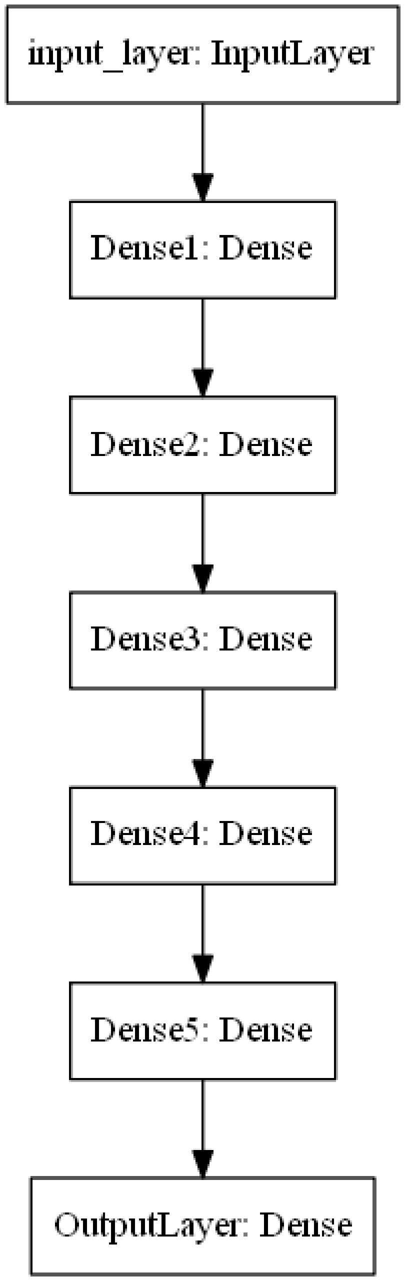
1. Download the data from here

2. Code the model to classify data like below image


input_layer: InputLayer
Dense1: Dense
Dense2: Dense
Dense3: Dense
Dense4: Dense
Dense5: Dense
OutputLayer: Dense

```
3. Write your own callback function, that has to print the micro F1 score and AUC score after each
epoch.

4. Save your model at every epoch if your validation accuracy is improved from previous epoch.

5. you have to decay learning based on below conditions
        Cond1. If your validation accuracy at that epoch is less than previous epoch accuracy, you
have to decrese the
               learning rate by 10%.
        Cond2. For every 3rd epoch, decay your learning rate by 5%.

6. If you are getting any NaN values(either weigths or loss) while training, you have to terminate
your training.

7. You have to stop the training if your validation accuracy is not increased in last 2 epochs.

8. Use tensorboard for every model and analyse your gradients. (you need to upload the screenshots
for each model for evaluation)

9. use cross entropy as loss function

10. Try the architecture params as given below.
```

Model-1

1. Use tanh as an activation for every layer except output layer.
2. use SGD with momentum as optimizer.
3. use RandomUniform(0,1) as initilizer.
3. Analyze your output and training process.

Model-2

1. Use relu as an activation for every layer except output layer.
2. use SGD with momentum as optimizer.
3. use RandomUniform(0,1) as initilizer.
3. Analyze your output and training process.

```
Model-3
1. Use relu as an activation for every layer except output layer.
2. use SGD with momentum as optimizer.
3. use he_uniform() as initilizer.
3. Analyze your output and training process.


Model-4
1. Try with any values to get better accuracy/f1 score.
```

IMPORT LIBRARIES & READ THE DATASET

In [1]:

```
#  necessary libraries
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
```

In [3]:

```
# Read the data & check the shape of it
mydata = pd.read_csv('data.csv')
print("The shape of the data :-",mydata.shape)
mydata.head()
```

```
The shape of the data :- (20000, 3)
```

Out[3]:

	f1	f2	label
0	0.450564	1.074305	0.0
1	0.085632	0.967682	0.0
2	0.117326	0.971521	1.0
3	0.982179	-0.380408	0.0
4	-0.720352	0.955850	0.0

In [4]:

```
mydata['label'].value_counts() #--> perfectly balanced data
```

Out[4]:

```
1.0    10000
0.0    10000
Name: label, dtype: int64
```

SPLIT THE DATA INTO TRAIN & TEST

In [5]:

```
# split the data into train & test
from sklearn.model_selection import train_test_split

X = mydata[['f1','f2']]
y = mydata['label']

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.30,random_state=42)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(14000, 2)
(6000, 2)
(14000,)
(6000,)
```

