Assignment 7.1: Deep Learning for Regression and Classification

Objective

In this assignment, you will be working on a real-world dataset to build deep neural networks for regression and classification problems. You will use the Keras library to build and train deep learning models. The dataset that you will be working on is the Boston Housing dataset, which contains information about houses in Boston.

Note: You are free to use any deep learning library of your choice (e.g., TensorFlow, PyTorch, and others).

`Part 1: Regression Problem`

You are required to build a deep neural network model to predict the median value of owner-occupied homes in thousands of dollars (target variable) using the given features in the dataset.

`Follow the steps below:`

- Load the Boston Housing dataset from the Keras library.
- Explore and preprocess the data (e.g., normalization, one-hot encoding, etc.).
- Split the data into training and testing sets.
- Define a deep neural network architecture for regression using Keras.
- Train the model on the training set and evaluate its performance on the testing set.
- Tune the hyperparameters of the model to achieve better performance (e.g., number of hidden layers, activation functions, learning rate, number of epochs, etc.).
- Compare the performance of the tuned model with the baseline model (i.e., the initial model without any hyperparameter tuning).

`Part 2: Classification Problem`

You are required to build a deep neural network model to classify whether a given house is expensive (1) or not (0) based on the given features in the dataset.

`Follow the steps below:`

- Load the Boston Housing dataset from the Keras library.
- Explore and preprocess the data (e.g., normalization, one-hot encoding, etc.).
- Convert the target variable into a binary variable (i.e., expensive or not expensive).
- Split the data into training and testing sets.
- Define a deep neural network architecture for classification using Keras.

- Train the model on the training set and evaluate its performance on the testing set.
- Tune the hyperparameters of the model to achieve better performance (e.g., number of hidden layers, activation functions, learning rate, number of epochs, etc.).
- Compare the performance of the tuned model with the baseline model (i.e., the initial model without any hyperparameter tuning).

In [ ]:
```
# standard libraries
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# ml libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
# loading data in pandas dataframe
df = pd.read_csv('BostonHousing.csv')
df.head()
```

Out[ ]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	

In [ ]:
```
# filling missing values with the mean of the column
df.fillna(df.mean(), inplace=True)
```

In [ ]:
```
# null values check
df.isnull().sum()
```

Out[ ]:
```
crim       0
zn         0
indus      0
chas       0
nox        0
rm         0
age        0
dis        0
rad        0
tax        0
ptratio    0
b          0
lstat      0
medv       0
dtype: int64
```

Dataset Description

- `CRIM` - per capita crime rate by town
- `ZN` - proportion of residential land zoned for lots over 25,000 sq.ft.
- `INDUS` - proportion of non-retail business acres per town.
- `CHAS` - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
- `NOX` - nitric oxides concentration (parts per 10 million)
- `RM` - average number of rooms per dwelling
- `AGE` - proportion of owner-occupied units built prior to 1940
- `DIS` - weighted distances to five Boston employment centres
- `RAD` - index of accessibility to radial highways
- `TAX` - full-value property-tax rate per $10,000
- `PTRATIO` - pupil-teacher ratio by town
- `B` - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- `LSTAT` - % lower status of the population
- `MEDV` - Median value of owner-occupied homes in $1000's

In [ ]:
```
# making a column from the mdev column x 1000 to make it the cost of the house
df['cost'] = df['medv'] * 1000
df.head()
```

Out[ ]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	

In [ ]:
```
# getting the mean of the cost column
mean_cost = df['cost'].mean()
print(mean_cost)
```

```
22532.806324110672
```

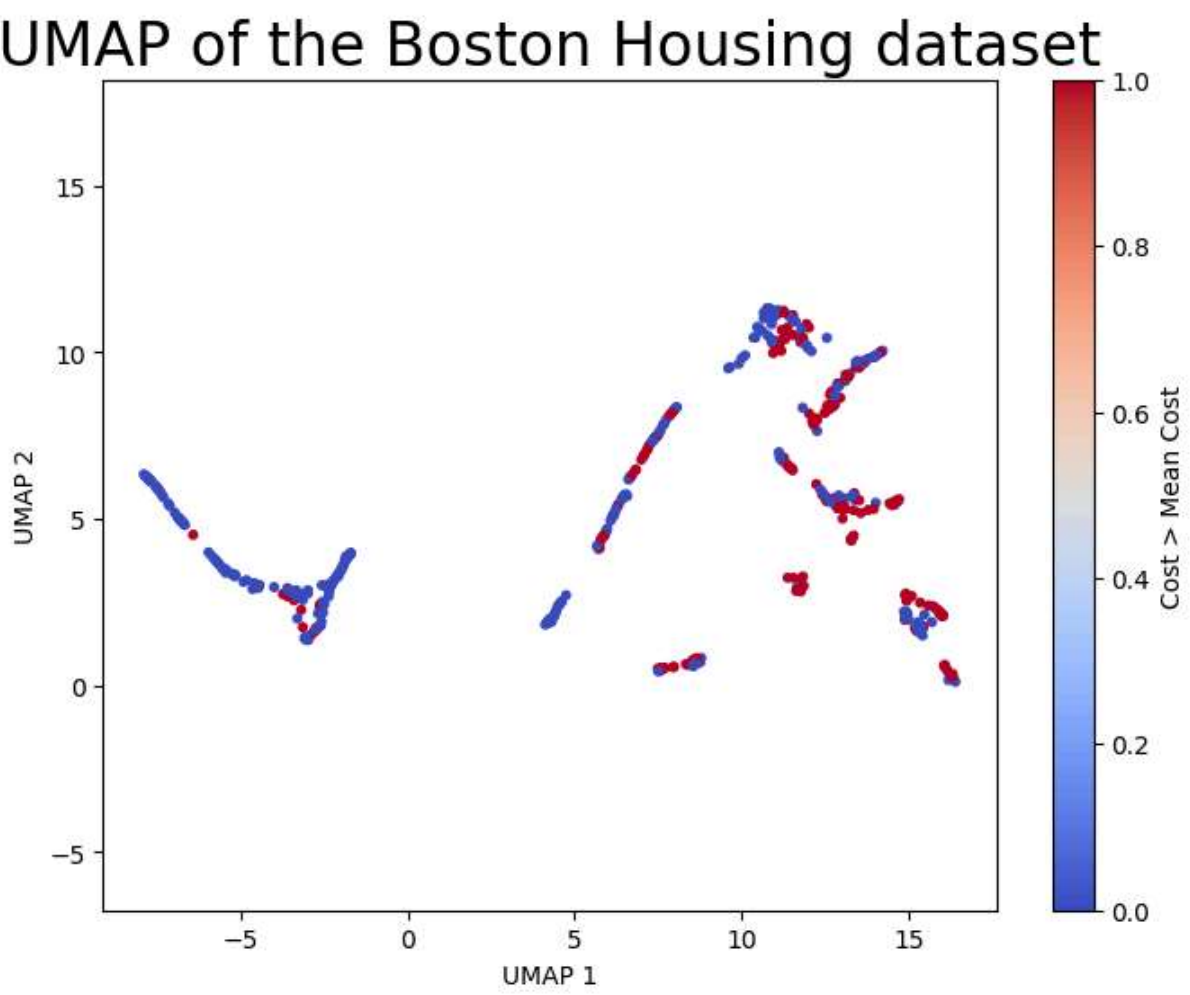
In [ ]:
```
# printing how many rows are over mean cost
print(df[df['cost'] > mean_cost].shape[0])
```

```
209
```

