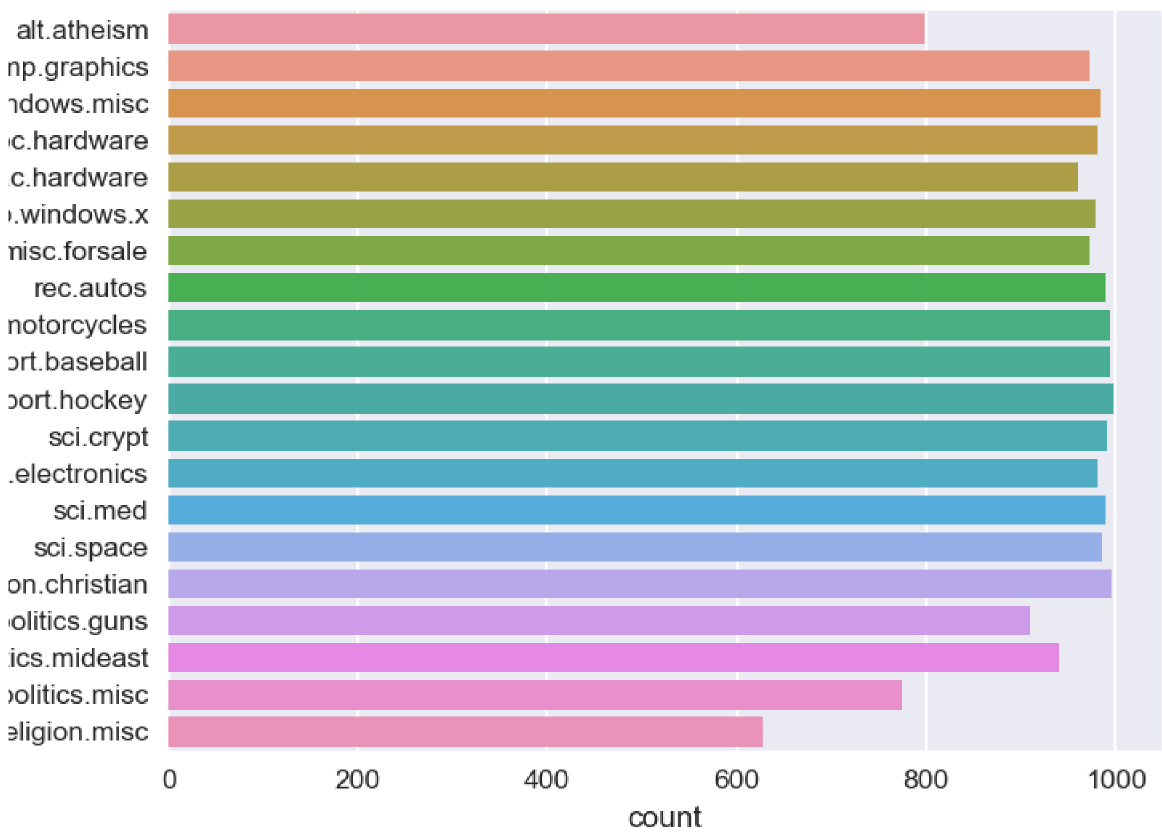
Text Classification:

Data

```
1. we have total of 20 types of documents(Text files) and total 18828 documents(text fil
2. You can download data from this link, in that you will get documents.rar folder.
If you unzip that, you will get total of 18828 documnets. document name is defined as'Cl
so from document name, you can extract the label for that document.
4. Now our problem is to classify all the documents into any one of the class.
5. Below we provided count plot of all the labels in our data.
```

```
### count plot of all the class labels.
```



Assignment:

```
!unrar x '/content/documents.rar'
```

sample document

```
Subject: A word of advice
From: jcopelan@nyx.cs.du.edu (The One and Only)

In article < 65882@mimsy.umd.edu > mangoe@cs.umd.edu (Charley Wingate) writes:
>
>I've said 100 times that there is no "alternative" that should think you
>might have caught on by now.  And there is no "alternative", but the point
>is, "rationality" isn't an alternative either.  The problems of metaphysical
>and religious knowledge are unsolvable-- or I should say, humans cannot
>solve them.

How does that saying go: Those who say it can't be done shouldn't interrupt
those who are doing it.

Jim
--
Have you washed your brain today?
```

Preprocessing:

useful links: http://www.pyregex.com/

1. Find all emails in the document and then get the text after the "@". and then split t
after that remove the words whose length is less than or equal to 2 and also remove'com'
In one doc, if we have 2 or more mails, get all.
Eg:[test@dm1.d.com, test2@dm2.dm3.com]-->[dm1.d.com, dm3.dm4.com]-->[dm1,d,com,dm2,dm3,c
append all those into one list/array. (This will give length of 18828 sentences i.e one
Some sample output was shown below.

> In the above sample document there are emails [jcopelan@nyx.cs.du.edu, 65882@mimsy.umd

preprocessing:
[jcopelan@nyx.cs.du.edu, 65882@mimsy.umd.edu, mangoe@cs.umd.edu] ==> [nyx cs du edu mims
[nyx edu mimsy umd edu umd edu]

2. Replace all the emails by space in the original text.

```
# we have collected all emails and preprocessed them, this is sample output
preprocessed_email
```

```
array(['juliet caltech edu',
       'coding bchs edu newsgate sps mot austlcm sps mot austlcm sps mot com  dna bch
       'batman bmd trw', ..., 'rbdc wsnc org dscomsa desy zeus  desy',
       'rbdc wsnc org morrow stanford edu pangea Stanford EDU',
       'rbdc wsnc org apollo apollo'], dtype=object)
```

```
len(preprocessed_email)
```

```
18828
```

3. Get subject of the text i.e. get the total lines where "Subject:" occur and remove the word which are before the ":" remove the newlines, tabs, punctuations, any special c
Eg: if we have sentance like "Subject: Re: Gospel Dating @ \r\r\n" --> You have to get "
Save all this data into another list/array.

4. After you store it in the list, Replace those sentances in original text by space.

5. Delete all the sentances where sentence starts with **"Write to:"** or **"From:"**.
> In the above sample document check the 2nd line, we should remove that

6. Delete all the tags like "< anyword >"
> In the above sample document check the 4nd line, we should remove that "< 65882@mimsy.

7. Delete all the data which are present in the brackets.
In many text data, we observed that, they maintained the explanation of sentence or translation of sentence to another language in brackets so remove all those.
Eg: "AAIC-The course that gets you HIRED(AAIC - Der Kurs, der Sie anstellt)" --> "AAIC-T

> In the above sample document check the 4nd line, we should remove that "(Charley Winga

8. Remove all the newlines('\n'), tabs('\t'), "-", "\".

9. Remove all the words which ends with ":".
Eg: "Anyword:"
> In the above sample document check the 4nd line, we should remove that "writes:"

10. Decontractions, replace words like below to full words.
please check the donors choose preprocessing for this
Eg: can't -> can not, 's -> is, i've -> i have, i'm -> i am, you're -> you are, i'll -->

There is no order to do point 6 to 10. but you have to get final output correctly

11. Do chunking on the text you have after above preprocessing.
Text chunking, also referred to as shallow parsing, is a task that

follows Part-Of-Speech Tagging and that adds more structure to the sentence.
So it combines the some phrases, named entities into single word.
So after that combine all those phrases/named entities by separating "_".
And remove the phrases/named entities if that is a "Person".
You can use **nltk.ne_chunk** to get these.
Below we have given one example. please go through it.

useful links:
https://www.nltk.org/book/ch07.html
https://stackoverflow.com/a/31837224/4084039
http://www.nltk.org/howto/tree.html
https://stackoverflow.com/a/44294377/4084039

```
#i am living in the New York
print("i am living in the New York -->", list(chunks))
print(" ")
print("-"*50)
print(" ")
#My name is Srikanth Varma
print("My name is Srikanth Varma -->", list(chunks1))
```

```
i am living in the New York --> [('i', 'NN'), ('am', 'VBP'), ('living', 'VBG'), ('in

--------------------------------------------------

My name is Srikanth Varma --> [('My', 'PRP$'), ('name', 'NN'), ('is', 'VBZ'), Tree('P
```

We did chunking for above two lines and then We got one list where each word is mapped t
POS(parts of speech) and also if you see "New York" and "Srikanth Varma",
they got combined and represented as a tree and "New York" was referred as "GPE" and "Sr
so now you have to Combine the "New York" with "_" i.e "New_York"
and remove the "Srikanth Varma" from the above sentence because it is a person.

13. Replace all the digits with space i.e delete all the digits.
> In the above sample document, the 6th line have digit 100, so we have to remove that.

14. After doing above points, we observed there might be few word's like
"_word_" (i.e starting and ending with the _), "_word" (i.e starting with the _), "word_" (i.e ending with the _) remove the _ from these type of words.

15. We also observed some words like **"OneLetter_word"- eg: d_berlin, "TwoLetters_word" - eg: dr_berlin** , in these words we remove the "OneLetter_" (d_berlin "TwoLetters_" (de_berlin ==> berlin). i.e remove the words which are length less than or equal to 2 after spliiting those words by "_".

16. Convert all the words into lower case and lowe case
and remove the words which are greater than or equal to 15 or less than or equal to 2.

17. replace all the words except "A-Za-z_" with space.

18. Now You got Preprocessed Text, email, subject. create a dataframe with those.
Below are the columns of the df.

```
import re
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping chunkers/maxent_ne_chunker.zip.
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Unzipping corpora/words.zip.
True
```

```
import os
files=os.listdir('/content/documents')
text =[]
Class=[]
for f in files:
  name=str(f).split('_')[0]
  Class.append(name.split('.')[-2]+'.'+name.split('.')[-1])
  #https://stackoverflow.com/questions/16883447/how-to-read-a-c-source-iso-8859-text
  with open('/content/documents/'+str(f),'r',encoding="ISO-8859-1") as f1:
    my_lines = f1.read()
  text.append(my_lines)
```

```
import pandas as pd
Df=pd.DataFrame()
Df['text']=text
Df['class']=Class
Df.head()
```

