In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.utils import class_weight
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
from scipy.sparse import hstack

from tqdm import tqdm
import os
import datetime

import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.initializers import RandomNormal
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Input, Embedding, Dense, Flatten, concate
from tensorflow.keras.layers import BatchNormalization, Activation, CuDNNLST
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, Tensor
```

Splitting data into Train and Test

In [2]:

```
prepeocessed_data = pd.read_csv('preprocessed_data.csv')
prepeocessed_data.head(2)
```

Out[2]:

	Unnamed: 0	Unnamed: 0.1	id	teacher_id	teacher
0	0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	
1	1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	

2 rows × 21 columns

In [3]:

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

Out[3]:

```
(109248, 20)
```

In [4]:

```
# 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=0.2, stra
X_test.shape
```

Out[4]:

```
(21850, 20)
```

In [5]:

```
X_train.head(2)
```

Out[5]:

	Unnamed: 0	Unnamed: 0.1	id	teacher_id	t
109078	109078	109837	p217509	c15f13b993b5478c32dcd13b9ebee016	
108069	108069	136104	p012508	6b93a7baa79ac6baade64f7c3484d970	

In [6]:

```
y_train
```

Out[6]:

```
array([1, 1, 1, ..., 1, 1, 1], dtype=int64)
```

In [7]:

```
X_test.head(2)
```

Out[7]:

	Unnamed: 0	Unnamed: 0.1	id	teacher_id	te
50340	50340	7583	p162251	49424684316a63797035055963389a0a	
65013	65013	32550	p204712	2b9f09cb8ce51fb624db6ba45f66d9ba	

In [8]:

```
y_test
```

Out[8]:

```
array([1, 1, 1, ..., 1, 0, 1], dtype=int64)
```

In [9]:

```
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
    return model
model = loadGloveModel('glove.42B.300d.txt')
```

```
951it [00:00, 9438.94it/s]

Loading Glove Model

1917495it [03:28, 9206.60it/s]

Done. 1917495  words loaded!
```

In [10]:

```
t = Tokenizer()
t.fit_on_texts(X_train['essay'].values)
vocab_size = len(t.word_index) + 1
# integer encode the documents
X_train_encoded_docs = t.texts_to_sequences(X_train['essay'].values)
X_test_encoded_docs = t.texts_to_sequences(X_test['essay'].values)

# pad documents to a max length of 300 words
max_length = 300
X_train_padded_docs = pad_sequences(X_train_encoded_docs, maxlen=max_length,
X_test_padded_docs = pad_sequences(X_test_encoded_docs, maxlen=max_length, pa

embedding_matrix = np.zeros((vocab_size, max_length))

for word, i in tqdm(t.word_index.items()):
    embedding_vector = model.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

print(X_train_padded_docs.shape)
print(X_test_padded_docs.shape)
```

```
100%|██████████████████████████████████████████████████████
███████████████| 61833/61833 [00:00<00:00, 412605.36it/s]

(87398, 300)
(21850, 300)
```

In [11]:

```
print(vocab_size)
print(embedding_matrix.shape)
print(embedding_matrix[1])
```

```
61834
(61834, 300)
[-2.4837e-01 -4.5461e-01  3.9227e-02 -2.8422e-01 -3.1852e-02
2.6355e-01
 -4.6323e+00  1.3890e-02 -5.3928e-01 -8.4454e-02  6.1556e-02 -
4.1552e-01
 -1.4599e-01 -5.9321e-01 -2.8738e-02 -3.4991e-02 -2.9698e-01 -
7.9850e-02
  2.7312e-01  2.2040e-01 -8.9859e-02  8.8265e-04 -4.1991e-01 -
1.2536e-01
 -5.4629e-02  3.0550e-02  1.9340e-01 -6.3945e-02  2.7405e-02
5.1193e-02
 -3.8656e-01 -1.1085e-01  1.7259e-01  2.9804e-01 -3.5183e-01
1.3150e-01
 -5.4006e-01 -7.6677e-01 -5.5168e-04  1.3076e-01  2.5101e-02
6.2106e-01
 -2.4797e-01 -3.9790e-01 -3.6116e-01 -5.1967e-01  3.0138e-02 -
5.2436e-02
  6.9281e-02  3.5252e-02 -2.1402e-01  2.4836e-01 -1.5693e-01
1.2829e-01
  3.5425e-01 -1.6080e-01 -5.0720e-03 -3.0656e-01 -2.9514e-01 -
1.3554e-01
 -1.4385e-01 -4.0552e-01  5.7233e-01 -2.7670e-01  3.0519e-01
1.5586e-01
  1.6086e-02 -2.2009e-01  4.8589e-01 -4.1384e-01  2.0546e-01
4.0491e-01
  4.1558e-02 -1.3542e-01  2.2544e-01 -2.3629e-01  1.5193e-01 -
1.0859e-02
 -8.2662e-02 -5.5484e-01 -6.1584e-02 -1.1112e-01 -1.1982e-01 -
3.7064e-01
  1.6501e-01  4.4063e-01 -3.3883e-01 -5.7676e-01  5.0847e-01 -
3.5707e-02
 -5.9233e-02  3.0748e-02 -2.7689e-01 -7.0433e-02  2.7786e-02 -
5.9336e-01
 -2.8220e+00 -1.0052e-01  6.7168e-01 -1.7046e-01 -2.5902e-01
2.7938e-01
  3.9992e-01  3.7480e-02 -2.6409e-01 -2.6378e-01  2.0645e-01
1.7564e-01
 -8.0807e-02 -3.8376e-01  2.6602e-01  3.6214e-01 -9.5112e-02
3.5199e-01
 -8.6994e-01 -1.5747e-01 -2.2550e-01 -6.4948e-02 -2.4845e-01
1.5038e-01
 -3.2951e-01 -2.2285e-01 -2.5509e-02 -2.9725e-01 -3.7715e-01
8.9296e-02
 -3.4399e-02  3.3640e-01  3.5534e-01  3.8253e-01  1.7646e-01
1.3305e-01
```

