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)

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.

Import Section

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
import os
import datetime
from sklearn.metrics import f1_score,roc_auc_score
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.callbacks import
ModelCheckpoint,EarlyStopping,LearningRateScheduler,ReduceLROnPlateau,
TensorBoard
from tensorflow.keras.layers import Dense,Input,Activation,Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.utils import plot_model
from collections import Iterable
import warnings
warnings.filterwarnings('ignore')
```

Lets download the dataset

```
!gdown --id 15dCNcmKskcFVjs7R0ElQkR61Ex53uJpM
```

```
/usr/local/lib/python3.8/dist-packages/gdown/cli.py:127:
FutureWarning: Option `--id` was deprecated in version 4.3.1 and will
be removed in 5.0. You don't need to pass it anymore to use a file ID.
  warnings.warn(
Downloading...
From: https://drive.google.com/uc?id=15dCNcmKskcFVjs7R0ElQkR61Ex53uJpM
To: /content/data.csv
100% 887k/887k [00:00<00:00, 83.2MB/s]
```

```
dataset = pd.read_csv('/content/data.csv')
dataset.head()
```

```
         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
```

```
dataset.describe()
```

```
                 f1            f2         label
count  20000.000000  20000.000000  20000.000000
mean       0.000630     -0.000745      0.500000
std        0.671165      0.674704      0.500013
min       -1.649781     -1.600645      0.000000
25%       -0.589878     -0.596424      0.000000
50%        0.001795     -0.003113      0.500000
75%        0.586631      0.597803      1.000000
max        1.629722      1.584291      1.000000
```

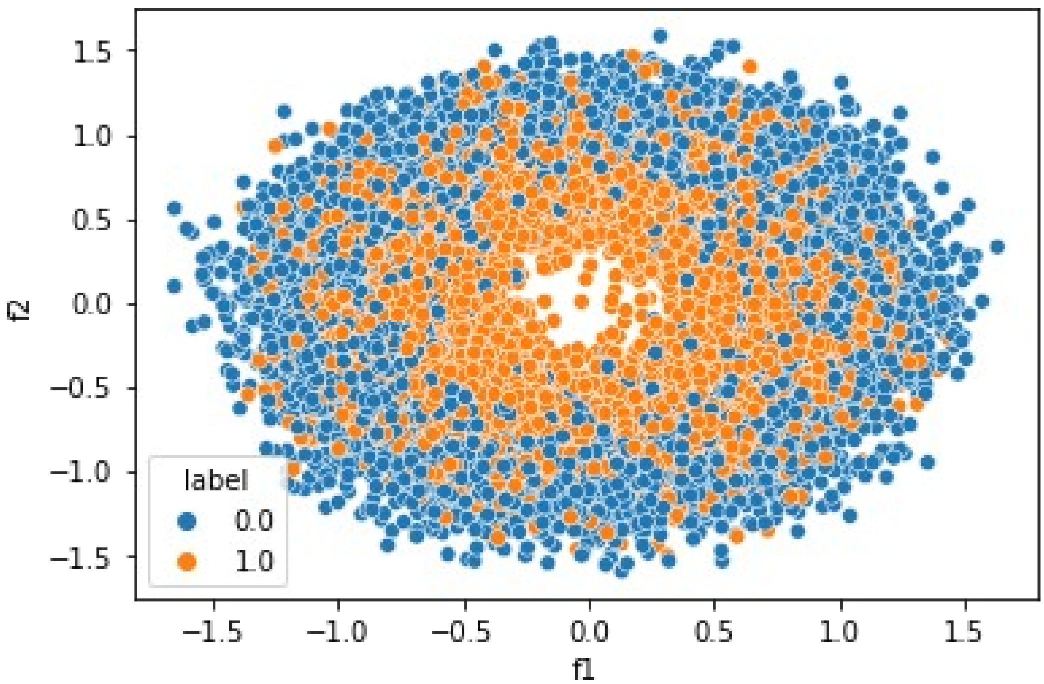
```
dataset['label'].value_counts()
```

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

2-D Scatter Plot

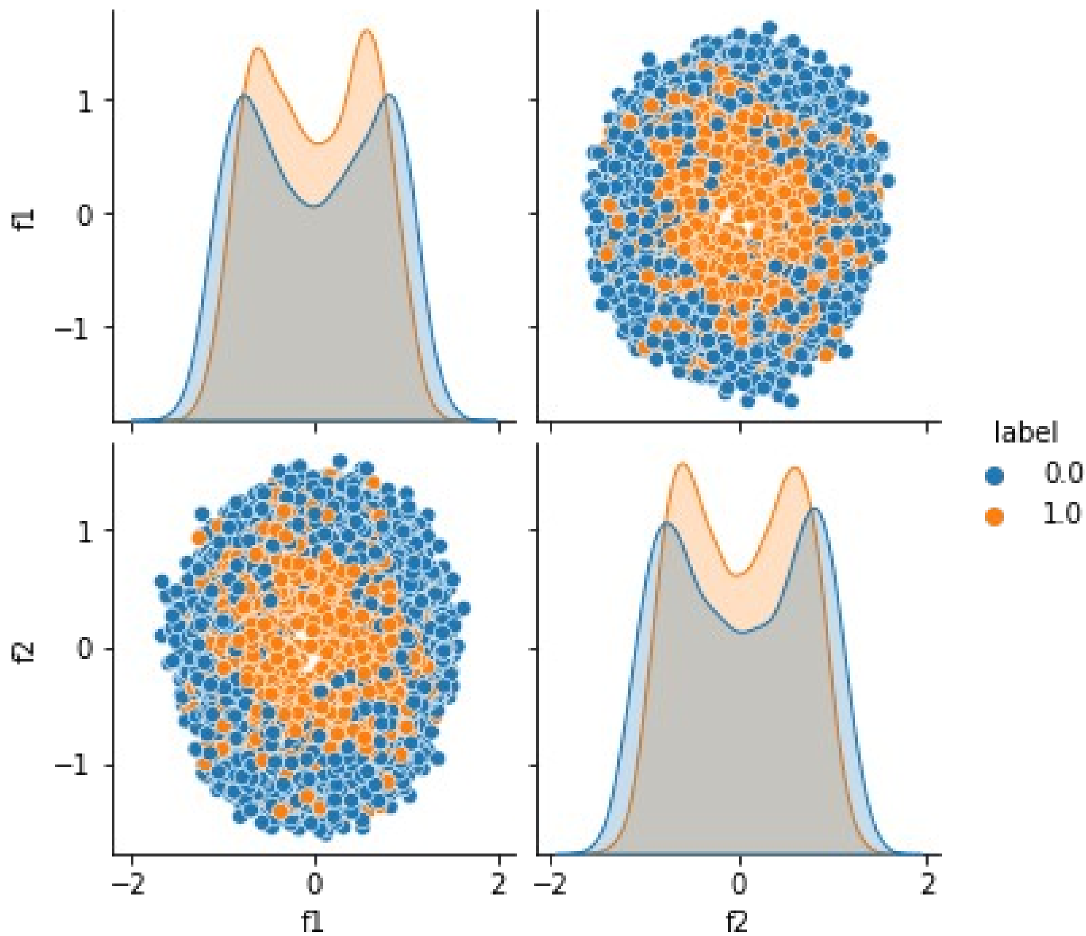
```python
sns.scatterplot(data=dataset, x="f1", y="f2", hue="label")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f33e40cb400>
```




