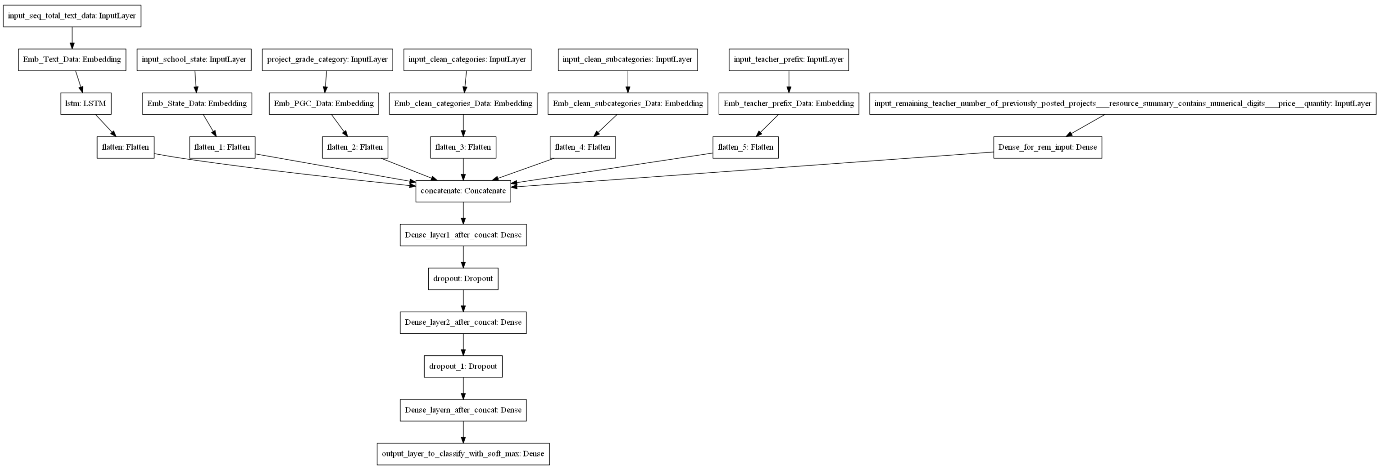
# Assignment : 14

```
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 type of
architectures shown below.
6. You can use any one of the optimizers and choice of Learning rate and momentum, resources:
cs231n class notes, cs231n class video.
7. For all the model's use TensorBoard and plot the Metric value and Loss with epoch. While
submitting, take a screenshot of plots and include those images in .ipynb notebook and PDF.
8. Use Categorical Cross Entropy as Loss to minimize.
```

### Model-1

Build and Train deep neural network as shown below



input_seq_total_text_data: InputLayer
Emb_Text_Data: Embedding
input_school_state: InputLayer
project_grade_category: InputLayer
input_clean_categories: InputLayer
input_clean_subcategories: InputLayer
input_teacher_prefix: InputLayer
lstm: LSTM
Emb_State_Data: Embedding
Emb_PGC_Data: Embedding
Emb_clean_categories_Data: Embedding
Emb_clean_subcategories_Data: Embedding
Emb_teacher_prefix_Data: Embedding
input_remaining_teacher_number_of_previously_posted_projects___resource_summary_contains_numerical_digits___price__quantity: InputLayer
flatten: Flatten
flatten_1: Flatten
flatten_2: Flatten
flatten_3: Flatten
flatten_4: Flatten
flatten_5: Flatten
Dense_for_rem_input: Dense
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/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.

- 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.

In [ ]:

```
# 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)
```

#### 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.

### 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.

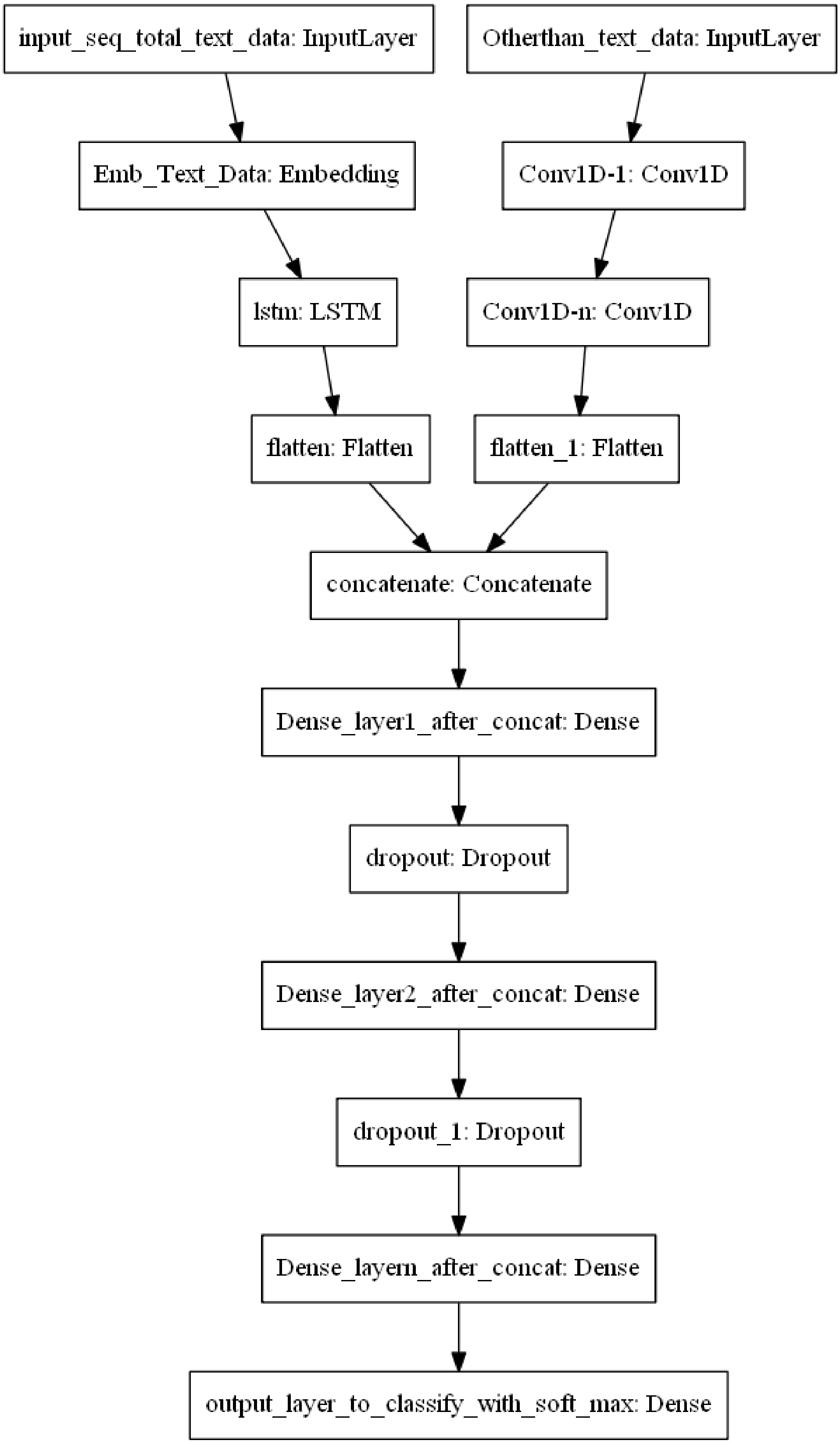
1. Train the TF-IDF on the Train data

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)

### 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

- **input_seq_total_text_data**:

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

  . 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 output.

  . 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 concatenate all these onehot
vectors

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

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

</pre>

In [38]:

```
import shutil

shutil.rmtree('logs_model3')
```

## Download datasets

In [1]:

```
!gdown --id 1pGd5tLwA30M7wkbJKdXHaae9tYVDICJ_
```

```
Downloading...
From: https://drive.google.com/uc?id=1pGd5tLwA30M7wkbJKdXHaae9tYVDICJ_
To: /content/glove_vectors
128MB [00:00, 167MB/s]
```

In [2]:

```
!gdown --id 1GpATd_pM4mcnWWIs28-s1lgqdAg2Wdv-
```

```
Downloading...
From: https://drive.google.com/uc?id=1GpATd_pM4mcnWWIs28-s1lgqdAg2Wdv-
To: /content/preprocessed_data.csv
124MB [00:01, 102MB/s]
```

## Import libraries

In [3]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from tensorflow.keras import Model
from tensorflow.keras.layers import Dense, Flatten, Input, Concatenate, Dropout, LSTM, Conv1D, BatchNorma
from tensorflow.keras import regularizers
from tensorflow.keras.layers import Embedding
from tensorflow.keras.callbacks import TensorBoard, Callback
from tensorflow.keras.utils import plot_model
import pickle
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import roc_auc_score
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
import matplotlib.pyplot as plt
import seaborn as sns
import os
import datetime
from scipy.sparse import hstack
```

## Sample Data

In [4]:

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

Out[4]:

| | school_state | teacher_prefix | project_grade_category | teacher_number_of_previously_posted_projects | project_is_approved | clean_categories | cle |
|---|---|---|---|---|---|---|---|
| **0** | ca | mrs | grades_prek_2 | 53 | 1 | math_science | |
| **1** | ut | ms | grades_3_5 | 4 | 1 | specialneeds | |
| **2** | ca | mrs | grades_prek_2 | 10 | 1 | literacy_language | |
| **3** | ga | mrs | grades_prek_2 | 2 | 1 | appliedlearning | |
| **4** | wa | mrs | grades_3_5 | 2 | 1 | literacy_language | |

In [5]:

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

Out[5]:

| | school_state | teacher_prefix | project_grade_category | teacher_number_of_previously_posted_projects | clean_categories | clean_subcategories | |
|---|---|---|---|---|---|---|---|
| **0** | ca | mrs | grades_prek_2 | 53 | math_science | appliedsciences health_lifescience | fort e us tale ki |

In [6]:

```
y[:5]
```

Out[6]:

```
array([1, 1, 1, 1, 1])
```

## Train test split

In [7]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, stratify=y)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(81936, 8)
(27312, 8)
(81936,)
(27312,)
```

In [8]:

```
print(np.unique(y_train))
print(np.unique(y_test))
```

```
[0 1]
[0 1]
```

### Text distribution analysis

In [9]:

```
# Check word length distribution in essay
text_size = []
for text in X_train['essay']:
    text_size.append(len(text.split(' ')))
print(len(text_size))
print(text_size[:2])
```

```
81936
[122, 172]
```

In [10]:

```
for i in range(0,11):
    print(90+i,'percentile value is',np.percentile(text_size,90+i))
```

```
90 percentile value is 208.0
91 percentile value is 212.0
92 percentile value is 216.0
93 percentile value is 221.0
94 percentile value is 225.0
95 percentile value is 231.0
96 percentile value is 237.0
97 percentile value is 245.0
98 percentile value is 255.0
99 percentile value is 269.0
100 percentile value is 339.0
```

In [11]:

```
for i in range(10,110,10):
    print(99+(i/100),'percentile value is',np.percentile(text_size,99+(i/100)))
```

```
99.1 percentile value is 271.0
99.2 percentile value is 273.0
99.3 percentile value is 275.0
99.4 percentile value is 278.0
99.5 percentile value is 281.0
99.6 percentile value is 284.0
99.7 percentile value is 289.0
99.8 percentile value is 294.0
99.9 percentile value is 302.0
100.0 percentile value is 339.0
```

Maxlen can be taken as 350.

