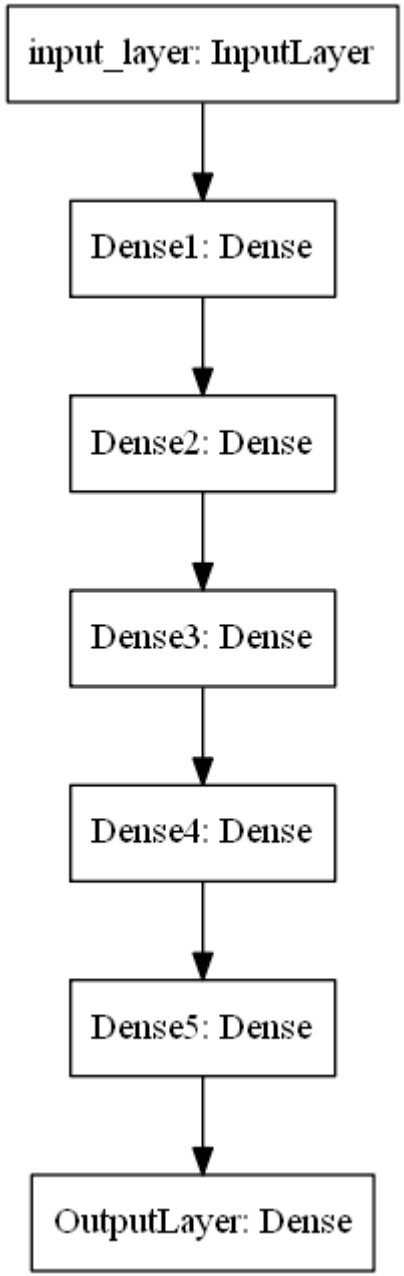
1. Download the data from here

2. Code the model to classify data like below image



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.

In [3]:
```
import numpy as np
import pandas as pd
```

In [4]:
```
from google.colab import drive
drive.mount("/content/gdrive")
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remou
nt=True).
```

In [5]:
```
df_dl = pd.read_csv('/content/gdrive/My Drive/Colab Notebooks/data.csv')
```

In [6]:
```
print(df_dl)
```

```
             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
...         ...       ...    ...
19995 -0.491252 -0.561558    0.0
19996 -0.813124  0.049423    1.0
19997 -0.010594  0.138790    1.0
19998  0.671827  0.804306    0.0
19999 -0.854865 -0.588826    0.0

[20000 rows x 3 columns]
```

In [7]:
```
y = df_dl['label'].values
X = df_dl.drop(['label'],axis = 1)
```

In [8]:
```
print(X.shape)
print(y.shape)
```

```
(20000, 2)
(20000,)
```

In [9]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train , y_test = train_test_split(X,y,test_size = 0.33, stratify = y)
```

In [10]:
```
print(X_train.shape)
print(X_test.shape)
```

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

from keras.utils import np_utils

Y_train = np_utils.to_categorical(y_train, 2)
Y_test = np_utils.to_categorical(y_test, 2)
```

```
(13400, 2)
(6600, 2)
(13400,)
(6600,)
```

In [11]:
```
print(Y_train[0])
```

```
[0. 1.]
```

In [12]:
```
import tensorflow as tf
import tensorflow_addons as tfa
from tensorflow.keras.layers import Dense,Input,Activation
from tensorflow.keras.models import Model

import random as rn
import datetime
import os

from sklearn.metrics import roc_curve, auc, f1_score
from keras.callbacks import Callback
from sklearn.metrics import confusion_matrix, f1_score, precision_score, recall_score
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.callbacks import LearningRateScheduler
```