```python
sns.pairplot(dataset, hue="label")
```

```
<seaborn.axisgrid.PairGrid at 0x7f0aeb58b310>
```



Lets create train and test datasets

```
X = dataset.iloc[:,:-1]
y = dataset.iloc[:,-1]
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.30,stratify = y , 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,)
```

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.

```
class Custom_AUC_f1(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={}):
    self.val_f1s = []
    self.val_rocs = []
  def on_epoch_end(self, epoch, logs={}):
     y_predict = (np.asarray(self.model.predict(self.x_test))).round()
     val_f1 = f1_score(y_test, y_predict.round())
     roc_val=roc_auc_score(y_test, y_predict)
     self.val_f1s.append(val_f1)
     self.val_rocs.append(roc_val)
     print("-f1 score :",val_f1,"-ROCValue :", roc_val)

Custom_AUC_F1=Custom_AUC_f1(validation_data=[X_test,y_test])
```

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

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

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%.

```
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy',
factor=0.9,patience=1, min_lr=0.0001)

def changeLearningRate(epoch,lr):
    #here we are performing exponential decay of the learning rate
    if (epoch+1)% 3 == 0:
      return lr*0.95
    return lr
lrschedule = LearningRateScheduler(changeLearningRate, verbose=1)
```

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

```
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
        def flatten(lis):
          for item in lis:
            if isinstance(item, Iterable) and not isinstance(item, 
str):
              for x in flatten(item):
                yield x
            else:
              yield item
        weights = 
np.array(list(flatten(np.array(self.model.get_weights()))))
        if(np.isnan(weights).any() or np.isinf(weights).any()):
          print("Invalid weights and terminated at epoch 
{}".format(epoch))
          self.model.stop_training = True

terminate = TerminateNaN()
```

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

```
earlystop = EarlyStopping(monitor='val_accuracy', min_delta=0.35, 
patience=2, verbose=1)
```

Use tensorboard for every model and analyse your scalar plots and histograms.

```
# Load the TensorBoard notebook extension
%load_ext tensorboard
log_dir = os.path.join("logs",'fits', 
datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = 
tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1,write_
graph=True)
%reload_ext tensorboard
```

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.
4. Analyze your output and training process.

```
X_train.shape
```

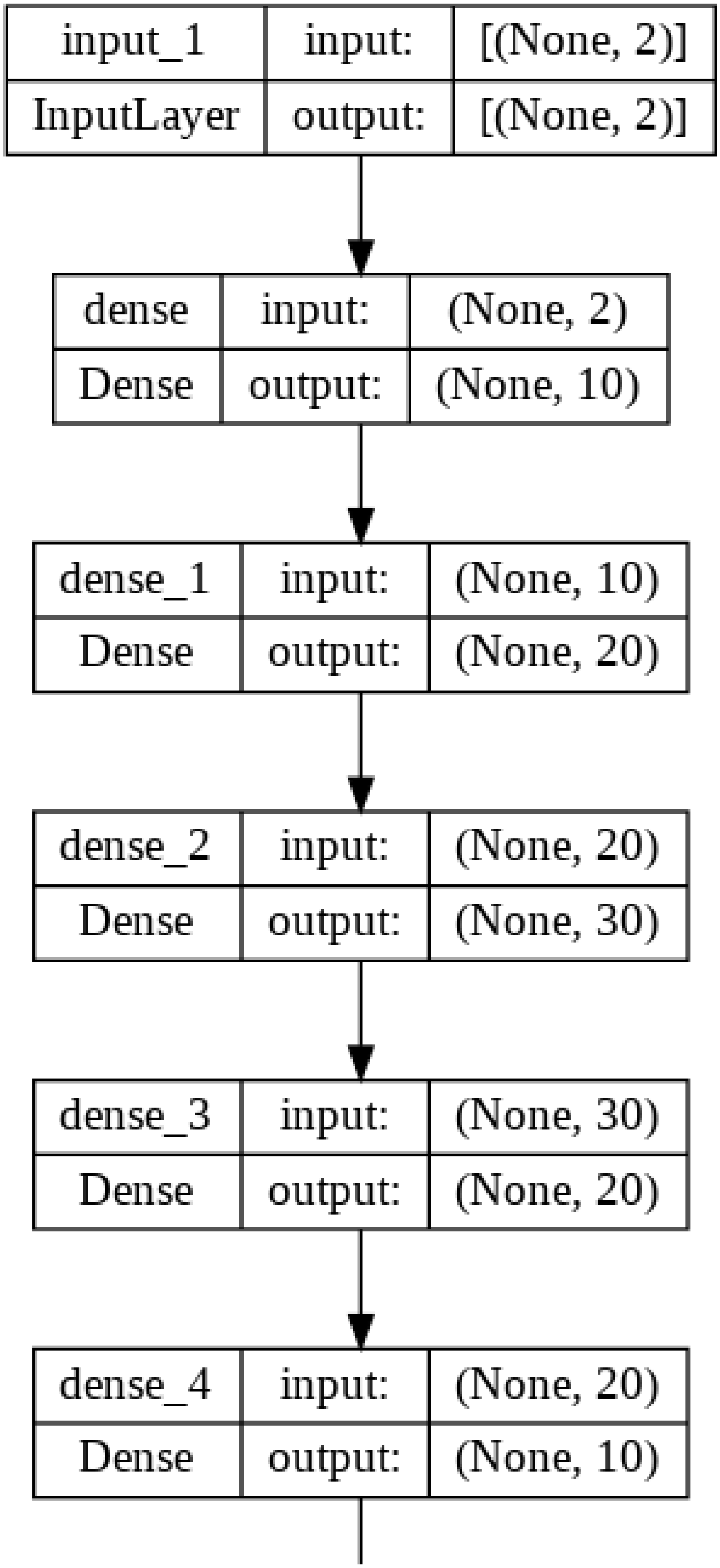
```
(14000, 2)
```

