LAB3: BINARY CLASSIOFICATION OF HEART DISEASE OF PATIENTS USING DEEP NEURAL NETWORK

In [1]:
```
import pandas as pd
```

In [2]:
```
df = pd.read_csv("heart_data.csv")
df.head(10)
```

Out[2]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
5	57	1	0	140	192	0	1	148	0	0.4	1	0	1	1
6	56	0	1	140	294	0	0	153	0	1.3	1	0	2	1
7	44	1	1	120	263	0	1	173	0	0.0	2	0	3	1
8	52	1	2	172	199	1	1	162	0	0.5	2	0	3	1
9	57	1	2	150	168	0	1	174	0	1.6	2	0	2	1

In [3]:
```
df.shape
```

Out[3]: (303, 14)

In [4]:
```
df.size
```

Out[4]: 4242

In [5]:
```
df.columns
```

Out[5]:
```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

2. Split the dataset for training and testing (test size = 20%)

In [6]:
```
X = df.drop('target',axis=1)
y = df['target']
```

In [8]:
```
X
```

Out[8]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2

303 rows × 13 columns

In [13]:
```
y
```

Out[13]:
```
0      1
1      1
2      1
3      1
4      1
      ..
298    0
299    0
300    0
301    0
302    0
Name: target, Length: 303, dtype: int64
```

In [14]:
```
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_s
```

In [15]:
```
X_train.shape
```

Out[15]:
```
(242, 13)
```

In [12]:
```
X_test.shape
```

Out[12]:
```
(61, 13)
```

3. Create a neural network based on the following requirements

In [16]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

In [17]:
```
model = Sequential()
model.add(Dense(8, input_dim=13, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

4. Compile your model with learning rate = 0.001, optimizer as ‘RMSprop’, Mean square error loss and metrics as ‘accuracy’.

In [18]:
```
from tensorflow import keras
```

In [19]:
```
optimizer = keras.optimizers.RMSprop(learning_rate=0.001)
```

In [20]:
```
model.compile(loss='mse', optimizer=optimizer, metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=30, verbose=1)
```

```
Epoch 1/10
9/9 [==============================] - 3s 9ms/step - loss: 0.5496 - accuracy:
0.4504
Epoch 2/10
9/9 [==============================] - 0s 2ms/step - loss: 0.5496 - accuracy:
0.4504
Epoch 3/10
9/9 [==============================] - 0s 2ms/step - loss: 0.5496 - accuracy:
0.4504
Epoch 4/10
9/9 [==============================] - 0s 1ms/step - loss: 0.5496 - accuracy:
0.4504
Epoch 5/10
9/9 [==============================] - 0s 1ms/step - loss: 0.5496 - accuracy:
0.4504
Epoch 6/10
9/9 [==============================] - 0s 1ms/step - loss: 0.5496 - accuracy:
0.4504
Epoch 7/10
9/9 [==============================] - 0s 1ms/step - loss: 0.5496 - accuracy:
0.4504
Epoch 8/10
9/9 [==============================] - 0s 2ms/step - loss: 0.5496 - accuracy:
0.4504
Epoch 9/10
9/9 [==============================] - 0s 1ms/step - loss: 0.5496 - accuracy:
0.4504
Epoch 10/10
9/9 [==============================] - 0s 2ms/step - loss: 0.5496 - accuracy:
0.4504
```

Out[20]:
```
<keras.callbacks.History at 0x1d7869cf0d0>
```

In [21]:
```
model.evaluate(X_test, y_test)
```

```
2/2 [==============================] - 0s 3ms/step - loss: 0.5246 - accuracy:
0.4754
```

Out[21]:
```
[0.5245901346206665, 0.4754098355770111]
```

5. Print the summary of the model: model.summary()

In [22]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 8)                 112

 dense_1 (Dense)             (None, 1)                 9

=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
```

6. Train the model for 200 epochs and batch size as 10

In [23]:
```
model.compile(loss='mse', optimizer=optimizer, metrics=['accuracy'])
model.fit(X_train, y_train, epochs=200, batch_size=10, verbose=1)
```