In [6]:

```
# Preprocessing target column to one-hot encoded column
Y_train = keras.utils.to_categorical(y_train, 2)
Y_test = keras.utils.to_categorical(y_test, 2)
print(Y_train.shape)
print(Y_test.shape)
```

```
(14000, 2)
(6000, 2)
```

In [7]:

```
# for add-on metrics we need this library
!pip install tensorflow-addons
```

```
Collecting tensorflow-addons
```

```
  Downloading https://files.pythonhosted.org/packages/66/4b/e893d194e626c24b3df2253066aa418f46a432fdb68250cde14bf9bb0
700/tensorflow_addons-0.13.0-cp37-cp37m-manylinux2010_x86_64.whl (679kB)
     |████████████████████████████████| 686kB 4.3MB/s 
Requirement already satisfied: typeguard>=2.7 in /usr/local/lib/python3.7/dist-packages (from tensorflow-addons) (2.
7.1)
Installing collected packages: tensorflow-addons
Successfully installed tensorflow-addons-0.13.0
```

In [8]:

```python
# To check the version of tensorflow
print(tf.__version__)
```

```
2.5.0
```

REQUIRED TENSORFLOW-KERAS LIBRARIES

In [79]:

```python
# required tensorflow & keras libraries to  build deep learning models
from keras.layers import Dense,Input,Activation,BatchNormalization
from keras.models import Model
import random as rn
import tensorflow_addons as tfa
```

In [10]:

```python
# callbacks for various purposes
from keras.callbacks import ModelCheckpoint
from keras.callbacks import LearningRateScheduler
from keras.callbacks import EarlyStopping
```

CUSTOM CALLBACK FUNCTION

In [53]:

```python
# Required libraries
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score

# Our Custom Callback function()
class LossHistory(keras.callbacks.Callback):

    def on_train_begin(self, logs={}):
        # on begin of training, we are creating an instance varible called history
        # it is a dict with keys as various metrics & to hold their respective values
        self.history={'loss': [],'acc': [],'val_loss': [],'val_acc' : []}
        self.train_f1score = [] ;self.val_f1score = []
        self.train_auc = [] ;self.val_auc = []

    def on_epoch_end(self, epoch, logs={}):

        # On epoch end, we wanted to display the metric values for train & test data
        # and also their corresponding losses.
        loss = logs.get('loss')
        # Terminate the training process when the loss is ' nan / inf '
        if loss is not None:
            if np.isnan(loss) or np.isinf(loss):
                print("Invalid loss and terminated at epoch {}".format(epoch))
                self.model.stop_training = True

        # Access the layer weights & terminate when their values are not valid numbers
        L1_w = self.model.layers[1].get_weights()[0]
        L2_w = self.model.layers[2].get_weights()[0]
        L3_w = self.model.layers[3].get_weights()[0]
        L4_w = self.model.layers[4].get_weights()[0]
        L5_w = self.model.layers[5].get_weights()[0]

        if (np.isnan(L1_w).any()) or (np.isnan(L2_w).any()) or (np.isnan(L3_w).any()) or (np.isnan(L4_w).any()) or (n
            print("Invalid weights at epoch {}".format(epoch))
            self.model.stop_training = True

        if (np.isinf(L1_w).any()) or (np.isinf(L2_w).any()) or (np.isinf(L3_w).any()) or (np.isinf(L4_w).any()) or (n
            print("Invalid weights at epoch {}".format(epoch))
            self.model.stop_training = True


        ## on end of each epoch, we will get logs dict and update the self.history dictionary
        self.history['loss'].append(logs.get('loss'))
        self.history['acc'].append(logs.get('acc'))

           # F1SCORE
        y_pred_train = (np.asarray(self.model.predict(X_train.to_numpy()))).round()
        y_true_train = np.asarray(Y_train)
        y_pred_val = (np.asarray(self.model.predict(X_test.to_numpy()))).round()
        y_true_val = np.asarray(Y_test)
        f1_train =  f1_score(y_true_train,y_pred_train,average = 'micro')
        f1_val = f1_score(y_true_val,y_pred_val,average = 'micro')
        self.train_f1score.append(f1_train)
        self.val_f1score.append(f1_val)
        print("Train F1-score : {} - Val F1-score : {}".format(f1_train,f1_val))

           # AUC
        y_pred_train_auc = (np.asarray(self.model.predict(X_train.to_numpy())))
        y_pred_val_auc = (np.asarray(self.model.predict(X_test.to_numpy())))
        auc_train = roc_auc_score(y_true_train,y_pred_train_auc)
        auc_val = roc_auc_score(y_true_val,y_pred_val_auc)
        self.train_auc.append(auc_train)
        self.val_auc.append(auc_val)
        print("Train AUC : {} - Val AUC : {}".format(auc_train,auc_val))

          # Val loss & accuracy
        if logs.get('val_loss', -1) != -1:
            self.history['val_loss'].append(logs.get('val_loss'))
        if logs.get('val_acc', -1) != -1:
            self.history['val_acc'].append(logs.get('val_acc'))

# Creating an object to access the class LossHistory()
history_own = LossHistory()
```