```
#Input layer
input_layer = Input(shape=(X_train.shape[1],))
#Dense hidden layer
```

```python
layer1 =
Dense(10,activation='tanh',kernel_initializer=tf.keras.initializers.Ra
ndomUniform(minval=-0, maxval=1))(input_layer)
#Dense hidden layer
layer2 =
Dense(20,activation='tanh',kernel_initializer=tf.keras.initializers.Ra
ndomUniform(minval=-0, maxval=1))(layer1)
#Dense hidden layer
layer3 =
Dense(30,activation='tanh',kernel_initializer=tf.keras.initializers.Ra
ndomUniform(minval=-0, maxval=1))(layer2)
#Dense hidden layer
layer4 =
Dense(20,activation='tanh',kernel_initializer=tf.keras.initializers.Ra
ndomUniform(minval=-0, maxval=1))(layer3)
#Dense hidden layer
layer5 =
Dense(10,activation='tanh',kernel_initializer=tf.keras.initializers.Ra
ndomUniform(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
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, 10)                30

 dense_1 (Dense)             (None, 20)                220

 dense_2 (Dense)             (None, 30)                630

 dense_3 (Dense)             (None, 20)                620

 dense_4 (Dense)             (None, 10)                210

 dense_5 (Dense)             (None, 1)                 11

=================================================================
Total params: 1,721
Trainable params: 1,721
Non-trainable params: 0
_________________________________________________________________
```

```
plot_model(model_one, show_shapes=True)
```


input_1
InputLayer
input:
output:
[(None, 2)]
[(None, 2)]
dense
Dense
input:
output:
(None, 2)
(None, 10)
dense_1
Dense
input:
output:
(None, 10)
(None, 20)
dense_2
Dense
input:
output:
(None, 20)
(None, 30)
dense_3
Dense
input:
output:
(None, 30)
(None, 20)
dense_4
Dense
input:
output:
(None, 20)
(None, 10)

```
#Callbacks
model_one.compile(optimizer='sgd',
loss='binary_crossentropy',metrics=['accuracy'])
model_one.fit(X_train,y_train,epochs=10,
validation_data=(X_test,y_test), batch_size=16,
callbacks=[Custom_AUC_F1,checkpoint,reduce_lr,lrschedule,terminate,ear
lystop,tensorboard_callback])
```

```
Epoch 1: LearningRateScheduler setting learning rate to
0.009999999776482582.
Epoch 1/10
  1/875 [..............................] - ETA: 15:21 - loss: 2.2287 -
accuracy: 0.5625

WARNING:tensorflow:Callback method `on_train_batch_end` is slow
compared to the batch time (batch time: 0.0024s vs
`on_train_batch_end` time: 0.0026s). Check your callbacks.

188/188 [==============================] - 0s 1ms/step
-f1 score : 0.49246401354784086 -ROCValue : 0.5005

Epoch 1: val_accuracy improved from -inf to 0.50050, saving model to
model_save/weights-01-0.5005.hdf5
875/875 [==============================] - 5s 4ms/step - loss: 0.7930
- accuracy: 0.4991 - val_loss: 0.6932 - val_accuracy: 0.5005 - lr:
0.0100

Epoch 2: LearningRateScheduler setting learning rate to
0.009999999776482582.
Epoch 2/10
188/188 [==============================] - 0s 1ms/step
-f1 score : 0.5076442544796975 -ROCValue : 0.5008333333333335

Epoch 2: val_accuracy improved from 0.50050 to 0.50083, saving model
to model_save/weights-02-0.5008.hdf5
875/875 [==============================] - 3s 3ms/step - loss: 0.6936
- accuracy: 0.5060 - val_loss: 0.6931 - val_accuracy: 0.5008 - lr:
0.0100

Epoch 3: LearningRateScheduler setting learning rate to
0.009499999787658453.
Epoch 3/10
188/188 [==============================] - 0s 1ms/step
-f1 score : 0.459092533047517 -ROCValue : 0.4953333333333335

Epoch 3: val_accuracy did not improve from 0.50083
875/875 [==============================] - 3s 3ms/step - loss: 0.6936
```

```
- accuracy: 0.5016 - val_loss: 0.6932 - val_accuracy: 0.4953 - lr:
0.0085
Epoch 3: early stopping
```

