Text Classification:

Data

1. we have total of 20 types of documents(Text files) and total 18828 documents(text files).
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'Classl
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:

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 those after that remove the words whose length is less than or equal to 2 and also remove'com' wor

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

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

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

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-faecd4605953> in <module>()
      1 # we have collected all emails and preprocessed them, this is sample output
----> 2 preprocessed_email

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

SEARCH STACK OVERFLOW

```
len(preprocessed_email)
```

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 chars
Eg: if we have sentance like "Subject: Re: Gospel Dating @ \r\r\n" --> You have to get "Gosp
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.umd.

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-The c

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

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 --> i v

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))
```

We did chunking for above two lines and then We got one list where each word is mapped to a
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 "Srikar
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.

```
data.columns
```

```
data.iloc[400]
```

To get above mentioned data frame --> Try to Write Total Preprocessing steps in One Function Named Preprocess as below.

Code checking:

After Writing preprocess function. call that functoin with the input text of 'alt.atheism_49960' doc and print the output of the preprocess function
This will help us to evaluate faster, based on the output we can suggest you if there are any changes.

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 colum

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 model.

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 acceptable.

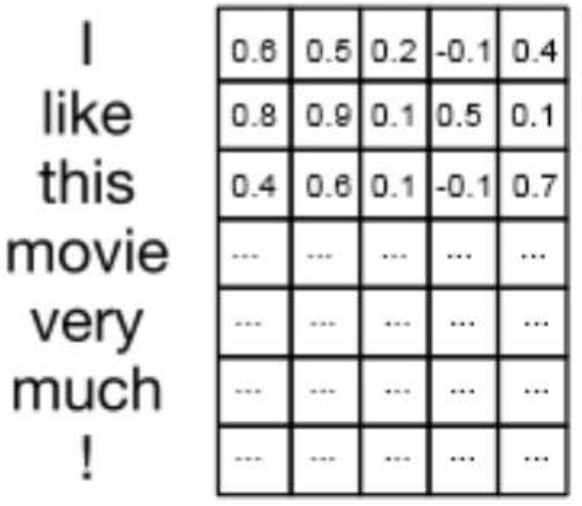
13. Try to use **Early Stopping** technique or any of the callback techniques that you did in th

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.

Model-1: Using 1D convolutions with word embeddings

Encoding of the Text --> For a given text data create a Matrix with Embedding layer as shov
In the example we have considered d = 5, but in this assignment we will get d = dimension of
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 layer

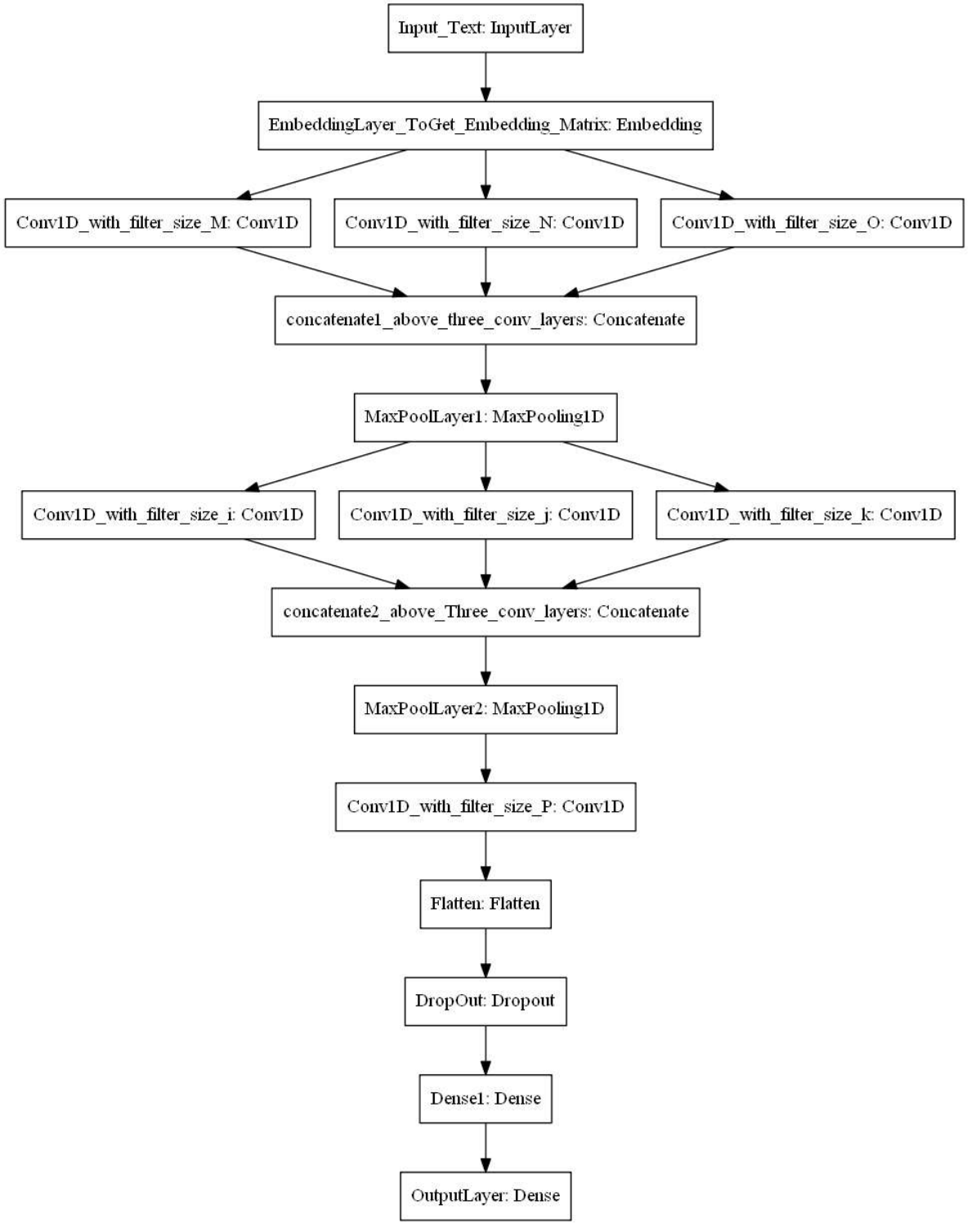


Ref: https://i.imgur.com/kiVQuk1.png

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

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/



ref: 'https://i.imgur.com/fv1GvFJ.png'

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

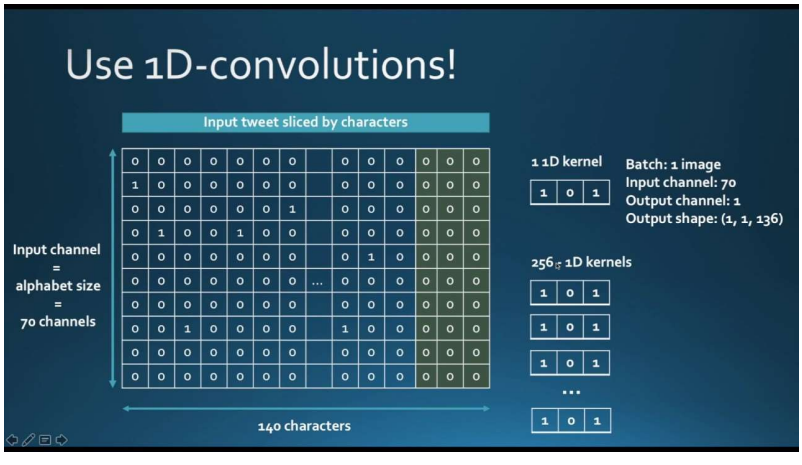
2. use concatenate layer is to concatenate all the filters/channels.

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 param
(Only recommendation if you have less computing power)

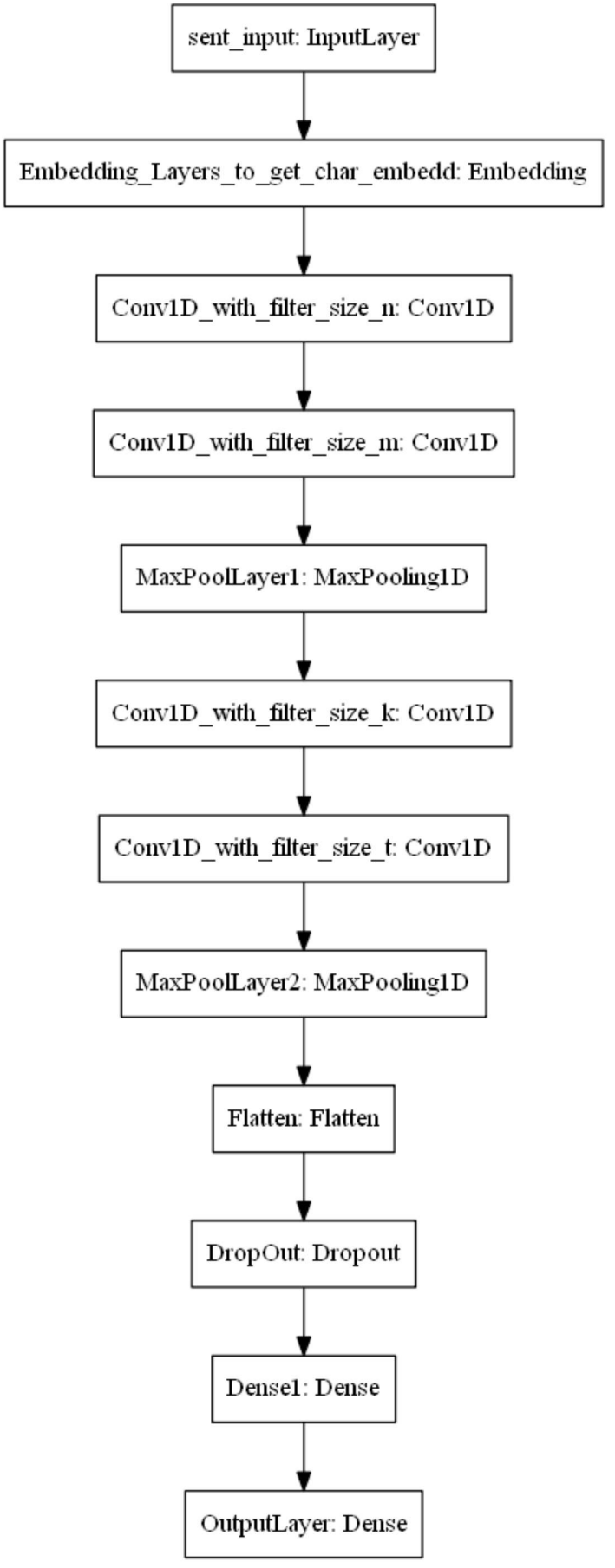
5. You can use any number of layers after the Flatten Layer.

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 Text C
2. Yoon Kim, Yacine Jernite, David Sontag, Alexander M. Rush. Character-Aware Neural Lang
3. Shaojie Bai, J. Zico Kolter, Vladlen Koltun. An Empirical Evaluation of Generic Convol
4. Use the pratrained char embeddings https://github.com/minimaxir/char-embeddings/blob/m


sent_input: InputLayer
Embedding_Layers_to_get_char_embedd: Embedding
Conv1D_with_filter_size_n: Conv1D
Conv1D_with_filter_size_m: Conv1D
MaxPoolLayer1: MaxPooling1D
Conv1D_with_filter_size_k: Conv1D
Conv1D_with_filter_size_t: Conv1D
MaxPoolLayer2: MaxPooling1D
Flatten: Flatten
DropOut: Dropout
Dense1: Dense
OutputLayer: Dense