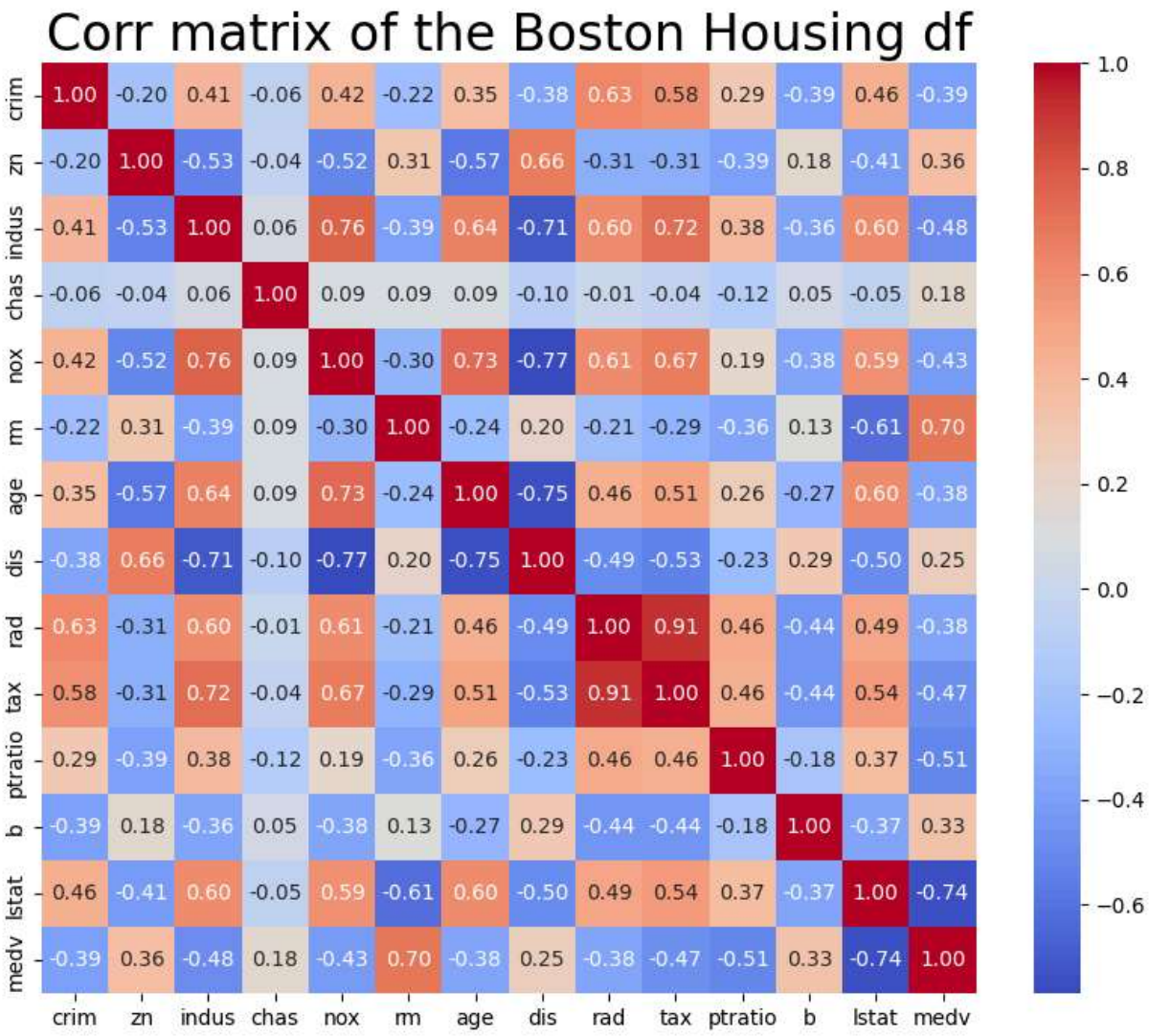
In [ ]:
```
print(df.shape)
```

```
(506, 15)
```

In [ ]:
```
# making a 2d umap plot where if the cost is over the mean it is red and if it is u

import umap.umap_ as umap

# umap projection
reducer = umap.UMAP()
embedding = reducer.fit_transform(df.drop(['cost', 'medv'], axis=1))


plt.figure(figsize=(8, 6))
plt.scatter(embedding[:, 0], embedding[:, 1], c=(df['cost'] > mean_cost), cmap='coo
plt.gca().set_aspect('equal', 'datalim')
plt.colorbar(label='Cost > Mean Cost')
plt.title('UMAP of the Boston Housing dataset', fontsize=24)
plt.xlabel('UMAP 1')
plt.ylabel('UMAP 2')
plt.show()
```



In [ ]:
```
# making a correlation matrix of the columns, dropping the cost column as it is the
corr = df.drop('cost', axis=1).corr()

# plotting the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Corr matrix of the Boston Housing df', fontsize=24)
plt.show()
```