```
y: 0.4504
Epoch 13/200
25/25 [==============================] - 0s 1ms/step - loss: 0.5496 - accurac
y: 0.4504
Epoch 14/200
25/25 [==============================] - 0s 1ms/step - loss: 0.5496 - accurac
y: 0.4504
Epoch 15/200
25/25 [==============================] - 0s 1ms/step - loss: 0.5496 - accurac
y: 0.4504
Epoch 16/200
25/25 [==============================] - 0s 1ms/step - loss: 0.5496 - accurac
y: 0.4504
Epoch 17/200
25/25 [==============================] - 0s 1ms/step - loss: 0.5496 - accurac
y: 0.4504
Epoch 18/200
25/25 [==============================] - 0s 1ms/step - loss: 0.5496 - accurac
y: 0.4504
```

In [24]:
```
model.evaluate(X_test, y_test)
```

```
2/2 [==============================] - 0s 2ms/step - loss: 0.5246 - accuracy:
0.4754
```

Out[24]:
```
[0.5245901346206665, 0.47540983557701111]
```

7. Save the trained model in a variable, such as, history. Also, you can split your training data for validation such as 20% of training data.

In [26]:
```
history = model.fit(X_train, y_train, validation_split=0.2, epochs=100, batch_siz
```

```
Epoch 1/100
20/20 [==============================] - 0s 6ms/step - loss: 0.5440 - accurac
y: 0.4560 - val_loss: 0.5714 - val_accuracy: 0.4286
Epoch 2/100
20/20 [==============================] - 0s 3ms/step - loss: 0.5440 - accurac
y: 0.4560 - val_loss: 0.5714 - val_accuracy: 0.4286
Epoch 3/100
20/20 [==============================] - 0s 3ms/step - loss: 0.5440 - accurac
y: 0.4560 - val_loss: 0.5714 - val_accuracy: 0.4286
Epoch 4/100
20/20 [==============================] - 0s 3ms/step - loss: 0.5440 - accurac
y: 0.4560 - val_loss: 0.5714 - val_accuracy: 0.4286
Epoch 5/100
20/20 [==============================] - 0s 3ms/step - loss: 0.5440 - accurac
y: 0.4560 - val_loss: 0.5714 - val_accuracy: 0.4286
Epoch 6/100
20/20 [==============================] - 0s 3ms/step - loss: 0.5440 - accurac
y: 0.4560 - val_loss: 0.5714 - val_accuracy: 0.4286
Epoch 7/100
```

8. Evaluate the trained model to predict the probability values for the test data set (ie., xtest and ytest)

In [27]:
```
model.evaluate(X_test, y_test)
```

