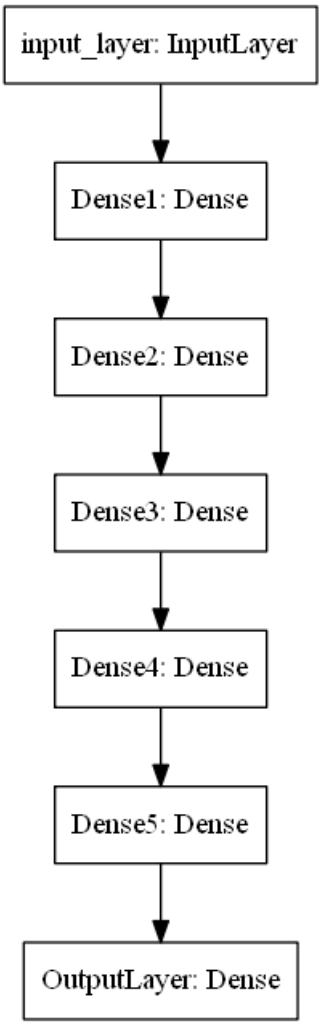
**1. Download the data from here. You have to use data.csv file for this assignment**

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




In [1]:

```
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
```

In [2]:

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

Mounted at /content/drive

In [3]:

```
data=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Callbacks/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 |

| | f1 | f2 | label |
|---|---|---|---|

In [4]:

```
y = data['label'].values
X = data.drop(['label'], axis=1)
```

In [5]:

```
# train test split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20,stratify=y, random_state=42)
```

In [6]:

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

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

In [7]:

```
!sudo pip3 install keras
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: keras in /usr/local/lib/python3.7/dist-packages (2.8.0)
```

In [8]:

```
import tensorflow as tf

from tensorflow.keras.layers import Dense,Input,Activation
from tensorflow.keras.models import Model
import random as rn
import os
import datetime
```

# 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)

In [10]:

```
from sklearn.metrics import f1_score,roc_auc_score

class CalculateMetrics(tf.keras.callbacks.Callback):
    def __init__(self,validation_data):
      self.x_test = validation_data[0]
      self.y_test= validation_data[1]
    def on_train_begin(self, logs={}):
        ## on begin of training, we are creating a instance varible called history
        ## it is a dict with keys [loss, acc, val_loss, val_acc]
        self.history={'loss': [],'accuracy': [],'val_loss': [],'val_accuracy': [],'val_f1score': [],'val_auc': []}
```

```
    self.history={'loss': [], 'accuracy': [], 'val_loss': [], 'val_accuracy': [], 'val_f1score': [], 'val_auc': []}

  def on_epoch_end(self, epoch, logs={}):
    ## on end of each epoch, we will get logs and update the self.history dict
    self.history['loss'].append(logs.get('loss'))
    self.history['accuracy'].append(logs.get('accuracy'))

    if logs.get('val_loss', -1) != -1:
      self.history['val_loss'].append(logs.get('val_loss'))
    if logs.get('val_accuracy', -1) != -1:
      self.history['val_accuracy'].append(logs.get('val_accuracy'))

    y_pred= self.model.predict(self.x_test)
    #we can also calcualte predefined metrics using callbacks
    y_pred_label=[1 if x>=0.5 else 0 for x in y_pred]
    f1score=f1_score(self.y_test, y_pred_label, average='micro')
    #auc=roc_auc_score(self.y_test, y_pred_label)
    auc=roc_auc_score(self.y_test, self.model.predict(self.x_test))
    self.history['val_f1score'].append(f1score)
    self.history['val_auc'].append(auc)
    print('F1_Score: ',f1score,'AUC: ',auc)

metrics=CalculateMetrics(validation_data=[X_test,y_test])
```

In [11]:

```
from tensorflow.keras.callbacks import ModelCheckpoint
```

**Decay learning rate based on epoch number**

In [12]:

```
from tensorflow.keras.callbacks import LearningRateScheduler

def changeLearningRate(epoch,lr):
  #For every 3rd epoch, decay learning rate by 5%
  if (epoch+1)%3==0:
    lr = lr*(1-0.05)
  return lr
```

In [13]:

```
from tensorflow.keras.callbacks import ReduceLROnPlateau
```

In [14]:

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

  def on_epoch_end(self, epoch, logs={}):
    loss = logs.get('loss')
    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

    model_weights = self.model.get_weights()
    if model_weights is not None:
      if np.any([np.any(np.isnan(x)) for x in model_weights]):
        print("Invalid weights and terminated at epoch {}".format(epoch))
        self.model.stop_training = True
```

