1. Read the dataset
2. Drop the columns which are unique for all users like IDs (2.5 points)
3. Distinguish the feature and target set (2.5 points)
4. Divide the data set into train and test sets
5. Normalize the train and test data (2.5 points)
6. Initialize & build the model (10 points)
7. Optimize the model (5 points)
8. Predict the results using 0.5 as a threshold (5 points)
9. Print the Accuracy score and confusion matrix (2.5 points)

```
from google.colab import drive
drive.mount('/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=9473189

Enter your authorization code:
··········
Mounted at /gdrive
```

Description

Given a dataset consisiting of Bank Customer information, we are asked to build a classifier which wi
not.

```
%matplotlib inline
import math, random, warnings
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from IPython.core.interactiveshell import InteractiveShell
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the %tenso

```
# Configure for any default setting of any library
InteractiveShell.ast_node_interactivity = "all"
warnings.filterwarnings('ignore')
```

```
data_churn = pd.read_csv("/gdrive/My Drive/greatlakes/Projects/NeuralNetwork/Churn.csv")
```

```
data_churn.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balanc
0	1	15634602	Hargrave	619	France	Female	42	2	0.0
1	2	15647311	Hill	608	Spain	Female	41	1	83807.8
2	3	15619304	Onio	502	France	Female	42	8	159660.8
3	4	15701354	Boni	699	France	Female	39	1	0.0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.8

Performing EDA

Univariate analysis - data types and description of the independent attributes which should include (n central values (mean and median), standard deviation and quartiles, analysis of the body of distributic

Bivariate analysis between the predictor variables and between the predictor variables and target colu of their relationship and degree of relation if any. Presence of leverage points. Visualize the analysis u or density curves. Select the most appropriate attributes

Strategies to address the different data challenges such as data pollution, outliers and missing values

Inspect the Dataset

The dataset is divided into two parts, namely, **feature matrix** and the **response vector** .

Feature matrix contains all the vectors(rows) of dataset in which each vector consists of the value of features are 'RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography','Gender', 'Age', 'Tenure', 'E 'HasCrCard','IsActiveMember', 'EstimatedSalary'.

Response vector contains the value of class variable(prediction or output) for each row of feature ma name is 'Exited'.

```
data_churn.shape
```

```
(10000, 14)
```

```
data_churn.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
RowNumber          10000 non-null int64
CustomerId         10000 non-null int64
Surname            10000 non-null object
CreditScore        10000 non-null int64
Geography          10000 non-null object
Gender             10000 non-null object
Age                10000 non-null int64
Tenure             10000 non-null int64
Balance            10000 non-null float64
NumOfProducts      10000 non-null int64
HasCrCard          10000 non-null int64
IsActiveMember     10000 non-null int64
EstimatedSalary    10000 non-null float64
Exited             10000 non-null int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

There are **10000 rows** in the dataset and **14 columns**.

There are **No null/missing values** present in the dataset.

```
data_churn.columns
```

```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

We have to consider which features play a role in someone exiting a bank and we will be removing irre

```
data_churn.sample(4)
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Bal
7237	7238	15753550	Levien	684	France	Female	43	7	
7097	7098	15664793	Scott	754	Spain	Female	50	7	14677
618	619	15594594	Loggia	546	Spain	Male	42	7	13907
8762	8763	15765173	Lin	350	France	Female	60	3	

We can see that 'RowNumber', 'CustomerId', 'Surname' doesnot play any role in someone churning , s

```
#data_churn[~data_churn.applymap(np.isreal).all(1)]
```

2. Drop the columns which are unique for all users like IDs

```
data_churn.drop(columns=['RowNumber', 'CustomerId', 'Surname'],axis=1,inplace=True)
```

```
data_churn.sample(4)
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard
441	790	France	Female	31	9	0.00	2	1
7338	708	Germany	Female	54	8	145151.40	1	0
3320	712	France	Male	24	2	0.00	1	0
9069	619	Spain	Female	32	4	175406.13	2	1

```
data_churn.shape
```

```
(10000, 11)
```

We have removed 'RowNumber', 'CustomerId', 'Surname' , now our dataset contains 11 columns

```
data_churn.describe().T
```

	count	mean	std	min	25%	50%	
CreditScore	10000.0	650.528800	96.653299	350.00	584.00	652.000	718.
Age	10000.0	38.921800	10.487806	18.00	32.00	37.000	44.
Tenure	10000.0	5.012800	2.892174	0.00	3.00	5.000	7.
Balance	10000.0	76485.889288	62397.405202	0.00	0.00	97198.540	127644.
NumOfProducts	10000.0	1.530200	0.581654	1.00	1.00	1.000	2.
HasCrCard	10000.0	0.705500	0.455840	0.00	0.00	1.000	1.
IsActiveMember	10000.0	0.515100	0.499797	0.00	0.00	1.000	1.
EstimatedSalary	10000.0	100090.239881	57510.492818	11.58	51002.11	100193.915	149388.
Exited	10000.0	0.203700	0.402769	0.00	0.00	0.000	0.