```
<keras.callbacks.History at 0x7f0ae30a1d60>
```

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

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

Model-2

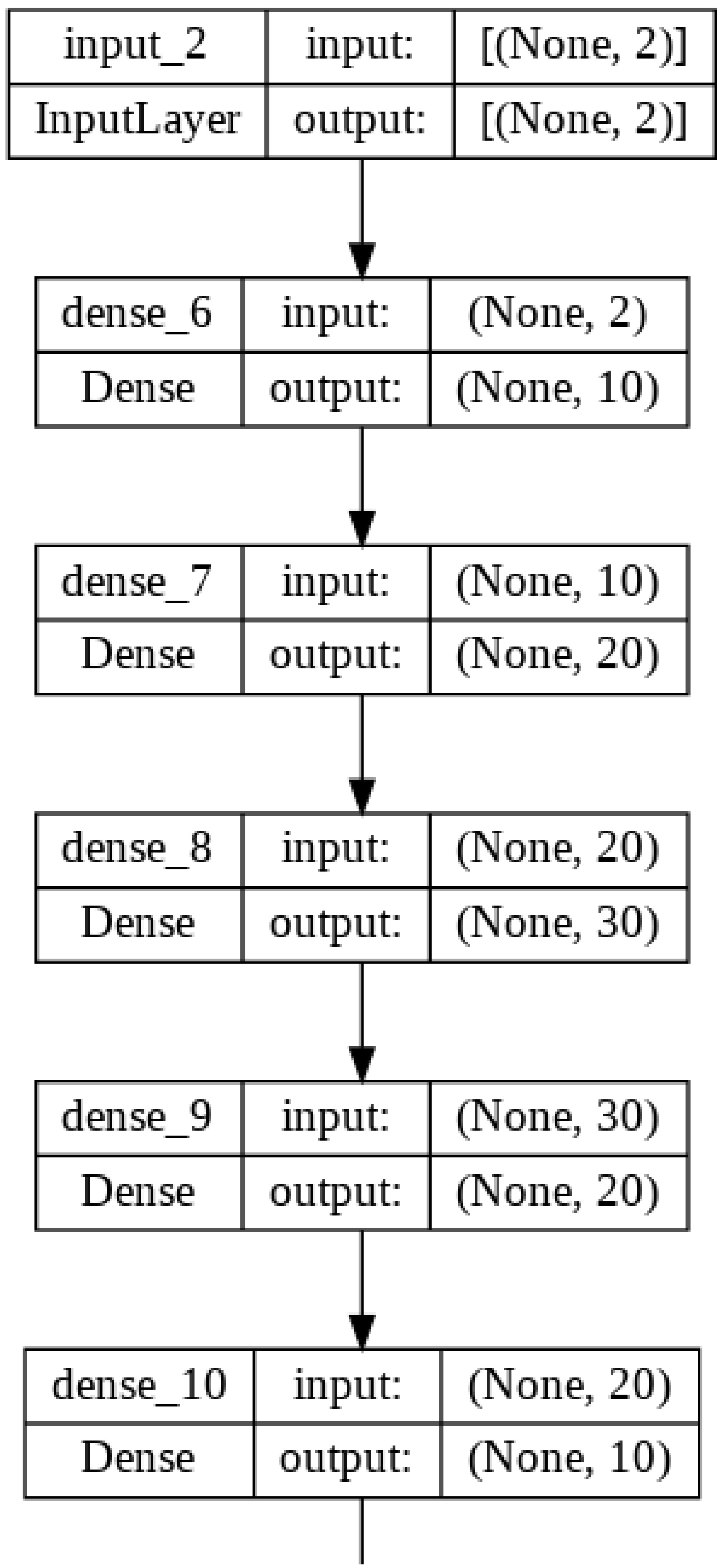
1. Use relu as an activation for every layer except output layer.
2. use SGD with momentum as optimizer.
3. use RandomUniform(0,1) as initilizer.
4. Analyze your output and training process.

```
#Input layer
input_layer = Input(shape=(X_train.shape[1],))
#Dense hidden layer
layer1 =
Dense(10,activation='relu',kernel_initializer=tf.keras.initializers.Ra
ndomUniform(minval=-0, maxval=1))(input_layer)
#Dense hidden layer
layer2 =
Dense(20,activation='relu',kernel_initializer=tf.keras.initializers.Ra
ndomUniform(minval=-0, maxval=1))(layer1)
#Dense hidden layer
layer3 =
Dense(30,activation='relu',kernel_initializer=tf.keras.initializers.Ra
ndomUniform(minval=-0, maxval=1))(layer2)
#Dense hidden layer
layer4 =
Dense(20,activation='relu',kernel_initializer=tf.keras.initializers.Ra
ndomUniform(minval=-0, maxval=1))(layer3)
#Dense hidden layer
layer5 =
Dense(10,activation='relu',kernel_initializer=tf.keras.initializers.Ra
ndomUniform(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
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, 10)                30

 dense_7 (Dense)             (None, 20)                220

 dense_8 (Dense)             (None, 30)                630

 dense_9 (Dense)             (None, 20)                620

 dense_10 (Dense)            (None, 10)                210

 dense_11 (Dense)            (None, 1)                 11

=================================================================
Total params: 1,721
Trainable params: 1,721
Non-trainable params: 0
_________________________________________________________________
```

```
plot_model(model_two, show_shapes=True)
```



input_2
input:
[(None, 2)]
InputLayer
output:
[(None, 2)]
dense_6
input:
(None, 2)
Dense
output:
(None, 10)
dense_7
input:
(None, 10)
Dense
output:
(None, 20)
dense_8
input:
(None, 20)
Dense
output:
(None, 30)
dense_9
input:
(None, 30)
Dense
output:
(None, 20)
dense_10
input:
(None, 20)
Dense
output:
(None, 10)

```
#Callbacks
optimers_sgd_with_momemtum =
tf.keras.optimizers.SGD(learning_rate=0.01,momentum=0.7,nesterov=True)

model_two.compile(optimizer=optimers_sgd_with_momemtum,
loss='binary_crossentropy',metrics=['accuracy'])
model_two.fit(X_train,y_train,epochs=10,
validation_data=(X_test,y_test), batch_size=16,
callbacks=[Custom_AUC_F1,checkpoint,reduce_lr,lrschedule,terminate,ear
lystop,tensorboard_callback])
```

```
Epoch 1: LearningRateScheduler setting learning rate to
0.009999999776482582.
Epoch 1/10
  1/875 [..............................] - ETA: 8:15 - loss: 4027.7900
- accuracy: 0.2500

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: 0.0027s). Check your callbacks.

