Credit Card Fraud Detection - Classification

Anonymized credit card transactions labeled as fraudulent or genuine

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

Context

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

Content

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Given the class imbalance ratio, we recommend measuring the accuracy using the Area Under the Precision-Recall Curve (AUPRC). Confusion matrix accuracy is not meaningful for unbalanced classification.

```
# Importing Required Lib
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
import tensorflow as tf

import warnings
warnings.filterwarnings('ignore')
```

```
# reading the data set
data_df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Deep Learning/creditcard.csv')
```

```
#showing top 5 records
data_df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.2
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.3
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.2

5 rows × 31 columns

```
# Checking shape of data set
data_df.shape
```

```
(284807, 31)
```

```
# Checking Data Types
data_df.dtypes
```

```
Time      float64
V1        float64
V2        float64
```

```
V3        float64
V4        float64
V5        float64
V6        float64
V7        float64
V8        float64
V9        float64
V10       float64
V11       float64
V12       float64
V13       float64
V14       float64
V15       float64
V16       float64
V17       float64
V18       float64
V19       float64
V20       float64
V21       float64
V22       float64
V23       float64
V24       float64
V25       float64
V26       float64
V27       float64
V28       float64
Amount    float64
Class       int64
dtype: object
```

```
# Checking Null values
data_df.isnull().sum()
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

```
# NO null values
```

```
# Duplicate Entry
data_df.duplicated().sum()
```

```
1081
```

```
# We have 1081 Duplicate Entry out of 284807 entry. Let remove it first before moving forward.

data_df.drop_duplicates(inplace = True)
```

```
data_df.duplicated().sum()
```

```
0
```

```
# Let Describe our data
data_df.describe()
```

	Time	V1	V2	V3	V4	
count	283726.000000	283726.000000	283726.000000	283726.000000	283726.000000	28
mean	94811.077600	0.005917	-0.004135	0.001613	-0.002966	
std	47481.047891	1.948026	1.646703	1.508682	1.414184	
min	0.000000	-56.407510	-72.715728	-48.325589	-5.683171	
25%	54204.750000	-0.915951	-0.600321	-0.889682	-0.850134	
50%	84692.500000	0.020384	0.063949	0.179963	-0.022248	
75%	139298.000000	1.316068	0.800283	1.026960	0.739647	
max	172792.000000	2.454930	22.057729	9.382558	16.875344	

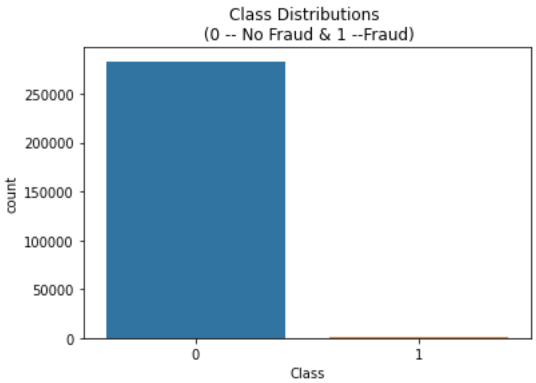
8 rows × 31 columns

```
# The classes are heavily skewed we need to solve this issue later.
print('No Frauds', round(data_df['Class'].value_counts()[0]/len(data_df) * 100,2), '% of the dataset')
print('Frauds', round(data_df['Class'].value_counts()[1]/len(data_df) * 100,2), '% of the dataset')
```

```
No Frauds 99.83 % of the dataset
Frauds 0.17 % of the dataset
```