In [15]:

```
from tensorflow.keras.callbacks import EarlyStopping
```

**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 [16]:

```
# Load the TensorBoard notebook extension
%load_ext tensorboard
```

In [17]:

```
#Input layer
input_layer = Input(shape=(2,))
#Dense hidden layer 1
layer1 = Dense(32,activation='tanh',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(input_layer)
#Dense hidden layer 2
layer2 = Dense(32,activation='tanh',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(layer1)
#Dense hidden layer 3
layer3 = Dense(32,activation='tanh',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(layer2)
#Dense hidden layer 4
layer4 = Dense(32,activation='tanh',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(layer3)
#Dense hidden layer 5
layer5 = Dense(32,activation='tanh',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(layer4)
#output layer
output = Dense(1,activation='sigmoid',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(layer5)
#Creating a model
model1 = Model(inputs=input_layer,outputs=output)
```

In [18]:

```
metrics=CalculateMetrics(validation_data=[X_test,y_test])

#Callbacks
#file path, it saves the model in the 'model_save' folder and we are naming model with epoch number
#and val acc to differtiate with other models
#you have to create model_save folder before running the code.
filepath="model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_loss',  verbose=1, save_best_only=True, mode='auto')
lrschedule = LearningRateScheduler(changeLearningRate, verbose=1)
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', factor=0.1,patience=2, min_lr=0.001)
terminate_nan=TerminateNaN()

#you can monitor any quantity (here i am monitoring val_loss), you can give any number for patience based on your need.
#i am terminating training if my validation loss incresing at once than previous loss. so maintained 1. you can give min delta
#an absolute change of less than min_delta, will count as no improvement. i maintained 0.35 because i don't want to run
#so many epoch to see termination
earlystop = EarlyStopping(monitor='val_accuracy', min_delta=0.35, patience=2, verbose=1)


optimizer=tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9, nesterov=False, name="SGD")

model1.compile(optimizer=optimizer,loss='binary_crossentropy',metrics=['accuracy'])


logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1, write_graph=True, write_grads=True)

# here we are creating a list with all the callbacks we want
callback_list = [metrics,checkpoint,lrschedule,reduce_lr, terminate_nan,earlystop,tensorboard_callback ]

model1.fit(X_train,y_train,epochs=10,validation_data=(X_test,y_test),callbacks=callback_list)
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.

Epoch 1: LearningRateScheduler setting learning rate to 0.009999999776482582.
Epoch 1/10
  1/500 [..............................] - ETA: 7:02 - loss: 10.3637 - accuracy: 0.4062

WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0021s vs `on_train_batch_end` time: 0.0034s). Check your callbacks.

483/500 [===========================>..] - ETA: 0s - loss: 0.9256 - accuracy: 0.5230F1_Score:  0.5125 AUC:  0.512252

Epoch 1: val_loss improved from inf to 0.68998, saving model to model_save/weights-01-0.5125.hdf5
500/500 [==============================] - 3s 4ms/step - loss: 0.9176 - accuracy: 0.5228 - val_loss: 0.6900 - val_accuracy: 0.5125 - lr: 0.01
00

Epoch 2: LearningRateScheduler setting learning rate to 0.009999999776482582.
Epoch 2/10
477/500 [===========================>..] - ETA: 0s - loss: 0.6968 - accuracy: 0.5043F1_Score:  0.5 AUC:  0.50775

Epoch 2: val_loss did not improve from 0.68998
```

```
500/500 [==============================] - 1s 3ms/step - loss: 0.6964 - accuracy: 0.5056 - val_loss: 0.6947 - val_accuracy: 0.5000 - lr: 0.01
00

Epoch 3: LearningRateScheduler setting learning rate to 0.009499999787658453.
Epoch 3/10
497/500 [============================>.] - ETA: 0s - loss: 0.7000 - accuracy: 0.5020F1_Score:  0.5 AUC:  0.5

Epoch 3: val_loss did not improve from 0.68998
500/500 [==============================] - 1s 3ms/step - loss: 0.7001 - accuracy: 0.5016 - val_loss: 0.6943 - val_accuracy: 0.5000 - lr: 0.00
95
Epoch 3: early stopping
```

