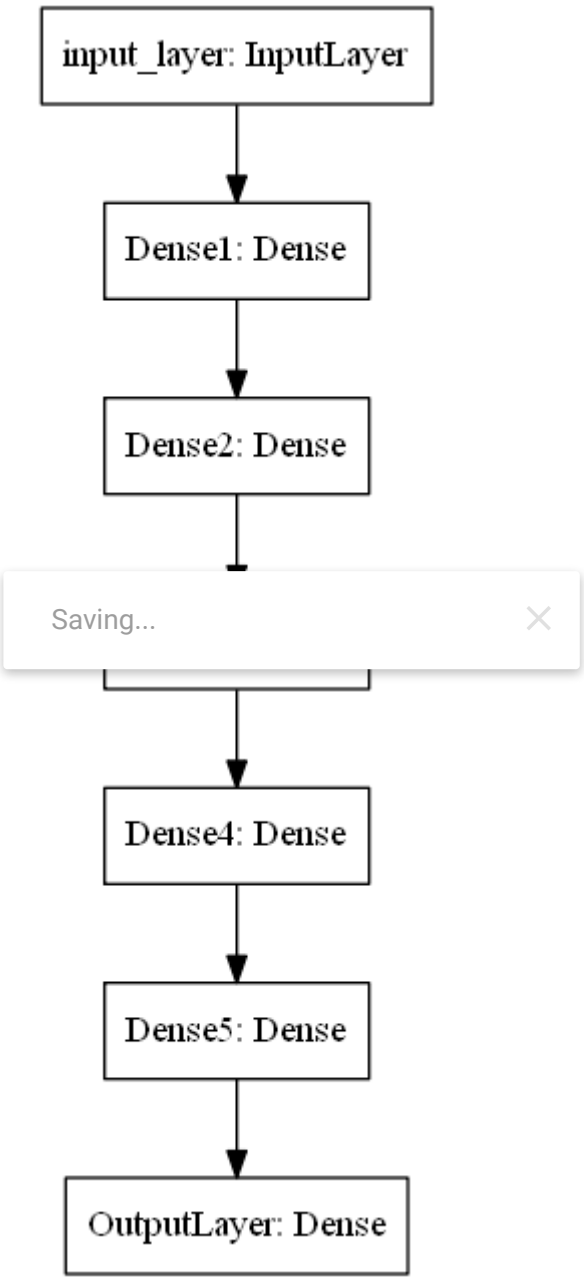
1. Download the data from here. You have to use data.csv file for this assignment

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



Saving...

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 a
       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)

```
1 pip install tensorflow scikeras scikit-learn
```

```
                                                                    i.org/simple, https://us-python.pkg.dev/colab-wheels/p
                                                                    tensorflow in /usr/local/lib/python3.7/dist-packages (
                                                                    scikeras in /usr/local/lib/python3.7/dist-packages (0
    Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages
    Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.7/d
    Requirement already satisfied: tensorflow-estimator<2.10.0,>=2.9.0rc0 in /usr/local/l
    Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.7/dist-p
    Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.7/dist-p
    Requirement already satisfied: tensorboard<2.10,>=2.9 in /usr/local/lib/python3.7/dis
    Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.7/dist-packages
    Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib
    Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.7/dist-pac
    Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.7/dist-p
    Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.7/dist-packages
    Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.7/dist-packag
    Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (1
    Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (
    Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.7/dist-package
    Requirement already satisfied: keras<2.10.0,>=2.9.0rc0 in /usr/local/lib/python3.7/di
    Requirement already satisfied: keras-preprocessing>=1.1.1 in /usr/local/lib/python3.7
    Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.7/dist-pac
    Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.7/dist-pack
    Requirement already satisfied: protobuf<3.20,>=3.9.2 in /usr/local/lib/python3.7/dist
    Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.7/dist-packages
    Requirement already satisfied: flatbuffers<2,>=1.12 in /usr/local/lib/python3.7/dist-
    Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.7/dist-pack
    Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.7/dist-pa
    Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packa
    Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.7/dist
    Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/pyt
    Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-packa
    Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.7/dist-packa
    Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local/li
    Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python
```

```
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.7/dist
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.7/dis
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.7/d
Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/python3.7/di
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/d
```

```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd#pandas to create small dataframes
# if numpy is not installed already : pip3 install numpy
import numpy as np#Do aritmetic operations on arrays
# matplotlib: used to plot graphs
```