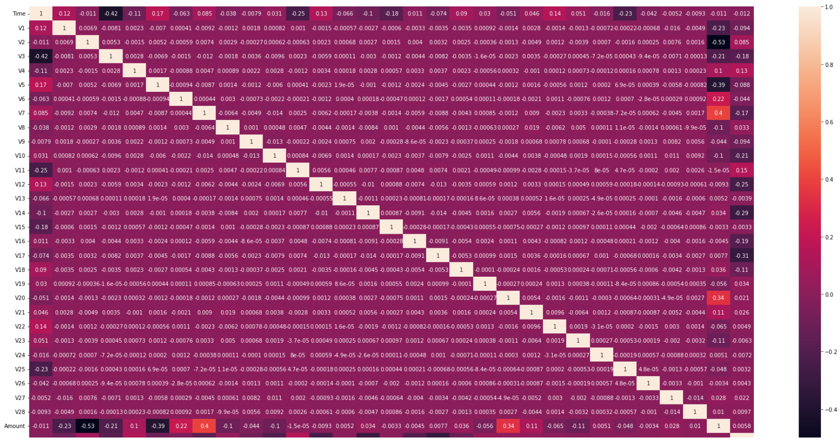
Note: Most of the transactions are non-fraud. If we use this dataframe as the base for our predictive models and analysis we might get a lot of errors and our algorithms will probably overfit since it will "assume" that most transactions are not fraud. But we don't want our model to assume, we want our model to detect patterns that give signs of fraud!

```
sb.countplot(data = data_df, x = 'Class')
plt.title('Class Distributions \n (0 -- No Fraud & 1 --Fraud)')
plt.show()
```




```
plt.figure(figsize = (30,15))
sb.heatmap(data_df.corr(), annot = True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fade9489df0>
```



```
X = data_df[['Amount', 'Time']]
Y = data_df['Class']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train,X_test, Y_train,Y_test = train_test_split(X,Y, test_size = 0.3, random_state = 1)
```

```
Y_train.value_counts()
```

```
0    198260
1       348
Name: Class, dtype: int64
```

```
# Converts minority class to majority class

from imblearn.over_sampling import RandomOverSampler
```

```
ros = RandomOverSampler(random_state= 1)
```

```
X_train1, Y_train1 = ros.fit_resample(X_train, Y_train)
```

```
X_test1, Y_test1 = ros.fit_resample(X_test,Y_test)
```

```
Y_train1.value_counts()
```

```
0    198260
1    198260
Name: Class, dtype: int64
```

```
Y_test1.value_counts()
```

```
0    84993
1    84993
Name: Class, dtype: int64
```

```
from sklearn.preprocessing import StandardScaler
```

```
ss = StandardScaler()
X_train1 = ss.fit_transform(X_train1)
X_test1 = ss.transform(X_test1)
```

```
# Creat model
model = tf.keras.Sequential([
    # First Layer
    tf.keras.layers.Dense(units = 64, activation = 'relu', input_shape = (X.shape[1],)),
    # Second Layer
    tf.keras.layers.Dense(units = 32, activation = 'relu'),
    # Third Layer
    tf.keras.layers.Dense(units = 16, activation = 'relu'),
    # Fourth Layer
    tf.keras.layers.Dense(units = 8, activation = 'relu'),
```

```
    # output
    tf.keras.layers.Dense(units = 1, activation = 'sigmoid'),
                                ])
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                192

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 16)                528

 dense_3 (Dense)             (None, 8)                 136

 dense_4 (Dense)             (None, 1)                 9

=================================================================
Total params: 2,945
Trainable params: 2,945
Non-trainable params: 0
_________________________________________________________________
```

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

```
# create a procedure for EARLY STOPING
# calling inbuilt class:   EarlyStopping

from tensorflow.keras.callbacks import EarlyStopping

#create object
cb = EarlyStopping(monitor = 'val_loss',   # Mention parameter to monitor .. it may me loss or score
                   min_delta = 0.0001,     # diff btw new and previous loss: bydefault we take 0.0001
                   patience = 20,
                   verbose =1,
                   mode = 'auto',
                   baseline = None,
                   restore_best_weights =False)
```

```
# train the model: use inbuilt method fit() of Sequential class

train_model = model.fit(X_train1, Y_train1,epochs =5000,validation_data=(X_test1,Y_test1), callbacks = cb)
```