	text	class
0	From: julie@eddie.jpl.nasa.gov (Julie Kangas)\...	politics.misc
1	From: scrowe@hemel.bull.co.uk (Simon Crowe)\nS...	comp.graphics
2	From: art@cs.UAlberta.CA (Art Mulder)\nSubject...	windows.x
3	From: rem@buitc.bu.edu (Robert Mee)\nSubject: ...	ms-windows.misc

```
def mail_text(text):
  c=[]
  #https://stackoverflow.com/questions/17681670/extract-email-sub-strings-from-large-docum
  b=re.findall(r'[\w\.-]+@[\w\.-]+\.\w+', text)
  for mail in b:
    d=mail.split('@')[-1].split('.')
    c.extend(d)
  return ' '.join([w for w in c if len(w)>2])


def subject_1(text):
  b=re.findall("Subject:.*",text)
  c=re.sub("Subject: Re?",'',b[0])
  d = re.sub('[^A-Za-z0-9]+', ' ',c)
  #remove extra space
  e=re.sub(' +', ' ',d)

  return e


def decontracted(phrase):
 # specific
 phrase = re.sub(r"won't", "will not", phrase)
 phrase = re.sub(r"can\'t", "can not", phrase)
 # general
 phrase = re.sub(r"n\'t", " not", phrase)
 phrase = re.sub(r"\'re", " are", phrase)
 phrase = re.sub(r"\'s", " is", phrase)
 phrase = re.sub(r"\'d", " would", phrase)
 phrase = re.sub(r"\'ll", " will", phrase)
 phrase = re.sub(r"\'t", " not", phrase)
 phrase = re.sub(r"\'ve", " have", phrase)
 phrase = re.sub(r"\'m", " am", phrase)
 return phrase


def chunking(text):
  persion=[]
  gep=[]
  for sent in nltk.sent_tokenize(text):
    for chunk in nltk.ne_chunk(nltk.pos_tag(nltk.word_tokenize(sent))):
      if hasattr(chunk, 'label'):
        if chunk.label()=='PERSON':
          persion.append(list(chunk))
        if chunk.label()=='GPE' :
          gep.append(list(chunk))
  for i in gep:
```

```
      if len(i)==2:
        text=re.sub(i[0][0]+' '+i[1][0],i[0][0]+'_'+i[1][0],text)

    for i in persion:
        if len(i)==2:
          text= re.sub(i[0][0]+' '+i[1][0],'',text)

    return text




  def preprocess(text):
    # 1.,2.  https://stackoverflow.com/questions/17681670/extract-email-sub-strings-from-lar
    text=re.sub('[\w\.-]+@[\w\.-]+\.\w+', ' ',text)
    text=re.sub("Subject:.*\w+",'',text)
    #3. Delete all the sentances where sentence starts with "Write to:" or "From:".
    text=re.sub("From:.*?", ' ',text)
    text=re.sub("Write to:.*?",' ',text)
    # 4. Delete all the tags like "< anyword >"
    clean = re.compile('<.*?>')
    text=re.sub(clean,' ',text)
    # 5. Delete all the data which are present in the brackets.
    clean1 = re.compile('\(.*\)')
    text=re.sub(clean1,'',text)
    #6. Remove all the newlines('\n'), tabs('\t'), "-", "\".
    #https://stackoverflow.com/questions/10711116/strip-spaces-tabs-newlines-python
    text= re.sub(r"[\n\t-]*", "", text)

    #text= re.sub('[^A-Za-z0-9]+', ' ',text)
    #Remove all the words which ends with ":".
    #https://stackoverflow.com/questions/2589200/how-can-i-remove-all-words-that-end-in-from
    text= re.sub(r'\w+:\s?',' ',text)
    text= re.sub('[^A-Za-z0-9]+', ' ',text)
    #Decontractions, replace words like below to full words.
    #text=re.sub('[^\w\s]',"",text)
    text = decontracted(text)
  ##########################################################################################
    text = chunking(text)
    text= re.sub("[0-9]+","",text)
    text= re.sub(r"\b_([a-zA-z]+)_\b",r"\1",text) #replace _word_ to word

    text= re.sub(r"\b_([a-zA-z]+)\b",r"\1",text) #replace_word to word
    text= re.sub(r"\b([a-zA-z]+)_\b",r"\1",text) #replace word_ to word
    text= re.sub(r"\b[a-zA-Z]{1}_([a-zA-Z]+)",r"\1",text) #d_berlin to berlin
    text= re.sub(r"\b[a-zA-Z]{2}_([a-zA-Z]+)",r"\1",text) #mr_cat to cat

   #https://gist.github.com/sebleier/554280
    text = ' '.join(e.lower() for e in text.split(' '))
    text= ' '.join(e for e in text.split(' ')  if len(e)>2 and len(e)<15)
    # replace all the words except "A-Za-z_" with space.
    text= re.sub(r"[^a-zA-Z_]"," ",text)
    return text


  from tqdm import tqdm
```

```
a=[]
b=[]
c=[]

for i in tqdm(range(Df.shape[0])):
  a.append(mail_text(Df['text'].values[i]))
  b.append(subject_1(Df['text'].values[i]))
  c.append(preprocess(Df['text'].values[i]))
```

```
100%|██████████| 18828/18828 [25:29<00:00, 12.31it/s]
```

```
Df['preprocessed_text']=c
Df['preprocessed_subject']=b
Df['preprocessed_emails']=a
```

```
Df.iloc[5]
```

```
text                    From: ak333@cleveland.Freenet.Edu (Martin Lins...
class                                                     ms-windows.misc
preprocessed_text       previous article friend mine uses windows most...
preprocessed_subject                              Changing Windows fonts
preprocessed_emails     cleveland Freenet Edu husc8 harvard edu clevel...
Name: 5, dtype: object
```

```
import pickle
##save all your results to disk so that, no need to run all again.
pickle.dump((Df),open('/content/drive/My Drive/Df.pkl','wb'))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-38405aaaec7d> in <module>()
      1 import pickle
      2 ##save all your results to disk so that, no need to run all again.
----> 3 pickle.dump((Df),open('/content/drive/My Drive/Df.pkl','wb'))

NameError: name 'Df' is not defined
```

SEARCH STACK OVERFLOW

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

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

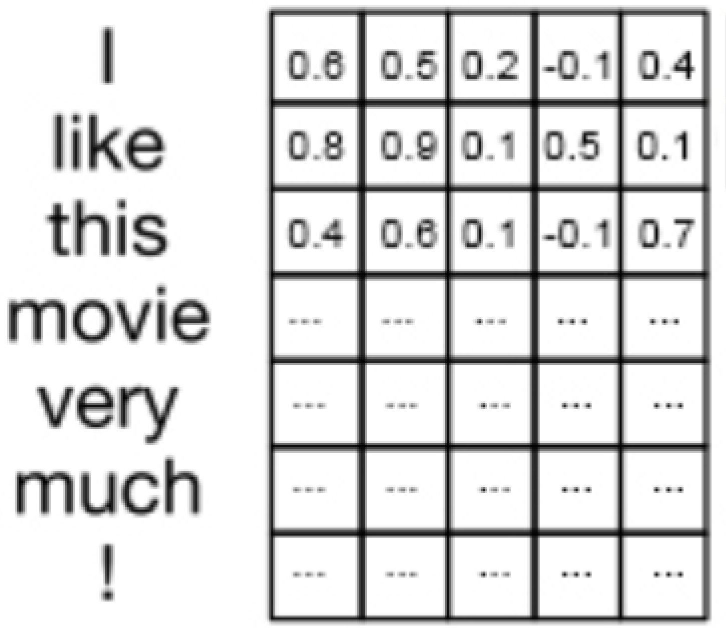
```
import pickle
with open('/content/drive/My Drive/Df.pkl', 'rb') as f:
    Df = pickle.load(f)
```

```
Df.iloc[5]
```

```
4. Do Tokenizer i.e convert text into numbers. please be careful while doing it.
if you are using tf.keras "Tokenizer" API, it removes the "_", but we need that.

5. code the model's ( Model-1, Model-2 ) as discussed below
and try to optimize that models.

6. For every model use predefined Glove vectors.
Don't train any word vectors while Training the model.

7. Use "categorical_crossentropy" as Loss.

8. Use Accuracy and Micro Avgeraged F1 score as your as Key metrics to evaluate your mod

9.  Use Tensorboard to plot the loss and Metrics based on the epoches.

10. Please save your best model weights in to 'best_model_L.h5' ( L = 1 or 2 ).

11. You are free to choose any Activation function, learning rate, optimizer.
But have to use the same architecture which we are giving below.

12. You can add some layer to our architecture but you deletion of layer is not acceptab

13. Try to use Early Stopping technique or any of the callback techniques that you did i

14. For Every model save your model to image ( Plot the model) with shapes
and inlcude those images in the notebook markdown cell,
upload those imgages to Classroom. You can use "plot_model"
please refer this if you don't know how to plot the model with shapes.
```

```
Encoding of the Text  --> For a given text data create a Matrix with Embedding layer as
In the example we have considered d = 5, but in this assignment we will get d = dimensio
 i.e if we have maximum of 350 words in a sentence and embedding of 300 dim word vector,
```

we result in 350*300 dimensional matrix for each sentance as output after embedding lay



Ref: https:///i.imgur.com/kiVQuk1.png

Reference:
https://stackoverflow.com/a/43399308/4084039
https://missinglink.ai/guides/keras/keras-conv1d-working-1d-convolutional-neural-network

[How EMBEDDING LAYER WORKS](#)

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/

```
train_data=Df['preprocessed_emails']+Df['preprocessed_subject']+Df['preprocessed_text']
```

```
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(train_data,Df['class'], test_size=0.25
```

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense,Input,Activation,BatchNormalization,Dropout,Embe
from tensorflow.keras.models import Model
import random as rn
from sklearn.metrics import roc_auc_score
from sklearn.metrics import f1_score
from tensorflow.keras import layers
```

```
text                  From: ak333@cleveland.Freenet.Edu (Martin Lins...
class                                                   ms-windows.misc
preprocessed_text     previous article friend mine uses windows most...
preprocessed_subject                              Changing Windows fonts
preprocessed_emails   cleveland Freenet Edu husc8 harvard edu clevel...
```

```
#text
Df['text'].iloc[0]
```

```
'From: billc@col.hp.com (Bill Claussen)\nSubject: RE:  alt.psychoactives\n\nFYI...I
just posted this on alt.psychoactives as a response to\nwhat the group is fo
r......\n\n\nA note to the users of alt.psychoactives....\n\nThis group was original
ly a takeoff from sci.med.  The reason for\nthe formation of this group was to discu
ss prescription psychoactive\ndrugs....such as antidepressents(tri-cyclics, Prozac,
Lithium,etc),\nantipsychotics(Melleral(sp?), etc), OCD drugs(Anafranil, etc), and\ns
o on and so forth.  It didn't take long for this group to degenerate\ninto a psudo a
lt.drugs atmosphere.  That's to bad, for most of the\nserious folks that wanted to s
```

```
#processed
Df['preprocessed_emails'].iloc[0]
```

```
'col com'
```

```
#SUBJECT
Df['preprocessed_subject'].iloc[0]
```

```
'E alt psychoactives'
```

```
Df['preprocessed_text'].iloc[0]
```

```
'fyi just posted this alt psychoactives response towhat the group for note the users
alt psychoactives this group was originally takeoff from sci med the reason forthe f
ormation this group was discuss prescription such antipsychotics andso and forth did
n take long for this group degenerateinto psudo alt drugs atmosphere that bad for mo
st theserious folks that wanted start this group the first place haveleft and gone b
ack sci med where you have cypher unrelated articles find psychoactive data was also
discuss reallife experiences and side effects ofthe above mentioned well had unsubsc
```

After writing Preprocess function, call the function for each of the document(18828 docs) and then create a dataframe as mentioned above.

Training The models to Classify:

```
1. Combine "preprocessed_text", "preprocessed_subject", "preprocessed_emails" into one c

2. Now Split the data into Train and test. use 25% for test also do a stratify split.

3. Analyze your text data and pad the sequnce if required.
Sequnce length is not restricted, you can use anything of your choice.
you need to give the reasoning
```

```
from tensorflow.keras import layers
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import ModelCheckpoint ,TensorBoard,EarlyStopping,Learning
#from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from tensorflow.keras.layers import concatenate
```

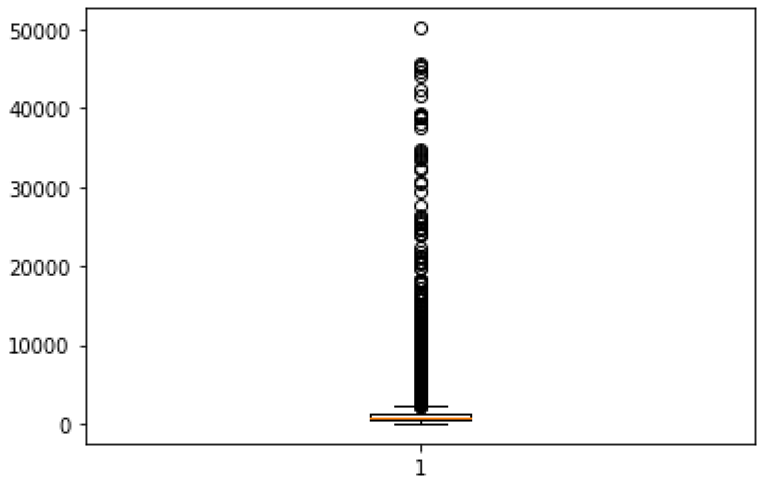
```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
encoder.fit(y_train)
y_train_encoded = encoder.transform(y_train)
y_test_encoded = encoder.transform(y_test)
y_train_ohe = tf.keras.utils.to_categorical(y_train_encoded)
y_test_ohe = tf.keras.utils.to_categorical(y_test_encoded)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

```
print(y_train_ohe.shape)
print(y_test_ohe.shape)
```

```
(14121, 20)
(4707, 20)
```

```
length_of_text=[]
for i in range(X_train.shape[0]):
  length_of_text.append(len(X_train.iloc[i]))
```

```
#box plot of length of text
import matplotlib.pyplot as plt
plt.boxplot(length_of_text)
plt.show()
```




```
#max length
print('max length of text : ',max(length_of_text))
#mean length
import statistics
print('mean length of text : ',statistics.mean(length_of_text) )
# return 50th percentile, e.g median.
import numpy as np
a = np.array(length_of_text)
```

```
a = np.array(length_of_text)
p = np.percentile(a, 90)
print('90th percentile of text :',p)
```

```
max length of text :  50198
mean length of text :  1182.874583952978
90th percentile of text : 2125.0
```

```
#https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer
#https://www.analyticsvidhya.com/blog/2020/03/pretrained-word-embeddings-nlp/
tokenizer=tf.keras.preprocessing.text.Tokenizer(filters='!"#$%&()*+,-./:;<=>?@[\\]`{|}~\t\
tokenizer.fit_on_texts(X_train.tolist())
train_token = tokenizer.texts_to_sequences(X_train)
test_token = tokenizer.texts_to_sequences(X_test)
```