Out[18]:

```
<keras.callbacks.History at 0x7f6e35a00bd0>
```

In [19]:

```
%tensorboard --logdir logs
```

In [20]:

```
# Clear any logs from previous runs
!rm -rf ./logs/
```

**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 [21]:

```
#Input layer
input_layer = Input(shape=(2,))
#Dense hidden layer 1
layer1 = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(input_layer)
#Dense hidden layer 2
layer2 = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(layer1)
#Dense hidden layer 3
layer3 = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(layer2)
#Dense hidden layer 4
layer4 = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(layer3)
#Dense hidden layer 5
layer5 = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(layer4)
#output layer
output = Dense(1,activation='sigmoid',kernel_initializer=tf.keras.initializers.RandomUniform(minval=0, maxval=1))(layer5)
#Creating a model
model1 = Model(inputs=input_layer,outputs=output)
```

In [22]:

```
metrics=CalculateMetrics(validation_data=[X_test,y_test])

#Callbacks
#file path, it saves the model in the 'model_save' folder and we are naming model with epoch number
#and val acc to differtiate with other models
#you have to create model_save folder before running the code.
filepath="model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_loss',  verbose=1, save_best_only=True, mode='auto')
lrschedule = LearningRateScheduler(changeLearningRate, verbose=1)
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', factor=0.1,patience=2, min_lr=0.001)
terminate_nan=TerminateNaN()

#you can monitor any quantity (here i am monitoring val_loss), you can give any number for patience based on your need.
#i am terminating training if my validation loss incresing at once than previous loss. so maintained 1. you can give min delta
#an absolute change of less than min_delta, will count as no improvement. i maintained 0.35 because i don't want to run
#so many epoch to see termination
earlystop = EarlyStopping(monitor='val_accuracy', min_delta=0.35, patience=2, verbose=1)


optimizer=tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9, nesterov=False, name="SGD")
```

```
model1.compile(optimizer=optimizer,loss='binary_crossentropy',metrics=['accuracy'])


logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1, write_graph=True, write_grads=True)

# here we are creating a list with all the callbacks we want
callback_list = [metrics,checkpoint,lrschedule,reduce_lr, terminate_nan,earlystop,tensorboard_callback ]

model1.fit(X_train,y_train,epochs=10,validation_data=(X_test,y_test),callbacks=callback_list)
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.
```

```
Epoch 1: LearningRateScheduler setting learning rate to 0.009999999776482582.
Epoch 1/10
  1/500 [..............................] - ETA: 4:30 - loss: 131043.6562 - accuracy: 0.5312
```

```
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: 0.0028s). Check your callbacks.
```

```
496/500 [============================>.] - ETA: 0s - loss: 264.8904 - accuracy: 0.4996F1_Score:  0.5 AUC:  0.5

Epoch 1: val_loss improved from inf to 0.69315, saving model to model_save/weights-01-0.5000.hdf5
500/500 [==============================] - 2s 4ms/step - loss: 262.7768 - accuracy: 0.4992 - val_loss: 0.6931 - val_accuracy: 0.5000 - lr: 0.0100

Epoch 2: LearningRateScheduler setting learning rate to 0.009999999776482582.
Epoch 2/10
471/500 [===========================>..] - ETA: 0s - loss: 0.6934 - accuracy: 0.4922F1_Score:  0.5 AUC:  0.5

Epoch 2: val_loss did not improve from 0.69315
500/500 [==============================] - 2s 3ms/step - loss: 0.6934 - accuracy: 0.4941 - val_loss: 0.6934 - val_accuracy: 0.5000 - lr: 0.0100

Epoch 3: LearningRateScheduler setting learning rate to 0.009049999787658453.
Epoch 3/10
483/500 [===========================>..] - ETA: 0s - loss: 0.6935 - accuracy: 0.4970F1_Score:  0.5 AUC:  0.5

Epoch 3: val_loss did not improve from 0.69315
500/500 [==============================] - 2s 3ms/step - loss: 0.6935 - accuracy: 0.4963 - val_loss: 0.6931 - val_accuracy: 0.5000 - lr: 0.0095
Epoch 3: early stopping
```

Out[22]:

```
<keras.callbacks.History at 0x7f6e32190ed0>
```

In [23]:

```
%tensorboard --logdir logs
```

```
Reusing TensorBoard on port 6006 (pid 433), started 0:27:59 ago. (Use '!kill 433' to kill it.)
```