```
# importing useful libraries
import os
import pandas as pd
import numpy as np
import re
```

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

```
     Mounted at /content/drive
```

```
# https://exerror.com/unicodedecodeerror-utf-8-codec-cant-decode-byte-0xff-in-position-0-inval
with open(r"/content/drive/MyDrive/documents/alt.atheism_49960.txt", encoding="utf8", errors='
    file= f.read()
```

We will do Preprocessing on Single Document First to check whether its giving correct results or not and then we will preprocess all the document.

```
# Task 1
def preprocessing_emails(file):
  email_preprocessed = []
  for i in re.findall(r'[\w\-\.]+@[\w\.-]+\b', file):  #extracting from a document
      print(i)
  # \w → Match any alphanumeric character
  #\-\. → Match “-” and “.” ( “\” is used to escape special characters)
  #[]+ → Match one or more than one characters inside the brackets
  #Match “@” after [\w\-\.]
      email = i.split('@')[1]      #taking text after @ from email
      email = email.split('.')     # splitting the words by '.'
      file=file.replace(i,' ')     # Replacing all the emails by space in the original text.
      for word in email:            #removing words less than 2
          if len(word)>2 and word != 'com':
              email_preprocessed.append(word)
```

```
    email_preprocessed=set(email_preprocessed) #here we are using set becuase it may possible th
    #eg :
    #['mantis', 'co', 'uk']
    #['netcom', 'com']
    #['mantis', 'co', 'uk']
    # here mantis word occur 2 times in emails of a single file
    # output : netcom mantis
    return ' '.join(email_preprocessed)
# Reference : https://towardsdatascience.com/how-i-preprocessed-text-data-using-regular-expres

preprocessing_emails(file)
#before Preprocessing --->mathew@mantis.co.uk
#                     --->figmo@netcom.com
#                     --->archive-server@mantis.co.uk
# after preprocessing --> 'netcom mantis'
```

```
mathew@mantis.co.uk
figmo@netcom.com
archive-server@mantis.co.uk
'mantis netcom'
```

```
# Preprocessing subject
temp1 = re.findall(r'^Subject.*$',file, re.MULTILINE)
print(temp1)
sub = temp1[0]
print(sub)
sub = sub[7:]
print(sub)
# https://stackoverflow.com/questions/25045373/use-regex-re-sub-to-remove-everything-before-an
sub = re.sub('^(.*)(?=:)',"",sub)
sub
```

```
['Subject: Alt.Atheism FAQ: Atheist Resources']
Subject: Alt.Atheism FAQ: Atheist Resources
: Alt.Atheism FAQ: Atheist Resources
': Atheist Resources'
```

```
# Rough to check implementation is correct or not
import string
for i in string.punctuation:   #remove all the non-alphanumeric
    sub = sub.replace(i," ")
    #sub = re.sub(r"re","",sub, flags=re.IGNORECASE) #removing Re
    sub = sub.lower()  #lower-casing
    sub = sub.strip()
print(sub)
#re.sub(r'Subject.*$'," ", flags=re.MULTILINE)  #removing subject
```

```
 atheist resources
```

```
def preprocessed_subject(file):
    var = re.findall(r'^Subject.*$',file, re.MULTILINE) # returns list of subject matches
    for i in var:
      file=file.replace(i,' ') # replacing the subjects with an empty space
      sub = i
      sub = sub[7:] #getting all the characters after 7 character (i.e getting all the charact
      sub = re.sub('^(.*)(?=:)',"",sub) # removing all the caharcters before colon (eg: Alt.At
      # https://stackoverflow.com/questions/25045373/use-regex-re-sub-to-remove-everything-bef
```

```
        for i in string.punctuation: # this function from string library returns all the sets of
             #https://www.geeksforgeeks.org/string-punctuation-in-python/
            sub = sub.replace(i," ") # replacing all the punctuation with empty space
            sub = sub.lower() # lowercase
            sub = sub.strip() # removing trailing and leading spaces
        return sub

preprocessed_subject(file)
# before preprocessing ---> ['Subject: Alt.Atheism FAQ: Atheist Resources']
# after preprocessing ---> [atheist resources]
```

```
'atheist resources'
```

```
# task 5
#  Delete all the sentances where sentence starts with "Write to:" or "From:".
String = " Hello Write to: and Form:"
String = re.sub(r"Write to:.*$"," ",String, flags=re.MULTILINE)   #Deleting  all the sentances
String = re.sub(r"From:.*$"," ",String, flags=re.MULTILINE) #Deleting  all the sentances where
print(String)
# https://towardsdatascience.com/how-i-preprocessed-text-data-using-regular-expressions-for-my
```

```
 Hello
```

```
# task 6
# Delete all the tags like "< anyword >"
String =  "Hello My name is Prince <Bari>"
String = re.sub(r"<.*>","",String, flags=re.MULTILINE)
String
```

```
'Hello My name is Prince '
```

```
# task 7
#  Delete all the data which are present in the brackets.
String = "My Name is Prince (Bari)"
String = re.sub(r"\(.*\)","",String,flags=re.MULTILINE)
String
```

```
'My Name is Prince '
```

```
#task 8
#Remove all the newlines('\n'), tabs('\t'), "-", "\".
String = "\t Data -is -Gold \ \n"
String = re.sub(r"[\n\t\-\\\/]","",String, flags=re.MULTILINE)
String
```

```
' Data is Gold  '
```

```
#task 9
# Remove all the words which ends with ":".
String = " DataScience: Data Science is the Hottest Job of 21st Century "
String=re.sub(r'[\w]+:','',String)
String
```

```
'  Data Science is the Hottest Job of 21st Century '
```

```
#Decontractions
# https://stackoverflow.com/a/47091490/4084039
```

```
def decontracted(file):
    # specific
    file = re.sub(r"won't", "will not", file)
    file = re.sub(r"can\'t", "can not", file)

    # general
    file = re.sub(r"n\'t", " not", file)
    file = re.sub(r"\'re", " are", file)
    file = re.sub(r"\'s", " is", file)
    file = re.sub(r"\'d", " would", file)
    file = re.sub(r"\'ll", " will", file)
    file = re.sub(r"\'t", " not", file)
    file = re.sub(r"\'ve", " have", file)
    file = re.sub(r"\'m", " am", file)
    return file
# Reference : Donor Choose Preprocessing Noteook


string =  "hey it won't happen and machine learning 'll be future"
decontracted(string)
```

```
'hey it will not happen and machine learning  will be future'
```

```
# importing useful library
!pip install nltk
import re
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/publi
Requirement already satisfied: nltk in /usr/local/lib/python3.7/dist-packages (3.7)
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages (from nltk
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from nltk)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from nl
[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
```

```
#Chunking
from nltk import word_tokenize, pos_tag, ne_chunk
from nltk import Tree
string = "i am living in the New York and My name is Bari Prince"
chunks = []
chunks=(list(ne_chunk(pos_tag(word_tokenize(string)))))
print(chunks)
# if the type is tree and label is GPE than its a place
```