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

```
159015
```

```
# truncate and/or pad input sequences
max_review_length = 2000
X_train_seq = sequence.pad_sequences(train_token, maxlen=max_review_length)
X_test_seq = sequence.pad_sequences(test_token , maxlen=max_review_length)
```

```
import pickle
```

```
XmyCaoEzXYo_tRDwLTsfeA2F3K3j?e=download&authuser=0&nonce=p847le4mhs0ag&user=14084173634687
```

```
--2020-10-03 02:08:58--  https://doc-0o-34-docs.googleusercontent.com/docs/securesc/
Resolving doc-0o-34-docs.googleusercontent.com (doc-0o-34-docs.googleusercontent.com)
Connecting to doc-0o-34-docs.googleusercontent.com (doc-0o-34-docs.googleusercontent
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/octet-stream]
Saving to: ‘glove_vectors’

glove_vectors           [         <=>        ] 121.60M  30.4MB/s    in 4.0s

2020-10-03 02:09:02 (30.4 MB/s) - ‘glove_vectors’ saved [127506004]
```

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

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

# create a weight matrix for words in training docs
```

```
embedding_matrix = np.zeros((size_of_vocabulary, 300))

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

Model-1: Using 1D convolutions with word embeddings

```
1. all are Conv1D layers with any number of filter and filter sizes, there is no restric

2. use concatenate layer is to concatenate all the filters/channels.

3. You can use any pool size and stride for maxpooling layer.

4. Don't use more than 16 filters in one Conv layer becuase it will increase the no of p
( Only recommendation if you have less computing power )

5. You can use any number of layers after the Flatten Layer.
```

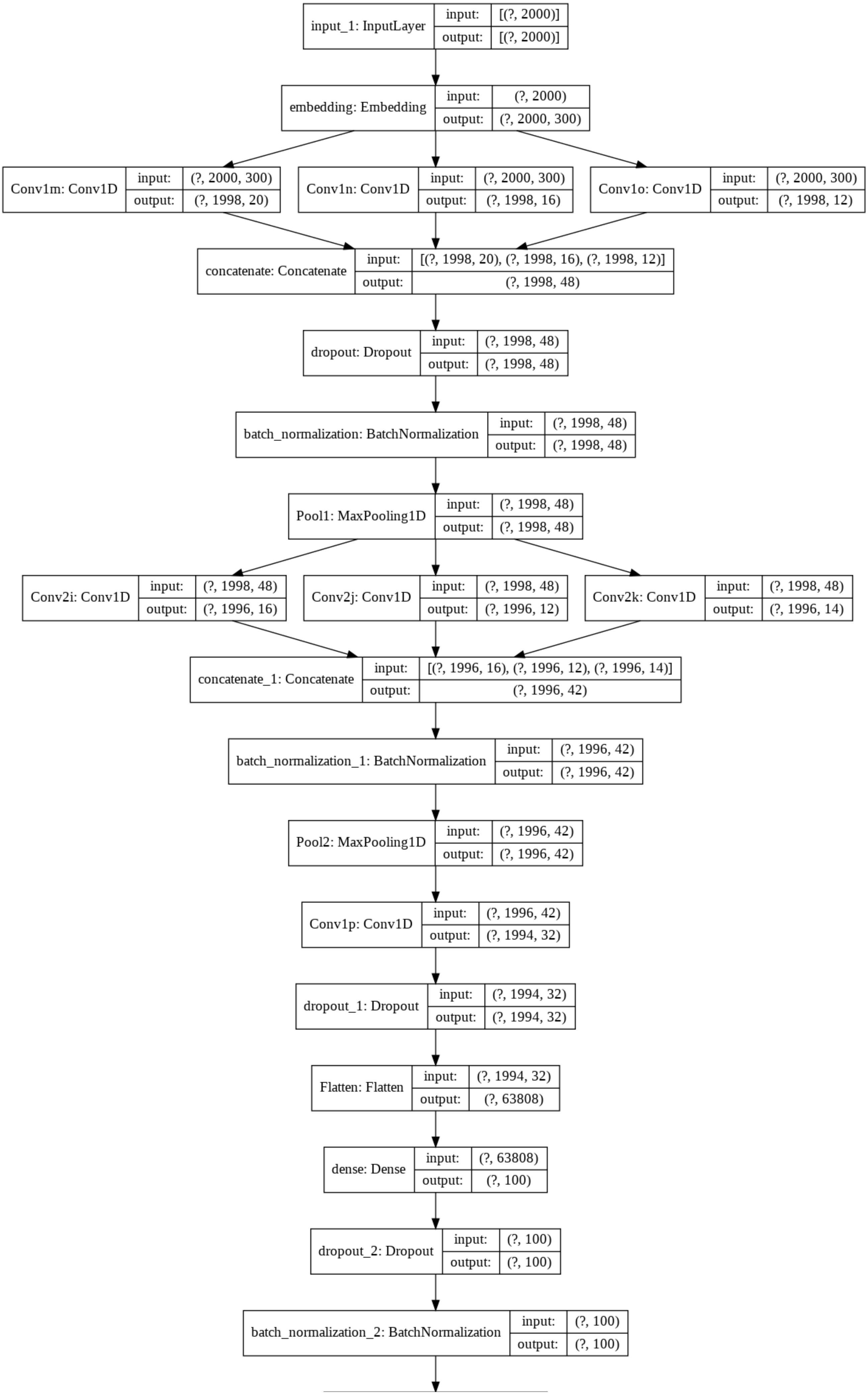
```
tf.keras.backend.clear_session()
#input layer
input = Input(shape=(2000,))
#embedding layer
#embedding layer
embedding = Embedding(size_of_vocabulary,300,weights=[embedding_matrix],input_length=2000,
#Conv Layer
Conv1m = Conv1D(filters=20,kernel_size=3,strides=1,padding='valid',data_format='channels_l
              activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=34
                                              name='Conv1m')(embedding)
#Conv Layer
Conv1n= Conv1D(filters=16,kernel_size=3,strides=1,padding='valid',data_format='channels_la
              activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=35
                                              name='Conv1n')(embedding)
#conv Layer

Conv1o = Conv1D(filters=12,kernel_size=3,strides=1,padding='valid',data_format='channels_l
              activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=36
                                              name='Conv1o')(embedding)
#concatination
concat1 = concatenate([Conv1m,Conv1n,Conv1o])
drop =Dropout(0.15)(concat1)
batch_norm=BatchNormalization()(drop)

#MaxPool Layer
Pool1 = MaxPool1D(pool_size=1,strides=1,padding='valid',data_format='channels_last',name='
```

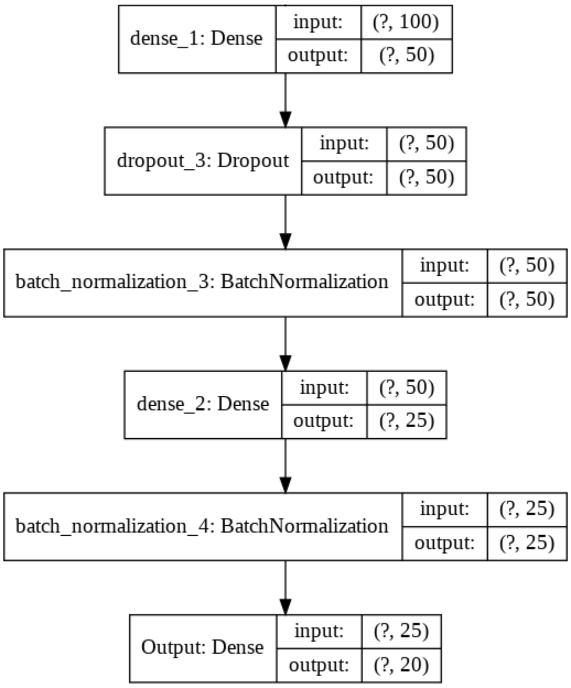
```
#Conv Layer
Conv2i = Conv1D(filters=16,kernel_size=3,strides=1,padding='valid',data_format='channels_l
              activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=30
                                                name='Conv2i')(Pool1)
#Conv Layer
Conv2j= Conv1D(filters=12,kernel_size=3,strides=1,padding='valid',data_format='channels_la
              activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=31
                                                         name='Conv2j')(Pool1)
#conv Layer

Conv2k = Conv1D(filters=14,kernel_size=3,strides=1,padding='valid',data_format='channels_l
              activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=32
                                                              name='Conv2k')(Pool1)

#concatenate

concat2 = concatenate([Conv2i,Conv2j,Conv2k])
#drop=Dropout(0.0)(concat2)

batch_norm = BatchNormalization()(concat2)

#maxpool layer

Pool2 = MaxPool1D(pool_size=1,strides=1,padding='valid',data_format='channels_last',name='

#Conv Layer
Conv3p = Conv1D(filters=32,kernel_size=3,strides=1,padding='valid',data_format='channels_l
              activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=33
                                                                       name='Conv1p')(Pool2

drop1 =Dropout(0.35)(Conv3p)

#Flatten
flatten = Flatten(data_format='channels_last',name='Flatten')(drop1)
#x1 = Dense(8,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=30
#x2 = Dense(12,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=3
#x3 = Dense(16,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=3
#concat3 = concatenate([x1,x2,x3])
# dense layer3
x = Dense(100,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=30
x  =  Dropout(0.25)(x)
x   = BatchNormalization()(x)
x = Dense(50,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=30)
x  =  Dropout(0.35)(x)
x   = BatchNormalization()(x)
x = Dense(25,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=30)
x   = BatchNormalization()(x)
#output layer
Out = Dense(units=20,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_
model11= Model(inputs=input,outputs=Out)



model11.summary()
```

```
Model: "functional_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 2000)]       0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 2000, 300)    47704500    input_1[0][0]
__________________________________________________________________________________________________
Conv1m (Conv1D)                 (None, 1998, 20)     18020       embedding[0][0]
__________________________________________________________________________________________________
Conv1n (Conv1D)                 (None, 1998, 16)     14416       embedding[0][0]
__________________________________________________________________________________________________
Conv1o (Conv1D)                 (None, 1998, 12)     10812       embedding[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 1998, 48)     0           Conv1m[0][0]
                                                                 Conv1n[0][0]
                                                                 Conv1o[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 1998, 48)     0           concatenate[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 1998, 48)     192         dropout[0][0]
__________________________________________________________________________________________________
Pool1 (MaxPooling1D)            (None, 1998, 48)     0           batch_normalization|
__________________________________________________________________________________________________
Conv2i (Conv1D)                 (None, 1996, 16)     2320        Pool1[0][0]
__________________________________________________________________________________________________
Conv2j (Conv1D)                 (None, 1996, 12)     1740        Pool1[0][0]
__________________________________________________________________________________________________
Conv2k (Conv1D)                 (None, 1996, 14)     2030        Pool1[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 1996, 42)     0           Conv2i[0][0]
                                                                 Conv2j[0][0]
                                                                 Conv2k[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 1996, 42)     168         concatenate_1[0][0]
__________________________________________________________________________________________________
Pool2 (MaxPooling1D)            (None, 1996, 42)     0           batch_normalization_
__________________________________________________________________________________________________
Conv1p (Conv1D)                 (None, 1994, 32)     4064        Pool2[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 1994, 32)     0           Conv1p[0][0]
__________________________________________________________________________________________________
Flatten (Flatten)               (None, 63808)        0           dropout_1[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 100)          6380900     Flatten[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 100)          0           dense[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 100)          400         dropout_2[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 50)           5050        batch_normalization_
__________________________________________________________________________________________________
dropout_3 (Dropout)             (None, 50)           0           dense_1[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 50)           200         dropout_3[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 25)           1275        batch_normalization_
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 25)           100         dense_2[0][0]
__________________________________________________________________________________________________
Output (Dense)                  (None, 20)           520         batch_normalization_
```

```
==========================================================================================
Total params: 54,146,707
Trainable params: 6,441,677
Non-trainable params: 47,705,030
```

```
# summarize the model
from tensorflow.keras.utils import plot_model
plot_model(model11, 'model.png', show_shapes=True)
```