Comments

This ".describe()" function generates descriptive statistics that summarizes the central tendency, disp
distribution, excluding NaN values.

25% is also known as First Quartile (Q1), 50% as Second Quartile or Median (Q2) and 75% as Third Qu

Observations

The dataset contains data about customers who are of age 18 yrs (minimum) and 92(maximum) . Me

Some of the customers have **0** account balance, Infact **25%** of people have **0 account balance**.

```
# Compare class wise mean
pd.pivot_table(data_churn,index='Exited',aggfunc=['mean'])
```

	mean					
	Age	Balance	CreditScore	EstimatedSalary	HasCrCard	IsActiveMember
Exited						
0	37.408389	72745.296779	651.853196	99738.391772	0.707146	0.554565
1	44.837997	91108.539337	645.351497	101465.677531	0.699067	0.360825

Age,Balance,EstimatedSalary mean of customer churning is more than customer not churning

```
# Compare class wise count
data_churn['Exited'].value_counts()
```

```
0    7963
1    2037
Name: Exited, dtype: int64
```

```
# List the numerical and categorical columns
numeric_cols = data_churn.select_dtypes(include=[np.number]).columns.tolist()
categ_cols = data_churn.select_dtypes(include=[np.object]).columns.tolist()
print('The numeric attributes are:', numeric_cols)
print('The categorical attributes are:', categ_cols)
```

```
The numeric attributes are: ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts',
The categorical attributes are: ['Geography', 'Gender']
```

```
data_churn[categ_cols].nunique()
```

```
Geography    3
Gender       2
dtype: int64
```

```
data_churn['Geography'].value_counts()
```

```
France     5014
Germany    2509
Spain      2477
Name: Geography, dtype: int64
```

```
pd.pivot_table(data_churn[['Age','Balance','Gender','Tenure','Geography','Exited']],index='Ex
```

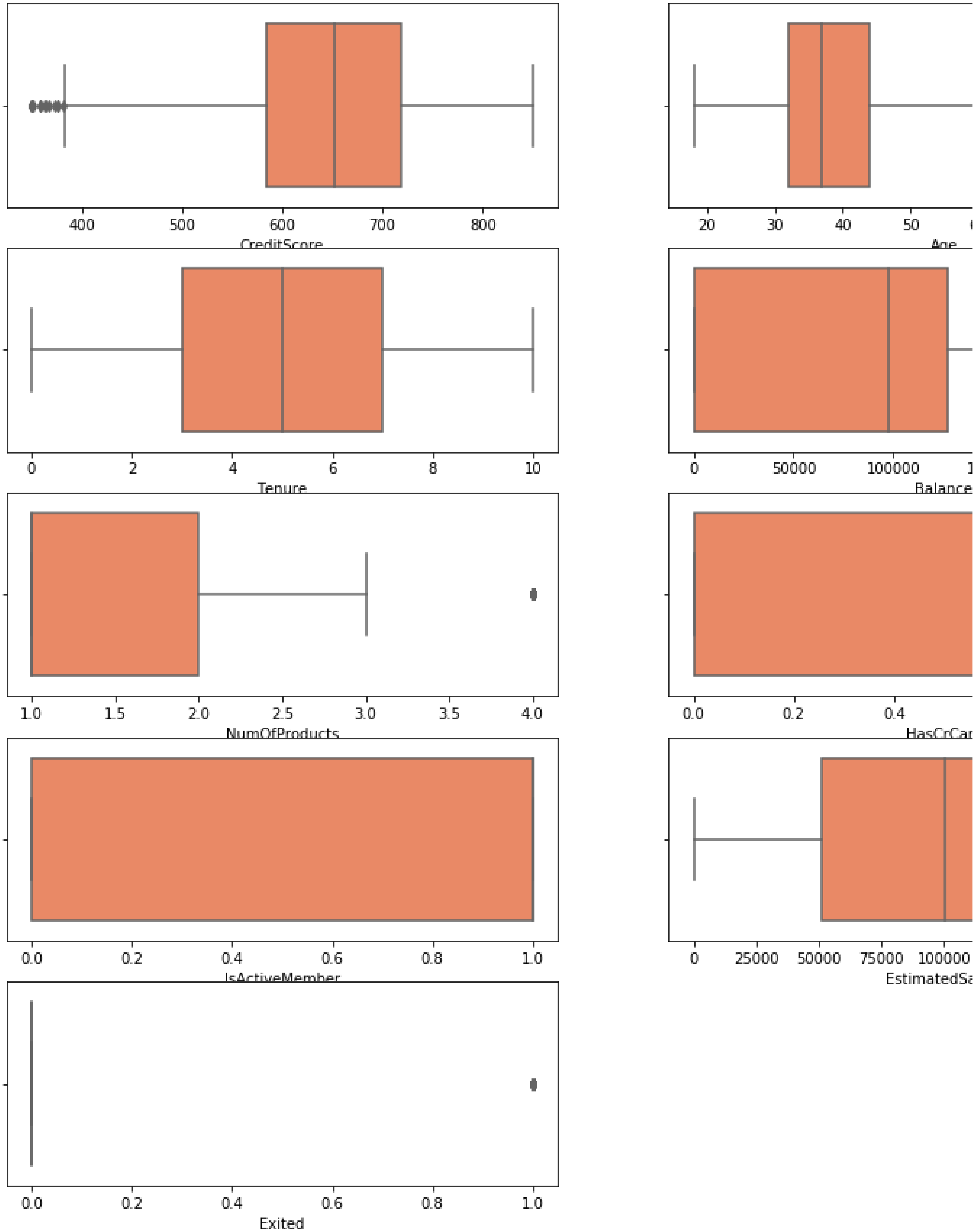
	count									
	Age			Balance			Gender			Tenur
Geography	France	Germany	Spain	France	Germany	Spain	France	Germany	Spain	Franc
Exited										
0	4204	1695	2064	4204	1695	2064	4204	1695	2064	420
1	810	814	413	810	814	413	810	814	413	81

```
len(numeric_cols)
```

```
9
```