CHANGE LEARNING RATE FUNCTION

In [54]:

```
# Custom learning rate scheduler function:
def changeLearningRate(epoch,Lr):

    # Access the validation accuracies after each epoch &
    # then based on the conditions we decay the learning rate by 10%
    val_accuracies = history_own.history['val_acc']
    if (epoch > 1):
        if (val_accuracies[-1] < val_accuracies[-2]):
            Lr = ( (Lr) - (Lr * (0.1)) )
        else:
            pass
    else:
        pass

    # For every 3rd epoch we decay the learning rate by 5%
    if ((epoch+1) % 3 == 0):
        Lr = ( (Lr) - (Lr * (0.05)) )
    else:
        pass

    return Lr # finally return the updated learning rate.
```

LAUNCH ,REFRESH & DISPLAY TENSORBOARD RESULTS

In [55]:

```
# %load_ext tensorboard
%load_ext tensorboard
```

In [ ]:

```
!kill 1604
```

In [57]:

```
# Clear any logs of previous executions/runs
import shutil
path = './logfiles_model1'
shutil.rmtree(path,ignore_errors=True)
```

In [58]:

```
# Tensorboard outputs
import os ; import datetime
#Reference :--> https://www.dlology.com/blog/how-to-run-tensorboard-in-jupyter-notebook/
logs_base_dir = "./logfiles_model1"
os.makedirs(logs_base_dir, exist_ok = True) # Creating a directory to store logs
# Launch the tensorboard  & then refresh it after executing the model.
%tensorboard --logdir {logs_base_dir}
```

MODEL 1 & OUTPUT

In [59]:

```
# MODEL 1 ARCHITECTURE:-

tf.keras.backend.clear_session() # For easy reset of notebook state.
initializer = keras.initializers.RandomUniform(minval=0.,maxval=1.,seed = 6) # weights Initializer
```

```python
#Input layer
input_layer = Input(shape=(2,))
#Dense hidden layer 1
layer1 = Dense(512,activation='tanh',kernel_initializer=initializer)(input_layer)
#Dense hidden layer 2
layer2 = Dense(512,activation='tanh',kernel_initializer=initializer)(layer1)
#Dense hidden layer 3
layer3 = Dense(256,activation='tanh',kernel_initializer=initializer)(layer2)
#Dense hidden layer 4
layer4 = Dense(256,activation='tanh',kernel_initializer=initializer)(layer3)
#Dense hidden layer 5
layer5 = Dense(100,activation='tanh',kernel_initializer=initializer)(layer4)
#output layer
output = Dense(2,activation='softmax',kernel_initializer=initializer)(layer5)

#Creating a model
model = Model(inputs=input_layer,outputs=output)

#Callbacks
history_own = LossHistory()

# Optimizer
optimizer = keras.optimizers.SGD(learning_rate=0.01,momentum= 0.9)

#save model at every epoch end if validation accuracy is improved
filepath="model_save_1/weights-{epoch:02d}-{val_acc:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_acc',verbose=1, save_best_only=True, mode='auto')

#Learning rate scheduler
lrschedule = LearningRateScheduler(changeLearningRate, verbose=1)

#early stopping
earlystop = EarlyStopping(monitor='val_acc',patience=2, verbose=1)

#tensorboard callbacks # reference :-:--> https://www.dlology.com/blog/how-to-run-tensorboard-in-jupyter-notebook/
log_dir = os.path.join(logs_base_dir,datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tb_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1,write_graph=True)

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

