In [84]:

```

```

Introduction this is a binary classification problem where we predict whether the customer is going to exit any particular bank/company or not where they are used the products so if they dont want them to exit the bank they should provide some more services so we will create a model to predict whether cstomer will exit the bank or not

IMPLEMENTATION OF ANN

In [85]:

```
!pip install tensorflow-gpu
```

```
Collecting tensorflow-gpu
  Using cached tensorflow-gpu-2.12.0.tar.gz (2.6 kB)
  error: subprocess-exited-with-error

  × python setup.py egg_info did not run successfully.
  │ exit code: 1
  ╰─> See above for output.

  note: This error originates from a subprocess, and is likely not a problem with pip.
  Preparing metadata (setup.py) ... error
error: metadata-generation-failed

× Encountered error while generating package metadata.
╰─> See above for output.

note: This is an issue with the package mentioned above, not pip.
hint: See above for details.
```

In [86]:

```
# Step 1: Install TensorFlow
!pip install tensorflow

# Step 2: Verify GPU Availability
import tensorflow as tf

# Check TensorFlow version
print("TensorFlow version:", tf.__version__)

# Check if GPU is available
print("GPU available:", tf.test.is_gpu_available())

# Print GPU device name
print("GPU device name:", tf.test.gpu_device_name())
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.11/dist-packages (2.1
7.1)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-packages
(from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.11/dist-packag
es (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.11/dist-pac
kages (from tensorflow) (25.1.21)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/pyth
on3.11/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.11/dist-pack
ages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=3.10.0 in /usr/local/lib/python3.11/dist-packages (f
rom tensorflow) (3.12.1)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.11/dist-package
s (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in /usr/local/lib/python3.11/dist-
```

```
packages (from tensorflow) (0.4.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.11/dist-packag
es (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from
tensorflow) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.2
1.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (4.25
.5)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.11/dist-pack
ages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (fro
m tensorflow) (75.1.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-packages (fr
om tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist-package
s (from tensorflow) (2.5.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.11/dist
-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-packages (
from tensorflow) (1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/dist-pack
ages (from tensorflow) (1.69.0)
Requirement already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.11/dist-
packages (from tensorflow) (2.17.1)
Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.11/dist-packages (f
rom tensorflow) (3.5.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/pyt
hon3.11/dist-packages (from tensorflow) (0.37.1)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.11/dist-pac
kages (from tensorflow) (1.26.4)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packa
ges (from astunparse>=1.6.0->tensorflow) (0.45.1)
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from kera
s>=3.2.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from ker
as>=3.2.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from ke
ras>=3.2.0->tensorflow) (0.14.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist
-packages (from requests<3,>=2.21.0->tensorflow) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (f
rom requests<3,>=2.21.0->tensorflow) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packa
ges (from requests<3,>=2.21.0->tensorflow) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packa
ges (from requests<3,>=2.21.0->tensorflow) (2024.12.14)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.11/dist-packages
(from tensorboard<2.18,>=2.17->tensorflow) (3.7)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/py
thon3.11/dist-packages (from tensorboard<2.18,>=2.17->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.11/dist-packages
(from tensorboard<2.18,>=2.17->tensorflow) (3.1.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packag
es (from werkzeug>=1.0.1->tensorboard<2.18,>=2.17->tensorflow) (3.0.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-pa
ckages (from rich->keras>=3.2.0->tensorflow) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-
packages (from rich->keras>=3.2.0->tensorflow) (2.18.0)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (fro
m markdown-it-py>=2.2.0->rich->keras>=3.2.0->tensorflow) (0.1.2)
TensorFlow version: 2.17.1
GPU available: False
GPU device name:
```

IMPORT NECESSARY LIBRARIES

In [87]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [88]:

```
dataset=pd.read_csv('/content/Churn_Modelling.csv')
```

In [89]:

```
dataset.head()
```

Out[89]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCar
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	

here in above o/p 'Exited' is dependent feature while others are independent features

In [90]:

```
#now we need to split the dataset in independent and dependet features
X=dataset.iloc[:,3:13]
Y=dataset.iloc[:,13]
```

here for X : [:,3:13] beacuse we want all rows and features from 3 to row 12. in iloc indexing starts from zero and ends with digit next to the features row for example if last column number is 12 then we will write 13 and indexing starts from 0, 1, 2 , 3 and so on.
and for Y : we selected [:,13] beacuse we want all rows and only want 13th column(feaure) the last one which is dependent one

here we do not need to focus on first three features ie.'RowNumber', 'CustomerId', 'Surname' beacuse they wont contribute much to model building while remaning features like 'CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary' which will be our independent features

