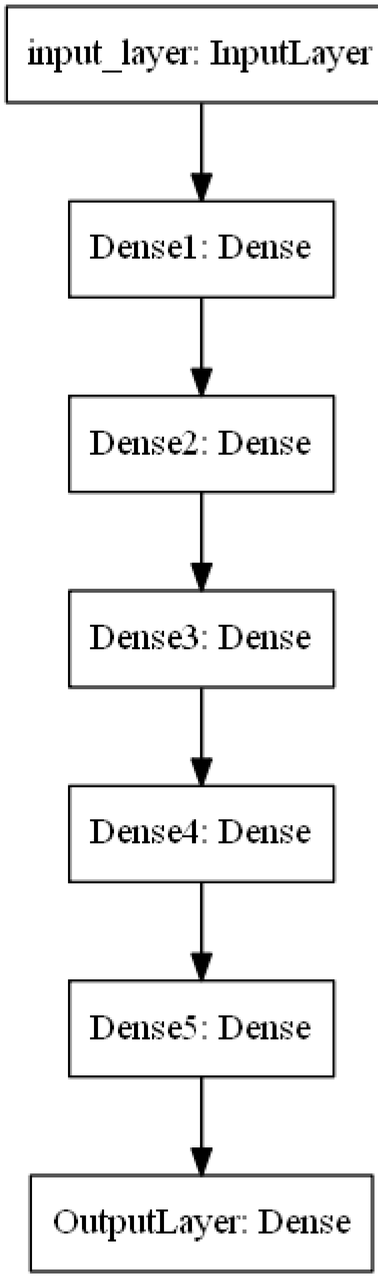
1. Download the data from here. You have to use data.csv file for this assignment

2. Code the model to classify data like below image. You can use any number of units in your Dense layers.



3. Writing Callbacks

You have to implement the following callbacks

- Write your own callback function, that has to print the micro F1 score and AUC score after each epoch.Do not use tf.keras.metrics for calculating AUC and F1 score.
- Save your model at every epoch if your validation accuracy is improved from previous epoch.
- 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%.
- If you are getting any NaN values(either weigths or loss) while training, you have to terminate your training.
- You have to stop the training if your validation accuracy is not increased in last 2 epochs.
- Use tensorboard for every model and analyse your scalar plots and histograms. (you need to upload the screenshots and write the observations for each model for evaluation)

```
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.
```

In [1]:
```
import numpy as np
import tensorflow as tf
import pandas as pd
from tensorflow.keras.layers import Dense,Input,Activation
from tensorflow.keras.models import Model
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, roc_auc_score
from tensorflow.keras.initializers import RandomUniform
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, LearningRate

import os
import datetime
import shutil
```

In [2]:
```
%load_ext tensorboard
try:
    if os.path.isdir("logs"):
        shutil.rmtree("logs")
    if os.path.isdir("model_save"):
        shutil.rmtree("model_save")
except:
    print('Path not present')
```

In [3]:
```
data = pd.read_csv('data.csv')
data.head()
```

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]:
```
# We need variables with numpy nd array datatype or tensor datatype to do tensor op
y = data.label.values
x = data[['f1','f2']].values
```

In [5]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_s
```

In [6]:
```
Y_train = tf.keras.utils.to_categorical(y_train, 2)
Y_test = tf.keras.utils.to_categorical(y_test, 2)
```

In [7]:
```
print(x_train.shape)
print(x_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(16000, 2)
(4000, 2)
(16000, 2)
(4000, 2)
```

3.1 Callbacks

In [8]:
```
class F1_score_auc_score(tf.keras.callbacks.Callback):
    def __init__(self, validation_data):
        self.x_test = validation_data[0]
        self.y_test = validation_data[1]
        self.log_dir = "logs/fits/"+datetime.datetime.now().strftime("%Y%m%d-%H%M%S

    def on_train_begin(self, logs={}):
        self.additional_metrics= {'f1_score':[],'AUC':[]}