In [13]:
```
#Metrics class for F1 score calculation
class Metrics(Callback):

    def on_train_begin(self, logs={}):
        self.f1sc=[]
        self.auc_score = []

    def on_epoch_end(self, epoch, logs={}):
          val_predict = (np.argmax(model.predict(X_test), axis=-1))
          val_targ = y_test.astype(int)
```

```
            _val_f1 = f1_score(val_targ, val_predict , average='micro').round(4)
            self.f1sc.append(_val_f1)
            print("\nValidation F1Score : " , _val_f1)

            val_auc_predict = (np.asarray( self.model.predict(X_test)[:,1] ))
            val_auc_target = y_test
            val_fpr , val_tpr , val_thresholds = roc_curve(val_auc_target , val_auc_predict)
            val_auc =auc(val_fpr, val_tpr)
            self.auc_score.append(val_auc)
            print("Validation AUC Score : " , val_auc)




metrics_custom = Metrics()

#Terminate if loss is NAN or Any of the weights are NAN

class TerminateNaN(tf.keras.callbacks.Callback):

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

            loss = logs.get('loss')
            weight = self.model.get_weights()

            cond = False

            for i in weight:
              array_sum = 0.0
              array_sum = np.sum(i)
              if np.isnan(array_sum) == True:
                cond = True
                break

            if loss is not None:
                if np.isnan(loss) or np.isinf(loss) or cond:
                    print("Invalid loss and terminated at epoch {}".format(epoch))
                    self.model.stop_training = True

terminateNaN = TerminateNaN()
```

In [14]:
```
%load_ext tensorboard
```

MODEL 1

In [15]:
```
input_layer = Input(shape=(2,))

layer1 = Dense(32,activation='tanh',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(input_lay
layer2 = Dense(16,activation='tanh',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(layer1)
layer3 = Dense(8,activation='tanh',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(layer2)
layer4 = Dense(4,activation='tanh',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(layer3)
layer5 = Dense(2,activation='tanh',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(layer4)

output = Dense(2,activation='softmax',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(layer5)

#model 1
model = Model(inputs=input_layer,outputs=output)


def scheduler(epoch, lr):

    if (epoch+1) % 3 == 0:
      return lr - (lr * 0.05)
    else:
      return lr

#Decaying Learning rate by 5% on every third epoch
lrschedule = LearningRateScheduler(scheduler, verbose=1)

#Reducing Learning rate by 10% if validation accuracy at current epoch is less than previous epoch
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', factor=0.9,patience=1, min_lr=0.001, mode = 'auto',verbose = 1)

#SGD with momemtum optimizer
optimizer = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)

#compiling model with optimizer and loss function
model.compile(optimizer=optimizer, loss='categorical_crossentropy',metrics=['accuracy'])

#Log dir to save latest logs
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))

#stopping if validation loss has not improved in last 2 epochs
earlystop = EarlyStopping(monitor='val_accuracy', min_delta=0.001, patience=2, verbose=1 , mode = 'max')

#Model checkpoint - Saving after every epoch if val_accuracy is improving
filepath="model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath,save_freq = 'epoch', monitor='val_accuracy',  verbose=1, mode='max')


#tensorboard callback to visualize training
```

```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir,histogram_freq=1, write_graph=True)

#Fit the model
model.fit(X_train,Y_train,epochs=10, validation_data=(X_test,Y_test), batch_size=16, callbacks=[earlystop,metrics_custom,
```

```
Epoch 1/10

Epoch 00001: LearningRateScheduler reducing learning rate to 0.009999999776482582.
  1/838 [..............................] - ETA: 8:33 - loss: 0.6939 - accuracy: 0.5000WARNING:tensorflow:Callback method
`on_train_batch_end` is slow compared to the batch time (batch time: 0.0016s vs `on_train_batch_end` time: 0.0063s). Chec
k your callbacks.
838/838 [==============================] - 2s 2ms/step - loss: 0.6947 - accuracy: 0.5043 - val_loss: 0.6961 - val_accurac
y: 0.5000

Validation F1Score :  0.5
Validation AUC Score :  0.509845684113866

Epoch 00001: saving model to model_save/weights-01-0.5000.hdf5
Epoch 2/10

Epoch 00002: LearningRateScheduler reducing learning rate to 0.009999999776482582.
838/838 [==============================] - 1s 1ms/step - loss: 0.6951 - accuracy: 0.4973 - val_loss: 0.7034 - val_accurac
y: 0.4930

Validation F1Score :  0.493
Validation AUC Score :  0.4919531221303949

Epoch 00002: saving model to model_save/weights-02-0.4930.hdf5

Epoch 00002: ReduceLROnPlateau reducing learning rate to 0.008999999798834325.
Epoch 3/10

Epoch 00003: LearningRateScheduler reducing learning rate to 0.008549999631941318.
838/838 [==============================] - 1s 2ms/step - loss: 0.6953 - accuracy: 0.4923 - val_loss: 0.6948 - val_accurac
y: 0.4921

Validation F1Score :  0.4921
Validation AUC Score :  0.49477387511478416

Epoch 00003: saving model to model_save/weights-03-0.4921.hdf5

Epoch 00003: ReduceLROnPlateau reducing learning rate to 0.007694999501109123.
Epoch 00003: early stopping
```