In [91]:

```
#Inspecting data
X.head()
```

Out[91]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
0	619	France	Female	42	2	0.00	1	1	1	101348.88
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58
2	502	France	Female	42	8	159660.80	3	1	0	113931.57
3	699	France	Female	39	1	0.00	2	0	0	93826.63
4	850	Spain	Female	43	2	125510.82	1	1	1	79084.10

In [92]:

```
Y.head()
```

Out[92]:

Exited

	Exited
0	1
1	0
2	1
3	0
4	0

dtype: int64

here in this data we have a categorical features like 'Geography', 'Gender' so we can use one hot encoding for them or use get_dummies in panda

Feature Engineering

In [93]:

```
Geography=pd.get_dummies(X['Geography'],drop_first=True,dtype=int)
Gender=pd.get_dummies(X['Gender'],drop_first=True ,dtype=int)
```

'drop_first=True' is used here beacuse to remove first column france from here so that there are only two columns germany and france who respresent all three columns

and dtype=int beacuse we want values to be 0 and 1 instead of true or false.

In [94]:

```
Geography
```

Out[94]:

	Germany	Spain
0	0	0
1	0	1
2	0	0
3	0	0
4	0	1
...	...	...
9995	0	0
9996	0	0
9997	0	0
9998	1	0
9999	0	0

10000 rows × 2 columns

In [95]:

```
Gender
```

Out[95]:

	Male
0	0
1	0
2	0

	Male
2	0
3	0
4	0
...	...
9995	1
9996	1
9997	0
9998	1
9999	0

10000 rows × 1 columns

In [96]:

```
#concatinate this variables with dataframe
X=X.drop(['Geography','Gender'],axis=1)
```

In [97]:

```
X
```

Out[97]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
0	619	42	2	0.00	1	1	1	101348.88
1	608	41	1	83807.86	1	0	1	112542.58
2	502	42	8	159660.80	3	1	0	113931.57
3	699	39	1	0.00	2	0	0	93826.63
4	850	43	2	125510.82	1	1	1	79084.10
...	...	...	...	...	...	...	...	...
9995	771	39	5	0.00	2	1	0	96270.64
9996	516	35	10	57369.61	1	1	1	101699.77
9997	709	36	7	0.00	1	0	1	42085.58
9998	772	42	3	75075.31	2	1	0	92888.52
9999	792	28	4	130142.79	1	1	0	38190.78

10000 rows × 8 columns

In [98]:

```
pd.concat([X, Geography, Gender],axis=1)
```

Out[98]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Germany	Spain
0	619	42	2	0.00	1	1	1	101348.88	0	0
1	608	41	1	83807.86	1	0	1	112542.58	0	1
2	502	42	8	159660.80	3	1	0	113931.57	0	0
3	699	39	1	0.00	2	0	0	93826.63	0	0
4	850	43	2	125510.82	1	1	1	79084.10	0	1
...	...	...	...	...	...	...	...	...	...	...
9995	771	39	5	0.00	2	1	0	96270.64	0	0
9996	516	35	10	57369.61	1	1	1	101699.77	0	0

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Germany	Spain
9997	709	36	7	0.00	1	0	1	42085.58	0	0
9998	772	42	3	75075.31	2	1	0	92888.52	1	0
9999	792	28	4	130142.79	1	1	0	38190.78	0	0

10000 rows × 11 columns

In [99]:

```
#ASSIGN THE PREVIOUS CELL VALUE X WHICH WAS PREVIOS X OUR INDEPENDENT FEATURES
X=pd.concat([X, Geography, Gender],axis=1)
```

SPLITTING THE DATASET INTO TRAIN TEST SPLIT

In [100]:

```
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=0)
```

In [101]:

```
#FEATURE SCALING
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

In [102]:

```
X_train
```

Out[102]:

```
array([[ 0.16958176, -0.46460796,  0.00666099, ..., -0.5698444 ,
         1.74309049, -1.09168714],
       [-2.30455945,  0.30102557, -1.37744033, ...,  1.75486502,
        -0.57369368,  0.91601335],
       [-1.19119591, -0.94312892, -1.031415  , ..., -0.5698444 ,
        -0.57369368, -1.09168714],
       ...,
       [ 0.9015152 , -0.36890377,  0.00666099, ..., -0.5698444 ,
        -0.57369368,  0.91601335],
       [-0.62420521, -0.08179119,  1.39076231, ..., -0.5698444 ,
         1.74309049, -1.09168714],
       [-0.28401079,  0.87525072, -1.37744033, ...,  1.75486502,
        -0.57369368, -1.09168714]])
```

In [103]:

```
X_test
```

Out[103]:

```
array([[-0.55204276, -0.36890377,  1.04473698, ...,  1.75486502,
        -0.57369368, -1.09168714],
       [-1.31490297,  0.10961719, -1.031415  , ..., -0.5698444 ,
        -0.57369368, -1.09168714],
       [ 0.57162971,  0.30102557,  1.04473698, ..., -0.5698444 ,
         1.74309049, -1.09168714],
       ...,
       [-0.74791227, -0.27319958, -1.37744033, ..., -0.5698444 ,
         1.74309049,  0.91601335],
       [-0.00566991, -0.46460796, -0.33936434, ...,  1.75486502,
        -0.57369368,  0.91601335],
       [-0.79945688, -0.84742473,  1.04473698, ...,  1.75486502,
        -0.57369368,  0.91601335]])
```

In [104]:

```
X_train.shape,X_test.shape
```

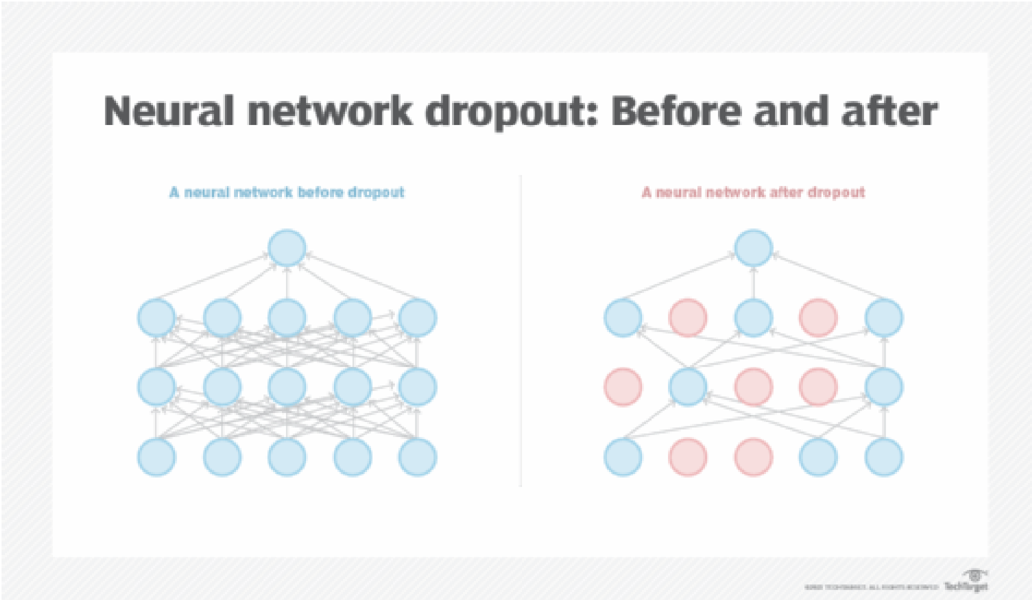
Out[104]:

```
((8000, 11), (2000, 11))
```

BUILDING ANN MODEL

In [105]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LeakyReLU,PReLU,ELU,ReLU
from tensorflow.keras.layers import Dropout
```



neurons help to solve complex problem as this is a binary classification problem by forward and backward propogation and by continiously monitoring the loss function , using optimizers

here

1)we use sequential beacuse ofr forward and backward propogation

2)Dense is used to create dense i/p, hidden and o/p layers

3)LeakyReLU,PReLU,ELU,ReLU are all activation functions used in hidden layer.

4)Droput is used to drop neurons, from above diagram tou can see on left side we have previous neural network and on right side is neural network after some neurons were deactivated or dead so the neurons will be reomoves from model and so the layer will be cutoff between them, it will be of no use and this way entire traing process will go ahead. whenever we are using dropout layer we try to reduce overfitting.so here we use ropout layer like normalization(L1,L2 Norm) in Machine Learning

In [106]:

```
#INITIALIZING ANN
classifier=Sequential()
```

In [107]:

```
#Adding i/p layer
classifier.add(Dense(units=11,activation='relu'))
```

In [108]:

```
#Adding 3 hidden layers
classifier.add(Dense(units=7,activation='relu'))
classifier.add(Dropout(0.3))
```

In [109]:

```
classifier.add(Dense(units=7,activation='relu'))
classifier.add(Dropout(0.2))
```

In [110]:

```
classifier.add(Dense(units=6,activation='relu'))
classifier.add(Dropout(0.2))
```

In [111]:

```
#Adding o/p layer
classifier.add(Dense(units=1,activation='sigmoid'))
```

from X_train.shape we can we there ar 11 inputs so in input layer wi'll have 11 nodes in i/p and in o/p layer as it is a classifvction problem so units=1 and activation function used here in o/p layer is sigmoid

In [112]:

```
#Traing the model
classifier.compile(optimizer='adam',loss='binary_crossentropy', metrics=['accuracy'] )
```

in above step we first compile the entire neural netwrok and use adam optimizer which is best optimizer beacuse it solves smoothening problem and makes sure learning rate is adaptive while reaching global minima in gradient descent graph.

in here while using adam ,adam has a default learning rate of 0.1
and as this is a binary classification problem we use loss function as binary_cross_entropy and accuracy is evaluation metrics.

