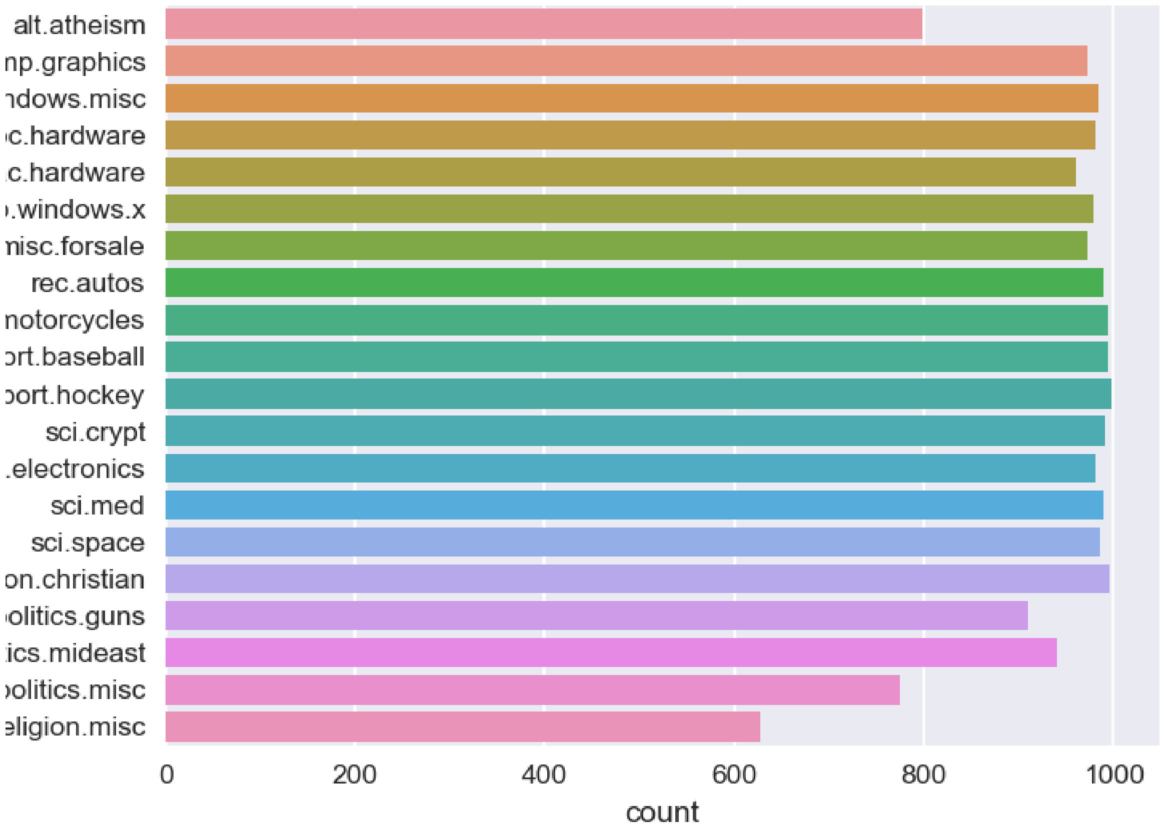
Text Classification:

Data

```
1. we have total of 20 types of documents(Text files) and total 18828 documents(text fi
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'C
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.
```



sample document

Assignment:

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

```
    Extracting  documents/talk.religion.misc_84332.txt                    OK
    Extracting  documents/talk.religion.misc_84334.txt                    OK
    Extracting  documents/talk.religion.misc_84336.txt                    OK
    Extracting  documents/talk.religion.misc_84338.txt                    OK
    Extracting  documents/talk.religion.misc_84339.txt                    OK
    Extracting  documents/talk.religion.misc_84340.txt                    OK
    Extracting  documents/talk.religion.misc_84341.txt                    OK
    Extracting  documents/talk.religion.misc_84342.txt                    OK
    Extracting  documents/talk.religion.misc_84343.txt                    OK
    Extracting  documents/talk.religion.misc_84344.txt                    OK
    Extracting  documents/talk.religion.misc_84345.txt                    OK
    Extracting  documents/talk.religion.misc_84346.txt                    OK
    Extracting  documents/talk.religion.misc_84347.txt                    OK
    Extracting  documents/talk.religion.misc_84348.txt                    OK
    Extracting  documents/talk.religion.misc_84349.txt                    OK
    Extracting  documents/talk.religion.misc_84350.txt                    OK
    Extracting  documents/talk.religion.misc_84351.txt                    OK
    Extracting  documents/talk.religion.misc_84352.txt                    OK
    Extracting  documents/talk.religion.misc_84353.txt                    OK
    Extracting  documents/talk.religion.misc_84354.txt                    OK
    Extracting  documents/talk.religion.misc_84357.txt                    OK
    Extracting  documents/talk.religion.misc_84358.txt                    OK
    Extracting  documents/talk.religion.misc_84360.txt                    OK
    Extracting  documents/talk.religion.misc_84380.txt                    OK
    Extracting  documents/talk.religion.misc_84395.txt                    OK
    Extracting  documents/talk.religion.misc_84396.txt                    OK
    Extracting  documents/talk.religion.misc_84397.txt                    OK
    Extracting  documents/talk.religion.misc_84398.txt                    OK
    Extracting  documents/talk.religion.misc_84399.txt                    OK
    Extracting  documents/talk.religion.misc_84401.txt                    OK
    Extracting  documents/talk.religion.misc_84414.txt                    OK
    Extracting  documents/talk.religion.misc_84422.txt                    OK
    Extracting  documents/talk.religion.misc_84423.txt                    OK
    Extracting  documents/talk.religion.misc_84428.txt                    OK
    Extracting  documents/talk.religion.misc_84429.txt                    OK
    Extracting  documents/talk.religion.misc_84430.txt                    OK
    Extracting  documents/talk.religion.misc_84431.txt                    OK
    Extracting  documents/talk.religion.misc_84433.txt                    OK
    Extracting  documents/talk.religion.misc_84434.txt                    OK
    Extracting  documents/talk.religion.misc_84435.txt                    OK
    Extracting  documents/talk.religion.misc_84436.txt                    OK
    Extracting  documents/talk.religion.misc_84437.txt                    OK
    Extracting  documents/talk.religion.misc_84438.txt                    OK
    Extracting  documents/talk.religion.misc_84439.txt                    OK
    Extracting  documents/talk.religion.misc_84440.txt                    OK
    Extracting  documents/talk.religion.misc_84441.txt                    OK
    Extracting  documents/talk.religion.misc_84442.txt                    OK
    Extracting  documents/talk.religion.misc_84443.txt                    OK
    Extracting  documents/talk.religion.misc_84444.txt                    OK
    Extracting  documents/talk.religion.misc_84445.txt                    OK
    Extracting  documents/talk.religion.misc_84446.txt                    OK
    Extracting  documents/talk.religion.misc_84447.txt                    OK
    Extracting  documents/talk.religion.misc_84448.txt                    OK
    Extracting  documents/talk.religion.misc_84449.txt                    OK
    Extracting  documents/talk.religion.misc_84450.txt                    OK
    Extracting  documents/talk.religion.misc_84451.txt                    OK
    Extracting  documents/talk.religion.misc_84452.txt                    OK
    Extracting  documents/talk.religion.misc_84506.txt                    OK
    Extracting  documents/talk.religion.misc_84507.txt                    OK
```

```
Extracting  documents/talk.religion.misc_84507.txt                    OK
```

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
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,
append all those into one list/array. ( This will give length of 18828 sentences i.e on
Some sample output was shown below.

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

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

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-

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

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('
```

We did chunking for above two lines and then We got one list where each word is mapped
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 "S
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('/documents.rar')
text =[]
Class=[]
for f in files:
  name=str(f).split('_')[0]
  Class.append(name.split('.')[-2]+'.'+name.split('.')[-1])
  with open('/documents.rar'+str(f),'r',encoding="ISO-8859-1") as f1:
    my_lines = f1.read()
  text.append(my_lines)
```

```
import pandas as pd
data=pd.DataFrame()
data['text']=text
data['class']=Class
data.head()
data.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
4	From: kardank@ERE.UMontreal.CA (Kardan Kaveh)\...	comp.graphics

```
def mail_text(text):
  h=[]
  #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('.')
    h.extend(d)
  return ' '.join([w for w in h if len(w)>2])
```

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

  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 ":".

  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)

  text= re.sub(r"\b_([a-zA-z]+)\b",r"\1",text)
  text= re.sub(r"\b([a-zA-z]+)_\b",r"\1",text)
  text= re.sub(r"\b[a-zA-Z]{1}_([a-zA-Z]+)",r"\1",text)
  text= re.sub(r"\b[a-zA-Z]{2}_([a-zA-Z]+)",r"\1",text)


  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)
  text=   .join(e for e in text.split(   ) if len(e)>2 and len(e)<15)

  # replace all the words with space except "A-Za-z_"
  text= re.sub(r"[^a-zA-Z_]"," ",text)
  return text
```

```
from tqdm import tqdm
a=[]
b=[]
c=[]

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

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

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

```
data.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
pickle.dump((data),open('/Df.pkl','wb'))
```

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

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

```
import pickle
with open('/Df.pkl', 'rb') as f:
    data = pickle.load(f)
```

```
data.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
```

```
#text
data['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 origina
    lly a takeoff from sci.med.  The reason for\nthe formation of this group was to dis
    cuss prescription psychoactive\ndrugs....such as antidepressents(tri-cyclics, Proza
    c, Lithium,etc),\nantipsychotics(Melleral(sp?), etc), OCD drugs(Anafranil, etc), an
    d\nso on and so forth.  It didn't take long for this group to degenerate\ninto a ps
    udo alt.drugs atmosphere.  That's to bad, for most of the\nserious folks that wante
    d to start this group in the first place have\nleft and gone back to sci.med, where
    you have to cypher through\nhundreds of unrelated articles to find psychoactive dat
```