```
# Fit the model
model.fit(X_train,Y_train,epochs=10,validation_data=(X_test,Y_test),
          batch_size=32,callbacks=[history_own,lrschedule,checkpoint,earlystop,tb_callback])
```

```
Epoch 1/10

Epoch 00001: LearningRateScheduler reducing learning rate to 0.009999999776482582.
  3/438 [..............................] - ETA: 14s - loss: 25.2084 - acc: 0.4062 WARNING:tensorflow:Callback method
`on_train_batch_end` is slow compared to the batch time (batch time: 0.0073s vs `on_train_batch_end` time: 0.0086s).
Check your callbacks.
438/438 [==============================] - 5s 9ms/step - loss: 4.8033 - acc: 0.4942 - val_loss: 0.7163 - val_acc: 0.5
060
Train F1-score : 0.4997857142857143 - Val F1-score : 0.506
Train AUC : 0.4997926923502436 - Val AUC : 0.5060202970363589

Epoch 00001: val_acc improved from -inf to 0.50600, saving model to model_save_1/weights-01-0.5060.hdf5
Epoch 2/10

Epoch 00002: LearningRateScheduler reducing learning rate to 0.009999999776482582.
438/438 [==============================] - 4s 9ms/step - loss: 0.7028 - acc: 0.4959 - val_loss: 0.7283 - val_acc: 0.5
060
Train F1-score : 0.4997857142857143 - Val F1-score : 0.506
Train AUC : 0.4997926923502436 - Val AUC : 0.5060202970363589

Epoch 00002: val_acc did not improve from 0.50600
Epoch 3/10

Epoch 00003: LearningRateScheduler reducing learning rate to 0.009499999787658453.
438/438 [==============================] - 4s 8ms/step - loss: 0.7059 - acc: 0.5038 - val_loss: 0.6957 - val_acc: 0.4
940
Train F1-score : 0.5002142857142857 - Val F1-score : 0.494
Train AUC : 0.5002073076497564 - Val AUC : 0.49397970296364113

Epoch 00003: val_acc did not improve from 0.50600
Epoch 00003: early stopping
```

```
Out[59]: <keras.callbacks.History at 0x7f09c7204190>
```

LAUNCH, REFRESH & DISPLAY TENSORBOARD RESULTS

```
In [60]:
# Clear any logs of previous executions/runs
import shutil
```

```
path = './logfiles_model2'
shutil.rmtree(path,ignore_errors=True)
```

In [61]:

```
# Tensorboard outputs
import os ; import datetime
#Reference :--> https://www.dlology.com/blog/how-to-run-tensorboard-in-jupyter-notebook/
logs_base_dir = "./logfiles_model2"
os.makedirs(logs_base_dir, exist_ok = True) # Creating a directory to store logs
# Launch the tensorboard  & then refresh it after executing the model.
%tensorboard --logdir {logs_base_dir}
```

MODEL 2 & OUTPUT

In [62]:

```
# MODEL 2 ARCHITECTURE:-

tf.keras.backend.clear_session() # For easy reset of notebook state.
initializer = keras.initializers.RandomUniform(minval=0.,maxval=1.,seed = 6) # Weights initializer

#Input layer
input_layer = Input(shape=(2,))
#Dense hidden layer 1
layer1 = Dense(512,activation='relu',kernel_initializer=initializer)(input_layer)
#Dense hidden layer 2
layer2 = Dense(512,activation='relu',kernel_initializer=initializer)(layer1)
#Dense hidden layer 3
layer3 = Dense(256,activation='relu',kernel_initializer=initializer)(layer2)
#Dense hidden layer 4
layer4 = Dense(256,activation='relu',kernel_initializer=initializer)(layer3)
#Dense hidden layer 5
layer5 = Dense(100,activation='relu',kernel_initializer=initializer)(layer4)
#output layer
output = Dense(2,activation='softmax',kernel_initializer=initializer)(layer5)

#Creating a model
model = Model(inputs=input_layer,outputs=output)

#Callbacks
history_own = LossHistory()
```

```python
#optimizer
optimizer = keras.optimizers.SGD(learning_rate=0.01,momentum= 0.9)

#save model at every epoch end if validation accuracy is improved
filepath="model_save_2/weights-{epoch:02d}-{val_acc:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_acc',verbose=1,save_best_only=True, mode='auto')