    def on_epoch_end(self, epoch, logs={}):
        y_pred= np.argmax(self.model.predict(self.x_test),axis=1)
        f1_val = f1_score(y_test, y_pred, average = 'micro')
        auc_val = roc_auc_score(y_test, y_pred)
        self.additional_metrics['f1_score'].append(f1_val)
        self.additional_metrics['AUC'].append(auc_val)
        print('F1_score: ',f1_val,'AUC: ',auc_val)
```

In [9]:
```
#https://stackoverflow.com/questions/64806541/performing-np-isnan-on-keras-model-we
# Most of the code below is copied from Call_Backs_Reference.ipynb notebook

class TerminateNaN(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        loss = logs.get('loss')
        nan_weights = np.any([np.any(np.isnan(x)) for x in self.model.get_weights()
        inf_weights = np.any([np.any(np.isinf(x)) for x in self.model.get_weights()
        if loss is not None:
            if np.isnan(loss) or np.isinf(loss) or nan_weights or inf_weights:
                print("Invalid loss and terminated at epoch {}".format(epoch))
                self.model.stop_training = True
```

In [10]:
```
def changeLearningRate(epoch, learning_rate):
    changed = learning_rate
    if epoch % 3 == 0:
        changed = learning_rate * 0.95

    return changed
```

In [11]:
```
f1_auc = F1_score_auc_score(validation_data = [x_test,Y_test])

filepath="model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy',  verbose=1,

earlystop = EarlyStopping(monitor='val_accuracy', min_delta=0.01, patience=2, verbo

lrschedule = LearningRateScheduler(changeLearningRate, verbose=1)

reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', factor=0.9,
                              patience=1, min_lr=0.001)  # SGD has default learning

terminateNAN = TerminateNaN()

#log_dir = os.path.join("logs",'fits', datetime.datetime.now().strftime("%Y%m%d-%H%
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir= ("logs/fits/"+dateti
                       ,histogram_freq=1,write_graph=True)

call_back_list = [f1_auc, lrschedule, reduce_lr, terminateNAN, earlystop, checkpoin
```

3.2 Model 1 - tanh activation, SGD with momentum optimizer and RandomUniform initializer

In [12]:
```
input_layer = Input(shape=(2,))
initializer = RandomUniform(minval=0, maxval=1, seed=0)
layer1 = Dense(4,activation='tanh',kernel_initializer= initializer)(input_layer)
layer2 = Dense(8,activation='tanh',kernel_initializer= initializer)(layer1)
layer3 = Dense(16,activation='tanh',kernel_initializer= initializer)(layer2)
layer4 = Dense(32,activation='tanh',kernel_initializer= initializer)(layer3)
layer5 = Dense(16,activation='tanh',kernel_initializer= initializer)(layer4)
output = Dense(2,activation='softmax',kernel_initializer= initializer)(layer5)

model_one = Model(inputs=input_layer,outputs=output)
model_one.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 2)]               0
_________________________________________________________________
dense (Dense)                (None, 4)                 12
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 40
_________________________________________________________________
dense_2 (Dense)              (None, 16)                144
_________________________________________________________________
dense_3 (Dense)              (None, 32)                544
_________________________________________________________________
dense_4 (Dense)              (None, 16)                528
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 34
=================================================================
Total params: 1,302
Trainable params: 1,302
Non-trainable params: 0
_________________________________________________________________
```

In [13]:
```
optim_1 = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)

model_one.compile(optimizer=optim_1, loss='categorical_crossentropy',metrics=['accu

model_one.fit(x_train,Y_train,epochs=10, validation_data=(x_test,Y_test), batch_siz
Epoch 1/10

Epoch 00001: LearningRateScheduler setting learning rate to 0.009499999787658453.
 4/32 [==>...........................] - ETA: 1s - loss: 0.6937 - accuracy: 0.5044
WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the ba
tch time (batch time: 0.0034s vs `on_train_batch_end` time: 0.0200s). Check your c
allbacks.
32/32 [==============================] - 2s 15ms/step - loss: 0.6932 - accuracy: 
0.5083 - val_loss: 0.6933 - val_accuracy: 0.5033
F1_score:  0.50325 AUC:  0.5032500000000001

Epoch 00001: val_accuracy improved from -inf to 0.50325, saving model to model_sav
e\weights-01-0.5033.hdf5
Epoch 2/10