Saving...

```
import seaborn as sns#Plots

from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.layers import Dense,Input,Activation
from tensorflow.keras.models import Model,Sequential
import random as rn
from tensorflow import keras
import datetime, os

from keras.callbacks import Callback
from keras.initializers import RandomUniform,HeUniform
from sklearn.metrics import roc_auc_score, f1_score
from scikeras.wrappers import KerasClassifier
from keras.optimizers import SGD
from tensorflow.keras.callbacks import TerminateOnNaN,ReduceLROnPlateau,EarlyStopping,L
from keras.callbacks import TensorBoard
```

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
data=pd.read_csv("drive/My Drive/Working-with-callbacks/data.csv")
data.head()
```

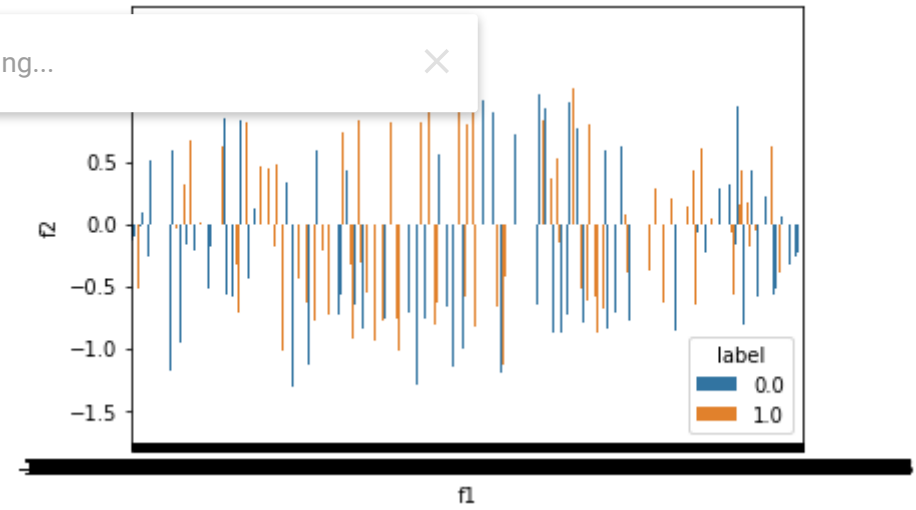
	f1	f2	label
0	0.450564	1.074305	0.0
1	0.085632	0.967682	0.0
2	0.117326	0.971521	1.0
3	0.982179	-0.380408	0.0
4	-0.720352	0.955850	0.0

```
1 y=data['label'].values
2 X=data[["f1","f2"]].values
```

```
1 set(y)
```

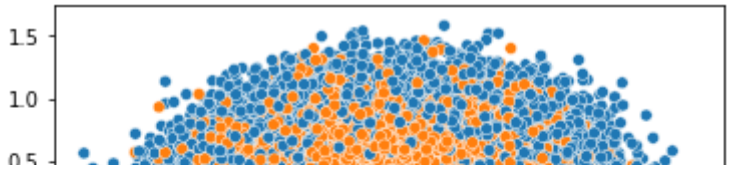
```
{0.0, 1.0}
```

```
1 sns.barplot(x = "f1",y= "f2",hue = "label",data = data)
2 plt.figure(figsize = (15,8))
3 plt.show()
```


```
<Figure size 1080x576 with 0 Axes>
```

```
1 sns.scatterplot(x = "f1",y= "f2",hue = "label",data = data)
2 plt.figure(figsize = (15,10))
3 plt.show()
```




```
1 data.shape
```

```
(20000, 3)
```

```
1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

```
1 y_test.shape
```

```
(5000,)
```

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. ...g process.