```
#preprocessed_emails:
data['preprocessed_emails'].iloc[0]
```

```
    'col com'
```

```
#preprocessed_subject
data['preprocessed_subject'].iloc[0]
```

```
    'E alt psychoactives'
```

```
#preprocessed_text
data['preprocessed_text'].iloc[0]
```

```
    'fyi just posted this alt psychoactives response towhat the group for note the user
    s alt psychoactives this group was originally takeoff from sci med the reason forth
    e formation this group was discuss prescription such antipsychotics andso and forth
    didn take long for this group degenerateinto psudo alt drugs atmosphere that bad fo
    r most theserious folks that wanted start this group the first place haveleft and g
    one back sci med where you have cypher unrelated articles find psychoactive data wa
    s also discuss reallife experiences and side effects ofthe above mentioned well had
    unsubscribed this group for some time and decidedto check today see anything had ch
    anged nope same oldnine ten crap articles that this group was never intended for th
```

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

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

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 mo

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.

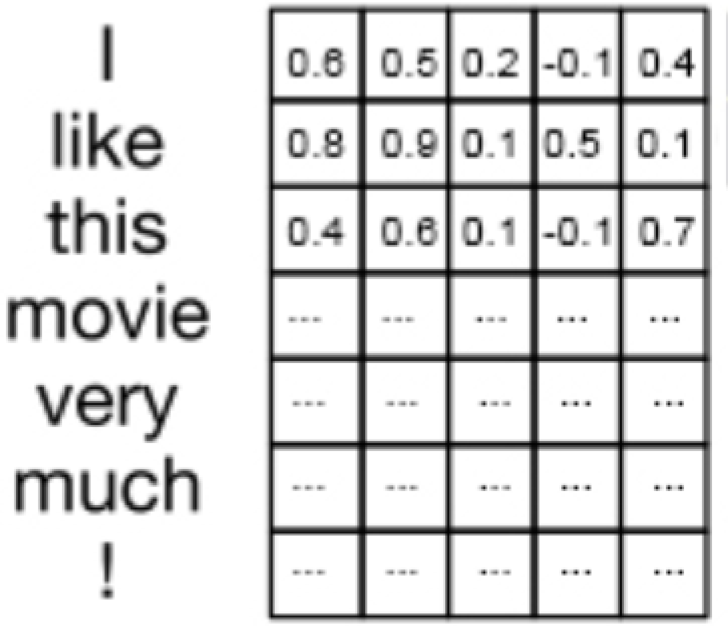
12. You can add some layer to our architecture but you **deletion** of layer is not acceptal

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

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 = dimensi
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 la
```



I	0.6	0.5	0.2	-0.1	0.4
like	0.8	0.9	0.1	0.5	0.1
this	0.4	0.6	0.1	-0.1	0.7
movie	...	...	...	...	...
very	...	...	...	...	...
much	...	...	...	...	...
!	...	...	...	...	...

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-networ

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=data['preprocessed_emails']+data['preprocessed_subject']+data['preprocessed_te
```

```
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(train_data,data['class'], test_size=0.
```

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

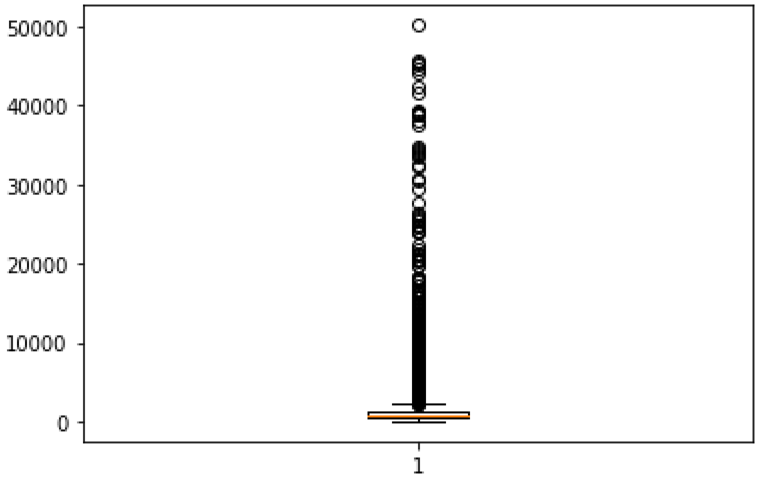
```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
encoder.fit(y_train)
y_train_encoded = encoder.transform(y_train)
y_test_encoded = encoder.transform(y_test)
y_train_ohe = tf.keras.utils.to_categorical(y_train_encoded)
y_test_ohe = tf.keras.utils.to_categorical(y_test_encoded)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

```
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)
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

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
```

```
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
```