Epoch 00002: LearningRateScheduler setting learning rate to 0.009499999694526196.
32/32 [==============================] - 0s 6ms/step - loss: 0.6935 - accuracy: 0.
4984 - val_loss: 0.6935 - val_accuracy: 0.5033
F1_score:  0.50325 AUC:  0.50325

Epoch 00002: val_accuracy did not improve from 0.50325
Epoch 3/10

Epoch 00003: LearningRateScheduler setting learning rate to 0.008549999445676804.
32/32 [==============================] - 0s 6ms/step - loss: 0.6935 - accuracy: 0.
4930 - val_loss: 0.6933 - val_accuracy: 0.4975
F1_score:  0.4975 AUC:  0.49749999999999994

Epoch 00003: val_accuracy did not improve from 0.50325
Epoch 00003: early stopping
```

Out[13]:
```
<keras.callbacks.History at 0x1db93e85be0>
```

In [14]:
```
%tensorboard --logdir logs/fits
```

3.3 Model 2 - relu activation, SGD with momentum optimizer and RandomUniform initializer

```
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.

In [15]:
```
input_layer = Input(shape=(2,))
initializer = RandomUniform(minval=0, maxval=1, seed=0)
layer1 = Dense(4,activation='relu',kernel_initializer=initializer)(input_layer)
layer2 = Dense(8,activation='relu',kernel_initializer=initializer)(layer1)
layer3 = Dense(16,activation='relu',kernel_initializer=initializer)(layer2)
layer4 = Dense(32,activation='relu',kernel_initializer=initializer)(layer3)
layer5 = Dense(16,activation='relu',kernel_initializer=initializer)(layer4)
output = Dense(2,activation='softmax',kernel_initializer=initializer)(layer5)

model_two = Model(inputs=input_layer,outputs=output)
model_two.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 2)]               0
_________________________________________________________________
dense_6 (Dense)              (None, 4)                 12
_________________________________________________________________
dense_7 (Dense)              (None, 8)                 40
_________________________________________________________________
dense_8 (Dense)              (None, 16)                144
_________________________________________________________________
dense_9 (Dense)              (None, 32)                544
_________________________________________________________________
dense_10 (Dense)             (None, 16)                528
_________________________________________________________________
dense_11 (Dense)             (None, 2)                 34
=================================================================
Total params: 1,302
Trainable params: 1,302
Non-trainable params: 0
_________________________________________________________________
```

In [16]:
```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir= ("logs/fits/"+dateti
                       ,histogram_freq=1,write_graph=True)
call_back_list = [f1_auc, lrschedule, reduce_lr, terminateNAN, earlystop, checkpoin

optim_2 = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)
model_two.compile(optimizer=optim_2, loss='categorical_crossentropy',metrics=['accu

model_two.fit(x_train,Y_train,epochs=10, validation_data=(x_test,Y_test), batch_siz
Epoch 1/10

Epoch 00001: LearningRateScheduler setting learning rate to 0.009499999787658453.
 3/32 [=>............................] - ETA: 9s - loss: 801.0432 - accuracy: 0.50
98 WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the
batch time (batch time: 0.0058s vs `on_train_batch_end` time: 0.1051s). Check your
callbacks.
32/32 [==============================] - 1s 32ms/step - loss: 77.5268 - accuracy:
0.5017 - val_loss: 0.6932 - val_accuracy: 0.5000
F1_score:  0.5 AUC:  0.5

Epoch 00001: val_accuracy did not improve from 0.50325
Epoch 2/10

Epoch 00002: LearningRateScheduler setting learning rate to 0.009499999694526196.
32/32 [==============================] - 0s 5ms/step - loss: 0.6932 - accuracy: 0.
4908 - val_loss: 0.6931 - val_accuracy: 0.5000
F1_score:  0.5 AUC:  0.5

Epoch 00002: val_accuracy did not improve from 0.50325
Epoch 3/10

Epoch 00003: LearningRateScheduler setting learning rate to 0.008549999445676804.
32/32 [==============================] - 0s 6ms/step - loss: 0.6933 - accuracy: 0.
5000 - val_loss: 0.6932 - val_accuracy: 0.5000
F1_score:  0.5 AUC:  0.5

Epoch 00003: val_accuracy did not improve from 0.50325
Epoch 00003: early stopping
```

