3557/3557 [=====]-16s 4ms/sample - loss: 6.2277 e-04

Epoch 50/50
3557/3557 [=======]-16s 4ms/sample - loss: 6.457 1e-04
<tensorflow.python.keras.callbacks. History at 0x230c796F940>

#Testing

past_60_days = data training.tail(60)
df= past_60_days.append (data_tes, tignore_index = True)

df= df_drop(['Date',’Adj Close'], axis = 1)

inputs=scaler.transform (df)
X_test = []
y_test = []
for i in range (60,inputs.shape[0]):
X_test.append (inputs[i-60:i])
y test.append (inputs[i, 0])
X_test, y_test = np.array (X_test), np.array(y_test)
y_pred = regressior.predict(X_test)

scale = 1/8.18605127e-04

```
y_pred = y_pred*scale
y_test = y_test*scale

# Visualising the results


plt.figure(figsize=(14,5))

pltplot(y_test, color = 'red', label = 'Real Google Stock Price')

plt.plot(y pred, color = ‘blue', label = 'Predicted Google Stock Price')

plt.title('Google Stock Price Prediction')

plt.xlabel(''Time')

plt.ylabel ('Google Stock Price')

plt.legend ()

plt.show0
```