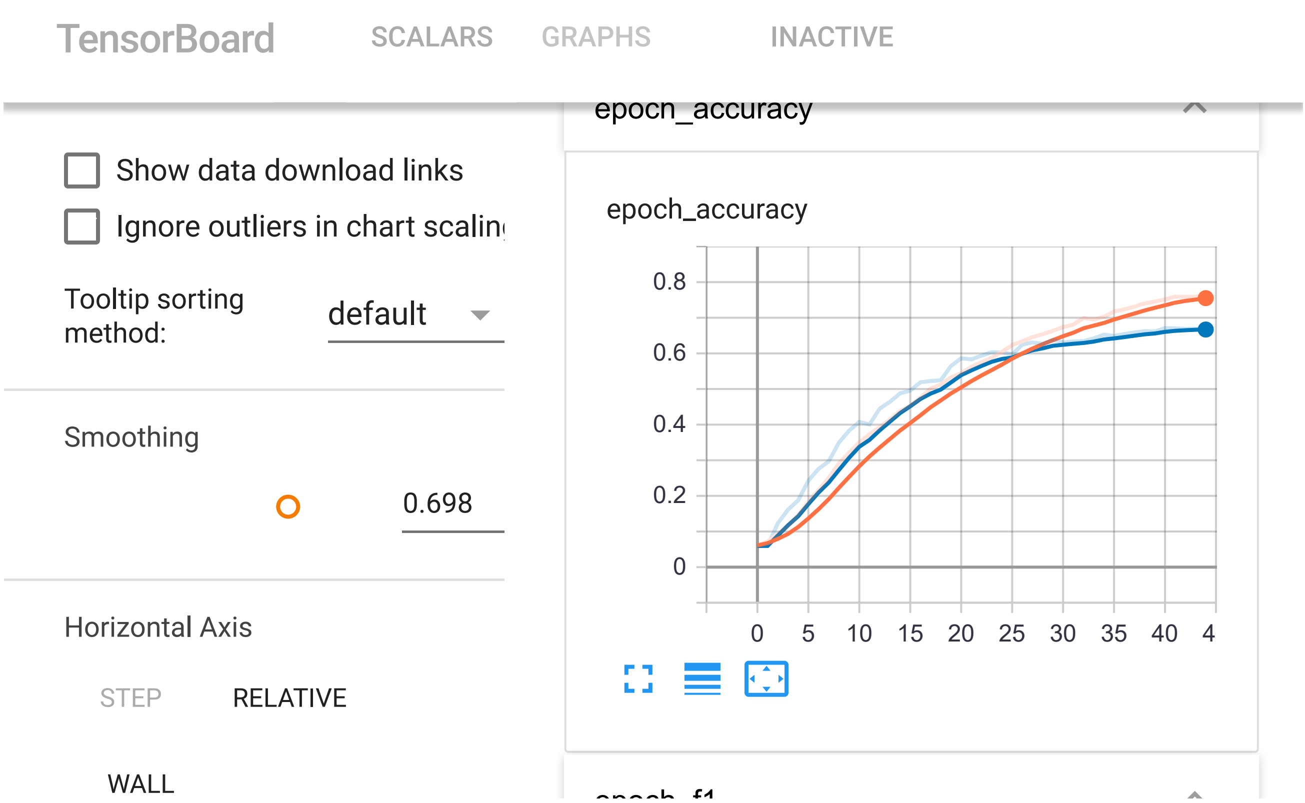

input_1: InputLayer | input: [(?, 2000)] | output: [(?, 2000)]
embedding: Embedding | input: (?, 2000) | output: (?, 2000, 300)
Conv1m: Conv1D | input: (?, 2000, 300) | output: (?, 1998, 20)
Conv1n: Conv1D | input: (?, 2000, 300) | output: (?, 1998, 16)
Conv1o: Conv1D | input: (?, 2000, 300) | output: (?, 1998, 12)
concatenate: Concatenate | input: [(?, 1998, 20), (?, 1998, 16), (?, 1998, 12)] | output: (?, 1998, 48)
dropout: Dropout | input: (?, 1998, 48) | output: (?, 1998, 48)
batch_normalization: BatchNormalization | input: (?, 1998, 48) | output: (?, 1998, 48)
Pool1: MaxPooling1D | input: (?, 1998, 48) | output: (?, 1998, 48)
Conv2i: Conv1D | input: (?, 1998, 48) | output: (?, 1996, 16)
Conv2j: Conv1D | input: (?, 1998, 48) | output: (?, 1996, 12)
Conv2k: Conv1D | input: (?, 1998, 48) | output: (?, 1996, 14)
concatenate_1: Concatenate | input: [(?, 1996, 16), (?, 1996, 12), (?, 1996, 14)] | output: (?, 1996, 42)
batch_normalization_1: BatchNormalization | input: (?, 1996, 42) | output: (?, 1996, 42)
Pool2: MaxPooling1D | input: (?, 1996, 42) | output: (?, 1996, 42)
Conv1p: Conv1D | input: (?, 1996, 42) | output: (?, 1994, 32)
dropout_1: Dropout | input: (?, 1994, 32) | output: (?, 1994, 32)
Flatten: Flatten | input: (?, 1994, 32) | output: (?, 63808)
dense: Dense | input: (?, 63808) | output: (?, 100)
dropout_2: Dropout | input: (?, 100) | output: (?, 100)
batch_normalization_2: BatchNormalization | input: (?, 100) | output: (?, 100)




```
import tensorflow as tf
import keras.backend as K
import os
import datetime

#https://www.kaggle.com/c/liverpool-ion-switching/discussion/132646

def f1(y_true, y_pred):
    y_pred = K.round(y_pred)
    tp = K.sum(K.cast(y_true*y_pred, 'float'), axis=0)
    # tn = K.sum(K.cast((1-y_true)*(1-y_pred), 'float'), axis=0)
    fp = K.sum(K.cast((1-y_true)*y_pred, 'float'), axis=0)
    fn = K.sum(K.cast(y_true*(1-y_pred), 'float'), axis=0)

    p = tp / (tp + fp + K.epsilon())
    r = tp / (tp + fn + K.epsilon())

    f1 = 2*p*r / (p+r+K.epsilon())
    f1 = tf.where(tf.math.is_nan(f1), tf.zeros_like(f1), f1)
    return K.mean(f1)


def changeLearningRate(epochs,learning_rate):
```

```
  if epochs<40:
    learning_rate=0.0001
    return learning_rate
  else :
    learning_rate=0.00001
    return learning_rate



lrschedule = LearningRateScheduler(changeLearningRate)



optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001)
model11.compile(optimizer=optimizer, loss='categorical_crossentropy',metrics=['accuracy',f




#earlystop
earlystop = EarlyStopping(monitor='val_accuracy', min_delta=0.0005, patience=4, verbose=1)
#model 'best_model_L.h5'
filepath="best_model_L1.h5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy',  verbose=1, save_b



%load_ext tensorboard


#tensorbord for model1
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)
%tensorboard --logdir $logdir
```

```
#Conv Layer
Convn = Conv1D(filters=64,kernel_size=5,strides=1,padding='valid',data_format='channels_la
              activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=30
                                                                    kernel_regularizer=l

#MaxPool Layer
Pool1 = MaxPool1D(pool_size=1,strides=1,padding='valid',data_format='channels_last',name='
batch_norm = BatchNormalization()(Pool1)


drop_new2=Dropout(0.25)(batch_norm)

#conv layer
Convk = Conv1D(filters=32,kernel_size=3,strides=1,padding='valid',data_format='channels_la
              activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=30
                                                        kernel_regularizer=l2(0.00001),n

#Conv Layer
Convt = Conv1D(filters=16,kernel_size=1,strides=1,padding='valid',data_format='channels_la
              activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=30
                                                               kernel_regularizer=l2(0.0

#MaxPool Layer
Pool2 = MaxPool1D(pool_size=1,strides=1,padding='valid',data_format='channels_last',name='
batch_norm = BatchNormalization()(Pool2)


drop1 =Dropout(0.25)(batch_norm)

#Flatten
flatten = Flatten(data_format='channels_last',name='Flatten')(drop1)

drop2 =Dropout(0.25)(flatten)

batch_norm = BatchNormalization()(drop2)

# dense layer3
dense = Dense(64,activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed
#output layer
Out = Dense(units=20,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_
model2= Model(inputs=input,outputs=Out)


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



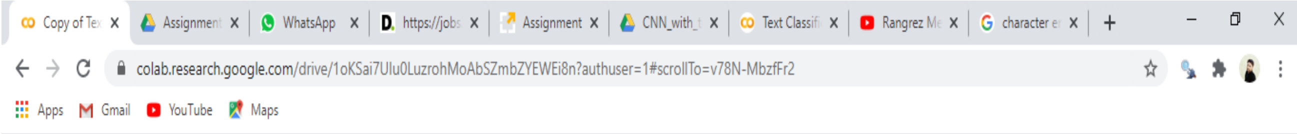

```
model11.fit(X_train_seq,y_train_ohe,epochs=100, validation_data=(X_test_seq,y_test_ohe), b
            callbacks=[checkpoint,tensorboard_callback,earlystop,lrschedule])
```