```
2/2 [==============================] - 0s 3ms/step - loss: 0.5246 - accuracy:
0.4754
```

Out[27]:
```
[0.5245901346206665, 0.47540983557701111]
```

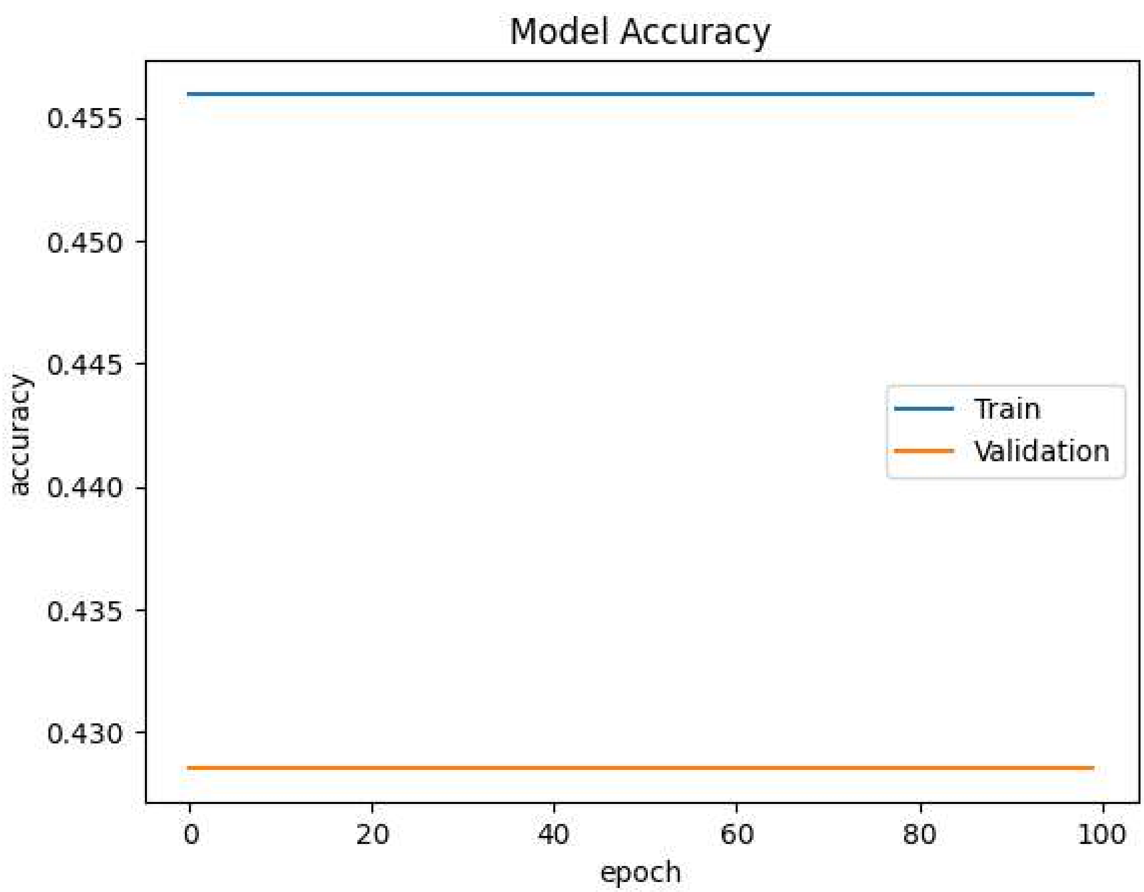
9. Print the model accuracy and model loss as below (Use can use the ‘history’ object we have saved). Sample code is given below.

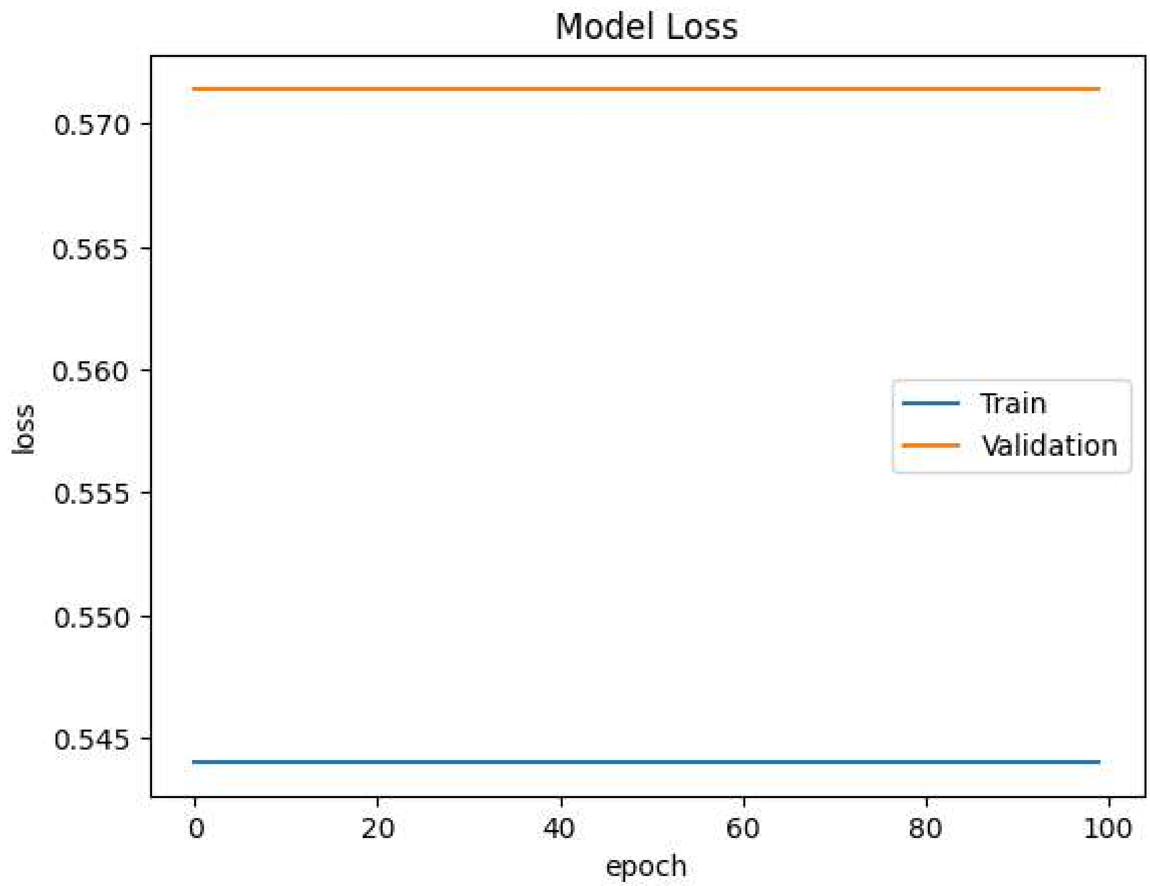
In [28]:
```
history.history.keys()
```

Out[28]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [31]:
```
import matplotlib.pyplot as plt
```

In [32]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'])
plt.show()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'])
plt.show()
```