### Tokenization text input 'Essay' for pretrained embedding

In [12]:

```
def tokonize(x_train, x_test, maxlen):
    '''train tokonizer with train data and create padded sequences for train and test texts'''
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(x_train)
    x_train_seq = tokenizer.texts_to_sequences(x_train)
    x_train_pad_seq = pad_sequences(x_train_seq, padding='post', maxlen=maxlen)

    x_test_seq = tokenizer.texts_to_sequences(x_test)
    x_test_pad_seq = pad_sequences(x_test_seq, padding='post', maxlen=maxlen)
    word_index = tokenizer.word_index
    vocab_size = len(word_index) + 1

    return np.array(x_train_pad_seq), np.array(x_test_pad_seq), vocab_size, word_index
```

In [13]:

```
# Tokenize and pad Essay data
X_train_essay, X_test_essay, vocab_size_essay, tok_word_index_essay = tokonize(X_train['essay'], X_test['
print(X_train_essay.shape)
print(X_test_essay.shape)
print(vocab_size_essay)
print(len(tok_word_index_essay))
```

```
(81936, 350)
(27312, 350)
50362
50361
```

### Create embedding matrix for word embedding of Essay input text

In [14]:

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

In [15]:

```
def build_embedding_matrix(vocab_size, tok_word_index):
    '''It uses the glove w2v and input tokenizer word index to create embedding matrix in word level'''
    embedding_matrix = np.zeros((vocab_size, 300))
    for word, i in tok_word_index.items():
        embedding_vector = model.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector

    return embedding_matrix
```

In [16]:

```
embedding_matrix = build_embedding_matrix(vocab_size_essay, tok_word_index_essay)
print(embedding_matrix.shape)
```

```
(50362, 300)
```

### Encoding for categorical values

In [ ]:

```
def tokenize_categorical(x_train, x_test):
  tokenizer = Tokenizer(filters='!"#$%&()*+,-./:;<=>?@[\\]^`{|}~\t\n',)
  tokenizer.fit_on_texts(x_train)
  x_train_seq = tokenizer.texts_to_sequences(x_train)
  x_train_pad_seq = pad_sequences(x_train_seq, padding='post', truncating='post')
  x_test_seq = tokenizer.texts_to_sequences(x_test)
  x_test_pad_seq = pad_sequences(x_test_seq, padding='post', truncating='post')

  vocab = len(tokenizer.word_index)+1
  length = len(x_train_pad_seq[0])
  return np.array(x_train_pad_seq), np.array(x_test_pad_seq), vocab, length
```

In [ ]:

```
# Tokenize and pad school_state data
X_train_school_state, X_test_school_state, vocab_size_school_state, length_school_state = \
tokenize_categorical(X_train['school_state'], X_test['school_state'])

print(X_train_school_state.shape)
print(X_test_school_state.shape)
print(vocab_size_school_state)
print(length_school_state)
```

```
(81936, 1)
(27312, 1)
52
1
```

In [ ]:

```
# Tokenize and pad teacher_prefix data
X_train_teacher_prefix, X_test_teacher_prefix, vocab_size_teacher_prefix, length_teacher_prefix = \
tokenize_categorical(X_train['teacher_prefix'], X_test['teacher_prefix'])

print(X_train_teacher_prefix.shape)
print(X_test_teacher_prefix.shape)
print(vocab_size_teacher_prefix)
print(length_teacher_prefix)
```

```
(81936, 1)
(27312, 1)
6
1
```

In [ ]:

```
# Tokenize and pad project_grade_category data
X_train_project_grade_category, X_test_project_grade_category, vocab_size_project_grade_category, length_
tokenize_categorical(X_train['project_grade_category'], X_test['project_grade_category'])

print(X_train_project_grade_category.shape)
print(X_test_project_grade_category.shape)
print(vocab_size_project_grade_category)
print(length_project_grade_category)
```

```
(81936, 1)
(27312, 1)
5
1
```

In [ ]:

```
# Tokenize and pad clean_categories data
X_train_clean_categories, X_test_clean_categories, vocab_size_clean_categories, length_clean_categories =
tokenize_categorical(X_train['clean_categories'], X_test['clean_categories'])

print(X_train_clean_categories.shape)
print(X_test_clean_categories.shape)
print(vocab_size_clean_categories)
print(length_clean_categories)
```

```
(81936, 3)
(27312, 3)
10
3
```

In [ ]:

```
# Tokenize and pad clean_subcategories data
X_train_clean_subcategories, X_test_clean_subcategories, vocab_size_clean_subcategories, length_clean_sub
tokenize_categorical(X_train['clean_subcategories'], X_test['clean_subcategories'])

print(X_train_clean_subcategories.shape)
print(X_test_clean_subcategories.shape)
print(vocab_size_clean_subcategories)
print(length_clean_subcategories)
```

```
(81936, 3)
(27312, 3)
31
3
```

In [ ]:

```
# Concat rest of numerical features
X_train_num = X_train[['teacher_number_of_previously_posted_projects', 'price']]
X_test_num = X_test[['teacher_number_of_previously_posted_projects', 'price']]
print(X_train_num.shape)
print(X_test_num.shape)
```

```
(81936, 2)
(27312, 2)
```

In [ ]:

```
# Normalize numeric inputs
normalizer = MinMaxScaler()
normalizer.fit(X_train_num)
X_train_num = normalizer.transform(X_train_num)
X_test_num = normalizer.transform(X_test_num)

print(X_train_num.shape)
print(X_test_num.shape)
```

```
(81936, 2)
(27312, 2)
```

In [ ]:

```
print(X_train_num[:2])
```

```
[[0.         0.06933351]
 [0.05986696 0.01549157]]
```

### Custom Callback

In [34]:

```
def auc(y_true, y_pred):
    return tf.py_function(roc_auc_score, (y_true, y_pred), tf.double)
```

In [35]:

```
class My_callback(Callback):
    '''Custom callback to save best model, stop training if accuracy is not improving and compute F1-sco
    def __init__(self, no, epochs, validation_data):
        self.no = no
        self.epochs = epochs
        self.x_val = validation_data[0]
        self.y_val = validation_data[1]

    def save_best(self):
        # Saving the best model based on high val_accuracy
        best_val_auc = max(self.history['model'].keys())
        print("Saving best weights before terminating having val_auc {}".format(best_val_auc))
        best_model = self.history['model'].get(best_val_auc)
        filepath="model-" + str(self.no) + "weights-" + str(best_val_auc) + ".hdf5"
        best_model.save(filepath)

    def on_train_begin(self, logs={}):
        # On begin of training, history dict is created with keys [model, loss, accuracy, val_loss, val_
        self.history={'model': dict(), 'loss': [], 'accuracy': [], 'val_loss': [], 'val_accuracy': [], 'v

    def on_epoch_end(self, epoch, logs={}):
        # In each epoch end, update loss, accuracy and F1-score
        loss = logs.get('loss')
        if (loss is not None):
            if (np.isnan(loss) or np.isinf(loss)):
                print('Invalid loss. Terminating at epoch {}'.format(epoch))
            else:
                if self.no == 3:
                  if logs.get('val_auc', -1) != -1:
                    self.history['val_auc'].append(logs.get('val_auc'))
                    self.history['model'][logs.get('val_auc')] = self.model
                else:
                  y_value = self.y_val
                  y_predict = self.model.predict(self.x_val)
                  val_auc = roc_auc_score(y_value, y_predict)
                  if val_auc is None:
                      val_auc = 0

                  self.history['val_auc'].append(val_auc)
                  self.history['model'][val_auc] = self.model
                  print('- val_auc: {}'.format(round(self.history['val_auc'][-1],4)))

                self.history['loss'].append(logs.get('loss'))
                self.history['accuracy'].append(logs.get('accuracy'))
                if logs.get('val_loss', -1) != -1:
                    self.history['val_loss'].append(logs.get('val_loss'))
                if logs.get('val_accuracy', -1) != -1:
                    self.history['val_accuracy'].append(logs.get('val_accuracy'))

                if len(self.history['val_auc']) >= 3:
                        if ((self.history['val_auc'][-1] < self.history['val_auc'][-3]) and
                          (self.history['val_auc'][-1] < self.history['val_auc'][-2])):
                                print("Validation auc not improving and terminated at epoch {}".format(ep
                                self.save_best()
                                self.model.stop_training = True
                loss = logs.get('loss')

                if logs.get('val_loss', -1) != -1:
                    val_loss = logs.get('val_loss')

                if val_loss is not None and val_loss - loss > 0.25:
                    print("Val-loss is much more than train loss and terminated at epoch {}".format(epoch
                    self.save_best()
                    self.model.stop_training = True

                if epoch == self.epochs-1:
                    self.save_best()
```