```
Epoch 1/100
  1/221 [..............................] - ETA: 0s - loss: 3.5602 - accuracy: 0.0781
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
221/221 [==============================] - ETA: 0s - loss: 3.6888 - accuracy: 0.0608
Epoch 00001: val_accuracy improved from -inf to 0.05927, saving model to best_model_l
221/221 [==============================] - 25s 112ms/step - loss: 3.6888 - accuracy:
Epoch 2/100
221/221 [==============================] - ETA: 0s - loss: 3.4988 - accuracy: 0.0719
Epoch 00002: val_accuracy improved from 0.05927 to 0.05991, saving model to best_mode
221/221 [==============================] - 24s 108ms/step - loss: 3.4988 - accuracy:
Epoch 3/100
221/221 [==============================] - ETA: 0s - loss: 3.3490 - accuracy: 0.0934
Epoch 00003: val_accuracy improved from 0.05991 to 0.12301, saving model to best_mode
221/221 [==============================] - 24s 108ms/step - loss: 3.3490 - accuracy:
Epoch 4/100
221/221 [==============================] - ETA: 0s - loss: 3.1847 - accuracy: 0.1145
Epoch 00004: val_accuracy improved from 0.12301 to 0.16104, saving model to best_mode
221/221 [==============================] - 24s 107ms/step - loss: 3.1847 - accuracy:
Epoch 5/100
221/221 [==============================] - ETA: 0s - loss: 3.0034 - accuracy: 0.1469
Epoch 00005: val_accuracy improved from 0.16104 to 0.18759, saving model to best_mode
221/221 [==============================] - 23s 106ms/step - loss: 3.0034 - accuracy:
Epoch 6/100
221/221 [==============================] - ETA: 0s - loss: 2.8518 - accuracy: 0.1829
Epoch 00006: val_accuracy improved from 0.18759 to 0.24283, saving model to best_mode
221/221 [==============================] - 23s 106ms/step - loss: 2.8518 - accuracy:
Epoch 7/100
221/221 [==============================] - ETA: 0s - loss: 2.7004 - accuracy: 0.2158
Epoch 00007: val_accuracy improved from 0.24283 to 0.27576, saving model to best_mode
221/221 [==============================] - 24s 107ms/step - loss: 2.7004 - accuracy:
Epoch 8/100
221/221 [==============================] - ETA: 0s - loss: 2.5778 - accuracy: 0.2508
Epoch 00008: val_accuracy improved from 0.27576 to 0.29700, saving model to best_mode
221/221 [==============================] - 24s 107ms/step - loss: 2.5778 - accuracy:
Epoch 9/100
221/221 [==============================] - ETA: 0s - loss: 2.4449 - accuracy: 0.2919
Epoch 00009: val_accuracy improved from 0.29700 to 0.34948, saving model to best_mode
221/221 [==============================] - 23s 105ms/step - loss: 2.4449 - accuracy:
Epoch 10/100
221/221 [==============================] - ETA: 0s - loss: 2.3374 - accuracy: 0.3216
Epoch 00010: val_accuracy improved from 0.34948 to 0.38326, saving model to best_mode
221/221 [==============================] - 23s 105ms/step - loss: 2.3374 - accuracy:
Epoch 11/100
221/221 [==============================] - ETA: 0s - loss: 2.2458 - accuracy: 0.3500
Epoch 00011: val_accuracy improved from 0.38326 to 0.40769, saving model to best_mode
221/221 [==============================] - 23s 105ms/step - loss: 2.2458 - accuracy:
Epoch 12/100
221/221 [==============================] - ETA: 0s - loss: 2.1660 - accuracy: 0.3731
Epoch 00012: val_accuracy did not improve from 0.40769
221/221 [==============================] - 22s 101ms/step - loss: 2.1660 - accuracy:
Epoch 13/100
221/221 [==============================] - ETA: 0s - loss: 2.0907 - accuracy: 0.3928
Epoch 00013: val_accuracy improved from 0.40769 to 0.44487, saving model to best_mode
221/221 [==============================] - 23s 104ms/step - loss: 2.0907 - accuracy:
Epoch 14/100
221/221 [==============================] - ETA: 0s - loss: 2.0171 - accuracy: 0.4143
Epoch 00014: val_accuracy improved from 0.44487 to 0.46399, saving model to best_mode
221/221 [==============================] - 24s 107ms/step - loss: 2.0171 - accuracy:
Epoch 15/100
221/221 [==============================] - ETA: 0s - loss: 1.9506 - accuracy: 0.4379
```

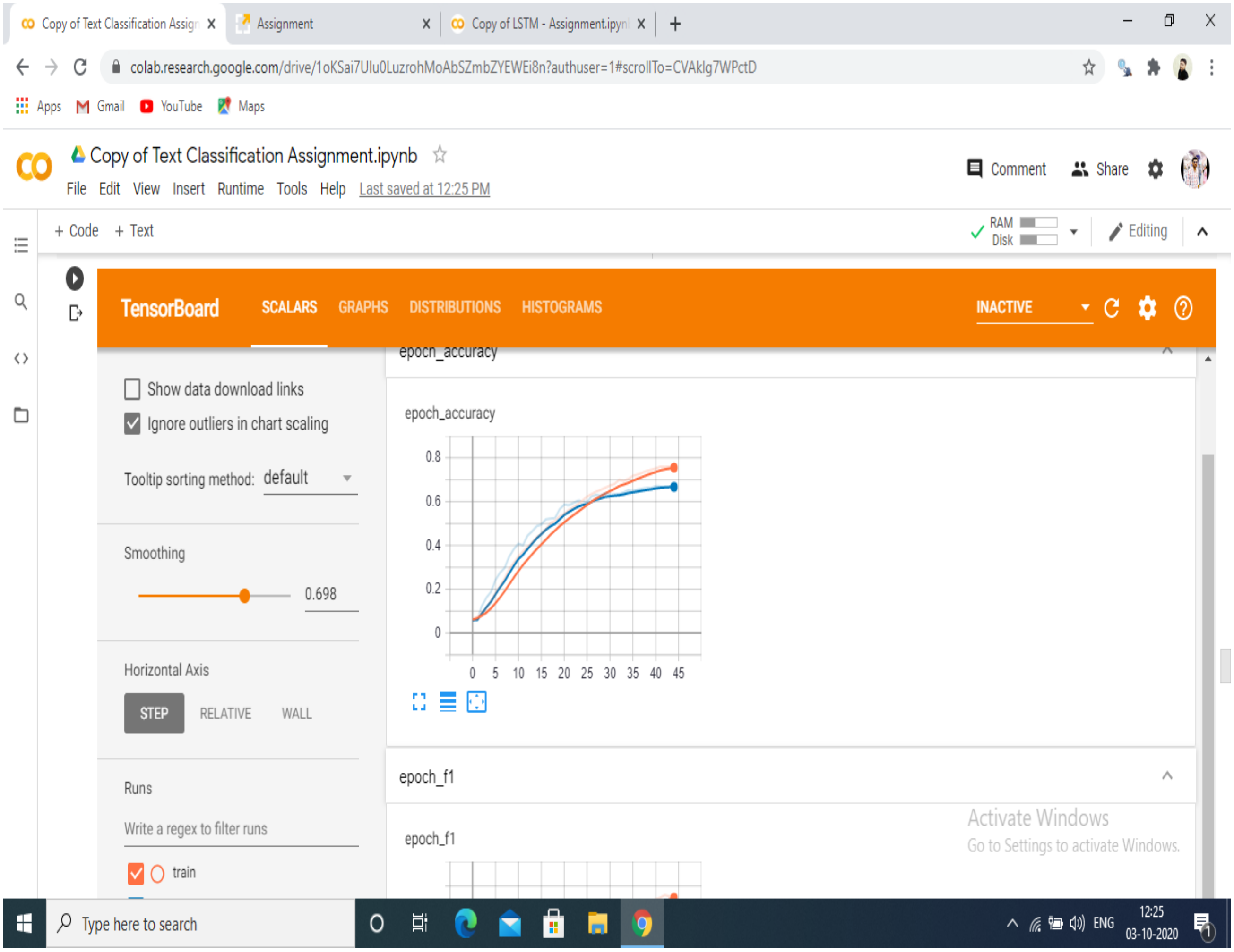
```
Epoch 00015: val_accuracy improved from 0.46399 to 0.48778, saving model to best_mode
221/221 [==============================] - 24s 107ms/step - loss: 1.9506 - accuracy:
Epoch 16/100
221/221 [==============================] - ETA: 0s - loss: 1.8865 - accuracy: 0.4533
Epoch 00016: val_accuracy improved from 0.48778 to 0.49543, saving model to best_mode
221/221 [==============================] - 24s 106ms/step - loss: 1.8865 - accuracy:
Epoch 17/100
221/221 [==============================] - ETA: 0s - loss: 1.8301 - accuracy: 0.4760
Epoch 00017: val_accuracy improved from 0.49543 to 0.51901, saving model to best_mode
221/221 [==============================] - 23s 106ms/step - loss: 1.8301 - accuracy:
Epoch 18/100
221/221 [==============================] - ETA: 0s - loss: 1.7671 - accuracy: 0.5012
Epoch 00018: val_accuracy improved from 0.51901 to 0.52241, saving model to best_mode
221/221 [==============================] - 23s 106ms/step - loss: 1.7671 - accuracy:
Epoch 19/100
221/221 [==============================] - ETA: 0s - loss: 1.7141 - accuracy: 0.5124
Epoch 00019: val_accuracy improved from 0.52241 to 0.52475, saving model to best_mode
221/221 [==============================] - 24s 108ms/step - loss: 1.7141 - accuracy:
Epoch 20/100
221/221 [==============================] - ETA: 0s - loss: 1.6663 - accuracy: 0.5330
Epoch 00020: val_accuracy improved from 0.52475 to 0.56469, saving model to best_mode
221/221 [==============================] - 24s 107ms/step - loss: 1.6663 - accuracy:
Epoch 21/100
221/221 [==============================] - ETA: 0s - loss: 1.6117 - accuracy: 0.5440
Epoch 00021: val_accuracy improved from 0.56469 to 0.58636, saving model to best_mode
221/221 [==============================] - 23s 106ms/step - loss: 1.6117 - accuracy:
Epoch 22/100
221/221 [==============================] - ETA: 0s - loss: 1.5604 - accuracy: 0.5624
Epoch 00022: val_accuracy did not improve from 0.58636
221/221 [==============================] - 23s 102ms/step - loss: 1.5604 - accuracy:
Epoch 23/100
221/221 [==============================] - ETA: 0s - loss: 1.5285 - accuracy: 0.5752
Epoch 00023: val_accuracy improved from 0.58636 to 0.59380, saving model to best_mode
221/221 [==============================] - 23s 105ms/step - loss: 1.5285 - accuracy:
Epoch 24/100
221/221 [==============================] - ETA: 0s - loss: 1.4745 - accuracy: 0.5880
Epoch 00024: val_accuracy improved from 0.59380 to 0.60336, saving model to best_mode
221/221 [==============================] - 23s 106ms/step - loss: 1.4745 - accuracy:
Epoch 25/100
221/221 [==============================] - ETA: 0s - loss: 1.4301 - accuracy: 0.6042
Epoch 00025: val_accuracy did not improve from 0.60336
221/221 [==============================] - 23s 102ms/step - loss: 1.4301 - accuracy:
Epoch 26/100
221/221 [==============================] - ETA: 0s - loss: 1.3754 - accuracy: 0.6233
Epoch 00026: val_accuracy did not improve from 0.60336
221/221 [==============================] - 23s 102ms/step - loss: 1.3754 - accuracy:
Epoch 27/100
221/221 [==============================] - ETA: 0s - loss: 1.3437 - accuracy: 0.6341
Epoch 00027: val_accuracy improved from 0.60336 to 0.62460, saving model to best_mode
221/221 [==============================] - 23s 105ms/step - loss: 1.3437 - accuracy:
Epoch 28/100
221/221 [==============================] - ETA: 0s - loss: 1.3149 - accuracy: 0.6456
Epoch 00028: val_accuracy improved from 0.62460 to 0.63076, saving model to best_mode
221/221 [==============================] - 23s 106ms/step - loss: 1.3149 - accuracy:
Epoch 29/100
221/221 [==============================] - ETA: 0s - loss: 1.2852 - accuracy: 0.6538
Epoch 00029: val_accuracy did not improve from 0.63076
221/221 [==============================] - 23s 102ms/step - loss: 1.2852 - accuracy:
Epoch 30/100
221/221 [==============================] - ETA: 0s - loss: 1.2479 - accuracy: 0.6635
Epoch 00030: val_accuracy improved from 0.63076 to 0.63799, saving model to best_mode
221/221 [==============================] - 23s 105ms/step - loss: 1.2479 - accuracy:
```