Out[16]:
```
<keras.callbacks.History at 0x1db8dd35af0>
```

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

```
Reusing TensorBoard on port 6006 (pid 2352), started 0:03:02 ago. (Use '!kill 235
2' to kill it.)
```

3.4 Model 3 - relu activation, SGD with momentum optimizer and he_uniform initializer

```
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.

In [18]:
```
initializer = tf.keras.initializers.HeUniform(seed = 0)

input_layer = Input(shape=(2,))
layer1 = Dense(4,activation='relu',kernel_initializer=initializer)(input_layer)
layer2 = Dense(8,activation='relu',kernel_initializer=initializer)(layer1)
layer3 = Dense(16,activation='relu',kernel_initializer=initializer)(layer2)
layer4 = Dense(32,activation='relu',kernel_initializer=initializer)(layer3)
layer5 = Dense(16,activation='relu',kernel_initializer=initializer)(layer4)
output = Dense(2,activation='softmax',kernel_initializer=initializer)(layer5)

model_three = Model(inputs=input_layer,outputs=output)
model_three.summary()
```

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 2)]               0
_________________________________________________________________
dense_12 (Dense)             (None, 4)                 12
_________________________________________________________________
dense_13 (Dense)             (None, 8)                 40
_________________________________________________________________
dense_14 (Dense)             (None, 16)                144
_________________________________________________________________
dense_15 (Dense)             (None, 32)                544
_________________________________________________________________
dense_16 (Dense)             (None, 16)                528
_________________________________________________________________
dense_17 (Dense)             (None, 2)                 34
=================================================================
Total params: 1,302
Trainable params: 1,302
Non-trainable params: 0
_________________________________________________________________
```

In [19]:
```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir= ("logs/fits/"+dateti
                       ,histogram_freq=1,write_graph=True)
call_back_list = [f1_auc, lrschedule, reduce_lr, terminateNAN, earlystop, checkpoin

optim_3 = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)
model_three.compile(optimizer= optim_3, loss='categorical_crossentropy',metrics=['a

model_three.fit(x_train,Y_train,epochs=10, validation_data=(x_test,Y_test), batch_s
Epoch 1/10

Epoch 00001: LearningRateScheduler setting learning rate to 0.009499999787658453.
 5/32 [===>..........................] - ETA: 4s - loss: 0.6907 - accuracy: 0.4980
WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the ba
tch time (batch time: 0.0148s vs `on_train_batch_end` time: 0.1101s). Check your c
allbacks.
32/32 [==============================] - 1s 34ms/step - loss: 0.6854 - accuracy:
0.5378 - val_loss: 0.6782 - val_accuracy: 0.5972
F1_score:  0.59725 AUC:  0.5972500000000001

Epoch 00001: val_accuracy improved from 0.50325 to 0.59725, saving model to model_
save\weights-01-0.5972.hdf5
Epoch 2/10

Epoch 00002: LearningRateScheduler setting learning rate to 0.009499999694526196.
32/32 [==============================] - 0s 4ms/step - loss: 0.6727 - accuracy: 0.
6011 - val_loss: 0.6656 - val_accuracy: 0.6050
F1_score:  0.605 AUC:  0.605

Epoch 00002: val_accuracy improved from 0.59725 to 0.60500, saving model to model_
save\weights-02-0.6050.hdf5
Epoch 3/10

Epoch 00003: LearningRateScheduler setting learning rate to 0.009499999694526196.
32/32 [==============================] - 0s 4ms/step - loss: 0.6610 - accuracy: 0.
6219 - val_loss: 0.6546 - val_accuracy: 0.6277
F1_score:  0.62775 AUC:  0.6277499999999999

Epoch 00003: val_accuracy improved from 0.60500 to 0.62775, saving model to model_
save\weights-03-0.6277.hdf5
Epoch 4/10

Epoch 00004: LearningRateScheduler setting learning rate to 0.009024999709799886.
32/32 [==============================] - 0s 4ms/step - loss: 0.6518 - accuracy: 0.
6254 - val_loss: 0.6454 - val_accuracy: 0.6457
F1_score:  0.64575 AUC:  0.64575

Epoch 00004: val_accuracy improved from 0.62775 to 0.64575, saving model to model_
save\weights-04-0.6457.hdf5
Epoch 5/10

Epoch 00005: LearningRateScheduler setting learning rate to 0.009025000035762787.
32/32 [==============================] - 0s 5ms/step - loss: 0.6441 - accuracy: 0.
6332 - val_loss: 0.6390 - val_accuracy: 0.6495
F1_score:  0.6495 AUC:  0.6495

Epoch 00005: val_accuracy improved from 0.64575 to 0.64950, saving model to model_
save\weights-05-0.6495.hdf5
Epoch 6/10

Epoch 00006: LearningRateScheduler setting learning rate to 0.009025000035762787.
32/32 [==============================] - 0s 4ms/step - loss: 0.6372 - accuracy: 0.
6401 - val_loss: 0.6334 - val_accuracy: 0.6535
F1_score:  0.6535 AUC:  0.6535
```

```
Epoch 00006: val_accuracy improved from 0.64950 to 0.65350, saving model to model_
save\weights-06-0.6535.hdf5
Epoch 00006: early stopping
```