#LR scheduler
lrschedule = LearningRateScheduler(changeLearningRate,verbose=1)

#early stopping
earlystop = EarlyStopping(monitor='val_acc',patience=2, verbose=1)

#tensorboard callbacks # reference :-:--> https://www.dlology.com/blog/how-to-run-tensorboard-in-jupyter-notebook/
log_dir = os.path.join(logs_base_dir,datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tb_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1,write_graph=True)

#Compile the model
model.compile(optimizer=optimizer, loss='binary_crossentropy',
              metrics = ['acc'])
#Fit the model
model.fit(X_train,Y_train,epochs=10,validation_data=(X_test,Y_test),
          batch_size=32,callbacks=[history_own,lrschedule,checkpoint,earlystop,tb_callback])
```

```
Epoch 1/10

Epoch 00001: LearningRateScheduler reducing learning rate to 0.009999999776482582.
  3/438 [..............................] - ETA: 20s - loss: 3736841685.3333 - acc: 0.4219 WARNING:tensorflow:Callback
method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0096s vs `on_train_batch_end` time: 0.0
136s). Check your callbacks.
438/438 [==============================] - 5s 10ms/step - loss: 92807046.0820 - acc: 0.4897 - val_loss: 0.6934 - val_
acc: 0.4968
Train F1-score : 0.5013571428571428 - Val F1-score : 0.49683333333333335
Train AUC : 0.5 - Val AUC : 0.5

Epoch 00001: val_acc improved from -inf to 0.49683, saving model to model_save_2/weights-01-0.4968.hdf5
Epoch 2/10

Epoch 00002: LearningRateScheduler reducing learning rate to 0.009999999776482582.
438/438 [==============================] - 4s 9ms/step - loss: 0.6933 - acc: 0.4973 - val_loss: 0.6932 - val_acc: 0.4
968
Train F1-score : 0.5013571428571428 - Val F1-score : 0.49683333333333335
Train AUC : 0.5 - Val AUC : 0.5

Epoch 00002: val_acc did not improve from 0.49683
Epoch 3/10

Epoch 00003: LearningRateScheduler reducing learning rate to 0.009499999787658453.
438/438 [==============================] - 4s 9ms/step - loss: 0.6933 - acc: 0.5064 - val_loss: 0.6931 - val_acc: 0.5
032
Train F1-score : 0.49864285714285717 - Val F1-score : 0.5031666666666667
Train AUC : 0.5 - Val AUC : 0.5

Epoch 00003: val_acc improved from 0.49683 to 0.50317, saving model to model_save_2/weights-03-0.5032.hdf5
Epoch 4/10

Epoch 00004: LearningRateScheduler reducing learning rate to 0.009499999694526196.
438/438 [==============================] - 4s 9ms/step - loss: 0.6932 - acc: 0.4982 - val_loss: 0.6932 - val_acc: 0.4
968
Train F1-score : 0.5013571428571428 - Val F1-score : 0.49683333333333335
Train AUC : 0.5 - Val AUC : 0.5

Epoch 00004: val_acc did not improve from 0.50317
Epoch 5/10

Epoch 00005: LearningRateScheduler reducing learning rate to 0.008549999725073577.
438/438 [==============================] - 4s 9ms/step - loss: 0.6932 - acc: 0.4932 - val_loss: 0.6931 - val_acc: 0.5
032
Train F1-score : 0.49864285714285717 - Val F1-score : 0.5031666666666667
Train AUC : 0.5 - Val AUC : 0.5

Epoch 00005: val_acc did not improve from 0.50317
Epoch 00005: early stopping
```

```
Out[62]: <keras.callbacks.History at 0x7f09bd3a8890>
```

LAUNCH, REFRESH & DISPLAY TENSORBOARD RESULTS

```
In [70]:
# Clear any logs of previous executions/runs
import shutil
path = './logfiles_model3'
shutil.rmtree(path,ignore_errors=True)
```

```
In [69]:
!kill 1754
```

In [71]:

```
# Tensorboard outputs
import os ; import datetime
#Reference :--> https://www.dlology.com/blog/how-to-run-tensorboard-in-jupyter-notebook/
logs_base_dir = "./logfiles_model3"
os.makedirs(logs_base_dir, exist_ok = True) # Creating a directory to store logs
# Launch the tensorboard  & then refresh it after executing the model.
%tensorboard --logdir {logs_base_dir}
```

MODEL 3 & OUTPUT

In [72]:

```
# MODEL 3 ARCHITECTURE:-

tf.keras.backend.clear_session() # For easy reset of notebook state.
initializer = tf.keras.initializers.HeUniform(seed = 6) # weights initializer

#Input layer
input_layer = Input(shape=(2,))
#Dense hidden layer 1
layer1 = Dense(512,activation='relu',kernel_initializer=initializer)(input_layer)
#Dense hidden layer 2
layer2 = Dense(512,activation='relu',kernel_initializer=initializer)(layer1)
#Dense hidden layer 3
layer3 = Dense(256,activation='relu',kernel_initializer=initializer)(layer2)
#Dense hidden layer 4
layer4 = Dense(256,activation='relu',kernel_initializer=initializer)(layer3)
#Dense hidden layer 5
layer5 = Dense(100,activation='relu',kernel_initializer=initializer)(layer4)
#output layer
output = Dense(2,activation='softmax',kernel_initializer=initializer)(layer5)

#Creating a model
model = Model(inputs=input_layer,outputs=output)

#Callbacks
history_own = LossHistory()

#Optimizer
```

```
optimizer = keras.optimizers.SGD(learning_rate=0.01,momentum= 0.9)

#save model at every epoch end if validation accuracy is improved
filepath="model_save_3/weights-{epoch:02d}-{val_acc:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_acc',verbose=1, save_best_only=True, mode='auto')

#LR scheduler
lrschedule = LearningRateScheduler(changeLearningRate, verbose=1)

#early stopping
earlystop = EarlyStopping(monitor='val_acc',patience=2, verbose=1)

#tensorboard callbacks # reference :-:--> https://www.dlology.com/blog/how-to-run-tensorboard-in-jupyter-notebook/
log_dir = os.path.join(logs_base_dir,datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tb_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1,write_graph=True)

#Model compile
model.compile(optimizer=optimizer, loss='binary_crossentropy',
              metrics = ['acc'])
#Fit the model
model.fit(X_train,Y_train,epochs=10,validation_data=(X_test,Y_test),
          batch_size=32,callbacks=[history_own,lrschedule,checkpoint,earlystop,tb_callback])
```

```
Epoch 1/10

Epoch 00001: LearningRateScheduler reducing learning rate to 0.009999999776482582.
  3/438 [..............................] - ETA: 18s - loss: 0.7721 - acc: 0.4583 WARNING:tensorflow:Callback method `
on_train_batch_end` is slow compared to the batch time (batch time: 0.0091s vs `on_train_batch_end` time: 0.0119s). C
heck your callbacks.
438/438 [==============================] - 5s 10ms/step - loss: 0.6919 - acc: 0.5462 - val_loss: 0.6497 - val_acc: 0.
6317
Train F1-score : 0.6515 - Val F1-score : 0.6316666666666667
Train AUC : 0.7325828716819731 - Val AUC : 0.7093652589842769

Epoch 00001: val_acc improved from -inf to 0.63167, saving model to model_save_3/weights-01-0.6317.hdf5
Epoch 2/10

Epoch 00002: LearningRateScheduler reducing learning rate to 0.009999999776482582.
438/438 [==============================] - 4s 9ms/step - loss: 0.6405 - acc: 0.6468 - val_loss: 0.6338 - val_acc: 0.6
443
Train F1-score : 0.6512142857142857 - Val F1-score : 0.6443333333333333
Train AUC : 0.7224026640277901 - Val AUC : 0.7060214026362612
```

```
Epoch 00002: val_acc improved from 0.63167 to 0.64433, saving model to model_save_3/weights-02-0.6443.hdf5
Epoch 3/10

Epoch 00003: LearningRateScheduler reducing learning rate to 0.009499999787658453.
438/438 [==============================] - 4s 9ms/step - loss: 0.6205 - acc: 0.6615 - val_loss: 0.6382 - val_acc: 0.6
337
Train F1-score : 0.6520714285714285 - Val F1-score : 0.6336666666666667
Train AUC : 0.7314485265901653 - Val AUC : 0.7130174888126068