188/188 [==============================] - 0s 1ms/step
-f1 score : 0.0 -ROCValue : 0.5

Epoch 1: val_accuracy did not improve from 0.50083
875/875 [==============================] - 4s 4ms/step - loss: 5.2957
- accuracy: 0.4976 - val_loss: 0.6932 - val_accuracy: 0.5000 - lr:
0.0100

Epoch 2: LearningRateScheduler setting learning rate to
0.009999999776482582.
Epoch 2/10
188/188 [==============================] - 0s 1ms/step
-f1 score : 0.6666666666666666 -ROCValue : 0.5

Epoch 2: val_accuracy did not improve from 0.50083
875/875 [==============================] - 3s 3ms/step - loss: 0.6933
- accuracy: 0.4973 - val_loss: 0.6932 - val_accuracy: 0.5000 - lr:
0.0090

Epoch 3: LearningRateScheduler setting learning rate to
0.008549999631941318.
Epoch 3/10
188/188 [==============================] - 0s 1ms/step
-f1 score : 0.6666666666666666 -ROCValue : 0.5

Epoch 3: val_accuracy did not improve from 0.50083
```

```
875/875 [==============================] - 3s 3ms/step - loss: 0.6933
- accuracy: 0.4923 - val_loss: 0.6933 - val_accuracy: 0.5000 - lr:
0.0077
Epoch 3: early stopping
```

```
<keras.callbacks.History at 0x7f0adf04d2e0>
```

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

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

Model-3

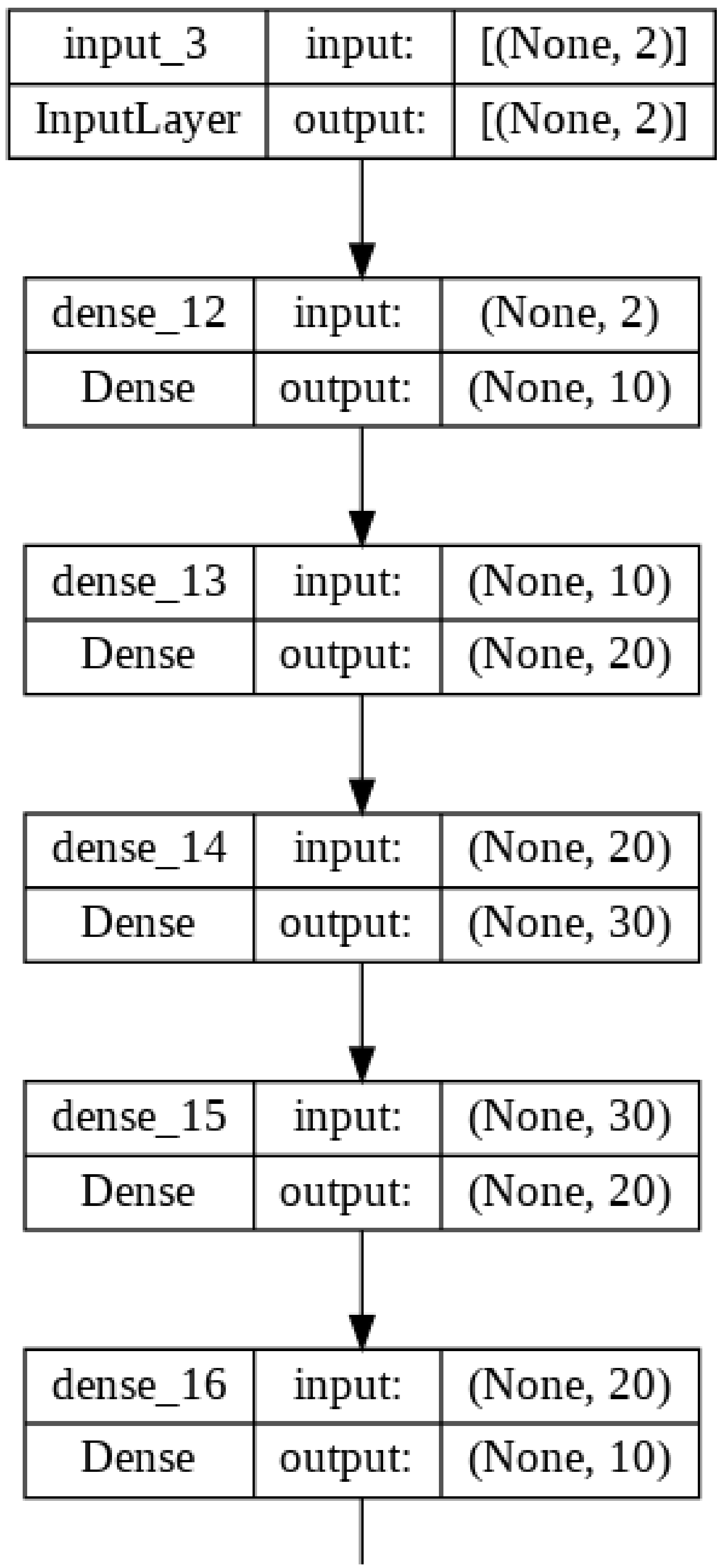
1. Use relu as an activation for every layer except output layer.
2. use SGD with momentum as optimizer.
3. use he_uniform() as initilizer.
4. Analyze your output and training process.

```
#Input layer
input_layer = Input(shape=(X_train.shape[1],))
#Dense hidden layer
layer1 =
Dense(10,activation='relu',kernel_initializer=tf.keras.initializers.he
_uniform())(input_layer)
#Dense hidden layer
layer2 =
Dense(20,activation='relu',kernel_initializer=tf.keras.initializers.he
_uniform())(layer1)
#Dense hidden layer
layer3 =
Dense(30,activation='relu',kernel_initializer=tf.keras.initializers.he
_uniform())(layer2)
#Dense hidden layer
layer4 =
Dense(20,activation='relu',kernel_initializer=tf.keras.initializers.he
_uniform())(layer3)
#Dense hidden layer
layer5 =
Dense(10,activation='relu',kernel_initializer=tf.keras.initializers.he
_uniform())(layer4)
#output layer
output =
Dense(1,activation='sigmoid',kernel_initializer=tf.keras.initializers.
he_uniform())(layer5)
#Creating a model
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, 10)                30

 dense_13 (Dense)            (None, 20)                220

 dense_14 (Dense)            (None, 30)                630

 dense_15 (Dense)            (None, 20)                620

 dense_16 (Dense)            (None, 10)                210

 dense_17 (Dense)            (None, 1)                 11

=================================================================
Total params: 1,721
Trainable params: 1,721
Non-trainable params: 0
_________________________________________________________________
```