```
# if the plave has more than one word than we will join them with "_" (eg : New York ----> New
for i in chunks:
    if type(i)==Tree:
        if i.label() == "GPE":
                if len(i.leaves())>1:
                    gpe = "_".join([term for term,pos in i.leaves()])
                    l = i.leaves() # we will
                    # l contains the leaf node so l[1][0] contains 2nd term("York") and l[0][0
                    # so we will replace the 2nd term with gpe i.e we will replace the York wi
                    # We will remove first term of leaf node from the string i.e we will delet
                    string = re.sub(rf'{l[1][0]}',gpe,string, flags=re.MULTILINE)
                    string = re.sub(rf'\b{l[0][0]}\b',"",string, flags=re.MULTILINE)
        if i.label()=="PERSON":
            # if type is tree and if labels is not "GPE" then we will check for whether label
            # if label is  person than we will remove the name of the person from tyhe origin
            for term,pos in i.leaves():
                # To delete Person, we use re.escape because the term can contain regular expr
                # https://docs.python.org/3/library/re.html
                string = re.sub(re.escape(term),"",string, flags=re.MULTILINE)
print(string)
#Reference : https://towardsdatascience.com/how-i-preprocessed-text-data-using-regular-express
```

```
[('i', 'NN'), ('am', 'VBP'), ('living', 'VBG'), ('in', 'IN'), ('the', 'DT'), Tree('GPE',
i am living in the  New_York and My name is
```

```
# task 13
# replacing digits with space
String = "Hello 1 2 3 4 Prince"
String = re.sub(r'\d',"",String, flags=re.MULTILINE)
String
# Reference : https://www.geeksforgeeks.org/python-replace-all-numbers-by-k-in-given-string/
```

```
'Hello     Prince'
```

```
# task 14
# Replacing “_word_” , “_word” , “word_” to word
f = "_word_"
f = re.sub(r"\b_([a-zA-z]+)_\b",r"\1",f) #replacinhg _word_ to word

f = re.sub(r"\b_([a-zA-z]+)\b",r"\1",f) #replacing_word to word
f = re.sub(r"\b([a-zA-z]+)_\b",r"\1",f) #replacing word_ to word
f

# Here we are apturing only words and rplacing it with pattern eg: _word_ ----> we are only ca
# Here we have one group here to extract the group we use r ‘\1’ to extract the particular gro
# Reference : https://stackoverflow.com/questions/61379982/how-to-keep-a-captured-group-only-i
```

```
'word'
```

```
# task 15
f = "d_berlin and dr_berlin"
f = re.sub(r"[\w]+_([a-zA-Z]+)",r"\1",f) #d_berlin to berlin
print(f)
```

```
berlin and berlin
```

```
# task 16
```

```
# Converting all the words into the lower case
file = file.lower()

# task 17
# replacing all the words except "A-Za-z_" with space.
f = "Hello 1 2 3$ 901 # "
f = re.sub(r"[^a-zA-Z_]"," ",f)
f
```

```
'Hello              '
```

Preprocessing on All the Files

```
# Task 1
def preprocessed_emails(file):
  email_preprocessed = []
  for i in re.findall(r'[\w\-\.]+@[\w\.-]+\b', file):  #extracting from a document
  # \w → Match any alphanumeric character
  #\-\. → Match “-” and “.” ( “\” is used to escape special characters)
  #[]+ → Match one or more than one characters inside the brackets
  #Match “@” after [\w\-\.]
      email = i.split('@')[1]      #taking text after @ from email
      email = email.split('.')     # splitting the words by '.'
      #file=file.replace(i,' ')     # Replacing all the emails by space in the original text.
      for word in email:            #removing words less than 2
          if len(word)>2 and word != 'com':
              email_preprocessed.append(word)
  email_preprocessed=set(email_preprocessed) #here we are using set becuase it may possible th
  #eg :
  #['mantis', 'co', 'uk']
  #['netcom', 'com']
  #['mantis', 'co', 'uk']
  # here mantis word occur 2 times in emails of a single file
  # output : netcom mantis
  return ' '.join(email_preprocessed)
# Reference : https://towardsdatascience.com/how-i-preprocessed-text-data-using-regular-expres


# Preprocessing Subject
import string
def preprocessed_subject(file):
    var = re.findall(r'^Subject.*$',file, flags = re.MULTILINE) # returns list of subject matc
    for i in var:
      #file=file.replace(i,' ') # replacing the subjects with an empty space
      sub = i
      sub = sub[7:] #getting all the characters after 7 character (i.e getting all the charact
      sub = re.sub('^(.*)(?=:)',"",sub) # removing all the caharcters before colon (eg: Alt.At
      # https://stackoverflow.com/questions/25045373/use-regex-re-sub-to-remove-everything-bef

      for i in string.punctuation: # this function from string library returns all the sets of
           #https://www.geeksforgeeks.org/string-punctuation-in-python/
          sub = sub.replace(i," ") # replacing all the punctuation with empty space
          sub = sub.lower() # lowercase
          sub = sub.strip() # removing trailing and leading spaces
      return sub


#task 11
# chunking
#Chunking
```

```
from nltk import word_tokenize, pos_tag, ne_chunk
from nltk import Tree
def chunking(file):
    chunks = []
    chunks=(list(ne_chunk(pos_tag(word_tokenize(file)))))
    #print(chunks)
    # if the type is tree and label is GPE than its a place
    # if the plave has more than one word than we will join them with "_" (eg : New York ---->
    for i in chunks:
        if type(i)==Tree:
            if i.label() == "GPE":
                if len(i.leaves())>1:
                    gpe = "_".join([term for term,pos in i.leaves()])
                    l = i.leaves() # we will
                    # l contains the leaf node so l[1][0] contains 2nd term("York") and l[0][0
                    # so we will replace the 2nd term with gpe i.e we will replace the York wi
                    # We will remove first term of leaf node from the string i.e we will delet
                    string = re.sub(rf'{l[1][0]}',gpe,file, flags=re.MULTILINE)
                    string = re.sub(rf'\b{l[0][0]}\b',"",file, flags=re.MULTILINE)
            if i.label()=="PERSON":
                # if type is tree and if labels is not "GPE" then we will check for whether la
                # if label is  person than we will remove the name of the person from tyhe orr
                for term,pos in i.leaves():
                    # To delete Person, we use re.escape because the term can contain regular
                    # https://docs.python.org/3/library/re.html
                    file = re.sub(re.escape(term),"",file, flags=re.MULTILINE)
    return file
#Reference : https://towardsdatascience.com/how-i-preprocessed-text-data-using-regular-express


def preprocessed_text(file):
   file=re.sub(r"[\w\-\.]+@[\w\.-]+\b",' ',file, flags = re.MULTILINE) ## replacing email with
   file=re.sub(r'^Subject.*$',' ',file,flags = re.MULTILINE) # replacing the sentances that st