Epoch 00003: val_acc did not improve from 0.64433
Epoch 4/10

Epoch 00004: LearningRateScheduler reducing learning rate to 0.008549999725073577.
438/438 [==============================] - 4s 9ms/step - loss: 0.6121 - acc: 0.6602 - val_loss: 0.6175 - val_acc: 0.6
638
Train F1-score : 0.6692142857142858 - Val F1-score : 0.6638333333333334
Train AUC : 0.7394239475927568 - Val AUC : 0.719056175475483

Epoch 00004: val_acc improved from 0.64433 to 0.66383, saving model to model_save_3/weights-04-0.6638.hdf5
Epoch 5/10

Epoch 00005: LearningRateScheduler reducing learning rate to 0.008549999445676804.
438/438 [==============================] - 4s 9ms/step - loss: 0.6034 - acc: 0.6666 - val_loss: 0.6200 - val_acc: 0.6
583
Train F1-score : 0.6686428571428571 - Val F1-score : 0.6583333333333333
Train AUC : 0.7376199343019649 - Val AUC : 0.716843808957226

Epoch 00005: val_acc did not improve from 0.66383
Epoch 6/10

Epoch 00006: LearningRateScheduler reducing learning rate to 0.007310249526053667.
438/438 [==============================] - 4s 9ms/step - loss: 0.6097 - acc: 0.6628 - val_loss: 0.6170 - val_acc: 0.6
593
Train F1-score : 0.6714285714285714 - Val F1-score : 0.6593333333333333
Train AUC : 0.7380209627666849 - Val AUC : 0.7185822120198376

Epoch 00006: val_acc did not improve from 0.66383
Epoch 00006: early stopping
```

Out[72]:

```
<keras.callbacks.History at 0x7f09bcfbf450>
```

LAUNCH, REFRESH & DISPLAY TENSORBOARD RESULTS

In [80]:

```
%reload_ext tensorboard
```

In [104…

```
# Clear any logs of previous executions/runs
import shutil
path = './logfiles_model4'
shutil.rmtree(path,ignore_errors=True)
```

In [106…

```
!kill 3974
```

In [107…

```
# Tensorboard outputs
import os ; import datetime
#Reference :--> https://www.dlology.com/blog/how-to-run-tensorboard-in-jupyter-notebook/
logs_base_dir = "./logfiles_model4"
os.makedirs(logs_base_dir, exist_ok = True) # Creating a directory to store logs
# Launch the tensorboard  & then refresh it after executing the model.
%tensorboard --logdir {logs_base_dir}
```

MODEL 4 & OUTPUT

In [108…

```
# MODEL 4 ARCHITECTURE:-

tf.keras.backend.clear_session() # For easy reset of notebook state.
initializer = keras.initializers.HeUniform(seed = 6)

#Input layer
input_layer = Input(shape=(2,))
#Dense hidden layer 1
layer1 = Dense(100,activation='relu',kernel_initializer=initializer)(input_layer)
#Dense hidden layer 2
layer2 = Dense(128,activation='relu',kernel_initializer=initializer)(layer1)
#Dense hidden layer 3
layer3 = Dense(128,activation='relu',kernel_initializer=initializer)(layer2)
#Dense hidden layer 4
layer4 = Dense(256,activation='relu',kernel_initializer=initializer)(layer3)
#Dense hidden layer 5
layer5 = Dense(256,activation='relu',kernel_initializer=initializer)(layer4)
```

```
#output layer
output = Dense(2,activation='softmax',kernel_initializer=initializer)(layer5)

#Creating a model
model = Model(inputs=input_layer,outputs=output)

#Callbacks
history_own = LossHistory()
optimizer = keras.optimizers.Adam(learning_rate=0.01)

#save model at every epoch end if validation accuracy is improved
filepath="model_save_4/weights-{epoch:02d}-{val_acc:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_acc',verbose=1, save_best_only=True, mode='auto')

#LR scheduler
lrschedule = LearningRateScheduler(changeLearningRate,verbose=1)

#early stopping
earlystop = EarlyStopping(monitor='val_acc',patience=2, verbose=1)

#tensorboard
log_dir = os.path.join(logs_base_dir,datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tb_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1,write_graph=True)

#Compile the model
model.compile(optimizer=optimizer, loss='binary_crossentropy',
              metrics = ['acc'])
#Fit the model
model.fit(X_train,Y_train,epochs=15,validation_data=(X_test,Y_test),
          batch_size=32,callbacks=[history_own,checkpoint,earlystop,lrschedule,tb_callback])
```

```
Epoch 1/15

Epoch 00001: LearningRateScheduler reducing learning rate to 0.009999999776482582.
  3/438 [..............................] - ETA: 15s - loss: 2.2322 - acc: 0.4236 WARNING:tensorflow:Callback method `
on_train_batch_end` is slow compared to the batch time (batch time: 0.0036s vs `on_train_batch_end` time: 0.0110s). C
heck your callbacks.
438/438 [==============================] - 3s 4ms/step - loss: 0.7317 - acc: 0.5922 - val_loss: 0.6224 - val_acc: 0.6
510
Train F1-score : 0.6647142857142857 - Val F1-score : 0.651
Train AUC : 0.7327088675081872 - Val AUC : 0.7137863529859365

Epoch 00001: val_acc improved from -inf to 0.65100, saving model to model_save_4/weights-01-0.6510.hdf5
Epoch 2/15

Epoch 00002: LearningRateScheduler reducing learning rate to 0.009999999776482582.
438/438 [==============================] - 2s 4ms/step - loss: 0.6049 - acc: 0.6727 - val_loss: 0.6197 - val_acc: 0.6
647
Train F1-score : 0.6754285714285714 - Val F1-score : 0.6646666666666666
Train AUC : 0.7415507234655341 - Val AUC : 0.7195692793899844

Epoch 00002: val_acc improved from 0.65100 to 0.66467, saving model to model_save_4/weights-02-0.6647.hdf5
Epoch 3/15

Epoch 00003: LearningRateScheduler reducing learning rate to 0.009499999787658453.
438/438 [==============================] - 2s 3ms/step - loss: 0.6047 - acc: 0.6681 - val_loss: 0.6433 - val_acc: 0.6
463
Train F1-score : 0.6549285714285714 - Val F1-score : 0.6463333333333333
Train AUC : 0.7409609385081388 - Val AUC : 0.7210135595438882

Epoch 00003: val_acc did not improve from 0.66467
Epoch 4/15

Epoch 00004: LearningRateScheduler reducing learning rate to 0.008549999725073577.
438/438 [==============================] - 2s 3ms/step - loss: 0.6033 - acc: 0.6746 - val_loss: 0.6115 - val_acc: 0.6
660
Train F1-score : 0.6743571428571429 - Val F1-score : 0.666
Train AUC : 0.7451869288261491 - Val AUC : 0.7247912388485804

Epoch 00004: val_acc improved from 0.66467 to 0.66600, saving model to model_save_4/weights-04-0.6660.hdf5
Epoch 5/15

Epoch 00005: LearningRateScheduler reducing learning rate to 0.008549999445676804.
438/438 [==============================] - 2s 4ms/step - loss: 0.5997 - acc: 0.6760 - val_loss: 0.6116 - val_acc: 0.6
695
Train F1-score : 0.6734285714285714 - Val F1-score : 0.6695
Train AUC : 0.7455386303968483 - Val AUC : 0.7245154222297139

Epoch 00005: val_acc improved from 0.66600 to 0.66950, saving model to model_save_4/weights-05-0.6695.hdf5
Epoch 6/15

Epoch 00006: LearningRateScheduler reducing learning rate to 0.008122499473392964.
438/438 [==============================] - 2s 4ms/step - loss: 0.5984 - acc: 0.6760 - val_loss: 0.6147 - val_acc: 0.6
648
Train F1-score : 0.6705714285714286 - Val F1-score : 0.6648333333333334
Train AUC : 0.7434692324978149 - Val AUC : 0.723075503361857

Epoch 00006: val_acc did not improve from 0.66950
Epoch 7/15

Epoch 00007: LearningRateScheduler reducing learning rate to 0.007310249190777754.
438/438 [==============================] - 2s 4ms/step - loss: 0.6018 - acc: 0.6685 - val_loss: 0.6174 - val_acc: 0.6
662
Train F1-score : 0.6730714285714285 - Val F1-score : 0.6661666666666667
Train AUC : 0.7459041330896337 - Val AUC : 0.7244929213271776

Epoch 00007: val_acc did not improve from 0.66950
Epoch 00007: early stopping
```

```
Out[108… <keras.callbacks.History at 0x7f09a7847150>
```

NOTE :- REFER THE TENSORBOARD RESULTS(screenshots) in the attached pdf file.