```
plot_model(model_three, show_shapes=True)
```


input_3
input:
[(None, 2)]
InputLayer
output:
[(None, 2)]
dense_12
input:
(None, 2)
Dense
output:
(None, 10)
dense_13
input:
(None, 10)
Dense
output:
(None, 20)
dense_14
input:
(None, 20)
Dense
output:
(None, 30)
dense_15
input:
(None, 30)
Dense
output:
(None, 20)
dense_16
input:
(None, 20)
Dense
output:
(None, 10)

```
#Callbacks
optimers_sgd_with_momemtum =
tf.keras.optimizers.SGD(learning_rate=0.01,momentum=0.5,nesterov=True)

model_three.compile(optimizer=optimers_sgd_with_momemtum,
loss='binary_crossentropy',metrics=['accuracy'])
model_three.fit(X_train,y_train,epochs=10,
validation_data=(X_test,y_test), batch_size=16,
callbacks=[Custom_AUC_F1,checkpoint,reduce_lr,lrschedule,terminate,ear
lystop,tensorboard_callback])
```

```
Epoch 1: LearningRateScheduler setting learning rate to
0.009999999776482582.
Epoch 1/10
  1/875 [..............................] - ETA: 17:47 - loss: 0.7128 -
accuracy: 0.3750

WARNING:tensorflow:Callback method `on_train_batch_end` is slow
compared to the batch time (batch time: 0.0031s vs
`on_train_batch_end` time: 0.0137s). Check your callbacks.

188/188 [==============================] - 0s 2ms/step
-f1 score : 0.6203506472226774 -ROCValue : 0.6138333333333333

Epoch 1: val_accuracy improved from 0.50083 to 0.61383, saving model
to model_save/weights-01-0.6138.hdf5
875/875 [==============================] - 6s 5ms/step - loss: 0.6717
- accuracy: 0.5809 - val_loss: 0.6506 - val_accuracy: 0.6138 - lr:
0.0100

Epoch 2: LearningRateScheduler setting learning rate to
0.009999999776482582.
Epoch 2/10
188/188 [==============================] - 0s 1ms/step
-f1 score : 0.6764005713378829 -ROCValue : 0.6601666666666666

Epoch 2: val_accuracy improved from 0.61383 to 0.66017, saving model
to model_save/weights-02-0.6602.hdf5
875/875 [==============================] - 4s 4ms/step - loss: 0.6248
- accuracy: 0.6481 - val_loss: 0.6152 - val_accuracy: 0.6602 - lr:
0.0100

Epoch 3: LearningRateScheduler setting learning rate to
0.009499999787658453.
Epoch 3/10
188/188 [==============================] - 0s 1ms/step
-f1 score : 0.5922848664688427 -ROCValue : 0.6565000000000001

Epoch 3: val_accuracy did not improve from 0.66017
```

```
875/875 [==============================] - 4s 5ms/step - loss: 0.6068
- accuracy: 0.6639 - val_loss: 0.6226 - val_accuracy: 0.6565 - lr:
0.0085
Epoch 3: early stopping

<keras.callbacks.History at 0x7f0ae9d93400>
```

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

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

Model-4

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

```
!rm -rf ./logs/
tf.compat.v1.reset_default_graph()
```

Lets employ few techniques to better the model

Lets resplit the data with 90:10 as deep learning perform better with more data.

```
X_model4 = dataset.iloc[:,:-1]
y_model4 = dataset.iloc[:,-1]
X_train_model4, X_test_model4, y_train_model4, y_test_model4 =
train_test_split(X_model4, y_model4, test_size=0.10,stratify =
y_model4 , random_state = 42)
print(X_train_model4.shape)
print(X_test_model4.shape)
print(y_train_model4.shape)
print(y_test_model4.shape)
```

```
(18000, 2)
(2000, 2)
(18000,)
(2000,)
```

Lets rescale the data using minmaxscalar and bring data in range 0-1.

```
X_train_model4.describe()
```

```
                 f1            f2
count  18000.000000  18000.000000
mean       0.000661     -0.003546
std        0.670208      0.674711
min       -1.649781     -1.600645
25%       -0.588755     -0.598939
50%        0.000253     -0.008212
75%        0.586433      0.593109
max        1.629722      1.584291
```

```
from sklearn.preprocessing import MinMaxScaler
minmaxscalar = MinMaxScaler()
```

```
X_train_model4 = minmaxscalar.fit_transform(X_train_model4)
X_test_model4 = minmaxscalar.transform(X_test_model4)
```

```
print(X_train_model4[:,0].min())
print(X_train_model4[:,0].max())
print(X_test_model4[:,0].min())
print(X_test_model4[:,0].max())
```

```
0.0
1.0
0.01485105225468586
0.9655135024755643
```

- We can see values are in range 0 to 1

Lets find the perfect hyper parameters using keras tuner

```
#pip install keras-tuner
```

```
import keras_tuner as kt
from tensorflow import keras
```

```
def build_model(hp):
  model = keras.Sequential()
  counter = 0
  for i in range(hp.Int('num_layers',min_value = 1, max_value = 5)):
    if counter ==0:
      model.add(
          keras.layers.Dense(
              hp.Int('units'+str(i), min_value = 8, max_value =
100,step =8),
              activation =
hp.Choice('activation'+str(i),values=['relu','tanh','sigmoid']),
              kernel_initializer =
hp.Choice('initialiser'+str(i),values =
['he_normal','glorot_normal']),
              input_dim = X_train_model4.shape[1]

          )
          )
      model.add(

keras.layers.Dropout(hp.Choice('dropout'+str(i),values=[0.1,0.2,0.3,0.
4,0.5,0.6,0.7,0.8,0.9]))
          )
    else:
      model.add(
          keras.layers.Dense(
              hp.Int('units'+str(i), min_value = 8, max_value =
100,step =8),
              activation =
hp.Choice('activation'+str(i),values=['relu','tanh','sigmoid']),
```