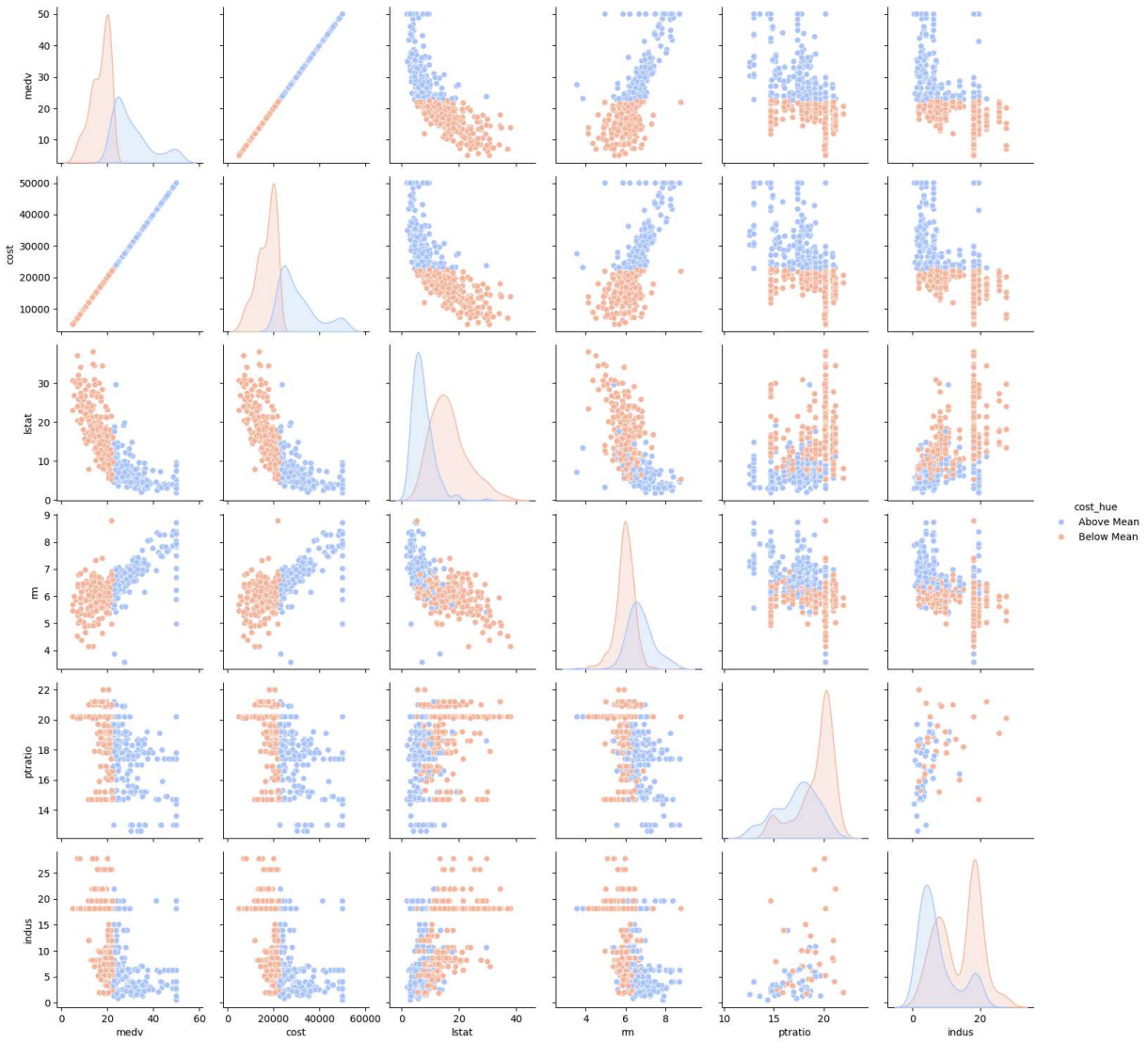
Dataset Description

- `CRIM` - per capita crime rate by town
- `ZN` - proportion of residential land zoned for lots over 25,000 sq.ft.
- `INDUS` - proportion of non-retail business acres per town.
- `CHAS` - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
- `NOX` - nitric oxides concentration (parts per 10 million)
- `RM` - average number of rooms per dwelling
- `AGE` - proportion of owner-occupied units built prior to 1940
- `DIS` - weighted distances to five Boston employment centres
- `RAD` - index of accessibility to radial highways
- `TAX` - full-value property-tax rate per $10,000
- `PTRATIO` - pupil-teacher ratio by town
- `B` - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- `LSTAT` - % lower status of the population
- `MEDV` - Median value of owner-occupied homes in $1000's

In [ ]:
```
# priniting the top 5 correlated columns with the cost column
print(corr['medv'].sort_values(ascending=False).head(6))
```

```
medv    1.000000
rm      0.695375
zn      0.360445
b       0.333461
dis     0.249929
chas    0.175260
Name: medv, dtype: float64
```

In [ ]:
```
# making a pairplot of the top 5 correlated columns with the medv column and for if

corr = df.corr()
top5_corr = corr['medv'].abs().sort_values(ascending=False).head(6).index.tolist()
top5_corr.remove('medv')
df['cost_hue'] = np.where(df['cost'] > mean_cost, 'Above Mean', 'Below Mean')
sns.pairplot(df[['medv'] + top5_corr + ['cost_hue']], hue='cost_hue', palette='cool
plt.show()
```



In [ ]:
```
# using tensorflow to train the regression model
import tensorflow as tf
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

model = Sequential()
model.add(Dense(64, activation='relu', input_dim=13))
model.add(Dense(64, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mean_squared_error')

# using mean squared error as the loss function due to it being a regression proble
```

In [ ]:
```
# reload for fresh start
df = pd.read_csv('BostonHousing.csv')
df.fillna(df.mean(), inplace=True)

# split the data
X = df.drop(['medv'], axis=1)
y = df['medv']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [ ]:
```
# scaling the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [ ]:
```
# another split
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2,
```

In [ ]:
```
# converting to numpy array
y_train = y_train.to_numpy()
y_val = y_val.to_numpy()
```

In [ ]:
```
# Compiling the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Training the model
history = model.fit(X_train, y_train, epochs=30, batch_size=30, validation_split=0.
```

```
Epoch 1/30
9/9 [==============================] - 1s 24ms/step - loss: 616.7062 - val_loss: 47
4.9155
Epoch 2/30
9/9 [==============================] - 0s 5ms/step - loss: 559.9329 - val_loss: 404.
6406
Epoch 3/30
9/9 [==============================] - 0s 5ms/step - loss: 438.9229 - val_loss: 264.
9190
Epoch 4/30
9/9 [==============================] - 0s 5ms/step - loss: 230.9891 - val_loss: 115.
8200
Epoch 5/30
9/9 [==============================] - 0s 5ms/step - loss: 112.4836 - val_loss: 113.
7071
Epoch 6/30
9/9 [==============================] - 0s 5ms/step - loss: 63.2695 - val_loss: 47.69
70
Epoch 7/30
9/9 [==============================] - 0s 6ms/step - loss: 44.0942 - val_loss: 39.15
60
Epoch 8/30
9/9 [==============================] - 0s 8ms/step - loss: 32.3978 - val_loss: 32.44
70
Epoch 9/30
9/9 [==============================] - 0s 6ms/step - loss: 26.4213 - val_loss: 29.63
00
Epoch 10/30
9/9 [==============================] - 0s 6ms/step - loss: 23.1585 - val_loss: 28.17
94
Epoch 11/30
9/9 [==============================] - 0s 5ms/step - loss: 21.2412 - val_loss: 26.33
58
Epoch 12/30
9/9 [==============================] - 0s 6ms/step - loss: 20.0266 - val_loss: 24.00
87
Epoch 13/30
9/9 [==============================] - 0s 6ms/step - loss: 18.7126 - val_loss: 22.31
39
Epoch 14/30
9/9 [==============================] - 0s 5ms/step - loss: 17.9539 - val_loss: 20.89
58
Epoch 15/30
9/9 [==============================] - 0s 5ms/step - loss: 16.9226 - val_loss: 19.34
32
Epoch 16/30
9/9 [==============================] - 0s 5ms/step - loss: 16.5066 - val_loss: 17.84
38
Epoch 17/30
9/9 [==============================] - 0s 5ms/step - loss: 15.6202 - val_loss: 16.88
80
Epoch 18/30
9/9 [==============================] - 0s 6ms/step - loss: 15.0272 - val_loss: 15.54
45
Epoch 19/30
9/9 [==============================] - 0s 11ms/step - loss: 14.4428 - val_loss: 15.1
```

```
431
Epoch 20/30
9/9 [==============================] - 0s 6ms/step - loss: 14.0111 - val_loss: 14.04
45
Epoch 21/30
9/9 [==============================] - 0s 7ms/step - loss: 13.2856 - val_loss: 13.58
82
Epoch 22/30
9/9 [==============================] - 0s 5ms/step - loss: 12.9599 - val_loss: 12.68
07
Epoch 23/30
9/9 [==============================] - 0s 5ms/step - loss: 12.3907 - val_loss: 12.50
69
Epoch 24/30
9/9 [==============================] - 0s 5ms/step - loss: 12.2353 - val_loss: 12.29
92
Epoch 25/30
9/9 [==============================] - 0s 5ms/step - loss: 11.7941 - val_loss: 11.65
87
Epoch 26/30
9/9 [==============================] - 0s 5ms/step - loss: 11.4376 - val_loss: 11.17
53
Epoch 27/30
9/9 [==============================] - 0s 6ms/step - loss: 11.1657 - val_loss: 11.11
81
Epoch 28/30
9/9 [==============================] - 0s 5ms/step - loss: 10.8887 - val_loss: 10.78
90
Epoch 29/30
9/9 [==============================] - 0s 5ms/step - loss: 10.6584 - val_loss: 10.79
66
Epoch 30/30
9/9 [==============================] - 0s 6ms/step - loss: 10.3995 - val_loss: 10.28
14
```

In [ ]:

```python
# Evaluating the model on the test data
test_loss = model.evaluate(X_test, y_test)
print(f'Test Loss: {test_loss}')

# Making predictions
y_pred = model.predict(X_test)

# Calculate the prediction accuracy
accuracy = np.mean(np.abs(y_pred - y_test.values.reshape(-1, 1)) < 1)
within_1 = np.sum(np.abs(y_pred - y_test.values.reshape(-1, 1)) < 1)
within_2 = np.sum(np.abs(y_pred - y_test.values.reshape(-1, 1)) < 2)

print(f'Accuracy: {accuracy}')
print(f'Number of predictions within 1 unit of the actual value: {within_1}')
print(f'Number of predictions within 2 units of the actual value: {within_2}')
```

```
4/4 [==============================] - 0s 0s/step - loss: 14.4959
Test Loss: 14.49594497680664
4/4 [==============================] - 0s 930us/step
Accuracy: 0.30392156862745096
Number of predictions within 1 unit of the actual value: 31
Number of predictions within 2 units of the actual value: 54
```