## Model 1

In [ ]:

```
# For essay
input_essay = Input(shape=(350), name='Input_essay')
emb_essay = Embedding(vocab_size_essay, 300, weights=[embedding_matrix],
                      input_length=350, trainable=False, name='Emdedding_essay')(input_essay)
lstm_essay = LSTM(units=512, name='LSTM_essay')(emb_essay)
flatten_essay = Flatten(name='Flatten_essay')(lstm_essay)

# For school_state
input_ss = Input(shape=(length_school_state), name='input_school_state')
emb_ss = Embedding(vocab_size_school_state, 2, input_length=length_school_state, name='Embedding_school_s
flatten_ss = Flatten(name='Flatten_school_state')(emb_ss)

# For project_grade_category
input_pgc = Input(shape=(length_project_grade_category), name='input_project_grade_category')
emb_pgc = Embedding(vocab_size_project_grade_category, 2,
                    input_length=length_project_grade_category, name='Embedding_project_grade_category')
```

```
flatten_pgc = Flatten(name='Flatten_project_grade_category')(emb_pgc)

# For clean_categories
input_cc = Input(shape=(length_clean_categories), name='input_clean_categories')
emb_cc = Embedding(vocab_size_clean_categories, 2,
                    input_length=length_clean_categories, name='Embedding_clean_categories')(input_cc)
flatten_cc = Flatten(name='Flatten_clean_categories')(emb_cc)

# For clean_subcategories
input_csc = Input(shape=(length_clean_subcategories), name='input_clean_subcategories')
emb_csc = Embedding(vocab_size_clean_subcategories, 2,
                    input_length=length_clean_subcategories, name='Embedding_clean_subcategories')(input_
flatten_csc = Flatten(name='Flatten_clean_subcategories')(emb_csc)

# For teacher_prefix
input_tp = Input(shape=(length_teacher_prefix), name='input_teacher_prefix')
emb_tp = Embedding(vocab_size_teacher_prefix, 2,
                    input_length=length_teacher_prefix, name='Embedding_teacher_prefix')(input_tp)
flatten_tp = Flatten(name='Flatten_teacher_prefix')(emb_tp)

# For numeric data
input_num = Input(shape=(X_train_num.shape[1]), name='input_numeric')
dense_num = Dense(64, activation='relu', name='Dense_numeric', kernel_initializer='he_normal')(input_num)

# Concat all the previous results
con = Concatenate(axis=1)([flatten_essay, flatten_ss, flatten_pgc, flatten_cc, flatten_csc, flatten_tp, d

dense1 = Dense(128, activation='relu', kernel_initializer='he_normal', name='Dense1')(con)
dropout1 = Dropout(0.5, name='Dropout1')(dense1)
dense2 = Dense(64, activation='relu', kernel_initializer='he_normal', name='Dense2')(dropout1)
dropout2 = Dropout(0.5, name='Dropout2')(dense2)
dense3 = Dense(32, activation='relu', kernel_initializer='he_normal', name='Dense3')(dropout2)
output = Dense(1, activation='sigmoid', name='Output')(dense3)


model1 = Model([input_essay, input_ss, input_pgc, input_cc, input_csc, input_tp, input_num], output)
model1.summary()
```

```
Model: "model_10"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
Input_essay (InputLayer)        [(None, 350)]        0
__________________________________________________________________________________________________
Emdedding_essay (Embedding)     (None, 350, 300)     15081300    Input_essay[0][0]
__________________________________________________________________________________________________
input_school_state (InputLayer) [(None, 1)]          0
__________________________________________________________________________________________________
input_project_grade_category (I [(None, 1)]          0
__________________________________________________________________________________________________
input_clean_categories (InputLa [(None, 3)]          0
__________________________________________________________________________________________________
input_clean_subcategories (Inpu [(None, 3)]          0
__________________________________________________________________________________________________
input_teacher_prefix (InputLaye [(None, 1)]          0
__________________________________________________________________________________________________
LSTM_essay (LSTM)               (None, 512)          1665024     Emdedding_essay[0][0]
__________________________________________________________________________________________________
Embedding_school_state (Embeddi (None, 1, 2)         104         input_school_state[0][0]
__________________________________________________________________________________________________
Embedding_project_grade_categor (None, 1, 2)         10          input_project_grade_category[0][0
__________________________________________________________________________________________________
Embedding_clean_categories (Emb (None, 3, 2)         20          input_clean_categories[0][0]
__________________________________________________________________________________________________
Embedding_clean_subcategories ( (None, 3, 2)         62          input_clean_subcategories[0][0]
__________________________________________________________________________________________________
Embedding_teacher_prefix (Embed (None, 1, 2)         12          input_teacher_prefix[0][0]
__________________________________________________________________________________________________
input_numeric (InputLayer)      [(None, 2)]          0
__________________________________________________________________________________________________
Flatten_essay (Flatten)         (None, 512)          0           LSTM_essay[0][0]
__________________________________________________________________________________________________
Flatten_school_state (Flatten)  (None, 2)            0           Embedding_school_state[0][0]
__________________________________________________________________________________________________
Flatten_project_grade_category  (None, 2)            0           Embedding_project_grade_category[
__________________________________________________________________________________________________
Flatten_clean_categories (Flatt (None, 6)            0           Embedding_clean_categories[0][0]
__________________________________________________________________________________________________
Flatten_clean_subcategories (Fl (None, 6)            0           Embedding_clean_subcategories[0][
__________________________________________________________________________________________________
Flatten_teacher_prefix (Flatten (None, 2)            0           Embedding_teacher_prefix[0][0]
__________________________________________________________________________________________________
Dense_numeric (Dense)           (None, 64)           192         input_numeric[0][0]
__________________________________________________________________________________________________
concatenate_10 (Concatenate)    (None, 594)          0           Flatten_essay[0][0]
                                                                 Flatten_school_state[0][0]
                                                                 Flatten_project_grade_category[0]
                                                                 Flatten_clean_categories[0][0]
                                                                 Flatten_clean_subcategories[0][0]
                                                                 Flatten_teacher_prefix[0][0]
                                                                 Dense_numeric[0][0]
__________________________________________________________________________________________________
Dense1 (Dense)                  (None, 128)          76160       concatenate_10[0][0]
__________________________________________________________________________________________________
Dropout1 (Dropout)              (None, 128)          0           Dense1[0][0]
__________________________________________________________________________________________________
Dense2 (Dense)                  (None, 64)           8256        Dropout1[0][0]
__________________________________________________________________________________________________
Dropout2 (Dropout)              (None, 64)           0           Dense2[0][0]
__________________________________________________________________________________________________
Dense3 (Dense)                  (None, 32)           2080        Dropout2[0][0]
__________________________________________________________________________________________________
Output (Dense)                  (None, 1)            33          Dense3[0][0]
==================================================================================================
Total params: 16,833,253
Trainable params: 1,751,953
Non-trainable params: 15,081,300
__________________________________________________________________________________________________
```

In [42]:

```
%load_ext tensorboard
```

**Fit model 1 with tesnboard, custom call back using adam optimizer with learning rate 0.001 and accuracy metric**

In [ ]:

```
logdir1 = os.path.join("logs_model1", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
```

```
tensorboard_callback1 = TensorBoard(log_dir=logdir1,histogram_freq=1, write_graph=True)
myCallback1 = My_callback(1, 20, ([X_test_essay, X_test_school_state, X_test_project_grade_category, X_te
                              X_test_clean_subcategories,X_test_teacher_prefix, X_test_num],y_test))

opti = tf.keras.optimizers.RMSprop(learning_rate = 0.0001)
model1.compile(optimizer=opti,loss='binary_crossentropy',metrics=['accuracy'])
model1.fit([X_train_essay, X_train_school_state, X_train_project_grade_category, X_train_clean_categories
             X_train_teacher_prefix, X_train_num],
            y_train,epochs=20,
            validation_data=([X_test_essay, X_test_school_state, X_test_project_grade_category, X_test_cle
                              X_test_clean_subcategories,X_test_teacher_prefix, X_test_num],
                             y_test),
            batch_size=32, callbacks=[tensorboard_callback1, myCallback1])
```

```
Epoch 1/20
   6/2561 [..............................] - ETA: 5:52 - loss: 0.6863 - accuracy: 
0.6008WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch
time: 0.0372s vs `on_train_batch_end` time: 0.0883s). Check your callbacks.
2561/2561 [==============================] - 191s 73ms/step - loss: 0.4614 - accuracy: 0.8435 -
val_loss: 0.4228 - val_accuracy: 0.8486
- val_auc: 0.6112
Epoch 2/20
2561/2561 [==============================] - 186s 73ms/step - loss: 0.4289 - accuracy: 0.8471 -
val_loss: 0.4178 - val_accuracy: 0.8486
- val_auc: 0.617
Epoch 3/20
2561/2561 [==============================] - 187s 73ms/step - loss: 0.4219 - accuracy: 0.8483 -
val_loss: 0.4160 - val_accuracy: 0.8486
- val_auc: 0.618
Epoch 4/20
2561/2561 [==============================] - 187s 73ms/step - loss: 0.4161 - accuracy: 0.8507 -
val_loss: 0.4173 - val_accuracy: 0.8486
- val_auc: 0.619
Epoch 5/20
2561/2561 [==============================] - 186s 73ms/step - loss: 0.4184 - accuracy: 0.8487 -
val_loss: 0.4174 - val_accuracy: 0.8486
- val_auc: 0.6227
Epoch 6/20
2561/2561 [==============================] - 186s 73ms/step - loss: 0.4219 - accuracy: 0.8478 -
val_loss: 0.4157 - val_accuracy: 0.8486
- val_auc: 0.6462
Epoch 7/20
2561/2561 [==============================] - 185s 72ms/step - loss: 0.4082 - accuracy: 0.8495 -
val_loss: 0.3975 - val_accuracy: 0.8486
- val_auc: 0.7019
Epoch 8/20
2561/2561 [==============================] - 185s 72ms/step - loss: 0.4007 - accuracy: 0.8481 -
val_loss: 0.3883 - val_accuracy: 0.8486
- val_auc: 0.735
Epoch 9/20
2561/2561 [==============================] - 187s 73ms/step - loss: 0.3893 - accuracy: 0.8485 -
val_loss: 0.3838 - val_accuracy: 0.8486
- val_auc: 0.7483
Epoch 10/20
2561/2561 [==============================] - 187s 73ms/step - loss: 0.3852 - accuracy: 0.8461 -
val_loss: 0.3977 - val_accuracy: 0.8486
- val_auc: 0.7467
Epoch 11/20
2561/2561 [==============================] - 187s 73ms/step - loss: 0.3777 - accuracy: 0.8501 -
val_loss: 0.4217 - val_accuracy: 0.8486
- val_auc: 0.7511
Epoch 12/20
2561/2561 [==============================] - 187s 73ms/step - loss: 0.3890 - accuracy: 0.8469 -
val_loss: 0.3922 - val_accuracy: 0.8486
- val_auc: 0.7597
Epoch 13/20
2561/2561 [==============================] - 187s 73ms/step - loss: 0.3755 - accuracy: 0.8490 -
val_loss: 0.4167 - val_accuracy: 0.8486
- val_auc: 0.735
Validation auc not improving and terminated at epoch 13
Saving best weights before terminating having val_auc 0.7597097846918943
```

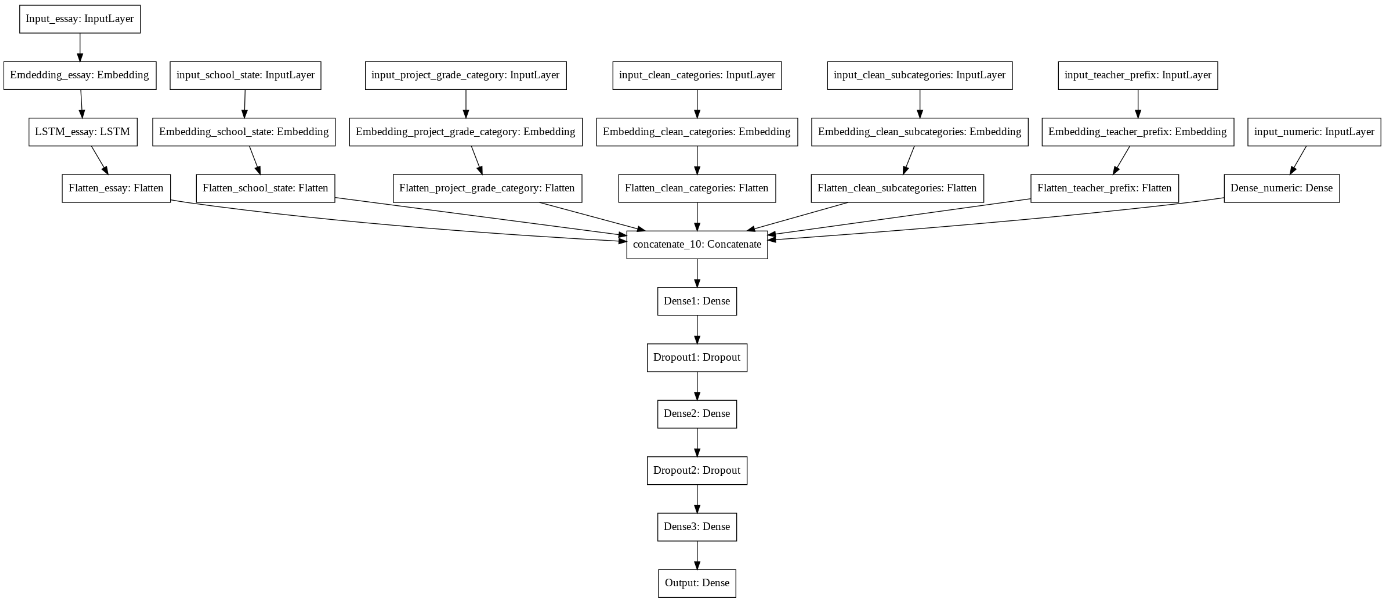
Out[ ]:

```
<tensorflow.python.keras.callbacks.History at 0x7f8a904a2f60>
```