```
221/221 [==============================] - 23s 105ms/step - loss: 1.2479 - accuracy:
Epoch 31/100
221/221 [==============================] - ETA: 0s - loss: 1.2255 - accuracy: 0.6737
Epoch 00031: val_accuracy did not improve from 0.63799
221/221 [==============================] - 22s 102ms/step - loss: 1.2255 - accuracy:
Epoch 32/100
221/221 [==============================] - ETA: 0s - loss: 1.1978 - accuracy: 0.6808
Epoch 00032: val_accuracy did not improve from 0.63799
221/221 [==============================] - 23s 102ms/step - loss: 1.1978 - accuracy:
Epoch 33/100
221/221 [==============================] - ETA: 0s - loss: 1.1567 - accuracy: 0.6992
Epoch 00033: val_accuracy did not improve from 0.63799
221/221 [==============================] - 23s 102ms/step - loss: 1.1567 - accuracy:
Epoch 34/100
221/221 [==============================] - ETA: 0s - loss: 1.1519 - accuracy: 0.6951
Epoch 00034: val_accuracy improved from 0.63799 to 0.64202, saving model to best_mode
221/221 [==============================] - 23s 106ms/step - loss: 1.1519 - accuracy:
Epoch 35/100
221/221 [==============================] - ETA: 0s - loss: 1.1224 - accuracy: 0.7025
Epoch 00035: val_accuracy improved from 0.64202 to 0.65264, saving model to best_mode
221/221 [==============================] - 23s 106ms/step - loss: 1.1224 - accuracy:
Epoch 36/100
221/221 [==============================] - ETA: 0s - loss: 1.0776 - accuracy: 0.7176
Epoch 00036: val_accuracy did not improve from 0.65264
221/221 [==============================] - 22s 102ms/step - loss: 1.0776 - accuracy:
Epoch 37/100
221/221 [==============================] - ETA: 0s - loss: 1.0739 - accuracy: 0.7233
Epoch 00037: val_accuracy improved from 0.65264 to 0.65434, saving model to best_mode
221/221 [==============================] - 23s 105ms/step - loss: 1.0739 - accuracy:
Epoch 38/100
221/221 [==============================] - ETA: 0s - loss: 1.0396 - accuracy: 0.7310
Epoch 00038: val_accuracy improved from 0.65434 to 0.65859, saving model to best_mode
```

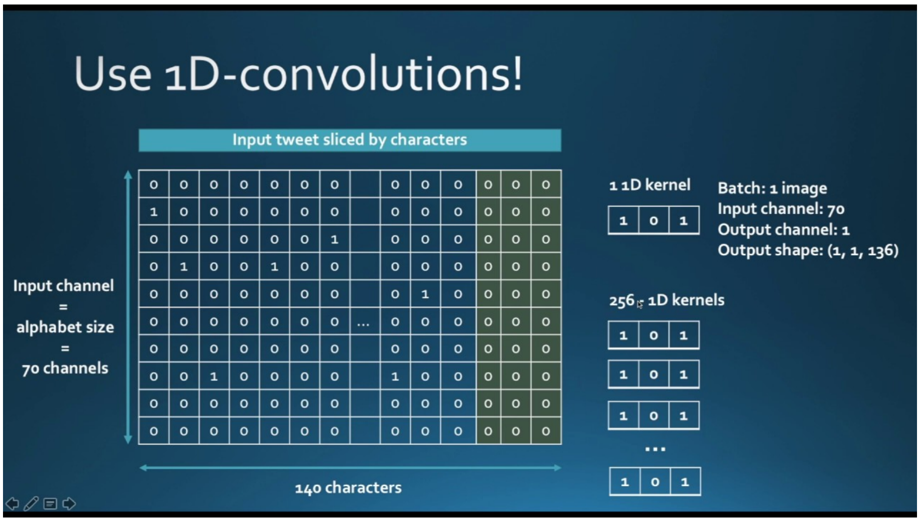
```
from IPython.display import Image
Image('/content/Screenshot (219).png',width=800,height=500)
```




```
from IPython.display import Image
Image('/content/Screenshot (220).png',width=800,height=500)
```



Model-2 : Using 1D convolutions with character embedding



Here are the some papers based on Char-CNN

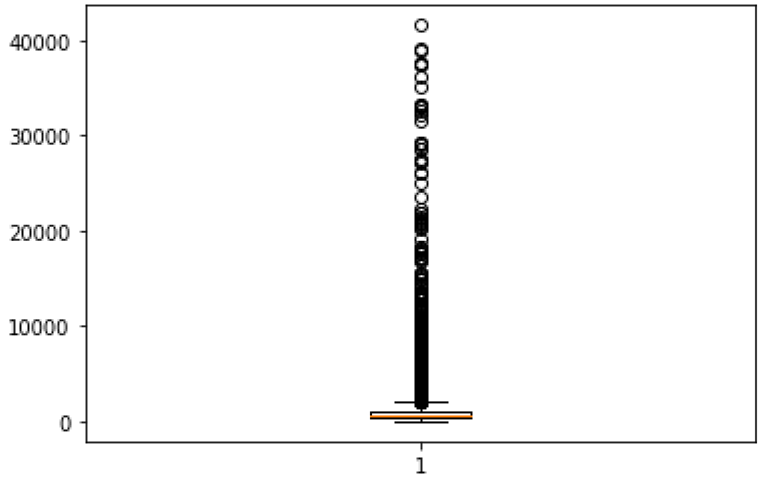
1. Xiang Zhang, Junbo Zhao, Yann LeCun. [Character-level Convolutional Networks for Te]()
2. Yoon Kim, Yacine Jernite, David Sontag, Alexander M. Rush. [Character-Aware Neural]()
3. Shaojie Bai, J. Zico Kolter, Vladlen Koltun. [An Empirical Evaluation of Generic Cc]()
4. Use the pratrained char embeddings [https://github.com/minimaxir/char-embeddings/bl]()

```
import re


def corpus(x):
  x= x.lower()
  x= re.sub(r"[^a-z_]"," ",x)
  x=re.sub(' ','',x)
  return x


X_char=[]
for i in range(X_train.shape[0]):
  X_char.append(corpus(X_train.iloc[i]))


#https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer
#https://www.analyticsvidhya.com/blog/2020/03/pretrained-word-embeddings-nlp/
tokenizer=tf.keras.preprocessing.text.Tokenizer(char_level=True,filters='!"#$%&()*+,-./:;<
tokenizer.fit_on_texts(X_char)
train_token = tokenizer.texts_to_sequences(X_train)
test_token = tokenizer.texts_to_sequences(X_test)


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

```
28
```

```
len_char=[]
```

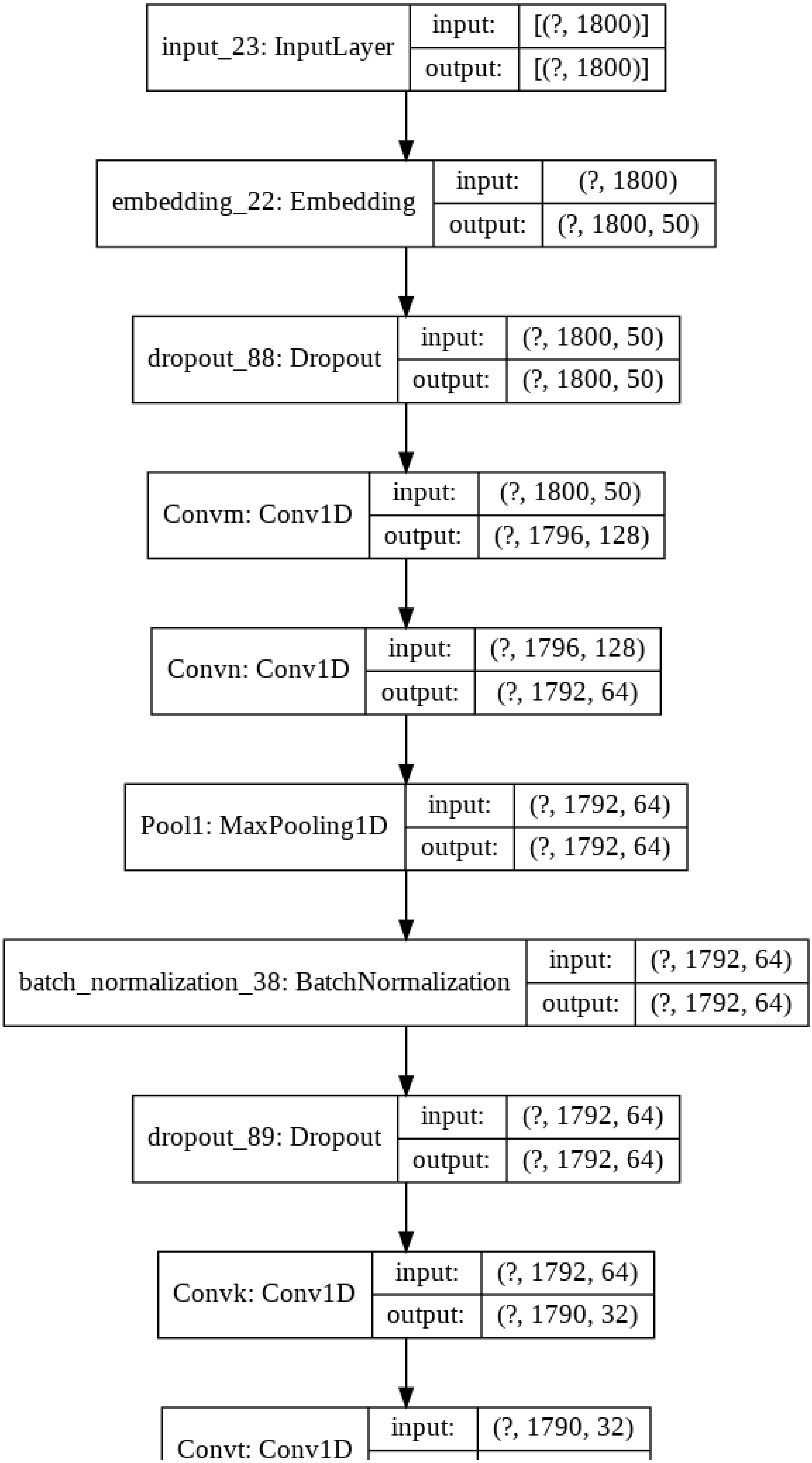
```
for i in range(X_train.shape[0]):
    a=len(re.sub(' ',"",X_train.iloc[i]))
    len_char.append(a)


#box plot of length of text
import matplotlib.pyplot as plt
plt.boxplot(len_char)
plt.show()
```