In [24]:

```
# Clear any logs from previous runs
!rm -rf ./logs/
```

**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 [25]:

```
#Input layer
input_layer = Input(shape=(2,))
#Dense hidden layer 1
```

```
layer1 = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.HeUniform())(input_layer)
#Dense hidden layer 2
layer2 = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.HeUniform())(layer1)
#Dense hidden layer 3
layer3 = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.HeUniform())(layer2)
#Dense hidden layer 4
layer4 = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.HeUniform())(layer3)
#Dense hidden layer 5
layer5 = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.HeUniform())(layer4)
#output layer
output = Dense(1,activation='sigmoid',kernel_initializer=tf.keras.initializers.HeUniform())(layer5)
#Creating a model
model1 = Model(inputs=input_layer,outputs=output)
```

In [26]:

```
metrics=CalculateMetrics(validation_data=[X_test,y_test])

#Callbacks
#file path, it saves the model in the 'model_save' folder and we are naming model with epoch number
#and val acc to differtiate with other models
#you have to create model_save folder before running the code.
filepath="model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_loss',  verbose=1, save_best_only=True, mode='auto')
lrschedule = LearningRateScheduler(changeLearningRate, verbose=1)
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', factor=0.1,patience=2, min_lr=0.001)
terminate_nan=TerminateNaN()

#you can monitor any quantity (here i am monitoring val_loss), you can give any number for patience based on your need.
#i am terminating training if my validation loss incresing at once than previous loss. so maintained 1. you can give min delta
#an absolute change of less than min_delta, will count as no improvement. i maintained 0.35 because i don't want to run
#so many epoch to see termination
earlystop = EarlyStopping(monitor='val_accuracy', min_delta=0.35, patience=2, verbose=1)


optimizer=tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9, nesterov=False, name="SGD")

model1.compile(optimizer=optimizer,loss='binary_crossentropy',metrics=['accuracy'])


logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1, write_graph=True, write_grads=True)

# here we are creating a list with all the callbacks we want
callback_list = [metrics,checkpoint,lrschedule,reduce_lr, terminate_nan,earlystop,tensorboard_callback ]

model1.fit(X_train,y_train,epochs=10,validation_data=(X_test,y_test),callbacks=callback_list)
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.

Epoch 1: LearningRateScheduler setting learning rate to 0.009999999776482582.
Epoch 1/10
  1/500 [..............................] - ETA: 4:25 - loss: 0.9565 - accuracy: 0.4688

WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0018s vs `on_train_batch_end` time: 0.0027s). Check your callbacks.

489/500 [============================>.] - ETA: 0s - loss: 0.6374 - accuracy: 0.6389F1_Score:  0.67525 AUC:  0.73794425

Epoch 1: val_loss improved from inf to 0.60431, saving model to model_save/weights-01-0.6752.hdf5
500/500 [==============================] - 3s 5ms/step - loss: 0.6371 - accuracy: 0.6392 - val_loss: 0.6043 - val_accuracy: 0.6752 - lr: 0.01
00

Epoch 2: LearningRateScheduler setting learning rate to 0.009999999776482582.
Epoch 2/10
496/500 [============================>.] - ETA: 0s - loss: 0.6116 - accuracy: 0.6639F1_Score:  0.675 AUC:  0.7367535

Epoch 2: val_loss improved from 0.60431 to 0.60298, saving model to model_save/weights-02-0.6750.hdf5
500/500 [==============================] - 3s 5ms/step - loss: 0.6114 - accuracy: 0.6641 - val_loss: 0.6030 - val_accuracy: 0.6750 - lr: 0.01
00

Epoch 3: LearningRateScheduler setting learning rate to 0.009499999787658453.
Epoch 3/10
498/500 [============================>.] - ETA: 0s - loss: 0.6072 - accuracy: 0.6678F1_Score:  0.66825 AUC:  0.73364525

Epoch 3: val_loss did not improve from 0.60298
500/500 [==============================] - 2s 3ms/step - loss: 0.6071 - accuracy: 0.6679 - val_loss: 0.6093 - val_accuracy: 0.6683 - lr: 0.00
95
Epoch 3: early stopping
```

Out[26]:

<keras.callbacks.History at 0x7f6e311b0ad0>

In [27]:

```
%tensorboard --logdir logs
```

Reusing TensorBoard on port 6006 (pid 433), started 0:52:39 ago. (Use '!kill 433' to kill it.)