In [ ]:

```
plot_model(model1, to_file='model1.png')
```

Out[ ]:

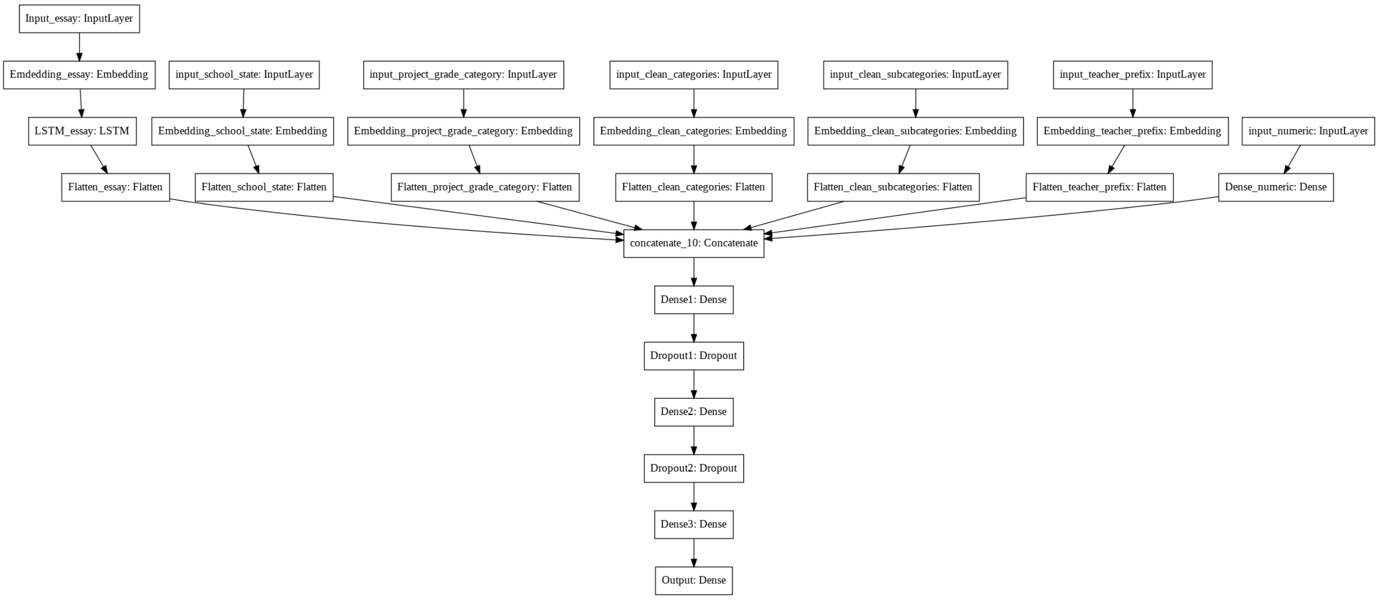

Input_essay: InputLayer
Emdedding_essay: Embedding
input_school_state: InputLayer
input_project_grade_category: InputLayer
input_clean_categories: InputLayer
input_clean_subcategories: InputLayer
input_teacher_prefix: InputLayer
LSTM_essay: LSTM
Embedding_school_state: Embedding
Embedding_project_grade_category: Embedding
Embedding_clean_categories: Embedding
Embedding_clean_subcategories: Embedding
Embedding_teacher_prefix: Embedding
input_numeric: InputLayer
Flatten_essay: Flatten
Flatten_school_state: Flatten
Flatten_project_grade_category: Flatten
Flatten_clean_categories: Flatten
Flatten_clean_subcategories: Flatten
Flatten_teacher_prefix: Flatten
Dense_numeric: Dense
concatenate_10: Concatenate
Dense1: Dense
Dropout1: Dropout
Dense2: Dense
Dropout2: Dropout
Dense3: Dense
Output: Dense

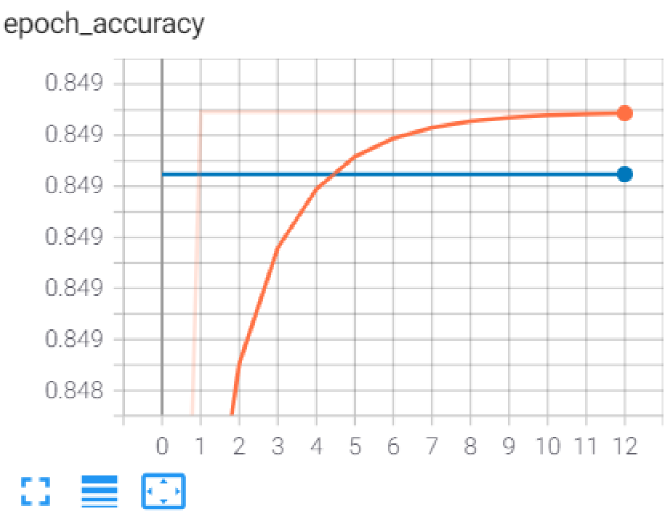
In [ ]:

```
%tensorboard --logdir logs_model1
```

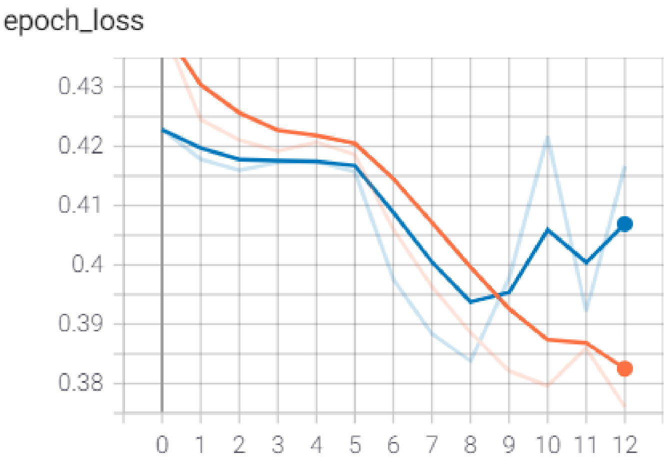
## Model1:

AUC: 0.7597


Input_essay: InputLayer
Emdedding_essay: Embedding
input_school_state: InputLayer
input_project_grade_category: InputLayer
input_clean_categories: InputLayer
input_clean_subcategories: InputLayer
input_teacher_prefix: InputLayer
LSTM_essay: LSTM
Embedding_school_state: Embedding
Embedding_project_grade_category: Embedding
Embedding_clean_categories: Embedding
Embedding_clean_subcategories: Embedding
Embedding_teacher_prefix: Embedding
input_numeric: InputLayer
Flatten_essay: Flatten
Flatten_school_state: Flatten
Flatten_project_grade_category: Flatten
Flatten_clean_categories: Flatten
Flatten_clean_subcategories: Flatten
Flatten_teacher_prefix: Flatten
Dense_numeric: Dense
concatenate_10: Concatenate
Dense1: Dense
Dropout1: Dropout
Dense2: Dense
Dropout2: Dropout
Dense3: Dense
Output: Dense



epoch_loss



## Model 2

### Filter words from essay having very high and very low IDF value for model 2

In [ ]:

```
def getIDFFromTrainText(text):
  tfidf = TfidfVectorizer()
  tfidf.fit(text)
  idfs = dict(zip(tfidf.get_feature_names(), list(tfidf.idf_)))
  return idfs
```

In [ ]:

```
word_idfs = getIDFFromTrainText(X_train['essay'])
print(len(word_idfs))
```

```
50380
```

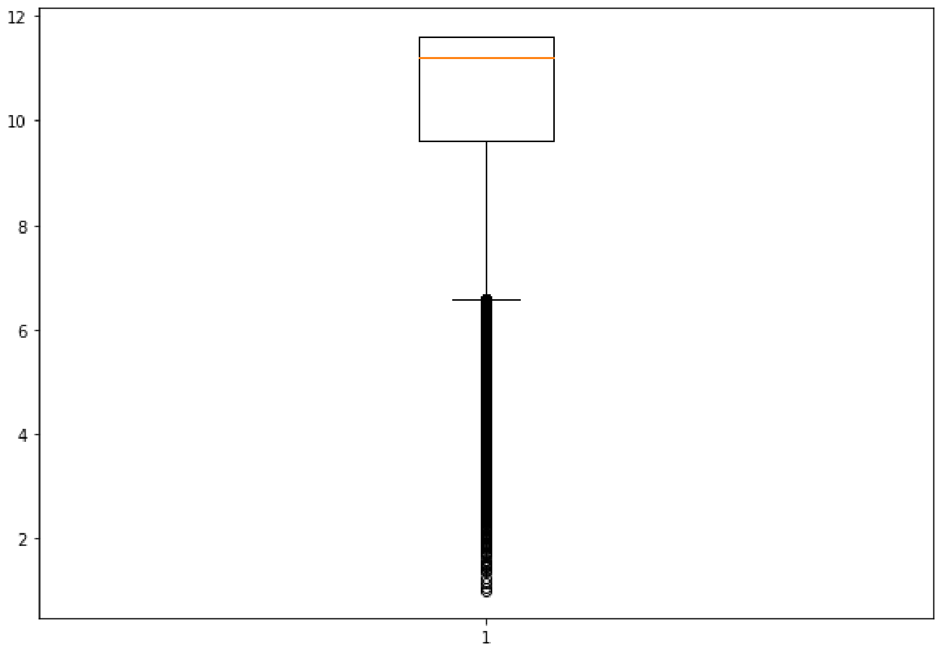
In [ ]:

```
idfs = [i for i in word_idfs.values()]
print(idfs[:5])
```

```
[7.189741958872126, 5.93019930339138, 11.62055875771544, 11.215093649607276, 11.62055875771544]
```

In [ ]:

```
fig = plt.figure(figsize =(10, 7))
plt.boxplot(idfs)
plt.show()
```



In [ ]:

```
for i in range(10,100,10):
    print(i,'percentile value is',np.percentile(idfs,i))
```

```
10 percentile value is 7.560115747169021
20 percentile value is 9.055609400253903
30 percentile value is 10.01112084528134
40 percentile value is 10.704268025841285
50 percentile value is 11.215093649607276
60 percentile value is 11.215093649607276
70 percentile value is 11.62055875771544
80 percentile value is 11.62055875771544
90 percentile value is 11.62055875771544
```

In [ ]:

```
for i in range(1,11):
    print(i,'percentile value is',np.percentile(idfs,i))
```

```
1 percentile value is 4.112784367021209
2 percentile value is 4.9707589372955585
3 percentile value is 5.477844296716331
4 percentile value is 5.887217480817695
5 percentile value is 6.275835018353248
6 percentile value is 6.609923463619184
7 percentile value is 6.8756266293521895
8 percentile value is 7.12632013243463
9 percentile value is 7.357878880674124
10 percentile value is 7.560115747169021
```

In [ ]:

```
for i in range(0,11):
    print(90+i,'percentile value is',np.percentile(idfs,90+i))
```

```
90 percentile value is 11.62055875771544
91 percentile value is 11.62055875771544
92 percentile value is 11.62055875771544
93 percentile value is 11.62055875771544
94 percentile value is 11.62055875771544
95 percentile value is 11.62055875771544
96 percentile value is 11.62055875771544
97 percentile value is 11.62055875771544
98 percentile value is 11.62055875771544
99 percentile value is 11.62055875771544
100 percentile value is 11.62055875771544
```

We can take low range as 2 and high range 11 of idf values to consider words within that range.