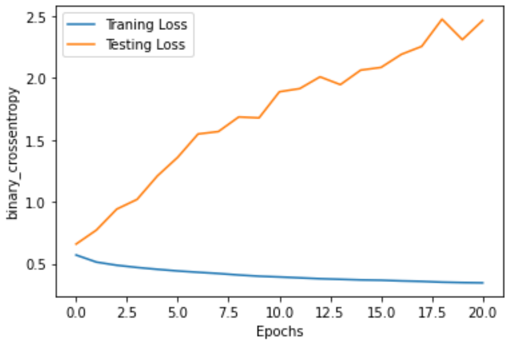
```
Epoch 1/5000
12392/12392 [==============================] - 36s 3ms/step - loss: 0.5708 - accuracy: 0.7071 - val_loss: 0.6592 - val_accuracy: 0.
Epoch 2/5000
12392/12392 [==============================] - 39s 3ms/step - loss: 0.5137 - accuracy: 0.7463 - val_loss: 0.7721 - val_accuracy: 0.
Epoch 3/5000
12392/12392 [==============================] - 40s 3ms/step - loss: 0.4882 - accuracy: 0.7600 - val_loss: 0.9411 - val_accuracy: 0.
Epoch 4/5000
12392/12392 [==============================] - 35s 3ms/step - loss: 0.4704 - accuracy: 0.7688 - val_loss: 1.0196 - val_accuracy: 0.
Epoch 5/5000
12392/12392 [==============================] - 36s 3ms/step - loss: 0.4553 - accuracy: 0.7771 - val_loss: 1.2092 - val_accuracy: 0.
Epoch 6/5000
12392/12392 [==============================] - 40s 3ms/step - loss: 0.4421 - accuracy: 0.7840 - val_loss: 1.3596 - val_accuracy: 0.
Epoch 7/5000
12392/12392 [==============================] - 39s 3ms/step - loss: 0.4315 - accuracy: 0.7899 - val_loss: 1.5474 - val_accuracy: 0.
Epoch 8/5000
12392/12392 [==============================] - 39s 3ms/step - loss: 0.4215 - accuracy: 0.7956 - val_loss: 1.5670 - val_accuracy: 0.
Epoch 9/5000
12392/12392 [==============================] - 40s 3ms/step - loss: 0.4095 - accuracy: 0.8023 - val_loss: 1.6848 - val_accuracy: 0.
Epoch 10/5000
12392/12392 [==============================] - 40s 3ms/step - loss: 0.3999 - accuracy: 0.8089 - val_loss: 1.6774 - val_accuracy: 0.
Epoch 11/5000
12392/12392 [==============================] - 38s 3ms/step - loss: 0.3936 - accuracy: 0.8123 - val_loss: 1.8880 - val_accuracy: 0.
Epoch 12/5000
12392/12392 [==============================] - 36s 3ms/step - loss: 0.3866 - accuracy: 0.8175 - val_loss: 1.9136 - val_accuracy: 0.
Epoch 13/5000
12392/12392 [==============================] - 39s 3ms/step - loss: 0.3794 - accuracy: 0.8217 - val_loss: 2.0086 - val_accuracy: 0.
Epoch 14/5000
12392/12392 [==============================] - 36s 3ms/step - loss: 0.3748 - accuracy: 0.8249 - val_loss: 1.9457 - val_accuracy: 0.
Epoch 15/5000
12392/12392 [==============================] - 40s 3ms/step - loss: 0.3698 - accuracy: 0.8277 - val_loss: 2.0627 - val_accuracy: 0.
Epoch 16/5000
12392/12392 [==============================] - 39s 3ms/step - loss: 0.3675 - accuracy: 0.8295 - val_loss: 2.0846 - val_accuracy: 0.
Epoch 17/5000
12392/12392 [==============================] - 34s 3ms/step - loss: 0.3619 - accuracy: 0.8330 - val_loss: 2.1900 - val_accuracy: 0.
Epoch 18/5000
12392/12392 [==============================] - 38s 3ms/step - loss: 0.3578 - accuracy: 0.8352 - val_loss: 2.2543 - val_accuracy: 0.
```

```
Epoch 19/5000
12392/12392 [==============================] - 38s 3ms/step - loss: 0.3518 - accuracy: 0.8390 - val_loss: 2.4740 - val_accuracy: 0.
Epoch 20/5000
12392/12392 [==============================] - 34s 3ms/step - loss: 0.3483 - accuracy: 0.8405 - val_loss: 2.3090 - val_accuracy: 0.
Epoch 21/5000
12392/12392 [==============================] - 38s 3ms/step - loss: 0.3462 - accuracy: 0.8427 - val_loss: 2.4638 - val_accuracy: 0.
Epoch 21: early stopping
```