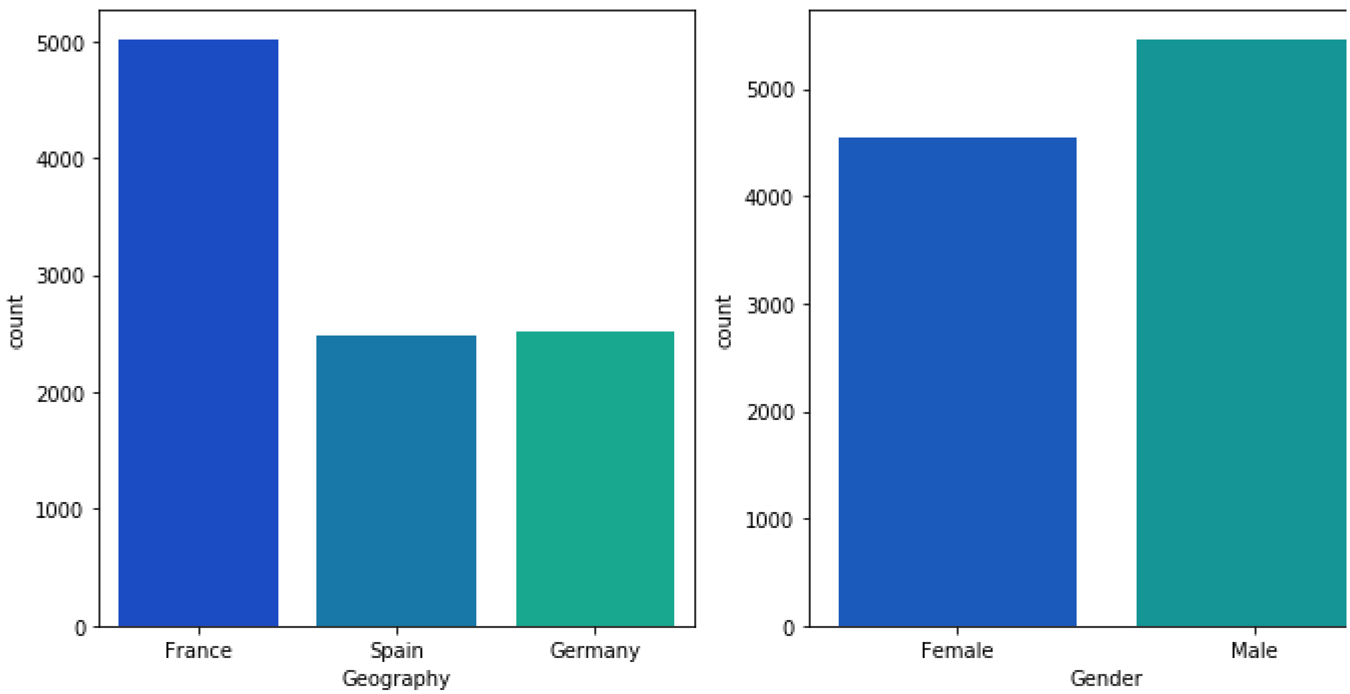
```
# Check the distribution Central Tendency
plt.figure(figsize=(15,15))
index = 1
for col in numeric_cols:
    plt.subplot(round(len(numeric_cols) / 2) +1, 2, index)
    sns.boxplot(data_churn[col], color='coral')
    index += 1
```

```
<Figure size 1080x1080 with 0 Axes><matplotlib.axes._subplots.AxesSubplot at 0x7f57c2f93
```