In [ ]:

```
def filterWords(input_text):
  updated_rec = []
  for word in input_text.split(' '):
    idf = word_idfs.get(word)
    if (idf is not None and idf > 2 and idf < 11):
        updated_rec.append(word)
  new_rec = ' '.join(updated_rec)
  return new_rec
```

In [ ]:

```
# Update the essay texts as per model 2
new_train_text = []
new_test_text = []
for text in X_train['essay']:
  new_train_text.append(filterWords(text))
for text in X_test['essay']:
  new_test_text.append(filterWords(text))

m2_X_train_essay = np.array(new_train_text)
m2_X_test_essay = np.array(new_test_text)

print(m2_X_train_essay.shape)
print(m2_X_test_essay.shape)
```

```
(81936,)
(27312,)
```

In [ ]:

```
print(m2_X_train_essay[:2])
```

```
['community families different racial ethnic socio economic backgrounds together around common cause eng
aging rich educational experience works constructivist model education taking real life experience
background knowledge applying throughout educational settings classrooms strive dynamic reach individual
child every lesson role support speech language social pragmatic difficulties benefit specialized
services individual small group basis intervention delivered using collaborative model general education
teacher appropriate access curriculum content level pace given special supports needed creative flexible
community looks child guidance effectively simply facilitate lead child toward benefit social skills
books capes reading rug supplies directly indirectly speech language intervention social skills
interventions key making list julia cook books well superhero capes essential whole small group sessions
small number office supplies aide professional documenting progress adequately ipad keyboard velcro put
good visual schedules rug serve quiet space take book read short time speech language room with natural
light quiet place sit hope inspire books'
 'located south bronx poorest congressional district funding extracurricular activities afterthought quit
e time compete baseball team every year yearning safe fun productive way spend free time as heart south
bronx space nearly nonexistent parks around safe places us practice reality never dampened players
spirits each one comes every three ready practice options basketball gym share two schools horseshoe
shaped concrete recess area behind building although times played actual baseball diamond games team pla
yed fierce tenacity never making excuses baseball gloves fundamental part sport student players lack
tried corral donations staff truth people baseball gloves laying around if worn missing laces when
players gloves often fit properly these gloves used develop players skills despite disadvantages
physical environment resources additionally since place gym softer practice balls mess hardwood floors p
ut maximum effort equipment necessary address basic component game project aims address provide
productive experience as immigrants children immigrants dominican republican baseball provides
opportunity see connections american dominican culture teachers seen undeniable positive affect baseball
academic performance players kids play']
```

### Text distribution analysis for model 2

In [ ]:

```
# Check word length distribution in essay
text_size = []
for text in m2_X_train_essay:
    text_size.append(len(text.split(' ')))
print(len(text_size))
print(text_size[:2])
```

```
81936
[149, 166]
```

In [ ]:

```
for i in range(0,11):
    print(90+i,'percentile value is',np.percentile(text_size,90+i))
```

```
90 percentile value is 163.0
91 percentile value is 166.0
92 percentile value is 169.0
93 percentile value is 173.0
94 percentile value is 177.0
95 percentile value is 182.0
96 percentile value is 187.0
97 percentile value is 194.0
98 percentile value is 202.0
99 percentile value is 216.0
100 percentile value is 294.0
```

In [ ]:

```
for i in range(10,110,10):
    print(99+(i/100),'percentile value is',np.percentile(text_size,99+(i/100)))
```

```
99.1 percentile value is 218.0
99.2 percentile value is 220.0
99.3 percentile value is 222.0
99.4 percentile value is 225.0
99.5 percentile value is 228.0
99.6 percentile value is 231.0
99.7 percentile value is 235.0
99.8 percentile value is 240.0
99.9 percentile value is 249.0
100.0 percentile value is 294.0
```

Maxlen can be taken as 250.

In [ ]:

```
# Tokenize and pad Essay data for model 2
m2_X_train_essay, m2_X_test_essay, vocab_size_essay_m2, tok_word_index_essay_m2 = tokonize(m2_X_train_ess
print(m2_X_train_essay.shape)
print(m2_X_test_essay.shape)
print(vocab_size_essay_m2)
print(len(tok_word_index_essay_m2))
```

```
(81936, 250)
(27312, 250)
24961
24960
```

In [ ]:

```
# Embedding matrix for model2
embedding_matrix_m2 = build_embedding_matrix(vocab_size_essay_m2, tok_word_index_essay_m2)
print(embedding_matrix_m2.shape)
```

```
(24961, 300)
```

In [ ]:

```
# For essay
input_essay = Input(shape=(250), name='Input_essay')
emb_essay = Embedding(vocab_size_essay_m2, 300, weights=[embedding_matrix_m2],
                      input_length=250, trainable=False, name='Emdedding_essay')(input_essay)
lstm_essay = LSTM(units=512, name='LSTM_essay')(emb_essay)
flatten_essay = Flatten(name='Flatten_essay')(lstm_essay)

# For school_state
input_ss = Input(shape=(length_school_state), name='input_school_state')
emb_ss = Embedding(vocab_size_school_state, 2, input_length=length_school_state, name='Embedding_school_s
flatten_ss = Flatten(name='Flatten_school_state')(emb_ss)

# For project_grade_category
input_pgc = Input(shape=(length_project_grade_category), name='input_project_grade_category')
emb_pgc = Embedding(vocab_size_project_grade_category, 2,
                    input_length=length_project_grade_category, name='Embedding_project_grade_category')
flatten_pgc = Flatten(name='Flatten_project_grade_category')(emb_pgc)

# For clean_categories
input_cc = Input(shape=(length_clean_categories), name='input_clean_categories')
emb_cc = Embedding(vocab_size_clean_categories, 2,
                    input_length=length_clean_categories, name='Embedding_clean_categories')(input_cc)
flatten_cc = Flatten(name='Flatten_clean_categories')(emb_cc)

# For clean_subcategories
input_csc = Input(shape=(length_clean_subcategories), name='input_clean_subcategories')
emb_csc = Embedding(vocab_size_clean_subcategories, 2,
                    input_length=length_clean_subcategories, name='Embedding_clean_subcategories')(input_
flatten_csc = Flatten(name='Flatten_clean_subcategories')(emb_csc)

# For teacher_prefix
input_tp = Input(shape=(length_teacher_prefix), name='input_teacher_prefix')
emb_tp = Embedding(vocab_size_teacher_prefix, 2,
                    input_length=length_teacher_prefix, name='Embedding_teacher_prefix')(input_tp)
flatten_tp = Flatten(name='Flatten_teacher_prefix')(emb_tp)

# For numeric data
input_num = Input(shape=(X_train_num.shape[1]), name='input_numeric')
dense_num = Dense(64, activation='relu', name='Dense_numeric', kernel_initializer='he_normal')(input_num)

# Concat all the previous results
con = Concatenate(axis=1)([flatten_essay, flatten_ss, flatten_pgc, flatten_cc, flatten_csc, flatten_tp, de

dense1 = Dense(128, activation='relu', kernel_initializer='he_normal', name='Dense1')(con)
```

```
dropout1 = Dropout(0.5, name='Dropout1')(dense1)
dense2 = Dense(64, activation='relu', kernel_initializer='he_normal', name='Dense2')(dropout1)
dropout2 = Dropout(0.5, name='Dropout2')(dense2)
dense3 = Dense(32, activation='relu', kernel_initializer='he_normal', name='Dense3')(dropout2)
output = Dense(1, activation='sigmoid', name='Output')(dense3)


model2 = Model([input_essay, input_ss, input_pgc, input_cc, input_csc, input_tp, input_num], output)
model2.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to                     
==================================================================================================
Input_essay (InputLayer)        [(None, 250)]        0                                            
__________________________________________________________________________________________________
Emdedding_essay (Embedding)     (None, 250, 300)     7488300     Input_essay[0][0]                
__________________________________________________________________________________________________
input_school_state (InputLayer) [(None, 1)]          0                                            
__________________________________________________________________________________________________
input_project_grade_category (I [(None, 1)]          0                                            
__________________________________________________________________________________________________
input_clean_categories (InputLa [(None, 3)]          0                                            
__________________________________________________________________________________________________
input_clean_subcategories (Inpu [(None, 3)]          0                                            
__________________________________________________________________________________________________
input_teacher_prefix (InputLaye [(None, 1)]          0                                            
__________________________________________________________________________________________________
LSTM_essay (LSTM)               (None, 512)          1665024     Emdedding_essay[0][0]            
__________________________________________________________________________________________________
Embedding_school_state (Embeddi (None, 1, 2)         104         input_school_state[0][0]         
__________________________________________________________________________________________________
Embedding_project_grade_categor (None, 1, 2)         10          input_project_grade_category[0][0
__________________________________________________________________________________________________
Embedding_clean_categories (Emb (None, 3, 2)         20          input_clean_categories[0][0]     
__________________________________________________________________________________________________
Embedding_clean_subcategories ( (None, 3, 2)         62          input_clean_subcategories[0][0]  
__________________________________________________________________________________________________
Embedding_teacher_prefix (Embed (None, 1, 2)         12          input_teacher_prefix[0][0]       
__________________________________________________________________________________________________
input_numeric (InputLayer)      [(None, 2)]          0                                            
__________________________________________________________________________________________________
Flatten_essay (Flatten)         (None, 512)          0           LSTM_essay[0][0]                 
__________________________________________________________________________________________________
Flatten_school_state (Flatten)  (None, 2)            0           Embedding_school_state[0][0]     
__________________________________________________________________________________________________
Flatten_project_grade_category  (None, 2)            0           Embedding_project_grade_category[
__________________________________________________________________________________________________
Flatten_clean_categories (Flatt (None, 6)            0           Embedding_clean_categories[0][0] 
__________________________________________________________________________________________________
Flatten_clean_subcategories (Fl (None, 6)            0           Embedding_clean_subcategories[0][
__________________________________________________________________________________________________
Flatten_teacher_prefix (Flatten (None, 2)            0           Embedding_teacher_prefix[0][0]   
__________________________________________________________________________________________________
Dense_numeric (Dense)           (None, 64)           192         input_numeric[0][0]              
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 594)          0           Flatten_essay[0][0]              
                                                                 Flatten_school_state[0][0]       
                                                                 Flatten_project_grade_category[0]
                                                                 Flatten_clean_categories[0][0]   
                                                                 Flatten_clean_subcategories[0][0]
                                                                 Flatten_teacher_prefix[0][0]     
                                                                 Dense_numeric[0][0]              
__________________________________________________________________________________________________
Dense1 (Dense)                  (None, 128)          76160       concatenate[0][0]                
__________________________________________________________________________________________________
Dropout1 (Dropout)              (None, 128)          0           Dense1[0][0]                     
__________________________________________________________________________________________________
Dense2 (Dense)                  (None, 64)           8256        Dropout1[0][0]                   
__________________________________________________________________________________________________
Dropout2 (Dropout)              (None, 64)           0           Dense2[0][0]                     
__________________________________________________________________________________________________
Dense3 (Dense)                  (None, 32)           2080        Dropout2[0][0]                   
__________________________________________________________________________________________________
Output (Dense)                  (None, 1)            33          Dense3[0][0]                     
==================================================================================================
Total params: 9,240,253
Trainable params: 1,751,953
Non-trainable params: 7,488,300
__________________________________________________________________________________________________
```