In [28]:

```
# Clear any logs from previous runs
!rm -rf ./logs/
```

**Model-4**

1. Try with any values to get better accuracy/f1 score.

In [29]:

```
#Input layer
input_layer = Input(shape=(2,))
#Dense hidden layer 1
layer1 = Dense(128,activation='relu',kernel_initializer=tf.keras.initializers.HeUniform())(input_layer)
#Dense hidden layer 2
layer2 = Dense(128,activation='relu',kernel_initializer=tf.keras.initializers.HeUniform())(layer1)
#Dense hidden layer 3
layer3 = Dense(128,activation='relu',kernel_initializer=tf.keras.initializers.HeUniform())(layer2)
#Dense hidden layer 4
layer4 = Dense(128,activation='relu',kernel_initializer=tf.keras.initializers.HeUniform())(layer3)
#Dense hidden layer 5
layer5 = Dense(128,activation='relu',kernel_initializer=tf.keras.initializers.HeUniform())(layer4)
#output layer
output = Dense(1,activation='sigmoid',kernel_initializer=tf.keras.initializers.HeUniform())(layer5)
#Creating a model
model1 = Model(inputs=input_layer,outputs=output)
```

In [30]:

```
metrics=CalculateMetrics(validation_data=[X_test,y_test])

#Callbacks
#file path, it saves the model in the 'model_save' folder and we are naming model with epoch number
#and val acc to differtiate with other models
#you have to create model_save folder before running the code.
filepath="model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_loss',  verbose=1, save_best_only=True, mode='auto')
lrschedule = LearningRateScheduler(changeLearningRate, verbose=1)
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', factor=0.1,patience=2, min_lr=0.001)
terminate_nan=TerminateNaN()

#you can monitor any quantity (here i am monitoring val_loss), you can give any number for patience based on your need.
#i am terminating training if my validation loss incresing at once than previous loss. so maintained 1. you can give min delta
#an absolute change of less than min_delta, will count as no improvement. i maintained 0.35 because i don't want to run
#so many epoch to see termination
earlystop = EarlyStopping(monitor='val_accuracy', min_delta=0.35, patience=2, verbose=1)


optimizer=tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.95, nesterov=False, name="SGD")

model1.compile(optimizer=optimizer,loss='binary_crossentropy',metrics=['accuracy'])


logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1, write_graph=True, write_grads=True)

# here we are creating a list with all the callbacks we want
callback_list = [metrics,checkpoint,lrschedule,reduce_lr, terminate_nan,earlystop,tensorboard_callback ]

model1.fit(X_train,y_train,epochs=10,validation_data=(X_test,y_test),callbacks=callback_list)
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.
```

```
Epoch 1: LearningRateScheduler setting learning rate to 0.009999999776482582.
Epoch 1/10
490/500 [============================>.] - ETA: 0s - loss: 0.6504 - accuracy: 0.6225F1_Score:  0.647 AUC:  0.71382125

Epoch 1: val_loss improved from inf to 0.63106, saving model to model_save/weights-01-0.6470.hdf5
500/500 [==============================] - 3s 5ms/step - loss: 0.6496 - accuracy: 0.6231 - val_loss: 0.6311 - val_accuracy: 0.6470 - lr: 0.01
00

Epoch 2: LearningRateScheduler setting learning rate to 0.009999999776482582.
Epoch 2/10
488/500 [============================>.] - ETA: 0s - loss: 0.6176 - accuracy: 0.6571F1_Score:  0.651 AUC:  0.731722875

Epoch 2: val_loss improved from 0.63106 to 0.61689, saving model to model_save/weights-02-0.6510.hdf5
500/500 [==============================] - 2s 4ms/step - loss: 0.6175 - accuracy: 0.6572 - val_loss: 0.6169 - val_accuracy: 0.6510 - lr: 0.01
00

Epoch 3: LearningRateScheduler setting learning rate to 0.009499999787658453.
Epoch 3/10
490/500 [============================>.] - ETA: 0s - loss: 0.6157 - accuracy: 0.6621F1_Score:  0.67075 AUC:  0.7336295

Epoch 3: val_loss improved from 0.61689 to 0.60498, saving model to model_save/weights-03-0.6708.hdf5
500/500 [==============================] - 2s 4ms/step - loss: 0.6150 - accuracy: 0.6623 - val_loss: 0.6050 - val_accuracy: 0.6708 - lr: 0.00
95
Epoch 3: early stopping
```

Out[30]:

```
<keras.callbacks.History at 0x7f6e356734d0>
```

In [31]:

```
%tensorboard --logdir logs
```

```
Reusing TensorBoard on port 6006 (pid 433), started 1:12:07 ago. (Use '!kill 433' to kill it.)
```

In [32]:

```
# Clear any logs from previous runs
!rm -rf ./logs/
```