10. Do further experiments

In [33]:
```
model1 = Sequential()
model1.add(Dense(16, input_dim=13, activation='relu'))
model1.add(Dense(8, activation='relu'))
model1.add(Dense(1, activation='sigmoid'))
```

In [34]:
```
model1.compile(loss='mse', optimizer=optimizer, metrics=['accuracy'])
model1.fit(X_train, y_train, epochs=10, batch_size=30, verbose=1)
```

```
Epoch 1/10
9/9 [==============================] - 1s 1ms/step - loss: 0.4397 - accuracy:
0.5207
Epoch 2/10
9/9 [==============================] - 0s 1ms/step - loss: 0.3482 - accuracy:
0.5868
Epoch 3/10
9/9 [==============================] - 0s 2ms/step - loss: 0.3434 - accuracy:
0.5992
Epoch 4/10
9/9 [==============================] - 0s 1ms/step - loss: 0.3535 - accuracy:
0.5992
Epoch 5/10
9/9 [==============================] - 0s 1ms/step - loss: 0.3287 - accuracy:
0.6157
Epoch 6/10
9/9 [==============================] - 0s 1ms/step - loss: 0.3256 - accuracy:
0.6198
Epoch 7/10
9/9 [==============================] - 0s 1ms/step - loss: 0.3232 - accuracy:
0.6157
Epoch 8/10
9/9 [==============================] - 0s 1ms/step - loss: 0.3080 - accuracy:
0.6446
Epoch 9/10
9/9 [==============================] - 0s 1ms/step - loss: 0.2986 - accuracy:
0.6612
Epoch 10/10
9/9 [==============================] - 0s 1ms/step - loss: 0.3306 - accuracy:
0.6198
```