In [ ]:

```
logdir2 = os.path.join("logs_model2", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback2 = TensorBoard(log_dir=logdir2,histogram_freq=1, write_graph=True)
myCallback2 = My_callback(2, 20, ([m2_X_test_essay, X_test_school_state, X_test_project_grade_category, X
                             X_test_clean_subcategories,X_test_teacher_prefix, X_test_num],y_test))
```

```
opti = tf.keras.optimizers.RMSprop(learning_rate = 0.0001)
model2.compile(optimizer=opti,loss='binary_crossentropy',metrics=['accuracy'])
model2.fit([m2_X_train_essay, X_train_school_state, X_train_project_grade_category, X_train_clean_categor
            X_train_teacher_prefix, X_train_num],
           y_train,epochs=20,
           validation_data=([m2_X_test_essay, X_test_school_state, X_test_project_grade_category, X_test_
                             X_test_clean_subcategories,X_test_teacher_prefix, X_test_num],
                            y_test),
           batch_size=32, callbacks=[tensorboard_callback2, myCallback2])
```

```
Epoch 1/20
   5/2561 [..............................] - ETA: 3:53 - loss: 0.6887 - accuracy:
0.6435WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch
time: 0.0255s vs `on_train_batch_end` time: 0.0515s). Check your callbacks.
2561/2561 [==============================] - 138s 51ms/step - loss: 0.4564 - accuracy: 0.8457 -
val_loss: 0.4314 - val_accuracy: 0.8486
- val_auc: 0.608
Epoch 2/20
2561/2561 [==============================] - 130s 51ms/step - loss: 0.4244 - accuracy: 0.8493 -
val_loss: 0.4119 - val_accuracy: 0.8486
- val_auc: 0.6358
Epoch 3/20
2561/2561 [==============================] - 130s 51ms/step - loss: 0.4167 - accuracy: 0.8482 -
val_loss: 0.4063 - val_accuracy: 0.8486
- val_auc: 0.677
Epoch 4/20
2561/2561 [==============================] - 130s 51ms/step - loss: 0.4010 - accuracy: 0.8488 -
val_loss: 0.3921 - val_accuracy: 0.8486
- val_auc: 0.7219
Epoch 5/20
2561/2561 [==============================] - 129s 51ms/step - loss: 0.3927 - accuracy: 0.8485 -
val_loss: 0.3866 - val_accuracy: 0.8486
- val_auc: 0.7145
Epoch 6/20
2561/2561 [==============================] - 129s 50ms/step - loss: 0.3848 - accuracy: 0.8489 -
val_loss: 0.3993 - val_accuracy: 0.8486
- val_auc: 0.7184
Epoch 7/20
2561/2561 [==============================] - 130s 51ms/step - loss: 0.3840 - accuracy: 0.8482 -
val_loss: 0.4770 - val_accuracy: 0.8486
- val_auc: 0.7412
Epoch 8/20
2561/2561 [==============================] - 129s 50ms/step - loss: 0.3826 - accuracy: 0.8493 -
val_loss: 0.3994 - val_accuracy: 0.8486
- val_auc: 0.7498
Epoch 9/20
2561/2561 [==============================] - 128s 50ms/step - loss: 0.3769 - accuracy: 0.8482 -
val_loss: 0.3779 - val_accuracy: 0.8486
- val_auc: 0.7517
Epoch 10/20
2561/2561 [==============================] - 128s 50ms/step - loss: 0.3763 - accuracy: 0.8483 -
val_loss: 0.4006 - val_accuracy: 0.8486
- val_auc: 0.6918
Validation auc not improving and terminated at epoch 10
Saving best weights before terminating having val_auc 0.7516624061758678
```

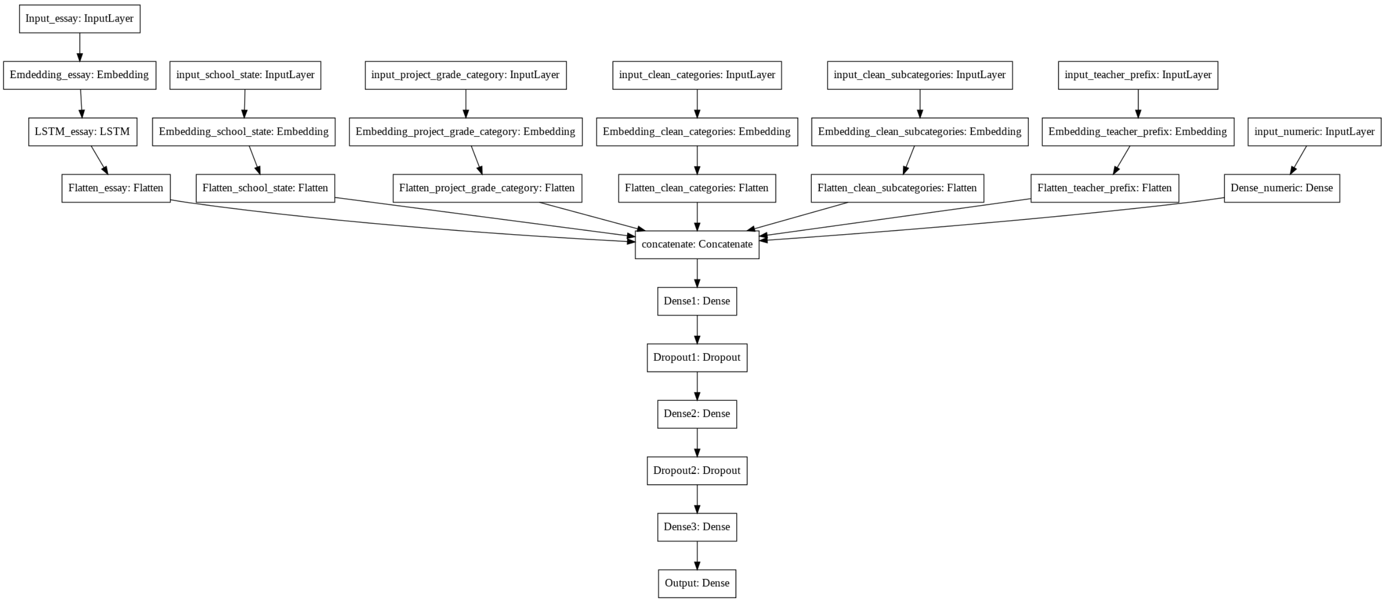
Out[ ]:

```
<tensorflow.python.keras.callbacks.History at 0x7f74536e3b00>
```

In [ ]:

```
plot_model(model2, to_file='model2.png')
```

Out[ ]:

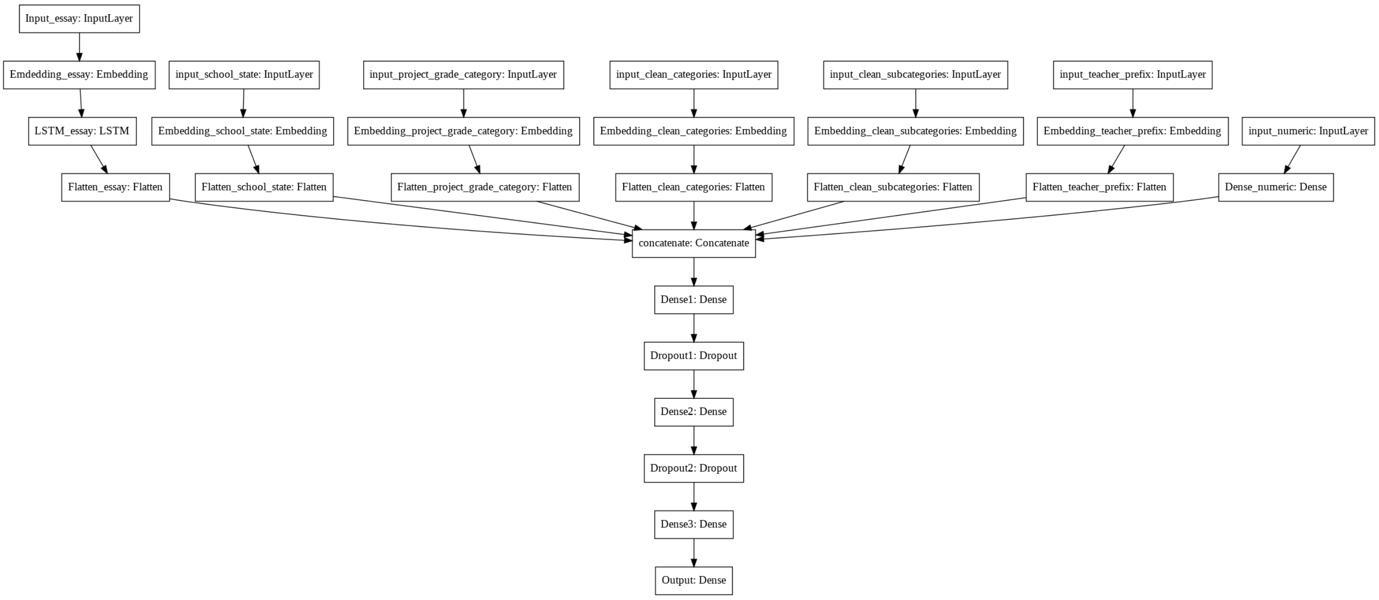

Input_essay: InputLayer
Emdedding_essay: Embedding
input_school_state: InputLayer
input_project_grade_category: InputLayer
input_clean_categories: InputLayer
input_clean_subcategories: InputLayer
input_teacher_prefix: InputLayer
LSTM_essay: LSTM
Embedding_school_state: Embedding
Embedding_project_grade_category: Embedding
Embedding_clean_categories: Embedding
Embedding_clean_subcategories: Embedding
Embedding_teacher_prefix: Embedding
input_numeric: InputLayer
Flatten_essay: Flatten
Flatten_school_state: Flatten
Flatten_project_grade_category: Flatten
Flatten_clean_categories: Flatten
Flatten_clean_subcategories: Flatten
Flatten_teacher_prefix: Flatten
Dense_numeric: Dense
concatenate: Concatenate
Dense1: Dense
Dropout1: Dropout
Dense2: Dense
Dropout2: Dropout
Dense3: Dense
Output: Dense

