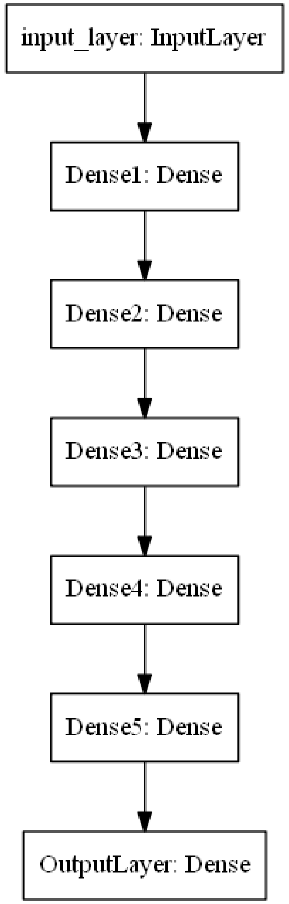
task pending

1. Download the data from [here (https://drive.google.com/file/d/15dCNcmKskcFVjs7R0ElQkR61Ex53uJpM/view?usp=sharing)](https://drive.google.com/file/d/15dCNcmKskcFVjs7R0ElQkR61Ex53uJpM/view?usp=sharing). 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)

In [1]:

```python
import numpy as np
import pandas as pd
import tensorflow as tf

from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import LearningRateScheduler
from tensorflow.keras.layers import Dense,Input,Activation

import random as rn
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from keras import backend
import pdb
import shutil
from sklearn.metrics import f1_score

from itertools import combinations
import os
import datetime

from sklearn.preprocessing import label_binarize


```

In [2]:

```python
import os
import datetime
```

In [3]:

```python
# %load_ext tensorboard

```

In [4]:

```python
data = pd.read_csv('data.csv')

```

In [5]:

```
data.head()
```

Out[5]:

	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 [6]:

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

data.shape, y.shape
```

Out[6]:

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

In [7]:

```
# train, test split
from sklearn.model_selection import train_test_split

# label_binarize for 2 class ref: https://stackoverflow.com/questions/31947140/sklearn-labelbinarizer-returns-vector-when-there-are-2-classes
y = np.array([[1,0] if l==0 else [0,1] for l in y])  # == MultiLabelBinarizer()

X_train, X_test, Y_train, Y_test = train_test_split(data, y, test_size= 0.20, stratify = y)


X_train.shape, X_test.shape, Y_train.shape
```

Out[7]:

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

In [8]:

```
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

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

In [9]:

```
X_train = np.array(X_train)
Y_train = np.array(Y_train)
```

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.
```

loading tensoreboard and removing file from logs

In [10]:

```
# there are other ways of doing this: https://www.dlology.com/blog/quick-guide-to-run-tensorboard-in-google-colab/ you can try this way also
%load_ext tensorboard
# Clear any logs from previous runs
# shutil.rmtree(r"C:\Users\ashutosh tiwari\Documents\ML_programs\assignment_aai\20_callbacks_dl\logs" )

```

In [ ]:

```

```

defining class for callbacks

In [11]:

```
# terminating model if we get NAN value in loss or weights

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

        # checking weights
        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]):
                self.model.stop_training = True

#         print("not contain any NAN value")

```

In [12]:

```
# change Learning Rate with conditions

class changeInLearningRate(tf.keras.callbacks.Callback):
    def __init__(self,validation_data):
        # self.x_test = validation_data[0]
        # self.y_test= validation_data[1]
        self.validation = []

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

    def on_epoch_begin(self, epoch, logs={}):
        print('list of validation -',np.round(self.validation,4) )

    def on_epoch_end(self, epoch, logs={}):
        self.validation.append(logs.get('val_accuracy'))

    # change Learning Rate
    def schedule_lr(self, epoch, lr):

        # condiont 1 : decaying learning rate by 5%
        rng = [i for i in range(0, epoch+1, 2)]
        if epoch in rng and epoch != 0: # become true at 3, 6, 9, 12 ...
            per = lr* (5/100)  # decaying learning rate by 5%
            lr = lr - per


        # condiont 2 :decaying learning rate by 10% if validation acc is less than previous val_accuracy
        if epoch >1  and self.validation[epoch-1] < self.validation[epoch-2] :  # decaying after 2 epoch
            percen = lr* (10/100)
            lr1  = lr - percen  # decaying 10%
            return lr1

        return lr


```

In [13]:

```
# Printing costum made accuracy on validation data

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

    def __init__(self,validation_data):
        self.x_test = validation_data[0]
        self.y_test= validation_data[1]


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

        # # we can get a list of all predicted values at the end of the epoch
        # # we can use these predicted value and the true values to calculate any custom evaluation score if it is needed for our model
        # # Here we are taking log of all true positives and then taking average of it
        self.y_pred= self.model.predict(self.x_test)
        self.y_label_pred=np.argmax(self.y_pred,axis=1)

        #computing acu score for each class and f1 score
        auc_n_classes = roc_auc_score(self.y_test[:,1], self.y_pred[:,1])

        #calcualting  f1_score through sklearn
        y_pred2 = [1 if x >=0.5 else 0 for x in self.y_pred[:,1]]
#         pdb.set_trace()
        f1 = f1_score( self.y_test[:,1], y_pred2, average = "micro")

        print(' AUC Score: ',auc_n_classes, 'f1 score', f1)


# history_own=LossHistory(validation_data=[X_test,Y_test])

```

creating model

In [14]:

```
#Model 1 -
tf.random.set_seed(10)

def create_callbacks():
    #create a callback list of 4 callback

    earlystop = EarlyStopping(monitor='val_accuracy', min_delta=0.010, patience=2, verbose=1)
    # saving model
    filepath="model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
    checkpoint = ModelCheckpoint(filepath=filepath, monitor='accuracy', verbose=1, save_best_only=True, mode='auto')

    # custom changing learning rate
    obj_lr = changeInLearningRate(validation_data=[X_test,Y_test])
    lrschedule = LearningRateScheduler(obj_lr.schedule_lr, verbose=0.1)

    # custom accuracy
    history_own=LossHistory(validation_data=[X_test,Y_test])
    # tuncating program if NAN in loss val
    truncate_if_Nan = TerminateNaN()

    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)

    # here we are creating a list with all the callbacks we want
    callback_list = [obj_lr,lrschedule, checkpoint,history_own ,earlystop ,truncate_if_Nan, tensorboard_callback] #  earlystop

    return callback_list



def create_model1():
    #Input layer
    input_layer = Input(shape=(2))
    #Dense hidden layer
    layer1 = Dense(60,activation='tanh',kernel_initializer=tf.keras.initializers.RandomNormal(mean =0, stddev=1 ))(input_layer)
    #Dense hidden layer
    layer2 = Dense(40,activation='tanh',kernel_initializer=tf.keras.initializers.RandomNormal(mean =0, stddev=1 ))(layer1)
    #Dense hidden layer
    layer3 = Dense(40,activation='tanh',kernel_initializer=tf.keras.initializers.RandomNormal(mean =0, stddev=1 ))(layer2)
    #Dense hidden layer
    layer4 = Dense(40,activation='tanh',kernel_initializer=tf.keras.initializers.RandomNormal(mean =0, stddev=1 ))(layer3)
    #Dense hidden layer
    layer5 = Dense(10,activation='tanh',kernel_initializer=tf.keras.initializers.RandomNormal(mean =0, stddev=1 ))(layer4)
    # output layer
    output = Dense(2,activation='softmax',kernel_initializer=tf.keras.initializers.RandomNormal(mean =0, stddev=1 ))(layer5)


    #Creating a model
    model = Model(inputs=input_layer,outputs=output)

    return model

```

In [15]:

```python
tf.keras.backend.clear_session()  # For easy reset of notebook state.
tf.random.set_seed(10)
callback_list = None
callback_list = create_callbacks()

model = None
model = create_model1()



```

In [16]:

```python
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 2)]               0         
                                                                 
 dense (Dense)               (None, 60)                180       
                                                                 
 dense_1 (Dense)             (None, 40)                2440      
                                                                 
 dense_2 (Dense)             (None, 40)                1640      
                                                                 
 dense_3 (Dense)             (None, 40)                1640      
                                                                 
 dense_4 (Dense)             (None, 10)                410       
                                                                 
 dense_5 (Dense)             (None, 2)                 22        
                                                                 
=================================================================
Total params: 6,332
Trainable params: 6,332
Non-trainable params: 0
_________________________________________________________________
```

In [17]:

```
model.compile(tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.5),loss='categorical_crossentropy',metrics=['accuracy'])
model.fit(X_train,Y_train,epochs=10,validation_data=(X_test,Y_test),batch_size=64, callbacks = callback_list) #callbacks = callback_list

```

```
list of validation - []

Epoch 1: LearningRateScheduler setting learning rate to 0.009999999776482582.
Epoch 1/10
  1/250 [..............................] - ETA: 3:10 - loss: 1.5202 - accuracy: 0.5625WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch tim
e (batch time: 0.0016s vs `on_train_batch_end` time: 0.0030s). Check your callbacks.
241/250 [===========================>..] - ETA: 0s - loss: 0.8097 - accuracy: 0.5596
Epoch 1: accuracy improved from -inf to 0.56012, saving model to model_save\weights-01-0.5713.hdf5
125/125 [==============================] - 0s 2ms/step
 AUC Score:  0.6160955 f1 score 0.57125
250/250 [==============================] - 2s 6ms/step - loss: 0.8051 - accuracy: 0.5601 - val_loss: 0.7075 - val_accuracy: 0.5713 - lr: 0.0100
list of validation - [0.5713]

Epoch 2: LearningRateScheduler setting learning rate to 0.009999999776482582.
Epoch 2/10
226/250 [==========================>...] - ETA: 0s - loss: 0.6871 - accuracy: 0.5841
Epoch 2: accuracy improved from 0.56012 to 0.58438, saving model to model_save\weights-02-0.6055.hdf5
125/125 [==============================] - 0s 1ms/step
 AUC Score:  0.637364375 f1 score 0.6055
250/250 [==============================] - 1s 4ms/step - loss: 0.6873 - accuracy: 0.5844 - val_loss: 0.6800 - val_accuracy: 0.6055 - lr: 0.0100
list of validation - [0.5713 0.6055]

Epoch 3: LearningRateScheduler setting learning rate to 0.009499999787658453.
Epoch 3/10
224/250 [=========================>....] - ETA: 0s - loss: 0.6656 - accuracy: 0.6044
Epoch 3: accuracy improved from 0.58438 to 0.60544, saving model to model_save\weights-03-0.5925.hdf5
125/125 [==============================] - 0s 2ms/step
 AUC Score:  0.645317375 f1 score 0.5925
250/250 [==============================] - 1s 5ms/step - loss: 0.6657 - accuracy: 0.6054 - val_loss: 0.6684 - val_accuracy: 0.5925 - lr: 0.0095
list of validation - [0.5713 0.6055 0.5925]

Epoch 4: LearningRateScheduler setting learning rate to 0.008549999725073577.
Epoch 4/10
245/250 [============================>.] - ETA: 0s - loss: 0.6593 - accuracy: 0.6157
Epoch 4: accuracy improved from 0.60544 to 0.61575, saving model to model_save\weights-04-0.6110.hdf5
125/125 [==============================] - 0s 2ms/step
 AUC Score:  0.656123 f1 score 0.611
250/250 [==============================] - 1s 4ms/step - loss: 0.6589 - accuracy: 0.6158 - val_loss: 0.6673 - val_accuracy: 0.6110 - lr: 0.0085
Epoch 4: early stopping
```

Out[17]:

```
<keras.callbacks.History at 0x1e15681a2b0>
```

TensorBoard

In [18]:
```
%tensorboard --logdir logs
```

```
Reusing TensorBoard on port 6006 (pid 11012), started 1 day, 11:35:29 ago. (Use '!kill 11012' to kill it.)
```

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 [19]:

```
# pip install shutil
```

In [21]:

```
# Clear any logs from previous runs

shutil.rmtree(r'C:\Users\ashutosh tiwari\Documents\ML_programs\assignment_aai\20_callbacks_dl\logs')
```

In [20]:

```
#Model 2 -


def create_model():
    return tf.keras.models.Sequential([
    tf.keras.layers.Input((2)),
    tf.keras.layers.Dense(160, activation='relu',kernel_initializer = tf.keras.initializers.RandomUniform(minval=0, maxval=1, seed=21)),
    tf.keras.layers.Dense(100, activation='relu', kernel_initializer = tf.keras.initializers.RandomUniform(minval=0, maxval=1, seed=22)),
    tf.keras.layers.Dense(70, activation='relu', kernel_initializer = tf.keras.initializers.RandomUniform(minval=0, maxval=1, seed=23)),
    tf.keras.layers.Dense(64, activation='relu', kernel_initializer = tf.keras.initializers.RandomUniform(minval=0, maxval=1, seed=24)),
    tf.keras.layers.Dense(54, activation='relu', kernel_initializer = tf.keras.initializers.RandomUniform(minval=0, maxval=1, seed=25)),
    tf.keras.layers.Dense(2, activation='softmax')
    ])



tf.keras.backend.clear_session()  # For easy reset of notebook state.

callback_list2 = None
callback_list2 = create_callbacks()

# creating model building
model2 = create_model()
# model.summary()


```

In [21]:

```


model2.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.5),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# training model ,callbacks=callback_list
model.fit(X_train,Y_train,epochs=10,validation_data=(X_test,Y_test),batch_size=64, callbacks=callback_list2)

```

```
list of validation - []

Epoch 1: LearningRateScheduler setting learning rate to 0.008549999445676804.
Epoch 1/10
  1/250 [..............................] - ETA: 28s - loss: 0.6158 - accuracy: 0.6406WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch ti
me (batch time: 0.0016s vs `on_train_batch_end` time: 0.0035s). Check your callbacks.
248/250 [============================>.] - ETA: 0s - loss: 0.6619 - accuracy: 0.6131
Epoch 1: accuracy improved from -inf to 0.61350, saving model to model_save\weights-01-0.6192.hdf5
125/125 [==============================] - 0s 2ms/step
 AUC Score:  0.661776 f1 score 0.61925
250/250 [==============================] - 1s 5ms/step - loss: 0.6622 - accuracy: 0.6135 - val_loss: 0.6649 - val_accuracy: 0.6192 - lr: 0.0085
list of validation - [0.6192]

Epoch 2: LearningRateScheduler setting learning rate to 0.008549999445676804.
Epoch 2/10
233/250 [==========================>...] - ETA: 0s - loss: 0.6580 - accuracy: 0.6214
Epoch 2: accuracy improved from 0.61350 to 0.62181, saving model to model_save\weights-02-0.6192.hdf5
125/125 [==============================] - 0s 2ms/step
 AUC Score:  0.66380225 f1 score 0.61925
```

In [24]:

```
%tensorboard --logdir logs

```

```
Reusing TensorBoard on port 6006 (pid 11012), started 1 day, 11:34:10 ago. (Use '!kill 11012' to kill it.)
```

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 [23]:

```
#Model 3 -

def create_model():
    return tf.keras.models.Sequential([
    tf.keras.layers.Input(2),
    tf.keras.layers.Dense(164, activation='relu',kernel_initializer = tf.keras.initializers.HeUniform(seed=21)),
    tf.keras.layers.Dense(100, activation='relu', kernel_initializer = tf.keras.initializers.HeUniform(seed=22)),
    tf.keras.layers.Dense(90, activation='relu',kernel_initializer = tf.keras.initializers.HeUniform(seed=23)),
    tf.keras.layers.Dense(80, activation='relu', kernel_initializer = tf.keras.initializers.HeUniform(seed=24)),
    tf.keras.layers.Dense(64, activation='relu',kernel_initializer = tf.keras.initializers.HeUniform(seed=25)),
    tf.keras.layers.Dense(2, activation='softmax',kernel_initializer = tf.keras.initializers.HeUniform(seed=26))
    ])


tf.keras.backend.clear_session()  # For easy reset of notebook state.

callback_list = None
callback_list3 = create_callbacks()

# creating model building
model3 = create_model()
model3.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 164)               492       
                                                                 
 dense_1 (Dense)             (None, 100)               16500     
                                                                 
 dense_2 (Dense)             (None, 90)                9090      
                                                                 
 dense_3 (Dense)             (None, 80)                7280      
                                                                 
 dense_4 (Dense)             (None, 64)                5184      
                                                                 
 dense_5 (Dense)             (None, 2)                 130       
                                                                 
=================================================================
Total params: 38,676
Trainable params: 38,676
Non-trainable params: 0
_________________________________________________________________
```

In [24]:

```

# #create a callback list of 4 callback

# earlystop = EarlyStopping(monitor='val_loss', min_delta=0.010, patience=3, verbose=1)
# # saving model
# filepath="model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
# checkpoint = ModelCheckpoint(filepath=filepath, monitor='accuracy', verbose=1, save_best_only=True, mode='auto')

# # custom changing learning rate
# obj_lr3 = changeInLearningRate(validation_data=[X_test,Y_test])
# lrschedule = LearningRateScheduler(obj_lr3.schedule_lr, verbose=0.1)

# # custom accuracy
# history_own3=LossHistory(validation_data=[X_test,Y_test])
# # tuncating program if NAN in loss val
# truncate_if_Nan = TerminateNaN()

# # here we are creating a list with all the callbacks we want
# callback_list = [obj_lr3,lrschedule, checkpoint,history_own3 ,earlystop ,truncate_if_Nan, tensorboard_callback] #  earlystop

# 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)


model.compile(tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.5),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# training model
model.fit(X_train,Y_train,epochs=5,validation_data=(X_test,Y_test),batch_size=64,callbacks=callback_list3) # callbacks=callback_list
```

```
list of validation - []

Epoch 1: LearningRateScheduler setting learning rate to 0.009999999776482582.
Epoch 1/5
  1/250 [..............................] - ETA: 4:38 - loss: 0.6467 - accuracy: 0.6406WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch tim
e (batch time: 0.0016s vs `on_train_batch_end` time: 0.0064s). Check your callbacks.
240/250 [===========================>..] - ETA: 0s - loss: 0.6515 - accuracy: 0.6132
Epoch 1: accuracy improved from -inf to 0.61169, saving model to model_save\weights-01-0.5980.hdf5
125/125 [==============================] - 1s 3ms/step
 AUC Score:  0.66547825 f1 score 0.598
250/250 [==============================] - 4s 10ms/step - loss: 0.6523 - accuracy: 0.6117 - val_loss: 0.6466 - val_accuracy: 0.5980 - lr: 0.0100
list of validation - [0.598]

Epoch 2: LearningRateScheduler setting learning rate to 0.009999999776482582.
Epoch 2/5
241/250 [===========================>..] - ETA: 0s - loss: 0.6537 - accuracy: 0.6144
Epoch 2: accuracy improved from 0.61169 to 0.61537, saving model to model_save\weights-02-0.6037.hdf5
125/125 [==============================] - 0s 3ms/step
 AUC Score:  0.66261825 f1 score 0.60375
250/250 [==============================] - 2s 7ms/step - loss: 0.6529 - accuracy: 0.6154 - val_loss: 0.6637 - val_accuracy: 0.6037 - lr: 0.0100
list of validation - [0.598  0.6037]

Epoch 3: LearningRateScheduler setting learning rate to 0.009499999787658453.
Epoch 3/5
238/250 [===========================>..] - ETA: 0s - loss: 0.6491 - accuracy: 0.6211
Epoch 3: accuracy improved from 0.61537 to 0.62094, saving model to model_save\weights-03-0.6252.hdf5
125/125 [==============================] - 0s 3ms/step
 AUC Score:  0.6725352499999999 f1 score 0.62525
250/250 [==============================] - 2s 7ms/step - loss: 0.6496 - accuracy: 0.6209 - val_loss: 0.6510 - val_accuracy: 0.6252 - lr: 0.0095
list of validation - [0.598  0.6037 0.6252]

Epoch 4: LearningRateScheduler setting learning rate to 0.009499999694526196.
Epoch 4/5
240/250 [===========================>..] - ETA: 0s - loss: 0.6520 - accuracy: 0.6170
Epoch 4: accuracy did not improve from 0.62094
125/125 [==============================] - 0s 3ms/step
 AUC Score:  0.680649875 f1 score 0.6325
250/250 [==============================] - 2s 7ms/step - loss: 0.6509 - accuracy: 0.6179 - val_loss: 0.6429 - val_accuracy: 0.6325 - lr: 0.0095
list of validation - [0.598  0.6037 0.6252 0.6325]

Epoch 5: LearningRateScheduler setting learning rate to 0.009024999709799886.
Epoch 5/5
239/250 [===========================>..] - ETA: 0s - loss: 0.6467 - accuracy: 0.6253
Epoch 5: accuracy improved from 0.62094 to 0.62612, saving model to model_save\weights-05-0.6162.hdf5
125/125 [==============================] - 0s 3ms/step
 AUC Score:  0.6867749999999999 f1 score 0.61625
250/250 [==============================] - 2s 8ms/step - loss: 0.6467 - accuracy: 0.6261 - val_loss: 0.6388 - val_accuracy: 0.6162 - lr: 0.0090
Epoch 5: early stopping
```

Out[24]:
```
<keras.callbacks.History at 0x173d70c0730>
```

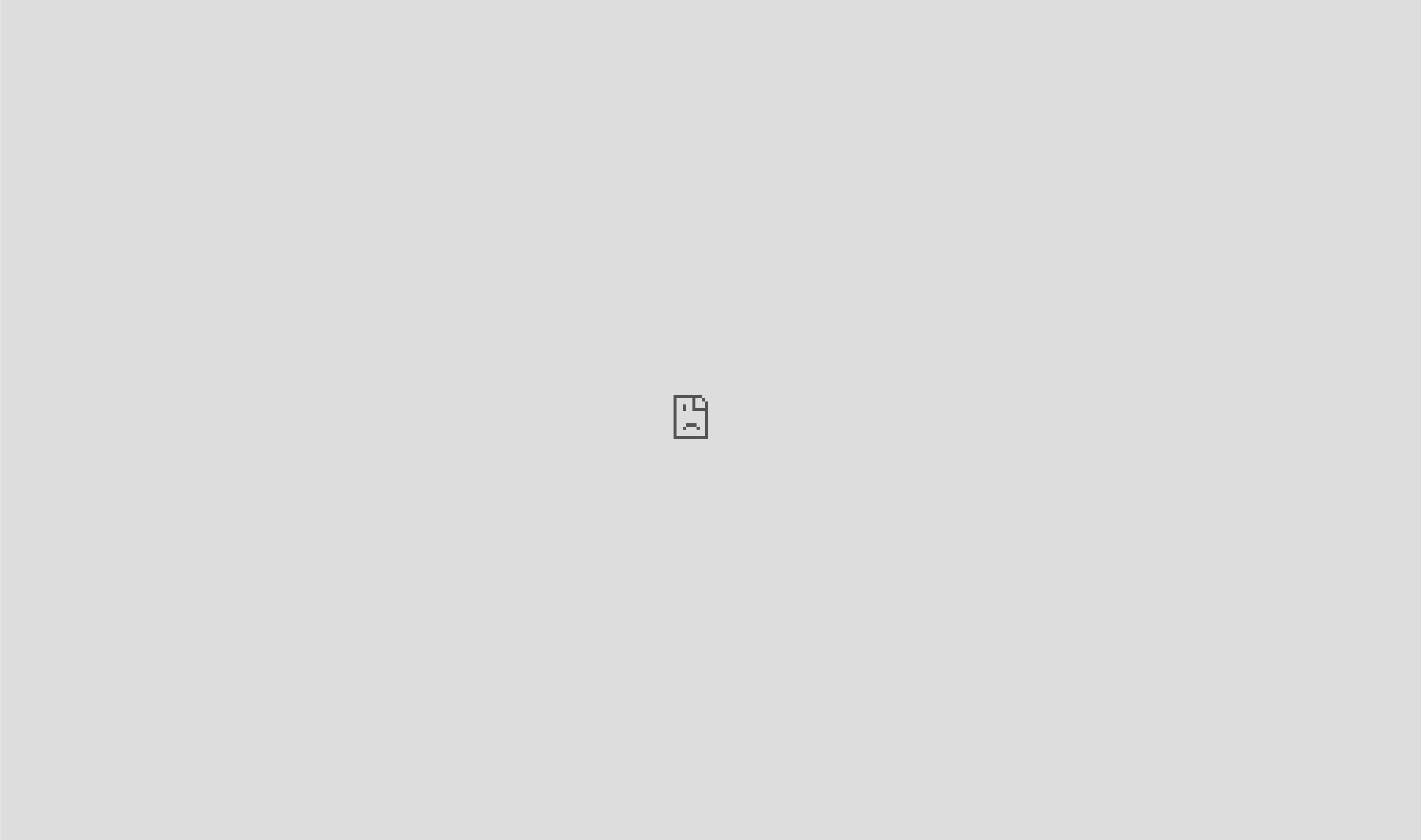
In [25]:
```
# Clear any logs from previous runs
!rm -rf ./logs/
```

```
'rm' is not recognized as an internal or external command,
operable program or batch file.
```

In [26]:

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

```
Reusing TensorBoard on port 6006 (pid 11012), started 0:00:18 ago. (Use '!kill 11012' to kill it.)
```



observation

```
- getting NAN value because of inisilizer
```

Model-4

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

In [27]:

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

```
'rm' is not recognized as an internal or external command,
operable program or batch file.
```

In [28]:

```
#Model 4 -
tf.random.set_seed(110)
def create_model4():
    return tf.keras.models.Sequential([
    tf.keras.layers.Input(2),
    tf.keras.layers.Dense(528, activation='relu',kernel_initializer = tf.keras.initializers.HeUniform(seed=31)),
    tf.keras.layers.Dense(528, activation='relu',kernel_initializer = tf.keras.initializers.HeUniform(seed=32)),

    tf.keras.layers.Dense(2, activation='softmax',kernel_initializer = tf.keras.initializers.HeUniform(seed=33))
    ])



# creating model building
model = None
model = create_model4()
# model.summary()




```

In [29]:

```python
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
callback_list4 = create_callbacks()

# training model
model.fit(X_train,Y_train,epochs=10,validation_data=(X_test,Y_test),batch_size=64 ,callbacks=callback_list4) # ,callbacks=callback_list

```

```
list of validation - []

Epoch 1: LearningRateScheduler setting learning rate to 0.009999999776482582.
Epoch 1/10
  4/250 [..............................] - ETA: 4s - loss: 0.7502 - accuracy: 0.4453   WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch ti
me (batch time: 0.0138s vs `on_train_batch_end` time: 0.2007s). Check your callbacks.
249/250 [============================>.] - ETA: 0s - loss: 0.6944 - accuracy: 0.5213
Epoch 1: accuracy improved from -inf to 0.52181, saving model to model_save\weights-01-0.5335.hdf5
125/125 [==============================] - 1s 4ms/step
 AUC Score:  0.663851625 f1 score 0.5335
250/250 [==============================] - 7s 15ms/step - loss: 0.6942 - accuracy: 0.5218 - val_loss: 0.6870 - val_accuracy: 0.5335 - lr: 0.0100
list of validation - [0.5335]

Epoch 2: LearningRateScheduler setting learning rate to 0.009999999776482582.
Epoch 2/10
245/250 [============================>.] - ETA: 0s - loss: 0.6708 - accuracy: 0.5807
Epoch 2: accuracy improved from 0.52181 to 0.58056, saving model to model_save\weights-02-0.5922.hdf5
125/125 [==============================] - 1s 4ms/step
 AUC Score:  0.685372375 f1 score 0.59225
250/250 [==============================] - 3s 13ms/step - loss: 0.6708 - accuracy: 0.5806 - val_loss: 0.6649 - val_accuracy: 0.5922 - lr: 0.0100
list of validation - [0.5335 0.5922]

Epoch 3: LearningRateScheduler setting learning rate to 0.009499999787658453.
Epoch 3/10
246/250 [============================>.] - ETA: 0s - loss: 0.6558 - accuracy: 0.6152
Epoch 3: accuracy improved from 0.58056 to 0.61406, saving model to model_save\weights-03-0.6595.hdf5
125/125 [==============================] - 1s 4ms/step
 AUC Score:  0.725004 f1 score 0.6595
250/250 [==============================] - 3s 12ms/step - loss: 0.6563 - accuracy: 0.6141 - val_loss: 0.6412 - val_accuracy: 0.6595 - lr: 0.0095
list of validation - [0.5335 0.5922 0.6595]

Epoch 4: LearningRateScheduler setting learning rate to 0.009499999694526196.
Epoch 4/10
247/250 [============================>.] - ETA: 0s - loss: 0.6459 - accuracy: 0.6261
Epoch 4: accuracy improved from 0.61406 to 0.62706, saving model to model_save\weights-04-0.6087.hdf5
125/125 [==============================] - 0s 3ms/step
 AUC Score:  0.6997822499999999 f1 score 0.60875
250/250 [==============================] - 3s 12ms/step - loss: 0.6455 - accuracy: 0.6271 - val_loss: 0.6566 - val_accuracy: 0.6087 - lr: 0.0095
list of validation - [0.5335 0.5922 0.6595 0.6087]

Epoch 5: LearningRateScheduler setting learning rate to 0.008122499738819898.
Epoch 5/10
241/250 [===========================>..] - ETA: 0s - loss: 0.6330 - accuracy: 0.6467
Epoch 5: accuracy improved from 0.62706 to 0.64688, saving model to model_save\weights-05-0.6210.hdf5
125/125 [==============================] - 0s 3ms/step
 AUC Score:  0.7052863749999999 f1 score 0.621
250/250 [==============================] - 3s 10ms/step - loss: 0.6332 - accuracy: 0.6469 - val_loss: 0.6435 - val_accuracy: 0.6210 - lr: 0.0081
Epoch 5: early stopping
```

```
Out[29]: <keras.callbacks.History at 0x173d844c250>
```

obeservation

```
- train accuracy 0.9561
- avg AUC Score:  0.9939327466444144, f1 score : 0.9381307745880598 on validation data
```

In [30]:

```
# tensor board
%tensorboard --logdir logs
```

```
Reusing TensorBoard on port 6006 (pid 11012), started 0:00:37 ago. (Use '!kill 11012' to kill it.)
```

summary

link - https://docs.google.com/document/d/18caMBJdSJ-Q8quasyyI2EEk8Y_UjiGyDfGU0frpim4w/edit?usp=sharing (https://docs.google.com/document/d/18caMBJdSJ-Q8quasyyI2EEk8Y_UjiGyDfGU0frpim4w/edit?usp=sharing)