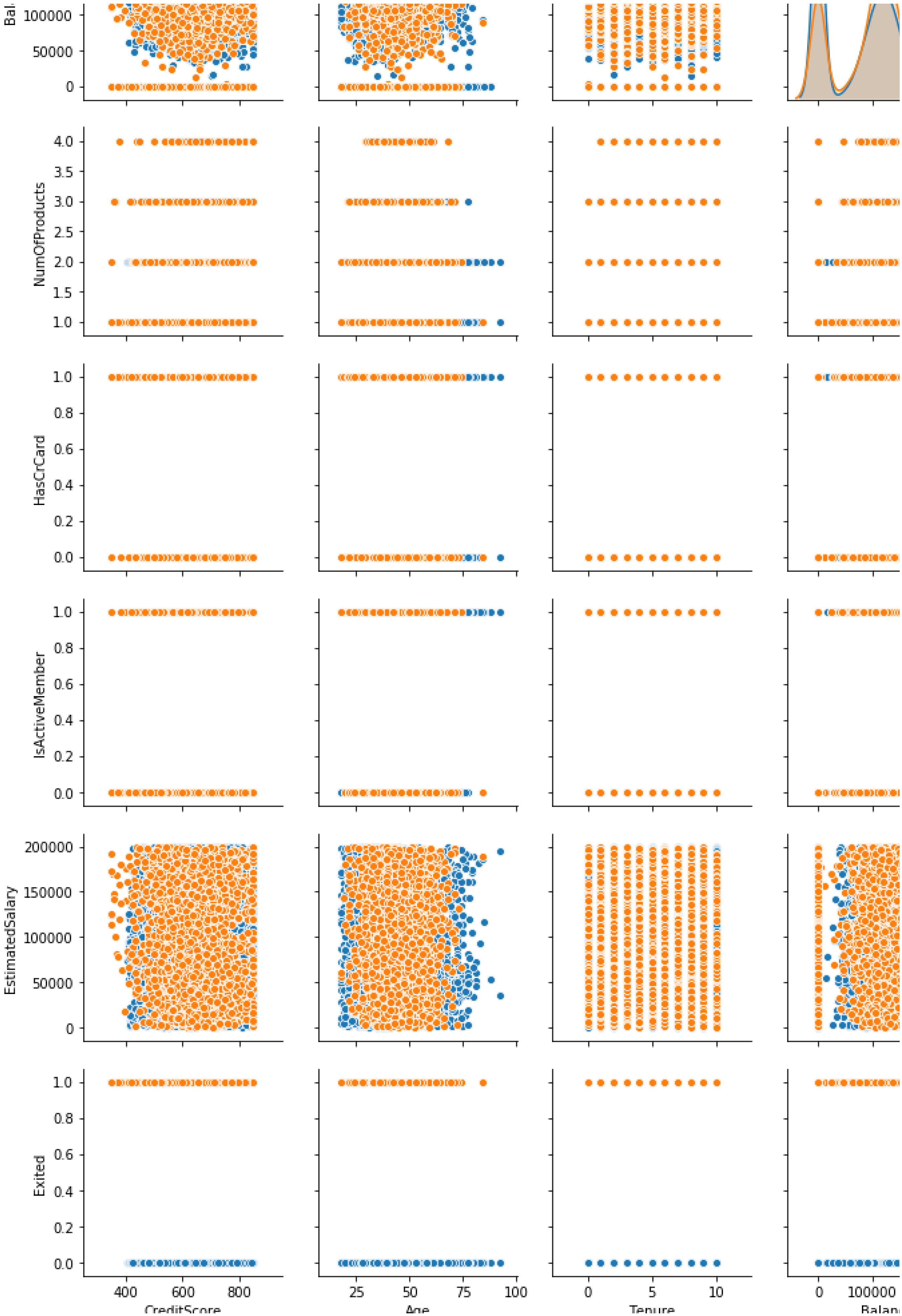


```
# Check the frequency inside each categorical features
plt.figure(figsize=(10,5))
index = 1
for col in categ_cols:
    plt.subplot(round(len(categ_cols) / 2), 2, index)
    sns.countplot(col, data=data_churn, palette='winter')
    if col == 'job':
        plt.xticks(rotation='vertical')
    index += 1
plt.tight_layout()
```

```
<Figure size 720x360 with 0 Axes><matplotlib.axes._subplots.AxesSubplot at 0x7f57c27cde4
```




```
# Pairwise relationship of numerical features in each of the category of Target
sns.pairplot(data_churn, hue='Exited', diag_kind='kde')
```


Bal
NumOfProducts
HasCrCard
IsActiveMember
EstimatedSalary
Exited
CreditScore
Age
Tenure
Balan

```
data_churn = pd.get_dummies(data_churn, columns=['Geography'])
```

```
gender_encoder = LabelEncoder()
data_churn['Gender'] = gender_encoder.fit_transform(data_churn['Gender'])
```

```
list(gender_encoder.classes_)
```

```
['Female', 'Male']
```

```
data_churn.sample(5)
```

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMem
9266	683	0	38	5	127616.56	1	1	
3068	653	0	31	7	102575.04	1	1	
3731	554	0	51	7	105701.91	1	0	
9507	808	1	41	0	0.00	1	1	

```
data_churn.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 13 columns):
CreditScore          10000 non-null int64
Gender               10000 non-null int64
Age                  10000 non-null int64
Tenure               10000 non-null int64
Balance              10000 non-null float64
NumOfProducts        10000 non-null int64
HasCrCard            10000 non-null int64
IsActiveMember       10000 non-null int64
EstimatedSalary      10000 non-null float64
Exited               10000 non-null int64
Geography_France     10000 non-null uint8
Geography_Germany    10000 non-null uint8
Geography_Spain      10000 non-null uint8
dtypes: float64(2), int64(8), uint8(3)
memory usage: 810.7 KB
```

```
X = data_churn.drop(['Exited'],axis=1)
y = data_churn['Exited']
```

Splitting the dataset into the Training and Testing set.

```
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)
```

Normalize the train and test data aka Feature scaling

Feature scaling is a method used to standardize the range of independent variables or features of dat to be even so that one independent variable does not dominate another

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
#X_train = pd.DataFrame(X_scaled_train, columns=X.columns)
#X_test = pd.DataFrame(X_scaled_test, columns=X.columns)
```

```
type(X_train)
```

```
numpy.ndarray
```

Initialize & build the model (10 points)

```
#Clear out model from current memory
tf.keras.backend.clear_session()

#Initialize Sequential model
model = tf.keras.models.Sequential()


model.add(tf.keras.layers.Dense(11, kernel_initializer='he_normal',activation = 'relu'))
model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.Dense(100, kernel_initializer='he_normal',activation = 'relu'))

model.add(tf.keras.layers.Dense(25, kernel_initializer='he_normal',activation = 'relu'))

#Add OUTPUT layer
# we have an output of 1 node, which is the the desired dimensions of our output (stay with t
# We use the sigmoid because we want probability outcomes
# If we want more than two categories, then we will need to change softmax
model.add(tf.keras.layers.Dense(1, kernel_initializer='he_normal',
                                activation='sigmoid'))
```

Compiling the Neural Network

Tuning the individual weights on each neuron

optimizer: [adam The] algorithm we want to use to find the optimal set of weights in the neural netwo

loss: [binary_crossentropy] This is the loss function used within adam. If our dependent (output variak Categorical, then it is called categorical_crossentropy

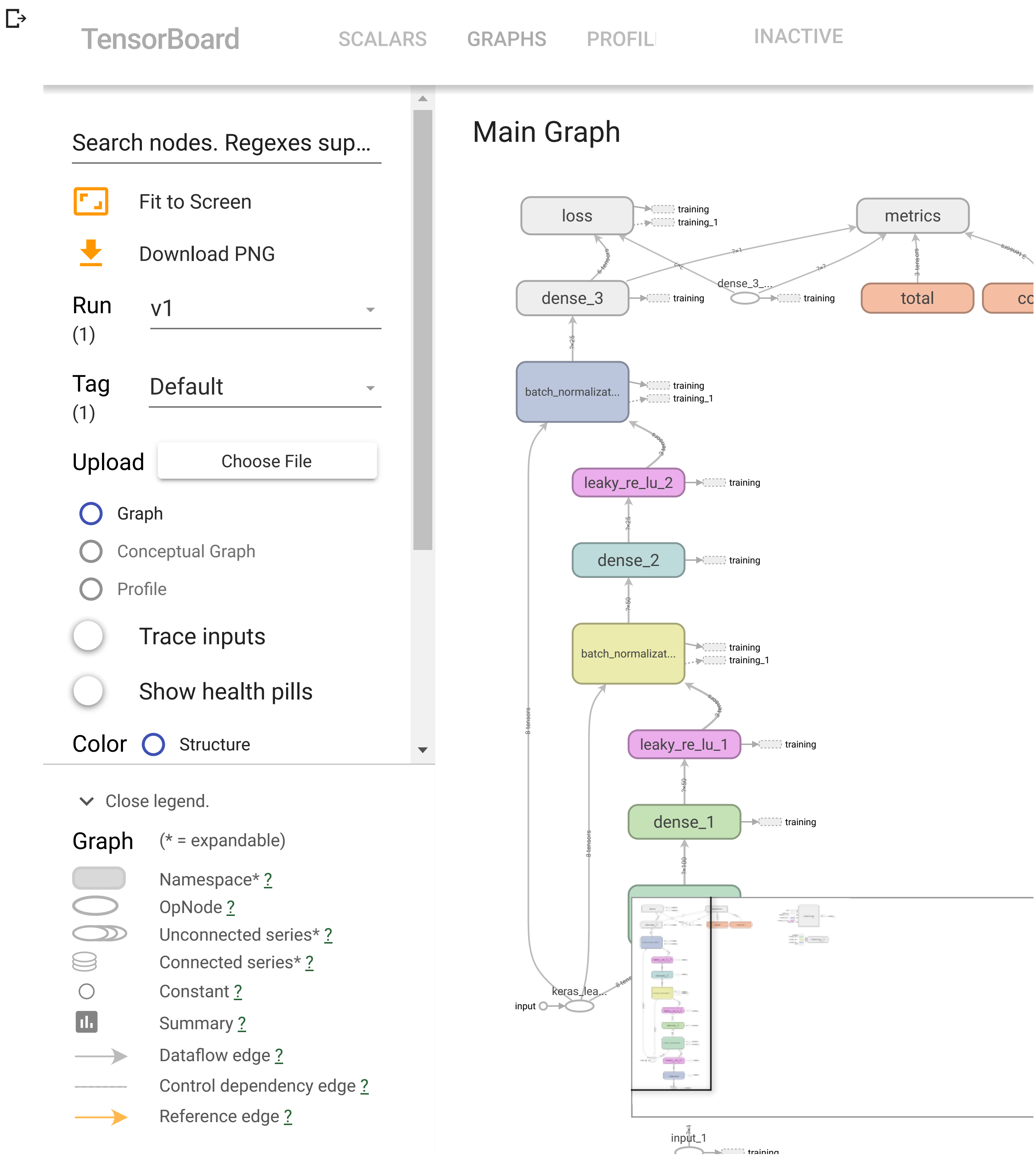
metrics: [accuracy] The accuracy metrics which will be evaluated by the model

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


#Define tensorboard callback
tboard = tf.keras.callbacks.TensorBoard(log_dir='./logs/v1')


#Load tensorboard module
%load_ext tensorboard
```

```
#Start tensorboard
%tensorboard --logdir logs
```



```
history = model.fit(X_train,y_train.to_numpy(),
          validation_data=(X_test,y_test.to_numpy()),
          epochs=20,
          batch_size=32, callbacks=[tboard])
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/op
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/op
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Train on 8000 samples, validate on 2000 samples
Epoch 1/20
8000/8000 [==============================] - 1s 135us/sample - loss: 0.4613 - acc: 0.805
Epoch 2/20
8000/8000 [==============================] - 1s 73us/sample - loss: 0.4167 - acc: 0.8259
Epoch 3/20
8000/8000 [==============================] - 1s 67us/sample - loss: 0.3914 - acc: 0.8356
Epoch 4/20
8000/8000 [==============================] - 1s 68us/sample - loss: 0.3810 - acc: 0.8422
Epoch 5/20
8000/8000 [==============================] - 1s 71us/sample - loss: 0.3686 - acc: 0.8470
Epoch 6/20
8000/8000 [==============================] - 1s 81us/sample - loss: 0.3690 - acc: 0.8465
Epoch 7/20
8000/8000 [==============================] - 1s 72us/sample - loss: 0.3621 - acc: 0.8496
Epoch 8/20
8000/8000 [==============================] - 1s 66us/sample - loss: 0.3556 - acc: 0.8558
Epoch 9/20
8000/8000 [==============================] - 1s 69us/sample - loss: 0.3536 - acc: 0.8547
Epoch 10/20
8000/8000 [==============================] - 1s 67us/sample - loss: 0.3520 - acc: 0.8528
Epoch 11/20
8000/8000 [==============================] - 1s 67us/sample - loss: 0.3505 - acc: 0.8564
Epoch 12/20
```

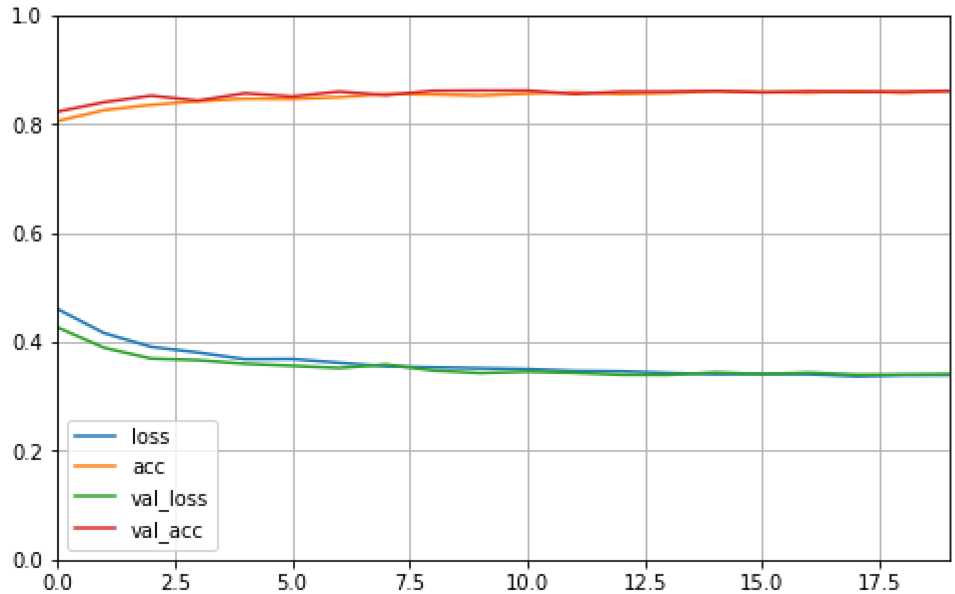
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                multiple                  143
_________________________________________________________________
batch_normalization (BatchNo multiple                  44
_________________________________________________________________
dense_1 (Dense)              multiple                  1200
_________________________________________________________________
dense_2 (Dense)              multiple                  2525
_________________________________________________________________
dense_3 (Dense)              multiple                  26
=================================================================
Total params: 3,938
Trainable params: 3,916
Non-trainable params: 22
_________________________________________________________________
```