Saving...

```
1 class Metrics(tf.keras.callbacks.Callback):
2   def __init__(self):
3     self.validation_data=(X_test,y_test)
4   def on_train_begin(self, logs={}):
5     self.val_f1s = []
6   def on_epoch_end(self, epoch, logs={}):
7      val_predict = (np.asarray(self.model.predict(self.validation_data[0])))
8      val_target = self.validation_data[1]
9      val_predict_prob =np.argmax(val_predict,axis=1)
10     val_f1 = f1_score(val_target, val_predict_prob,average='micro')
11     roc_val = roc_auc_score(val_target, val_predict_prob)
12     self.val_f1s.append(val_f1)
13     print("f1 score :",val_f1,"AUC value :", roc_val)
14
15
```

```
1 # Stop training if NaN is encountered
2 class TerminateNaN(tf.keras.callbacks.Callback):
3     def on_epoch_end_loss(self, epoch, logs={}):
4         loss = self.model.get_loss()
5         if loss is not None:
6             if np.isnan(loss) or np.isinf(loss):
7                 print("Invalid loss and terminated at epoch {}".format(epoch))
```

```
                self.model.stop_training = True

    def on_epoch_end_weight(self,epoch,logs ={}):
      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 weight and terminated at epoch {}".format(epoch))
          self.model.stop_training = True

NanStop = TerminateNaN()
```

```
#For every 3rd epoch, decay your learning rate by 5%.
def schedule(epoch,lr):
  if epoch % 3 == 0:
    lr = lr - (lr*.05)
    return lr
  return lr

# Decrease learning rate by 5% for every 3rd epoch
LrScheduler = LearningRateScheduler(schedule,verbose=0)

# Decrease learning rate by 10%
LrValAccuracy = ReduceLROnPlateau(monitor='val_accuracy', patience=1, factor= 0.9, mode

                                  your validation accuracy is not increased in last 2 e
                                  ='val_accuracy', min_delta=0.35, patience=2, verbose=1

#Save your model at every epoch if your validation accuracy is improved from previous e
filepath="drive/My Drive/Working-with-callbacks/best_model.hdf5"
CheckPoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy',  verbose=1, sav

```


Double-click (or enter) to edit

```
def build_model(activation_fun,initializer_val):
  model = Sequential()
  model.add(Dense(units = 2,kernel_initializer = initializer_val, activation = activati
  model.add(Dense(units = 20,kernel_initializer = initializer_val, activation = activat
  model.add(Dense(units = 20,kernel_initializer = initializer_val, activation = activat
  model.add(Dense(units = 20,kernel_initializer = initializer_val, activation = activat
  model.add(Dense(units = 20,kernel_initializer = initializer_val, activation = activat
  model.add(Dense(units = 20,kernel_initializer = initializer_val, activation = activat
  model.add(Dense(units = 1,kernel_initializer = initializer_val, activation = "sigmoid

  return model

```

```
activation_fun_1  = "tanh"
initializer_val_1 = RandomUniform(0,1)
# create model
model_1 = build_model(activation_fun_1,initializer_val_1)
```