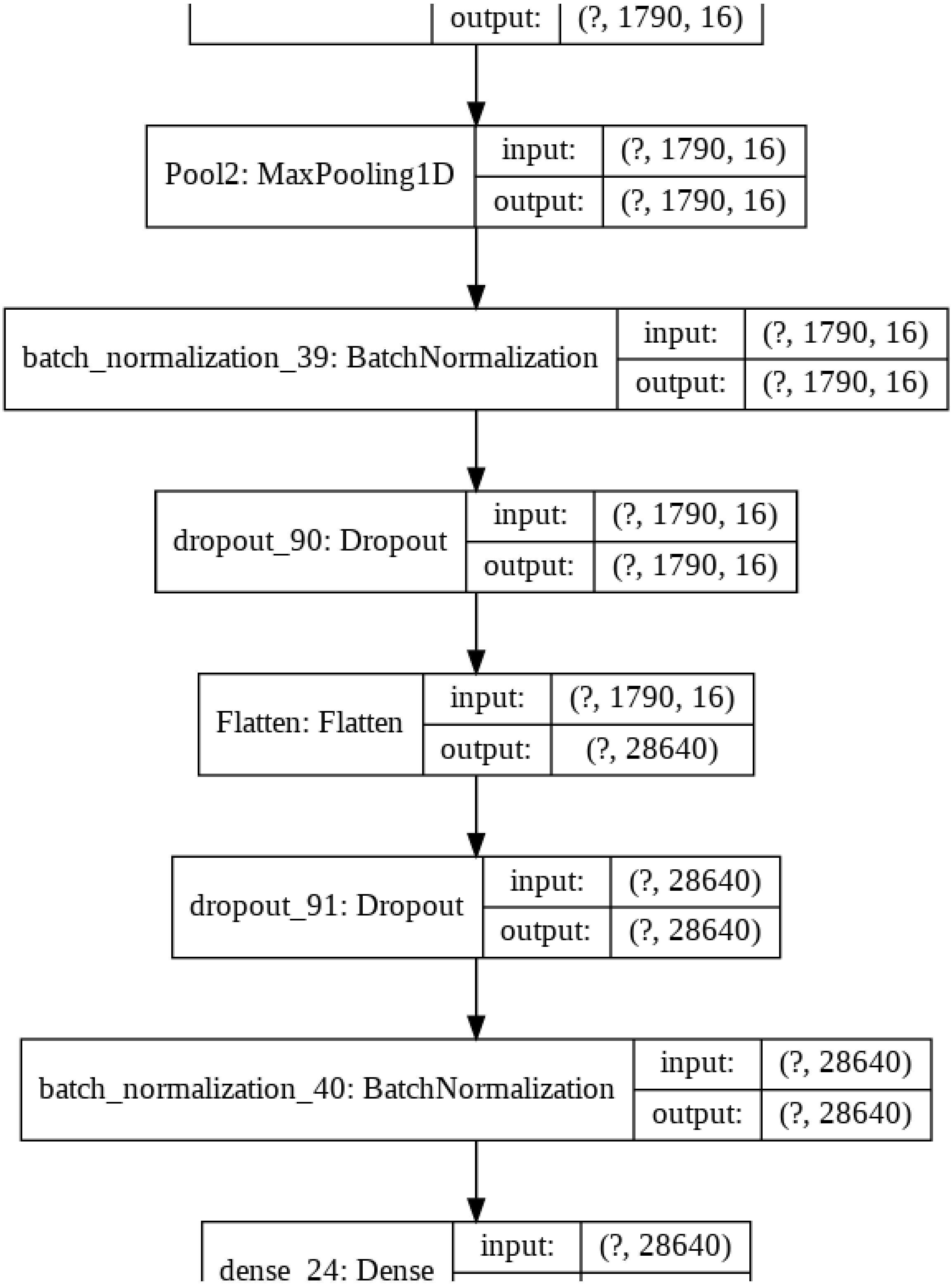



```
#max length
print('max length of text : ',max(len_char))
#mean length
import statistics
print('mean length of text : ',statistics.mean(len_char) )
# return 50th percentile, e.g median.
import numpy as np
a = np.array(len_char)
p = np.percentile(a, 90)
print('90th percentile of text :',p)
```

```
max length of text :  41629
mean length of text :  997.0806600099144
90th percentile of text : 1789.0
```

```
# truncate and/or pad input sequences
max_review_length = 1800
X_train_seq_char = sequence.pad_sequences(train_token, maxlen=max_review_length)
X_test_seq_char = sequence.pad_sequences(test_token , maxlen=max_review_length)


input = Input(shape=(1800,))
Embedding_layer= Embedding(input_dim= 1800,output_dim= 50,embeddings_initializer='uniform'
drop_new1=Dropout(0.1)(Embedding_layer)


#conv layer
Convm = Conv1D(filters=128,kernel_size=5,strides=1,padding='valid',data_format='channels_l
              activation='relu',kernel_initializer=tf.keras.initializers.he_normal(seed=30
                                                                  kernel_regularizer=l2(0.00001),name='Conv
```


input_23: InputLayer
input: [(?, 1800)]
output: [(?, 1800)]
embedding_22: Embedding
input: (?, 1800)
output: (?, 1800, 50)
dropout_88: Dropout
input: (?, 1800, 50)
output: (?, 1800, 50)
Convm: Conv1D
input: (?, 1800, 50)
output: (?, 1796, 128)
Convn: Conv1D
input: (?, 1796, 128)
output: (?, 1792, 64)
Pool1: MaxPooling1D
input: (?, 1792, 64)
output: (?, 1792, 64)
batch_normalization_38: BatchNormalization
input: (?, 1792, 64)
output: (?, 1792, 64)
dropout_89: Dropout
input: (?, 1792, 64)
output: (?, 1792, 64)
Convk: Conv1D
input: (?, 1792, 64)
output: (?, 1790, 32)
Convt: Conv1D
input: (?, 1790, 32)




```
model2.summary()
```

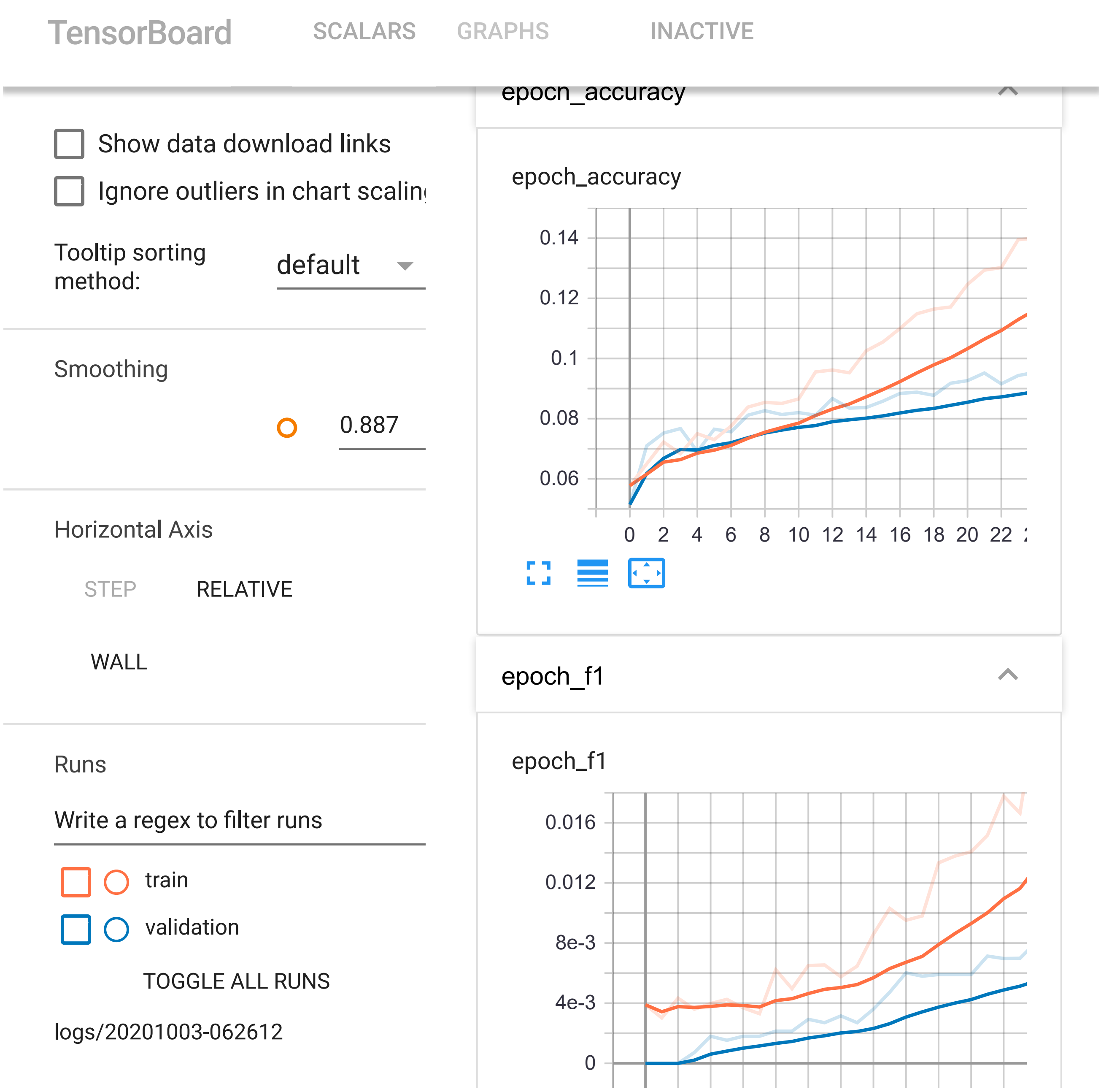
```
Model: "functional_45"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_23 (InputLayer)        [(None, 1800)]            0         
_________________________________________________________________
embedding_22 (Embedding)     (None, 1800, 50)          90000     
_________________________________________________________________
dropout_88 (Dropout)         (None, 1800, 50)          0         
_________________________________________________________________
Convm (Conv1D)               (None, 1796, 128)         32128     
_________________________________________________________________
Convn (Conv1D)               (None, 1792, 64)          41024     
_________________________________________________________________
Pool1 (MaxPooling1D)         (None, 1792, 64)          0         
_________________________________________________________________
batch_normalization_38 (Batc (None, 1792, 64)          256       
_________________________________________________________________
dropout_89 (Dropout)         (None, 1792, 64)          0         
_________________________________________________________________
Convk (Conv1D)               (None, 1790, 32)          6176      
_________________________________________________________________
Convt (Conv1D)               (None, 1790, 16)          528       
_________________________________________________________________
Pool2 (MaxPooling1D)         (None, 1790, 16)          0         
_________________________________________________________________
batch_normalization_39 (Batc (None, 1790, 16)          64        
_________________________________________________________________
dropout_90 (Dropout)         (None, 1790, 16)          0         
_________________________________________________________________
Flatten (Flatten)            (None, 28640)             0         
```

```
optimizer = tf.keras.optimizers.Adam(learning_rate=0.00001)
model2.compile(optimizer=optimizer, loss='categorical_crossentropy',metrics=['accuracy',f1
```

```
dense 24 (Dense)             (None, 64)                1833024   
```

```
#earlystop
earlystop = EarlyStopping(monitor='val_accuracy', min_delta=0.0005, patience=4, verbose=1)
#model 'best_model_L.h5'
filepath="best_model_L2.h5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy',  verbose=1, save_b
```

```
#tensorbord for model1
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)
%tensorboard --logdir $logdir
```

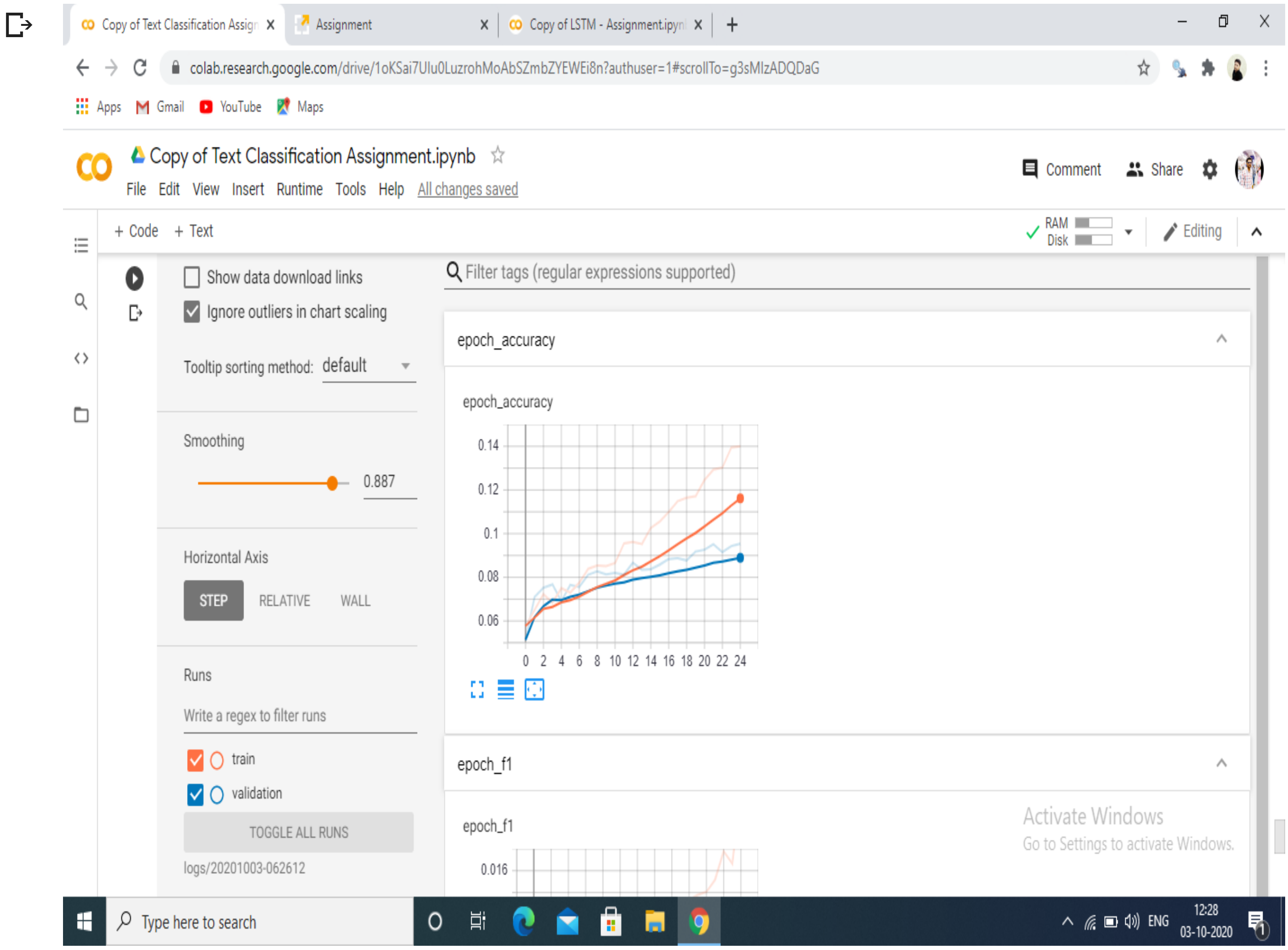



```
model2.fit(X_train_seq_char,y_train_ohe,epochs=25, validation_data=(X_test_seq_char,y_test_
          callbacks=[earlystop,checkpoint,tensorboard_callback])
```