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

```
--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]
```

```
# Load the glove vectors:
with open('/glove_vectors', 'rb') as f:
    glove_words= pickle.load(f)
```

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

embedding_matrix = np.zeros((size_of_vocabulary, 300)) # creating weight matrix for words:

for word, j in tokenizer.word_index.items():
    embedding_vector = glove_words.get(word)
```

```
        if embedding_vector is not None:
            embedding_matrix[j] = 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 restri

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 
( Only recommendation if you have less computing power )

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

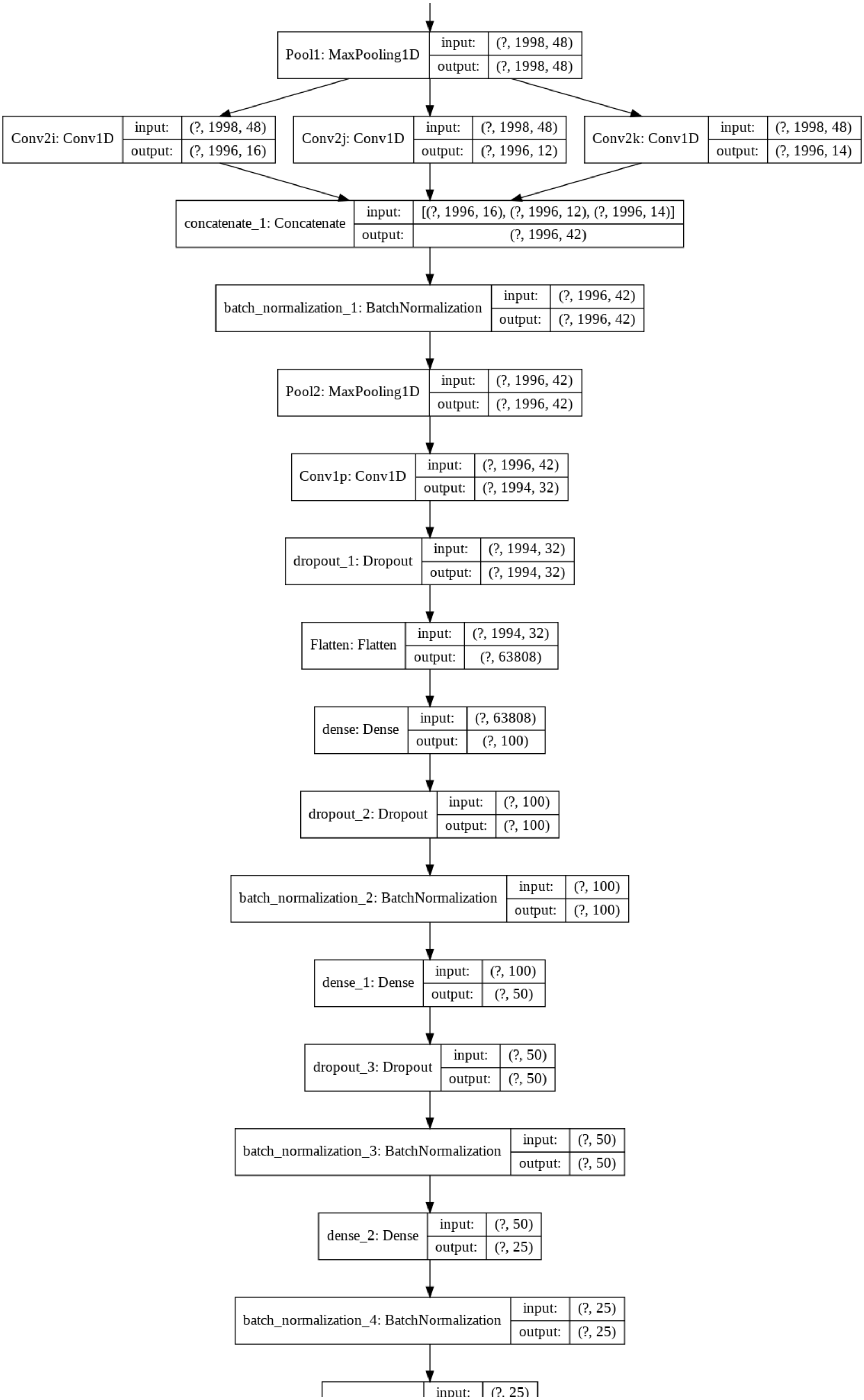
```
tf.keras.backend.clear_session()
#input layer
input = Input(shape=(2000,))

#embedding layer
embedding = 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

#now concatenate
concat2 = concatenate([Conv2i,Conv2j,Conv2k])

#drop=Dropout(0.0)(concat2)
batch_norm = BatchNormalization()(concat2)

#maxpool layer
MaxPool2 = MaxPool1D(pool_size=1,strides=1,padding='valid',data_format='channels_last',nam

#Conv Layer
Conv3p = Conv1D(filters=32,kernel_size=3,strides=1,padding='valid',data_format='channels_l

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()
```

```
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]
```

```
                                                                   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_normalizati

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][

Pool2 (MaxPooling1D)           (None, 1996, 42)     0           batch_normalizati

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_normalizati

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_normalizati

batch_normalization_4 (BatchNor (None, 25)          100         dense_2[0][0]

Output (Dense)                 (None, 20)           520         batch_normalizati
==================================================================================
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)
```