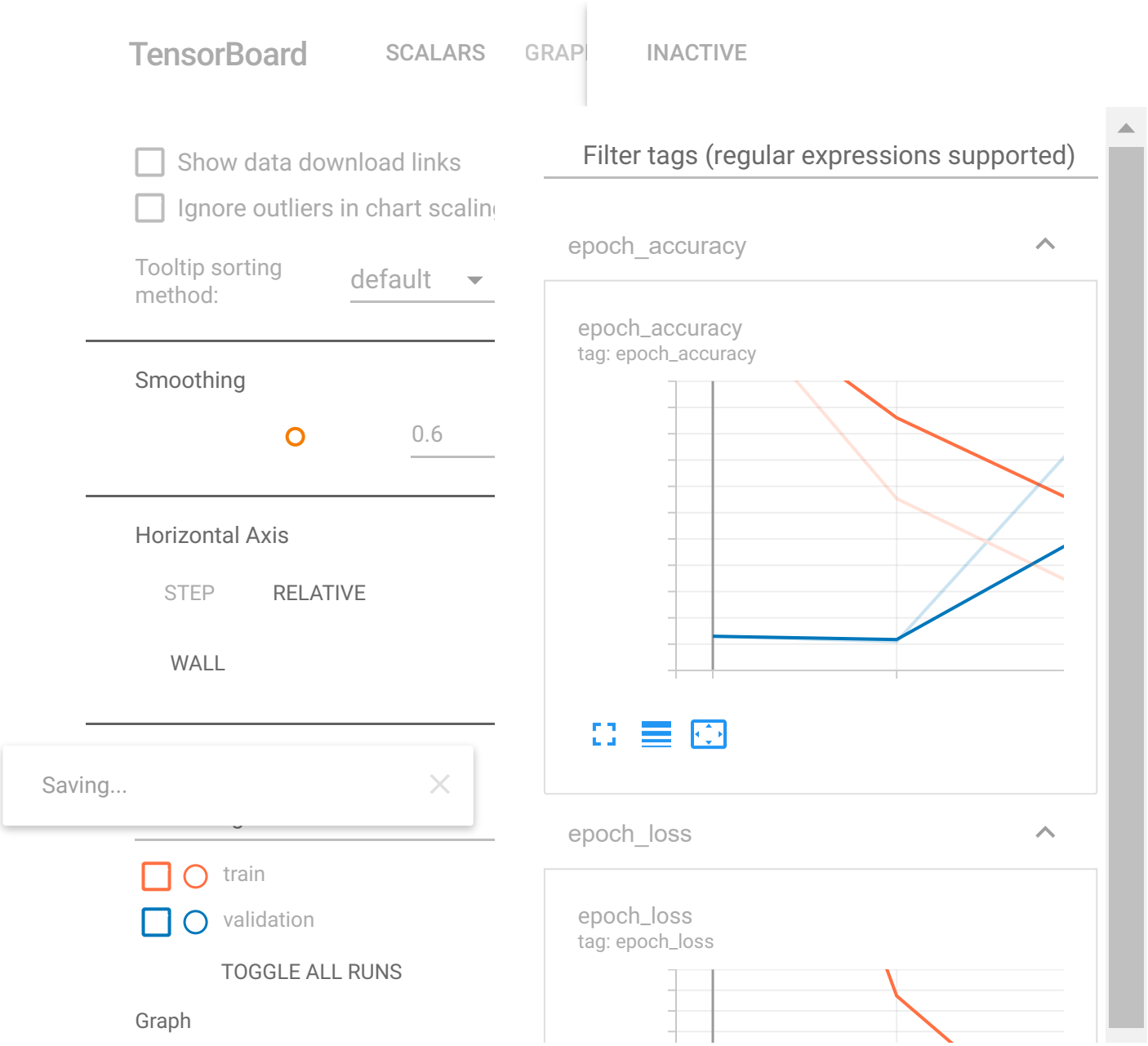
```
optimizer_1 = SGD(lr=0.01, momentum=0.9, nesterov=False,name='SGD')
model_1.compile(optimizer_1,loss='binary_crossentropy', metrics=['accuracy'])

metric_vals = Metrics()

tb = TensorBoard(log_dir='./Graph', histogram_freq=2,write_graph=True, write_images=Tru
tb.set_model(model_1)

cb  = [NanStop,LrValAccuracy,LrScheduler,CheckPoint,metric_vals,tb,EarlyStop]

model_1.fit(x = X_train, y = y_train,epochs=10,validation_data=(X_test,y_test),callback