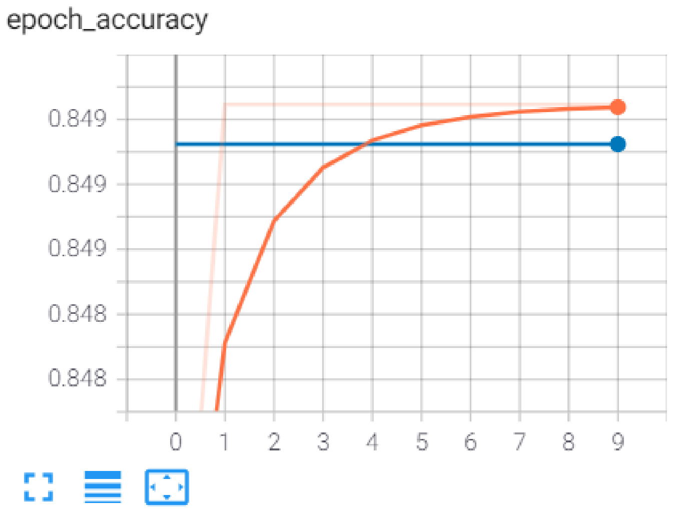
In [ ]:

```
%tensorboard --logdir logs_model2
```

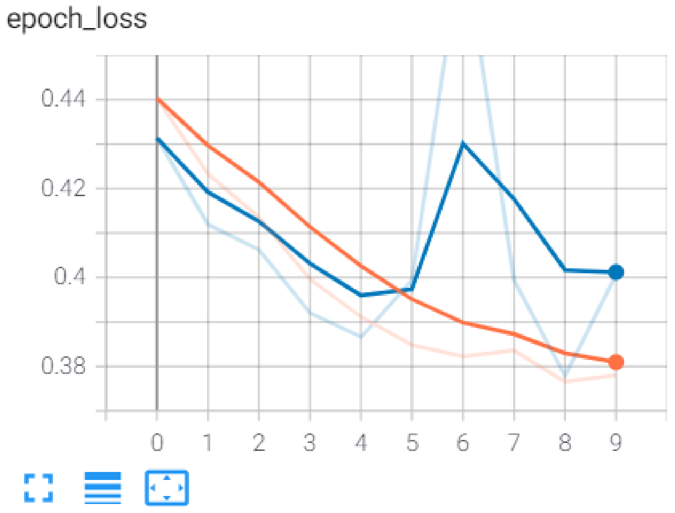
## model 2:

AUC: 0.7516


Input_essay: InputLayer
Emdedding_essay: Embedding
input_school_state: InputLayer
input_project_grade_category: InputLayer
input_clean_categories: InputLayer
input_clean_subcategories: InputLayer
input_teacher_prefix: InputLayer
LSTM_essay: LSTM
Embedding_school_state: Embedding
Embedding_project_grade_category: Embedding
Embedding_clean_categories: Embedding
Embedding_clean_subcategories: Embedding
Embedding_teacher_prefix: Embedding
input_numeric: InputLayer
Flatten_essay: Flatten
Flatten_school_state: Flatten
Flatten_project_grade_category: Flatten
Flatten_clean_categories: Flatten
Flatten_clean_subcategories: Flatten
Flatten_teacher_prefix: Flatten
Dense_numeric: Dense
concatenate: Concatenate
Dense1: Dense
Dropout1: Dropout
Dense2: Dense
Dropout2: Dropout
Dense3: Dense
Output: Dense



epoch_loss



## Model 3

### Encode categorical and numerical features

In [17]:

```
from sklearn.feature_extraction.text import CountVectorizer
```

In [18]:

```
def encodeCategoricalValues(train_cat, test_cat):
    """This function takes any categorical feature values for train and text
    and applies One Hot Encoding using CountVectorizer"""

    encoder = CountVectorizer()
    output = dict()

    # Use vectorizer to fit and transform the input categories
    Xtrain_cat = encoder.fit_transform(train_cat)
    Xtest_cat = encoder.transform(test_cat)
    return Xtrain_cat, Xtest_cat
```

In [19]:

```
# Encode school_state data
X_train_school_state, X_test_school_state = encodeCategoricalValues(X_train['school_state'], X_test['scho

print(X_train_school_state.shape)
print(X_test_school_state.shape)
```

```
(81936, 51)
(27312, 51)
```

In [20]:

```
# Encode teacher_prefix data
X_train_teacher_prefix, X_test_teacher_prefix = encodeCategoricalValues(X_train['teacher_prefix'], X_test

print(X_train_teacher_prefix.shape)
print(X_test_teacher_prefix.shape)
```

```
(81936, 5)
(27312, 5)
```

In [21]:

```
# Encode project_grade_category data
X_train_project_grade_category, X_test_project_grade_category = \
encodeCategoricalValues(X_train['project_grade_category'], X_test['project_grade_category'])

print(X_train_project_grade_category.shape)
print(X_test_project_grade_category.shape)
```

```
(81936, 4)
(27312, 4)
```

In [22]:

```
# Encode clean_categories data
X_train_clean_categories, X_test_clean_categories = encodeCategoricalValues(X_train['clean_categories'],

print(X_train_clean_categories.shape)
print(X_test_clean_categories.shape)
```

```
(81936, 9)
(27312, 9)
```

In [23]:

```
# Encode clean_subcategories data
X_train_clean_subcategories, X_test_clean_subcategories = \
encodeCategoricalValues(X_train['clean_subcategories'], X_test['clean_subcategories'])

print(X_train_clean_subcategories.shape)
print(X_test_clean_subcategories.shape)
```

```
(81936, 30)
(27312, 30)
```

In [24]:

```
# Concat rest of numerical features
X_train_num = X_train[['teacher_number_of_previously_posted_projects', 'price']]
X_test_num = X_test[['teacher_number_of_previously_posted_projects', 'price']]
print(X_train_num.shape)
print(X_test_num.shape)
```

```
(81936, 2)
(27312, 2)
```

In [25]:

```
# Normalize numeric inputs
normalizer = MinMaxScaler()
normalizer.fit(X_train_num)
X_train_num = normalizer.transform(X_train_num)
X_test_num = normalizer.transform(X_test_num)

print(X_train_num.shape)
print(X_test_num.shape)
```

```
(81936, 2)
(27312, 2)
```

In [26]:

```
X_train_school_state.shape
```

Out[26]:

```
(81936, 51)
```

In [27]:

```
 X_train_teacher_prefix.shape
```

Out[27]:

```
(81936, 5)
```

In [28]:

```
import scipy
```

In [29]:

```
X_train_rest = scipy.sparse.hstack((X_train_school_state, X_train_teacher_prefix, X_train_project_grade_c
        X_train_clean_categories, X_train_clean_subcategories, X_train_num)).todense()
X_test_rest = scipy.sparse.hstack((X_test_school_state, X_test_teacher_prefix, X_test_project_grade_categ
        X_test_clean_categories, X_test_clean_subcategories, X_test_num)).todense()
```

In [30]:

```
print(X_train_rest.shape)
print(X_test_rest.shape)

X_train_rest = X_train_rest[:,:,np.newaxis]
X_test_rest = X_test_rest[:,:,np.newaxis]
print(X_train_rest.shape)
print(X_test_rest.shape)
```

```
(81936, 101)
(27312, 101)
(81936, 101, 1)
(27312, 101, 1)
```

In [31]:

```
y_train = tf.keras.utils.to_categorical(y_train, 2)
y_test = tf.keras.utils.to_categorical(y_test, 2)
```

## model 3

In [37]:

```
# For essay
input_essay = Input(shape=(350), name='Input_essay')
emb_essay = Embedding(vocab_size_essay, 300, weights=[embedding_matrix],
                      input_length=350, trainable=False, name='Emdedding_essay')(input_essay)
lstm_essay = LSTM(units=64, name='LSTM_essay',
                  kernel_regularizer=regularizers.l2(0.001),
                  return_sequences=True)(emb_essay)
flatten_essay = Flatten(name='Flatten_essay')(lstm_essay)

# For encoded categorical and numeric data

input_rest = Input(shape = (101,1,), name='input_rest')
conv_rest1 = Conv1D(128, 3, activation='relu', kernel_initializer='he_normal', name='conv_rest1')(input_r
conv_rest2 = Conv1D(64, 3, activation='relu', kernel_initializer='he_normal', name='conv_rest2')(conv_res


flatten_rest = Flatten(name='Flatten_rest')(conv_rest2)

# Concat all the previous results
con = Concatenate(axis=1)([flatten_essay, flatten_rest])

dense1 = Dense(128, activation='relu', kernel_initializer='he_normal', name='Dense1', kernel_regularizer=
dense2 = Dense(64, activation='relu', kernel_initializer='he_normal', name='Dense2', kernel_regularizer=r

dense3 = Dense(32, activation='relu', kernel_initializer='he_normal', name='Dense3', kernel_regularizer=r
output = Dense(2, activation='softmax', name='Output', kernel_regularizer=regularizers.l2(0.001))(dense3)

model3 = Model([input_essay, input_rest], output)
model3.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
Input_essay (InputLayer)        [(None, 350)]        0
__________________________________________________________________________________________________
input_rest (InputLayer)         [(None, 101, 1)]     0
__________________________________________________________________________________________________
Emdedding_essay (Embedding)     (None, 350, 300)     15108600    Input_essay[0][0]
__________________________________________________________________________________________________
conv_rest1 (Conv1D)             (None, 99, 128)      512         input_rest[0][0]
__________________________________________________________________________________________________
LSTM_essay (LSTM)               (None, 350, 64)      93440       Emdedding_essay[0][0]
__________________________________________________________________________________________________
conv_rest2 (Conv1D)             (None, 97, 64)       24640       conv_rest1[0][0]
__________________________________________________________________________________________________
Flatten_essay (Flatten)         (None, 22400)        0           LSTM_essay[0][0]
__________________________________________________________________________________________________
Flatten_rest (Flatten)          (None, 6208)         0           conv_rest2[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 28608)        0           Flatten_essay[0][0]
                                                                 Flatten_rest[0][0]
__________________________________________________________________________________________________
Dense1 (Dense)                  (None, 128)          3661952     concatenate_1[0][0]
__________________________________________________________________________________________________
Dense2 (Dense)                  (None, 64)           8256        Dense1[0][0]
__________________________________________________________________________________________________
Dense3 (Dense)                  (None, 32)           2080        Dense2[0][0]
__________________________________________________________________________________________________
Output (Dense)                  (None, 2)            66          Dense3[0][0]
==================================================================================================
Total params: 18,899,546
Trainable params: 3,790,946
Non-trainable params: 15,108,600
__________________________________________________________________________________________________
```