Out[34]:
```
<keras.callbacks.History at 0x1d78bda9d80>
```

In [35]:
```
model1.evaluate(X_test, y_test)
```

```
2/2 [==============================] - 0s 3ms/step - loss: 0.2617 - accuracy:
0.7049
```

Out[35]:
```
[0.26170825958251953, 0.7049180269241333]
```

In [36]:
```
history1 = model.fit(X_train, y_train, validation_split=0.2, epochs=100, batch_si
```

```
0.4560 - val_loss: 0.5714 - val_accuracy: 0.4286
Epoch 5/100
7/7 [==============================] - 0s 6ms/step - loss: 0.5440 - accuracy:
0.4560 - val_loss: 0.5714 - val_accuracy: 0.4286
Epoch 6/100
7/7 [==============================] - 0s 6ms/step - loss: 0.5440 - accuracy:
0.4560 - val_loss: 0.5714 - val_accuracy: 0.4286
Epoch 7/100
7/7 [==============================] - 0s 6ms/step - loss: 0.5440 - accuracy:
0.4560 - val_loss: 0.5714 - val_accuracy: 0.4286
Epoch 8/100
7/7 [==============================] - 0s 6ms/step - loss: 0.5440 - accuracy:
0.4560 - val_loss: 0.5714 - val_accuracy: 0.4286
Epoch 9/100
7/7 [==============================] - 0s 6ms/step - loss: 0.5440 - accuracy:
0.4560 - val_loss: 0.5714 - val_accuracy: 0.4286
Epoch 10/100
7/7 [==============================] - 0s 6ms/step - loss: 0.5440 - accuracy:
0.4560 - val_loss: 0.5714 - val_accuracy: 0.4286
Epoch 11/100
```

In [37]:
```
model1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 16)                224

 dense_3 (Dense)             (None, 8)                 136

 dense_4 (Dense)             (None, 1)                 9

=================================================================
Total params: 369
Trainable params: 369
Non-trainable params: 0
_________________________________________________________________
```

In [38]:
```
ls = history1.history
```

In [39]:
```
new = pd.DataFrame.from_dict(ls)
new
```

Out[39]:

	loss	accuracy	val_loss	val_accuracy
0	0.544041	0.455959	0.571429	0.428571
1	0.544041	0.455959	0.571429	0.428571
2	0.544041	0.455959	0.571429	0.428571
3	0.544041	0.455959	0.571429	0.428571
4	0.544041	0.455959	0.571429	0.428571
...	...	...	...	...
95	0.544041	0.455959	0.571429	0.428571
96	0.544041	0.455959	0.571429	0.428571
97	0.544041	0.455959	0.571429	0.428571
98	0.544041	0.455959	0.571429	0.428571
99	0.544041	0.455959	0.571429	0.428571

100 rows × 4 columns

In [40]:
```
model2 = Sequential()
model2.add(Dense(32, input_dim=13, activation='relu'))
model2.add(Dense(16, activation='relu'))
model2.add(Dense(8, activation='relu'))
model2.add(Dense(1, activation='sigmoid'))
```

In [41]:
```
model2.compile(loss='mse', optimizer=optimizer, metrics=['accuracy'])
model2.fit(X_train, y_train, epochs=10, batch_size=30, verbose=1)
```

```
Epoch 1/10
9/9 [==============================] - 1s 2ms/step - loss: 0.4083 - accuracy:
0.4711
Epoch 2/10
9/9 [==============================] - 0s 2ms/step - loss: 0.3643 - accuracy:
0.4463
Epoch 3/10
9/9 [==============================] - 0s 2ms/step - loss: 0.3247 - accuracy:
0.4421
Epoch 4/10
9/9 [==============================] - 0s 2ms/step - loss: 0.3017 - accuracy:
0.5083
Epoch 5/10
9/9 [==============================] - 0s 2ms/step - loss: 0.2687 - accuracy:
0.5083
Epoch 6/10
9/9 [==============================] - 0s 2ms/step - loss: 0.2606 - accuracy:
0.5289
Epoch 7/10
9/9 [==============================] - 0s 1ms/step - loss: 0.2603 - accuracy:
0.5248
Epoch 8/10
9/9 [==============================] - 0s 2ms/step - loss: 0.2560 - accuracy:
0.6074
Epoch 9/10
9/9 [==============================] - 0s 2ms/step - loss: 0.2442 - accuracy:
0.5992
Epoch 10/10
9/9 [==============================] - 0s 2ms/step - loss: 0.2316 - accuracy:
0.6281
```

