Exp No: 2 Date:

Aim: To implement a Multilayer Perceptron (MLP) using Keras with TensorFlow for

Description:-

Multilayer perception is also known as MLP. It is fully connected dense layers, which transform any input dimension to the desired dimension. A multi-layer perceptron is a neural-network that has multiple layers. To create a neural network we combine neurons together so that the outputs of some neurons are inputs of other neurons.

Program:-

```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
```

Program :-

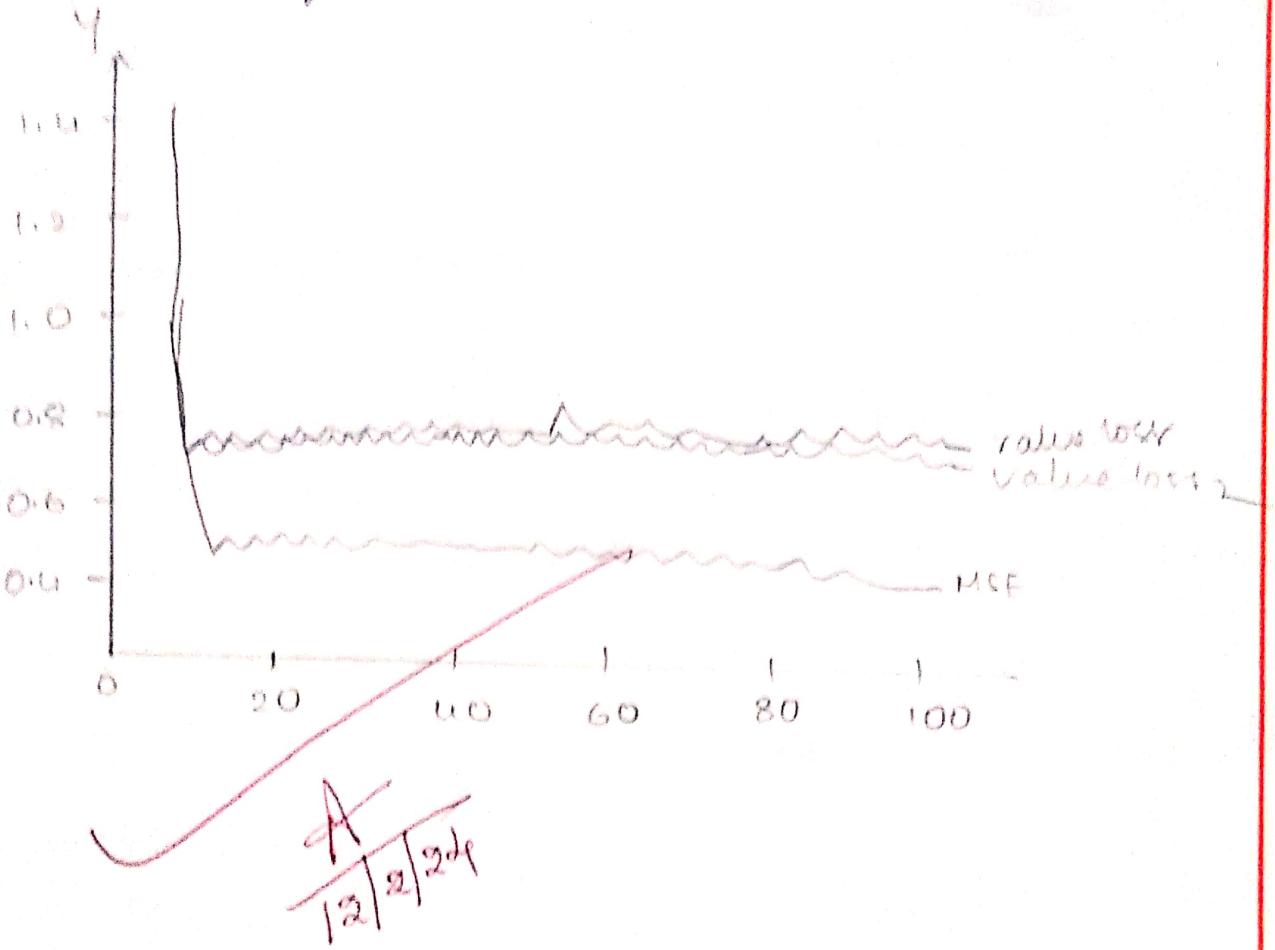
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
df = pd.read_csv("
print(df.shape)
df = df.drop('date', axis=1)
df = pd.DataFrame(df.drop(column = df.columns[4:]))
print(df.shape)
print(df.isnull().sum())
x = df.drop('price', axis=1)
y = df['price']
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y,
                    test_size=0.2, random_state=42)
x_train = pd.DataFrame(x_train)
y_train = pd.DataFrame(y_train)
s = StandardScaler()
x_train = s.fit_transform(x_train)
y_train = s.fit_transform(y_train)
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import
                    BatchNormalization
```