   # task 5
   #  Delete all the sentances where sentence starts with "Write to:" or "From:".
   file = re.sub(r"Write to:.*$"," ",file, flags=re.MULTILINE)   #Deleting  all the sentances
   file = re.sub(r"From:.*$"," ",file, flags=re.MULTILINE) #Deleting  all the sentances where
   # https://towardsdatascience.com/how-i-preprocessed-text-data-using-regular-expressions-for

   # task 6
   # Delete all the tags like "< anyword >"
   file = re.sub(r"<.*>","",file, flags=re.MULTILINE)

   # task 7
   #  Delete all the data which are present in the brackets.
   file = re.sub(r"\(.*\)","",file,flags=re.MULTILINE)

   #task 8
   #Remove all the newlines('\n'), tabs('\t'), "-", "\".
   file = re.sub(r"[\n\t\-\\\/]","",file, flags=re.MULTILINE)

   #task 9
   # Remove all the words which ends with ":".
   file = re.sub(r'[\w]+:','',file)

   # task 10
   # Decontractions, replace words like below to full words.
   #Eg: can't -> can not, 's -> is, i've -> i have, i'm -> i am, you're -> you are, i'll --> i
   file = decontracted(file)
```

```
    # task 11
    # Chunking
    file = chunking(file)

    # task 13
    # replacing digits with space
    file = re.sub(r'\d',"",file, flags=re.MULTILINE)
    # Reference : https://www.geeksforgeeks.org/python-replace-all-numbers-by-k-in-given-string


    # task14
    # Replacing “_word_” , “_word” , “word_” to word
    file = re.sub(r"\b_([a-zA-z]+)_\b",r"\1",file) #replacinhg _word_ to word
    file = re.sub(r"\b_([a-zA-z]+)\b",r"\1",file) #replacing_word to word
    file = re.sub(r"\b([a-zA-z]+)_\b",r"\1",file) #replacing word_ to word
    # Here we are apturing only words and rplacing it with pattern eg: _word_ ----> we are only
    # Here we have one group here to extract the group we use r ‘\1’ to extract the particular
    # Reference : https://stackoverflow.com/questions/61379982/how-to-keep-a-captured-group-onl

    # task 15
    file = re.sub(r"[\w]+_([a-zA-Z]+)",r"\1",file) #d_berlin to berlin

    # task 16
    # Converting all the words into the lower case
    file = file.lower()

    # task 17
    # replacing all the words except "A-Za-z_" with space.
    file = re.sub(r"[^a-zA-Z_]"," ",file)

    return file
```

Storing all the files path in a list

```
# stroing all the file path in a list
#https://www.techiedelight.com/iterate-over-files-directory-python/
from tqdm import tqdm
files_name = []
directory = '/content/drive/MyDrive/documents'
for filename in tqdm(os.listdir(directory)):
    f = os.path.join(directory, filename)
    if os.path.isfile(f):
        files_name.append(f)
```

```
 100%|██████████| 18828/18828 [00:02<00:00, 8183.89it/s]
```

```
# Preprocess function that will do all the preprocessing
def preprocess(file):
    """Do all the Preprocessing as shown above and
    return a tuple contain preprocess_email,preprocess_subject,preprocess_text for that Text_d
    pre_email = preprocessed_emails(file)

    pre_subject = preprocessed_subject(file)

    pre_text = preprocessed_text(file)

    return (pre_email , pre_subject , pre_text)
```

```
# checking for a single file to know whether every preprocessing step is working correct or no
with open(r"/content/drive/MyDrive/documents/alt.atheism_49960.txt", encoding="utf8", errors='
    file= f.read()
# callin a function
pre_email , pre_subjects , pre_texts = preprocess(file)


pre_email
```

```
'mantis netcom'
```

```
pre_subjects
```

```
'atheist resources'
```

```
pre_texts
```

```
'      december                                  atheist resources                       a
ddresses of atheist organizations                                        usafreedom from re
ligion foundation fish bumper stickers and assorted other atheist paraphernalia areavail
able from the freedom from religion foundation in the us    evolution signsevolution des
igns sell the   fish    it is a fish symbol  like the oness stick on their cars  but wit
h feet and the word    writteninside   the deluxe moulded d plastic fish is    postpaid
in the us             ca  people in the san francisco bay area can get   from   try mail
```

```
# preprocessing on all the files
preprocessed_emails_lst=[] # list to store the preprocessed emails of all the file
preprocessed_subjects_lst=[] # list  to store the preprocessed subjects of all the file
preprocessed_text_lst=[] # list to store the preprocessed text of all the files
for path in tqdm(files_name):
    with open(path, encoding="utf8", errors='ignore') as f:
        file= f.read()
    pre_email , pre_subjects , pre_texts = preprocess(file)
    preprocessed_emails_lst.append(pre_email)
    preprocessed_subjects_lst.append(pre_subjects)
    preprocessed_text_lst.append(pre_texts)
```

```
100%|██████████| 18828/18828 [1:44:00<00:00,  3.02it/s]
```

```
import pickle
with open('preprocess_data.pickle', 'wb') as f:
    pickle.dump([preprocessed_emails_lst,preprocessed_subjects_lst,preprocessed_text_lst], f)


# Extractacting class Label from document name
# alt.atheism_49960.txt   -----> alt.atheism as class label and 49960 as document number
# we are splitting the document number and class label using '_'
import os
class_label = []
document_number = []
for filename in os.listdir("/content/drive/MyDrive/documents"):
      i,j = filename.split('_')
      j = int(j.split('.')[0]) # splitting 49960.txt using '.' and storing only document numb
      class_label.append(i)
      document_number.append(j)


doc_num[:5]
```

```
[178802, 179113, 178876, 178543, 178982]
```

```
# Reading eah file and storing them into a list
from tqdm import tqdm
original_text=[]
for name in tqdm(files_name):
    with open(name , encoding="utf8", errors='ignore') as f:
        file= f.read() # reading file
    original_text.append(file)
```