```
 -3.2743e-01 -4.7115e-01  2.4673e-01 -1.5964e-01  1.8212e-01 -
4.1241e-01
  9.8565e-02  3.8118e-01  3.3043e-01  5.1987e-02 -2.1824e-01
2.2214e-01
 -5.9450e-02 -6.3743e-02  4.3723e-01  1.1068e-01  4.7444e-01
5.6891e-01
  3.1123e-01 -2.0272e-01  8.0078e-02 -4.3905e-01 -1.2246e-01 -
2.5057e-02
 -5.7162e-02  1.4250e-01  9.4468e-02  1.2991e-01  1.0444e-01 -
3.9447e-01
 -2.9337e-01 -2.0466e-01  2.0815e-01 -1.6010e-01 -1.4665e-01
5.4511e-01
  2.9740e-01 -2.2959e-01 -1.7050e-01 -6.2371e-02 -5.0399e-01 -
3.8000e-01
 -3.9528e-01  5.7552e-01 -4.6892e-01 -4.3308e-01  1.5018e-01 -
4.1179e-02
  6.2157e-01  1.9874e-02 -1.1969e-01 -2.5611e-01  2.6602e-01 -
3.7383e-01
  1.2936e-01 -5.0006e-02 -1.1554e-01 -1.7163e-01 -4.2430e-01
1.9844e-01
  5.0611e-01 -1.1093e-01 -1.3939e-01 -5.9377e-01  6.7338e-01
3.8497e-01
  6.2604e-01 -2.0128e-01  3.0058e-01 -1.3946e-01 -1.6186e-01
1.2168e-01
 -1.8410e-02  6.1356e-01 -1.9887e-01  1.9250e-01  8.4372e-03 -
5.0757e-01
  3.5858e-01 -4.9729e-01 -4.4725e-01  2.1423e-02 -2.0769e-01
8.3729e-02
  2.2032e-01  1.4404e-01  1.2590e-03 -4.4309e-01 -1.7242e-01 -
3.5300e-01
 -2.9477e-01  3.2898e-01 -3.1910e+00  3.8910e-01  3.5654e-01
5.2134e-02
  2.0576e-01 -8.8649e-02  1.6398e-01  1.1203e-01  2.8590e-01
2.8940e-01
 -4.4349e-01  9.1036e-01 -3.0902e-01 -1.3985e-01 -3.9499e-01 -
2.7299e-02
 -1.5201e-01  8.4418e-02 -3.7196e-01  4.9827e-02  1.4128e-01 -
1.5126e-01
 -1.6107e-01  4.0226e-03  1.6799e-01 -2.5429e-01 -1.5074e-01 -
5.7409e-01
 -1.5611e-01  6.8407e-02  2.4832e-01  1.6828e-01  7.2764e-02 -
8.6728e-02
  2.1982e-03  1.3593e-01  7.0224e-01 -4.5976e-01 -2.4506e-01 -
3.3874e-01
 -1.0952e-01  2.4698e-01 -5.5919e-01 -3.8866e-01 -1.3372e-01
9.1943e-02
 -1.0543e-01 -3.1319e-01 -2.9952e-01 -2.0611e-01  1.7976e-01
4.5800e-01
 -7.2402e-02  1.6118e-01 -4.1649e-01 -3.0103e-01  2.3234e-01 -
5.0139e-02
  1.0026e-01  3.8974e-01 -6.1342e-02  2.6626e-01 -1.5671e-01
7.5136e-02
 -4.2926e-01 -1.2025e-01  8.2736e-02 -6.2469e-01  4.4267e-02
```

```
6.0673e-01
 -1.2458e-01 -1.5443e-01 -1.6339e-01  5.3097e-02  1.5458e-01 -
3.8053e-01]
```

1.4 Encoding Categorical and Numerical features

1.4.1 encoding categorical features: clean_categories

In [12]:

```
vectorizer_cat = CountVectorizer()
vectorizer_cat.fit(X_train['clean_categories'].values) # fit has to happen on

X_train_cc_ohe = vectorizer_cat.transform(X_train['clean_categories'].values)
X_test_cc_ohe = vectorizer_cat.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_cc_ohe.shape, y_train.shape)
print(X_test_cc_ohe.shape, y_test.shape)
print(vectorizer_cat.get_feature_names())
```

```
After vectorizations
(87398, 9) (87398,)
(21850, 9) (21850,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_c
ivics', 'literacy_language', 'math_science', 'music_arts', 'sp
ecialneeds', 'warmth']
```

1.4.2 encoding categorical features: clean_subcategories

In [13]:

```
vectorizer_subcat = CountVectorizer()
vectorizer_subcat.fit(X_train['clean_subcategories'].values) # fit has to hap