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)
dense_1: Dense
input: (?, 100)
output: (?, 50)
dropout_3: Dropout
input: (?, 50)
output: (?, 50)
batch_normalization_3: BatchNormalization
input: (?, 50)
output: (?, 50)
dense_2: Dense
input: (?, 50)
output: (?, 25)
batch_normalization_4: BatchNormalization
input: (?, 25)
output: (?, 25)
input: (?, 25)

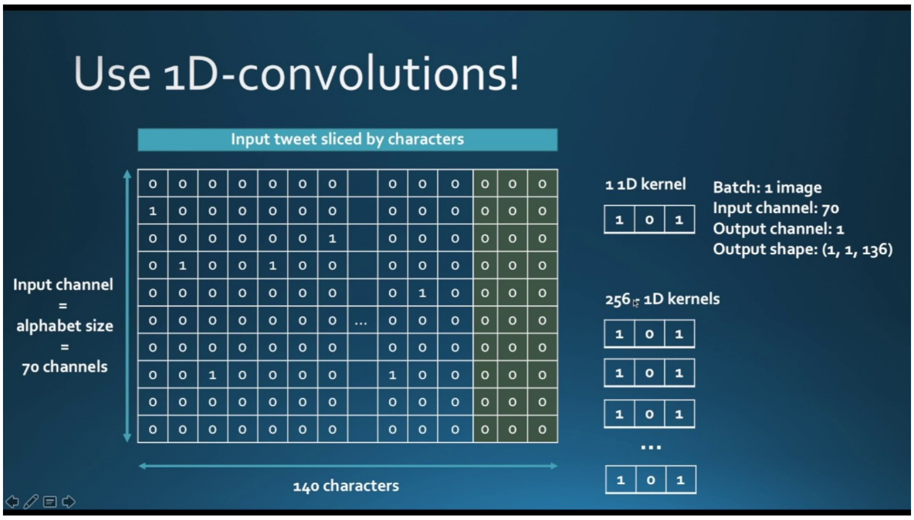
```
import tensorflow as tf
import keras.backend as K
import os
import datetime

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



    from keras.callbacks import TensorBoard
    model11.fit(X_train_seq,y_train_ohe,epochs=100, validation_data=(X_test_seq,y_test_ohe), b
```

```
     Epoch 32/100
     221/221 [==============================] - ETA: 0s - loss: 1.1978 - accuracy: 0.68
     Epoch 00032: val_accuracy did not improve from 0.63799
     221/221 [==============================] - 23s 102ms/step - loss: 1.1978 - accurac
     Epoch 33/100
     221/221 [==============================] - ETA: 0s - loss: 1.1567 - accuracy: 0.69
     Epoch 00033: val_accuracy did not improve from 0.63799
     221/221 [==============================] - 23s 102ms/step - loss: 1.1567 - accurac
     Epoch 34/100
     221/221 [==============================] - ETA: 0s - loss: 1.1519 - accuracy: 0.69
     Epoch 00034: val_accuracy improved from 0.63799 to 0.64202, saving model to best_m
     221/221 [==============================] - 23s 106ms/step - loss: 1.1519 - accurac
     Epoch 35/100
     221/221 [==============================] - ETA: 0s - loss: 1.1224 - accuracy: 0.70
     Epoch 00035: val_accuracy improved from 0.64202 to 0.65264, saving model to best_m
     221/221 [==============================] - 23s 106ms/step - loss: 1.1224 - accurac
     Epoch 36/100
     221/221 [==============================] - ETA: 0s - loss: 1.0776 - accuracy: 0.71
     Epoch 00036: val_accuracy did not improve from 0.65264
     221/221 [==============================] - 22s 102ms/step - loss: 1.0776 - accurac
     Epoch 37/100
     221/221 [==============================] - ETA: 0s - loss: 1.0739 - accuracy: 0.72
     Epoch 00037: val_accuracy improved from 0.65264 to 0.65434, saving model to best_m
```

```
221/221 [==============================] - 23s 105ms/step - loss: 1.0739 - accurac
Epoch 38/100
221/221 [==============================] - ETA: 0s - loss: 1.0396 - accuracy: 0.73
Epoch 00038: val_accuracy improved from 0.65434 to 0.65859, saving model to best_m
221/221 [==============================] - 23s 105ms/step - loss: 1.0396 - accurac
Epoch 39/100
221/221 [==============================] - ETA: 0s - loss: 1.0140 - accuracy: 0.74
Epoch 00039: val_accuracy improved from 0.65859 to 0.66242, saving model to best_m
221/221 [==============================] - 24s 106ms/step - loss: 1.0140 - accurac
Epoch 40/100
221/221 [==============================] - ETA: 0s - loss: 1.0035 - accuracy: 0.74
Epoch 00040: val_accuracy did not improve from 0.66242
221/221 [==============================] - 22s 102ms/step - loss: 1.0035 - accurac
Epoch 41/100
221/221 [==============================] - ETA: 0s - loss: 0.9763 - accuracy: 0.75
Epoch 00041: val_accuracy improved from 0.66242 to 0.67092, saving model to best_m
221/221 [==============================] - 23s 105ms/step - loss: 0.9763 - accurac
Epoch 42/100
221/221 [==============================] - ETA: 0s - loss: 0.9560 - accuracy: 0.75
Epoch 00042: val_accuracy did not improve from 0.67092
221/221 [==============================] - 23s 102ms/step - loss: 0.9560 - accurac
Epoch 43/100
221/221 [==============================] - ETA: 0s - loss: 0.9477 - accuracy: 0.75
Epoch 00043: val_accuracy did not improve from 0.67092
221/221 [==============================] - 22s 102ms/step - loss: 0.9477 - accurac
Epoch 44/100
221/221 [==============================] - ETA: 0s - loss: 0.9562 - accuracy: 0.75
Epoch 00044: val_accuracy did not improve from 0.67092
221/221 [==============================] - 22s 102ms/step - loss: 0.9562 - accurac
Epoch 45/100
221/221 [==============================] - ETA: 0s - loss: 0.9419 - accuracy: 0.76
Epoch 00045: val_accuracy did not improve from 0.67092
221/221 [==============================] - 22s 102ms/step - loss: 0.9419 - accurac
Epoch 00045: early stopping
<tensorflow.python.keras.callbacks.History at 0x7f34b4ef2630>
```