```
 100%|██████████| 18828/18828 [00:17<00:00, 1079.36it/s]
```

```
import pickle
with open(r'/content/preprocess_data.pickle', 'rb') as f:
     pickle_obj = pickle.load(f)
preprocessed_emails = pickle_obj[0]
preprocessed_subjects = pickle_obj[1]
preprocessed_texts = pickle_obj[2]
```

```
import pickle
with open('text_classification_data.pickle', 'wb') as f:
     pickle.dump([preprocessed_emails , preprocessed_subjects , preprocessed_texts , original_
```

```
#Creating pandas dataset
import pandas as pd
data = {'Text':original_text,'Preprocessed_Text':preprocessed_texts,'Preprocessed_Subject':pre
dataframe = pd.DataFrame(data)
```

```
dataframe.head()
```

	Text	Preprocessed_Text	Preprocessed_Subject	Preprocessed_
0	From: steveh@thor.isc-br.com (Steve Hendricks)...	in article in article fury of ...	the earth also pollutes	ISC-BR titan thor org nasa gov
1	From: cramer@optilink.COM (Clayton Cramer)\nSu...	in article in i used to think ...	new study out on gay percentage	galileo roc uhura edu
2	From: golchowy@alchemy.chem.utoronto.ca (Geral...	in article also in the april glo...	eight myths about national health insurance p...	news chem u buffalo utoron colu

```
dataframe.shape
```

```
(18828, 5)
```

```
dataframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18828 entries, 0 to 18827
Data columns (total 5 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   Text                 18828 non-null  object
 1   Preprocessed_Text    18828 non-null  object
 2   Preprocessed_Subject  18828 non-null  object
 3   Preprocessed_Email   18828 non-null  object
```

```
 4   Class                18828 non-null  object
dtypes: object(5)
memory usage: 735.6+ KB
```

```
# Combining "preprocessed_text", "preprocessed_subject", "preprocessed_emails" into one column
# https://datascientyst.com/combine-multiple-columns-into-single-one-in-pandas/
dataframe['Preprocessed_Document'] = dataframe[['Preprocessed_Text' ,   'Preprocessed_Subject'
```

```
dataframe.head()
```

	Text	Preprocessed_Text	Preprocessed_Subject	Preprocessed_
0	From: steveh@thor.isc-br.com (Steve Hendricks)...	in article in article fury of ...	the earth also pollutes	ISC-BR titan thor org nasa gov
1	From: cramer@optilink.COM (Clayton Cramer)\nSu...	in article in i used to think ...	new study out on gay percentage	galileo roc uhura edu
2	From: golchowy@alchemy.chem.utoronto.ca (Geral...	in article also in the april glo...	eight myths about national health insurance p...	news chem u buffalo utoron colu
3	From: kaldis@romulus.rutgers.edu	perhaps but most likely not more than	how many homosexuals	remus edu r

```
# Stroing final pandas dataframe into csv
dataframe.to_csv('final_dataset.csv')
```

```
# downloading the final_dataset from google drive and loading it into colab
!gdown 1nE76UGpSCGX8lSffIClHk8JunLoJRkzT
```

```
Downloading...
From: https://drive.google.com/uc?id=1nE76UGpSCGX8lSffIClHk8JunLoJRkzT
To: /content/final_dataset.csv
100% 89.8M/89.8M [00:00<00:00, 250MB/s]
```

```
import pandas as pd
dataframe = pd.read_csv("final_dataset.csv" , index_col=[0])
dataframe.head()
```

	Text	Preprocessed_Text	Preprocessed_Subject	Preprocessed_
0	From: steveh@thor.isc-br.com (Steve Hendricks)...	in article in article fury of ...	the earth also pollutes	ISC-BR titan thor org nasa gov
1	From: cramer@optilink.COM (Clayton Cramer)\nSu...	in article in i used to think ...	new study out on gay percentage	galileo roc uhura edu
2	From: golchowy@alchemy.chem.utoronto.ca (Geral...	in article also in the april glo...	eight myths about national health insurance p...	news chem u buffalo utoron colu
3	From: kaldis@romulus.rutgers.edu (Theodore A. ...	perhaps but most likely not more than ...	how many homosexuals are there	remus edu r

```
# seprating class label and Independent Features
x=dataframe['Preprocessed_Document']
y=dataframe['Class']


# Splitting the dataset into Train and test with startify sampling
from sklearn.model_selection import train_test_split
X_train , X_test , y_train , y_test = train_test_split(x, y, test_size=0.25, random_state=15,s


X_train.shape
```

```
(14121,)
```

```
X_test.shape
```

```
(4707,)
```

```
y_train.shape
```

```
(14121,)
```

```
y_test.shape
```

```
(4707,)
```

```
# Converting Categorical variable into One hot encoded vector
from sklearn.preprocessing import LabelEncoder
import numpy as np
import tensorflow as tf

label_encoder = LabelEncoder()
le_train = label_encoder.fit_transform(y_train)
train_label = tf.keras.utils.to_categorical(le_train, 20)
print(train_label)

le_test = label_encoder.transform(y_test)
test_label = tf.keras.utils.to_categorical(le_test, 20)
print(test_label)
```

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

```
# Example of learning an Embedding
# define documents
docs = ['Well done!',
        'Good work',
```

```
          'Great effort',
          'nice work',
          'Excellent!',
          'Weak',
          'Poor effort!',
          'not good',
          'poor work',
          'Could have done better.']
# define class labels
labels = np.array([1,1,1,1,1,0,0,0,0,0])

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
t = Tokenizer(filters='!"#$%&()*+,-./:;<=>?@[\\]^`{|}~\t\n') # here we are not filtering the '
t.fit_on_texts(docs)
t.word_index
```

```
{'better': 14,
 'could': 12,
 'done': 2,
 'effort': 4,
 'excellent': 9,
 'good': 3,
 'great': 7,
 'have': 13,
 'nice': 8,
 'not': 11,
 'poor': 5,
 'weak': 10,
 'well': 6,
 'work': 1}
```

Reference : https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/

https://medium.com/swlh/classification-of-documents-using-convolutional-neural-network-cnn-e0768bb81aad

```
# Encoding data to integer
encoded_data = t.texts_to_sequences(docs)
encoded_data
```