Out[41]:
```
<keras.callbacks.History at 0x1d78d06ed70>
```

In [42]:
```
model2.evaluate(X_test, y_test)
```

```
2/2 [==============================] - 0s 3ms/step - loss: 0.2314 - accuracy:
0.6066
```

Out[42]:
```
[0.23137971758842468, 0.6065573692321777]
```

In [43]:
```
model2.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_5 (Dense)             (None, 32)                448       
                                                                 
 dense_6 (Dense)             (None, 16)                528       
                                                                 
 dense_7 (Dense)             (None, 8)                 136       
                                                                 
 dense_8 (Dense)             (None, 1)                 9         
                                                                 
=================================================================
Total params: 1,121
Trainable params: 1,121
Non-trainable params: 0
_________________________________________________________________
```

In [44]:
```
model3 = Sequential()
model3.add(Dense(64, input_dim=13, activation='relu'))
model3.add(Dense(32, activation='relu'))
model3.add(Dense(16, activation='relu'))
model3.add(Dense(8, activation='relu'))
model3.add(Dense(1, activation='sigmoid'))
```

In [45]:
```
model3.compile(loss='mse', optimizer=optimizer, metrics=['accuracy'])
model3.fit(X_train, y_train, epochs=10, batch_size=30, verbose=1)
```

```
Epoch 1/10
9/9 [==============================] - 1s 2ms/step - loss: 0.4695 - accuracy:
0.4917
Epoch 2/10
9/9 [==============================] - 0s 2ms/step - loss: 0.3186 - accuracy:
0.5702
Epoch 3/10
9/9 [==============================] - 0s 2ms/step - loss: 0.2737 - accuracy:
0.6281
Epoch 4/10
9/9 [==============================] - 0s 2ms/step - loss: 0.2814 - accuracy:
0.5868
Epoch 5/10
9/9 [==============================] - 0s 2ms/step - loss: 0.3501 - accuracy:
0.5289
Epoch 6/10
9/9 [==============================] - 0s 2ms/step - loss: 0.2845 - accuracy:
0.5537
Epoch 7/10
9/9 [==============================] - 0s 2ms/step - loss: 0.2123 - accuracy:
0.6736
Epoch 8/10
9/9 [==============================] - 0s 2ms/step - loss: 0.3161 - accuracy:
0.5992
Epoch 9/10
9/9 [==============================] - 0s 2ms/step - loss: 0.2393 - accuracy:
0.6612
Epoch 10/10
9/9 [==============================] - 0s 1ms/step - loss: 0.2566 - accuracy:
0.5661
```

Out[45]:
```
<keras.callbacks.History at 0x1d78cf7ace0>
```

In [46]:
```
model3.evaluate(X_test, y_test)
```

```
2/2 [==============================] - 0s 4ms/step - loss: 0.3637 - accuracy:
0.4918
```

Out[46]:
```
[0.36367374658584595, 0.49180328845977783]
```

In [47]:
```
model3.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_9 (Dense)             (None, 64)                896       
                                                                 
 dense_10 (Dense)            (None, 32)                2080      
                                                                 
 dense_11 (Dense)            (None, 16)                528       
                                                                 
 dense_12 (Dense)            (None, 8)                 136       
                                                                 
 dense_13 (Dense)            (None, 1)                 9         
                                                                 
=================================================================
Total params: 3,649
Trainable params: 3,649
Non-trainable params: 0
_________________________________________________________________
```

In [ ]: