Classification of project approval on DonorsChoose DataSet

1. Preprocess all the Data we have in DonorsChoose Dataset use train.csv
2. Combine 4 essay's into one column named - 'preprocessed_essays'.
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 for using auc as a metric
5. You are free to choose any number of layers/hiddden units but you have to use same ty
6. You can use any one of the optimizers and choice of Learning rate and momentum, resou
7. For all the model's use TensorBoard and plot the Metric value and Loss with epoch. Wh
8. Use Categorical Cross Entropy as Loss to minimize.

```
!wget --header="Host: doc-0o-34-docs.googleusercontent.com" --header="User-Agent: Mozilla/
```

```
--2020-10-01 06:40:13--  https://doc-0o-34-docs.googleusercontent.com/docs/securesc/r
Resolving doc-0o-34-docs.googleusercontent.com (doc-0o-34-docs.googleusercontent.com)
Connecting to doc-0o-34-docs.googleusercontent.com (doc-0o-34-docs.googleusercontent
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/octet-stream]
Saving to: ‘glove_vectors’

glove_vectors           [         <=>        ] 121.60M  22.8MB/s    in 5.3s

2020-10-01 06:40:18 (22.8 MB/s) - ‘glove_vectors’ saved [127506004]
```

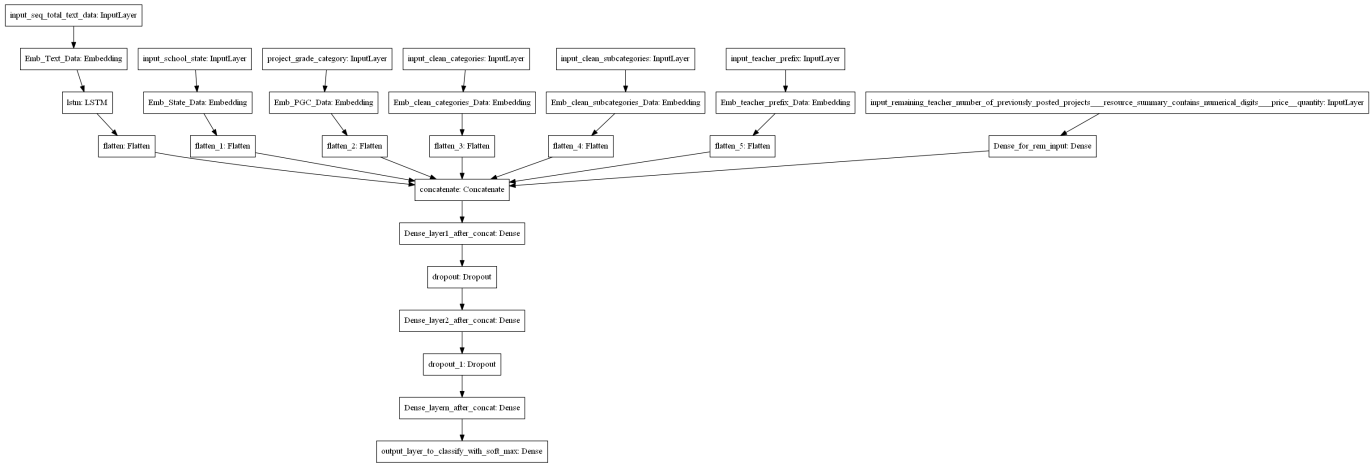
```
!wget --header="Host: doc-10-34-docs.googleusercontent.com" --header="User-Agent: Mozilla/
```

```
--2020-10-01 06:41:29--  https://doc-10-34-docs.googleusercontent.com/docs/securesc/r
Resolving doc-10-34-docs.googleusercontent.com (doc-10-34-docs.googleusercontent.com)
Connecting to doc-10-34-docs.googleusercontent.com (doc-10-34-docs.googleusercontent
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/csv]
Saving to: ‘preprocessed_data.csv’

preprocessed_data.c     [     <=>            ] 118.69M  69.5MB/s    in 1.7s

2020-10-01 06:41:32 (69.5 MB/s) - ‘preprocessed_data.csv’ saved [124454659]
```

```
import pandas as pd
import numpy as np
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
```

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.

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

- For LSTM, you can choose your sequence padding methods on your own or you can train your LSTM without padding, there is no restriction on that.

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

```
X_test_essay = sequence.pad_sequences(test_essay_token , maxlen=max_review_length)
```

```
X_train_eassy[0]
```

```
from sklearn.metrics import confusion_matrix
from sklearn import metrics
import pickle
from sklearn.metrics import auc
```

1.2 Splitting data into Train and cross validation(or test): Stratified Sampling

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

```
data.head(1)
```

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previousl
0	ca	mrs	grades_prek_2	

```
#separating y from dataframe

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

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previousl
0	ca	mrs	grades_prek_2	

```
# train test split
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)
print(X_test.shape)
```

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

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense,Input,Activation,BatchNormalization,Dropout,Embe
from tensorflow.keras.models import Model
import random as rn
from sklearn.metrics import roc_auc_score
from sklearn.metrics import f1_score
from tensorflow.keras import layers
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import ModelCheckpoint
#from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from tensorflow.keras.layers import concatenate
```

encoding of class labels : y

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
encoder.fit(y_train)
y_train_encoded = encoder.transform(y_train)
y_test_encoded = encoder.transform(y_test)
y_train_ohe = tf.keras.utils.to_categorical(y_train_encoded)
y_test_ohe = tf.keras.utils.to_categorical(y_test_encoded)
```

encoding categorical features : eassy

```
#https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer
#https://www.analyticsvidhya.com/blog/2020/03/pretrained-word-embeddings-nlp/
tokenizer=tf.keras.preprocessing.text.Tokenizer()
tokenizer.fit_on_texts(X_train["essay"].tolist())
train_eassy_token = tokenizer.texts_to_sequences(X_train["essay"])
test_eassy_token = tokenizer.texts_to_sequences(X_test["essay"])
```

```
#https://www.analyticsvidhya.com/blog/2020/03/pretrained-word-embeddings-nlp/

size_of_vocabulary=len(tokenizer.word_index) + 1 #+1 for padding
print(size_of_vocabulary)
```

```
48125
```

```
# truncate and/or pad input sequences
max_review_length = 600
X_train_eassy = sequence.pad_sequences(train_eassy_token, maxlen=max_review_length)
X_test_essay = sequence.pad_sequences(test_eassy_token, maxlen=max_review_length)
```

```
array([    0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
```

```
X_test_essay[0]
```

```
array([    0,      0,      0,      0,      0,      0,      0,      0,      0,
           0,      0,      0,      0,      0,      0,      0,      0,      0,
           0,      0,      0,      0,      0,      0,      0,      0,      0,
           0,      0,      0,      0,      0,      0,      0,      0,      0,
           0,      0,      0,      0,      0,      0,      0,      0,      0,
           0,      0,      0,      0,      0,      0,      0,      0,      0,
           0,      0,      0,      0,      0,      0,      0,      0,      0,
           0,      0,      0,      0,      0,      0,      0,      0,      0,
           0,      0,      0,      0,      0,      0,      0,      0,      0,
           0,      0,      0,      0,      0,      0,      0,      0,      0,
           0,      0,      0,      0,      0,      0,      0,      0,      0,
           0,      0,      0,      0,      0,      0,      0,      0,      0,
           0,      0,      0,      0,      0,      0,      0,      0,      0,
           0,      0,      0,      0,      0,      0,      0,      0,      0,
           0,      0,      0,      0,      0,      0,      0,      0,      0,
           0,      0,      0,      0,      0,      0,      0,      0,      0,
           0,      0,      0,      0,      0,      0,      0,      0,      0,
           0,      0,      0,      0,      0,      0,      0,      0,      0,
           0,      0,      0,      0,      0,      0,      0,      0,      0,
           0,      0,      0,      0,      0,      0,      0,      0,      0,
```

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pick
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    glove_words= pickle.load(f)
```

```
           0.     0.     0.     0.     0.     0.     0.     0.     0.
```

```
#https://www.analyticsvidhya.com/blog/2020/03/pretrained-word-embeddings-nlp/

# create a weight matrix for words in training docs
embedding_matrix = np.zeros((size_of_vocabulary, 300))

for word, i in tokenizer.word_index.items():
    embedding_vector = glove_words.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

```
           0.     0.     0.     0.     0.     0.     0.     0.     0.
```

encoding categorical features: school_state

```
           0,     0,     0,     0,     0,     0,    54, 10025,   126,
```

```
vectorizer_school_state = CountVectorizer(binary=True)
vectorizer_school_state.fit(X_train['school_state'].values) # fit has to happen only on tr

# we use the fitted CountVectorizer to convert the text to vector
X_train_school_state_ohe = np.array(vectorizer_school_state.transform(X_train['school_stat
X_test_school_state_ohe = np.array(vectorizer_school_state.transform(X_test['school_state'

print("After vectorizations")
print(X_train_school_state_ohe.shape, y_train.shape)
print(X_test_school_state_ohe.shape, y_test.shape)
```

```
After vectorizations
(73196, 51) (73196,)
(36052, 51) (36052,)
```

```
          18,   334,   391,   112,   785,     0,   619,   249,   112,
```

encoding categorical features:teacher_prefix

```
477. 392. 227. 1. 21. 306. 107. 8. 3996.
```

```
vectorizer_teacher_prefix = CountVectorizer(binary=True)
vectorizer_teacher_prefix.fit(X_train['teacher_prefix'].values) # fit has to happen only o

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = np.array(vectorizer_teacher_prefix.transform(X_train['teacher_prefix
X_test_teacher_ohe = np.array(vectorizer_teacher_prefix.transform(X_test['teacher_prefix']

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
```

```
After vectorizations
(73196, 5) (73196,)
(36052, 5) (36052,)
```

encoding categorical features:project_grade_category

```
vectorizer_project_grade_category = CountVectorizer(binary=True)
vectorizer_project_grade_category.fit(X_train['project_grade_category']) # fit has to happ

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = np.array(vectorizer_project_grade_category.transform(X_train['project_
X_test_grade_ohe = np.array(vectorizer_project_grade_category.transform(X_test['project_gr

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer_project_grade_category.get_feature_names())
print("="*100)
```

```
After vectorizations
(73196, 4) (73196,)
(36052, 4) (36052,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
====================================================================================
```

encoding categorical features: clean_categories

```
vectorizer_clean_categories = CountVectorizer(binary=True)
vectorizer_clean_categories.fit(X_train['clean_categories']) # fit has to happen only on t

# we use the fitted CountVectorizer to convert the text to vector
X_train_category_ohe= np.array(vectorizer_clean_categories.transform(X_train['clean_catego
X_test_category_ohe = np.array(vectorizer_clean_categories.transform(X_test['clean_categor

print("After vectorizations")
print(X_train_category_ohe.shape, y_train.shape)
```

```
print(X_test_category_ohe.shape, y_test.shape)
print(vectorizer_clean_categories.get_feature_names())
print("="*100)
```

```
After vectorizations
(73196, 9) (73196,)
(36052, 9) (36052,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_langu
==========================================================================================
```

encoding categorical features:clean_subcategories

```
vectorizer_clean_subcategories = CountVectorizer(binary=True)
vectorizer_clean_subcategories.fit(X_train['clean_subcategories']) # fit has to happen onl

# we use the fitted CountVectorizer to convert the text to vector
X_train_subcategory_ohe = np.array(vectorizer_clean_subcategories.transform(X_train['clean
X_test_subcategory_ohe = np.array(vectorizer_clean_subcategories.transform(X_test['clean_s

print("After vectorizations")
print(X_train_subcategory_ohe.shape, y_train.shape)
print(X_test_subcategory_ohe.shape, y_test.shape)
print(vectorizer_clean_subcategories.get_feature_names())
print("="*100)
```

```
After vectorizations
(73196, 30) (73196,)
(36052, 30) (36052,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'colleg
==========================================================================================
```

encoding categorical features: numerical featufeatures

```
train_remaining = np.array(pd.concat([X_train['price'],X_train['teacher_number_of_previous
test_remaining =  np.array(pd.concat([X_test['price'],X_test['teacher_number_of_previously
print(train_remaining.shape)
print(test_remaining.shape)
```

```
(73196, 2)
(36052, 2)
```

model1

```
tf.keras.backend.clear_session()
#input layer 1
input1 = Input(shape=(600,))
```

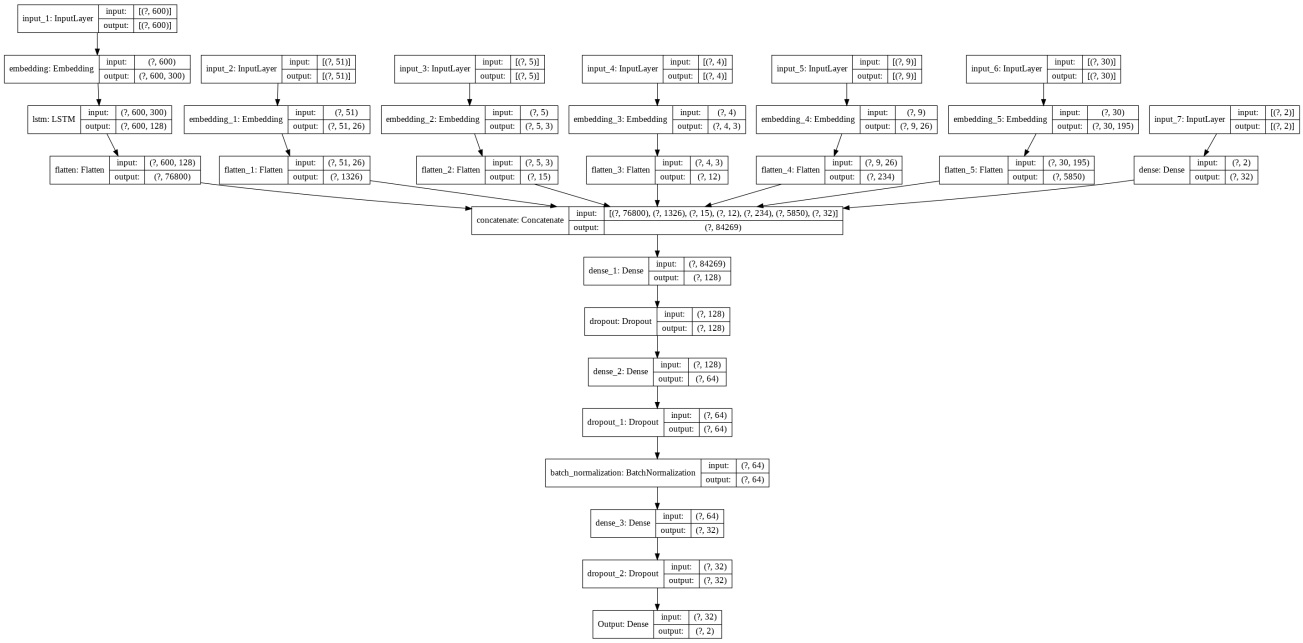
```
input1 = Input(shape=(600,))
#embedding layer
embedding = Embedding(size_of_vocabulary,300,weights=[embedding_matrix],input_length=600,t

#lstm layer

LSTM_layer= LSTM(128,return_sequences=True,dropout=0.2)(embedding)
flatten1= Flatten()(LSTM_layer)

#input layer 2
input2 = Input(shape=(51,))

embed_input_school_state = X_train['school_state'].nunique()
embed_out_school_state= embed_input_school_state//2+1
Embedding_layer2= Embedding(input_dim= embed_input_school_state,output_dim= embed_out_scho
flatten2= Flatten()(Embedding_layer2)
# input layer 3
input3 = Input(shape=(5,))

embed_input_school_state = X_train['teacher_prefix'].nunique()
embed_out_school_state= embed_input_school_state//2+1
Embedding_layer3= Embedding(input_dim= embed_input_school_state,output_dim= embed_out_scho
flatten3= Flatten()(Embedding_layer3)

# input layer 4

input4 = Input(shape=(4,))

embed_input_school_state = X_train['project_grade_category'].nunique()
embed_out_school_state= embed_input_school_state//2+1
Embedding_layer4= Embedding(input_dim= embed_input_school_state,output_dim= embed_out_scho
flatten4= Flatten()(Embedding_layer4)

# input layer 5

input5 = Input(shape=(9,))

embed_input_school_state = X_train['clean_categories'].nunique()
embed_out_school_state= embed_input_school_state//2+1
Embedding_layer5= Embedding(input_dim= embed_input_school_state,output_dim= embed_out_scho
flatten5 = Flatten()(Embedding_layer5)

# input layer 6

input6 = Input(shape=(30,))

embed_input_school_state = X_train['clean_subcategories'].nunique()
embed_out_school_state= embed_input_school_state//2+1
Embedding_layer6= Embedding(input_dim= embed_input_school_state,output_dim= embed_out_scho
flatten6 = Flatten()(Embedding_layer6)

#input layer 7

input7 = Input(shape=(2,))
input_numerical = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.he_n
```

```
#concatenation of  all the inputs
concat = concatenate([flatten1,flatten2,flatten3,flatten4,flatten5,flatten6,input_numerica

# dense layer1
dense1 = Dense(128,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(se
drop1 =Dropout(0.5)(dense1)

#dense layer 2
dense2 = Dense(64,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(see
drop2=Dropout(0.5)(dense2)
# dense layer3
batch_norm = BatchNormalization()(drop2)
dense3 = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(see
drop3= Dropout(0.5)(dense3)
#output layer
Out = Dense(units=2,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_n

model1= Model(inputs=[input1,input2,input3,input4,input5,input6,input7],outputs=Out)
```

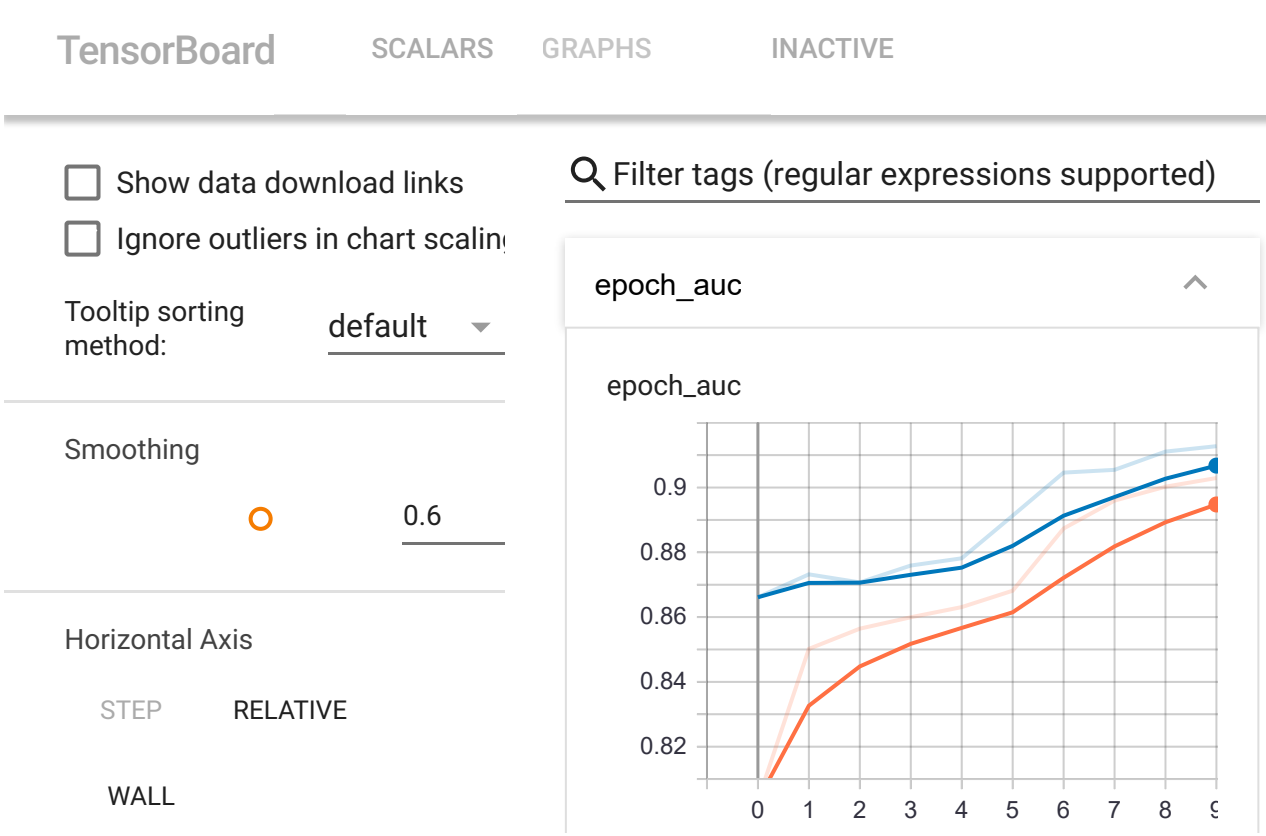
```
model1.summary()
```

```
Model: "functional_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 600)]        0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 600, 300)     14475000    input_1[0][0]
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 51)]         0
__________________________________________________________________________________________________
input_3 (InputLayer)            [(None, 5)]          0
__________________________________________________________________________________________________
input_4 (InputLayer)            [(None, 4)]          0
__________________________________________________________________________________________________
input_5 (InputLayer)            [(None, 9)]          0
__________________________________________________________________________________________________
input_6 (InputLayer)            [(None, 30)]         0
__________________________________________________________________________________________________
lstm (LSTM)                     (None, 600, 128)     219648      embedding[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 51, 26)       1326        input_2[0][0]
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 5, 3)         15          input_3[0][0]
__________________________________________________________________________________________________
embedding_3 (Embedding)         (None, 4, 3)         12          input_4[0][0]
__________________________________________________________________________________________________
embedding_4 (Embedding)         (None, 9, 26)        1326        input_5[0][0]
__________________________________________________________________________________________________
embedding_5 (Embedding)         (None, 30, 195)      75855       input_6[0][0]
__________________________________________________________________________________________________
input_7 (InputLayer)            [(None, 2)]          0
__________________________________________________________________________________________________
flatten (Flatten)               (None, 76800)        0           lstm[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 1326)         0           embedding_1[0][0]
__________________________________________________________________________________________________
flatten_2 (Flatten)             (None, 15)           0           embedding_2[0][0]
__________________________________________________________________________________________________
flatten_3 (Flatten)             (None, 12)           0           embedding_3[0][0]
__________________________________________________________________________________________________
flatten_4 (Flatten)             (None, 234)          0           embedding_4[0][0]
__________________________________________________________________________________________________
flatten_5 (Flatten)             (None, 5850)         0           embedding_5[0][0]
```

```
# summarize the model
from tensorflow.keras.utils import plot_model
plot_model(model1, 'model.png', show_shapes=True)
```




```
#https://stackoverflow.com/questions/43263111/defining-an-auc-metric-for-keras-to-support-
def auc( y_true, y_pred ) :
    score = tf.py_function( lambda y_true, y_pred : roc_auc_score( y_true, y_pred, average
    return score


import os
import datetime
%load_ext tensorboard
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)
%tensorboard --logdir $logdir
```

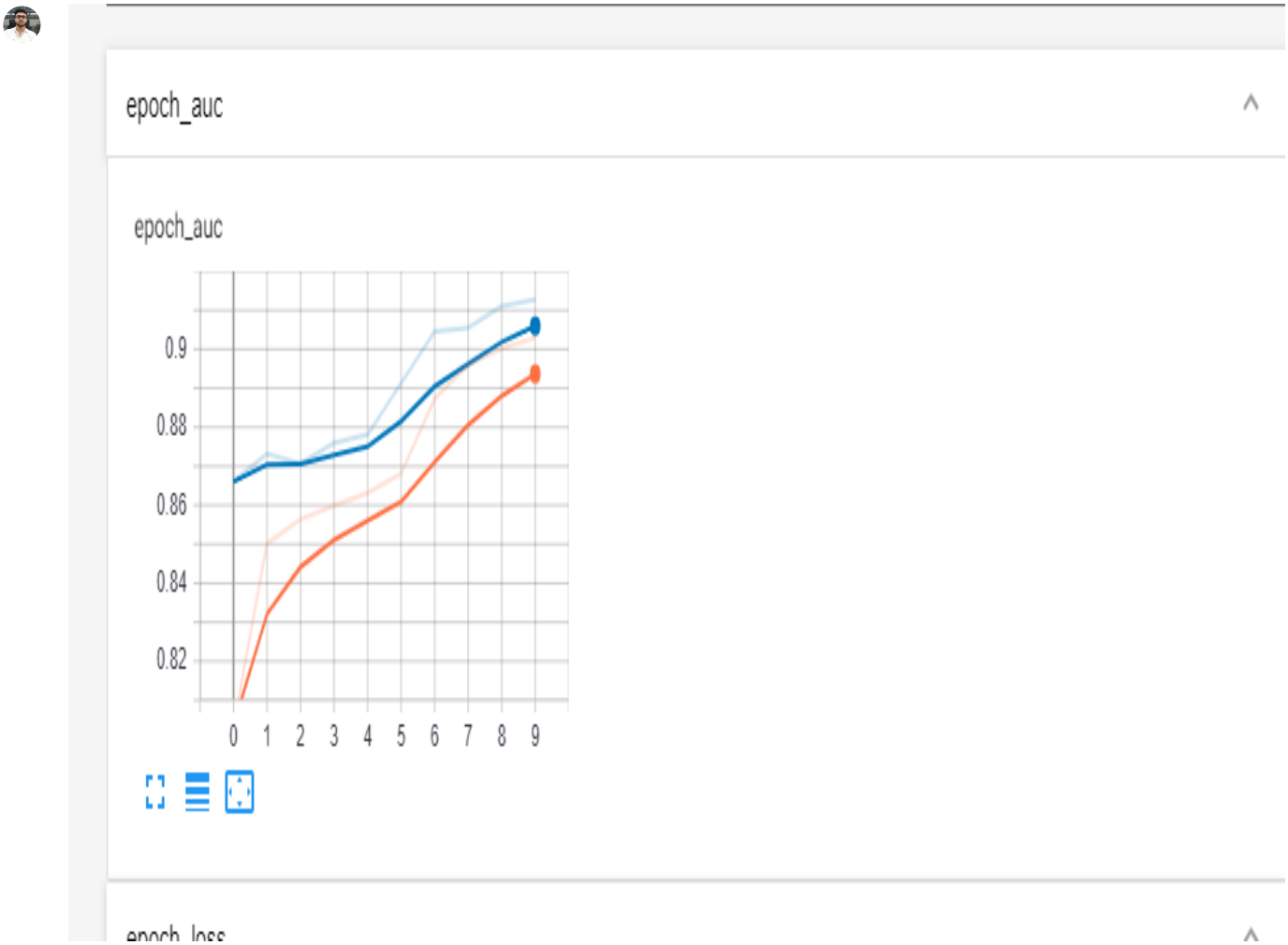



```
#compiling
model1.compile(optimizer=tf.keras.optimizers.Adam(lr=0.001),loss='categorical_crossentropy
```

```
model1.fit([X_train_eassy,X_train_school_state_ohe,X_train_teacher_ohe,X_train_grade_ohe,
            X_train_category_ohe,X_train_subcategory_ohe,train_remaining],y_train_ohe,epoc
validation_data=([X_test_essay,X_test_school_state_ohe,X_test_teacher_ohe,X_test_grade_ohe
,callbacks=[tensorboard_callback])
```

```
Epoch 1/10
 1/72 [..............................] - ETA: 8s - loss: 1.3695 - auc: 0.6536WARNING
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
72/72 [==============================] - 42s 582ms/step - loss: 0.8786 - auc: 0.8034
Epoch 2/10
72/72 [==============================] - 41s 571ms/step - loss: 0.6850 - auc: 0.8502
Epoch 3/10
72/72 [==============================] - 41s 574ms/step - loss: 0.6236 - auc: 0.8564
Epoch 4/10
72/72 [==============================] - 41s 572ms/step - loss: 0.5848 - auc: 0.8599
Epoch 5/10
72/72 [==============================] - 41s 572ms/step - loss: 0.5570 - auc: 0.8631
Epoch 6/10
72/72 [==============================] - 41s 570ms/step - loss: 0.5354 - auc: 0.8681
Epoch 7/10
72/72 [==============================] - 41s 570ms/step - loss: 0.5066 - auc: 0.8873
Epoch 8/10
72/72 [==============================] - 41s 570ms/step - loss: 0.4827 - auc: 0.8960
Epoch 9/10
72/72 [==============================] - 41s 568ms/step - loss: 0.4648 - auc: 0.9003
Epoch 10/10
72/72 [==============================] - 41s 569ms/step - loss: 0.4493 - auc: 0.9029
<tensorflow.python.keras.callbacks.History at 0x7fa782587eb8>
```

```
from IPython.display import Image
Image('/content/Screenshot (205).png',width=800,height=500)
```




```
from IPython.display import Image
Image('/content/Screenshot (206).png',width=800,height=500)
```

```
input2 = Input(shape=(51,))

embed_input_school_state = X_train['school_state'].nunique()
embed_out_school_state= embed_input_school_state//2+1
Embedding_layer2= Embedding(input_dim= embed_input_school_state,output_dim= embed_out_scho
flatten2= Flatten()(Embedding_layer2)
# input layer 3
input3 = Input(shape=(5,))

embed_input_school_state = X_train['teacher_prefix'].nunique()
embed_out_school_state= embed_input_school_state//2+1
Embedding_layer3= Embedding(input_dim= embed_input_school_state,output_dim= embed_out_scho
flatten3= Flatten()(Embedding_layer3)

# input layer 4

input4 = Input(shape=(4,))

embed_input_school_state = X_train['project_grade_category'].nunique()
embed_out_school_state= embed_input_school_state//2+1
Embedding_layer4= Embedding(input_dim= embed_input_school_state,output_dim= embed_out_scho
flatten4= Flatten()(Embedding_layer4)

# input layer 5

input5 = Input(shape=(9,))

embed_input_school_state = X_train['clean_categories'].nunique()
embed_out_school_state= embed_input_school_state//2+1
Embedding_layer5= Embedding(input_dim= embed_input_school_state,output_dim= embed_out_scho
flatten5 = Flatten()(Embedding_layer5)

# input layer 6

input6 = Input(shape=(30,))

embed_input_school_state = X_train['clean_subcategories'].nunique()
embed_out_school_state= embed_input_school_state//2+1
Embedding_layer6= Embedding(input_dim= embed_input_school_state,output_dim= embed_out_scho
flatten6 = Flatten()(Embedding_layer6)

#input layer 7

input7 = Input(shape=(2,))
input_numerical = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.he_n

#concatenation of  all the inputs
concat = concatenate([flatten1,flatten2,flatten3,flatten4,flatten5,flatten6,input_numerica

# dense layer1
dense1 = Dense(128,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(se
drop1 =Dropout(0.5)(dense1)

#dense layer 2
dense2 = Dense(64,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(see
```

Model-2

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.

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.

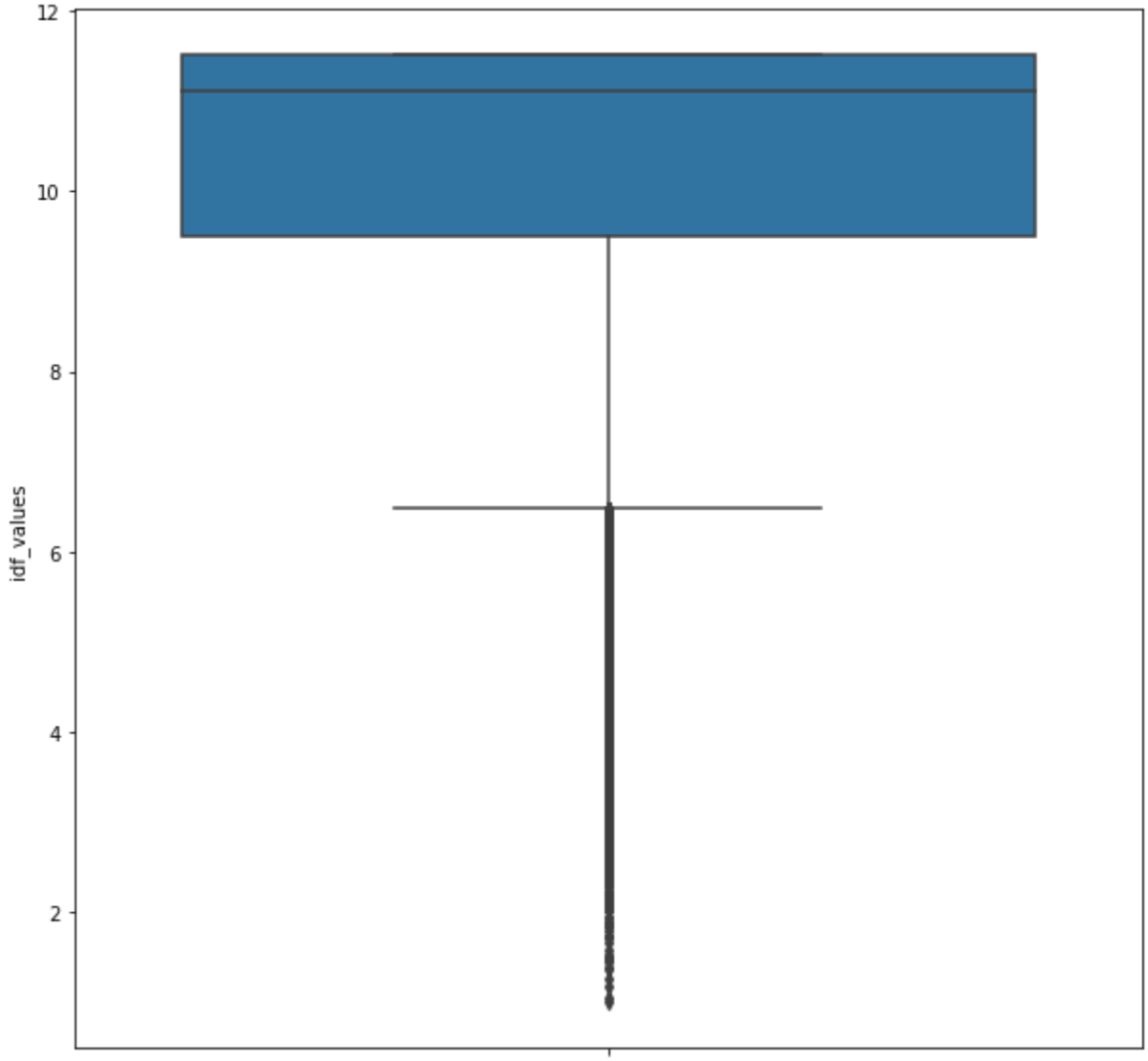
1. Train the TF-IDF on the Train data
2. Get the idf value for each word we have in the train data.
3. Remove the low idf value and high idf value words from our data. Do some analysis on the Idf values and based on those values choose the low and high threshold value. Because very frequent words and very very rare words don't give much information. (you can plot a box plots and take only the idf scores within IQR range and corresponding words)
4. Train the LSTM after removing the Low and High idf value words. (In model-1 Train on total data but in Model-2 train on data after removing some words based on IDF values)

```
vectorizer = TfidfVectorizer()
tf_idf= vectorizer.fit(X_train['essay'])
```

```
idf_df = pd.DataFrame(tf_idf.idf_, columns= ["idf_values"])
#box plot of idf_values
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10,10))
```

```
plt.figure(figsize=(10,10))


sns.boxplot(y = "idf_values", data = idf_df )
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa781b545f8>
```




```
q1 = idf_df['idf_values'].quantile(0.25)
q3 = idf_df['idf_values'].quantile(0.75)
print("25 percentile of idf score is :", q1)
print("75 percentile of idf score is :",q3)
```

```
25 percentile of idf score is : 9.492859514400784
75 percentile of idf score is : 11.50776253494305
```

```
All_words= zip(tf_idf.get_feature_names(),tf_idf.idf_)

IQR_range_words = []
for i,j in All_words :

    if j >= q1 and j <= q3:
        IQR_range_words.append(i)

print(len(IQR_range_words))
```

```
36225
```

```
type(IQR_range_words)
```

```
list
```

```
#https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer
#https://www.analyticsvidhya.com/blog/2020/03/pretrained-word-embeddings-nlp/
tokenizer=tf.keras.preprocessing.text.Tokenizer()
tokenizer.fit_on_texts(IQR_range_words)
train_eassy_token = tokenizer.texts_to_sequences(X_train["essay"])
test_eassy_token = tokenizer.texts_to_sequences(X_test["essay"])
```

```
#https://www.analyticsvidhya.com/blog/2020/03/pretrained-word-embeddings-nlp/

size_of_vocabulary=len(tokenizer.word_index) + 1 #+1 for padding
print(size_of_vocabulary)
```

```
36226
```

```
# truncate and/or pad input sequences
max_review_length = 250
X_train_eassy_new = sequence.pad_sequences(train_eassy_token, maxlen=max_review_length)
X_test_essay_new = sequence.pad_sequences(test_eassy_token , maxlen=max_review_length)
```

```
X_train_eassy_new[25]
```

```
array([    0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
```

```
X_test_essay_new[10]
```

```
array([    0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0, 15780, 15779, 15780, 26706, 19381, 15779], dtype=int32)
```

```
#https://www.analyticsvidhya.com/blog/2020/03/pretrained-word-embeddings-nlp/

# create a weight matrix for words in training docs
embedding_matrix1 = np.zeros((size_of_vocabulary, 300))

for word, i in tokenizer.word_index.items():
    embedding_vector = glove_words.get(word)
    if embedding_vector is not None:
        embedding_matrix1[i] = embedding_vector
```

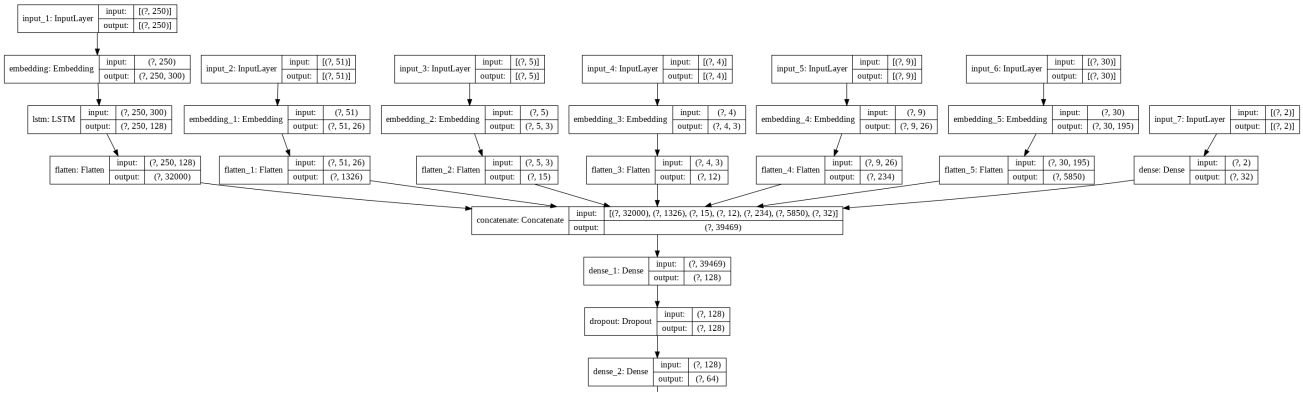
```
tf.keras.backend.clear_session()
#input layer 1
input1 = Input(shape=(250,))
#embedding layer
embedding = Embedding(size_of_vocabulary,300,weights=[embedding_matrix1],input_length=250,

#lstm layer

LSTM_layer= LSTM(128,return_sequences=True,dropout=0.2)(embedding)
flatten1= Flatten()(LSTM_layer)

#input layer 2
```

```
drop2=Dropout(0.5)(dense2)
# dense layer3
batch_norm = BatchNormalization()(drop2)
dense3 = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(see
drop3= Dropout(0.5)(dense3)
#output layer
Out = Dense(units=2,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_n

model2= Model(inputs=[input1,input2,input3,input4,input5,input6,input7],outputs=Out)
```

```
model2.summary()
```

```
Model: "functional_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 250)]        0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 250, 300)     10867800    input_1[0][0]
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 51)]         0
__________________________________________________________________________________________________
input_3 (InputLayer)            [(None, 5)]          0
__________________________________________________________________________________________________
input_4 (InputLayer)            [(None, 4)]          0
__________________________________________________________________________________________________
input_5 (InputLayer)            [(None, 9)]          0
__________________________________________________________________________________________________
input_6 (InputLayer)            [(None, 30)]         0
__________________________________________________________________________________________________
lstm (LSTM)                     (None, 250, 128)     219648      embedding[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 51, 26)       1326        input_2[0][0]
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 5, 3)         15          input_3[0][0]
__________________________________________________________________________________________________
embedding_3 (Embedding)         (None, 4, 3)         12          input_4[0][0]
__________________________________________________________________________________________________
embedding_4 (Embedding)         (None, 9, 26)        1326        input_5[0][0]
__________________________________________________________________________________________________
embedding_5 (Embedding)         (None, 30, 195)      75855       input_6[0][0]
__________________________________________________________________________________________________
input_7 (InputLayer)            [(None, 2)]          0
```

```
# summarize the model
from tensorflow.keras.utils import plot_model
plot_model(model2, 'model.png', show_shapes=True)
```

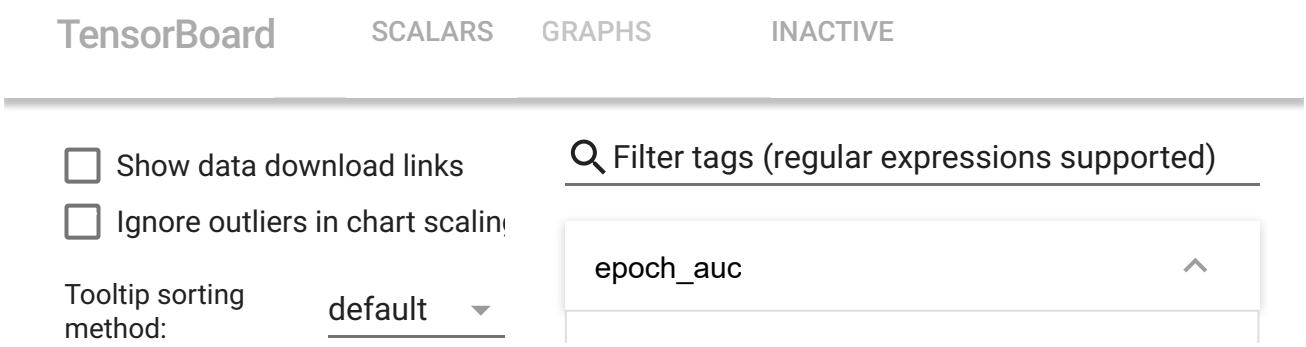



```
#compiling
model2.compile(optimizer=tf.keras.optimizers.Adam(lr=0.001),loss='categorical_crossentropy
```


```
import os
import datetime
%load_ext tensorboard
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)
%tensorboard --logdir $logdir
```

```
#input layer 2
input_2 = Input(shape=(101,1,))

#Conv Layer
Conv1 = Conv1D(filters=32,kernel_size=7,strides=1,padding='valid',data_format='channels_la
              activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=0)


Conv2 = Conv1D(filters=16,kernel_size=3,strides=1,padding='valid',data_format='channels_la
              activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=0)
flatten2= Flatten()(Conv2)

#concatenation of  all the inputs
concat = concatenate([flatten1,flatten2])

# dense layer1
dense1 = Dense(128,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(se
drop1 =Dropout(0.5)(dense1)

#dense layer 2
dense2 = Dense(64,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(see
drop2=Dropout(0.5)(dense2)
# dense layer3
batch_norm = BatchNormalization()(drop2)
dense3 = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(see
drop3= Dropout(0.5)(dense3)
#output layer
Out = Dense(units=2,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_n

model3= Model(inputs=[input_1,input_2],outputs=Out)



#MaxPool Layer


model3.summary()
```

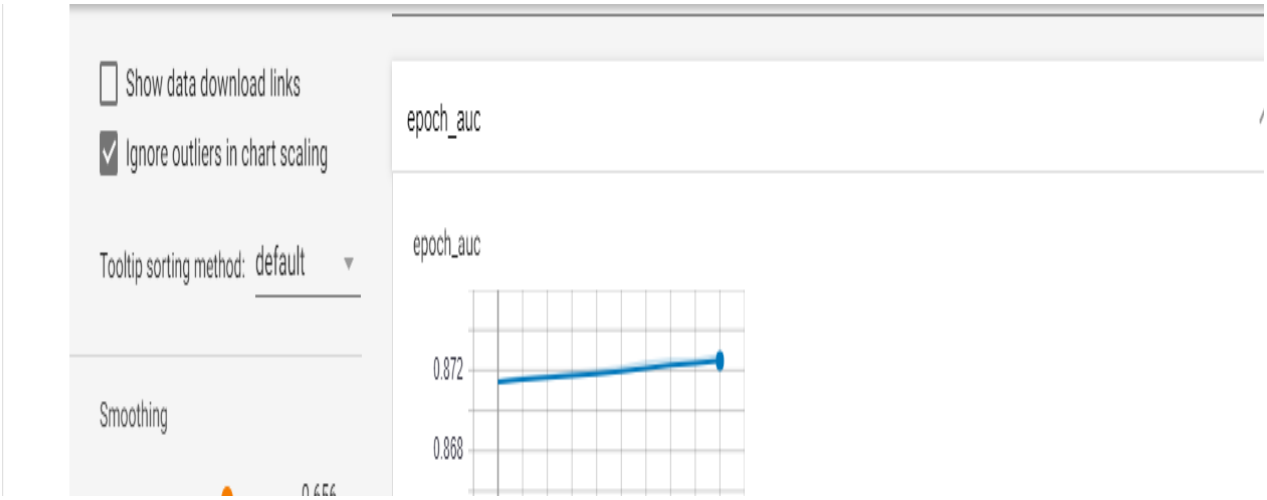
```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```



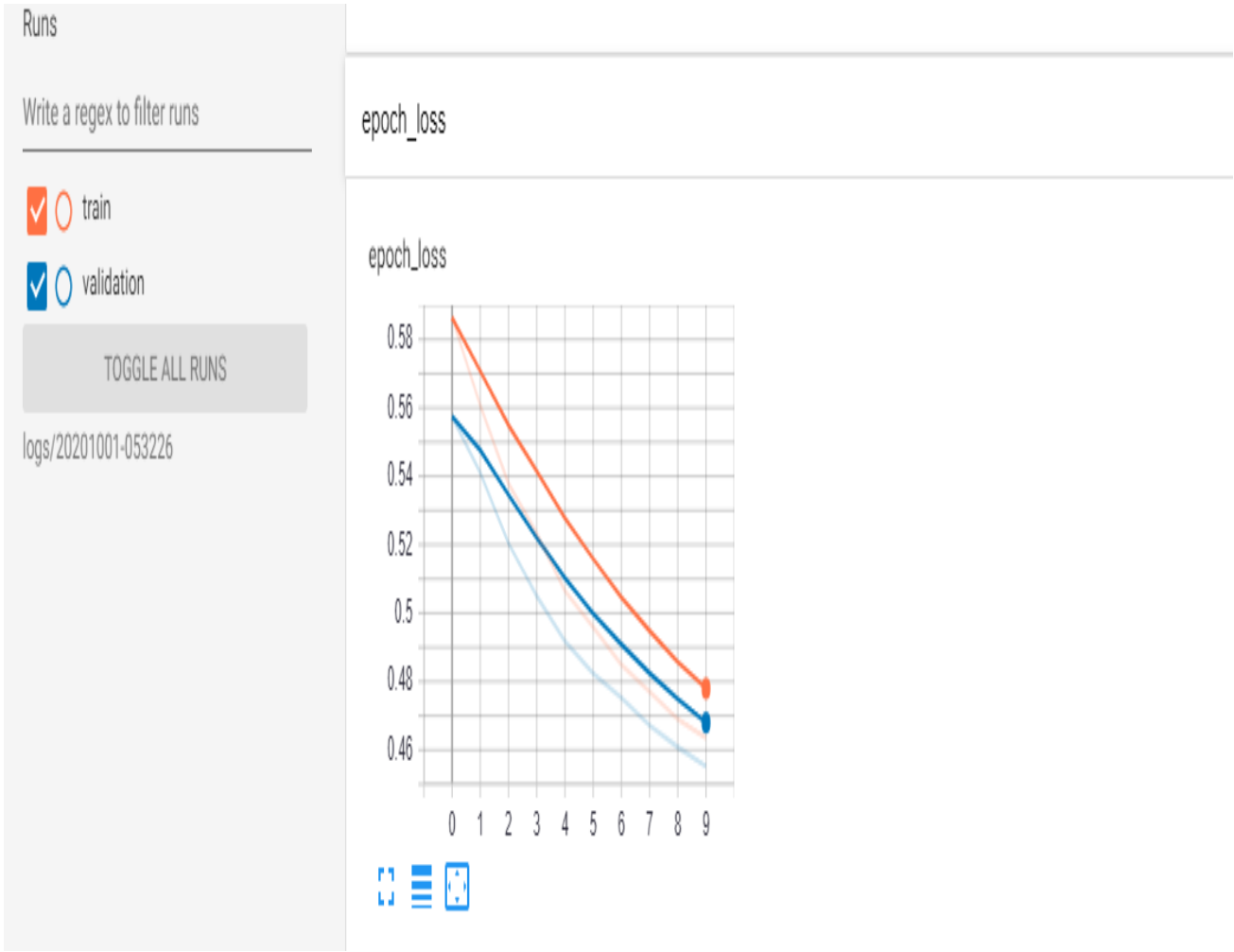

```
model2.fit([X_train_eassy_new,X_train_school_state_ohe,X_train_teacher_ohe,X_train_grade_o
            X_train_category_ohe,X_train_subcategory_ohe,train_remaining],y_train_ohe,epoc
validation_data=([X_test_essay_new,X_test_school_state_ohe,X_test_teacher_ohe,X_test_grade
,callbacks=[tensorboard_callback])
```

```
Epoch 1/10
72/72 [==============================] - 19s 265ms/step - loss: 0.5863 - auc: 0.8579
Epoch 2/10
72/72 [==============================] - 19s 262ms/step - loss: 0.5606 - auc: 0.8577
Epoch 3/10
72/72 [==============================] - 19s 260ms/step - loss: 0.5378 - auc: 0.8612
Epoch 4/10
72/72 [==============================] - 19s 261ms/step - loss: 0.5232 - auc: 0.8593
Epoch 5/10
72/72 [==============================] - 19s 261ms/step - loss: 0.5064 - auc: 0.8622
Epoch 6/10
72/72 [==============================] - 19s 261ms/step - loss: 0.4959 - auc: 0.8621
Epoch 7/10
72/72 [==============================] - 19s 260ms/step - loss: 0.4849 - auc: 0.8640
Epoch 8/10
72/72 [==============================] - 19s 260ms/step - loss: 0.4771 - auc: 0.8636
Epoch 9/10
72/72 [==============================] - 19s 260ms/step - loss: 0.4690 - auc: 0.8650
Epoch 10/10
72/72 [==============================] - 19s 262ms/step - loss: 0.4635 - auc: 0.8649
<tensorflow.python.keras.callbacks.History at 0x7fa49187deb8>
```

```
from IPython.display import Image
Image('/content/Screenshot (209).png',width=800,height=500)
```




```
from IPython.display import Image
Image('/content/Screenshot (210).png',width=800,height=500)
```



model3

- **input_seq_total_text_data**:

```
. Use text column('essay'), and use the Embedding layer to get word ve
```

. Use given predefined glove word vectors, don't train any word vectors

. Use LSTM that is given above, get the LSTM output and Flatten that ou

. You are free to preprocess the input text as you needed.

- **Other_than_text_data**:

. Convert all your Categorical values to onehot coded and then concate

. Neumerical values and use CNN1D as shown in above figure.

. You are free to choose all CNN parameters like kernel sizes, stride.

```
#https://www.analyticsvidhya.com/blog/2020/03/pretrained-word-embeddings-nlp/

# create a weight matrix for words in training docs
embedding_matrix1 = np.zeros((size_of_vocabulary, 300))

for word, i in tokenizer.word_index.items():
    embedding_vector = glove_words.get(word)
    if embedding_vector is not None:
        embedding_matrix1[i] = embedding_vector
```

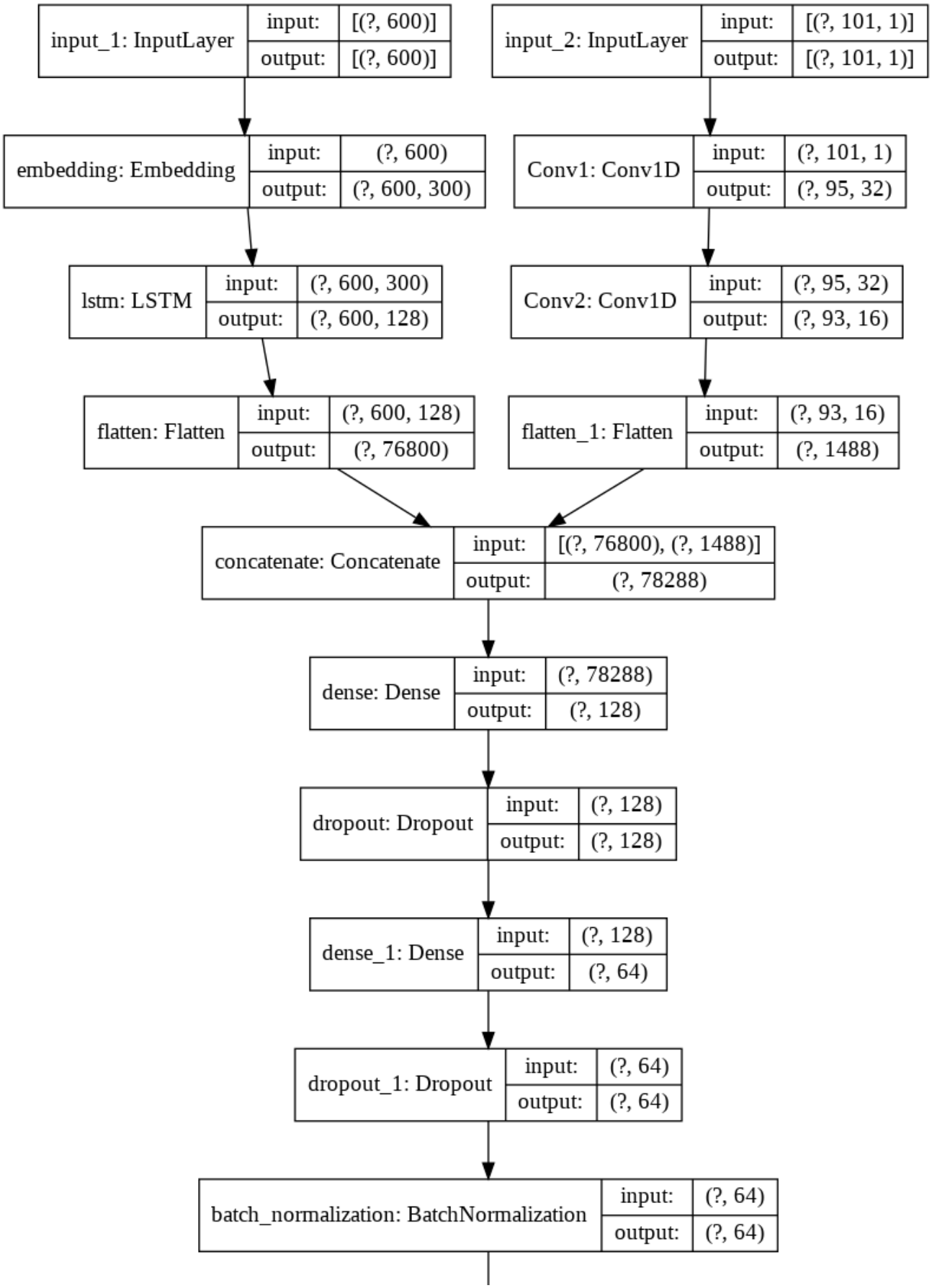
```
X_train_categorical_numerical =np.hstack((X_train_school_state_ohe,X_train_teacher_ohe,X_t
X_test_categorical_numerical= np.hstack((X_test_school_state_ohe,X_test_teacher_ohe,X_test
print(X_train_categorical_numerical.shape)
print(X_test_categorical_numerical.shape)
```

```
(73196, 101)
(36052, 101)
```

```
tf.keras.backend.clear_session()
#input layer 1
input_1 = Input(shape=(600,))
#embedding layer
embedding = Embedding(size_of_vocabulary,300,weights=[embedding_matrix],input_length=600,t

#lstm layer

LSTM_layer= LSTM(128,return_sequences=True,dropout=0.2)(embedding)
flatten1= Flatten()(LSTM_layer)
```

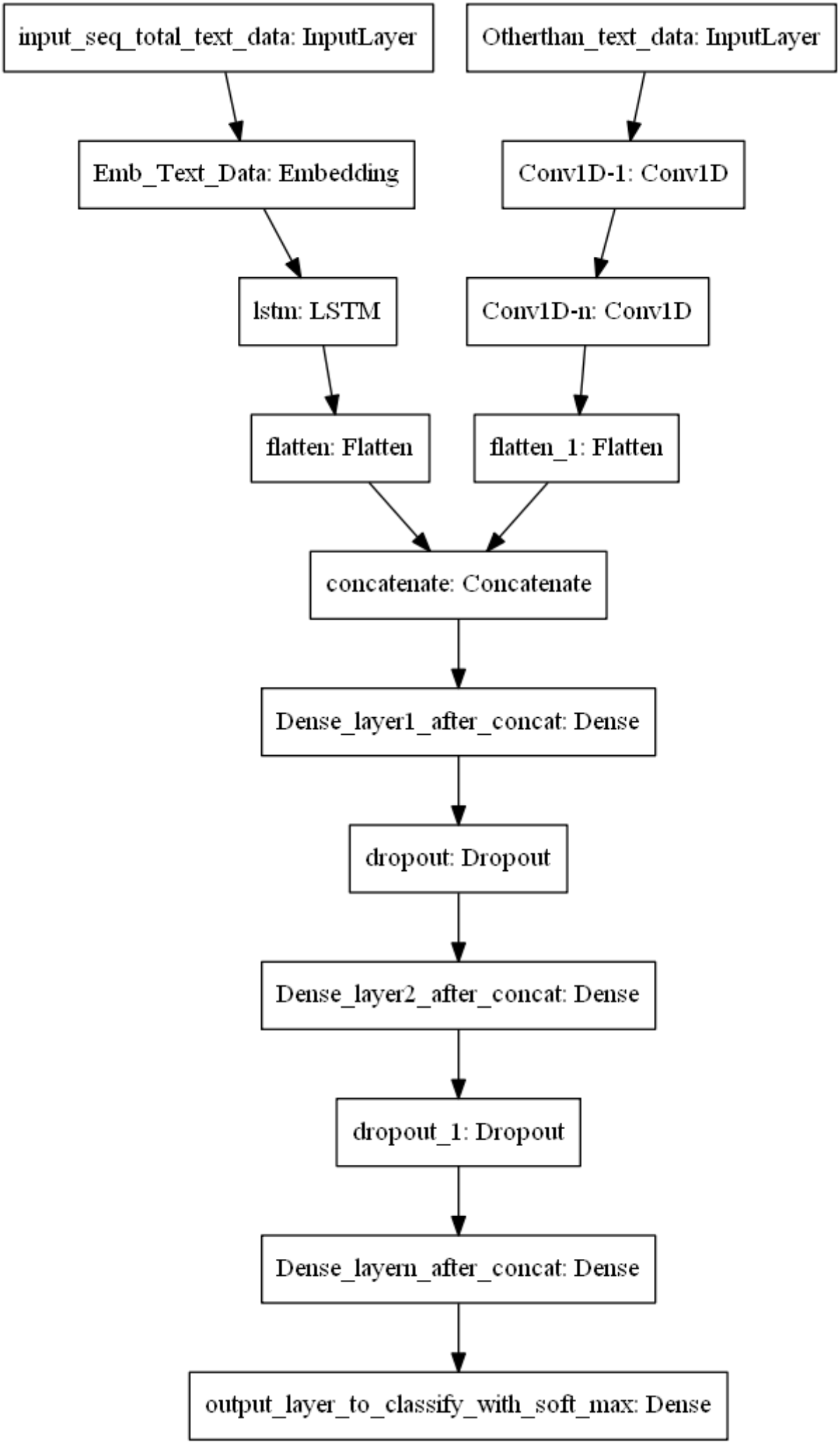
```
Model: "functional_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 600)]        0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 101, 1)]     0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 600, 300)     14437500    input_1[0][0]
__________________________________________________________________________________________________
Conv1 (Conv1D)                  (None, 95, 32)       256         input_2[0][0]
__________________________________________________________________________________________________
lstm (LSTM)                     (None, 600, 128)     219648      embedding[0][0]
__________________________________________________________________________________________________
Conv2 (Conv1D)                  (None, 93, 16)       1552        Conv1[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 76800)        0           lstm[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 1488)         0           Conv2[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 78288)        0           flatten[0][0]
                                                                 flatten_1[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 128)          10020992    concatenate[0][0]
__________________________________________________________________________________________________
```

```
# summarize the model
from tensorflow.keras.utils import plot_model
plot_model(model3, 'model.png', show_shapes=True)
```




```
#https://stackoverflow.com/questions/43263111/defining-an-auc-metric-for-keras-to-support-
def auc( y_true, y_pred ) :
    score = tf.py_function( lambda y_true, y_pred : roc_auc_score( y_true, y_pred, average
    return score
```

```
import os
import datetime
```


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

Observation:-

In this assigment we are predicting that a porject will be aproved or not on donors choose data set

Model1

1) Using level encoder we have done leveling of calss levals y

2) for feauter essay we have used tf.keras to tokenize text and pad sequence to convert it into sequence after that used pretrained glove vector for embedding

3) for cagtegorical feature we have used one hot encoding to vectoriz and after that used embedding layer

4)Remaining feature numericals just mearged it and passed it into embedding layer

5)created the LSTM model , and used auc as Performance metrics

**Obervation in tensorboard :

tarin test auc is almost same so no overfitting or underfitting

Model2

1)In model2 for essay feature trained tf_idf on train data

2) got the idf value of each word and removed the word which have low idf value

3)Others features are same as model1

4)created LSTM model as in model1

```
import datetime
%load_ext tensorboard
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)
%tensorboard --logdir $logdir
```

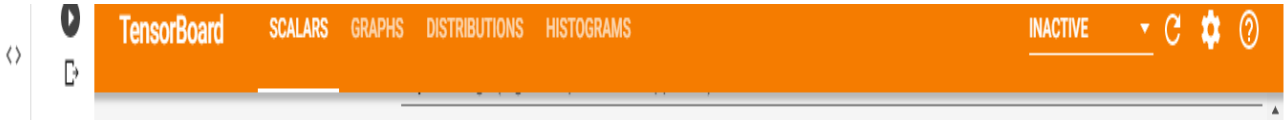
```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```


```
#compiling
model3.compile(optimizer=tf.keras.optimizers.Adam(lr=0.001),loss='categorical_crossentropy
```

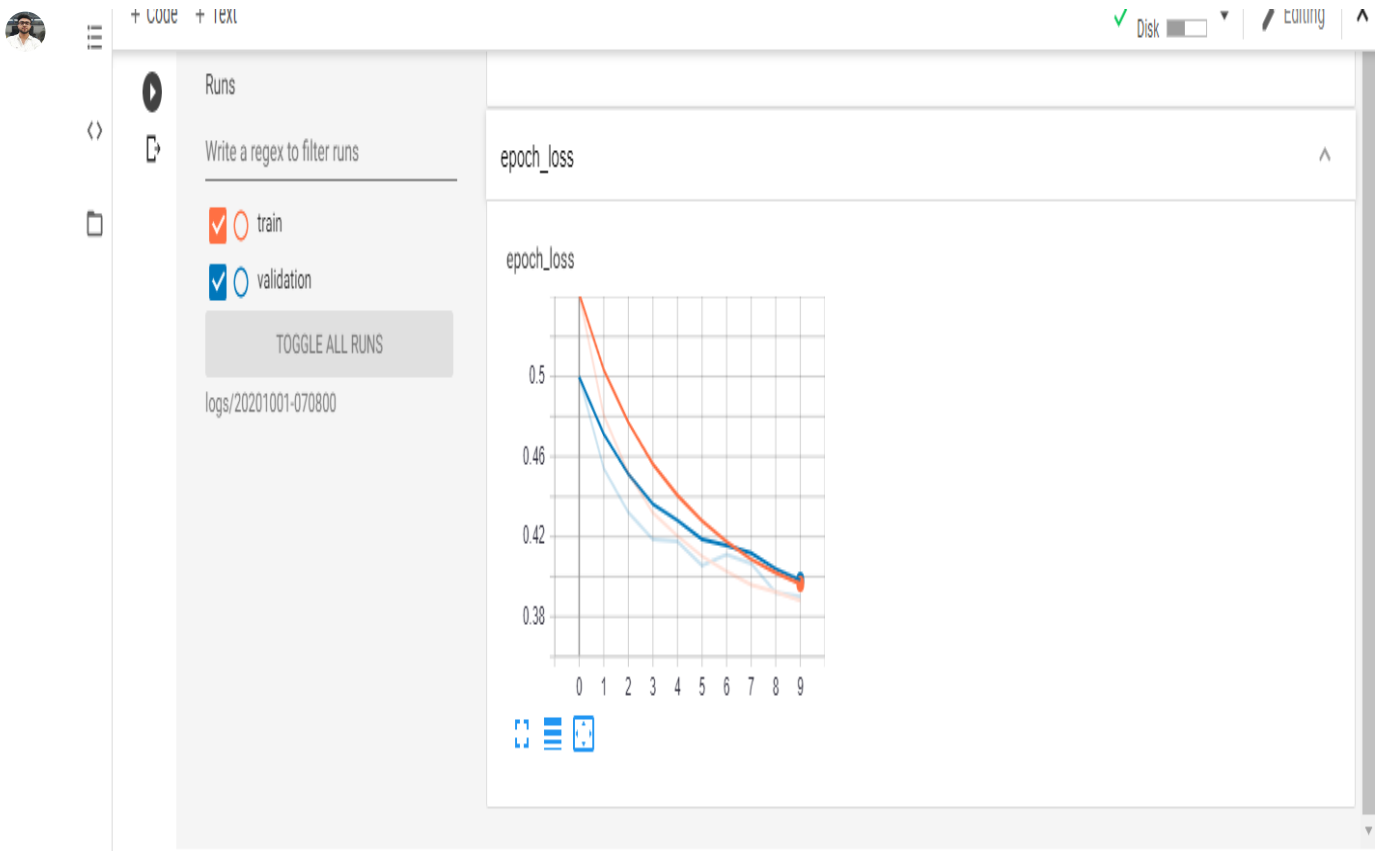
```
model3.fit([X_train_eassy,X_train_categorical_numerical],y_train_ohe,epochs=10,batch_size=
validation_data=([X_test_essay,X_test_categorical_numerical],y_test_ohe)
,callbacks=[tensorboard_callback])
```

```
Epoch 1/10
 1/72 [..............................] - ETA: 10s - loss: 0.5902 - auc: 0.8403WARNIN(
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
72/72 [==============================] - 48s 661ms/step - loss: 0.5411 - auc: 0.8594
Epoch 2/10
72/72 [==============================] - 48s 668ms/step - loss: 0.4807 - auc: 0.8857
Epoch 3/10
72/72 [==============================] - 49s 682ms/step - loss: 0.4517 - auc: 0.8953
Epoch 4/10
72/72 [==============================] - 50s 696ms/step - loss: 0.4318 - auc: 0.9019
Epoch 5/10
72/72 [==============================] - 51s 703ms/step - loss: 0.4202 - auc: 0.9054
Epoch 6/10
72/72 [==============================] - 51s 707ms/step - loss: 0.4102 - auc: 0.9096
Epoch 7/10
72/72 [==============================] - 51s 710ms/step - loss: 0.4026 - auc: 0.9119
Epoch 8/10
72/72 [==============================] - 51s 708ms/step - loss: 0.3959 - auc: 0.9132
Epoch 9/10
72/72 [==============================] - 51s 706ms/step - loss: 0.3923 - auc: 0.9152
Epoch 10/10
72/72 [==============================] - 51s 703ms/step - loss: 0.3880 - auc: 0.9169
<tensorflow.python.keras.callbacks.History at 0x7f14e2760710>
```

```
from IPython.display import Image
Image('/content/Screenshot (213).png',width=800,height=500)
```




```
from IPython.display import Image
Image('/content/Screenshot (214).png',width=800,height=500)
```




```
[51] #compiling
     model3.compile(optimizer=tf.keras.optimizers.Adam(lr=0.001),loss='categorical_crossentropy',metrics=[auc])
```