```
[[6, 2],
 [3, 1],
 [7, 4],
 [8, 1],
 [9],
 [10],
 [5, 4],
 [11, 3],
 [5, 1],
 [12, 13, 2, 14]]
```

```
# Applying padding to make all the data of same length
# here we have maximum lenght of sentence is of 4 so we will take max length of 4
paded_data = pad_sequences(encoded_data , maxlen = 4 , padding = 'post')
paded_data
```

```
array([[ 6,  2,  0,  0],
       [ 3,  1,  0,  0],
       [ 7,  4,  0,  0],
```

```
            [ 8,  1,  0,  0],
            [ 9,  0,  0,  0],
            [10,  0,  0,  0],
            [ 5,  4,  0,  0],
            [11,  3,  0,  0],
            [ 5,  1,  0,  0],
            [12, 13,  2, 14]], dtype=int32)
```

Word Embedding

```
# tokenization
# here we are not filtering the '_'
t = Tokenizer(lower= 1, oov_token="<OOV>",filters='!"#$%&()*+,-./:;<=>?@[\\]^`{|}~\t\n')
t.fit_on_texts(X_train)
word_index_train= t.word_index
encoded_train_data = t.texts_to_sequences(X_train)
```

```
# maximum length encoded vector
# document in which ther is maximum words and the length of that word is 13445
max_length_for_padding=[]
for i in encoded_train_data:
  max_length_for_padding.append(len(i))
print(max(max_length_for_padding))
print(max_length_for_padding)
```

```
13445
[1621, 194, 86, 40, 102, 138, 45, 68, 151, 69, 59, 90, 356, 128, 228, 8, 35, 299, 174, 3(
```

Here the maximum length for padding is 13445 but it will more training time so we are taking maximum length as 2000 for padding

```
# padding
#max_length=2000
max_length = 2000
X_train_padded_data = pad_sequences(encoded_train_data,padding='post',maxlen=max_length)
print(X_train_padded_data.shape)
```

```
(14121, 2000)
```

```
# padding on test data
# using data token fitted on train data to avoid data leakage
encoded_test_data = t.texts_to_sequences(X_test)
X_test_padded_data= pad_sequences(encoded_test_data ,padding='post',maxlen=max_length)
print(X_test_padded_data.shape)
```

```
(4707, 2000)
```

Using Predefined Glove Vector for our Data

```
!gdown 18FK2KpoldfEIijnbZkpYJluYsGPpsyAO
```

```
Downloading...
From: https://drive.google.com/uc?id=18FK2KpoldfEIijnbZkpYJluYsGPpsyAO
To: /content/glove.6B.100d.txt
100% 347M/347M [00:02<00:00, 138MB/s]
```

```
# buliding an index mapping words (as strings) to thier vector representation ( as number vect
# we are storing the word and its corresponding 100 dimension word vector in form of dictionar
# here we are taking glove vectors of 100 dimensions there are also 200 , 300 words dimensions
# There are total 4lacs word vector in the glove file
embedding_index = {} # dictionary to store the words as key and values as vector from glove

f = open(os.path.join("/content/glove.6B.100d.txt"),encoding = "utf8")
# the glove file is a text file consists of 1st columns as word and 2nd columns as vector corr
for line in f: # reading ecah line
    values = line.split()
    word = values[0] # storing the word
    vec  = np.asarray(values[1:] , dtype = 'float32') # vector of correponding word
    embedding_index[word] = vec
f.close()



#we will create the embedding matrix for our vocabulary
# we created using the training dataset and its vector representation will be taken from the G
from tqdm import tqdm

vocab_size=len(word_index_train)+1 # total number of words in our vocab
embedding_matrix = np.zeros((vocab_size, 100)) # creating a matrix to store the words from our
for word, i in tqdm(word_index_train.items()):
    embedding_vector = embedding_index.get(word) # getting vector of words from glove vector
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

 100%|██████████| 147672/147672 [00:00<00:00, 418443.17it/s]



embedding_matrix.shape

 (147673, 100)
```

Reference : https://www.youtube.com/watch?v=ivqXiW0X42Q&t=618s

Writing Callbacks

```
from keras.callbacks import Callback
from sklearn.metrics import roc_auc_score, f1_score
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.callbacks import LearningRateScheduler


# Callbacks to save best model weights
# Callback to save the model
#Saving the model at every epoch if your validation accuracy is improved from previous epoch
filepath="model_save/best_model_1.hdf5" # path
checkpoint_model_save = ModelCheckpoint(filepath=filepath, monitor='val_loss',  verbose=1, sav
# Reference : https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ModelCheckpoint


# # callback to stop the training if the training if the validation accuracy is not increased
earlystop = EarlyStopping(monitor='val_accuracy', patience=5, verbose=1)
# Reference : https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping
```

```
# Callback that  print micro F1 score
# Here we are inherting the parent class Callback
# we are writing our custom callbacks in a child class
# In this subclass we are writing custom callbacks to print f1_score and auc_score at the end
class F1_score(tf.keras.callbacks.Callback):
    def __init__(self,x_test,y_test):
       self.x_test=x_test
       self.y_test=y_test
    def on_epoch_end(self, epoch, logs={}):
      y_pred = (np.asarray(self.model.predict(self.x_test))).round()
      y = self.y_test
      f1_score_val = f1_score(y, y_pred.round() , average = "micro") # micro f1 score
      print(" f1_score :",f1_score_val)

Metrics = F1_score(x_test = X_test_padded_data , y_test = test_label)


# !pip install -q tf-nightly-2.0-preview
# if you want to use the tf2.0 please uncomment the above line
# Load the TensorBoard notebook extension

# there are other ways of doing this: https://www.dlology.com/blog/quick-guide-to-run-tensorbo
%load_ext tensorboard
# Clear any logs from previous runs
!rm -rf ./logs/



# Enabling Tensorboard Callback
import datetime
log_dir = os.path.join("logs",'fits', datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard= tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1,write_graph=True)
```

Model - 1 Architechture

```
from tensorflow.keras.layers import Dense, Dropout, Flatten,concatenate,Input,Embedding,Conv1D
import keras