```
import pandas as pd

pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f57bb4355f8>(0, 1)
```




```
model.evaluate(X_test, y_test)
```

```
2000/2000 [==============================] - 0s 26us/sample - loss: 0.3410 - acc: 0.8610
[0.34102151107788087, 0.861]
```

```
y_pred = model.predict(X_test)
print(y_pred)
```

```
[[0.39037624]
 [0.32012886]
 [0.15359083]
 ...
 [0.03462902]
 [0.11212406]
 [0.21432176]]
```

This provides us probabilities. We need to convert these probabilities into the form true or false. So w
whether they are likely to exit or not.

```
y_pred = (y_pred > 0.5)
y_pred
```

```
array([[False],
       [False],
       [False],
       ...,
       [False],
       [False]
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
0.861
```

Accuracy is 85 %

```
from sklearn import metrics
print(metrics.confusion_matrix(y_test, y_pred))
print(metrics.classification_report(y_test, y_pred))
```

```
[[1517   78]
 [ 200  205]]
              precision    recall  f1-score   support

           0       0.88      0.95      0.92      1595
           1       0.72      0.51      0.60       405

    accuracy                           0.86      2000
   macro avg       0.80      0.73      0.76      2000
weighted avg       0.85      0.86      0.85      2000
```

```
!ls -l
```

```
total 8
drwxr-xr-x 3 root root 4096 Dec  8 12:29 logs
drwxr-xr-x 1 root root 4096 Nov 27 22:38 sample_data
```

```
model.save('bank_churn_v1.h5')
```

```
!ls -l
```

```
total 108
-rw-r--r-- 1 root root 101336 Dec  8 12:29 bank_churn_v1.h5
drwxr-xr-x 3 root root   4096 Dec  8 12:29 logs
drwxr-xr-x 1 root root   4096 Nov 27 22:38 sample_data
```

Model 2 - Optimising

```
#Clear out model from current memory
tf.keras.backend.clear_session()

#Initialize Sequential model
```

```
model2 = tf.keras.models.Sequential()


model2.add(tf.keras.layers.Input(12))

model2.add(tf.keras.layers.Dense(100, kernel_initializer='he_normal'))
model2.add(tf.keras.layers.LeakyReLU())
model2.add(tf.keras.layers.BatchNormalization())

model2.add(tf.keras.layers.Dense(50, kernel_initializer='he_normal'))
model2.add(tf.keras.layers.LeakyReLU())
model2.add(tf.keras.layers.BatchNormalization())

model2.add(tf.keras.layers.Dense(25, kernel_initializer='he_normal'))
model2.add(tf.keras.layers.LeakyReLU())
model2.add(tf.keras.layers.BatchNormalization())
#Add OUTPUT layer
# we have an output of 1 node, which is the the desired dimensions of our output (stay with t
# We use the sigmoid because we want probability outcomes
# If we want more than two categories, then we will need to change softmax
model2.add(tf.keras.layers.Dense(1, kernel_initializer='he_normal',
                                activation='sigmoid'))

#Compile the model
model2.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['accuracy'])

#Modelcheckpoint callback
ckpt = tf.keras.callbacks.ModelCheckpoint('mnist_v2.hdf5', save_best_only=True,
                                          monitor='val_loss', mode='min')
tboard2= tf.keras.callbacks.TensorBoard(log_dir='./logs/v2')
history2 = model2.fit(X_train,y_train.to_numpy(),
          validation_data=(X_test,y_test.to_numpy()),
          epochs=20,
          batch_size=32, callbacks=[tboard,ckpt])
```