In [39]:

```
logdir3 = os.path.join("logs_model3", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback3 = TensorBoard(log_dir=logdir3,histogram_freq=1, write_graph=True)
myCallback3 = My_callback(3, 10, ([X_test_essay, X_test_rest],y_test))

opti = tf.keras.optimizers.Adam(learning_rate = 0.001)
model3.compile(optimizer=opti,loss='categorical_crossentropy',metrics=['accuracy', auc])
model3.fit([X_train_essay, X_train_rest], y_train,
           epochs=10,
           validation_data=([X_test_essay, X_test_rest], y_test),
           batch_size=512, callbacks=[tensorboard_callback3, myCallback3])
```

```
Epoch 1/10
161/161 [==============================] - 22s 123ms/step - loss: 1.0394 - accuracy: 0.8410 - auc:
0.6293 - val_loss: 0.5712 - val_accuracy: 0.8497 - val_auc: 0.7183
Epoch 2/10
161/161 [==============================] - 19s 118ms/step - loss: 0.5435 - accuracy: 0.8497 - auc:
0.7243 - val_loss: 0.4747 - val_accuracy: 0.8505 - val_auc: 0.7318
Epoch 3/10
161/161 [==============================] - 19s 118ms/step - loss: 0.4635 - accuracy: 0.8533 - auc:
0.7439 - val_loss: 0.4402 - val_accuracy: 0.8536 - val_auc: 0.7399
Epoch 4/10
161/161 [==============================] - 19s 118ms/step - loss: 0.4312 - accuracy: 0.8581 - auc:
0.7546 - val_loss: 0.4333 - val_accuracy: 0.8524 - val_auc: 0.7376
Epoch 5/10
161/161 [==============================] - 19s 118ms/step - loss: 0.4226 - accuracy: 0.8564 - auc:
0.7555 - val_loss: 0.4176 - val_accuracy: 0.8513 - val_auc: 0.7480
Epoch 6/10
161/161 [==============================] - 19s 118ms/step - loss: 0.4092 - accuracy: 0.8573 - auc:
0.7644 - val_loss: 0.4134 - val_accuracy: 0.8494 - val_auc: 0.7513
Epoch 7/10
161/161 [==============================] - 19s 118ms/step - loss: 0.4046 - accuracy: 0.8563 - auc:
0.7699 - val_loss: 0.4059 - val_accuracy: 0.8518 - val_auc: 0.7519
Epoch 8/10
161/161 [==============================] - 19s 118ms/step - loss: 0.3988 - accuracy: 0.8561 - auc:
0.7689 - val_loss: 0.4037 - val_accuracy: 0.8528 - val_auc: 0.7497
Validation auc not improving and terminated at epoch 8
Saving best weights before terminating having val_auc 0.751909613609314
```

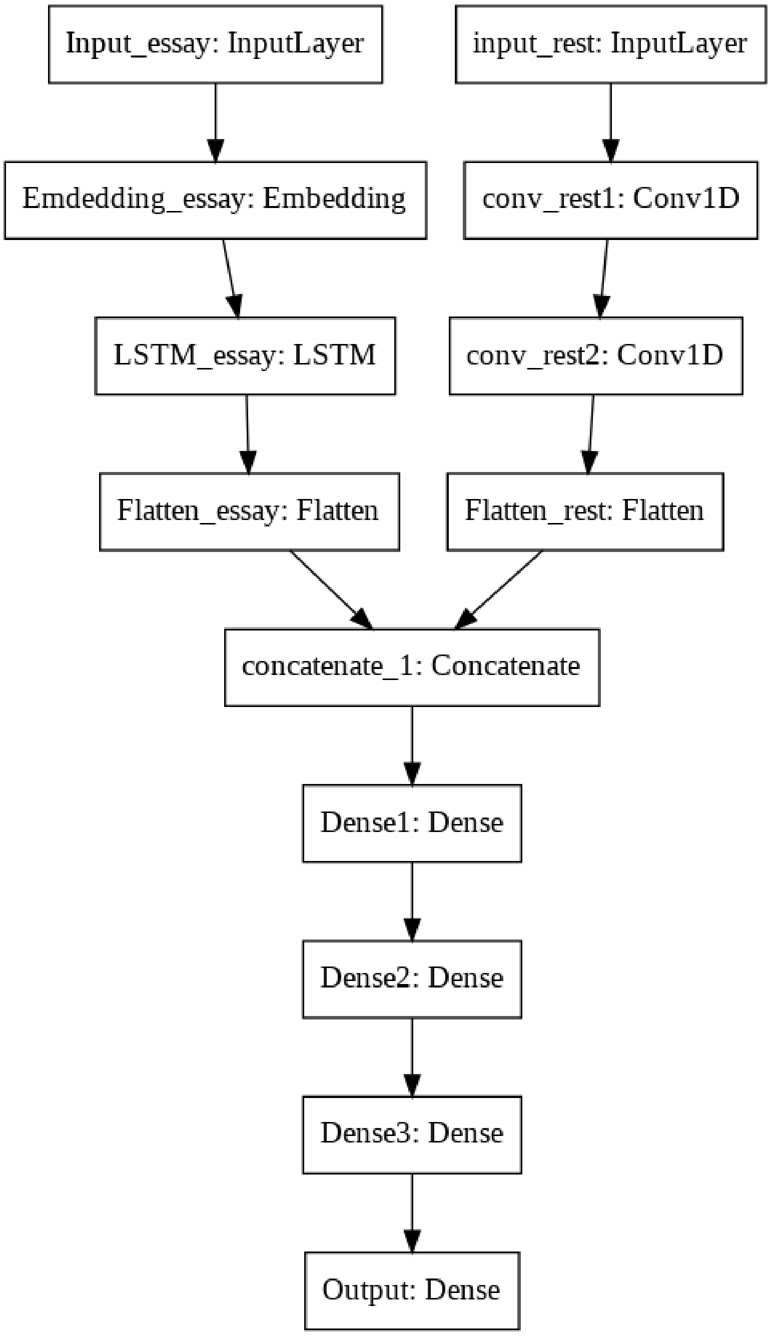
Out[39]:

```
<tensorflow.python.keras.callbacks.History at 0x7f90f5e9d2e8>
```

In [40]:

```
plot_model(model3, to_file='model3.png')
```

Out[40]:

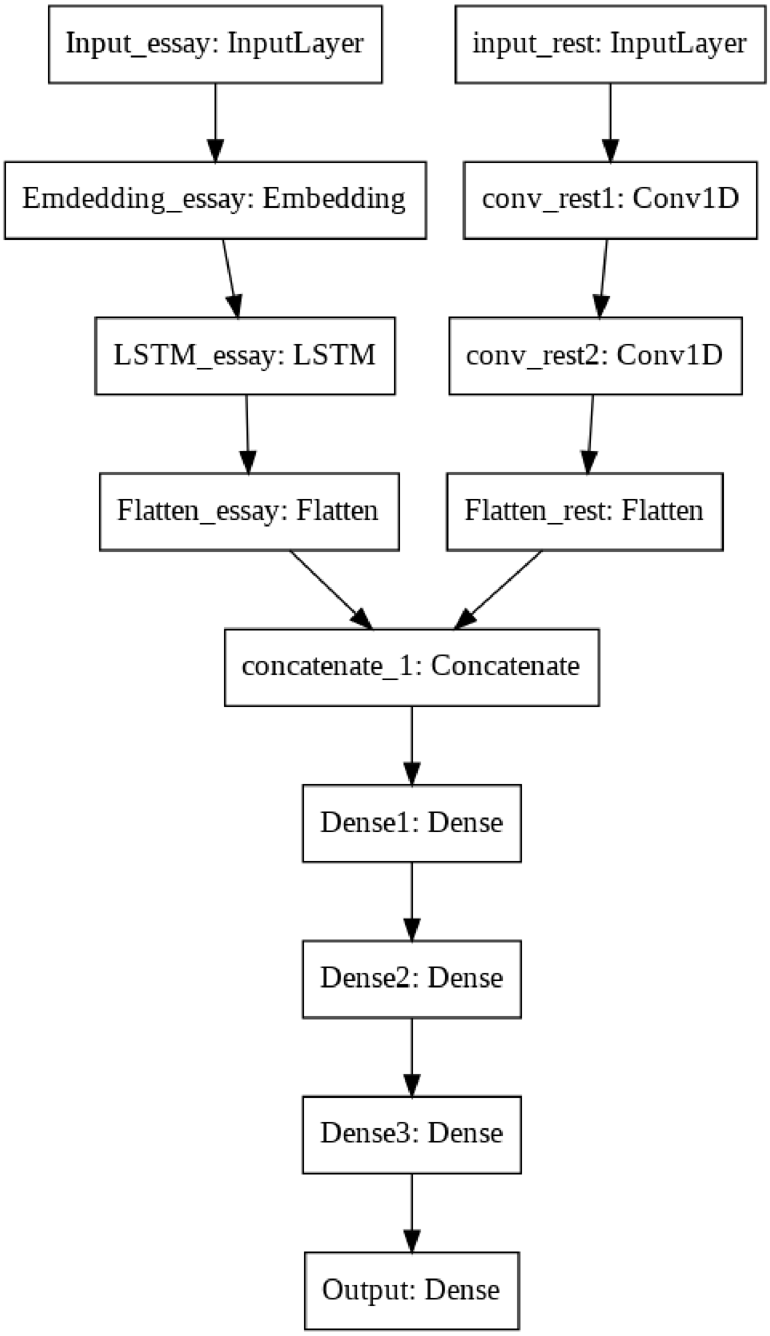


In [43]:

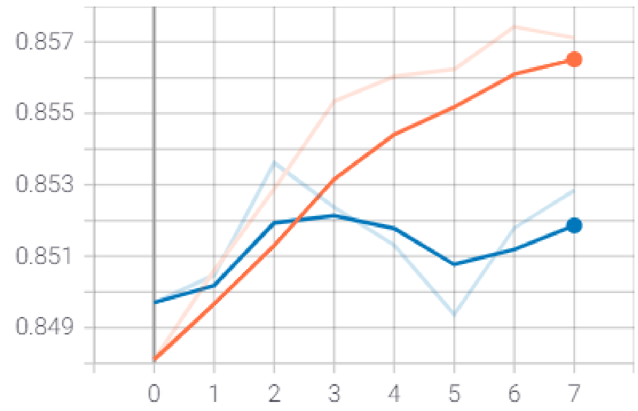
```
%tensorboard --logdir logs_model3
```

**Model 3:**

AUC: 0.7519

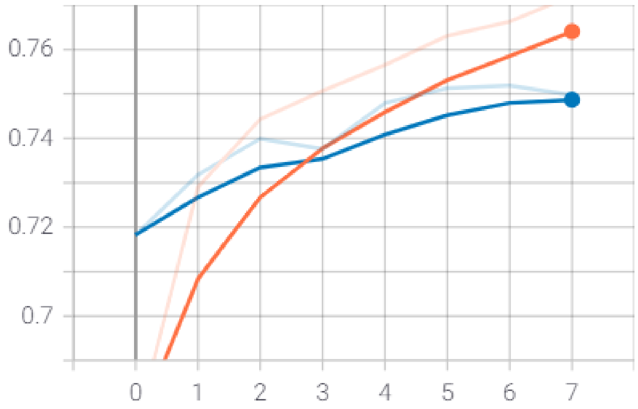

Input_essay: InputLayer
input_rest: InputLayer
Emdedding_essay: Embedding
conv_rest1: Conv1D
LSTM_essay: LSTM
conv_rest2: Conv1D
Flatten_essay: Flatten
Flatten_rest: Flatten
concatenate_1: Concatenate
Dense1: Dense
Dense2: Dense
Dense3: Dense
Output: Dense

epoch_accuracy



## epoch_auc

epoch_auc


0.76
0.74
0.72
0.7
0
1
2
3
4
5
6
7