In [113]:

```
#Early Stopping
import tensorflow as tf
early_stopping=tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    min_delta=0.0001,
    patience=20,
    verbose=1,
    mode="auto",
    baseline=None,
    restore_best_weights=False,
    start_from_epoch=0,
)
```

In [76]:

```
#Train NN
model_history=classifier.fit(X_train,Y_train,validation_split=0.33,batch_size=10,epochs=
1000,callbacks=early_stopping)
```

```
Epoch 1/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step - accuracy: 0.7447 - loss: 0.6082 - val_accuracy
: 0.7955 - val_loss: 0.5015
Epoch 2/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.7945 - loss: 0.5197 - val_accuracy
: 0.7955 - val_loss: 0.4915
Epoch 3/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.8002 - loss: 0.4930 - val_accuracy
: 0.7955 - val_loss: 0.4717
Epoch 4/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.7850 - loss: 0.4966 - val_accuracy
: 0.7955 - val_loss: 0.4498
Epoch 5/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.7976 - loss: 0.4652 - val_accuracy
: 0.7955 - val_loss: 0.4368
Epoch 6/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8094 - loss: 0.4412 - val_accuracy
: 0.7959 - val_loss: 0.4223
```

```
Epoch 7/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8091 - loss: 0.4406 - val_accuracy
: 0.7993 - val_loss: 0.4116
Epoch 8/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.7992 - loss: 0.4394 - val_accuracy
: 0.8183 - val_loss: 0.3987
Epoch 9/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8170 - loss: 0.4233 - val_accuracy
: 0.8292 - val_loss: 0.3944
Epoch 10/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8143 - loss: 0.4112 - val_accuracy
: 0.8243 - val_loss: 0.3930
Epoch 11/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8218 - loss: 0.4117 - val_accuracy
: 0.8307 - val_loss: 0.3894
Epoch 12/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8262 - loss: 0.3926 - val_accuracy
: 0.8273 - val_loss: 0.3922
Epoch 13/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8122 - loss: 0.4134 - val_accuracy
: 0.8353 - val_loss: 0.3871
Epoch 14/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8154 - loss: 0.4078 - val_accuracy
: 0.8372 - val_loss: 0.3838
Epoch 15/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8305 - loss: 0.3947 - val_accuracy
: 0.8383 - val_loss: 0.3824
Epoch 16/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8185 - loss: 0.3990 - val_accuracy
: 0.8376 - val_loss: 0.3844
Epoch 17/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8265 - loss: 0.3856 - val_accuracy
: 0.8387 - val_loss: 0.3826
Epoch 18/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8166 - loss: 0.4126 - val_accuracy
: 0.8410 - val_loss: 0.3809
Epoch 19/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8196 - loss: 0.4118 - val_accuracy
: 0.8372 - val_loss: 0.3823
Epoch 20/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8164 - loss: 0.3989 - val_accuracy
: 0.8402 - val_loss: 0.3796
Epoch 21/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8353 - loss: 0.3800 - val_accuracy
: 0.8398 - val_loss: 0.3751
Epoch 22/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8256 - loss: 0.3820 - val_accuracy
: 0.8387 - val_loss: 0.3781
Epoch 23/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8349 - loss: 0.3814 - val_accuracy
: 0.8417 - val_loss: 0.3771
Epoch 24/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8286 - loss: 0.3786 - val_accuracy
: 0.8440 - val_loss: 0.3783
Epoch 25/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8271 - loss: 0.3853 - val_accuracy
: 0.8440 - val_loss: 0.3772
Epoch 26/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8361 - loss: 0.3718 - val_accuracy
: 0.8444 - val_loss: 0.3782
Epoch 27/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8232 - loss: 0.3826 - val_accuracy
: 0.8478 - val_loss: 0.3784
Epoch 28/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8343 - loss: 0.3758 - val_accuracy
: 0.8440 - val_loss: 0.3764
Epoch 29/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8369 - loss: 0.3701 - val_accuracy
: 0.8497 - val_loss: 0.3733
Epoch 30/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8270 - loss: 0.3933 - val_accuracy