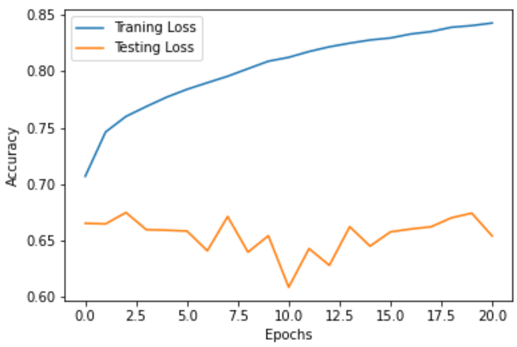
```
# Here we see that out model is Underfit ....No we will increse the no of Hidden layer
print("Traning Loss and Accuracy: ", model.evaluate(X_train1, Y_train1))
print("Testing Loss and Accuracy: ", model.evaluate(X_test1, Y_test1))
```

```
12392/12392 [==============================] - 21s 2ms/step - loss: 0.3306 - accuracy: 0.8433
Traning Loss and Accuracy:  [0.33063846826553345, 0.8433168530464172]
5313/5313 [==============================] - 9s 2ms/step - loss: 2.4638 - accuracy: 0.6542
Testing Loss and Accuracy:  [2.4637670516967773, 0.6542479991912842]
```

```
# Visulisation
plt.plot(train_model.history['loss'],label = 'Traning Loss')
plt.plot(train_model.history['val_loss'],label = 'Testing Loss')
plt.xlabel('Epochs')
plt.ylabel('binary_crossentropy')
plt.legend()
plt.show()
```




```
# Visulisation
plt.plot(train_model.history['accuracy'],label = 'Traning Loss')
plt.plot(train_model.history['val_accuracy'],label = 'Testing Loss')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



Here we see that our model is Overfit.Now we use dropout to reduce this overfiting.

```
# Create model with Dropout

from tensorflow.keras.layers import Dropout
model_1 = tf.keras.Sequential([
    # First Layer
    tf.keras.layers.Dense(units = 64, activation = 'relu', input_shape = (X.shape[1],)),Dropout(0.25),
    # Second Layer
    tf.keras.layers.Dense(units = 32, activation = 'relu'),Dropout(0.25),
    # Third Layer
    tf.keras.layers.Dense(units = 16, activation = 'relu'),Dropout(0.25),
```

```
    # Fourth Layer
    tf.keras.layers.Dense(units = 8, activation = 'relu'),Dropout(0.25),
    # output
    tf.keras.layers.Dense(units = 1, activation = 'sigmoid'),
                              ])
```