```
from sklearn.metrics import r2_score

model1 = Sequential()
model1.add(Dense(32, activation = 'relu', input_dim = 12))
model1.add(BatchNormalization())
model1.add(Dense(16, activation = 'relu'))
model1.add(BatchNormalization())
model1.add(Dense(1, activation = 'linear'))
model1.summary()
model2 = Sequential()
model2.add(Dense(32, activation = 'relu', input_dim=12))
model2.add(Dense(16, activation = 'relu'))
model2.add(Dense(1, activation='linear'))
model2.summary()
model1.compile(optimizer = 'adam', loss = 'mean_squared_error',
               metrics=['mean_squared_error'])
model2.compile(optimizer = 'adam', loss = 'mean_squared_error',
metrics = ['mean_squared_error'])
history1 = model1.fit(x_train, y_train, epochs=100,
                      validation_split=0.5)

plt.plot(history1.history["mean_squared_error"],
         color = "blue")
plt.plot(history1.history["val_loss"], color = "red")
plt.plot(history2.history["val_loss"], color="yellow")
plt.legend(["mean square error", "value loss",
    "value loss2"], loc="upper right")
```

val_mean_squared_error : 0.5089

Epoch 100/100

54/54 [==========] - 0s 5ms/step - loss: 0.2924 - mean_squared_error: 0.2924 - val_loss: 0.5593 - val_mean_squared_error: 0.5593



A
12/2/24

2.2

Program:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import linear Regression,
                                 Ridge, Lasso
from sklearn.metrics import mean_squared_error

file_path= "c:\\Users\\nikhitha\\Downloads\\housing-[ex-
                                              2.2].csv"

df = pd.read_csv(file_path, delim_whitespace=True,
                                 header= None)

df.columns=['CRIM','ZN','INDUS','CHAS','NOX','RM','AGE',
'DIS','RAD','TAX','PTRATIO','B','LSTAT','HOUSE Price']

x= df.drop('House Price', axis=1)

y= df['House Price']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size
                                              =0.25)

lreg = LinearRegression()
lreg.fit(x_train, y_train)
lreg_y_pred = lreg.predict(x_test)
mean_squared_error_lr = mean_squared_error(y_test,
                                 lreg_y_pred)

ridgeR= Ridge(alpha =1)

ridgeR.fit(x_train, y_train)

y_pred_ridge = ridgeR.predict(x_test)
```

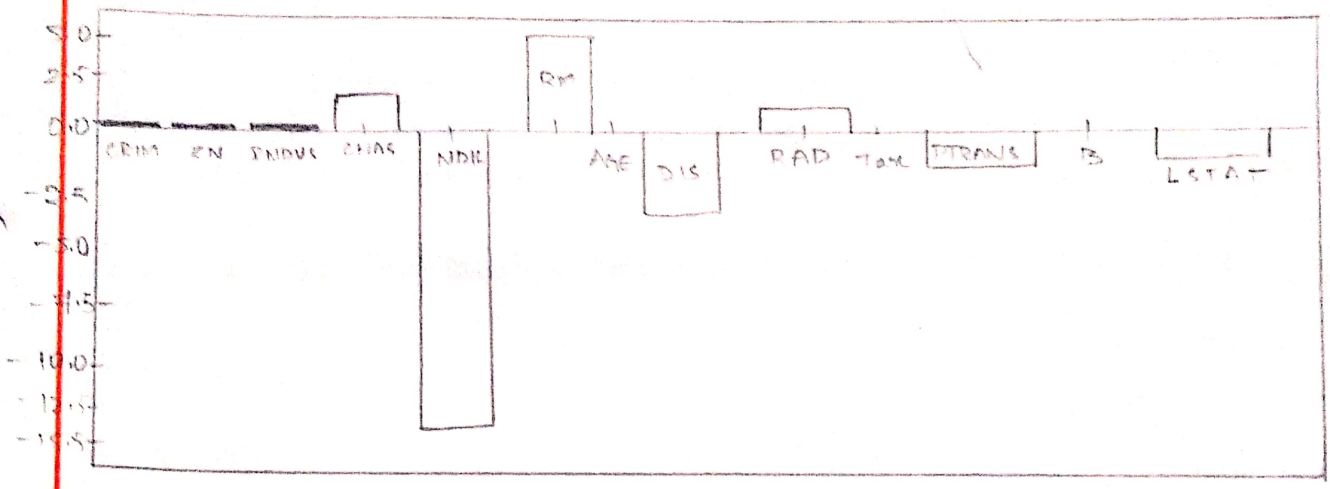
```
mean_squared_error_ridge = mean_squared_error(
                                y_test, y_pred_ridge)

lasso = Lasso(alpha=1)
lasso.fit(x_train, y_train)
y_pred_lasso = lasso.predict(x_test)
mean_squared_error_lasso = mean_squared_error(
                                y_test, y_pred_lasso)
print("Linear Regression Mean squared Error:", mean-
                                squared_error_lr)
print("Ridge Regression Mean Squared Error:", mean-
                                squared_error_ridge)
print("Lasso Regression Mean Squared Error:", mean-
                                squared_error_lasso)

def plot_coefficients(model, x_columns, color):
    fig, ax = plt.subplots(figsize=(20,10))
    ax.bar(x_columns, model.coef_, color=color)
    ax.spines["bottom"].set_position('zero')
    plt.style.use('ggplot')
    plt.xticks(rotation=45)
    plt.show()
plot_coefficients(lreg, x.columns, color='tab:blue')
plot_coefficients(ridgeR, x.columns, color='tab:
                                orange')
plot_coefficients(lasso, x.columns, color='tab:
                                green')
```

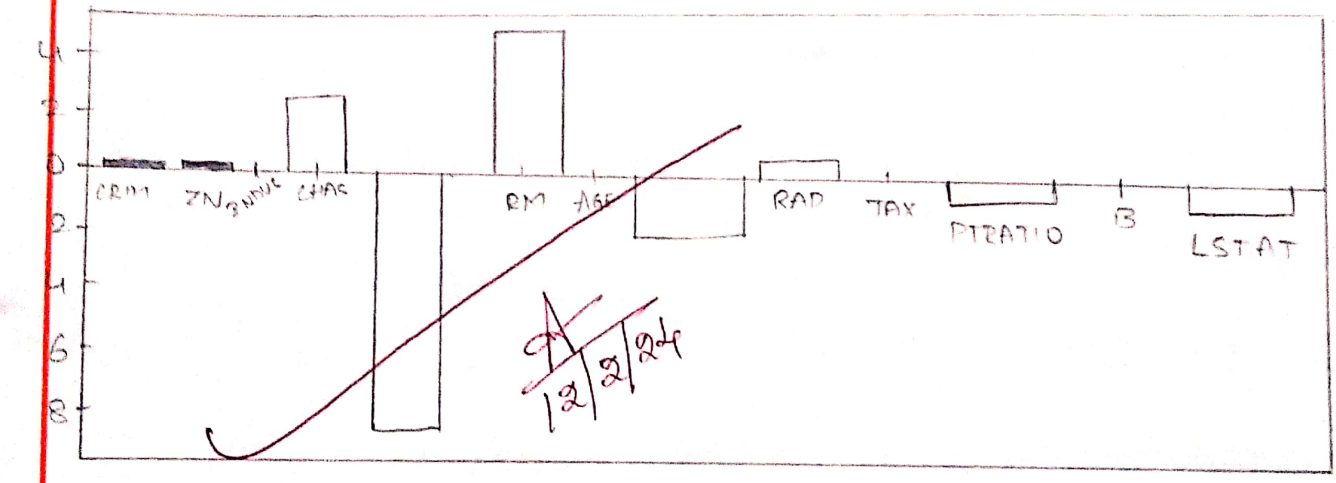
Output:-

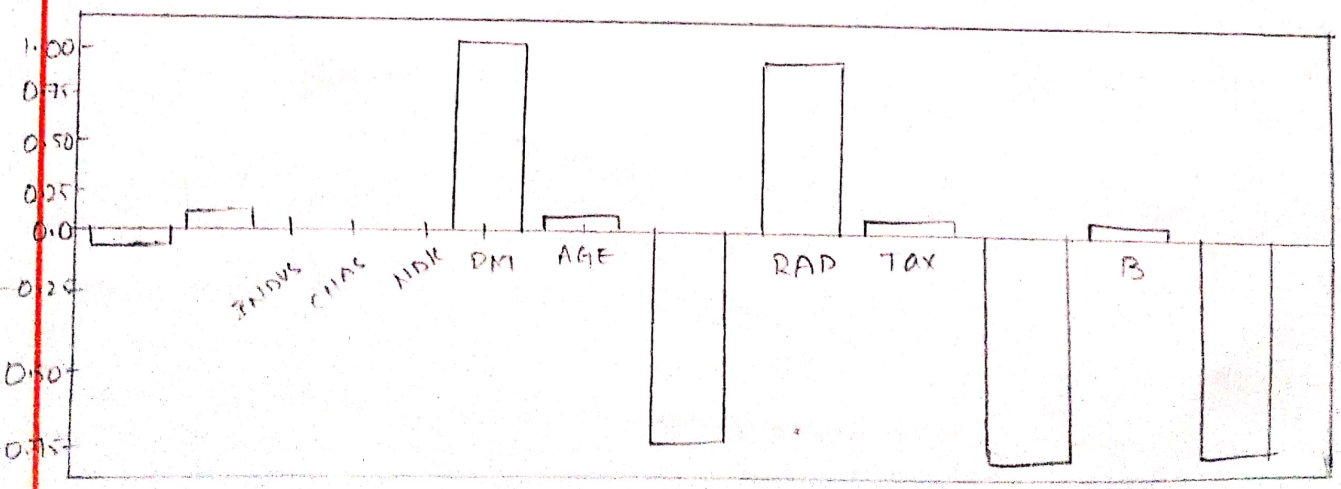
Linear Regression Mean Squared Error: 27.496145387634186

Ridge Regression Mean Squared Error: 28.030517210055088

Lasso Regression Mean Squared Error: 32.99109335605094







Exp.NO:- 4 Date:-

Aim:- To implement a convolution Neural Network (CNN) for dog/cat classification problem using keras.

Description:-

CNN is a type of artificial neural network designed for processing structured grid data, such as images. CNNs have proven highly effective in computer vision tasks, such as image recognition and object detection.

Key components of a CNN include convulutional layers, pooling layers and fully connected layers. Convulutional layers apply filters to the input data to detect features like edges and textures. Pooling layers reduce the spatial dimensions of the data, and fully connected layers make predictions based on the learned features.

Program :-

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,
        MaxPooling2D, flatten, Dense, Batchnormalisation, Dropout
from tensorflow.keras.preprocessing.image ImageData
                                          Generator
import matplotlib.pyplot as plt.

TRAIN_DIR = "dataset/train"
TEST_DIR = "dataset/test"
IMAGE_SIZE = (128,128)
BATCH_SIZE = 2
IMAGE_CHANNELS = 3
train_datagen = ImageDataGenerator(
    rescale = 1./255,
    rotation_range = 40,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range = 0.2,
    zoom_range = 0.2,
    horizontal_flip = True
)
test_datagen = ImageDataGenerator(
    rescale = 1./255,
    rotation_range = 40,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range = 0.2,
```

```
    zoom_range=0.2,
    horizontal_flip=True
)
train_generator = train_datagen.flow_from_directory(
    TRAIN_DIR,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='binary'
)
test_generator = test_datagen.flow_from_directory(
    TEST_DIR,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='binary'
)
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=
                        (128,128,IMAGE_CHANNELS)),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2,2)),
    Conv2D(64, (3,3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2,2)),
    Conv2D(128,(3,3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2,2)),
    Flatten(),
    Dense(512, activation='relu'),
    BatchNormalization(),
```

```
    Dropout(0.5),
    Dense(1, activation='sigmoid'), ])
model.compile(loss='binary_crossentropy', optimizer
                    ='adam', metrics=['accuracy'])
model.summary()
history = model.fit(train_generator, steps_per_epoch=6,
          epochs=11, validation_data=test_generator,
                                validation_steps=2)

train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
plt.plot(train_accuracy, label='Train Accuracy')
plt.plot(val_accuracy, label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
test_loss, test_accuracy = model.evaluate(test_generator)
print("Test Accuracy:", test_accuracy)
print("Test loss:" test_loss)
```

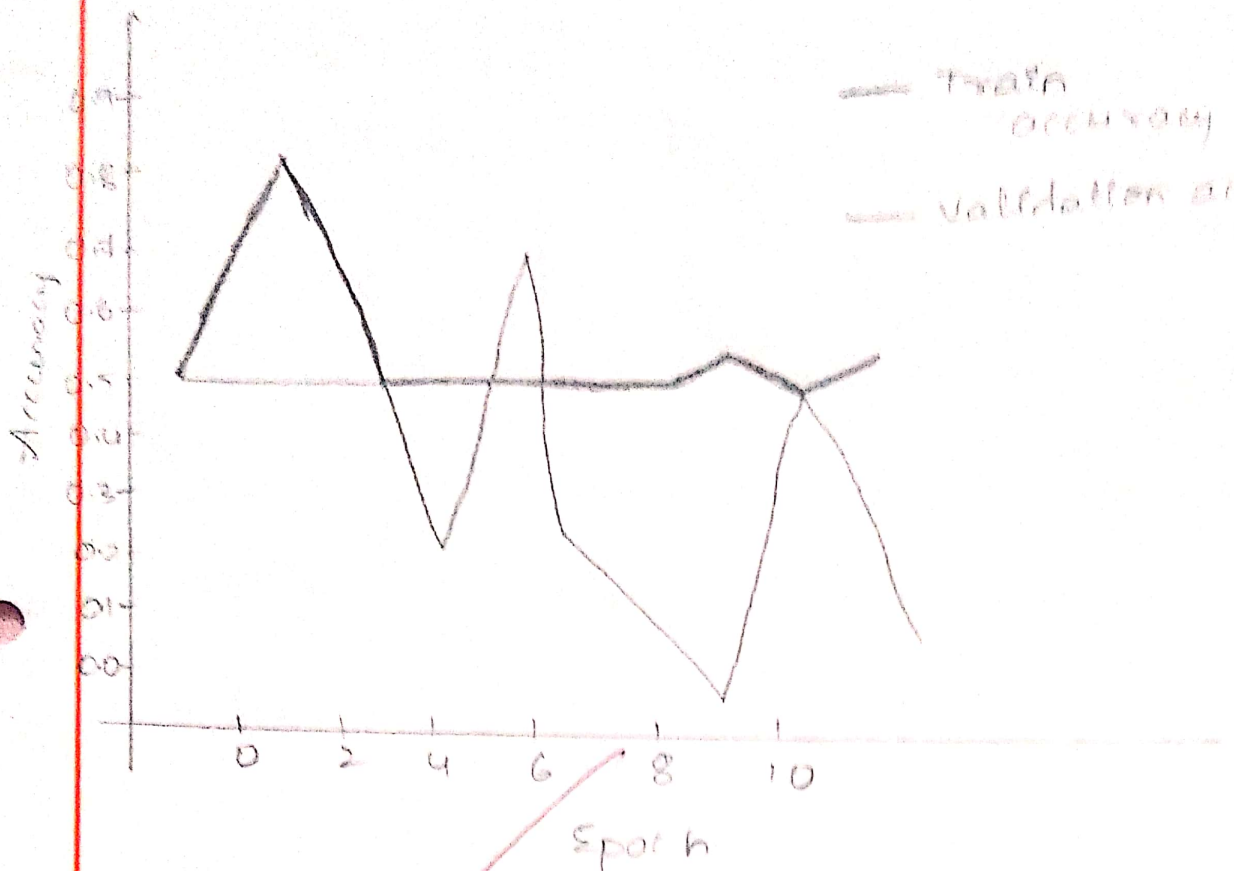
Output :-

```
found 34 images belonging to 2 classes.
found 11 images belonging to 2 classes.
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
```

conv2d (conv2d	(None, 126, 126, 32)	896
BatchNormalisation	(None, 126, 126, 32)	128
maxpooling2D	(None, 63, 63, 32)	0
conv2d-1 (conv2D)	(None, 61, 61, 64)	18496
batch_normalisation-1	(None, 61, 61, 64)	256
Max-pooling2d-1	(None, 30, 30, 64)	0
conv2d-2	(None, 28, 28, 128)	512
maxpooling2d	(None, 14, 14, 128)	0
flatten	(None, 25088)	0
dense	(None, 512)	12845568
dropout	(None, 512)	0
dense-1	(None, 1)	513

Total params : 12942273 (49.37 MB)
Trainable params : 12940801 (49.37 MB)
Non-trainable params : 1472 (5.75 KB)

Epoch 1/11
5/5 [=====] - 2s 133ms/step - loss : 3.4483 - accuracy : 0.5000 - val_loss : 0.7367 - val_accuracy : 0.5000
:
Test Accuracy : 0.36363637447357178
Test loss : 4.241678714752197


train accuracy
validation accuracy
Accuracy
0.8
0.6
0.4
0.2
0
2
4
6
8
10
Epoch

12/2/24.