```

```
Epoch 1/10
  1/469 [..............................] - ETA: 6:05 - loss: 3.7213 - accuracy: 0.562
454/469 [============================>.] - ETA: 0s - loss: 0.7765 - accuracy: 0.5205
Epoch 1: val_accuracy improved from -inf to 0.49460, saving model to drive/My Drive/W
157/157 [==============================] - 0s 1ms/step
f1 score : 0.4942 AUC value : 0.5
469/469 [==============================] - 3s 5ms/step - loss: 0.7739 - accuracy: 0.5
Epoch 2/10
468/469 [============================>.] - ETA: 0s - loss: 0.6964 - accuracy: 0.5051
Epoch 2: val_accuracy did not improve from 0.49460
157/157 [==============================] - 0s 2ms/step
f1 score : 0.4942 AUC value : 0.5
469/469 [==============================] - 2s 3ms/step - loss: 0.6964 - accuracy: 0.5
Saving...
=====>..] - ETA: 0s - loss: 0.6953 - accuracy: 0.4993
Epoch 3: val_accuracy improved from 0.49460 to 0.50960, saving model to drive/My Driv
157/157 [==============================] - 0s 1ms/step
f1 score : 0.4942 AUC value : 0.5
469/469 [==============================] - 2s 4ms/step - loss: 0.6955 - accuracy: 0.4
Epoch 3: early stopping
<keras.callbacks.History at 0x7fc077f75290>
```

```
print("TensorFlow version: ", tf.__version__)
# Load the TensorBoard notebook extension.
%load_ext tensorboard
```

```
TensorFlow version:  2.9.2
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
%tensorboard --logdir Graph
```




```
train_acc_model_1 = model_1.evaluate(X_train, y_train, verbose=0)
test_acc_model_1 = model_1.evaluate(X_test, y_test, verbose=0)

train_accuracy_model_1 = train_acc_model_1[1]
test_accuracy_model_1 = test_acc_model_1[1]
train_loss_model_1 = train_acc_model_1[0]
test_loss_model_1 = test_acc_model_1[0]

print("Train data accuracy:",train_accuracy_model_1)
print("Test data accuracy:",test_accuracy_model_1)
```

```
Train data accuracy: 0.5006666779518127
Test data accuracy: 0.5095999836921692
```

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.

```
activation_fun_2  = "relu"
initializer_val_2 = RandomUniform(0,1)

# create model
model_2 = build_model(activation_fun_2,initializer_val_2)
optimizer_2 = SGD(lr=0.01, momentum=0.9, nesterov=False,name='SGD')
model_2.compile(optimizer_2,loss='binary_crossentropy', metrics=['accuracy'])

metric_vals = Metrics()

tb = TensorBoard(log_dir='./Graph', histogram_freq=2,write_graph=True, write_images=Tru
tb.set_model(model_2)

cb  = [NanStop,LrValAccuracy,LrScheduler,CheckPoint,metric_vals,tb,EarlyStop]

model_2.fit(x = X_train, y = y_train,epochs=10,validation_data=(X_test,y_test),callback
```



```
Epoch 1/10
  1/469 [..............................] - ETA: 5:22 - loss: 7665.0679 - accuracy: 0
459/469 [============================>.] - ETA: 0s - loss: 17.3913 - accuracy: 0.5046
Epoch 1: val_accuracy did not improve from 0.50960
157/157 [==============================] - 0s 2ms/step
f1 score : 0.4942 AUC value : 0.5
469/469 [==============================] - 3s 4ms/step - loss: 17.0440 - accuracy: 0
Epoch 2/10
456/469 [============================>.] - ETA: 0s - loss: 0.6933 - accuracy: 0.4985
Epoch 2: val_accuracy did not improve from 0.50960
157/157 [==============================] - 0s 1ms/step
f1 score : 0.4942 AUC value : 0.5
469/469 [==============================] - 2s 4ms/step - loss: 0.6933 - accuracy: 0.4
Epoch 3/10
465/469 [============================>.] - ETA: 0s - loss: 0.6933 - accuracy: 0.4985
Epoch 3: val_accuracy did not improve from 0.50960
157/157 [==============================] - 0s 1ms/step
f1 score : 0.4942 AUC value : 0.5
469/469 [==============================] - 2s 4ms/step - loss: 0.6933 - accuracy: 0.4
Epoch 3: early stopping
<keras.callbacks.History at 0x7fc0595ec6d0>
```

```
train_acc_model_2 = model_2.evaluate(X_train, y_train, verbose=0)
test_acc_model_2 = model_2.evaluate(X_test, y_test, verbose=0)

train_accuracy_model_2 = train_acc_model_2[1]
test_accuracy_model_2 = test_acc_model_2[1]
train_loss_model_2 = train_acc_model_2[0]
```

```
test_loss_model_2 = test_acc_model_2[0]

print("Train data accuracy:",train_accuracy_model_2)
print("Test data accuracy:",test_accuracy_model_2)
```

```
Train data accuracy: 0.5019333362579346
Test data accuracy: 0.4941999912261963
```