```
model_1.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

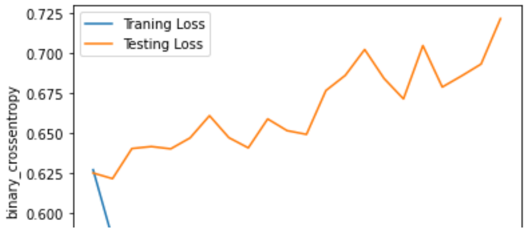
```
# train the model: use inbuilt method fit() of Sequential class

train_model = model_1.fit(X_train1, Y_train1,epochs =5000,validation_data=(X_test1,Y_test1), callbacks = cb)
```

```
Epoch 1/5000
12392/12392 [==============================] - 41s 3ms/step - loss: 0.6271 - accuracy: 0.6575 - val_loss: 0.6251 - val_accuracy: 0.
Epoch 2/5000
12392/12392 [==============================] - 39s 3ms/step - loss: 0.5839 - accuracy: 0.7064 - val_loss: 0.6215 - val_accuracy: 0.
Epoch 3/5000
12392/12392 [==============================] - 35s 3ms/step - loss: 0.5755 - accuracy: 0.7114 - val_loss: 0.6404 - val_accuracy: 0.
Epoch 4/5000
12392/12392 [==============================] - 44s 4ms/step - loss: 0.5693 - accuracy: 0.7154 - val_loss: 0.6417 - val_accuracy: 0.
Epoch 5/5000
12392/12392 [==============================] - 40s 3ms/step - loss: 0.5677 - accuracy: 0.7165 - val_loss: 0.6402 - val_accuracy: 0.
Epoch 6/5000
12392/12392 [==============================] - 41s 3ms/step - loss: 0.5650 - accuracy: 0.7181 - val_loss: 0.6471 - val_accuracy: 0.
Epoch 7/5000
12392/12392 [==============================] - 40s 3ms/step - loss: 0.5630 - accuracy: 0.7190 - val_loss: 0.6610 - val_accuracy: 0.
Epoch 8/5000
12392/12392 [==============================] - 45s 4ms/step - loss: 0.5621 - accuracy: 0.7199 - val_loss: 0.6472 - val_accuracy: 0.
Epoch 9/5000
12392/12392 [==============================] - 36s 3ms/step - loss: 0.5612 - accuracy: 0.7221 - val_loss: 0.6408 - val_accuracy: 0.
Epoch 10/5000
12392/12392 [==============================] - 43s 3ms/step - loss: 0.5606 - accuracy: 0.7222 - val_loss: 0.6590 - val_accuracy: 0.
Epoch 11/5000
12392/12392 [==============================] - 44s 4ms/step - loss: 0.5594 - accuracy: 0.7231 - val_loss: 0.6516 - val_accuracy: 0.
Epoch 12/5000
12392/12392 [==============================] - 37s 3ms/step - loss: 0.5580 - accuracy: 0.7229 - val_loss: 0.6493 - val_accuracy: 0.
Epoch 13/5000
12392/12392 [==============================] - 41s 3ms/step - loss: 0.5587 - accuracy: 0.7221 - val_loss: 0.6768 - val_accuracy: 0.
Epoch 14/5000
12392/12392 [==============================] - 37s 3ms/step - loss: 0.5591 - accuracy: 0.7210 - val_loss: 0.6863 - val_accuracy: 0.
Epoch 15/5000
12392/12392 [==============================] - 35s 3ms/step - loss: 0.5614 - accuracy: 0.7202 - val_loss: 0.7025 - val_accuracy: 0.
Epoch 16/5000
12392/12392 [==============================] - 39s 3ms/step - loss: 0.5567 - accuracy: 0.7231 - val_loss: 0.6843 - val_accuracy: 0.
Epoch 17/5000
12392/12392 [==============================] - 40s 3ms/step - loss: 0.5575 - accuracy: 0.7220 - val_loss: 0.6715 - val_accuracy: 0.
Epoch 18/5000
12392/12392 [==============================] - 36s 3ms/step - loss: 0.5570 - accuracy: 0.7223 - val_loss: 0.7049 - val_accuracy: 0.
Epoch 19/5000
12392/12392 [==============================] - 34s 3ms/step - loss: 0.5578 - accuracy: 0.7209 - val_loss: 0.6790 - val_accuracy: 0.
Epoch 20/5000
12392/12392 [==============================] - 38s 3ms/step - loss: 0.5569 - accuracy: 0.7221 - val_loss: 0.6860 - val_accuracy: 0.
Epoch 21/5000
12392/12392 [==============================] - 35s 3ms/step - loss: 0.5600 - accuracy: 0.7165 - val_loss: 0.6934 - val_accuracy: 0.
Epoch 22/5000
12392/12392 [==============================] - 33s 3ms/step - loss: 0.5621 - accuracy: 0.7150 - val_loss: 0.7218 - val_accuracy: 0.
Epoch 22: early stopping
```

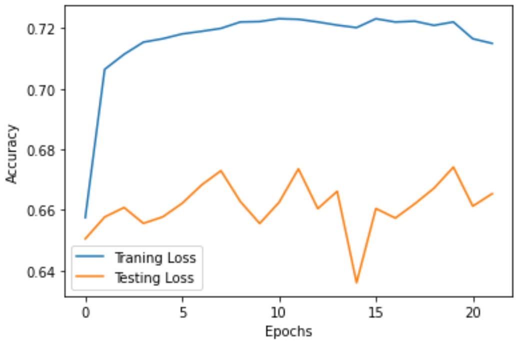
```
# Here we see that out model is Underfit ....No we will increse the no of Hidden layer
print("Traning Loss and Accuracy: ", model_1.evaluate(X_train1, Y_train1))
print("Testing Loss and Accuracy: ", model_1.evaluate(X_test1, Y_test1))
```

```
12392/12392 [==============================] - 21s 2ms/step - loss: 0.5185 - accuracy: 0.7544
Traning Loss and Accuracy:  [0.5184708833694458, 0.7543553709983826]
5313/5313 [==============================] - 8s 2ms/step - loss: 0.7218 - accuracy: 0.6654
Testing Loss and Accuracy:  [0.7218478918075562, 0.6653900742530823]
```

```
# Visulisation
plt.plot(train_model.history['loss'],label = 'Traning Loss')
plt.plot(train_model.history['val_loss'],label = 'Testing Loss')
plt.xlabel('Epochs')
plt.ylabel('binary_crossentropy')
plt.legend()
plt.show()
```




```
# Visulisation
plt.plot(train_model.history['accuracy'],label = 'Traning Loss')
plt.plot(train_model.history['val_accuracy'],label = 'Testing Loss')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



Still it look like overfit model. Now we use regularizer with Dropout.