X_train_csc_ohe = vectorizer_subcat.transform(X_train['clean_subcategories'].
X_test_csc_ohe = vectorizer_subcat.transform(X_test['clean_subcategories'].va

print("After vectorizations")
print(X_train_csc_ohe.shape, y_train.shape)
print(X_test_csc_ohe.shape, y_test.shape)
print(vectorizer_subcat.get_feature_names())
```

```
After vectorizations
(87398, 30) (87398,)
(21850, 30) (21850,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civi
cs_government', 'college_careerprep', 'communityservice', 'ear
lydevelopment', 'economics', 'environmentalscience', 'esl', 'e
xtracurricular', 'financialliteracy', 'foreignlanguages', 'gym
_fitness', 'health_lifescience', 'health_wellness', 'history_g
eography', 'literacy', 'literature_writing', 'mathematics', 'm
usic', 'nutritioneducation', 'other', 'parentinvolvement', 'pe
rformingarts', 'socialsciences', 'specialneeds', 'teamsports',
'visualarts', 'warmth']
```

1.4.3 encoding categorical features: school_state

In [14]:

```
vectorizer_school_state = CountVectorizer()
vectorizer_school_state.fit(X_train['school_state'].values)

X_train_state_ohe = vectorizer_school_state.transform(X_train['school_state']
X_test_state_ohe = vectorizer_school_state.transform(X_test['school_state'].

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer_school_state.get_feature_names())
```

```
After vectorizations
(87398, 51) (87398,)
(21850, 51) (21850,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl',
'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'm
d', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'n
h', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 's
c', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'w
y']
```

1.4.4 encoding categorical features: teacher_prefix

In [15]:

```
vectorizer_prefix = CountVectorizer()
vectorizer_prefix.fit(X_train['teacher_prefix'].values)

X_train_teacher_ohe = vectorizer_prefix.transform(X_train['teacher_prefix'].
X_test_teacher_ohe = vectorizer_prefix.transform(X_test['teacher_prefix'].va

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

```
After vectorizations
(87398, 5) (87398,)
(21850, 5) (21850,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
```

1.4.5 encoding categorical features: project_grade_category

In [16]:

```
vectorizer_grade = CountVectorizer()
vectorizer_grade.fit(X_train['project_grade_category'].values)

X_train_grade_ohe = vectorizer_grade.transform(X_train['project_grade_categor
X_test_grade_ohe = vectorizer_grade.transform(X_test['project_grade_category'

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer_grade.get_feature_names())
```

```
After vectorizations
(87398, 4) (87398,)
(21850, 4) (21850,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
```

1.4.6 encoding numerical features: price

In [17]:

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaler.fit(X_train['price'].values.reshape(-1,1))
X_train_price=scaler.transform(X_train['price'].values.reshape(-1,1))
X_test_price=scaler.transform(X_test['price'].values.reshape(-1,1))


print(X_train_price.shape)
print(X_test_price.shape)
print(X_train_price)
print(X_test_price)
```

```
(87398, 1)
(21850, 1)
[[0.02126153]
 [0.00848041]
 [0.02222569]
 ...
 [0.02341289]
 [0.02103149]
 [0.04478043]]
[[0.04994129]
 [0.01193498]
 [0.03926652]
 ...
 [0.01223203]
 [0.04165792]
 [0.01909417]]
```

1.4.7 encoding numerical features: teacher_number_of_previously_posted_projects

In [18]:

```
scaler = MinMaxScaler()
scaler.fit(X_train['teacher_number_of_previously_posted_projects'].values.res
X_train_pre_proj=scaler.transform(X_train['teacher_number_of_previously_poste
X_test_pre_proj=scaler.transform(X_test['teacher_number_of_previously_posted_


print(X_train_pre_proj.shape)
print(X_test_pre_proj.shape)
```

```
(87398, 1)
(21850, 1)
```

1.4.8 encoding numerical features: quantity

In [19]:

```
scaler = MinMaxScaler()
scaler.fit(X_train['quantity'].values.reshape(-1,1))
X_train_quantity=scaler.transform(X_train['quantity'].values.reshape(-1,1))
X_test_quantity=scaler.transform(X_test['quantity'].values.reshape(-1,1))


print(X_train_quantity.shape)
print(X_test_quantity.shape)
```

```
(87398, 1)
(21850, 1)
```

In [ ]:

```

```

In [20]:

```

X_train_cn = hstack((X_train_cc_ohe,X_train_csc_ohe,X_train_teacher_ohe,X_tra
                        X_train_state_ohe,X_train_price,X_train_quantity,X_tr

X_test_cn = hstack((X_test_cc_ohe,X_test_csc_ohe,X_test_teacher_ohe,X_test_gr
                       X_test_state_ohe,X_test_price,X_test_quantity,X_test_
```

In [21]:

```
print(X_train_cn.shape)
print(X_test_cn.shape)
```

```
(87398, 102)
(21850, 102)
```

In [22]:

```
print('vocabulary size is ',vocab_size)
```

```
vocabulary size is  61834
```

In [1]:

```
'''
These cells are to load the data if the system crashes during training
'''
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.utils import class_weight
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
from scipy.sparse import hstack

from tqdm import tqdm
import os
import datetime

import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.initializers import RandomNormal
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Input, Embedding, Dense, Flatten, concate
from tensorflow.keras.layers import BatchNormalization, Activation, CuDNNLST
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, Tensor
vocab_size = 61834
```

In [23]:

```
#Saving processed data

from sklearn.externals import joblib

joblib.dump(embedding_matrix, 'embedding_matrix3.pkl')
joblib.dump(X_train_padded_docs, 'X_train_padded_docs3.pkl')
joblib.dump(X_train_cn, 'X_train_cn3.pkl')
joblib.dump(y_train, 'y_train3.pkl')


joblib.dump(X_test_padded_docs, 'X_test_padded_docs3.pkl')
joblib.dump(X_test_cn, 'X_test_cn3.pkl')
joblib.dump(y_test, 'y_test3.pkl')
```

Out[23]:

```
['y_test3.pkl']
```

In [2]:

```
#Loading processed data

from sklearn.externals import joblib

embedding_matrix = joblib.load('embedding_matrix3.pkl')
X_train_padded_docs = joblib.load('X_train_padded_docs3.pkl')
X_train_cn = joblib.load('X_train_cn3.pkl')
y_train = joblib.load('y_train3.pkl')


X_test_padded_docs = joblib.load('X_test_padded_docs3.pkl')
X_test_cn = joblib.load('X_test_cn3.pkl')
y_test = joblib.load('y_test3.pkl')
```

In [3]:

```
from sklearn.metrics import roc_auc_score
def auc( y_true, y_pred ) :
    score = tf.py_func( lambda y_true, y_pred : roc_auc_score( y_true, y_pred
                        [y_true, y_pred],
                        'float32',
                        stateful=True,
                        name='sklearnAUC' )
    return score
```

Model's architecture

In [4]:

```
# Essay Layers
essay_input = Input(shape=(len(X_train_padded_docs[0]),), name='essay_input')

essay_input_1 = Embedding(input_dim=vocab_size,output_dim=300, input_length=
                              weights=[embedding_matrix], trainable=False)(es

essay_input_1 = CuDNNLSTM(units = 64,
                              kernel_initializer= 'he_normal',
                              return_sequences=True)(essay_input_1)
essay_input_1 = Flatten()(essay_input_1)

# Categorical and numerical Layers

cat_num_input = Input(shape=(X_train_cn.shape[1],), name='cat_num_input')

cn_input = Embedding(input_dim=X_train_cn.shape[1], output_dim=128, input_len

conv1d_1 = Conv1D(filters=64, kernel_size=(5), padding='same')(cn_input)

flat_2 = Flatten()(conv1d_1)

# Cconcatinating all the above Layers
x = concatenate([essay_input_1, flat_2])

# Dense Layers
output = Dense(units = 624,
                  activation = 'relu',
                            kernel_initializer = 'he_normal')(x)
output = Dropout(0.7)(output)
output = BatchNormalization()(output)

output = Dropout(0.7)(output)
output = BatchNormalization()(output)
output = Dense(units = 512,
                  activation = 'relu',
                            kernel_initializer = 'he_normal')(output)



output = Dropout(0.8)(output)
output = BatchNormalization()(output)
output = Dense(units = 512,
                  activation = 'relu',
                            kernel_initializer = 'he_normal')(output)
output = Dropout(0.45)(output)
output =Dense(1, activation='sigmoid', name='output')(output)

model_3 = Model(inputs = [essay_input, cat_num_input],
                  outputs=output)

model_3.compile(loss = 'binary_crossentropy',
```

```python
                 optimizer = 'adam',
                 metrics = ['accuracy', auc])
```

```
WARNING:tensorflow:From c:\users\addu\appdata\local\programs\p
ython\python37\lib\site-packages\tensorflow\python\keras\initi
alizers.py:119: calling RandomUniform.__init__ (from tensorflo
w.python.ops.init_ops) with dtype is deprecated and will be re
moved in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of p
assing it to the constructor
WARNING:tensorflow:From c:\users\addu\appdata\local\programs\p
ython\python37\lib\site-packages\tensorflow\python\ops\init_op
s.py:1251: calling VarianceScaling.__init__ (from tensorflow.p
ython.ops.init_ops) with dtype is deprecated and will be remov
ed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of p
assing it to the constructor
WARNING:tensorflow:Large dropout rate: 0.7 (>0.5). In TensorFl
ow 2.x, dropout() uses dropout rate instead of keep_prob. Plea
se ensure that this is intended.
WARNING:tensorflow:Large dropout rate: 0.7 (>0.5). In TensorFl
ow 2.x, dropout() uses dropout rate instead of keep_prob. Plea
se ensure that this is intended.
WARNING:tensorflow:Large dropout rate: 0.8 (>0.5). In TensorFl
ow 2.x, dropout() uses dropout rate instead of keep_prob. Plea
se ensure that this is intended.
WARNING:tensorflow:From <ipython-input-3-85586a596984>:7: py_f
unc (from tensorflow.python.ops.script_ops) is deprecated and
will be removed in a future version.
Instructions for updating:
tf.py_func is deprecated in TF V2. Instead, there are two
    options available in V2.
    - tf.py_function takes a python function which manipulates
tf eager
    tensors instead of numpy arrays. It's easy to convert a tf
eager tensor to
    an ndarray (just call tensor.numpy()) but having access to
eager tensors
    means `tf.py_function`s can use accelerators such as GPUs
as well as
    being differentiable using a gradient tape.
    - tf.numpy_function maintains the semantics of the depreca
ted tf.py_func
    (it is not differentiable, and manipulates numpy arrays).
It drops the
    stateful argument making all functions stateful.