: 0.8478 - val_loss: 0.3764
```

```
Epoch 31/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8365 - loss: 0.3633 - val_accuracy
: 0.8482 - val_loss: 0.3751
Epoch 32/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8334 - loss: 0.3753 - val_accuracy
: 0.8470 - val_loss: 0.3754
Epoch 33/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8409 - loss: 0.3742 - val_accuracy
: 0.8519 - val_loss: 0.3725
Epoch 34/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8341 - loss: 0.3891 - val_accuracy
: 0.8527 - val_loss: 0.3734
Epoch 35/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8333 - loss: 0.3713 - val_accuracy
: 0.8523 - val_loss: 0.3714
Epoch 36/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8249 - loss: 0.3795 - val_accuracy
: 0.8531 - val_loss: 0.3721
Epoch 37/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8372 - loss: 0.3704 - val_accuracy
: 0.8489 - val_loss: 0.3738
Epoch 38/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8333 - loss: 0.3807 - val_accuracy
: 0.8504 - val_loss: 0.3721
Epoch 39/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8403 - loss: 0.3656 - val_accuracy
: 0.8527 - val_loss: 0.3678
Epoch 40/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8273 - loss: 0.3843 - val_accuracy
: 0.8508 - val_loss: 0.3710
Epoch 41/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8307 - loss: 0.3859 - val_accuracy
: 0.8546 - val_loss: 0.3670
Epoch 42/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8440 - loss: 0.3718 - val_accuracy
: 0.8516 - val_loss: 0.3692
Epoch 43/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8345 - loss: 0.3826 - val_accuracy
: 0.8523 - val_loss: 0.3691
Epoch 44/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8405 - loss: 0.3623 - val_accuracy
: 0.8531 - val_loss: 0.3695
Epoch 45/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8381 - loss: 0.3643 - val_accuracy
: 0.8542 - val_loss: 0.3699
Epoch 46/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8269 - loss: 0.3860 - val_accuracy
: 0.8504 - val_loss: 0.3714
Epoch 47/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8435 - loss: 0.3690 - val_accuracy
: 0.8519 - val_loss: 0.3702
Epoch 48/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8327 - loss: 0.3730 - val_accuracy
: 0.8516 - val_loss: 0.3695
Epoch 49/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8215 - loss: 0.3780 - val_accuracy
: 0.8519 - val_loss: 0.3715
Epoch 50/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8309 - loss: 0.3885 - val_accuracy
: 0.8508 - val_loss: 0.3717
Epoch 51/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8399 - loss: 0.3704 - val_accuracy
: 0.8527 - val_loss: 0.3730
Epoch 52/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8372 - loss: 0.3674 - val_accuracy
: 0.8538 - val_loss: 0.3700
Epoch 53/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8290 - loss: 0.3822 - val_accuracy
: 0.8542 - val_loss: 0.3694
Epoch 54/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8244 - loss: 0.3836 - val_accuracy
: 0.8523 - val_loss: 0.3728
```

```
Epoch 55/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8341 - loss: 0.3786 - val_accuracy
: 0.8519 - val_loss: 0.3699
Epoch 56/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8351 - loss: 0.3776 - val_accuracy
: 0.8531 - val_loss: 0.3671
Epoch 57/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8340 - loss: 0.3675 - val_accuracy
: 0.8523 - val_loss: 0.3682
Epoch 58/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8330 - loss: 0.3763 - val_accuracy
: 0.8493 - val_loss: 0.3741
Epoch 59/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8412 - loss: 0.3609 - val_accuracy
: 0.8523 - val_loss: 0.3695
Epoch 60/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8346 - loss: 0.3717 - val_accuracy
: 0.8501 - val_loss: 0.3693
Epoch 61/1000
536/536 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8314 - loss: 0.3888 - val_accuracy
: 0.8542 - val_loss: 0.3701
Epoch 61: early stopping
```

here we have selected 1000 epoches but we know after some time the result will be stagnent so we will use early stopping.