```
Train on 8000 samples, validate on 2000 samples
Epoch 1/20
8000/8000 [==============================] - 1s 161us/sample - loss: 0.5012 - acc: 0.778
Epoch 2/20
8000/8000 [==============================] - 1s 87us/sample - loss: 0.3810 - acc: 0.8403
Epoch 3/20
8000/8000 [==============================] - 1s 91us/sample - loss: 0.3692 - acc: 0.8449
Epoch 4/20
8000/8000 [==============================] - 1s 88us/sample - loss: 0.3653 - acc: 0.8486
Epoch 5/20
8000/8000 [==============================] - 1s 89us/sample - loss: 0.3625 - acc: 0.8455
Epoch 6/20
8000/8000 [==============================] - 1s 93us/sample - loss: 0.3586 - acc: 0.8500
Epoch 7/20
8000/8000 [==============================] - 1s 82us/sample - loss: 0.3509 - acc: 0.8506
Epoch 8/20
8000/8000 [==============================] - 1s 86us/sample - loss: 0.3558 - acc: 0.8534
Epoch 9/20
8000/8000 [==============================] - 1s 90us/sample - loss: 0.3494 - acc: 0.8568
Epoch 10/20
8000/8000 [==============================] - 1s 85us/sample - loss: 0.3469 - acc: 0.8561
Epoch 11/20
8000/8000 [==============================] - 1s 84us/sample - loss: 0.3443 - acc: 0.8606
Epoch 12/20
```

```python
from tensorboard import notebook
notebook.list() # View open TensorBoard instances
```

```
Known TensorBoard instances:
  - port 6006: logdir logs (started 0:00:36 ago; pid 329)
8000/8000 [==============================] - 1s 84us/sample - loss: 0.3447 - acc: 0.8562
```

```python
model_loaded = tf.keras.models.load_model('mnist_v2.hdf5')
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/op
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the construct
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/op
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the construct
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/op
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the construct
```

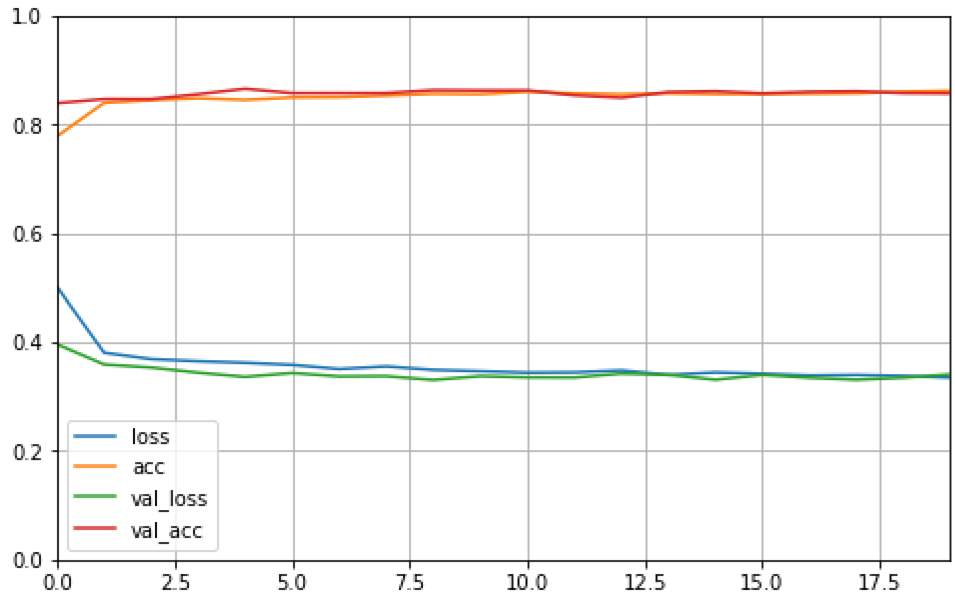
```python
model_loaded.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 100)               1300
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 100)               0
_________________________________________________________________
batch_normalization (BatchNo (None, 100)               400
_________________________________________________________________
dense_1 (Dense)              (None, 50)                5050
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 50)                0
_________________________________________________________________
batch_normalization_1 (Batch (None, 50)                200
_________________________________________________________________
dense_2 (Dense)              (None, 25)                1275
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 25)                0
_________________________________________________________________
batch_normalization_2 (Batch (None, 25)                100
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 26
=================================================================
```

```python
import pandas as pd

pd.DataFrame(history2.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f57b547c588>(0, 1)
```



```
model_loaded.evaluate(X_test, y_test)
```

```
2000/2000 [==============================] - 0s 70us/sample - loss: 0.3308 - acc: 0.8635
[0.3308159551620483, 0.8635]
```

```
from sklearn import metrics
y_pred_loaded = model_loaded.predict(X_test)
y_pred_loaded = (y_pred_loaded > 0.5)
print(y_pred_loaded)
print(metrics.confusion_matrix(y_test, y_pred_loaded))
print(metrics.classification_report(y_test, y_pred_loaded))
```

```
[[False]
 [False]
 [False]
 ...
 [False]
 [False]
 [False]]
[[1505   90]
 [ 183  222]]
              precision    recall  f1-score   support

           0       0.89      0.94      0.92      1595
           1       0.71      0.55      0.62       405

    accuracy                           0.86      2000
   macro avg       0.80      0.75      0.77      2000
weighted avg       0.86      0.86      0.86      2000
```

Over all accuracy has increased by 1 %