Changes from last model to this model are the layers, and some Neuron changes

In [ ]:
```python
# Remaking the model

model = Sequential()

model.add(Dense(128,activation  = 'relu',input_dim =13))
model.add(Dense(64,activation  = 'relu'))
model.add(Dense(32,activation  = 'relu'))
model.add(Dense(16,activation  = 'relu'))
model.add(Dense(1))
model.compile(optimizer = 'adam',loss = 'mean_squared_error')
```

In [ ]:
```python
history_2 = model.fit(X_train, y_train, epochs=30, batch_size=32, validation_data=(
```

```
Epoch 1/30
11/11 [==============================] - 1s 23ms/step - loss: 588.3419 - val_loss: 5
45.6750
Epoch 2/30
11/11 [==============================] - 0s 5ms/step - loss: 531.7380 - val_loss: 46
0.5020
Epoch 3/30
11/11 [==============================] - 0s 5ms/step - loss: 412.8017 - val_loss: 29
4.7219
Epoch 4/30
11/11 [==============================] - 0s 4ms/step - loss: 218.0894 - val_loss: 9
6.3502
Epoch 5/30
11/11 [==============================] - 0s 7ms/step - loss: 89.6801 - val_loss: 75.
1359
Epoch 6/30
11/11 [==============================] - 0s 4ms/step - loss: 60.0053 - val_loss: 48.
9807
Epoch 7/30
11/11 [==============================] - 0s 4ms/step - loss: 37.4621 - val_loss: 42.
7141
Epoch 8/30
11/11 [==============================] - 0s 4ms/step - loss: 27.4579 - val_loss: 37.
4692
Epoch 9/30
11/11 [==============================] - 0s 4ms/step - loss: 25.7285 - val_loss: 34.
3558
Epoch 10/30
11/11 [==============================] - 0s 5ms/step - loss: 22.9786 - val_loss: 33.
4034
Epoch 11/30
11/11 [==============================] - 0s 4ms/step - loss: 21.7818 - val_loss: 30.
5112
Epoch 12/30
11/11 [==============================] - 0s 5ms/step - loss: 20.3454 - val_loss: 28.
2733
Epoch 13/30
11/11 [==============================] - 0s 5ms/step - loss: 19.0601 - val_loss: 27.
0140
Epoch 14/30
11/11 [==============================] - 0s 5ms/step - loss: 17.9144 - val_loss: 25.
5732
Epoch 15/30
11/11 [==============================] - 0s 5ms/step - loss: 17.0078 - val_loss: 23.
6689
Epoch 16/30
11/11 [==============================] - 0s 5ms/step - loss: 16.0449 - val_loss: 23.
1695
Epoch 17/30
11/11 [==============================] - 0s 5ms/step - loss: 15.3868 - val_loss: 22.
0793
Epoch 18/30
11/11 [==============================] - 0s 5ms/step - loss: 14.5824 - val_loss: 21.
2954
Epoch 19/30
11/11 [==============================] - 0s 6ms/step - loss: 14.3939 - val_loss: 19.
```

```
8958
Epoch 20/30
11/11 [==============================] - 0s 4ms/step - loss: 13.3768 - val_loss: 19.
5425
Epoch 21/30
11/11 [==============================] - 0s 5ms/step - loss: 13.0104 - val_loss: 18.
6378
Epoch 22/30
11/11 [==============================] - 0s 5ms/step - loss: 12.3661 - val_loss: 18.
3879
Epoch 23/30
11/11 [==============================] - 0s 5ms/step - loss: 12.8882 - val_loss: 17.
9675
Epoch 24/30
11/11 [==============================] - 0s 4ms/step - loss: 12.7901 - val_loss: 17.
2162
Epoch 25/30
11/11 [==============================] - 0s 5ms/step - loss: 11.6812 - val_loss: 17.
6526
Epoch 26/30
11/11 [==============================] - 0s 4ms/step - loss: 10.9752 - val_loss: 16.
5630
Epoch 27/30
11/11 [==============================] - 0s 4ms/step - loss: 10.6874 - val_loss: 17.
4273
Epoch 28/30
11/11 [==============================] - 0s 5ms/step - loss: 10.4499 - val_loss: 16.
2808
Epoch 29/30
11/11 [==============================] - 0s 5ms/step - loss: 10.2829 - val_loss: 16.
7906
Epoch 30/30
11/11 [==============================] - 0s 5ms/step - loss: 10.7946 - val_loss: 16.
6116
Epoch 1/30
11/11 [==============================] - 0s 10ms/step - loss: 10.8377 - val_loss: 1
6.4196
Epoch 2/30
11/11 [==============================] - 0s 5ms/step - loss: 10.2909 - val_loss: 15.
5938
Epoch 3/30
11/11 [==============================] - 0s 5ms/step - loss: 9.6397 - val_loss: 15.4
163
Epoch 4/30
11/11 [==============================] - 0s 5ms/step - loss: 9.2791 - val_loss: 14.9
969
Epoch 5/30
11/11 [==============================] - 0s 5ms/step - loss: 9.6384 - val_loss: 15.8
252
Epoch 6/30
11/11 [==============================] - 0s 5ms/step - loss: 9.1752 - val_loss: 14.7
326
Epoch 7/30
11/11 [==============================] - 0s 4ms/step - loss: 9.0926 - val_loss: 15.5
801
Epoch 8/30
```

```
11/11 [==============================] - 0s 4ms/step - loss: 8.7992 - val_loss: 15.2
619
Epoch 9/30
11/11 [==============================] - 0s 6ms/step - loss: 9.0246 - val_loss: 15.1
838
Epoch 10/30
11/11 [==============================] - 0s 5ms/step - loss: 8.5751 - val_loss: 14.5
738
Epoch 11/30
11/11 [==============================] - 0s 5ms/step - loss: 8.3242 - val_loss: 14.6
139
Epoch 12/30
11/11 [==============================] - 0s 4ms/step - loss: 8.1809 - val_loss: 15.0
292
Epoch 13/30
11/11 [==============================] - 0s 4ms/step - loss: 8.0637 - val_loss: 14.5
290
Epoch 14/30
11/11 [==============================] - 0s 4ms/step - loss: 7.8722 - val_loss: 14.2
272
Epoch 15/30
11/11 [==============================] - 0s 4ms/step - loss: 7.7165 - val_loss: 14.3
924
Epoch 16/30
11/11 [==============================] - 0s 5ms/step - loss: 7.4542 - val_loss: 14.1
793
Epoch 17/30
11/11 [==============================] - 0s 5ms/step - loss: 7.9725 - val_loss: 14.7
732
Epoch 18/30
11/11 [==============================] - 0s 4ms/step - loss: 7.3044 - val_loss: 14.1
233
Epoch 19/30
11/11 [==============================] - 0s 4ms/step - loss: 7.3651 - val_loss: 14.4
197
Epoch 20/30
11/11 [==============================] - 0s 4ms/step - loss: 7.7228 - val_loss: 14.5
469
Epoch 21/30
11/11 [==============================] - 0s 4ms/step - loss: 7.4771 - val_loss: 13.8
090
Epoch 22/30
11/11 [==============================] - 0s 4ms/step - loss: 7.3232 - val_loss: 15.5
535
Epoch 23/30
11/11 [==============================] - 0s 5ms/step - loss: 6.6064 - val_loss: 13.6
950
Epoch 24/30
11/11 [==============================] - 0s 5ms/step - loss: 6.5774 - val_loss: 14.7
821
Epoch 25/30
11/11 [==============================] - 0s 4ms/step - loss: 7.0229 - val_loss: 13.6
458
Epoch 26/30
11/11 [==============================] - 0s 4ms/step - loss: 7.4412 - val_loss: 15.0
410
```

```
Epoch 27/30
11/11 [==============================] - 0s 4ms/step - loss: 6.5459 - val_loss: 14.1
912
Epoch 28/30
11/11 [==============================] - 0s 4ms/step - loss: 6.2868 - val_loss: 14.0
580
Epoch 29/30
11/11 [==============================] - 0s 4ms/step - loss: 6.1582 - val_loss: 14.4
981
Epoch 30/30
11/11 [==============================] - 0s 5ms/step - loss: 6.0266 - val_loss: 14.0
386
```

In [ ]:
```python
# Evaluating the model on the test data
test_loss = model.evaluate(X_test, y_test)
print(f'Test Loss: {test_loss}')
```

```
4/4 [==============================] - 0s 1ms/step - loss: 11.7100
Test Loss: 11.709955215454102
```

In [ ]:
```python
# printing the RMSE of the model to see how well it did
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f'RMSE: {rmse}')
```

```
RMSE: 3.8073540721824974
```

In [ ]:
```python
history_3 = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(
```

```
Epoch 1/50
11/11 [==============================] - 0s 9ms/step - loss: 0.6559 - val_loss: 14.4
762
Epoch 2/50
11/11 [==============================] - 0s 4ms/step - loss: 0.5971 - val_loss: 14.9
040
Epoch 3/50
11/11 [==============================] - 0s 4ms/step - loss: 0.6412 - val_loss: 14.7
191
Epoch 4/50
11/11 [==============================] - 0s 7ms/step - loss: 0.7789 - val_loss: 14.5
413
Epoch 5/50
11/11 [==============================] - 0s 4ms/step - loss: 0.6663 - val_loss: 14.8
518
Epoch 6/50
11/11 [==============================] - 0s 4ms/step - loss: 1.3298 - val_loss: 13.8
415
Epoch 7/50
11/11 [==============================] - 0s 5ms/step - loss: 1.0599 - val_loss: 14.8
756
Epoch 8/50
11/11 [==============================] - 0s 5ms/step - loss: 0.8532 - val_loss: 14.3
045
Epoch 9/50
11/11 [==============================] - 0s 4ms/step - loss: 0.7931 - val_loss: 15.0
264
Epoch 10/50
11/11 [==============================] - 0s 5ms/step - loss: 0.7634 - val_loss: 15.2
047
Epoch 11/50
11/11 [==============================] - 0s 4ms/step - loss: 0.7634 - val_loss: 14.7
785
Epoch 12/50
11/11 [==============================] - 0s 4ms/step - loss: 0.6421 - val_loss: 14.8
823
Epoch 13/50
11/11 [==============================] - 0s 4ms/step - loss: 0.6147 - val_loss: 14.4
302
Epoch 14/50
11/11 [==============================] - 0s 4ms/step - loss: 0.5813 - val_loss: 14.9
996
Epoch 15/50
11/11 [==============================] - 0s 4ms/step - loss: 0.6096 - val_loss: 15.4
463
Epoch 16/50
11/11 [==============================] - 0s 4ms/step - loss: 0.6281 - val_loss: 15.1
094
Epoch 17/50
11/11 [==============================] - 0s 4ms/step - loss: 0.6263 - val_loss: 14.0
708
Epoch 18/50
11/11 [==============================] - 0s 4ms/step - loss: 0.6869 - val_loss: 14.3
357
Epoch 19/50
11/11 [==============================] - 0s 5ms/step - loss: 0.6455 - val_loss: 14.1
```

```
159
Epoch 20/50
11/11 [==============================] - 0s 4ms/step - loss: 0.6526 - val_loss: 14.2
305
Epoch 21/50
11/11 [==============================] - 0s 4ms/step - loss: 0.6548 - val_loss: 15.3
858
Epoch 22/50
11/11 [==============================] - 0s 4ms/step - loss: 1.6285 - val_loss: 15.2
094
Epoch 23/50
11/11 [==============================] - 0s 4ms/step - loss: 1.1663 - val_loss: 14.4
259
Epoch 24/50
11/11 [==============================] - 0s 4ms/step - loss: 0.9163 - val_loss: 14.4
686
Epoch 25/50
11/11 [==============================] - 0s 4ms/step - loss: 0.8031 - val_loss: 14.0
669
Epoch 26/50
11/11 [==============================] - 0s 4ms/step - loss: 0.6784 - val_loss: 14.5
088
Epoch 27/50
11/11 [==============================] - 0s 4ms/step - loss: 0.6400 - val_loss: 15.2
489
Epoch 28/50
11/11 [==============================] - 0s 4ms/step - loss: 0.7130 - val_loss: 15.1
318
Epoch 29/50
11/11 [==============================] - 0s 4ms/step - loss: 0.7007 - val_loss: 15.7
727
Epoch 30/50
11/11 [==============================] - 0s 4ms/step - loss: 0.8867 - val_loss: 15.4
347
Epoch 31/50
11/11 [==============================] - 0s 4ms/step - loss: 0.8480 - val_loss: 16.2
665
Epoch 32/50
11/11 [==============================] - 0s 4ms/step - loss: 1.1215 - val_loss: 14.3
909
Epoch 33/50
11/11 [==============================] - 0s 4ms/step - loss: 0.8169 - val_loss: 14.0
284
Epoch 34/50
11/11 [==============================] - 0s 4ms/step - loss: 0.7070 - val_loss: 14.3
997
Epoch 35/50
11/11 [==============================] - 0s 4ms/step - loss: 0.5707 - val_loss: 14.8
447
Epoch 36/50
11/11 [==============================] - 0s 4ms/step - loss: 0.6880 - val_loss: 14.8
982
Epoch 37/50
11/11 [==============================] - 0s 4ms/step - loss: 0.6409 - val_loss: 14.2
838
Epoch 38/50
```

```
11/11 [==============================] - 0s 4ms/step - loss: 0.5610 - val_loss: 14.7
378
Epoch 39/50
11/11 [==============================] - 0s 5ms/step - loss: 0.5106 - val_loss: 14.3
997
Epoch 40/50
11/11 [==============================] - 0s 4ms/step - loss: 0.5759 - val_loss: 14.8
542
Epoch 41/50
11/11 [==============================] - 0s 4ms/step - loss: 0.6197 - val_loss: 15.0
483
Epoch 42/50
11/11 [==============================] - 0s 4ms/step - loss: 0.5329 - val_loss: 14.2
638
Epoch 43/50
11/11 [==============================] - 0s 4ms/step - loss: 0.4604 - val_loss: 14.0
627
Epoch 44/50
11/11 [==============================] - 0s 4ms/step - loss: 0.4916 - val_loss: 14.1
327
Epoch 45/50
11/11 [==============================] - 0s 4ms/step - loss: 0.6179 - val_loss: 14.4
008
Epoch 46/50
11/11 [==============================] - 0s 4ms/step - loss: 0.5440 - val_loss: 14.5
882
Epoch 47/50
11/11 [==============================] - 0s 4ms/step - loss: 0.4913 - val_loss: 14.8
237
Epoch 48/50
11/11 [==============================] - 0s 4ms/step - loss: 0.9561 - val_loss: 14.7
669
Epoch 49/50
11/11 [==============================] - 0s 4ms/step - loss: 1.0440 - val_loss: 13.8
509
Epoch 50/50
11/11 [==============================] - 0s 4ms/step - loss: 0.8616 - val_loss: 14.8
358
```

In [ ]:
```python
# Evaluating the model on the test data
test_loss = model.evaluate(X_test, y_test)
print(f'Test Loss: {test_loss}')
```

```
4/4 [==============================] - 0s 4ms/step - loss: 13.0513
Test Loss: 13.051252365112305
```

Our test loss from the 2nd model to this model actually got worse so the second model is the better model for our needs

Section 2

`Part 2: Classification Problem`

You are required to build a deep neural network model to classify whether a given house is expensive (1) or not (0) based on the given features in the dataset.

`Follow the steps below:`

- Load the Boston Housing dataset from the Keras library.
- Explore and preprocess the data (e.g., normalization, one-hot encoding, etc.).
- Convert the target variable into a binary variable (i.e., expensive or not expensive).
- Split the data into training and testing sets.
- Define a deep neural network architecture for classification using Keras.
- Train the model on the training set and evaluate its performance on the testing set.
- Tune the hyperparameters of the model to achieve better performance (e.g., number of hidden layers, activation functions, learning rate, number of epochs, etc.).
- Compare the performance of the tuned model with the baseline model (i.e., the initial model without any hyperparameter tuning).

In [ ]:
```
# loading the data again for a fresh start
df = pd.read_csv('BostonHousing.csv')
df.fillna(df.mean(), inplace=True)
df.head()
```

Out[ ]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	

In [ ]:
```
# making a cost column
df['cost'] = df['medv'] * 1000
df.head()
```

Out[ ]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	

In [ ]:
```
# making a new column called expensive where if the cost is over the mean cost it i
mean_cost = df['cost'].mean()
df['expensive'] = np.where(df['cost'] > mean_cost, 1, 0)
df.head()
```

Out[ ]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	

In [ ]:
```
# dropping the medv and cost columns from the dataframe
df.drop(['medv', 'cost'], axis=1, inplace=True)
df.head()
```

Out[ ]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	

In [ ]:
```
# expensive will be the target column for the classification model
X = df.drop('expensive', axis=1)
y = df['expensive']

# splitting the data to train and test sets for the classification model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# scaling the data for the classification model
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# another split for the classification model
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2,
```

In [ ]:
```
# making the classification model with tensorflow same as the regression model but
model = Sequential()
model.add(Dense(128, activation='relu', input_dim=13))
model.add(Dense(64, activation='relu'))
```

```
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

In [ ]:
```
# compiling the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [ ]:
```
# training the model
classifier_1_history = model.fit(X_train, y_train, epochs=30, batch_size=32, valida
```

```
Epoch 1/30
11/11 [==============================] - 1s 23ms/step - loss: 0.6208 - accuracy: 0.7
152 - val_loss: 0.5548 - val_accuracy: 0.7901
Epoch 2/30
11/11 [==============================] - 0s 5ms/step - loss: 0.4740 - accuracy: 0.82
66 - val_loss: 0.4870 - val_accuracy: 0.8148
Epoch 3/30
11/11 [==============================] - 0s 5ms/step - loss: 0.3766 - accuracy: 0.86
38 - val_loss: 0.4609 - val_accuracy: 0.8519
Epoch 4/30
11/11 [==============================] - 0s 4ms/step - loss: 0.3277 - accuracy: 0.88
54 - val_loss: 0.4260 - val_accuracy: 0.8395
Epoch 5/30
11/11 [==============================] - 0s 3ms/step - loss: 0.2962 - accuracy: 0.88
24 - val_loss: 0.4069 - val_accuracy: 0.8395
Epoch 6/30
11/11 [==============================] - 0s 4ms/step - loss: 0.2753 - accuracy: 0.90
40 - val_loss: 0.3902 - val_accuracy: 0.8395
Epoch 7/30
11/11 [==============================] - 0s 5ms/step - loss: 0.2604 - accuracy: 0.90
09 - val_loss: 0.3755 - val_accuracy: 0.8519
Epoch 8/30
11/11 [==============================] - 0s 5ms/step - loss: 0.2419 - accuracy: 0.91
02 - val_loss: 0.3538 - val_accuracy: 0.8519
Epoch 9/30
11/11 [==============================] - 0s 3ms/step - loss: 0.2497 - accuracy: 0.88
54 - val_loss: 0.3711 - val_accuracy: 0.8642
Epoch 10/30
11/11 [==============================] - 0s 4ms/step - loss: 0.2318 - accuracy: 0.89
47 - val_loss: 0.3621 - val_accuracy: 0.8395
Epoch 11/30
11/11 [==============================] - 0s 5ms/step - loss: 0.2042 - accuracy: 0.92
26 - val_loss: 0.3411 - val_accuracy: 0.8765
Epoch 12/30
11/11 [==============================] - 0s 4ms/step - loss: 0.2054 - accuracy: 0.91
64 - val_loss: 0.3815 - val_accuracy: 0.8765
Epoch 13/30
11/11 [==============================] - 0s 5ms/step - loss: 0.1904 - accuracy: 0.92
57 - val_loss: 0.3525 - val_accuracy: 0.8889
Epoch 14/30
11/11 [==============================] - 0s 5ms/step - loss: 0.1801 - accuracy: 0.92
88 - val_loss: 0.3588 - val_accuracy: 0.8765
Epoch 15/30
11/11 [==============================] - 0s 3ms/step - loss: 0.1934 - accuracy: 0.91
64 - val_loss: 0.3691 - val_accuracy: 0.8765
Epoch 16/30
11/11 [==============================] - 0s 4ms/step - loss: 0.1762 - accuracy: 0.93
81 - val_loss: 0.3632 - val_accuracy: 0.8642
Epoch 17/30
11/11 [==============================] - 0s 4ms/step - loss: 0.1895 - accuracy: 0.92
26 - val_loss: 0.3451 - val_accuracy: 0.8889
Epoch 18/30
11/11 [==============================] - 0s 4ms/step - loss: 0.1677 - accuracy: 0.92
26 - val_loss: 0.3546 - val_accuracy: 0.8765
Epoch 19/30
11/11 [==============================] - 0s 3ms/step - loss: 0.1717 - accuracy: 0.91
```

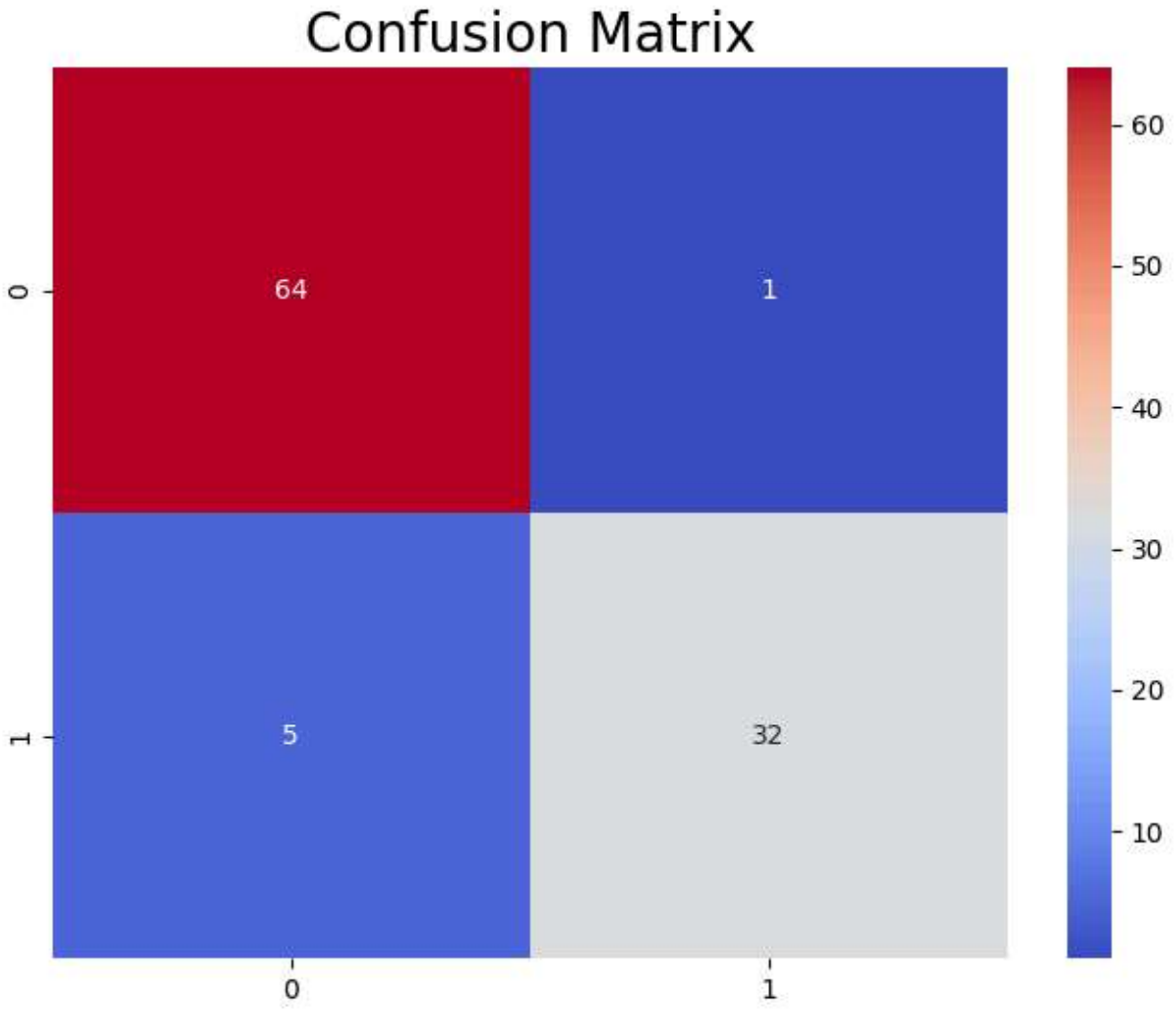
```
64 - val_loss: 0.3464 - val_accuracy: 0.8765
Epoch 20/30
11/11 [==============================] - 0s 4ms/step - loss: 0.1538 - accuracy: 0.94
74 - val_loss: 0.3450 - val_accuracy: 0.8889
Epoch 21/30
11/11 [==============================] - 0s 5ms/step - loss: 0.1404 - accuracy: 0.95
98 - val_loss: 0.3766 - val_accuracy: 0.8765
Epoch 22/30
11/11 [==============================] - 0s 4ms/step - loss: 0.1311 - accuracy: 0.95
36 - val_loss: 0.3578 - val_accuracy: 0.8642
Epoch 23/30
11/11 [==============================] - 0s 4ms/step - loss: 0.1245 - accuracy: 0.95
36 - val_loss: 0.3532 - val_accuracy: 0.9012
Epoch 24/30
11/11 [==============================] - 0s 4ms/step - loss: 0.1282 - accuracy: 0.96
28 - val_loss: 0.3467 - val_accuracy: 0.9012
Epoch 25/30
11/11 [==============================] - 0s 5ms/step - loss: 0.1268 - accuracy: 0.94
43 - val_loss: 0.3838 - val_accuracy: 0.8765
Epoch 26/30
11/11 [==============================] - 0s 4ms/step - loss: 0.1128 - accuracy: 0.95
67 - val_loss: 0.3720 - val_accuracy: 0.9012
Epoch 27/30
11/11 [==============================] - 0s 4ms/step - loss: 0.1100 - accuracy: 0.95
98 - val_loss: 0.3764 - val_accuracy: 0.8765
Epoch 28/30
11/11 [==============================] - 0s 4ms/step - loss: 0.1088 - accuracy: 0.96
28 - val_loss: 0.4130 - val_accuracy: 0.8765
Epoch 29/30
11/11 [==============================] - 0s 4ms/step - loss: 0.1054 - accuracy: 0.96
90 - val_loss: 0.3988 - val_accuracy: 0.8889
Epoch 30/30
11/11 [==============================] - 0s 3ms/step - loss: 0.0933 - accuracy: 0.97
52 - val_loss: 0.3852 - val_accuracy: 0.8642
```

In [ ]:

```python
# now evaluating the model using sklearn's classification report
from sklearn.metrics import classification_report

# making predictions
y_pred = model.predict(X_test)
y_pred = np.where(y_pred > 0.5, 1, 0)

# printing the classification report
print(classification_report(y_test, y_pred))
```

```
4/4 [==============================] - 0s 2ms/step
              precision    recall  f1-score   support

           0       0.93      0.98      0.96        65
           1       0.97      0.86      0.91        37

    accuracy                           0.94       102
   macro avg       0.95      0.92      0.93       102
weighted avg       0.94      0.94      0.94       102
```

In [ ]:
```
# making a confusion matrix to see how well the model did
from sklearn.metrics import confusion_matrix

conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='coolwarm')
plt.title('Confusion Matrix', fontsize=20)
plt.show()
```



In [ ]:
```
# making one more model with one more layer to see if we can get better results
model = Sequential()
model.add(Dense(128, activation='relu', input_dim=13))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

In [ ]:
```
# compiling the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [ ]:
```
# training the model
classifier_2_history = model.fit(X_train, y_train, epochs=30, batch_size=32, valida
```

```
Epoch 1/30
11/11 [==============================] - 1s 27ms/step - loss: 0.6370 - accuracy: 0.6
935 - val_loss: 0.5907 - val_accuracy: 0.7901
Epoch 2/30
11/11 [==============================] - 0s 4ms/step - loss: 0.5301 - accuracy: 0.79
88 - val_loss: 0.5234 - val_accuracy: 0.8272
Epoch 3/30
11/11 [==============================] - 0s 4ms/step - loss: 0.4454 - accuracy: 0.83
59 - val_loss: 0.4917 - val_accuracy: 0.8272
Epoch 4/30
11/11 [==============================] - 0s 4ms/step - loss: 0.3757 - accuracy: 0.86
38 - val_loss: 0.4424 - val_accuracy: 0.8395
Epoch 5/30
11/11 [==============================] - 0s 4ms/step - loss: 0.3327 - accuracy: 0.88
24 - val_loss: 0.4160 - val_accuracy: 0.8519
Epoch 6/30
11/11 [==============================] - 0s 4ms/step - loss: 0.3037 - accuracy: 0.88
85 - val_loss: 0.3886 - val_accuracy: 0.8642
Epoch 7/30
11/11 [==============================] - 0s 7ms/step - loss: 0.2759 - accuracy: 0.89
16 - val_loss: 0.3901 - val_accuracy: 0.8642
Epoch 8/30
11/11 [==============================] - 0s 4ms/step - loss: 0.2659 - accuracy: 0.89
78 - val_loss: 0.3953 - val_accuracy: 0.8519
Epoch 9/30
11/11 [==============================] - 0s 4ms/step - loss: 0.2532 - accuracy: 0.90
40 - val_loss: 0.3744 - val_accuracy: 0.8642
Epoch 10/30
11/11 [==============================] - 0s 4ms/step - loss: 0.2606 - accuracy: 0.89
47 - val_loss: 0.3808 - val_accuracy: 0.8642
Epoch 11/30
11/11 [==============================] - 0s 4ms/step - loss: 0.2991 - accuracy: 0.85
14 - val_loss: 0.3901 - val_accuracy: 0.8395
Epoch 12/30
11/11 [==============================] - 0s 5ms/step - loss: 0.2516 - accuracy: 0.89
16 - val_loss: 0.3538 - val_accuracy: 0.8765
Epoch 13/30
11/11 [==============================] - 0s 4ms/step - loss: 0.2234 - accuracy: 0.91
02 - val_loss: 0.3977 - val_accuracy: 0.8519
Epoch 14/30
11/11 [==============================] - 0s 5ms/step - loss: 0.2186 - accuracy: 0.90
71 - val_loss: 0.3787 - val_accuracy: 0.8765
Epoch 15/30
11/11 [==============================] - 0s 4ms/step - loss: 0.2046 - accuracy: 0.91
95 - val_loss: 0.3808 - val_accuracy: 0.8765
Epoch 16/30
11/11 [==============================] - 0s 6ms/step - loss: 0.1945 - accuracy: 0.92
57 - val_loss: 0.3853 - val_accuracy: 0.8765
Epoch 17/30
11/11 [==============================] - 0s 4ms/step - loss: 0.1849 - accuracy: 0.92
57 - val_loss: 0.3840 - val_accuracy: 0.8765
Epoch 18/30
11/11 [==============================] - 0s 4ms/step - loss: 0.1778 - accuracy: 0.93
50 - val_loss: 0.3653 - val_accuracy: 0.8765
Epoch 19/30
11/11 [==============================] - 0s 5ms/step - loss: 0.1721 - accuracy: 0.93
```

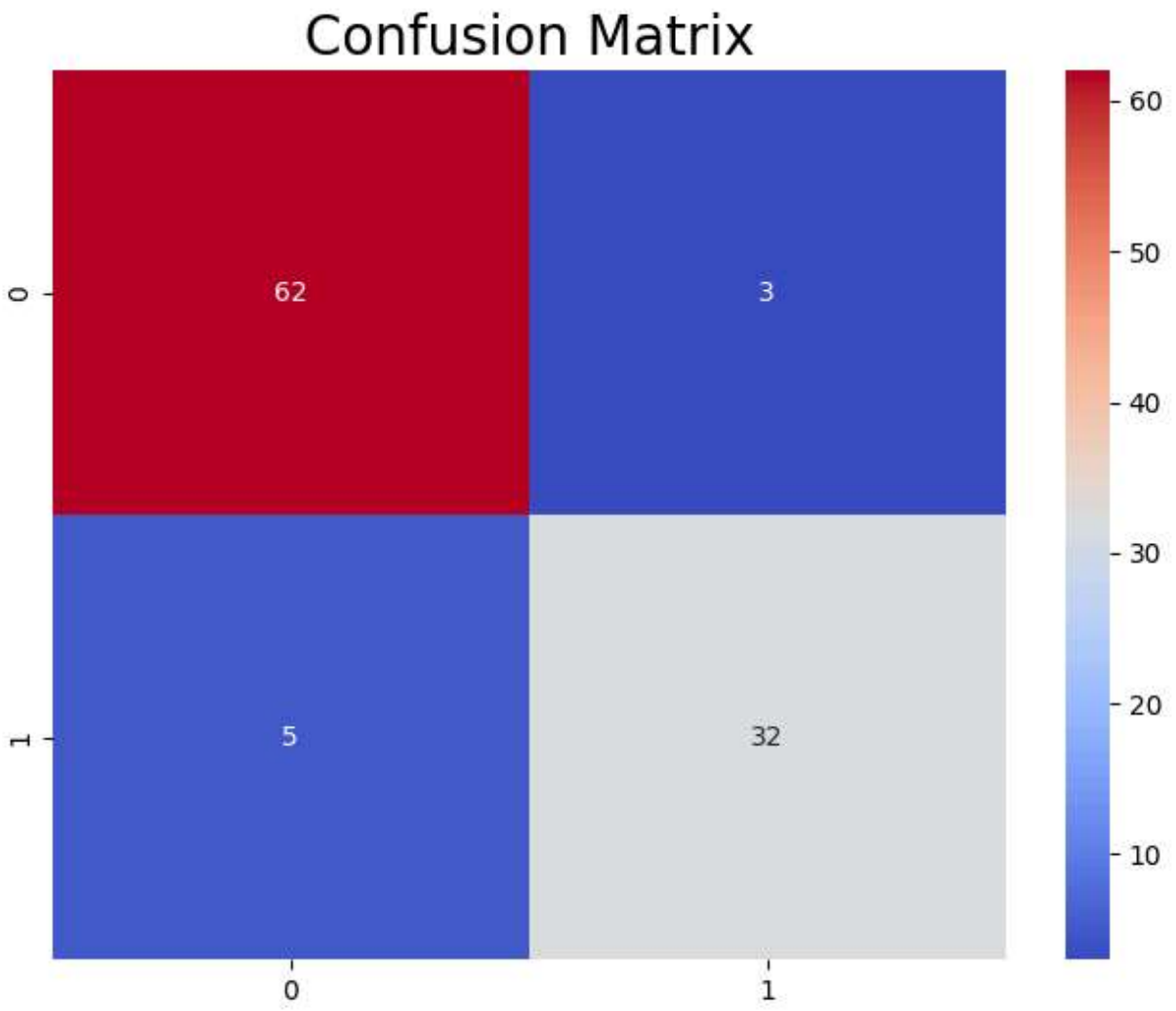
```
50 - val_loss: 0.3727 - val_accuracy: 0.8765
Epoch 20/30
11/11 [==============================] - 0s 7ms/step - loss: 0.1674 - accuracy: 0.93
81 - val_loss: 0.3906 - val_accuracy: 0.8889
Epoch 21/30
11/11 [==============================] - 0s 5ms/step - loss: 0.1564 - accuracy: 0.94
74 - val_loss: 0.4054 - val_accuracy: 0.8642
Epoch 22/30
11/11 [==============================] - 0s 5ms/step - loss: 0.1583 - accuracy: 0.93
50 - val_loss: 0.3861 - val_accuracy: 0.8765
Epoch 23/30
11/11 [==============================] - 0s 5ms/step - loss: 0.1596 - accuracy: 0.93
81 - val_loss: 0.4180 - val_accuracy: 0.8642
Epoch 24/30
11/11 [==============================] - 0s 5ms/step - loss: 0.1498 - accuracy: 0.93
19 - val_loss: 0.4020 - val_accuracy: 0.8642
Epoch 25/30
11/11 [==============================] - 0s 5ms/step - loss: 0.1438 - accuracy: 0.95
36 - val_loss: 0.3953 - val_accuracy: 0.8642
Epoch 26/30
11/11 [==============================] - 0s 5ms/step - loss: 0.1330 - accuracy: 0.94
74 - val_loss: 0.3858 - val_accuracy: 0.8642
Epoch 27/30
11/11 [==============================] - 0s 5ms/step - loss: 0.1359 - accuracy: 0.95
36 - val_loss: 0.4028 - val_accuracy: 0.8765
Epoch 28/30
11/11 [==============================] - 0s 5ms/step - loss: 0.1319 - accuracy: 0.94
12 - val_loss: 0.4177 - val_accuracy: 0.8889
Epoch 29/30
11/11 [==============================] - 0s 4ms/step - loss: 0.1366 - accuracy: 0.95
05 - val_loss: 0.4320 - val_accuracy: 0.8395
Epoch 30/30
11/11 [==============================] - 0s 5ms/step - loss: 0.1261 - accuracy: 0.94
43 - val_loss: 0.4021 - val_accuracy: 0.8765
```

In [ ]:
```python
# pred
y_pred = model.predict(X_test)
y_pred = np.where(y_pred > 0.5, 1, 0)

print(classification_report(y_test, y_pred))
```

```
4/4 [==============================] - 0s 1ms/step
              precision    recall  f1-score   support

           0       0.93      0.95      0.94        65
           1       0.91      0.86      0.89        37

    accuracy                           0.92       102
   macro avg       0.92      0.91      0.91       102
weighted avg       0.92      0.92      0.92       102
```

In [ ]:
```python
# confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='coolwarm')
```

```
plt.title('Confusion Matrix', fontsize=20)
plt.show()
```



All comparisons will be done in the report that is being turned in along with this notebook