In [77]:

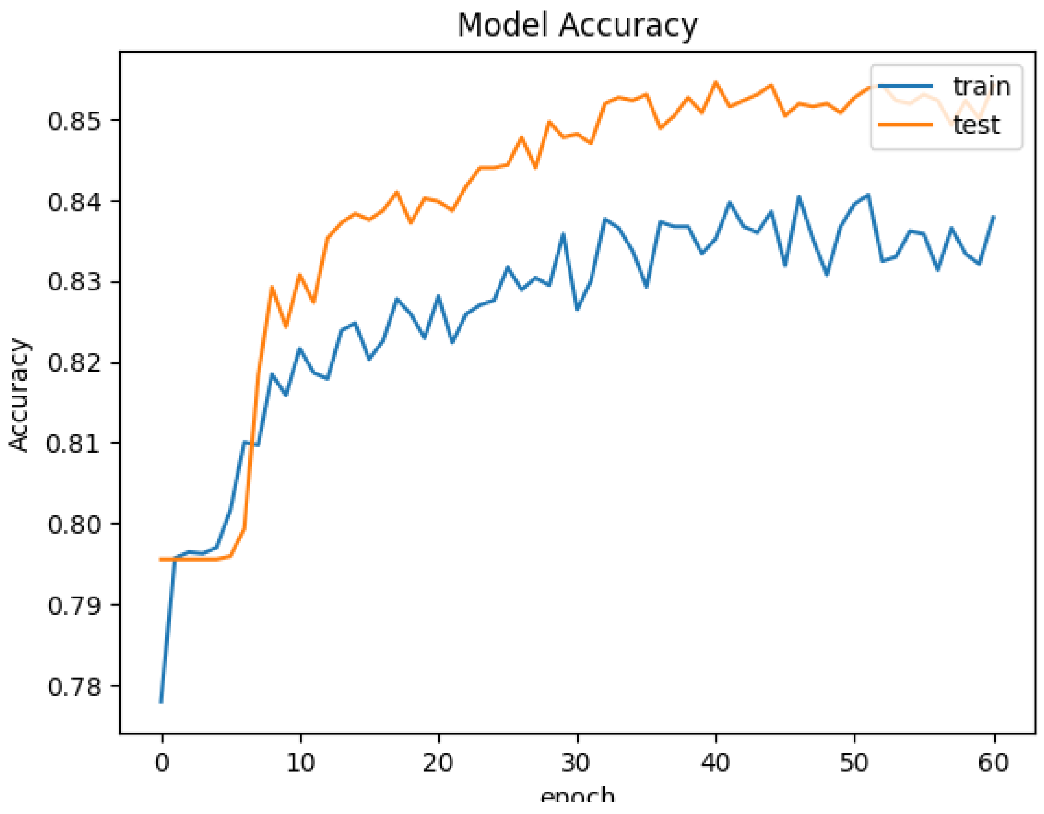
```
#To see parameters focused
model_history.history.keys()
```

Out[77]:

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

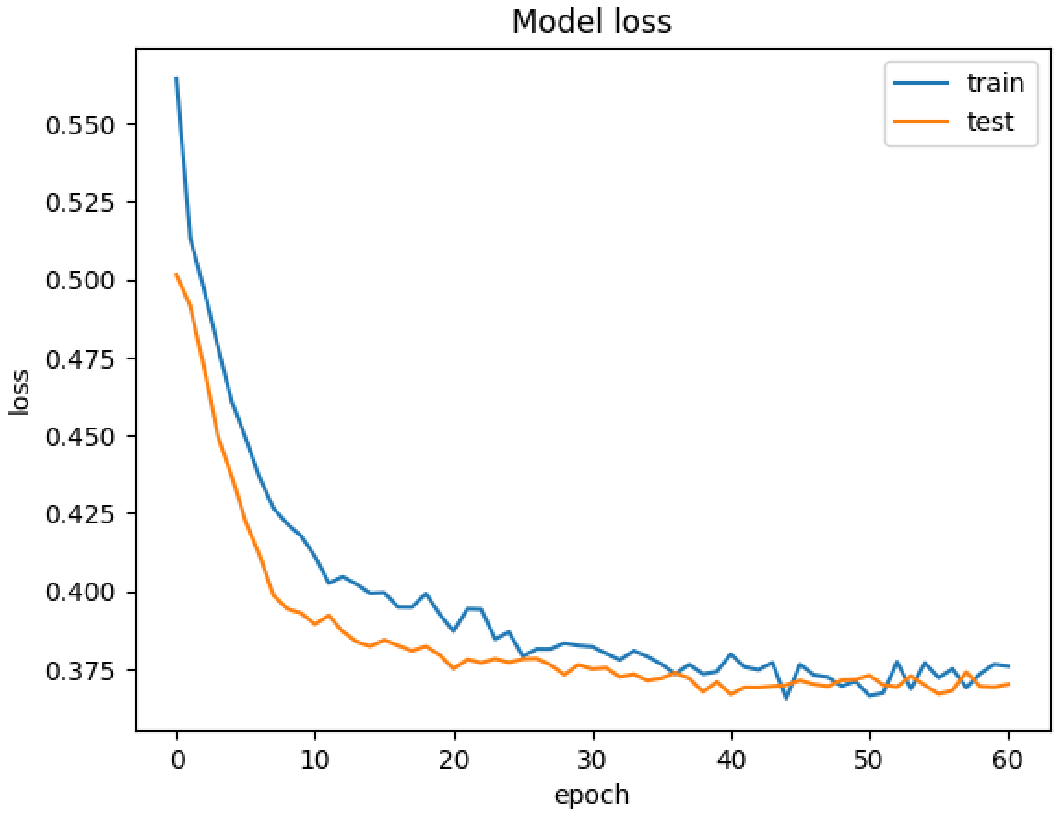
In [78]:

```
plt.plot(model_history.history['accuracy'])
plt.plot(model_history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'],loc='upper right')
plt.show()
```



In [79]:

```
plt.plot(model_history.history['loss'])
plt.plot(model_history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'],loc='upper right')
plt.show()
```



Predicting on Test Data

In [80]:

```
Y_pred=classifier.predict(X_test)
Y_pred=(Y_pred>0.5)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

In [81]:

```
#Confusion matrix
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(Y_test,Y_pred)
```

In [82]:

```
cm
```

Out[82]:

```
array([[1525,   70],
       [ 214,  191]])
```

In [83]:

```
#calculate Accuracy
from sklearn.metrics import accuracy_score
score=accuracy_score(Y_pred,Y_test)
score
```

Out[83]:

Out[83]:

```
0.858
```

In [84]:

```
#Get the Weights
classifier.get_weights()
```

Out[84]:

```
[array([[-1.55333132e-01, -1.13182038e-01,  1.09645873e-01,
          3.68294567e-02, -4.59205639e-03,  2.84451276e-01,
         -1.64702460e-02,  3.56704980e-01, -2.36808181e-01,
         -6.26784787e-02,  1.92279890e-01],
        [-9.41132903e-02,  4.11695629e-01, -7.99290359e-01,
          2.65168488e-01,  5.39045259e-02, -2.57003248e-01,
          5.48208058e-01,  1.11552160e-02, -8.02261412e-01,
         -7.07593143e-01, -2.64759541e-01],
        [ 8.28069542e-03, -1.69632345e-01,  8.92610988e-05,
         -2.12008193e-01,  9.65054855e-02,  1.46485433e-01,
          8.11525434e-02, -3.04413624e-02, -2.93456942e-01,
          1.00627184e-01,  3.56096238e-01],
        [-6.85115874e-01,  1.36739105e-01, -1.60886347e-01,
          3.65455538e-01,  8.31431225e-02, -4.35129881e-01,
         -1.79670855e-01,  7.86850378e-02, -3.30101326e-03,
         -2.53486037e-02, -7.62532353e-01],
        [-1.15213108e+00, -1.23090923e+00,  9.18601528e-02,
          2.18532190e-01,  1.06904984e+00, -1.70927960e-02,
         -1.55091539e-01,  7.59564579e-01, -2.77407289e-01,
         -1.07352823e-01, -2.83795185e-02],
        [-9.41387191e-02, -1.30830407e-01,  2.14002025e-03,
         -9.44426581e-02, -6.76972196e-02, -8.98115858e-02,
         -1.04232980e-02, -8.46888274e-02,  1.07698999e-01,
         -3.10412586e-01, -3.36021706e-02],
        [-1.05566919e-01, -6.67275488e-01, -8.38734210e-03,
         -4.98481095e-01, -1.39626056e-01,  2.29194432e-01,
          4.95790064e-01,  1.58131048e-01, -4.64813679e-01,
         -1.17063243e-02,  4.15402472e-01],
        [ 1.54507205e-01, -1.36172697e-01,  2.36498505e-01,
          2.63582826e-01,  4.85560205e-03, -4.39302921e-02,
          6.01616204e-02, -2.06673637e-01,  1.66745335e-01,
         -3.31180841e-01, -2.59210378e-01],
        [ 3.61870646e-01, -8.51783231e-02, -1.22948281e-01,
          5.77628970e-01, -1.86016902e-01,  2.67817140e-01,
          1.30132899e-01,  2.52091378e-01,  3.38820189e-01,
         -5.05491853e-01,  5.72675876e-02],
        [-8.17989558e-02, -1.35522977e-01,  8.55978206e-02,
          4.70119834e-01, -1.80681199e-02, -3.01117212e-01,
          6.66223243e-02,  1.41395882e-01, -1.60432532e-01,
         -1.86708346e-01,  2.90003810e-02],
        [ 5.00951372e-02, -3.30278315e-02,  5.09170294e-01,
          1.08670361e-01,  3.81540917e-02,  1.47623897e-01,
          1.52920619e-01,  1.11353479e-01, -6.79494217e-02,
          1.48137668e-02,  3.70657444e-01]], dtype=float32),
 array([-0.02282142,  0.11240263,  0.47672147, -0.22734135, -0.5850591 ,
         0.19307058, -0.8445043 , -0.10045127,  0.17979136,  0.4348892 ,
         0.13509151], dtype=float32),
 array([[ 0.5142357 , -0.5155043 ,  0.40836465, -0.02595253,  0.6907804 ,
          0.4387817 , -1.0975631 ],
        [ 0.24541238, -1.213532  ,  0.29134724,  0.3116756 ,  0.18285528,
          0.15819004, -0.20384912],
        [-0.43569124, -0.08237346, -0.19752286, -1.0271236 , -0.21727987,
         -0.2676733 ,  0.3031266 ],
        [ 0.26860967,  0.07172702,  0.0461184 ,  0.64101607, -0.05141769,
          0.2841023 , -0.13891868],
        [ 1.0068538 , -0.0805635 ,  0.8361861 , -0.03748529,  0.6795637 ,
         -0.16870381, -0.19995092],
        [ 0.23562501,  0.12148173, -0.48048544,  0.40249056, -0.23368916,
          0.13472535,  0.17837185],
        [-0.5591377 ,  0.59223545, -0.73406124, -0.60562164, -0.3960131 ,
         -0.30911678,  0.25539935],
        [-0.0655288 , -0.15045689,  0.17689998, -0.49821803,  0.23399249,
```

```
        -0.15076078, -0.33439264],
       [-0.40044814,  0.00558808, -0.36041924, -0.8070134 , -0.24057484,
        -0.13118242,  0.06878641],
       [-0.41644388,  0.20851254, -0.77845913, -0.3645166 , -0.35285467,
        -0.36922473, -0.41950825],
       [-0.25607437,  0.40479556, -0.43116963,  0.456178  , -0.10989977,
         0.0604284 ,  0.35848925]], dtype=float32),
 array([-0.1732138 , -0.24066222, -0.24538332, -0.16867913, -0.11375513,
        0.04599381,  0.11821616], dtype=float32),
 array([[-0.09297881, -0.12581486, -0.34289828,  0.55067176, -0.4620134 ,
        -0.7144243 ,  0.4392227 ],
       [-0.13977523, -0.05406994,  0.5357725 , -0.57188284,  0.33422613,
         0.2751652 , -0.12553619],
       [ 1.1072807 , -0.12208468, -1.1349947 ,  0.61522985, -0.9882458 ,
        -0.86723316,  0.8196368 ],
       [ 0.00973543, -0.32574385, -0.6389296 ,  0.5343556 , -0.8630342 ,
        -0.7882382 ,  0.5705699 ],
       [-0.11765885, -0.46853185, -0.68878263,  0.27744132, -0.6524759 ,
        -0.837222  ,  0.42016697],
       [ 0.11520908, -0.1475327 , -0.191323  ,  0.2291275 , -0.3757611 ,
        -0.97258365, -0.04419689],
       [-0.04895845, -0.23952687,  0.53331894, -0.7647282 ,  0.46729118,
         0.41053388, -0.14141223]], dtype=float32),
 array([-0.2215412 ,  0.        ,  0.18331961, -0.17452645,  0.16122827,
        0.11507549, -0.19814849], dtype=float32),
 array([[-1.1768224 , -0.99065953, -1.298079  , -0.8432118 , -0.83915865,
         0.81680137],
       [ 0.62090886,  0.37826526, -0.25090316,  0.26731157,  0.09376615,
         0.05791253],
       [ 0.7616234 ,  0.37092295,  0.49280578,  0.56214976,  0.47831783,
        -0.6079022 ],
       [-0.40662476, -0.3153818 , -0.54994273, -0.6363537 , -0.41936463,
         0.3162069 ],
       [ 0.60766995,  0.79542303,  0.4622157 ,  0.7289608 ,  0.5245403 ,
        -0.9909515 ],
       [ 1.1408001 ,  0.27328923,  1.0562503 ,  0.8307673 ,  0.5865433 ,
        -0.6426453 ],
       [-0.5300048 , -0.42317095, -0.92099786, -0.7856269 , -0.85758036,
         0.38651958]], dtype=float32),
 array([0.41802198, 0.25571403, 0.35050535, 0.2569159 , 0.18982328,
       0.04551965], dtype=float32),
 array([[-0.47592914],
       [-0.82827747],
       [-0.5390784 ],
       [-0.6897388 ],
       [-0.83817357],
       [ 0.66252464]], dtype=float32),
 array([-0.3235929], dtype=float32)]
```