Model-2 : Using 1D convolutions with character embedding



Use 1D-convolutions!

Input tweet sliced by characters

0	0	0	0	0	0	0		0	0	0	0	0	0
1	0	0	0	0	0	0		0	0	0	0	0	0
0	0	0	0	0	0	1		0	0	0	0	0	0
0	1	0	0	1	0	0		0	0	0	0	0	0
0	0	0	0	0	0	0		0	1	0	0	0	0
0	0	0	0	0	0	0	...	0	0	0	0	0	0
0	0	0	0	0	0	0		0	0	0	0	0	0
0	0	1	0	0	0	0		1	0	0	0	0	0
0	0	0	0	0	0	0		0	0	0	0	0	0
0	0	0	0	0	0	0		0	0	0	0	0	0

Input channel = alphabet size = 70 channels

140 characters

1 1D kernel

1	0	1

Batch: 1 image
Input channel: 70
Output channel: 1
Output shape: (1, 1, 136)

256 1D kernels

1	0	1
1	0	1
1	0	1
...		
1	0	1

```
Here are the some papers based on Char-CNN
```

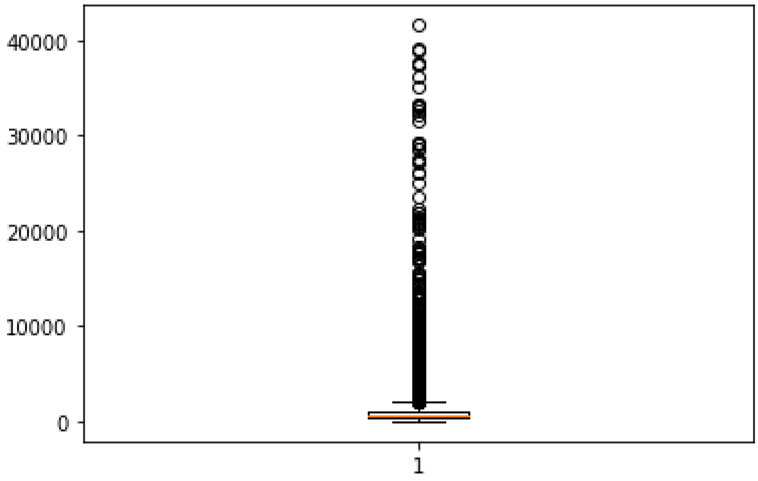
1. Xiang Zhang, Junbo Zhao, Yann LeCun. Character-level Convolutional Networks for T
2. Yoon Kim, Yacine Jernite, David Sontag, Alexander M. Rush. Character-Aware Neural
3. Shaojie Bai, J. Zico Kolter, Vladlen Koltun. An Empirical Evaluation of Generic C
4. Use the pratrained char embeddings https://github.com/minimaxir/char-embeddings/b

```
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

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)
```

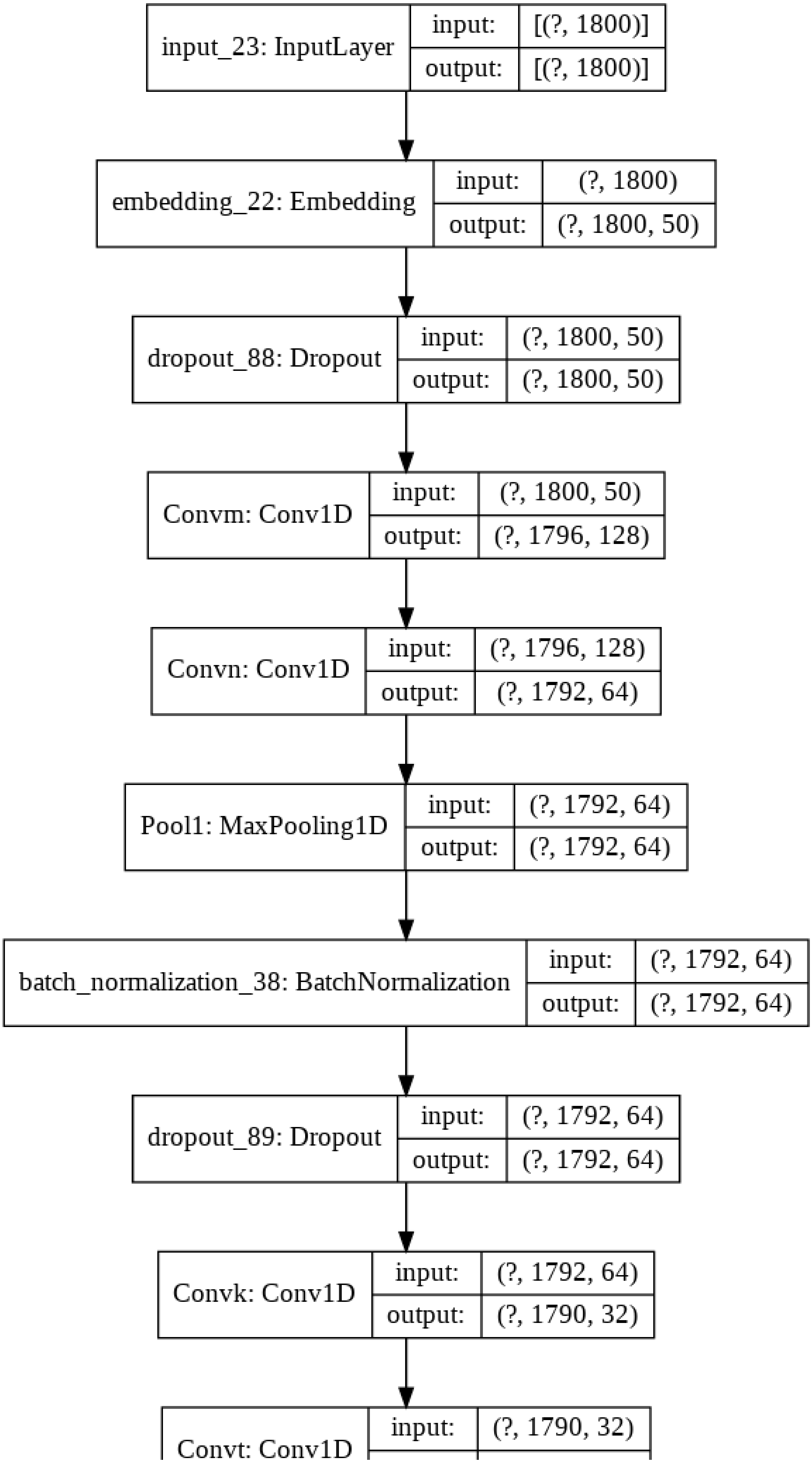
```
max length of text :  41629
mean length of text :  997.0806600099144
90th percentile of text : 1789.0
```

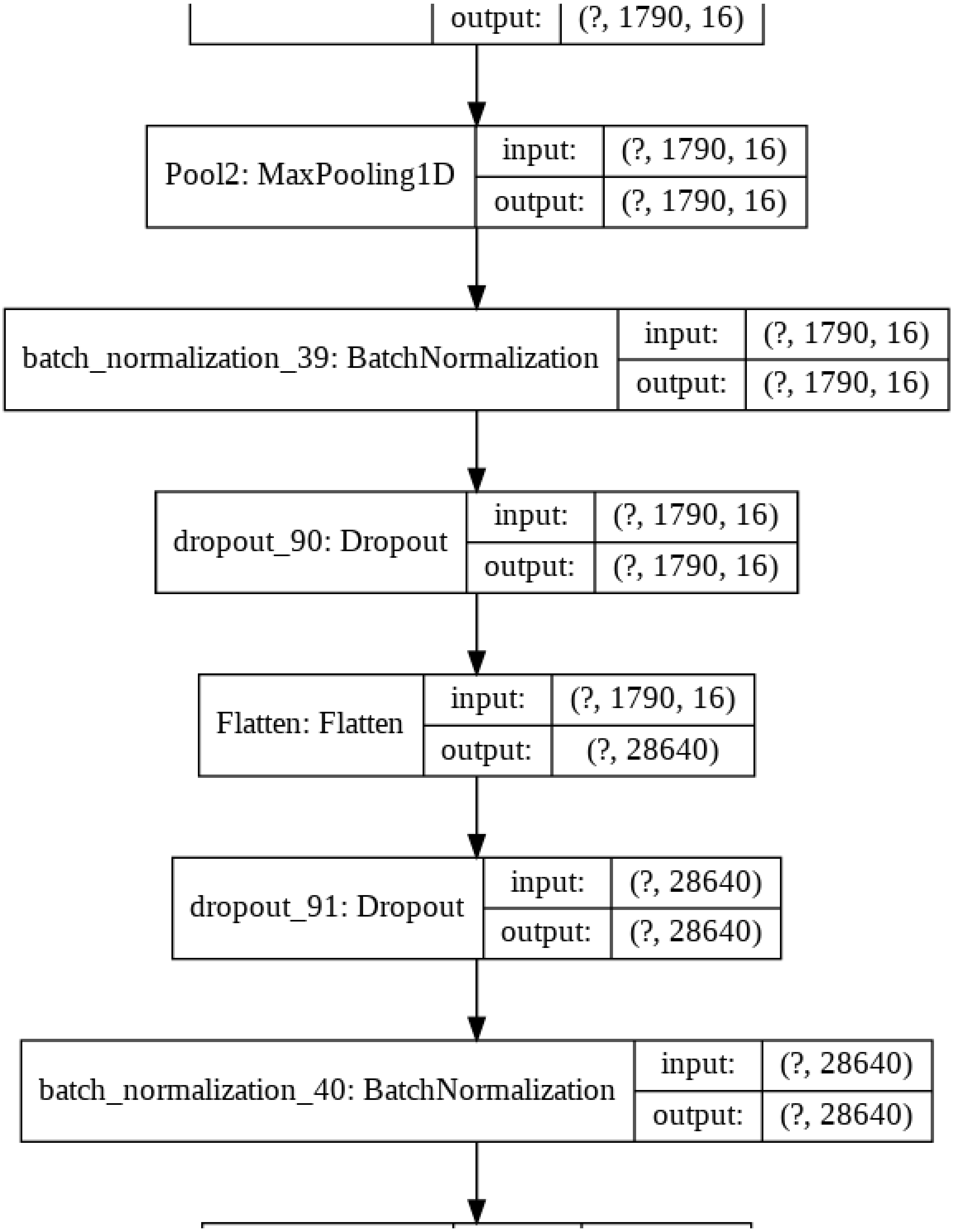
```
# truncate and/or pad input sequences
max_review_length = 1800
X_train_seq_char = sequence.pad_sequences(train_token, maxlen=max_review_length)
X_test_seq_char = sequence.pad_sequences(test_token , maxlen=max_review_length)
```

```
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
```

```
#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=3
                                                                      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=3
                                                          kernel_regularizer=l2(0.00001),r

#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=3
                                                                 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)
```


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
     _________________________________________________________________
     dropout_91 (Dropout)         (None, 28640)             0
     _________________________________________________________________
     batch_normalization_40 (Batc (None, 28640)             114560
     _________________________________________________________________
     dense_24 (Dense)             (None, 64)                1833024
     _________________________________________________________________
     Output (Dense)               (None, 20)                1300
     =================================================================
     Total params: 2,119,060
     Trainable params: 2,061,620
     Non-trainable params: 57,440
     _________________________________________________________________
```

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

```
#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
```

```
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: 0.0
     220/221 [============================>.] - ETA: 0s - loss: 3.5003 - accuracy: 0.05
     Epoch 00001: val_accuracy improved from -inf to 0.05141, saving model to best_mode
     221/221 [==============================] - 14s 62ms/step - loss: 3.5005 - accuracy
     Epoch 2/25
     220/221 [============================>.] - ETA: 0s - loss: 3.3809 - accuracy: 0.06
     Epoch 00002: val_accuracy improved from 0.05141 to 0.07096, saving model to best_m
     221/221 [==============================] - 13s 60ms/step - loss: 3.3809 - accuracy
     Epoch 3/25
     220/221 [============================>.] - ETA: 0s - loss: 3.3418 - accuracy: 0.07
     Epoch 00003: val_accuracy improved from 0.07096 to 0.07521, saving model to best_m
```