```
              kernel_initializer =
hp.Choice('initialiser'+str(i),values = ['he_normal','glorot_normal'])
          )
          )
      model.add(

keras.layers.Dropout(hp.Choice('dropout'+str(i),values=[0.1,0.2,0.3,0.
4,0.5,0.6,0.7,0.8,0.9]))
          )
    counter += 1
  model.add(keras.layers.Dense(1,activation = 'sigmoid'))
  model.compile(optimizer = hp.Choice('optimiser', values =
['rmsprop','sgd','adam','adadelta','nadam']),loss='binary_crossentropy
',metrics=['accuracy'])
  return model


tuner = kt.RandomSearch(build_model,objective =
'val_accuracy',max_trials = 3, directory =
"hyperparametertuning",project_name = 'callbackassignment1')

tuner.search(X_train_model4,y_train_model4,epochs = 5,validation_data
= (X_test_model4,y_test_model4))
```

```
Trial 3 Complete [00h 00m 11s]
val_accuracy: 0.5

Best val_accuracy So Far: 0.6704999804496765
Total elapsed time: 00h 00m 28s
```

lets get the best hyper parameters

```
tuner.get_best_hyperparameters()[0].values
```

```
{'num_layers': 2,
 'units0': 96,
 'activation0': 'relu',
 'initialiser0': 'glorot_normal',
 'dropout0': 0.4,
 'optimiser': 'adam',
 'units1': 8,
 'activation1': 'relu',
 'initialiser1': 'he_normal',
 'dropout1': 0.1}
```

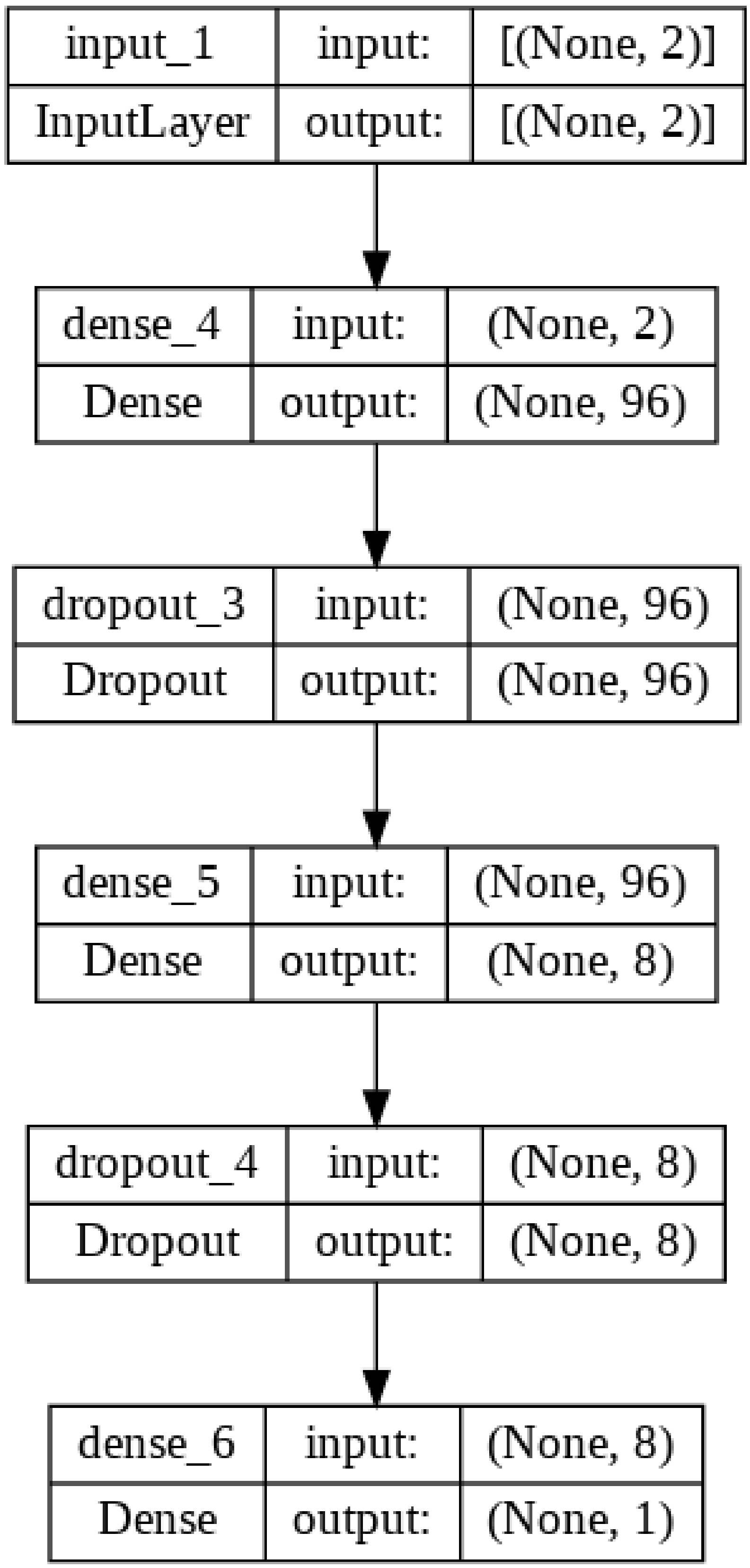
We got the best hyypeameters lets build the model 4

```
#Input layer
input_layer = Input(shape=(X_train_model4.shape[1],))
#Dense hidden layer
layer1 =
Dense(96,activation='relu',kernel_initializer='glorot_normal')
```

```
(input_layer)
#Dropout layer
dropout1 = Dropout(0.4)(layer1)
#Dense hidden layer
layer2 = Dense(8,activation='relu',kernel_initializer='he_normal')
(dropout1)
#Dropout layer
dropout2 = Dropout(0.1)(layer2)
#output layer
output = Dense(1,activation='sigmoid')(dropout2)
#Creating a model
model_four = Model(inputs=input_layer,outputs=output)
model_four.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 2)]               0

 dense_4 (Dense)             (None, 96)                288

 dropout_3 (Dropout)         (None, 96)                0

 dense_5 (Dense)             (None, 8)                 776

 dropout_4 (Dropout)         (None, 8)                 0

 dense_6 (Dense)             (None, 1)                 9

=================================================================
Total params: 1,073
Trainable params: 1,073
Non-trainable params: 0
_________________________________________________________________
```

