Notebook Resources

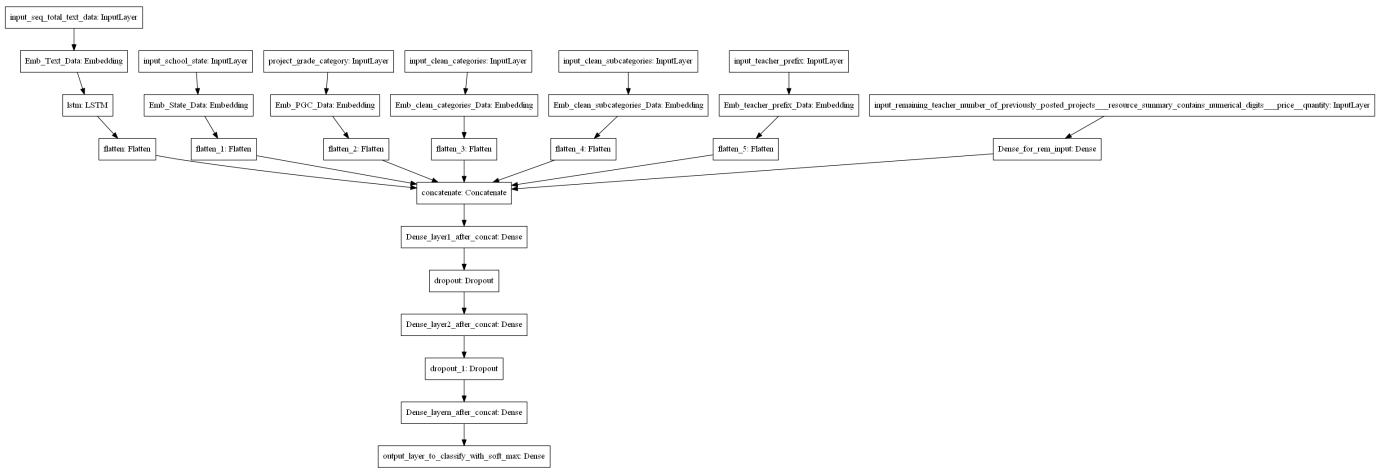
Assignment : 14

```
1. You can work with preprocessed_data.csv for the assignment. You can get the da
2. Load the data in your notebook.
3. After step 2 you have to train 3 types of models as discussed below.
4. For all the model use 'auc' as a metric. check this and this for using auc as
5. You are free to choose any number of layers/hiddden units but you have to use
6. You can use any one of the optimizers and choice of Learning rate and momentum
7. For all the model's use TensorBoard and plot the Metric value and Loss with ep
8. Make sure that you are using GPU to train the given models.
```

```
#you can use gdown modules to import dataset for the assignment
#for importing any file from drive to Colab you can write the syntax as !gdown
#you can run the below cell to import the required preprocessed data.csv file a
```

Model-1

Build and Train deep neural network as shown below



ref: https://i.imgur.com/w395Yk9.png

- **Input_seq_total_text_data** --- You have to give Total text data columns. After this use the Embedding layer to get word vectors. Use given predefined glove word vectors, don't train any word vectors. After this use LSTM and get the LSTM output and Flatten that output.
- **Input_school_state** --- Give 'school_state' column as input to embedding layer and Train the Keras Embedding layer.

- **Project_grade_category** --- Give 'project_grade_category' column as input to embedding layer and Train the Keras Embedding layer.
- **Input_clean_categories** --- Give 'input_clean_categories' column as input to embedding layer and Train the Keras Embedding layer.
- **Input_clean_subcategories** --- Give 'input_clean_subcategories' column as input to embedding layer and Train the Keras Embedding layer.
- **Input_clean_subcategories** --- Give 'input_teacher_prefix' column as input to embedding layer and Train the Keras Embedding layer.
- **Input_remaining_teacher_number_of_previously_posted_projects._resource_summary_contains_numerical_digits._price._quantity** ---concatenate remaining columns and add a Dense layer after that.

Below is an example of embedding layer for a categorical columns. In below code all are dummy values, we gave only for referance.

```
# # https://stats.stackexchange.com/questions/270546/how-does-keras-embedding-l
# input_layer = Input(shape=(n,))
# embedding = Embedding(no_1, no_2, input_length=n)(input_layer)
# flatten = Flatten()(embedding)
```

1. Go through this blog, if you have any doubt on using predefined Embedding values in Embedding layer - https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/

2. Please go through this link https://keras.io/getting-started/functional-api-guide/ and check the 'Multi-input and multi-output models' then you will get to know how to give multiple inputs.

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

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

Model-1

```
# import all the libraries
#make sure that you import your libraries from tf.keras and not just keras
import tensorflow as tf
```

```
# path = "/Users/yamasanimanoj-kumarreddy/Documents/AAIC/Named_Assignments/LSTM
path = "/content/drive/MyDrive/Named_Assignments/LSTM_DonorsChoose"
# path = "/content/drive/MyDrive/DeepLearn/LSTM_DonorsChoose"
```

```
#read the csv file
import pandas as pd

data = pd.read_csv(path+'/preprocessed_data.csv')
data.head(1)
```

	school_state	teacher_prefix	project_grade_category	teacher_number_of_pre
0	ca	mrs	grades_prek_2	

```
Y=data['project_is_approved']
X=data.drop('project_is_approved',axis=1)
```

```
print(Y.value_counts())
```

```
1    92706
0    16542
Name: project_is_approved, dtype: int64
```

```
import matplotlib.pyplot as plt
import numpy as np

pos_class = Y.value_counts()[1]/sum(Y.value_counts())
neg_class = Y.value_counts()[0]/sum(Y.value_counts())
y = np.array([neg_class, pos_class])
print(y)
mylabels = ["Negative","Positive"]

plt.pie(y, labels = mylabels)
plt.legend(title = "Distribution of classes")
plt.show()
```

```
[0.15141696 0.84858304]
```

From the above plot we can conclude that 85% of the times a project is approved.

From the above pie chart, we can observe that given data is highly imbalanced data set with 85% points of positive class and remaining 15% of points as negative class.

```
# Scaling by total/2 helps keep the loss to a similar magnitude.
# The sum of the weights of all examples stays the same.

total = sum(Y.value_counts())
weight_for_0 = (1 / Y.value_counts()[0]) * (total / 2.0)
weight_for_1 = (1 / Y.value_counts()[1]) * (total / 2.0)

class_weight = {0: weight_for_0, 1: weight_for_1}

print('Weight for class 0: {:.2f}'.format(weight_for_0))
print('Weight for class 1: {:.2f}'.format(weight_for_1))
```

```
Weight for class 0: 3.30
Weight for class 1: 0.59
```

perform stratified train test split on the dataset

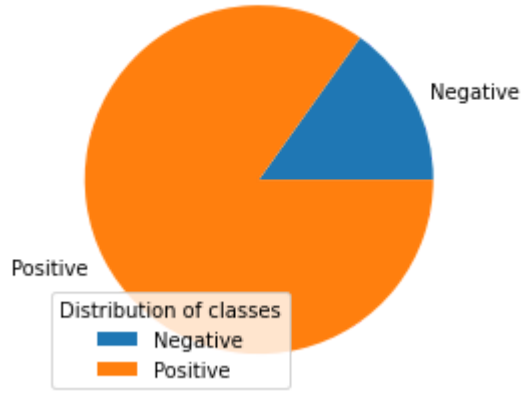
```
from sklearn.model_selection import train_test_split

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.33, stratify=Y
```

```
print(X_train.shape,Y_train.shape,X_test.shape,Y_test.shape,end=" ")
```

```
(73196, 8) (73196,) (36052, 8) (36052,)
```

```
import matplotlib.pyplot as plt
import numpy as np

pos_class = Y_train.value_counts()[1]/sum(Y_train.value_counts())
neg_class = Y_train.value_counts()[0]/sum(Y_train.value_counts())
y = np.array([neg_class, pos_class])
print(y)
mylabels = ["Negative","Positive"]

plt.pie(y, labels = mylabels)
plt.legend(title = "Distribution of classes")
plt.show()
```

```
[0.15141538 0.84858462]
```



We got similar distribution of points in class due to stratified sampling.

1.1 Text Vectorization

```
#since the data is already preprocessed, we can directly move to vectorization
#first we will vectorize the text data
#for vectorization of text data in deep learning we use tokenizer, you can go
# https://www.kdnuggets.com/2020/03/tensorflow-keras-tokenization-text-data-pre
#https://stackoverflow.com/questions/51956000/what-does-keras-tokenizer-method-
# after text vectorization you should get train_padded_docs and test_padded_do
```

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train['essay'])
vocab_size = len(tokenizer.word_index) + 1
print(vocab_size)
# integer encode the documents
train_encoded_docs = tokenizer.texts_to_sequences(X_train['essay'])
test_encoded_docs = tokenizer.texts_to_sequences(X_test['essay'])
```

```
48232
```

```
max_length = max([len(seq) for seq in train_encoded_docs])
print(max_length)
train_padded_docs = pad_sequences(train_encoded_docs, maxlen=max_length, paddi
test_padded_docs = pad_sequences(test_encoded_docs, maxlen=max_length, padding=
print(train_padded_docs.shape)
print(test_padded_docs.shape)
print(train_padded_docs[0][-10:])
print(test_padded_docs[0][-10:])
```

```
339
(73196, 339)
(36052, 339)
```

```
[0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0]
```

```
#after getting the padded_docs you have to use predefined glove vectors to get
# we will be storing this data in form of an embedding matrix and will use it w
# Please go through following blog's 'Example of Using Pre-Trained GloVe Embedd
# https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-ke
import pickle
import numpy as np

with open(path+'/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

```
embedding_matrix = np.zeros((vocab_size, 300))
for word, i in tokenizer.word_index.items():
    embedding_vector = model.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
print(embedding_matrix[1][:5])
print(embedding_matrix.shape)
print("The number of tokens present in this vocabulary are", embedding_matrix.s
```

```
[ 0.15243  -0.16945  -0.022748 -0.25051  -0.15213 ]
(48232, 300)
The number of tokens present in this vocabulary are 48232
```

1.2 Categorical feature Vectorization

```
# for model 1 and model 2, we have to assign a unique number to each feature in
# you can either use tokenizer,label encoder or ordinal encoder to perform the
# label encoder gives an error for 'unseen values' (values present in test but
# handle unseen values with label encoder - https://stackoverflow.com/a/568763
# ordinal encoder also gives error with unseen values but you can use modify ha
# documentation of ordianl encoder https://scikit-learn.org/stable/modules/gene
# after categorical feature vectorization you will have column_train_data and c

```

School_State

```
from sklearn.preprocessing import OrdinalEncoder
encoder = OrdinalEncoder(handle_unknown = 'use_encoded_value',unknown_value=-1)
train_school_state = encoder.fit_transform(X_train['school_state'].to_numpy().r
test_school_state = encoder.transform(X_test['school_state'].to_numpy().reshape
no_of_school_state = len(encoder.categories_[0])
print("No of categories present in school state are ",no_of_school_state)
print(train_school_state.shape)
print(test_school_state.shape)
```

```
No of categories present in school state are  51
(73196, 1)
(36052, 1)
```

project_grade_category

```
from sklearn.preprocessing import OrdinalEncoder
encoder = OrdinalEncoder(handle_unknown = 'use_encoded_value',unknown_value=-1)
train_project_grade = encoder.fit_transform(X_train['project_grade_category'].t
test_project_grade = encoder.transform(X_test['project_grade_category'].to_nump
no_of_project_grade = len(encoder.categories_[0])
print("No of categories present in project grade are ",no_of_project_grade)
print(train_project_grade.shape)
print(test_project_grade.shape)
```

```
No of categories present in project grade are  4
(73196, 1)
(36052, 1)
```

clean_categories

```
from sklearn.preprocessing import OrdinalEncoder
encoder = OrdinalEncoder(handle_unknown = 'use_encoded_value',unknown_value=-1)
train_clean_categories = encoder.fit_transform(X_train['clean_categories'].to_n
test_clean_categories  = encoder.transform(X_test['clean_categories'].to_numpy(
no_of_clean_categories = len(encoder.categories_[0])
print("No of categories present in clean categories are ",no_of_clean_categorie
print(train_clean_categories.shape)
print(test_clean_categories.shape)
```

```
No of categories present in clean categories are  51
(73196, 1)
(36052, 1)
```

clean_subcategories

```
from sklearn.preprocessing import OrdinalEncoder
encoder = OrdinalEncoder(handle_unknown = 'use_encoded_value',unknown_value=-1)
train_clean_subcategories = encoder.fit_transform(X_train['clean_subcategories
test_clean_subcategories  = encoder.transform(X_test['clean_subcategories'].to_
no_of_clean_subcategories = len(encoder.categories_[0])
print("No of categories present in clean sub categories are ",no_of_clean_subca
print(train_clean_subcategories.shape)
print(test_clean_subcategories.shape)
```

```
No of categories present in clean sub categories are  390
(73196, 1)
(36052, 1)
```

teacher_prefix

```
from sklearn.preprocessing import OrdinalEncoder
encoder = OrdinalEncoder(handle_unknown = 'use_encoded_value',unknown_value=-1)
train_teacher_prefix = encoder.fit_transform(X_train['teacher_prefix'].to_numpy
test_teacher_prefix  = encoder.transform(X_test['teacher_prefix'].to_numpy().re
no_of_teacher_prefix = len(encoder.categories_[0])
print("No of categories present in teacher prefix are ",no_of_teacher_prefix)
print(train_teacher_prefix.shape)
print(test_teacher_prefix.shape)
```

```
No of categories present in teacher prefix are  5
(73196, 1)
(36052, 1)
```

1.3 Numerical feature Vectorization

```
# you have to standardise the numerical columns
# stack both the numerical features
#after numerical feature vectorization you will have numerical_data_train and n
```

teacher_number_of_previously_posted_projects

Min-Max scaling of features

[ ] ↳ *1 cell hidden*

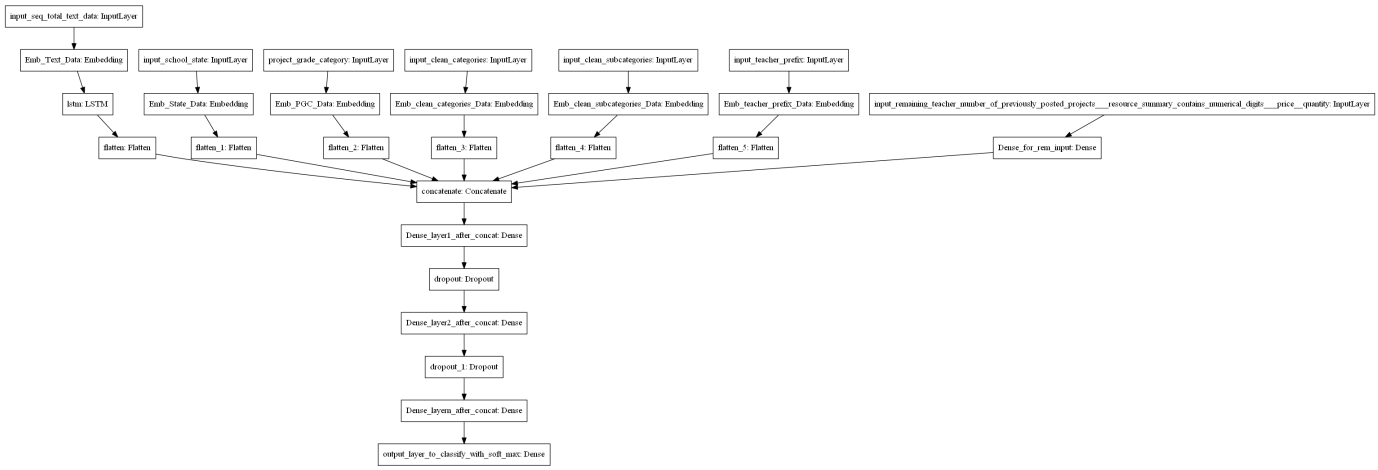
Standard scaling of features

[ ] ↳ *2 cells hidden*

price

[ ] ↳ *6 cells hidden*

1.4 Defining the model



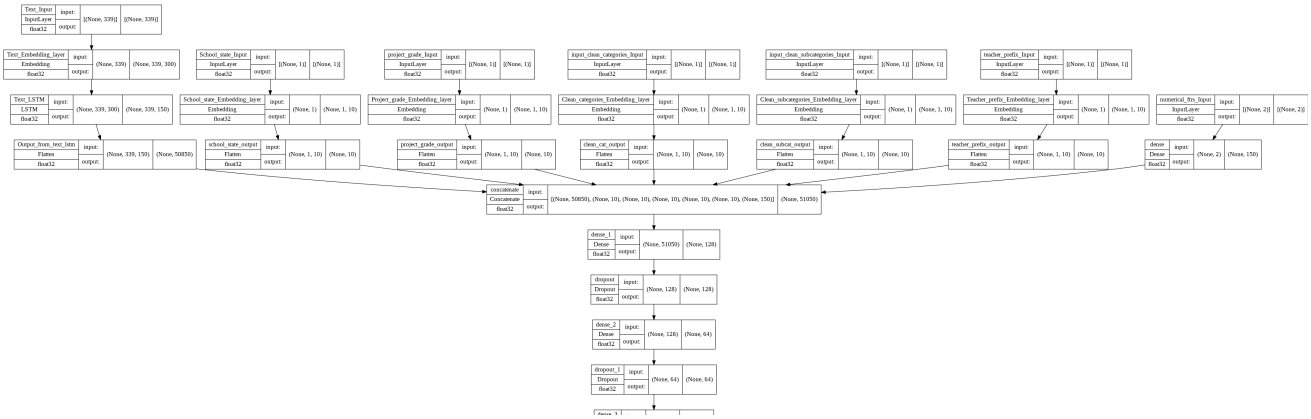

```
# as of now we have vectorized all our features now we will define our model.
# as it is clear from above image that the given model has multiple input layer
# Please go through - https://keras.io/guides/functional_api/
# it is a good programming practise to define your complete model i.e all input
# while defining your model make sure that you use variable names while defini
#for ex.- you should write the code as 'input_text = Input(shape=(pad_length,))
# the embedding layer for text data should be non trainable
# the embedding layer for categorical data should be trainable
# https://stats.stackexchange.com/questions/270546/how-does-keras-embedding-lay
# https://towardsdatascience.com/deep-embeddings-for-categorical-variables-cat2
#print model.summary() after you have defined the model
#plot the model using utils.plot_model module and make sure that it is similar
```

```
from tensorflow.keras import layers,Input,Model

# Hyper Parameters
Embedding_vector_size_text = 300
categorical_ftr_embedding_size = 20
max_seq_length = train_padded_docs.shape[1]
no_of_lstm_units_on_text = 150
no_of_Dense_units_on_numerical_ftr = 150
no_of_units_on_dense_layer1 = 128
no_of_units_on_dense_layer2 = 64
no_of_units_on_dense_layer3 = 32
no_of_units_on_output_dense_layer = 1

vocab_size_school_state = no_of_school_state
vocab_size_project_grade = no_of_project_grade
vocab_size_input_clean = no_of_clean_categories
vocab_size_input_clean_sub = no_of_clean_subcategories
vocab_size_teacher_prefix = no_of_teacher_prefix

def create_model(max_seq_length, vocab_size_text, embedding_matrix, categories_
                 categorical_ftr_embedding_size, intializer, numerical_ftr_dim,
  tf.keras.backend.clear_session()
```

```
# Text Feature block
input1 = Input(shape = (max_seq_length,),name = "Text_Input")
x = layers.Embedding(input_dim = vocab_size_text, output_dim=embedding_matri
                     input_length = max_seq_length, weights=[embedding_matri
                     name = "Text_Embedding_layer", trainable = False)(input1
x = layers.LSTM(no_of_lstm_units_on_text,return_sequences = True, name = "Tex
text_lstm_output = layers.Flatten(name = "Output_from_text_lstm")(x)

# Categorical feature school_state block
input2 = Input(shape = (categories_dim,),name = "School_state_Input")
x = layers.Embedding(input_dim = vocab_size_school_state,
                     output_dim= categorical_ftr_embedding_size,
                     embeddings_initializer = intializer,
                     input_length = categories_dim,
                     name = "School_state_Embedding_layer")(input2)
school_state_output = layers.Flatten(name = "school_state_output")(x)

# Categorical feature project_grade block
input3 = Input(shape = (categories_dim,),name = "project_grade_Input")
x = layers.Embedding(input_dim = vocab_size_project_grade,
                      output_dim = categorical_ftr_embedding_size,
                      embeddings_initializer = intializer,
                      input_length = categories_dim,
                      name = "Project_grade_Embedding_layer")(input3)
project_grade_output = layers.Flatten(name = "project_grade_output")(x)

# Categorical feature input_clean_categories block
input4 = Input(shape = (categories_dim,),name = "input_clean_categories_Input
x = layers.Embedding(input_dim = vocab_size_input_clean,
                      output_dim = categorical_ftr_embedding_size,
                      embeddings_initializer = intializer,
                      input_length = categories_dim,
                      name = "Clean_categories_Embedding_layer")(input4)
clean_cat_output = layers.Flatten(name = "clean_cat_output")(x)

# Categorical feature input_clean_subcategories block
input5 = Input(shape = (categories_dim,),name = "input_clean_subcategories_In
x = layers.Embedding(input_dim = vocab_size_input_clean_sub,
                      output_dim = categorical_ftr_embedding_size,
                      embeddings_initializer = intializer,
                      input_length = categories_dim,
                      name = "Clean_subcategories_Embedding_layer")(input5)
clean_subcat_output = layers.Flatten(name = "clean_subcat_output")(x)

# Categorical feature teacher_prefix block
input6 = Input(shape = (categories_dim,),name = "teacher_prefix_Input")
x = layers.Embedding(input_dim = vocab_size_teacher_prefix,
                      output_dim = categorical_ftr_embedding_size,
                      embeddings_initializer = intializer,
                      input_length = categories_dim,
                      name = "Teacher_prefix_Embedding_layer")(input6)
teacher_prefix_output = layers.Flatten(name = "teacher_prefix_output")(x)

# Numerical features block
input7 = Input(shape = (numerical_ftr_dim,),name = "numerical_ftrs_Input")
```

```
  numerical_ftr_output = layers.Dense(no_of_Dense_units_on_numerical_ftr,
                                      activation= activation_function,
                                      kernel_initializer = intializer)(input7)

  concat_output = layers.Concatenate(axis=1)([text_lstm_output, school_state_ou
                                              clean_cat_output, clean_subcat_ou
                                              numerical_ftr_output])

  dense_layer1_after_concat = layers.Dense(no_of_units_on_dense_layer1,
                                           activation = activation_function,
                                           kernel_initializer=intializer)(conca
  dropout_layer1 = layers.Dropout(dropout_rate)(dense_layer1_after_concat)

  dense_layer2_after_concat = layers.Dense(no_of_units_on_dense_layer2,
                                           activation = activation_function,
                                           kernel_initializer=intializer)(dropo
  dropout_layer2 = layers.Dropout(dropout_rate)(dense_layer2_after_concat)

  dense_layer3_after_concat = layers.Dense(no_of_units_on_dense_layer3,
                                           activation = activation_function,
                                           kernel_initializer=intializer)(dropo
  output = layers.Dense(no_of_units_on_output_dense_layer,
                        activation = 'sigmoid',
                        kernel_initializer=intializer)(dense_layer3_after_conca

  return Model(inputs = [input1,input2,input3,input4,input5,input6,input7],outp
```

```
from tensorflow.keras import utils,initializers

max_seq_length = train_padded_docs.shape[1]
vocab_size_text = vocab_size
categories_dim = 1
categorical_ftr_embedding_size = 10
initializer = initializers.HeUniform()
activation_function = 'relu'
dropout_rate = 0.3
numerical_ftr_dim = 2

model = create_model(max_seq_length, vocab_size_text, embedding_matrix, categor
                categorical_ftr_embedding_size, initializer,numerical_ftr_dim,

utils.plot_model(model, "Model_1.png", show_shapes=True,show_dtype = True)
```



1.5 Compiling and fititng your model

```
#define custom auc as metric , do not use tf.keras.metrics
# https://stackoverflow.com/a/46844409 - custom AUC reference 1
# https://www.kaggle.com/c/santander-customer-transaction-prediction/discussion
# compile and fit your model
```

```
from sklearn.metrics import roc_auc_score,roc_curve,auc
from keras.callbacks import Callback
class RocCallback(Callback):
    def __init__(self,training_data,validation_data):
        self.x = training_data[0]
        self.y = training_data[1]
        self.x_val = validation_data[0]
        self.y_val = validation_data[1]


    def on_train_begin(self, logs={}):
        self.model.auc_train = []
        self.model.auc_val = []
        return

    def on_train_end(self, logs={}):
        return

    def on_epoch_begin(self, epoch, logs={}):
        return

    def on_epoch_end(self, epoch, logs={}):
        y_pred_train = self.model.predict(self.x)
        auc_train = roc_auc_score(self.y, y_pred_train)
        y_pred_val = self.model.predict(self.x_val)
        auc_val = roc_auc_score(self.y_val, y_pred_val)
        self.model.auc_train.append(auc_train)
        self.model.auc_val.append(auc_val)
        print('\rauc_train: %s - auc_val: %s' % (str(round(auc_train,4)),str(ro
        return

    def on_batch_begin(self, batch, logs={}):
        return
```

```
    def on_batch_end(self, batch, logs={}):
        return
```

```
from tensorflow.keras import callbacks
import os

# path = "/Users/yamasanimanoj-kumarreddy/Documents/AAIC/Named_Assignments/LSTM
# path = "/content/drive/MyDrive/Named_Assignments/LSTM_DonorsChoose"

graph_saving_dir = os.path.join(path,'model_1', "Tensorboard_graphs")

roc_callback = RocCallback(training_data=([train_padded_docs,train_school_state
                                   train_project_grade,train_clean_categories,
                                   train_clean_subcategories,train_teacher_prefi
                                   numerical_data_train ],
                                   Y_train),
                   validation_data=([test_padded_docs,test_school_state,
                                   test_project_grade,test_clean_categories,
                                   test_clean_subcategories,test_teacher_prefix,
                                   numerical_data_test ],
                                   Y_test))


tensorboard_callback = callbacks.TensorBoard(log_dir=graph_saving_dir,histogram

callbacksList = [roc_callback, tensorboard_callback]
```

```
  from tensorflow.keras import initializers,optimizers, metrics,losses
  from sklearn.utils.class_weight import compute_class_weight

  # Types of intializers to try
  # initializers.HeUniform()
  # initializers.HeNormal()
  # initializers.GlorotNormal()
  # initializers.GlorotUniform()
  # initializers.LecunNormal()
  # initializers.LecunUniform()
  # initializers.RandomNormal(mean=0., stddev=1.)
  # initializers.RandomUniform(minval=0., maxval=1.)

  # Types of Optimizers to try
  # optimizers.Adadelta(learning_rate=0.001)
  # optimizers.Adagrad(learning_rate=0.001)
  # optimizers.Adamax(learning_rate=0.001)
  # optimizers.RMSprop(learning_rate=0.001)
  # tf.keras.optimizers.SGD(learning_rate=0.01,momentum=0.0)

  opt = optimizers.Adam()
  max_seq_length = train_padded_docs.shape[1]
  vocab_size_text = vocab_size
  categories_dim = 1
  categorical_ftr_embedding_size = 20
  initializer = initializers.HeNormal()
  activation_function = 'relu'
```

```
activation_function = 'relu'
dropout_rate = 0.7
numerical_ftr_dim = 2
class_weights = compute_class_weight(class_weight = "balanced",classes = np.un
class_weights = dict(zip(np.unique(Y_train), class_weights))
print(class_weights)

model = create_model(max_seq_length, vocab_size_text, embedding_matrix, catego
                 categorical_ftr_embedding_size, initializer,numerical_ftr_dim

batch_size = 1024
epochs = 10

model.compile(optimizer=opt,
              metrics = [metrics.BinaryAccuracy()],
              loss=losses.BinaryCrossentropy())

# fit the model
model.fit([train_padded_docs,train_school_state,
          train_project_grade,train_clean_categories,
          train_clean_subcategories,train_teacher_prefix,
          numerical_data_train ],
          Y_train, batch_size = batch_size,
          epochs=epochs, callbacks=callbacksList,
          class_weight = class_weights,
          validation_data = ([ test_padded_docs,test_school_state,
                               test_project_grade,test_clean_categories,
                               test_clean_subcategories,test_teacher_prefix,
                               numerical_data_test ],
                               Y_test))
# evaluate the model
loss, accuracy = model.evaluate([ test_padded_docs,test_school_state,
                               test_project_grade,test_clean_categories,
                               test_clean_subcategories,test_teacher_prefix,
                                 numerical_data_test ],
                               Y_test, batch_size = batch_size)
print('Test Accuracy: %f' % (accuracy*100))
```

```
{0: 3.3021745014887665, 1: 0.5892164281229372}
Epoch 1/10
 6/72 [=>............................] - ETA: 16s - loss: 0.9706 - binary_accu
auc_train: 0.5047 - auc_val: 0.5051
72/72 [==============================] - 59s 791ms/step - loss: 0.7168 - binar
Epoch 2/10
auc_train: 0.509 - auc_val: 0.5069
72/72 [==============================] - 56s 789ms/step - loss: 0.6936 - binar
Epoch 3/10
auc_train: 0.5068 - auc_val: 0.5072
72/72 [==============================] - 54s 758ms/step - loss: 0.6935 - binar
Epoch 4/10
auc_train: 0.5181 - auc_val: 0.5192
72/72 [==============================] - 55s 764ms/step - loss: 0.6931 - binar
Epoch 5/10
auc_train: 0.5629 - auc_val: 0.5615
72/72 [==============================] - 54s 758ms/step - loss: 0.6931 - binar
Epoch 6/10
auc_train: 0.7013 - auc_val: 0.6898
```

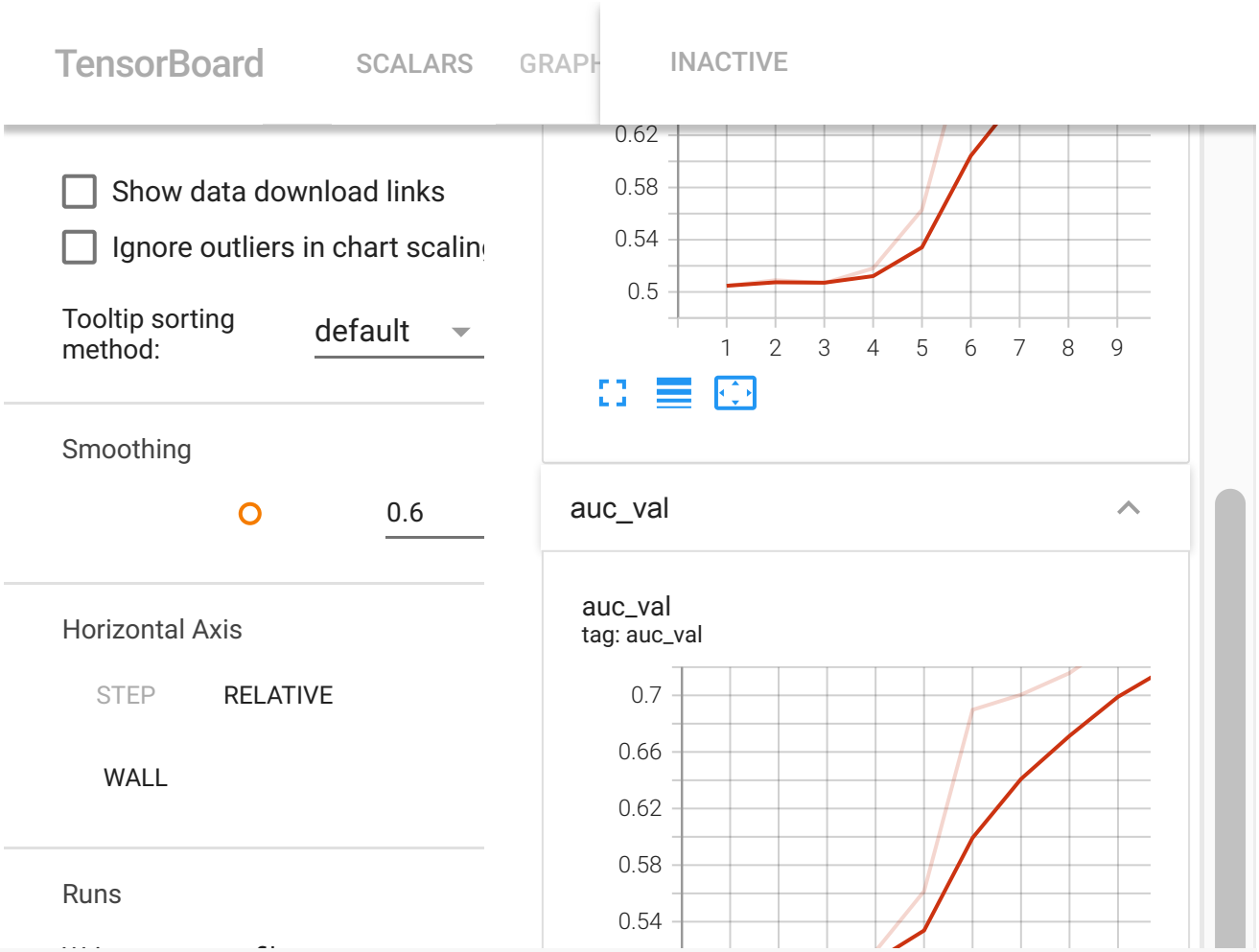
```
72/72 [==============================] - 49s 688ms/step - loss: 0.6895 - binar
Epoch 7/10
auc_train: 0.718 - auc_val: 0.7005
72/72 [==============================] - 56s 783ms/step - loss: 0.6850 - binar
Epoch 8/10
auc_train: 0.7322 - auc_val: 0.7158
72/72 [==============================] - 55s 772ms/step - loss: 0.6695 - binar
Epoch 9/10
auc_train: 0.7667 - auc_val: 0.7397
72/72 [==============================] - 51s 715ms/step - loss: 0.6516 - binar
Epoch 10/10
auc_train: 0.777 - auc_val: 0.7488
72/72 [==============================] - 56s 781ms/step - loss: 0.6367 - binar
36/36 [==============================] - 3s 83ms/step - loss: 0.6533 - binary_
Test Accuracy: 68.492734
```

```
#After fit method accessing the auc scores of train and validation
auc_train = model.auc_train
auc_val = model.auc_val

writer=tf.summary.create_file_writer(graph_saving_dir)
for idx in range(len(auc_train)):
    with writer.as_default(step=idx+1):
        tf.summary.scalar('auc_train', auc_train[idx])
        tf.summary.scalar('auc_val', auc_val[idx])
writer.flush ()
```

```
%load_ext tensorboard
```

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

```
%tensorboard --logdir /content/drive/MyDrive/Named_Assignments/LSTM_DonorsChoos
# %tensorboard --logdir /content/drive/MyDrive/DeepLearn/LSTM_DonorsChoose/mode
# %tensorboard --logdir /Users/yamasanimanoj-kumarreddy/Documents/AAIC/Named_As
```




```
!tensorboard dev upload --logdir /content/drive/MyDrive/Named Assignments/LSTM

# !tensorboard dev upload --logdir %tensorboard --logdir /content/drive/MyDriv
# !tensorboard dev upload --logdir /Users/yamasanimanoj-kumarreddy/Documents/A
```

```
Upload started and will continue reading any new data as it's added to the log

To stop uploading, press Ctrl-C.

New experiment created. View your TensorBoard at: https://tensorboard.dev/expe

[2022-07-26T21:30:19] Started scanning logdir.
[2022-07-26T21:30:23] Total uploaded: 80 scalars, 190 tensors (136.4 kB), 1 bi


Interrupted. View your TensorBoard at https://tensorboard.dev/experiment/heE6b
Traceback (most recent call last):
  File "/usr/local/bin/tensorboard", line 8, in <module>
    sys.exit(run_main())
  File "/usr/local/lib/python3.7/dist-packages/tensorboard/main.py", line 46,
    app.run(tensorboard.main, flags_parser=tensorboard.configure)
  File "/usr/local/lib/python3.7/dist-packages/absl/app.py", line 308, in run
    _run_main(main, args)
  File "/usr/local/lib/python3.7/dist-packages/absl/app.py", line 254, in _run
    sys.exit(main(argv))
  File "/usr/local/lib/python3.7/dist-packages/tensorboard/program.py", line 2
    return runner(self.flags) or 0
  File "/usr/local/lib/python3.7/dist-packages/tensorboard/uploader/uploader_s
    return _run(flags, self._experiment_url_callback)
  File "/usr/local/lib/python3.7/dist-packages/tensorboard/uploader/uploader_s
```

```
    intent.execute(server_info, channel)
KeyboardInterrupt
```

The above Tensorboard logs can be observed at this url
https://tensorboard.dev/experiment/heE6bKqjS4yFjs1OJVqL0Q/

Observations from the above Training of model

1. By using different intializers for kernel in Dense layers we could train different patterns of input data which might increase auc and accuracy.
2. At first we have tried without including `return_sequences = True` paramter in LSTM of text data. Due to this we get only 1 vector as output from LSTM cell and as a single vector cannot capture the entire esence of text sequence we got very less accuracy and auc values. But when we include the parameter `return_sequences = True` we get the output from every timestamp which increases the pattern information gained and thereby we got significant improvement in Test Accuracy and Auc.
3. From the training results we can observe that model is not overfitted and achieved **validation auc = 0.75**

Model-2

Use the same model as above but for 'input_seq_total_text_data' give only some words in the sentance not all the words. Filter the words as below.

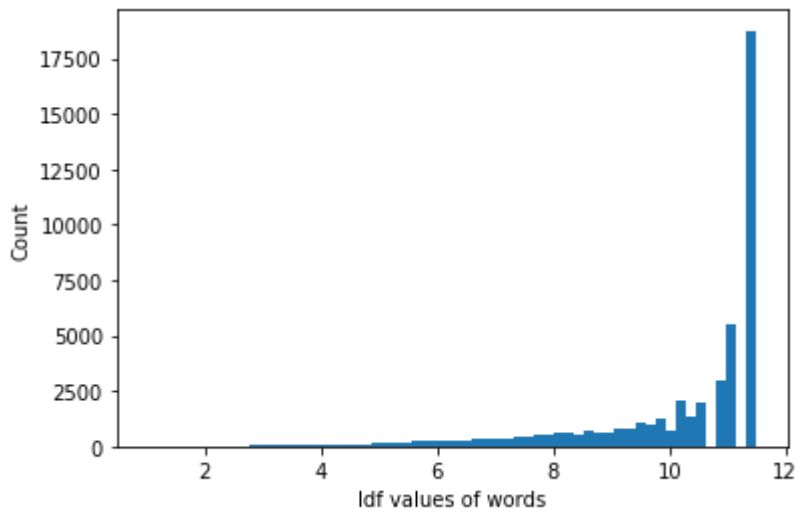
```
1. Fit TF-IDF vectorizer on the Train data

2. Get the idf value for each word we have in the train data. Please go through t


3. Do some analysis on the Idf values and based on those values choose the low an
frequent words and very very rare words don't give much information.
Hint - A preferable IDF range is 2-11 for model 2.

4.Remove the low idf value and high idf value words from the train and test data.
sentence of train and test data and include only those features(words) which are
5. Perform tokenization on the modified text data same as you have done for previ
6. Create embedding matrix for model 2 and then use the rest of the features simi
7. Define the model, compile and fit the model.
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
X = vectorizer.fit(X_train['essay'])
idf = vectorizer.idf_
word_idf = dict(zip(vectorizer.get_feature_names(), idf))
```

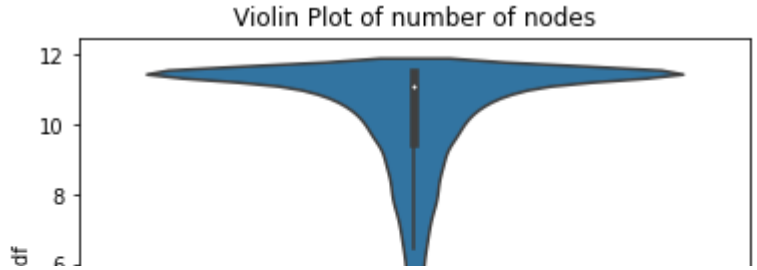
```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: Future
  warnings.warn(msg, category=FutureWarning)
```

plot histogram of idf values.

```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
import seaborn as sns

np.random.seed(42)

plt.hist(idf, density=False, bins=60)  # density=False would make counts
plt.ylabel('Count')
plt.xlabel('Idf values of words');

```




```
sns.violinplot(y='idf',data=pd.DataFrame({'idf':idf}))
plt.title("Violin Plot of number of nodes")
plt.show()
```



From the above cell output we can observe that most of the idf values are present in between 2 and 12.

```
remove_words = []
low_idf_threshold = 4
high_idf_threshold = 11.5
print(len(word_idf))
for k,v in word_idf.items():
    if (v>high_idf_threshold or v<low_idf_threshold):
        remove_words.append(k)
remove_words = set(remove_words)
print("No of words to be removed from training and test data are :",len(remove
# print(remove_words)
```

```
48195
No of words to be removed from training and test data are : 19213
```

```
if 'classroom' in remove_words:
  print("word 'classroom' is present in remove_words set")
```

```
word 'classroom' is present in remove_words set
```

+ Code + Text

```
check_word = "classroom"
sample = X_train['essay'].iloc[0]
print("Index of word '{0}' is {1}".format(check_word,sample.find(check_word)))
```

```
Index of word 'classroom' is 153
```

```
func = lambda x: ' '.join([item for item in x.split() if item.lower() not in re
X_train['modified_essay'] = X_train['essay'].apply(func)
X_test['modified_essay'] = X_test['essay'].apply(func)
```

```
check_word = "classroom"
sample = X_train['modified_essay'].iloc[0]
if (sample.find(check_word) == -1):
  print("word '{0}' is not found after modifying".format(check_word))
```

```
word 'classroom' is not found after modifying
```

```
print(X_train['essay'].iloc[1])
print(X_train['modified_essay'].iloc[1])
# checking whether words like 'we','service' exists in remove_words
print('we' in remove_words)
```

```
we service highly impoverished area 83 families participating free lunch too m
service highly impoverished 83 participating too deal homelessness hunger torn
True
```

Text Vectorization

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train['modified_essay'])
vocab_size = len(tokenizer.word_index) + 1
# integer encode the documents
train_encoded_docs = tokenizer.texts_to_sequences(X_train['modified_essay'])
test_encoded_docs = tokenizer.texts_to_sequences(X_test['modified_essay'])
```

```
max_length = max([len(seq) for seq in train_encoded_docs])
print(max_length)
train_padded_docs = pad_sequences(train_encoded_docs, maxlen=max_length, paddi
test_padded_docs = pad_sequences(test_encoded_docs, maxlen=max_length, padding=
print(train_padded_docs.shape)
print(test_padded_docs.shape)
print(train_padded_docs[0][-10:])
print(test_padded_docs[0][-10:])
```

```
192
(73196, 192)
(36052, 192)
[0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0]
```

```
#after getting the padded_docs you have to use predefined glove vectors to get
# we will be storing this data in form of an embedding matrix and will use it w
# Please go through following blog's 'Example of Using Pre-Trained GloVe Embedd
# https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-ke
import pickle
import numpy as np

# path = "/Users/yamasanimanoj-kumarreddy/Documents/AAIC/Named_Assignments/LSTM
# path = "/content/drive/MyDrive/Named_Assignments/LSTM_DonorsChoose/"

with open(path+'/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

```
embedding_matrix_model2 = np.zeros((vocab_size, 300))
for word, i in tokenizer.word_index.items():
    embedding_vector = model.get(word)
    if embedding_vector is not None:
```

```
        embedding_matrix_model2[i] = embedding_vector
print(embedding_matrix_model2[1][:5])
print(embedding_matrix_model2.shape)
```

```
[-0.043504 -0.18484  -0.14613  -0.21751   0.2025  ]
(29019, 300)
```

Defining model and compiling the model

```
from tensorflow.keras import callbacks
import os

# path = "/Users/yamasanimanoj-kumarreddy/Documents/AAIC/Named_Assignments/LSTM
# path = "/content/drive/MyDrive/Named_Assignments/LSTM_DonorsChoose"

graph_saving_dir = os.path.join(path,'model_2', "Tensorboard_graphs")

roc_callback = RocCallback(training_data=([train_padded_docs,train_school_state
                                   train_project_grade,train_clean_categories,
                                   train_clean_subcategories,train_teacher_prefi
                                   numerical_data_train ],
                                   Y_train),
                  validation_data=([test_padded_docs,test_school_state,
                                   test_project_grade,test_clean_categories,
                                   test_clean_subcategories,test_teacher_prefix,
                                   numerical_data_test ],
                                   Y_test))


tensorboard_callback = callbacks.TensorBoard(log_dir=graph_saving_dir,histogram

callbacksList = [roc_callback, tensorboard_callback]
```

```
from tensorflow.keras import initializers,optimizers, metrics,losses
from sklearn.utils.class_weight import compute_class_weight

# Types of intializers to try
# initializers.HeUniform()
# initializers.HeNormal()
# initializers.GlorotNormal()
# initializers.GlorotUniform()
# initializers.LecunNormal()
# initializers.LecunUniform()
# initializers.RandomNormal(mean=0., stddev=1.)
# initializers.RandomUniform(minval=0., maxval=1.)

# Types of Optimizers to try
# optimizers.Adadelta(learning_rate=0.001)
# optimizers.Adagrad(learning_rate=0.001)
# optimizers.Adamax(learning_rate=0.001)
# optimizers.RMSprop(learning_rate=0.001)
# tf.keras.optimizers.SGD(learning_rate=0.01,momentum=0.0)
```

```
opt = optimizers.Adam()
max_seq_length = train_padded_docs.shape[1]
vocab_size_text = vocab_size
categories_dim = 1
categorical_ftr_embedding_size = 20
initializer = initializers.HeNormal()
activation_function = 'relu'
dropout_rate = 0.7
numerical_ftr_dim = 2
class_weights = compute_class_weight(class_weight = "balanced",classes = np.uni
class_weights = dict(zip(np.unique(Y_train), class_weights))
print(class_weights)

model = create_model(max_seq_length, vocab_size_text, embedding_matrix_model2,
                 categorical_ftr_embedding_size, initializer,numerical_ftr_dim,

batch_size = 1024
epochs = 10

model.compile(optimizer=opt,
              metrics = [metrics.BinaryAccuracy()],
              loss=losses.BinaryCrossentropy())

# fit the model
model.fit([train_padded_docs,train_school_state,
          train_project_grade,train_clean_categories,
          train_clean_subcategories,train_teacher_prefix,
          numerical_data_train ],
          Y_train, batch_size = batch_size,
          epochs=epochs, callbacks=callbacksList,
          class_weight = class_weights,
          validation_data = ([ test_padded_docs,test_school_state,
                               test_project_grade,test_clean_categories,
                               test_clean_subcategories,test_teacher_prefix,
                               numerical_data_test ],
                               Y_test))
# evaluate the model
loss, accuracy = model.evaluate([ test_padded_docs,test_school_state,
                               test_project_grade,test_clean_categories,
                               test_clean_subcategories,test_teacher_prefix,
                                 numerical_data_test ],
                               Y_test, batch_size = batch_size)
print('Test Accuracy: %f' % (accuracy*100))
```

```
{0: 3.3021745014887665, 1: 0.5892164281229372}
Epoch 1/10
 6/72 [=>............................] - ETA: 9s - loss: 0.8705 - binary_accur
auc_train: 0.6831 - auc_val: 0.6839
72/72 [==============================] - 48s 646ms/step - loss: 0.7052 - binar
Epoch 2/10
auc_train: 0.7105 - auc_val: 0.7017
72/72 [==============================] - 45s 626ms/step - loss: 0.6725 - binar
Epoch 3/10
auc_train: 0.7329 - auc_val: 0.7105
72/72 [==============================] - 33s 457ms/step - loss: 0.6542 - binar
```

```
Epoch 4/10
auc_train: 0.7489 - auc_val: 0.7207
72/72 [==============================] - 32s 444ms/step - loss: 0.6397 - binar
Epoch 5/10
auc_train: 0.762 - auc_val: 0.7219
72/72 [==============================] - 32s 451ms/step - loss: 0.6267 - binar
Epoch 6/10
auc_train: 0.781 - auc_val: 0.7173
72/72 [==============================] - 46s 640ms/step - loss: 0.6141 - binar
Epoch 7/10
auc_train: 0.7985 - auc_val: 0.7213
72/72 [==============================] - 33s 455ms/step - loss: 0.6038 - binar
Epoch 8/10
auc_train: 0.8183 - auc_val: 0.7175
72/72 [==============================] - 46s 642ms/step - loss: 0.5886 - binar
Epoch 9/10
auc_train: 0.8383 - auc_val: 0.706
72/72 [==============================] - 46s 639ms/step - loss: 0.5734 - binar
Epoch 10/10
auc_train: 0.8545 - auc_val: 0.7088
72/72 [==============================] - 37s 525ms/step - loss: 0.5573 - binar
36/36 [==============================] - 2s 47ms/step - loss: 0.5535 - binary_
Test Accuracy: 71.729726
```

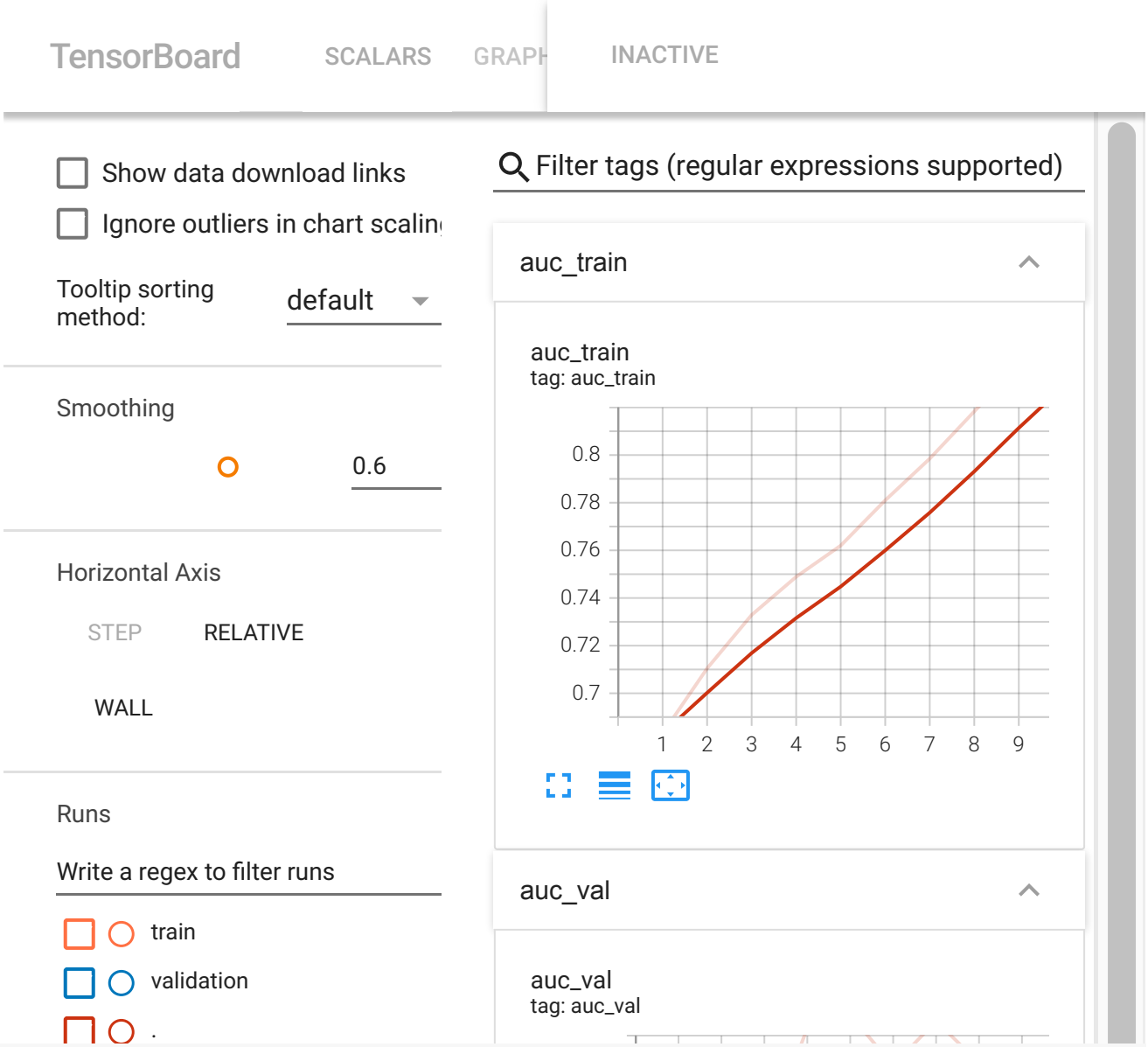
```
#After fit method accessing the auc scores of train and validation
auc_train = model.auc_train
auc_val = model.auc_val

writer=tf.summary.create_file_writer(graph_saving_dir)
for idx in range(len(auc_train)):
    with writer.as_default(step=idx+1):
        tf.summary.scalar('auc_train', auc_train[idx])
        tf.summary.scalar('auc_val', auc_val[idx])
writer.flush()
```

```
%load_ext tensorboard
```

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

```
%tensorboard --logdir /content/drive/MyDrive/Named_Assignments/LSTM_DonorsChoos
# %tensorboard --logdir /Users/yamasanimanoj-kumarreddy/Documents/AAIC/Named_As
```

```
Reusing TensorBoard on port 6006 (pid 810), started 0:17:29 ago. (Use '!kill
810' to kill it.)
```




```
1 !tensorboard dev upload --logdir /content/drive/MyDrive/Named_Assignments/LSTM
```

```
Upload started and will continue reading any new data as it's added to the log

To stop uploading, press Ctrl-C.

New experiment created. View your TensorBoard at: https://tensorboard.dev/expe

[2022-07-26T21:29:01] Started scanning logdir.
[2022-07-26T21:29:04] Total uploaded: 80 scalars, 190 tensors (136.4 kB), 1 bi


Interrupted. View your TensorBoard at https://tensorboard.dev/experiment/4A5yG
Traceback (most recent call last):
  File "/usr/local/bin/tensorboard", line 8, in <module>
    sys.exit(run_main())
  File "/usr/local/lib/python3.7/dist-packages/tensorboard/main.py", line 46,
    app.run(tensorboard.main, flags_parser=tensorboard.configure)
  File "/usr/local/lib/python3.7/dist-packages/absl/app.py", line 308, in run
    _run_main(main, args)
  File "/usr/local/lib/python3.7/dist-packages/absl/app.py", line 254, in _run
    sys.exit(main(argv))
```

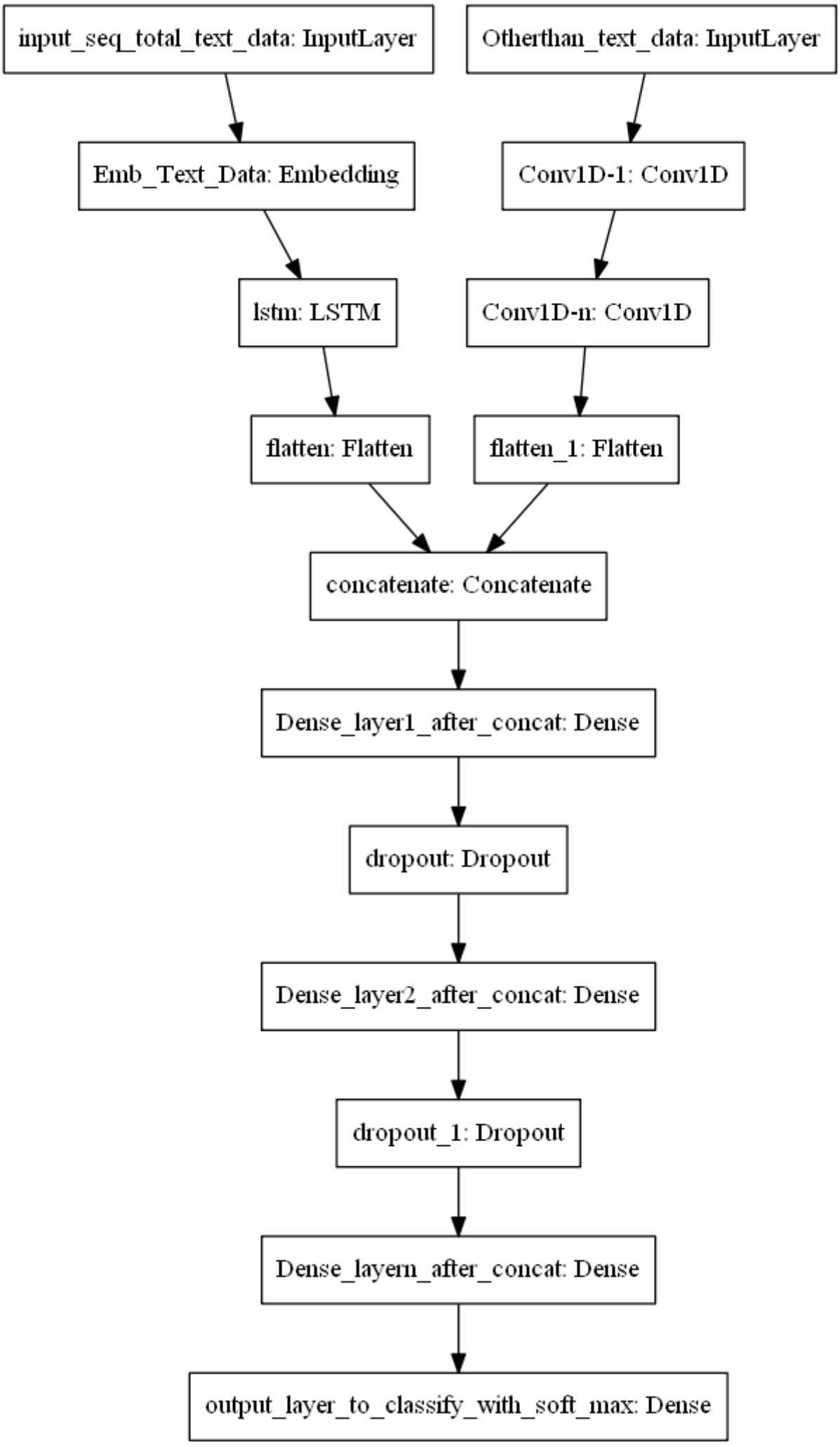
```
  File "/usr/local/lib/python3.7/dist-packages/tensorboard/program.py", line 2
    return runner(self.flags) or 0
  File "/usr/local/lib/python3.7/dist-packages/tensorboard/uploader/uploader_s
    return _run(flags, self._experiment_url_callback)
  File "/usr/local/lib/python3.7/dist-packages/tensorboard/uploader/uploader_s
    intent.execute(server_info, channel)
KeyboardInterrupt
```

The above Tensorboard logs can be observed at this url
https://tensorboard.dev/experiment/4A5yGvWDTLOqikhl3XZoHA/

Observations from the above Training of model

1. In this model due to the removal of stop words, unnecesseray information is removed from the text data and thereby our model gets trained faster and we achieve result at a less epoch number.
2. By using different intializers for kernel in Dense layers we could train different patterns of input data which might increase auc and accuracy.
3. At first we have tried without including `return_sequences = True` paramter in LSTM of text data. Due to this we get only 1 vector as output from LSTM cell and as a single vector cannot capture the entire esence of text sequence we got very less accuracy and auc values. But when we include the parameter `return_sequences = True` we get the output from every timestamp which increases the pattern information gained and thereby we got significant improvement in Test Accuracy and Auc.
4. From the training results we can observe that model gets overfitted in first five epochs only and thereby we achieved **validation auc = 0.70**

Model-3


input_seq_total_text_data: InputLayer
Otherthan_text_data: InputLayer
Emb_Text_Data: Embedding
Conv1D-1: Conv1D
lstm: LSTM
Conv1D-n: Conv1D
flatten: Flatten
flatten_1: Flatten
concatenate: Concatenate
Dense_layer1_after_concat: Dense
dropout: Dropout
Dense_layer2_after_concat: Dense
dropout_1: Dropout
Dense_layern_after_concat: Dense
output_layer_to_classify_with_soft_max: Dense

ref:

https://i.imgur.com/fkQ8nGo.png

```
#in this model you can use the text vectorized data from model1
#for other than text data consider the following steps
# you have to perform one hot encoding of categorical features. You can use one
# Stack up standardised numerical features and all the one hot encoded categori
#the input to conv1d layer is 3d, you can convert your 2d data to 3d using np.r
# Note - deep learning models won't work with sparse features, you have to conv
```

1.1 Text Vectorization

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train['essay'])
vocab_size = len(tokenizer.word_index) + 1
print(vocab_size)
# integer encode the documents
train_encoded_docs = tokenizer.texts_to_sequences(X_train['essay'])
test_encoded_docs = tokenizer.texts_to_sequences(X_test['essay'])
```

```
48232
```

```
max_length = max([len(seq) for seq in train_encoded_docs])
print(max_length)
train_padded_docs = pad_sequences(train_encoded_docs, maxlen=max_length, paddi
test_padded_docs = pad_sequences(test_encoded_docs, maxlen=max_length, padding=
print(train_padded_docs.shape)
print(test_padded_docs.shape)
print(train_padded_docs[0][-10:])
print(test_padded_docs[0][-10:])
```

```
339
(73196, 339)
(36052, 339)
[0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0]
```

```
#after getting the padded_docs you have to use predefined glove vectors to get
# we will be storing this data in form of an embedding matrix and will use it w
# Please go through following blog's 'Example of Using Pre-Trained GloVe Embedd
# https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-ke
import pickle
```

```
import numpy as np

with open(path+'/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

```
embedding_matrix_model3 = np.zeros((vocab_size, 300))
for word, i in tokenizer.word_index.items():
    embedding_vector = model.get(word)
    if embedding_vector is not None:
        embedding_matrix_model3[i] = embedding_vector
print(embedding_matrix_model3[1][:5])
print(embedding_matrix_model3.shape)
print("The number of tokens present in this vocabulary are", embedding_matrix_m
```

```
[ 0.15243  -0.16945  -0.022748 -0.25051  -0.15213 ]
(48232, 300)
The number of tokens present in this vocabulary are 48232
```

1.2 Categorical feature Vectorization

[ ] ↳ *11 cells hidden*

1.3 Numerical feature Vectorization

[ ] ↳ *13 cells hidden*

Non-Text Data Concatenation

```
from sklearn.decomposition import PCA

nontext_data_train = np.concatenate((train_school_state, train_project_grade,
                                     train_clean_categories, train_clean_subc
                                     train_tchr_prj,train_price),axis=1)
nontext_data_test = np.concatenate((test_school_state, test_project_grade,
                                    test_clean_categories, test_clean_subcate
                                    test_tchr_prj,test_price),axis=1)


```

```
# Due to onehot encoding, the features are sparse. To aovid this we use pca to
# We are taking 30 components as they combinedly retain more than 80% of the ir
reduced_dim = 30
pca = PCA(n_components = reduced_dim)
pca.fit(nontext_data_train)

```

```
nontext_data_train = pca.transform(nontext_data_train)
```

```
print(sum(pca.explained_variance_ratio_))
```

```
0.8052539834276029
```

```
nontext_data_train = np.expand_dims(nontext_data_train, axis=2)
nontext_data_test = np.expand_dims(nontext_data_test, axis=2)

print(nontext_data_train.shape)
print(nontext_data_test.shape)
```

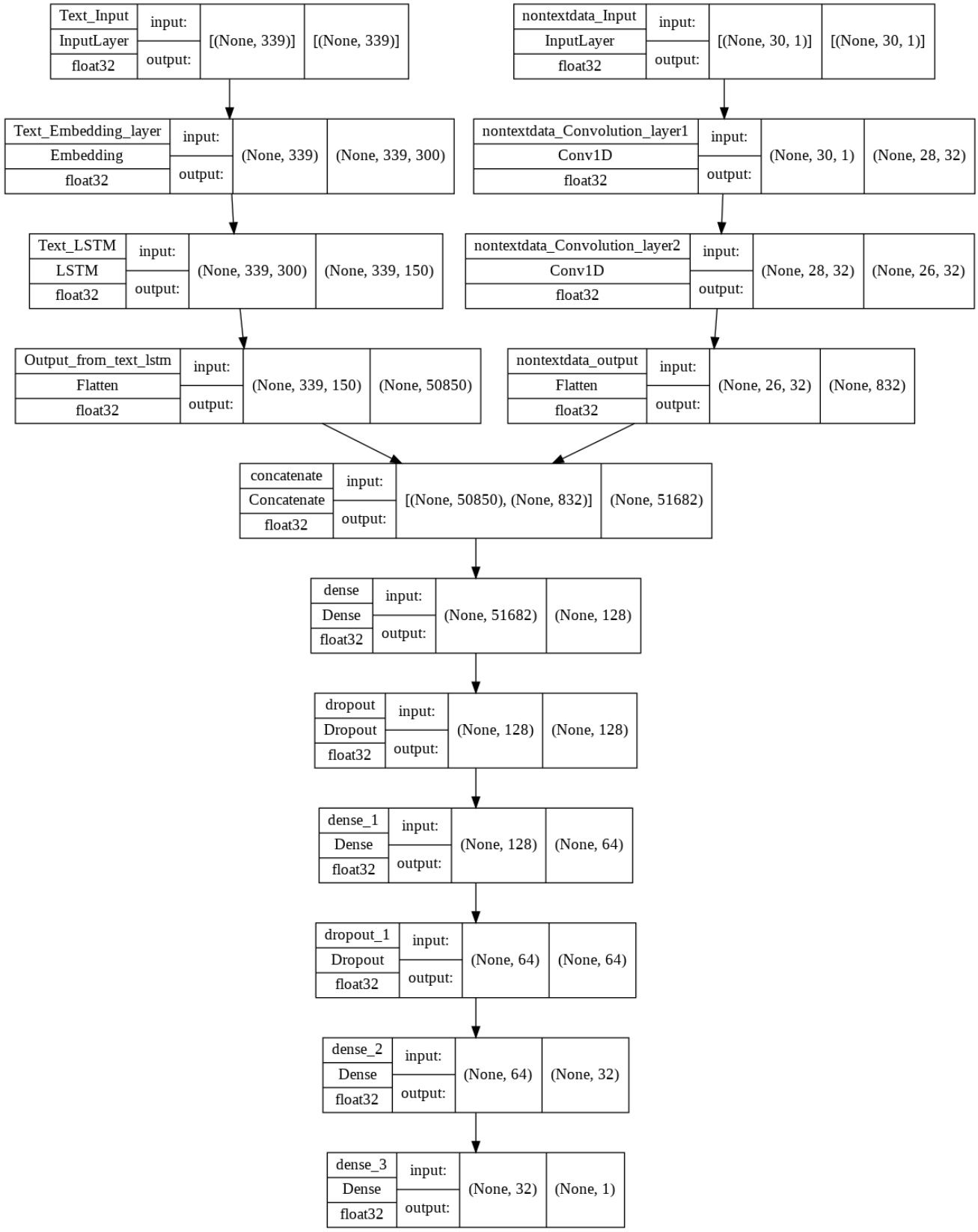
```
(73196, 30, 1)
(36052, 30, 1)
```

1.4 Defining the model

```
from tensorflow.keras import layers,Input,Model

# Hyper Parameters
Embedding_vector_size_text = 300
categorical_ftr_embedding_size = 20
max_seq_length = train_padded_docs.shape[1]
no_of_lstm_units_on_text = 150
no_of_Dense_units_on_numerical_ftr = 150
no_of_units_on_dense_layer1 = 128
no_of_units_on_dense_layer2 = 64
no_of_units_on_dense_layer3 = 32
no_of_units_on_output_dense_layer = 1

def create_model(max_seq_length, vocab_size_text, embedding_matrix, nontextdata
                 no_of_filters, kernel_size,
                 intializer, activation_function, dropout_rate):
  tf.keras.backend.clear_session()
  # Text Feature block
  input1 = Input(shape = (max_seq_length,),name = "Text_Input")
  x = layers.Embedding(input_dim = vocab_size_text, output_dim=embedding_matri
                       input_length = max_seq_length, weights=[embedding_matri
                       name = "Text_Embedding_layer", trainable = False)(input1
  x = layers.LSTM(no_of_lstm_units_on_text, return_sequences=True, name = "Text
  text_lstm_output = layers.Flatten(name = "Output_from_text_lstm")(x)

  # Nontext data block
  input2 = Input(shape = (nontextdata_dim,1,),name = "nontextdata_Input")
  x = layers.Conv1D(no_of_filters, kernel_size, activation=activation_function,
                    name = "nontextdata_Convolution_layer1")(input2)
  x = layers.Conv1D(no_of_filters, kernel_size, activation=activation_function,
                    name = "nontextdata_Convolution_layer2")(x)
  nontextdata_output = layers.Flatten(name = "nontextdata_output")(x)

  # Concatenate two flatten inputs from text and nontext data
  concat_output = layers.Concatenate(axis=1)([text_lstm_output, nontextdata_out
```

```
36
37   dense_layer1_after_concat = layers.Dense(no_of_units_on_dense_layer1,
38                                            activation = activation_function,
39                                            kernel_initializer=intializer)(conca
40   dropout_layer1 = layers.Dropout(dropout_rate)(dense_layer1_after_concat)
41
42   dense_layer2_after_concat = layers.Dense(no_of_units_on_dense_layer2,
43                                            activation = activation_function,
44                                            kernel_initializer=intializer)(dropo
45   dropout_layer2 = layers.Dropout(dropout_rate)(dense_layer2_after_concat)
46
47   dense_layer3_after_concat = layers.Dense(no_of_units_on_dense_layer3,
48                                            activation = activation_function,
49                                            kernel_initializer=intializer)(dropo
50   output = layers.Dense(no_of_units_on_output_dense_layer,
51                         activation = 'sigmoid',
52                         kernel_initializer=intializer)(dense_layer3_after_conca
53
54   return Model(inputs = [input1,input2],outputs = output, name = "Model_3")
```

```
 1 from tensorflow.keras import utils,initializers
 2
 3 max_seq_length = train_padded_docs.shape[1]
 4 vocab_size_text = vocab_size
 5 nontextdata_dim = nontext_data_train.shape[1]
 6 initializer = initializers.HeUniform()
 7 activation_function = 'relu'
 8 dropout_rate = 0.3
 9 no_of_filters = 32
10 kernel_size = 3
11
12 model = create_model(max_seq_length, vocab_size_text, embedding_matrix_model3,
13                   no_of_filters, kernel_size,
14                   initializer, activation_function, dropout_rate)
15
16 utils.plot_model(model, "Model_3.png", show_shapes=True,show_dtype = True)
```


Text_Input
InputLayer
float32
input:
output:
[(None, 339)]
[(None, 339)]
nontextdata_Input
InputLayer
float32
input:
output:
[(None, 30, 1)]
[(None, 30, 1)]
Text_Embedding_layer
Embedding
float32
input:
output:
(None, 339)
(None, 339, 300)
nontextdata_Convolution_layer1
Conv1D
float32
input:
output:
(None, 30, 1)
(None, 28, 32)
Text_LSTM
LSTM
float32
input:
output:
(None, 339, 300)
(None, 339, 150)
nontextdata_Convolution_layer2
Conv1D
float32
input:
output:
(None, 28, 32)
(None, 26, 32)
Output_from_text_lstm
Flatten
float32
input:
output:
(None, 339, 150)
(None, 50850)
nontextdata_output
Flatten
float32
input:
output:
(None, 26, 32)
(None, 832)
concatenate
Concatenate
float32
input:
output:
[(None, 50850), (None, 832)]
(None, 51682)
dense
Dense
float32
input:
output:
(None, 51682)
(None, 128)
dropout
Dropout
float32
input:
output:
(None, 128)
(None, 128)
dense_1
Dense
float32
input:
output:
(None, 128)
(None, 64)
dropout_1
Dropout
float32
input:
output:
(None, 64)
(None, 64)
dense_2
Dense
float32
input:
output:
(None, 64)
(None, 32)
dense_3
Dense
float32
input:
output:
(None, 32)
(None, 1)

1.5 Compiling and fititng your model

```
from sklearn.metrics import roc_auc_score,roc_curve,auc
from keras.callbacks import Callback
class RocCallback(Callback):
    def __init__(self,training_data,validation_data):
        self.x = training_data[0]
        self.y = training_data[1]
        self.x_val = validation_data[0]
        self.y_val = validation_data[1]


    def on_train_begin(self, logs={}):
        self.model.auc_train = []
        self.model.auc_val = []
        return

    def on_train_end(self, logs={}):
        return

    def on_epoch_begin(self, epoch, logs={}):
        return

    def on_epoch_end(self, epoch, logs={}):
        y_pred_train = self.model.predict(self.x)
        auc_train = roc_auc_score(self.y, y_pred_train)
        y_pred_val = self.model.predict(self.x_val)
        auc_val = roc_auc_score(self.y_val, y_pred_val)
        self.model.auc_train.append(auc_train)
        self.model.auc_val.append(auc_val)
        print('\rauc_train: %s - auc_val: %s' % (str(round(auc_train,4)),str(rc
        return

    def on_batch_begin(self, batch, logs={}):
        return

    def on_batch_end(self, batch, logs={}):
        return
```

```
from tensorflow.keras import callbacks
import os

# path = "/Users/yamasanimanoj-kumarreddy/Documents/AAIC/Named_Assignments/LSTM
# path = "/content/drive/MyDrive/Named_Assignments/LSTM_DonorsChoose"

graph_saving_dir = os.path.join(path,'model_3', "Tensorboard_graphs")

roc_callback = RocCallback(training_data=([train_padded_docs,nontext_data_trair
```

```
                                        Y_train),
                          validation_data=([test_padded_docs,nontext_data_test
                                            Y_test))

tensorboard_callback = callbacks.TensorBoard(log_dir=graph_saving_dir,histogram

callbacksList = [roc_callback, tensorboard_callback]
```

```
from tensorflow.keras import initializers,optimizers, metrics,losses
from sklearn.utils.class_weight import compute_class_weight

# Types of intializers to try
# initializers.HeUniform()
# initializers.HeNormal()
# initializers.GlorotNormal()
# initializers.GlorotUniform()
# initializers.LecunNormal()
# initializers.LecunUniform()
# initializers.RandomNormal(mean=0., stddev=1.)
# initializers.RandomUniform(minval=0., maxval=1.)

# Types of Optimizers to try
# optimizers.Adadelta(learning_rate=0.001)
# optimizers.Adagrad(learning_rate=0.001)
# optimizers.Adamax(learning_rate=0.001)
# optimizers.RMSprop(learning_rate=0.001)
# tf.keras.optimizers.SGD(learning_rate=0.01,momentum=0.0)

opt = optimizers.Adam()
max_seq_length = train_padded_docs.shape[1]
vocab_size_text = vocab_size
nontextdata_dim = nontext_data_train.shape[1]
initializer = initializers.HeUniform()
activation_function = 'relu'
dropout_rate = 0.7
no_of_filters = 64
kernel_size = 10
class_weights = compute_class_weight(class_weight = "balanced",classes = np.uni
class_weights = dict(zip(np.unique(Y_train), class_weights))
print(class_weights)

model = create_model(max_seq_length, vocab_size_text, embedding_matrix_model3,
                     no_of_filters, kernel_size,
                     initializer, activation_function, dropout_rate)

batch_size = 1024
epochs = 10

model.compile(optimizer=opt,
              metrics = [metrics.BinaryAccuracy()],
              loss=losses.BinaryCrossentropy())

# fit the model
model.fit([train_padded_docs,nontext_data_train],
```

```
          Y_train, batch_size = batch_size,
          epochs=epochs, callbacks=callbacksList,
          class_weight = class_weights,
          validation_data = ([test_padded_docs,nontext_data_test],
                                Y_test))
# evaluate the model
loss, accuracy = model.evaluate([test_padded_docs,nontext_data_test],
                                Y_test, batch_size = batch_size)
print('Test Accuracy: %f' % (accuracy*100))
```

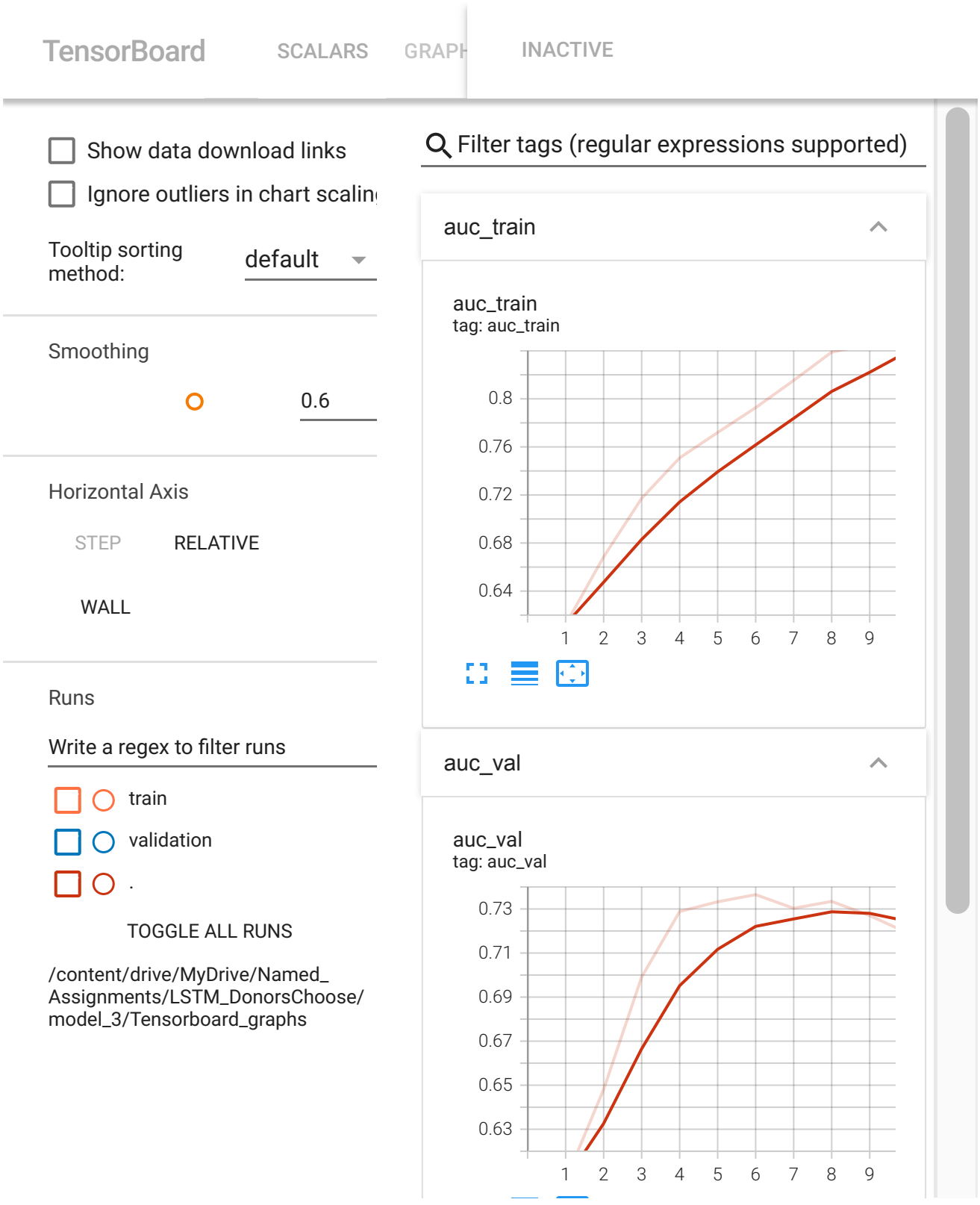
```
{0: 3.3021745014887665, 1: 0.5892164281229372}
Epoch 1/10
 6/72 [=>............................] - ETA: 16s - loss: 1.0236 - binary_accu
auc_train: 0.6123 - auc_val: 0.6068
72/72 [==============================] - 67s 826ms/step - loss: 0.7196 - binar
Epoch 2/10
auc_train: 0.6685 - auc_val: 0.6481
72/72 [==============================] - 52s 733ms/step - loss: 0.6808 - binar
Epoch 3/10
auc_train: 0.7172 - auc_val: 0.6991
72/72 [==============================] - 55s 772ms/step - loss: 0.6578 - binar
Epoch 4/10
auc_train: 0.7509 - auc_val: 0.7289
72/72 [==============================] - 55s 772ms/step - loss: 0.6335 - binar
Epoch 5/10
auc_train: 0.772 - auc_val: 0.7333
72/72 [==============================] - 57s 803ms/step - loss: 0.6181 - binar
Epoch 6/10
auc_train: 0.7927 - auc_val: 0.7365
72/72 [==============================] - 53s 740ms/step - loss: 0.6020 - binar
Epoch 7/10
auc_train: 0.8153 - auc_val: 0.7303
72/72 [==============================] - 57s 803ms/step - loss: 0.5803 - binar
Epoch 8/10
auc_train: 0.8389 - auc_val: 0.7334
72/72 [==============================] - 55s 771ms/step - loss: 0.5608 - binar
Epoch 9/10
auc_train: 0.8456 - auc_val: 0.727
72/72 [==============================] - 55s 768ms/step - loss: 0.5488 - binar
Epoch 10/10
auc_train: 0.8648 - auc_val: 0.7191
72/72 [==============================] - 57s 799ms/step - loss: 0.5326 - binar
36/36 [==============================] - 3s 82ms/step - loss: 0.4896 - binary_
Test Accuracy: 83.731830
```

```
#After fit method accessing the auc scores of train and validation
auc_train = model.auc_train
auc_val = model.auc_val

writer=tf.summary.create_file_writer(graph_saving_dir)
for idx in range(len(auc_train)):
    with writer.as_default(step=idx+1):
        tf.summary.scalar('auc_train', auc_train[idx])
        tf.summary.scalar('auc_val', auc_val[idx])
writer.flush ()
```

```
%load_ext tensorboard
```

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

```
%tensorboard --logdir /content/drive/MyDrive/Named_Assignments/LSTM_DonorsChoos
# %tensorboard --logdir /content/drive/MyDrive/DeepLearn/LSTM_DonorsChoose/mode
# %tensorboard --logdir /Users/yamasanimanoj-kumarreddy/Documents/AAIC/Named_As
```




```
!tensorboard dev upload --logdir /content/drive/MyDrive/Named_Assignments/LSTM_
```

```
2
3 # !tensorboard dev upload --logdir %tensorboard --logdir /content/drive/MyDrive
4 # !tensorboard dev upload --logdir /Users/yamasanimanoj-kumarreddy/Documents/A
```

The above Tensorboard logs can be observed at this url
https://tensorboard.dev/experiment/g1TO8oQxRESPPyEW0qvBnw/

Observations from the above Training of model

1. If we increase the no of components in PCA there is more information included in the components and thus we get more accuracy and auc.
2. If we increase the no of filters and kernel size the information gained from the words increases upto a certain point and later decreases due to addition of unnecessary data.
3. By using different intializers for kernel in Convolution layer we could train different patterns of input data which might increase auc and accuracy.
4. At first i have tried without including `return_sequences = True` paramter in LSTM of text data. Due to this we get only 1 vector as output from LSTM cell and as a single vector cannot capture the entire esence of text sequence we got very less accuracy and auc values. But when we include the parameter `return_sequences = True` we get the output from every timestamp which increases the pattern information gained and thereby we got significant improvement in Test Accuracy and Auc.
5. From the training results we can observe that model is not overfitted and achieved

validation auc = 0.7323