Out[19]:
```
<keras.callbacks.History at 0x1dc6f249d60>
```

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

```
Reusing TensorBoard on port 6006 (pid 2352), started 0:04:30 ago. (Use '!kill 235
2' to kill it.)
```

3.5 Model 4 - relu activation, Adam optimizer and he_uniform initializer

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

In [21]:
```
initializer = tf.keras.initializers.HeUniform(seed = 0)

input_layer = Input(shape=(2,))
layer1 = Dense(4,activation='relu',kernel_initializer=initializer)(input_layer)
layer2 = Dense(8,activation='relu',kernel_initializer=initializer)(layer1)
layer3 = Dense(16,activation='relu',kernel_initializer=initializer)(layer2)
layer4 = Dense(32,activation='relu',kernel_initializer=initializer)(layer3)
layer5 = Dense(16,activation='relu',kernel_initializer=initializer)(layer4)
output = Dense(2,activation='softmax',kernel_initializer=initializer)(layer5)

model_four = Model(inputs=input_layer,outputs=output)
model_four.summary()
```

```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 2)]               0
_________________________________________________________________
dense_18 (Dense)             (None, 4)                 12
_________________________________________________________________
dense_19 (Dense)             (None, 8)                 40
_________________________________________________________________
dense_20 (Dense)             (None, 16)                144
_________________________________________________________________
dense_21 (Dense)             (None, 32)                544
_________________________________________________________________
dense_22 (Dense)             (None, 16)                528
_________________________________________________________________
dense_23 (Dense)             (None, 2)                 34
=================================================================
Total params: 1,302
Trainable params: 1,302
Non-trainable params: 0
_________________________________________________________________
```