Out[15]: <tensorflow.python.keras.callbacks.History at 0x7f0c4e0bf7f0>

In [16]:
```
!kill 775
```

```
/bin/bash: line 0: kill: (775) - No such process
```

In [17]:
```
%tensorboard --logdir logs
```

```
Output hidden; open in https://colab.research.google.com to view.
```

MODEL 2

In [18]:
```
input_layer = Input(shape=(2,))

layer1 = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(input_lay
layer2 = Dense(16,activation='relu',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(layer1)
layer3 = Dense(8,activation='relu',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(layer2)
layer4 = Dense(4,activation='relu',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(layer3)
layer5 = Dense(2,activation='relu',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(layer4)

output = Dense(2,activation='softmax',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(layer5)

#model 2
model = Model(inputs=input_layer,outputs=output)


def scheduler(epoch, lr):

    if (epoch+1) % 3 == 0:
      return lr - (lr * 0.05)
    else:
      return lr

#Decaying Learning rate by 5% on every third epoch
lrschedule = LearningRateScheduler(scheduler, verbose=1)

#Reducing Learning rate by 10% if validation accuracy at current epoch is less than previous epoch
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', factor=0.9,patience=1, min_lr=0.001, mode = 'auto',verbose = 1)

#SGD with momemtum optimizer
optimizer = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)

#compiling model with optimizer and loss function
model.compile(optimizer=optimizer, loss='categorical_crossentropy',metrics=['accuracy'])

#log dir to save latest logs
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))

#stopping if validation loss has not improved in last 2 epochs
earlystop = EarlyStopping(monitor='val_accuracy', min_delta=0.001, patience=2, verbose=1 , mode = 'max')
```

```
#Model checkpoint - Saving after every epoch if val_accuracy is improving
filepath="model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath,save_freq = 'epoch', monitor='val_accuracy',  verbose=1, mode='max')


#tensorboard callback to visualize training
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir,histogram_freq=1, write_graph=True)

#Fit the model
model.fit(X_train,Y_train,epochs=10, validation_data=(X_test,Y_test), batch_size=16, callbacks=[earlystop,metrics_custom,
```

```
Epoch 1/10

Epoch 00001: LearningRateScheduler reducing learning rate to 0.009999999776482582.
  3/838 [..............................] - ETA: 25s - loss: 95.0791 - accuracy: 0.6285 WARNING:tensorflow:Callback method
`on_train_batch_end` is slow compared to the batch time (batch time: 0.0023s vs `on_train_batch_end` time: 0.0095s). Chec
k your callbacks.
838/838 [==============================] - 2s 2ms/step - loss: 3.1120 - accuracy: 0.4977 - val_loss: 0.6931 - val_accurac
y: 0.5000

Validation F1Score :  0.5
Validation AUC Score :  0.5

Epoch 00001: saving model to model_save/weights-01-0.5000.hdf5
Epoch 2/10

Epoch 00002: LearningRateScheduler reducing learning rate to 0.009999999776482582.
838/838 [==============================] - 1s 1ms/step - loss: 0.6940 - accuracy: 0.4942 - val_loss: 0.6996 - val_accurac
y: 0.5000

Validation F1Score :  0.5
Validation AUC Score :  0.5

Epoch 00002: saving model to model_save/weights-02-0.5000.hdf5

Epoch 00002: ReduceLROnPlateau reducing learning rate to 0.008999999798834325.
Epoch 3/10

Epoch 00003: LearningRateScheduler reducing learning rate to 0.008549999631941318.
838/838 [==============================] - 1s 1ms/step - loss: 0.6950 - accuracy: 0.4948 - val_loss: 0.6932 - val_accurac
y: 0.5000

Validation F1Score :  0.5
Validation AUC Score :  0.5

Epoch 00003: saving model to model_save/weights-03-0.5000.hdf5
```

```
Epoch 00003: ReduceLROnPlateau reducing learning rate to 0.007694999501109123.
Epoch 00003: early stopping
```

Out[18]:
```
<tensorflow.python.keras.callbacks.History at 0x7f0c47816780>
```

In [19]:
```
!kill 1137
```

```
/bin/bash: line 0: kill: (1137) - No such process
```