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.

```
activation_fun_3  = "relu"
initializer_val_3 = HeUniform()
```

Saving...

```
fun_3,initializer_val_3)
um=0.9, nesterov=False,name='SGD')
model_3.compile(optimizer_3,loss='binary_crossentropy', metrics=['accuracy'])

metric_vals = Metrics()

tb = TensorBoard(log_dir='./Graph', histogram_freq=2,write_graph=True, write_images=Tru
tb.set_model(model_3)

cb  = [NanStop,LrValAccuracy,LrScheduler,CheckPoint,metric_vals,tb,EarlyStop]

model_3.fit(x = X_train, y = y_train,epochs=10,validation_data=(X_test,y_test),callback
```

```
Epoch 1/10
  1/469 [..............................] - ETA: 5:40 - loss: 0.9259 - accuracy: 0.437
463/469 [============================>.] - ETA: 0s - loss: 0.6760 - accuracy: 0.5723
Epoch 1: val_accuracy improved from 0.50960 to 0.57840, saving model to drive/My Driv
157/157 [==============================] - 0s 1ms/step
f1 score : 0.4942 AUC value : 0.5
469/469 [==============================] - 3s 5ms/step - loss: 0.6756 - accuracy: 0.5
Epoch 2/10
456/469 [============================>.] - ETA: 0s - loss: 0.6672 - accuracy: 0.5874
Epoch 2: val_accuracy improved from 0.57840 to 0.58500, saving model to drive/My Driv
157/157 [==============================] - 0s 1ms/step
f1 score : 0.4942 AUC value : 0.5
469/469 [==============================] - 2s 4ms/step - loss: 0.6674 - accuracy: 0.5
Epoch 3/10
460/469 [============================>.] - ETA: 0s - loss: 0.6645 - accuracy: 0.5916
Epoch 3: val_accuracy improved from 0.58500 to 0.59280, saving model to drive/My Driv
157/157 [==============================] - 0s 1ms/step
f1 score : 0.4942 AUC value : 0.5
```

```
469/469 [==============================] - 2s 4ms/step - loss: 0.6644 - accuracy: 0.5
Epoch 3: early stopping
<keras.callbacks.History at 0x7fc058dc3650>
```

```
train_acc_model_3 = model_3.evaluate(X_train, y_train, verbose=0)
test_acc_model_3 = model_3.evaluate(X_test, y_test, verbose=0)

train_accuracy_model_3 = train_acc_model_3[1]
test_accuracy_model_3 = test_acc_model_3[1]
train_loss_model_3 = train_acc_model_3[0]
test_loss_model_3 = test_acc_model_3[0]

print("Train data accuracy:",train_accuracy_model_3)
print("Test data accuracy:",test_accuracy_model_3)
```

```
Train data accuracy: 0.5935999751091003
Test data accuracy: 0.5928000211715698
```