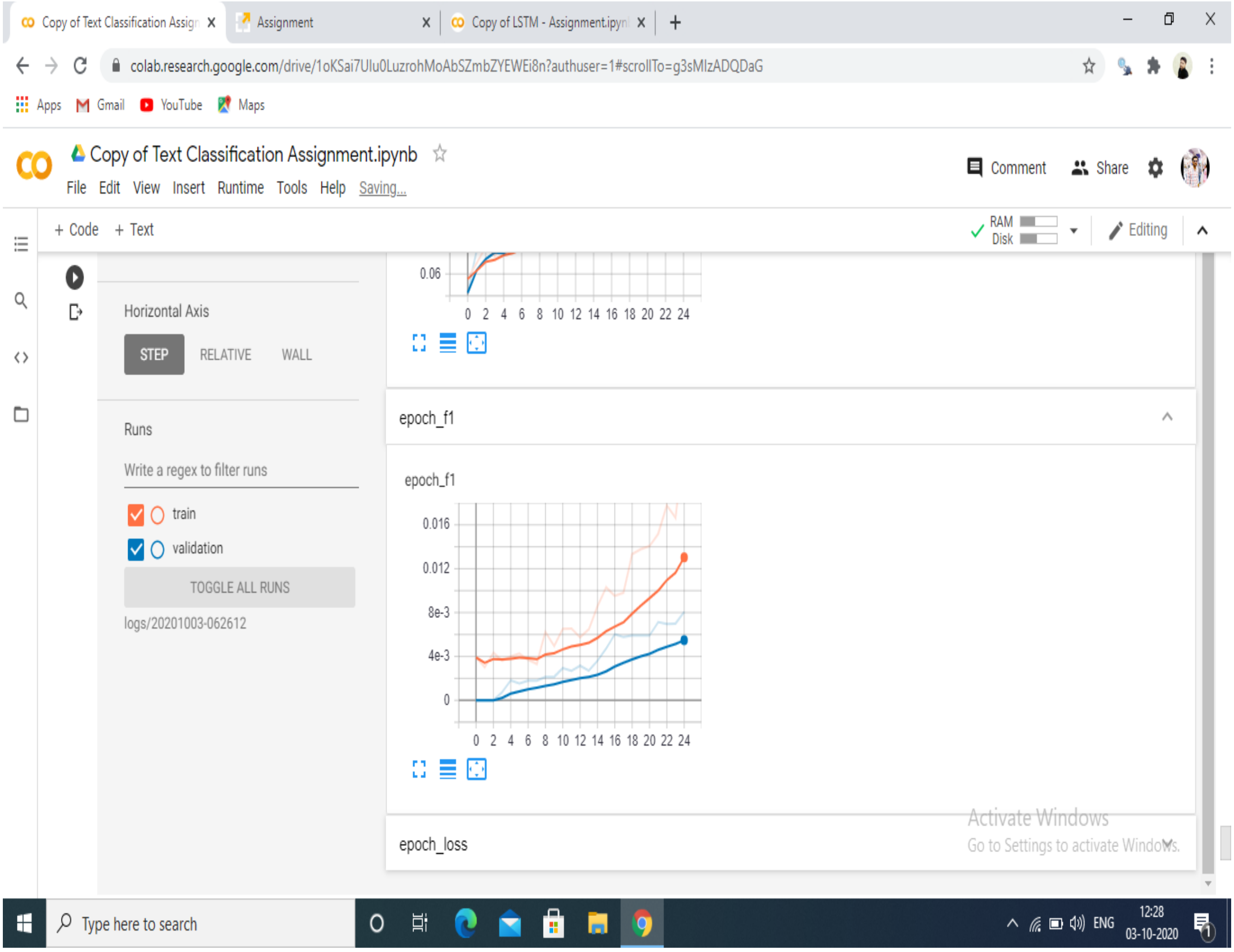
```
Epoch 1/25
  2/221 [..............................] - ETA: 23s - loss: 3.5168 - accuracy: 0.0625
220/221 [============================>.] - ETA: 0s - loss: 3.5003 - accuracy: 0.0577
Epoch 00001: val_accuracy improved from -inf to 0.05141, saving model to best_model_l
221/221 [==============================] - 14s 62ms/step - loss: 3.5005 - accuracy: 0
Epoch 2/25
220/221 [============================>.] - ETA: 0s - loss: 3.3809 - accuracy: 0.0648
Epoch 00002: val_accuracy improved from 0.05141 to 0.07096, saving model to best_mode
221/221 [==============================] - 13s 60ms/step - loss: 3.3809 - accuracy: 0
Epoch 3/25
220/221 [============================>.] - ETA: 0s - loss: 3.3418 - accuracy: 0.0722
Epoch 00003: val_accuracy improved from 0.07096 to 0.07521, saving model to best_mode
221/221 [==============================] - 13s 60ms/step - loss: 3.3419 - accuracy: 0
Epoch 4/25
220/221 [============================>.] - ETA: 0s - loss: 3.2949 - accuracy: 0.0680
Epoch 00004: val_accuracy improved from 0.07521 to 0.07669, saving model to best_mode
221/221 [==============================] - 13s 59ms/step - loss: 3.2938 - accuracy: 0
Epoch 5/25
220/221 [============================>.] - ETA: 0s - loss: 3.2465 - accuracy: 0.0749
Epoch 00005: val_accuracy did not improve from 0.07669
221/221 [==============================] - 13s 59ms/step - loss: 3.2466 - accuracy: 0
Epoch 6/25
220/221 [============================>.] - ETA: 0s - loss: 3.2235 - accuracy: 0.0731
Epoch 00006: val_accuracy did not improve from 0.07669
221/221 [==============================] - 13s 58ms/step - loss: 3.2236 - accuracy: 0
Epoch 7/25
220/221 [============================>.] - ETA: 0s - loss: 3.1912 - accuracy: 0.0777
Epoch 00007: val_accuracy did not improve from 0.07669
221/221 [==============================] - 13s 59ms/step - loss: 3.1910 - accuracy: 0
Epoch 8/25
220/221 [============================>.] - ETA: 0s - loss: 3.1621 - accuracy: 0.0837
Epoch 00008: val_accuracy improved from 0.07669 to 0.08116, saving model to best_mode
221/221 [==============================] - 13s 59ms/step - loss: 3.1623 - accuracy: 0
Epoch 9/25
220/221 [============================>.] - ETA: 0s - loss: 3.1382 - accuracy: 0.0855
Epoch 00009: val_accuracy improved from 0.08116 to 0.08264, saving model to best_mode
221/221 [==============================] - 13s 59ms/step - loss: 3.1377 - accuracy: 0
Epoch 10/25
220/221 [============================>.] - ETA: 0s - loss: 3.1269 - accuracy: 0.0852
Epoch 00010: val_accuracy did not improve from 0.08264
221/221 [==============================] - 13s 58ms/step - loss: 3.1273 - accuracy: 0
Epoch 11/25
220/221 [============================>.] - ETA: 0s - loss: 3.1060 - accuracy: 0.0868
Epoch 00011: val_accuracy did not improve from 0.08264
221/221 [==============================] - 13s 58ms/step - loss: 3.1059 - accuracy: 0
Epoch 12/25
220/221 [============================>.] - ETA: 0s - loss: 3.0671 - accuracy: 0.0958
Epoch 00012: val_accuracy did not improve from 0.08264
221/221 [==============================] - 13s 59ms/step - loss: 3.0672 - accuracy: 0
Epoch 13/25
220/221 [============================>.] - ETA: 0s - loss: 3.0395 - accuracy: 0.0962
Epoch 00013: val_accuracy improved from 0.08264 to 0.08668, saving model to best_mode
221/221 [==============================] - 13s 59ms/step - loss: 3.0397 - accuracy: 0
Epoch 14/25
220/221 [============================>.] - ETA: 0s - loss: 3.0203 - accuracy: 0.0950
Epoch 00014: val_accuracy did not improve from 0.08668
221/221 [==============================] - 13s 58ms/step - loss: 3.0198 - accuracy: 0
Epoch 15/25
220/221 [============================>.] - ETA: 0s - loss: 3.0056 - accuracy: 0.1024
Epoch 00015: val_accuracy did not improve from 0.08668
221/221 [==============================] - 13s 58ms/step - loss: 3.0054 - accuracy: 0
```

```
Epoch 16/25
220/221 [============================>.] - ETA: 0s - loss: 2.9838 - accuracy: 0.1055
Epoch 00016: val_accuracy did not improve from 0.08668
221/221 [==============================] - 13s 58ms/step - loss: 2.9841 - accuracy: 0
Epoch 17/25
220/221 [============================>.] - ETA: 0s - loss: 2.9698 - accuracy: 0.1100
Epoch 00017: val_accuracy improved from 0.08668 to 0.08838, saving model to best_mode
221/221 [==============================] - 13s 59ms/step - loss: 2.9698 - accuracy: 0
Epoch 18/25
220/221 [============================>.] - ETA: 0s - loss: 2.9520 - accuracy: 0.1148
Epoch 00018: val_accuracy improved from 0.08838 to 0.08880, saving model to best_mode
221/221 [==============================] - 13s 58ms/step - loss: 2.9521 - accuracy: 0
Epoch 19/25
220/221 [============================>.] - ETA: 0s - loss: 2.9220 - accuracy: 0.1166
Epoch 00019: val_accuracy did not improve from 0.08880
221/221 [==============================] - 13s 58ms/step - loss: 2.9219 - accuracy: 0
Epoch 20/25
220/221 [============================>.] - ETA: 0s - loss: 2.9113 - accuracy: 0.1168
Epoch 00020: val_accuracy improved from 0.08880 to 0.09178, saving model to best_mode
221/221 [==============================] - 13s 58ms/step - loss: 2.9105 - accuracy: 0
Epoch 21/25
220/221 [============================>.] - ETA: 0s - loss: 2.8864 - accuracy: 0.1249
Epoch 00021: val_accuracy improved from 0.09178 to 0.09263, saving model to best_mode
221/221 [==============================] - 13s 58ms/step - loss: 2.8877 - accuracy: 0
```

```
from IPython.display import Image
Image('/content/Screenshot (221).png',width=800,height=500)
```




```
from IPython.display import Image
Image('/content/Screenshot (222).png',width=800,height=500)
```




```
from prettytable import PrettyTable
ptable = PrettyTable()
ptable.title = " Model Comparision "
ptable.field_names = ["Model1",'Features','train_accuracy','test_acurray']
ptable.add_row(["model1","word_embedding",".76",".67"])
ptable.add_row(["model2","character_embedding","1.4",".095"])

print(ptable)
```

```
+--------+---------------------+----------------+--------------+
| Model1 |       Features      | train_accuracy | test_acurray |
+--------+---------------------+----------------+--------------+
| model1 |    word_embedding   |      .76       |     .67      |
| model2 | character_embedding |      1.4       |     .095     |
+--------+---------------------+----------------+--------------+
```