In [20]:
```
%tensorboard --logdir logs
```

```
Output hidden; open in https://colab.research.google.com to view.
```

MODEL 3

In [21]:
```
input_layer = Input(shape=(2,))

layer1 = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.he_uniform())(input_layer)
layer2 = Dense(16,activation='relu',kernel_initializer=tf.keras.initializers.he_uniform())(layer1)
layer3 = Dense(8,activation='relu',kernel_initializer=tf.keras.initializers.he_uniform())(layer2)
layer4 = Dense(4,activation='relu',kernel_initializer=tf.keras.initializers.he_uniform())(layer3)
layer5 = Dense(2,activation='relu',kernel_initializer=tf.keras.initializers.he_uniform())(layer4)

output = Dense(2,activation='softmax',kernel_initializer=tf.keras.initializers.he_uniform())(layer5)

#model 3
model = Model(inputs=input_layer,outputs=output)


def scheduler(epoch, lr):

    if (epoch+1) % 3 == 0:
      return lr - (lr * 0.05)
    else:
      return lr

#Decaying Learning rate by 5% on every third epoch
lrschedule = LearningRateScheduler(scheduler, verbose=1)

#Reducing Learning rate by 10% if validation accuracy at current epoch is less than previous epoch
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', factor=0.9,patience=1, min_lr=0.001, mode = 'auto',verbose = 1)

#SGD with momemtum optimizer
optimizer = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)
```

```
#compiling model with optimizer and loss function
model.compile(optimizer=optimizer, loss='categorical_crossentropy',metrics=['accuracy'])

#Log dir to save latest logs
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))

#stopping if validation loss has not improved in last 2 epochs
earlystop = EarlyStopping(monitor='val_accuracy', min_delta=0.001, patience=2, verbose=1 , mode = 'max')

#Model checkpoint - Saving after every epoch if val_accuracy is improving
filepath="model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath,save_freq = 'epoch', monitor='val_accuracy',  verbose=1, mode='max')


#tensorboard callback to visualize training
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir,histogram_freq=1, write_graph=True)

#Fit the model
model.fit(X_train,Y_train,epochs=10, validation_data=(X_test,Y_test), batch_size=16, callbacks=[earlystop,metrics_custom,
```

```
Epoch 1/10

Epoch 00001: LearningRateScheduler reducing learning rate to 0.009999999776482582.
  3/838 [..............................] - ETA: 27s - loss: 0.8244 - accuracy: 0.6319 WARNING:tensorflow:Callback method
`on_train_batch_end` is slow compared to the batch time (batch time: 0.0016s vs `on_train_batch_end` time: 0.0105s). Chec
k your callbacks.
838/838 [==============================] - 2s 2ms/step - loss: 0.6990 - accuracy: 0.4998 - val_loss: 0.6933 - val_accurac
y: 0.5000

Validation F1Score :  0.5
Validation AUC Score :  0.5

Epoch 00001: saving model to model_save/weights-01-0.5000.hdf5
Epoch 2/10

Epoch 00002: LearningRateScheduler reducing learning rate to 0.009999999776482582.
838/838 [==============================] - 1s 2ms/step - loss: 0.6940 - accuracy: 0.4946 - val_loss: 0.6941 - val_accurac
y: 0.5000

Validation F1Score :  0.5
Validation AUC Score :  0.5

Epoch 00002: saving model to model_save/weights-02-0.5000.hdf5

Epoch 00002: ReduceLROnPlateau reducing learning rate to 0.008999999798834325.
Epoch 3/10

Epoch 00003: LearningRateScheduler reducing learning rate to 0.008549999631941318.
```

```
838/838 [==============================] - 1s 2ms/step - loss: 0.6941 - accuracy: 0.4944 - val_loss: 0.6933 - val_accurac
y: 0.5000

Validation F1Score :  0.5
Validation AUC Score :  0.5

Epoch 00003: saving model to model_save/weights-03-0.5000.hdf5

Epoch 00003: ReduceLROnPlateau reducing learning rate to 0.007694999501109123.
Epoch 00003: early stopping
```

Out[21]:
```
<tensorflow.python.keras.callbacks.History at 0x7f0c47698ba8>
```

In [22]:
```
!kill 631
```

```
/bin/bash: line 0: kill: (631) - No such process
```

In [23]:
```
%tensorboard --logdir logs
```

```
Output hidden; open in https://colab.research.google.com to view.
```

MODEL 4

In [24]:
```
input_layer = Input(shape=(2,))

layer1 = Dense(32,activation='selu',kernel_initializer=tf.keras.initializers.GlorotNormal())(input_layer)
layer2 = Dense(16,activation='selu',kernel_initializer=tf.keras.initializers.GlorotNormal())(layer1)
layer3 = Dense(8,activation='selu',kernel_initializer=tf.keras.initializers.GlorotNormal())(layer2)
layer4 = Dense(4,activation='selu',kernel_initializer=tf.keras.initializers.GlorotNormal())(layer3)
layer5 = Dense(2,activation='selu',kernel_initializer=tf.keras.initializers.GlorotNormal())(layer4)

output = Dense(2,activation='softmax',kernel_initializer=tf.keras.initializers.GlorotNormal())(layer5)

#model 3
model = Model(inputs=input_layer,outputs=output)


def scheduler(epoch, lr):

    if (epoch+1) % 3 == 0:
      return lr - (lr * 0.05)
    else:
      return lr

#Decaying Learning rate by 5% on every third epoch
lrschedule = LearningRateScheduler(scheduler, verbose=1)
```

```
#Reducing Learning rate by 10% if validation accuracy at current epoch is less than previous epoch
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', factor=0.9,patience=1, min_lr=0.001, mode = 'auto',verbose = 1)

#SGD with momemtum optimizer
optimizer = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)

#compiling model with optimizer and loss function
model.compile(optimizer=optimizer, loss='categorical_crossentropy',metrics=['accuracy'])

#Log dir to save latest logs
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))

#stopping if validation loss has not improved in last 2 epochs
earlystop = EarlyStopping(monitor='val_accuracy', min_delta=0.001, patience=2, verbose=1 , mode = 'max')

#Model checkpoint - Saving after every epoch if val_accuracy is improving
filepath="model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath,save_freq = 'epoch', monitor='val_accuracy',  verbose=1, mode='max')


#tensorboard callback to visualize training
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir,histogram_freq=1, write_graph=True)

#Fit the model
model.fit(X_train,Y_train,epochs=10, validation_data=(X_test,Y_test), batch_size=16, callbacks=[earlystop,metrics_custom,
```

```
Epoch 1/10

Epoch 00001: LearningRateScheduler reducing learning rate to 0.009999999776482582.
  3/838 [..............................] - ETA: 29s - loss: 0.7500 - accuracy: 0.4306 WARNING:tensorflow:Callback method
`on_train_batch_end` is slow compared to the batch time (batch time: 0.0019s vs `on_train_batch_end` time: 0.0114s). Chec
k your callbacks.
838/838 [==============================] - 2s 2ms/step - loss: 0.6720 - accuracy: 0.5758 - val_loss: 0.6087 - val_accurac
y: 0.6685

Validation F1Score :  0.6685
Validation AUC Score :  0.7350482093663913

Epoch 00001: saving model to model_save/weights-01-0.6685.hdf5
Epoch 2/10

Epoch 00002: LearningRateScheduler reducing learning rate to 0.009999999776482582.
838/838 [==============================] - 1s 2ms/step - loss: 0.6275 - accuracy: 0.6522 - val_loss: 0.6152 - val_accurac
y: 0.6668

Validation F1Score :  0.6668
```

```
Validation AUC Score :  0.7255344811753903

Epoch 00002: saving model to model_save/weights-02-0.6668.hdf5

Epoch 00002: ReduceLROnPlateau reducing learning rate to 0.008999999798834325.
Epoch 3/10

Epoch 00003: LearningRateScheduler reducing learning rate to 0.008549999631941318.
838/838 [==============================] - 1s 1ms/step - loss: 0.6174 - accuracy: 0.6591 - val_loss: 0.6102 - val_accurac
y: 0.6662

Validation F1Score :  0.6662
Validation AUC Score :  0.7299570247933884

Epoch 00003: saving model to model_save/weights-03-0.6662.hdf5

Epoch 00003: ReduceLROnPlateau reducing learning rate to 0.007694999501109123.
Epoch 00003: early stopping
```

Out[24]: <tensorflow.python.keras.callbacks.History at 0x7f0c474a42b0>

In [25]:
```
%tensorboard --logdir logs
```

Output hidden; open in https://colab.research.google.com to view.