```
plot_model(model_four, show_shapes=True)
```


input_1
input:
[(None, 2)]
InputLayer
output:
[(None, 2)]
dense_4
input:
(None, 2)
Dense
output:
(None, 96)
dropout_3
input:
(None, 96)
Dropout
output:
(None, 96)
dense_5
input:
(None, 96)
Dense
output:
(None, 8)
dropout_4
input:
(None, 8)
Dropout
output:
(None, 8)
dense_6
input:
(None, 8)
Dense
output:
(None, 1)

```
reduce_lr_model4 = ReduceLROnPlateau(monitor='val_accuracy',
factor=0.9,patience=1, min_lr=0.0001)

earlystop_model4 = EarlyStopping(monitor='val_accuracy',
min_delta=0.00009, patience=3, verbose=1)

#Callbacks
model_four.compile(optimizer='adam',
loss='binary_crossentropy',metrics=['accuracy'])
model_four.fit(X_train_model4,y_train_model4,epochs=100,
validation_data=(X_test_model4,y_test_model4), batch_size=512,
callbacks=[Custom_AUC_F1,checkpoint,reduce_lr_model4,earlystop_model4,
tensorboard_callback])

Epoch 1/100
188/188 [==============================] - 0s 1ms/step
-f1 score : 0.13624141021810576 -ROCValue : 0.5181666666666667

Epoch 1: val_accuracy did not improve from 0.69600
36/36 [==============================] - 1s 19ms/step - loss: 0.6176 -
accuracy: 0.6606 - val_loss: 0.5937 - val_accuracy: 0.6915 - lr:
0.0010
Epoch 2/100
188/188 [==============================] - 0s 1ms/step
-f1 score : 0.13727245598328858 -ROCValue : 0.5181666666666667

Epoch 2: val_accuracy did not improve from 0.69600
36/36 [==============================] - 1s 15ms/step - loss: 0.6163 -
accuracy: 0.6614 - val_loss: 0.5929 - val_accuracy: 0.6955 - lr:
0.0010
Epoch 3/100
188/188 [==============================] - 0s 1ms/step
-f1 score : 0.13825983313468415 -ROCValue : 0.5179999999999999

Epoch 3: val_accuracy improved from 0.69600 to 0.69750, saving model
to model_save/weights-03-0.6975.hdf5
36/36 [==============================] - 1s 15ms/step - loss: 0.6140 -
accuracy: 0.6644 - val_loss: 0.5929 - val_accuracy: 0.6975 - lr:
0.0010
Epoch 4/100
188/188 [==============================] - 0s 1ms/step
-f1 score : 0.13714967203339296 -ROCValue : 0.5176666666666667

Epoch 4: val_accuracy did not improve from 0.69750
36/36 [==============================] - 0s 13ms/step - loss: 0.6170 -
accuracy: 0.6604 - val_loss: 0.5934 - val_accuracy: 0.6940 - lr:
0.0010
Epoch 5/100
188/188 [==============================] - 0s 1ms/step
-f1 score : 0.13714967203339296 -ROCValue : 0.5176666666666667
```

```
Epoch 5: val_accuracy did not improve from 0.69750
36/36 [==============================] - 1s 15ms/step - loss: 0.6150 -
accuracy: 0.6632 - val_loss: 0.5936 - val_accuracy: 0.6935 - lr:
0.0010
Epoch 6/100
188/188 [==============================] - 0s 1ms/step
-f1 score : 0.1389385807990459 -ROCValue : 0.5186666666666666

Epoch 6: val_accuracy improved from 0.69750 to 0.69800, saving model
to model_save/weights-06-0.6980.hdf5
36/36 [==============================] - 1s 15ms/step - loss: 0.6146 -
accuracy: 0.6647 - val_loss: 0.5929 - val_accuracy: 0.6980 - lr:
0.0010
Epoch 7/100
188/188 [==============================] - 0s 1ms/step
-f1 score : 0.14000595770032767 -ROCValue : 0.5188333333333334

Epoch 7: val_accuracy improved from 0.69800 to 0.70050, saving model
to model_save/weights-07-0.7005.hdf5
36/36 [==============================] - 1s 15ms/step - loss: 0.6161 -
accuracy: 0.6610 - val_loss: 0.5924 - val_accuracy: 0.7005 - lr:
0.0010
Epoch 8/100
188/188 [==============================] - 0s 1ms/step
-f1 score : 0.13825983313468415 -ROCValue : 0.5179999999999999

Epoch 8: val_accuracy did not improve from 0.70050
36/36 [==============================] - 0s 13ms/step - loss: 0.6158 -
accuracy: 0.6618 - val_loss: 0.5926 - val_accuracy: 0.6960 - lr:
0.0010
Epoch 9/100
188/188 [==============================] - 0s 1ms/step
-f1 score : 0.14051801131289074 -ROCValue : 0.5188333333333334

Epoch 9: val_accuracy did not improve from 0.70050
36/36 [==============================] - 0s 13ms/step - loss: 0.6147 -
accuracy: 0.6645 - val_loss: 0.5918 - val_accuracy: 0.6990 - lr:
0.0010
Epoch 10/100
188/188 [==============================] - 0s 1ms/step
-f1 score : 0.1394517282479142 -ROCValue : 0.5186666666666667

Epoch 10: val_accuracy did not improve from 0.70050
36/36 [==============================] - 0s 13ms/step - loss: 0.6137 -
accuracy: 0.6647 - val_loss: 0.5930 - val_accuracy: 0.6940 - lr:
0.0010
Epoch 10: early stopping

<keras.callbacks.History at 0x7f33da1407c0>
```

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

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

Observations:

Accuracy of model 1: 0.49234

Accuracy of model 2: 0.50083

Accuracy of model 3: 0.66017

Accuracy of model 4: 0.70050

Note: Succesfully improved the model performance using hyper parameter tuning.