```
# Creat model

from tensorflow.keras import regularizers
model_2 = tf.keras.Sequential([
    # First Layer
    tf.keras.layers.Dense(units = 64, activation = 'relu',
                          kernel_regularizer = regularizers.l2(0.01),input_shape = (X.shape[1],)),Dropout(0.
    # Second Layer
    tf.keras.layers.Dense(units = 32, activation = 'relu',kernel_regularizer = regularizers.l2(0.01)),Dropou
    # Third Layer
    tf.keras.layers.Dense(units = 16, activation = 'relu',kernel_regularizer = regularizers.l2(0.01)),Dropou
    # fourth Layer
    tf.keras.layers.Dense(units = 8, activation = 'relu',kernel_regularizer = regularizers.l2(0.01)),Dropout
    # output
    tf.keras.layers.Dense(units = 1, activation = 'sigmoid',kernel_regularizer = regularizers.l2(0.01)),
                              ])
```

```
model_2.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

```
# train the model: use inbuilt method fit() of Sequential class

train_model = model_2.fit(X_train1, Y_train1,epochs =5000,validation_data=(X_test1,Y_test1), callbacks = cb)
```

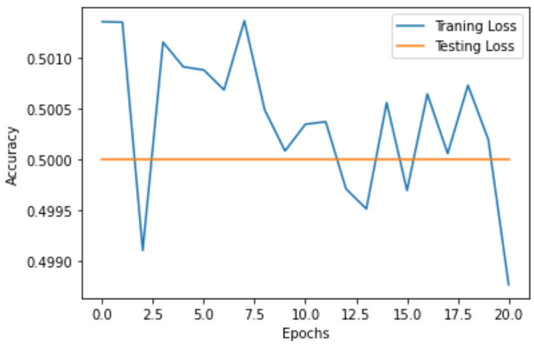
```
Epoch 1/5000
12392/12392 [==============================] - 37s 3ms/step - loss: 0.7009 - accuracy: 0.5014 - val_loss: 0.6932 - val_accuracy: 0.
Epoch 2/5000
12392/12392 [==============================] - 35s 3ms/step - loss: 0.6932 - accuracy: 0.5013 - val_loss: 0.6932 - val_accuracy: 0.
Epoch 3/5000
12392/12392 [==============================] - 39s 3ms/step - loss: 0.6932 - accuracy: 0.4991 - val_loss: 0.6931 - val_accuracy: 0.
Epoch 4/5000
12392/12392 [==============================] - 40s 3ms/step - loss: 0.6932 - accuracy: 0.5012 - val_loss: 0.6932 - val_accuracy: 0.
Epoch 5/5000
12392/12392 [==============================] - 36s 3ms/step - loss: 0.6932 - accuracy: 0.5009 - val_loss: 0.6932 - val_accuracy: 0.
Epoch 6/5000
12392/12392 [==============================] - 39s 3ms/step - loss: 0.6932 - accuracy: 0.5009 - val_loss: 0.6932 - val_accuracy: 0.
Epoch 7/5000
12392/12392 [==============================] - 35s 3ms/step - loss: 0.6932 - accuracy: 0.5007 - val_loss: 0.6932 - val_accuracy: 0.
Epoch 8/5000
12392/12392 [==============================] - 39s 3ms/step - loss: 0.6932 - accuracy: 0.5014 - val_loss: 0.6933 - val_accuracy: 0.
Epoch 9/5000
12392/12392 [==============================] - 36s 3ms/step - loss: 0.6932 - accuracy: 0.5005 - val_loss: 0.6932 - val_accuracy: 0.
Epoch 10/5000
```

```
12392/12392 [==============================] - 36s 3ms/step - loss: 0.6932 - accuracy: 0.5001 - val_loss: 0.6932 - val_accuracy: 0.
Epoch 11/5000