Model-4

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


```
activation_fun_4 = "relu"
initializer_val_4 = RandomUniform(0,1)

# create model
model_4 = build_model(activation_fun_4,initializer_val_4)
optimizer_4 = SGD(lr=0.01, momentum=0.9, nesterov=False,name='adam')
model_4.compile(optimizer_4,loss='binary_crossentropy', metrics=['accuracy'])

metric_vals = Metrics()

tb = TensorBoard(log_dir='./Graph', histogram_freq=2,write_graph=True, write_images=Tru
tb.set_model(model_4)

cb  = [NanStop,LrValAccuracy,LrScheduler,CheckPoint,metric_vals,tb,EarlyStop]

model_4.fit(x = X_train, y = y_train,epochs=10,validation_data=(X_test,y_test),callback
```

```
Epoch 1/10
  1/469 [..............................] - ETA: 5:26 - loss: 14089.4746 - accuracy: 0
463/469 [============================>.] - ETA: 0s - loss: 31.1226 - accuracy: 0.4984
Epoch 1: val_accuracy did not improve from 0.59280
157/157 [==============================] - 0s 1ms/step
f1 score : 0.4942 AUC value : 0.5
469/469 [==============================] - 3s 4ms/step - loss: 30.7494 - accuracy: 0
Epoch 2/10
```

```
454/469 [============================>.] - ETA: 0s - loss: 0.6934 - accuracy: 0.4986
Epoch 2: val_accuracy did not improve from 0.59280
157/157 [==============================] - 0s 1ms/step
f1 score : 0.4942 AUC value : 0.5
469/469 [==============================] - 2s 3ms/step - loss: 0.6934 - accuracy: 0.4
Epoch 3/10
455/469 [============================>.] - ETA: 0s - loss: 0.6934 - accuracy: 0.4917
Epoch 3: val_accuracy did not improve from 0.59280
157/157 [==============================] - 0s 1ms/step
f1 score : 0.4942 AUC value : 0.5
469/469 [==============================] - 2s 4ms/step - loss: 0.6934 - accuracy: 0.4
Epoch 3: early stopping
<keras.callbacks.History at 0x7fc059586810>
```

```
train_acc_model_4 = model_4.evaluate(X_train, y_train, verbose=0)
test_acc_model_4 = model_4.evaluate(X_test, y_test, verbose=0)

train_accuracy_model_4 = train_acc_model_4[1]
test_accuracy_model_4 = test_acc_model_4[1]
train_loss_model_4 = train_acc_model_4[0]
test_loss_model_4 = test_acc_model_4[0]

print("Train data accuracy:",train_accuracy_model_4)
print("Test data accuracy:",test_accuracy_model_4)
```

```
Train data accuracy: 0.5262579346
Test data accuracy: 0.4941999912261963
```

```
pretty_table = pd.DataFrame(columns = ['Optimizer','Initializer','Activation_fun','Trai
pretty_table['Optimizer'] = ["SGD","SGD","SGD","Adam"]
pretty_table['Initializer'] = ["Random_Uniform","Random_Uniform","He_Uniform","Random_U
pretty_table['Activation_fun'] = [activation_fun_1,activation_fun_2,activation_fun_3,ac
pretty_table['Train-Loss'] = [train_loss_model_1,train_loss_model_2,train_loss_model_3,
pretty_table['Test-Loss'] = [test_loss_model_1,test_loss_model_2,test_loss_model_3,test
pretty_table['Train-Accuracy'] = [train_accuracy_model_1,train_accuracy_model_2,train_a
pretty_table['Test-Accuracy'] = [test_accuracy_model_1,test_accuracy_model_2,test_accur
pretty_table
```

	Optimizer	Initializer	Activation_fun	Train-Loss	Train-Accuracy	Test-Loss
0	SGD	Random_Uniform	tanh	0.694091	0.500667	0.692691
1	SGD	Random_Uniform	relu	0.693317	0.501933	0.693668
2	SGD	He_Uniform	relu	0.663075	0.593600	0.664845
3	Adam	Random_Uniform	relu	0.693146	0.501933	0.693249

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.

After careful observations on each model we can say that adam (final model) gives good accuracy results than other optimizers beacuase it can speed up the optimization by accelerating the gradient descent.

Instead of sigmoid, using an activation function such as ReLU to solve vanishing gradient problem in deep learning.

After clear observation model 2 and model 3 weight initializers with activation functions are most important consideration to get better accuracy of the model beacause combination can solve vanishing and exploding gradient problem in deep learning.

In model 2 we used random intializer with relu and SGD optimizer it gives around 50 percent accuracy. In model 3 we used he_uniform initializer with relu and SGD optimizer it gives around 59 percent accuracy.

plots:

https://tensorboard.dev/experiment/bE729AL8RdSqAMya