WARNING:tensorflow:From c:\users\addu\appdata\local\programs\p
ython\python37\lib\site-packages\tensorflow\python\ops\nn_imp
l.py:180: add_dispatch_support.<locals>.wrapper (from tensorfl
ow.python.ops.array_ops) is deprecated and will be removed in
```

```
a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.w
here
```

Training the model with train data

In [5]:

```
log_dir="logs\\model_3_log" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S
tensorboard = TensorBoard(log_dir=log_dir)

mcp_save = ModelCheckpoint('.model3_best_weights.hdf5', save_best_only=True,

callbacks_list = [mcp_save,tensorboard]
```

In [6]:

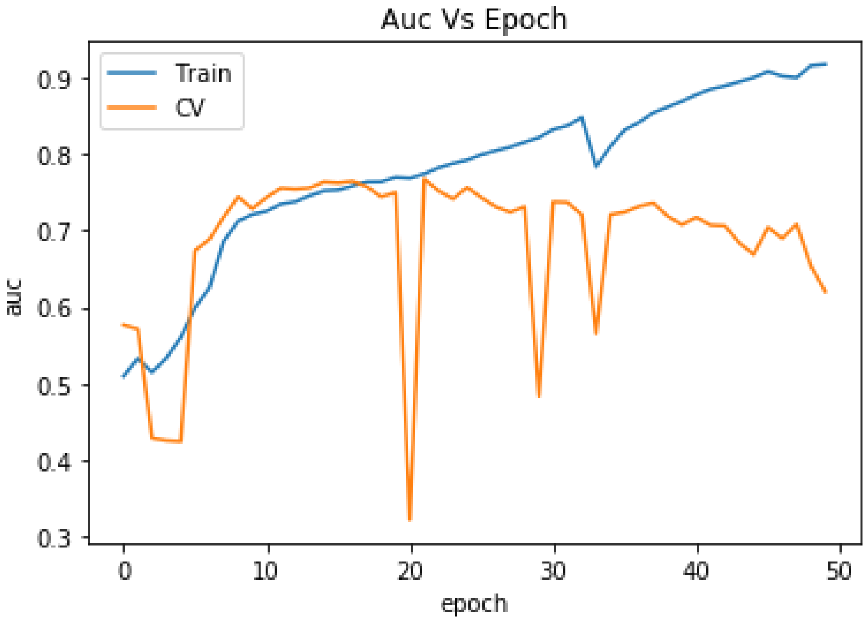
```
history = model_3.fit([X_train_padded_docs, X_train_cn.todense()],
              y_train,
              batch_size = 1024,
              epochs = 50,
              callbacks=callbacks_list,
              validation_split=0.2,
              verbose = 2)
```

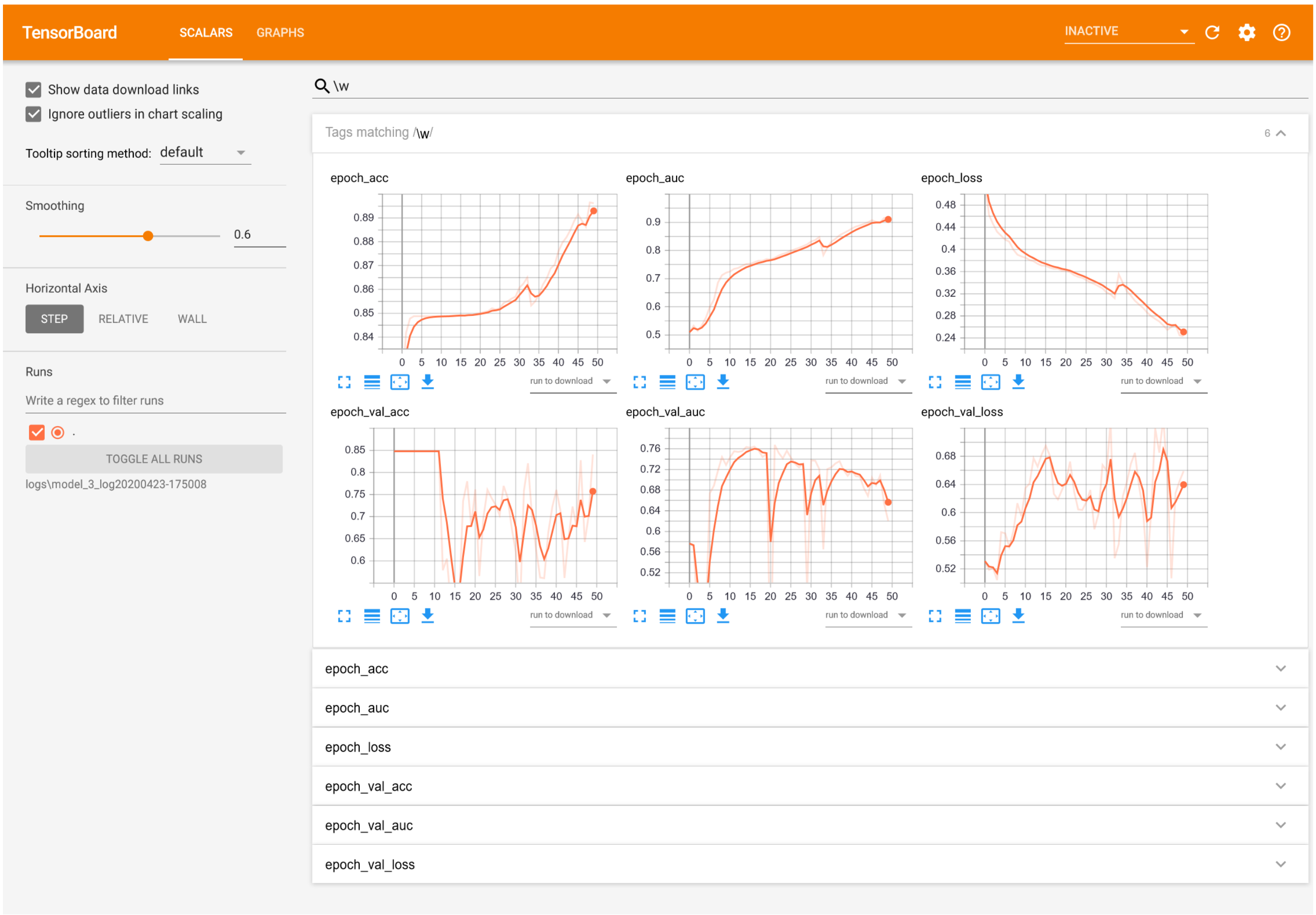
```
Train on 69918 samples, validate on 17480 samples
Epoch 1/50
69918/69918 - 8s - loss: 0.5265 - acc: 0.8183 - auc: 0.5096
- val_loss: 0.5312 - val_acc: 0.8478 - val_auc: 0.5765
Epoch 2/50
69918/69918 - 5s - loss: 0.4626 - acc: 0.8418 - auc: 0.5328
- val_loss: 0.5188 - val_acc: 0.8478 - val_auc: 0.5712
Epoch 3/50
69918/69918 - 5s - loss: 0.4460 - acc: 0.8478 - auc: 0.5148
- val_loss: 0.5211 - val_acc: 0.8478 - val_auc: 0.4286
Epoch 4/50
69918/69918 - 5s - loss: 0.4321 - acc: 0.8488 - auc: 0.5332
- val_loss: 0.5042 - val_acc: 0.8478 - val_auc: 0.4256
Epoch 5/50
69918/69918 - 5s - loss: 0.4249 - acc: 0.8488 - auc: 0.5603
- val_loss: 0.5722 - val_acc: 0.8478 - val_auc: 0.4244
Epoch 6/50
69918/69918 - 5s - loss: 0.4179 - acc: 0.8488 - auc: 0.5991
- val_loss: 0.5706 - val_acc: 0.8478 - val_auc: 0.6740
Epoch 7/50
```

Auc vs Epoch

In [7]:

```
# summarize history for accuracy
plt.plot(history.history['auc'])
plt.plot(history.history['val_auc'])
plt.title('Auc Vs Epoch')
plt.ylabel('auc')
plt.xlabel('epoch')
plt.legend(['Train', 'CV'], loc='right bottom')
plt.show()
```





☑ Show data download links

☑ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP RELATIVE WALL

Runs

Write a regex to filter runs

☑ ◉ .

logs\model_3_log20200423-175008

\w

Tags matching /\w/ 6

Testing the model with test data

In [8]:

```
# Essay Layers
essay_input = Input(shape=(len(X_train_padded_docs[0]),), name='essay_input')

essay_input_1 = Embedding(input_dim=vocab_size,output_dim=300, input_length=
                              weights=[embedding_matrix], trainable=False)(es

essay_input_1 = CuDNNLSTM(units = 64,
                              kernel_initializer= 'he_normal',
                              return_sequences=True)(essay_input_1)
essay_input_1 = Flatten()(essay_input_1)

# Categorical and numerical Layers

cat_num_input = Input(shape=(X_train_cn.shape[1],), name='cat_num_input')

cn_input = Embedding(input_dim=X_train_cn.shape[1], output_dim=128, input_len

conv1d_1 = Conv1D(filters=64, kernel_size=(5), padding='same')(cn_input)

flat_2 = Flatten()(conv1d_1)

# Cconcatinating all the above Layers
x = concatenate([essay_input_1, flat_2])

# Dense Layers
output = Dense(units = 624,
                  activation = 'relu',
                            kernel_initializer = 'he_normal')(x)
output = Dropout(0.7)(output)
output = BatchNormalization()(output)

output = Dropout(0.7)(output)
output = BatchNormalization()(output)
output = Dense(units = 512,
                  activation = 'relu',
                            kernel_initializer = 'he_normal')(output)



output = Dropout(0.8)(output)
output = BatchNormalization()(output)
output = Dense(units = 512,
                  activation = 'relu',
                            kernel_initializer = 'he_normal')(output)
output = Dropout(0.45)(output)
output =Dense(1, activation='sigmoid', name='output')(output)

model_3 = Model(inputs = [essay_input, cat_num_input],
                  outputs=output)

model_3.compile(loss = 'binary_crossentropy',
                  optimizer = 'adam',
                  metrics = ['accuracy', auc])
```

```
WARNING:tensorflow:Large dropout rate: 0.7 (>0.5). In TensorFl
ow 2.x, dropout() uses dropout rate instead of keep_prob. Plea
se ensure that this is intended.
WARNING:tensorflow:Large dropout rate: 0.7 (>0.5). In TensorFl
ow 2.x, dropout() uses dropout rate instead of keep_prob. Plea
se ensure that this is intended.
```

In [9]:

```
#loading the best weights obtained from traing model on train data
model_3.load_weights(".model3_best_weights.hdf5")
```

In [10]:

```
score=model_3.evaluate([X_test_padded_docs, X_test_cn.todense()],y_test,
                           batch_size=1024, verbose=0)[2]
```

Auc score of model on unseen Test data

In [11]:

```
print('Test Auc obtained is ',score)
```

```
Test Auc obtained is  0.75855786
```

In [12]:

```
model_3.summary()
```

```
Model: "model_1"
______________________________________________________________
____________________________________
Layer (type)                    Output Shape         Param #
Connected to
==============================================================
====================================
essay_input (InputLayer)        [(None, 300)]        0
______________________________________________________________
____________________________________
cat_num_input (InputLayer)      [(None, 102)]        0
______________________________________________________________
____________________________________
embedding_2 (Embedding)         (None, 300, 300)     18550200
essay_input[0][0]
______________________________________________________________
____________________________________
embedding_3 (Embedding)         (None, 102, 128)     13056
cat_num_input[0][0]
______________________________________________________________
____________________________________
cu_dnnlstm_1 (CuDNNLSTM)        (None, 300, 64)      93696
embedding_2[0][0]
______________________________________________________________
____________________________________
conv1d_1 (Conv1D)               (None, 102, 64)      41024
embedding_3[0][0]
______________________________________________________________
____________________________________
flatten_2 (Flatten)             (None, 19200)        0
cu_dnnlstm_1[0][0]
______________________________________________________________
____________________________________
flatten_3 (Flatten)             (None, 6528)         0
conv1d_1[0][0]
______________________________________________________________
____________________________________
concatenate_1 (Concatenate)     (None, 25728)        0
flatten_2[0][0]

flatten_3[0][0]
______________________________________________________________
____________________________________
dense_3 (Dense)                 (None, 624)          16054896
concatenate_1[0][0]
______________________________________________________________
____________________________________
dropout_4 (Dropout)             (None, 624)          0
```

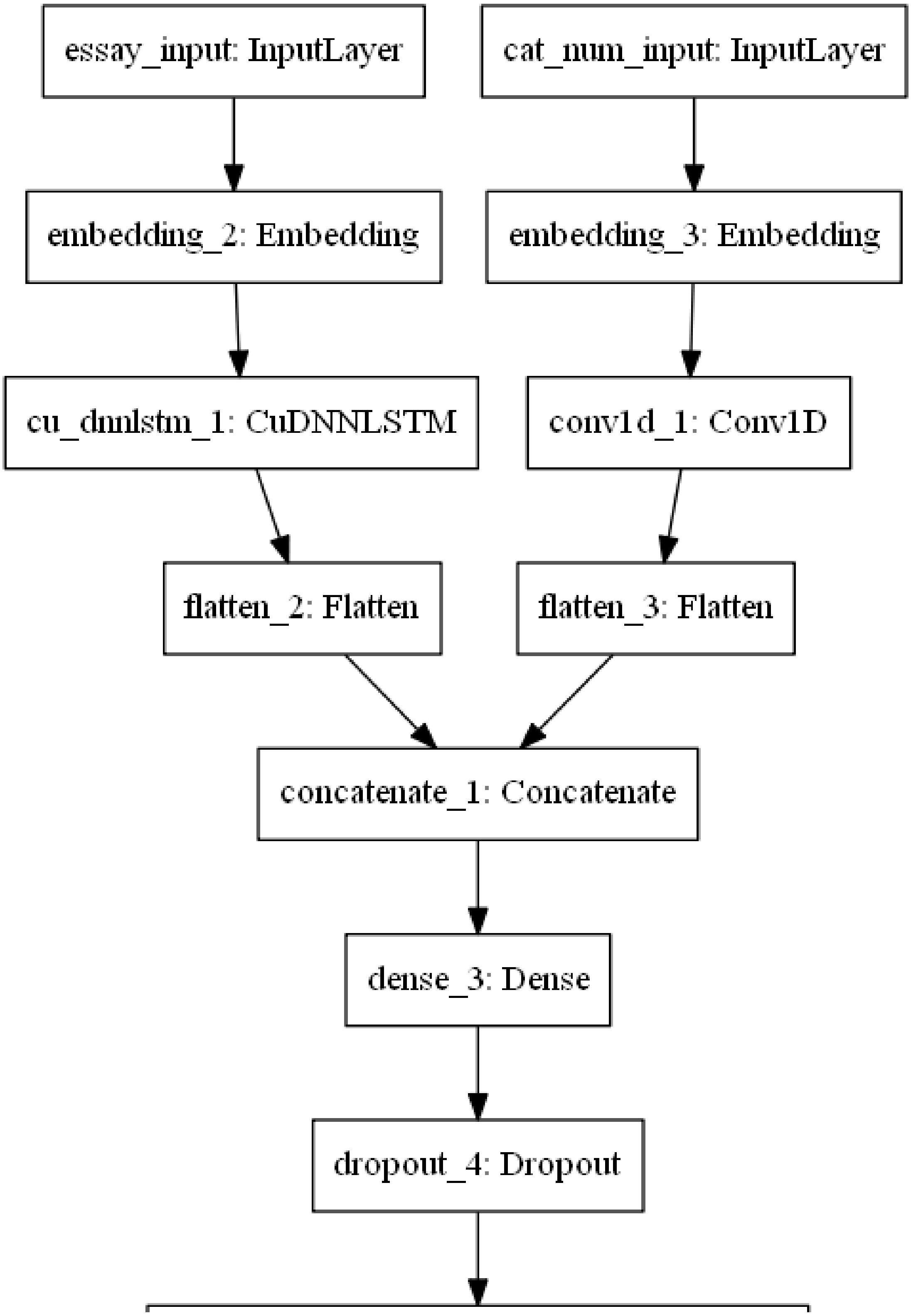
```
dense_3[0][0]
______________________________________________________________
____________________________________
batch_normalization_3 (BatchNor (None, 624)          2496
dropout_4[0][0]
______________________________________________________________
____________________________________
dropout_5 (Dropout)             (None, 624)          0
batch_normalization_3[0][0]
______________________________________________________________
____________________________________
batch_normalization_4 (BatchNor (None, 624)          2496
dropout_5[0][0]
______________________________________________________________
____________________________________
dense_4 (Dense)                 (None, 512)          320000
batch_normalization_4[0][0]
______________________________________________________________
____________________________________
dropout_6 (Dropout)             (None, 512)          0
dense_4[0][0]
______________________________________________________________
____________________________________
batch_normalization_5 (BatchNor (None, 512)          2048
dropout_6[0][0]
______________________________________________________________
____________________________________
dense_5 (Dense)                 (None, 512)          262656
batch_normalization_5[0][0]
______________________________________________________________
____________________________________
dropout_7 (Dropout)             (None, 512)          0
dense_5[0][0]
______________________________________________________________
____________________________________
output (Dense)                  (None, 1)            513
dropout_7[0][0]
==============================================================
====================================
Total params: 35,343,081
Trainable params: 16,789,361
Non-trainable params: 18,553,720
______________________________________________________________
____________________________________
```

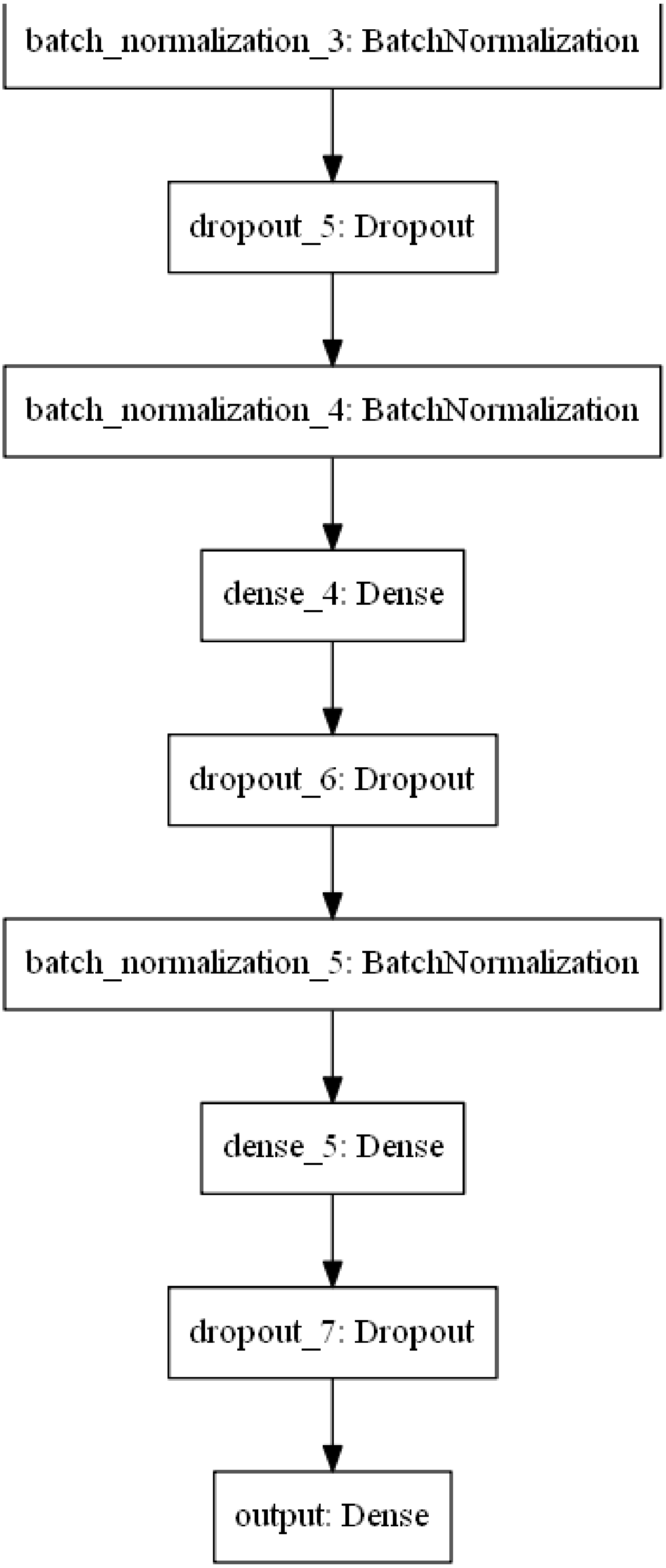
Model's Final Architecture

In [13]:

```
#drawing models
tf.keras.utils.plot_model(
    model_3,
    show_shapes=False,
    show_layer_names=True,
    rankdir='TB'
)
```

Out[13]:





batch_normalization_3: BatchNormalization
dropout_5: Dropout
batch_normalization_4: BatchNormalization
dense_4: Dense
dropout_6: Dropout
batch_normalization_5: BatchNormalization
dense_5: Dense
dropout_7: Dropout
output: Dense

Summary

- Created tensorflow model using text, categorical and numerical layers.
- Performed hyper parameter tuning manually on number of layers, activation functions and optimizers.
- Trained the network using the best obtained hyper parameters.
- Tested the model using Test data and obtained test aucroc score of 0.7585
- Printed the summary of the model along with it's image.