12392/12392 [==============================] - 40s 3ms/step - loss: 0.6932 - accuracy: 0.5003 - val_loss: 0.6932 - val_accuracy: 0.
Epoch 12/5000
12392/12392 [==============================] - 36s 3ms/step - loss: 0.6932 - accuracy: 0.5004 - val_loss: 0.6932 - val_accuracy: 0.
Epoch 13/5000
12392/12392 [==============================] - 40s 3ms/step - loss: 0.6932 - accuracy: 0.4997 - val_loss: 0.6931 - val_accuracy: 0.
Epoch 14/5000
12392/12392 [==============================] - 40s 3ms/step - loss: 0.6932 - accuracy: 0.4995 - val_loss: 0.6932 - val_accuracy: 0.
Epoch 15/5000
12392/12392 [==============================] - 42s 3ms/step - loss: 0.6932 - accuracy: 0.5006 - val_loss: 0.6931 - val_accuracy: 0.
Epoch 16/5000
12392/12392 [==============================] - 40s 3ms/step - loss: 0.6932 - accuracy: 0.4997 - val_loss: 0.6932 - val_accuracy: 0.
Epoch 17/5000
12392/12392 [==============================] - 44s 4ms/step - loss: 0.6932 - accuracy: 0.5006 - val_loss: 0.6932 - val_accuracy: 0.
Epoch 18/5000
12392/12392 [==============================] - 36s 3ms/step - loss: 0.6932 - accuracy: 0.5001 - val_loss: 0.6932 - val_accuracy: 0.
Epoch 19/5000
12392/12392 [==============================] - 44s 4ms/step - loss: 0.6932 - accuracy: 0.5007 - val_loss: 0.6932 - val_accuracy: 0.
Epoch 20/5000
12392/12392 [==============================] - 35s 3ms/step - loss: 0.6932 - accuracy: 0.5002 - val_loss: 0.6932 - val_accuracy: 0.
Epoch 21/5000
12392/12392 [==============================] - 35s 3ms/step - loss: 0.6932 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.
Epoch 21: early stopping
```

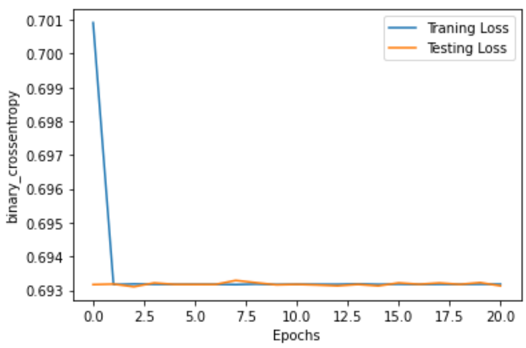
```
# Here we see that out model is Underfit ....No we will increse the no of Hidden layer
print("Traning Loss and Accuracy: ", model_2.evaluate(X_train1, Y_train1))
print("Testing Loss and Accuracy: ", model_2.evaluate(X_test1, Y_test1))
```

```
12392/12392 [==============================] - 19s 2ms/step - loss: 0.6931 - accuracy: 0.5000
Traning Loss and Accuracy:  [0.6931232810020447, 0.5]
5313/5313 [==============================] - 8s 2ms/step - loss: 0.6931 - accuracy: 0.5000
Testing Loss and Accuracy:  [0.6931323409080505, 0.5]
```

```
# Visulisation
plt.plot(train_model.history['accuracy'],label = 'Traning Loss')
plt.plot(train_model.history['val_accuracy'],label = 'Testing Loss')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```




```
# Visulisation
plt.plot(train_model.history['loss'],label = 'Traning Loss')
plt.plot(train_model.history['val_loss'],label = 'Testing Loss')
plt.xlabel('Epochs')
plt.ylabel('binary_crossentropy')
plt.legend()
plt.show()
```



0s completed at 2:17 PM