In [22]:
```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir= ("logs/fits/"+dateti
                       ,histogram_freq=1,write_graph=True)
call_back_list = [f1_auc, lrschedule, reduce_lr, terminateNAN, earlystop, checkpoin

optim_4 =  tf.keras.optimizers.Adam()
model_four.compile(optimizer= optim_4, loss='categorical_crossentropy',metrics=['ac

model_four.fit(x_train,Y_train,epochs=10, validation_data=(x_test,Y_test), batch_si
Epoch 1/10

Epoch 00001: LearningRateScheduler setting learning rate to 0.0009500000451225787.
 3/32 [=>...........................] - ETA: 10s - loss: 0.6904 - accuracy: 0.495
4WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the b
atch time (batch time: 0.0048s vs `on_train_batch_end` time: 0.1201s). Check your
callbacks.
32/32 [==============================] - 1s 34ms/step - loss: 0.6843 - accuracy:
0.5264 - val_loss: 0.6771 - val_accuracy: 0.5623
F1_score:  0.56225 AUC:  0.5622499999999999

Epoch 00001: val_accuracy did not improve from 0.65350
Epoch 2/10

Epoch 00002: LearningRateScheduler setting learning rate to 0.0009500000160187483.
32/32 [==============================] - 0s 4ms/step - loss: 0.6705 - accuracy: 0.
5949 - val_loss: 0.6621 - val_accuracy: 0.6240
F1_score:  0.624 AUC:  0.624

Epoch 00002: val_accuracy did not improve from 0.65350
Epoch 3/10

Epoch 00003: LearningRateScheduler setting learning rate to 0.0009500000160187483.
32/32 [==============================] - 0s 4ms/step - loss: 0.6558 - accuracy: 0.
6292 - val_loss: 0.6468 - val_accuracy: 0.6382
F1_score:  0.63825 AUC:  0.63825

Epoch 00003: val_accuracy did not improve from 0.65350
Epoch 4/10

Epoch 00004: LearningRateScheduler setting learning rate to 0.0009025000152178108.
32/32 [==============================] - 0s 4ms/step - loss: 0.6419 - accuracy: 0.
6384 - val_loss: 0.6353 - val_accuracy: 0.6432
F1_score:  0.64325 AUC:  0.64325

Epoch 00004: val_accuracy did not improve from 0.65350
Epoch 5/10

Epoch 00005: LearningRateScheduler setting learning rate to 0.0009025000035762787.
32/32 [==============================] - 0s 4ms/step - loss: 0.6308 - accuracy: 0.
6428 - val_loss: 0.6278 - val_accuracy: 0.6562
F1_score:  0.65625 AUC:  0.65625

Epoch 00005: val_accuracy improved from 0.65350 to 0.65625, saving model to model_
save\weights-05-0.6562.hdf5
Epoch 6/10

Epoch 00006: LearningRateScheduler setting learning rate to 0.0009025000035762787.
32/32 [==============================] - 0s 6ms/step - loss: 0.6211 - accuracy: 0.
6514 - val_loss: 0.6159 - val_accuracy: 0.6618
F1_score:  0.66175 AUC:  0.66175

Epoch 00006: val_accuracy improved from 0.65625 to 0.66175, saving model to model_
save\weights-06-0.6618.hdf5
Epoch 7/10
Epoch 00007: LearningRateScheduler setting learning rate to 0.0008573750033974647.
32/32 [==============================] - 0s 4ms/step - loss: 0.6140 - accuracy: 0.
6596 - val_loss: 0.6114 - val_accuracy: 0.6662
F1_score:  0.66625 AUC:  0.66625

Epoch 00007: val_accuracy improved from 0.66175 to 0.66625, saving model to model_
save\weights-07-0.6662.hdf5
Epoch 00007: early stopping
```

```
Out[22]: <keras.callbacks.History at 0x1dba87a5ca0>
```

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

```
Reusing TensorBoard on port 6006 (pid 2352), started 0:08:29 ago. (Use '!kill 235
2' to kill it.)
```

Note

Make sure that you are plotting tensorboard plots either in your notebook or you can try to create a pdf file with all the tensorboard screenshots.Please write your analysis of tensorboard results for each model.

Observations:

1. model_one and model_two that used sgd optimizer and Random uniform initializer stopped training at low values of epoch = 3 as there was no improvement in validation accuracy. These models could have been stuck at a saddle point.
2. model_three and model_four which used Relu activation function and he initializer were able to continue training till epoch = 6 and epoch = 7 after which there was no improvement in validation accuracy.
3. model_two with relu activation function and Random uniform initializer had the highest initial loss of 77.5268 after 1 epoch. This might be because Random uniform initializer does not work well with relu activation function.
4. model_three used relu activation function with he uniform initializer which had a initial loss of 0.6854 after 1 epoch which shows that he initializer works well with relu activation function.
5. The accuracy of model_one that used tanh activation function, sgd optimizer and Random uniform initializer started decreasing at epoch = 3.
6. model_four using relu activation function, adam optimizer and he uniform initializer had the lowest validation loss of 0.6109, highest validation accuracy of 0.669 out of all 4 models.