# Model
vocab_size=len(word_index_train)+1 # total number of words in our vocab

input = Input(shape=(X_train_padded_data.shape[1],))

embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix], input_length=1000, tr

conv_layer1=Conv1D(256,kernel_size = 2,kernel_initializer='he_normal',padding = 'same' , activ
conv_layer2=Conv1D(256,kernel_size = 2,kernel_initializer='he_normal',padding = 'same' ,activa
conv_layer3=Conv1D(256,kernel_size = 2,kernel_initializer='he_normal',padding = 'same' ,activa

layer4 = concatenate([conv_layer1,conv_layer2,conv_layer3])
layer5 = MaxPooling1D(pool_size = 2 , strides = 2)(layer4)

conv_layer6=Conv1D(128,kernel_size = 2,kernel_initializer='he_normal',padding = 'same',activat
conv_layer7=Conv1D(128,kernel_size = 2,kernel_initializer='he_normal',padding = 'same',activat
conv_layer8=Conv1D(128,kernel_size = 2,kernel_initializer='he_normal',padding = 'same',activat

layer9 = concatenate([conv_layer6,conv_layer7,conv_layer8])
```

```
layer10 = MaxPooling1D(pool_size = 2 , strides = 2)(layer9)

layer11 = Conv1D(64,kernel_size = 2,kernel_initializer='he_normal',activation='relu')(layer10)
flatten_layer = Flatten()(layer11)
Dropout_layer2 = Dropout(0.7)(flatten_layer)

dense_layer1=Dense(256, activation='relu',kernel_initializer='he_normal')(Dropout_layer2)
output_layer=Dense(20, activation='softmax',kernel_initializer='he_normal')(dense_layer1)

model_1= keras.Model(input, output_layer)
model_1.summary()
```

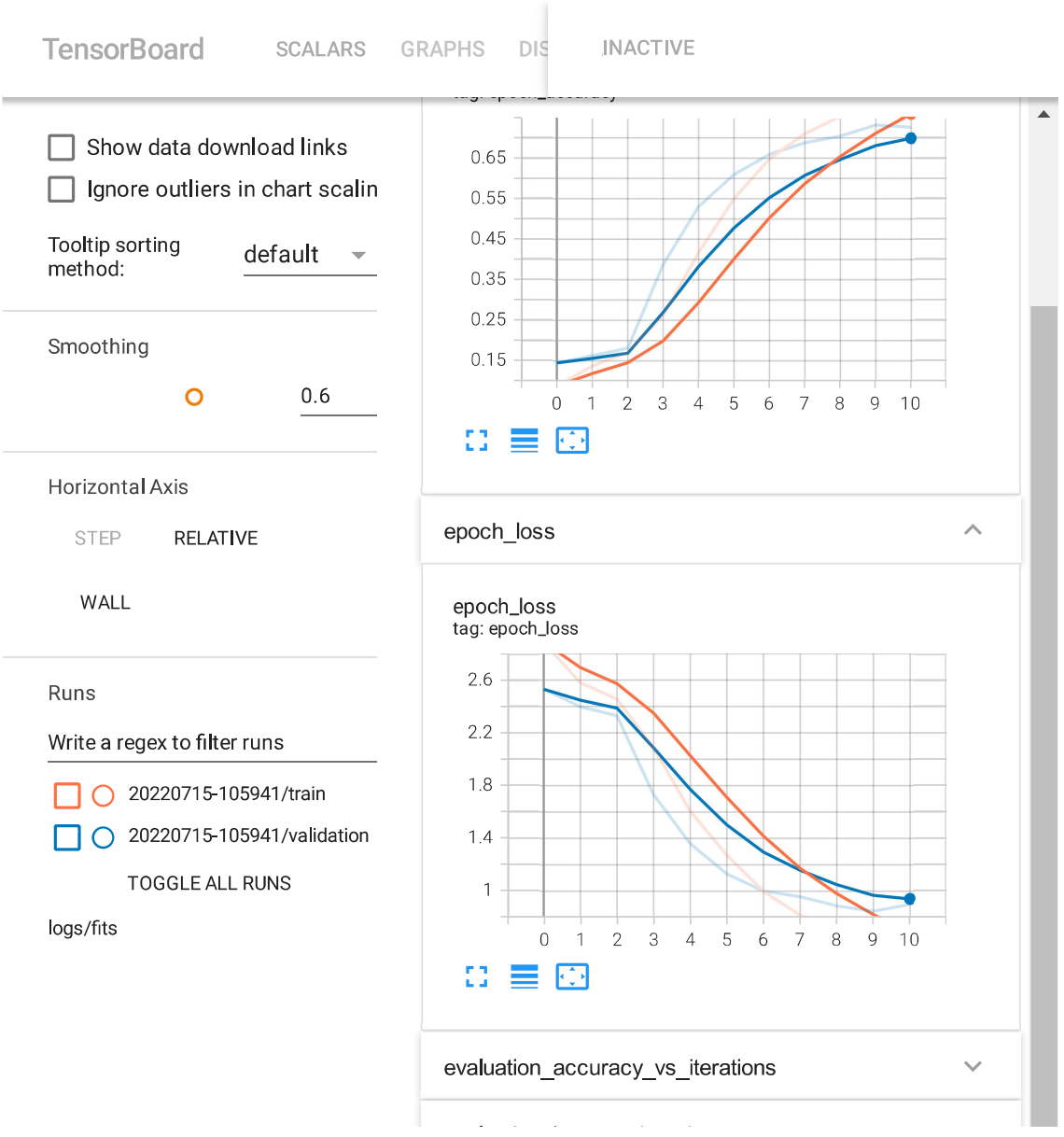
```
Model: "model"
_________________________________________________________________________________________________
 Layer (type)                    Output Shape         Param #     Connected to
=================================================================================================
 input_1 (InputLayer)            [(None, 2000)]       0           []

 embedding (Embedding)           (None, 2000, 100)    14767300    ['input_1[0][0]']

 conv1d (Conv1D)                 (None, 2000, 256)    51456       ['embedding[0][0]']

 conv1d_1 (Conv1D)               (None, 2000, 256)    51456       ['embedding[0][0]']

 conv1d_2 (Conv1D)               (None, 2000, 256)    51456       ['embedding[0][0]']

 concatenate (Concatenate)       (None, 2000, 768)    0           ['conv1d[0][0]',
                                                                  'conv1d_1[0][0]',
                                                                  'conv1d_2[0][0]']

 max_pooling1d (MaxPooling1D)    (None, 1000, 768)    0           ['concatenate[0][0]']

 conv1d_3 (Conv1D)               (None, 1000, 128)    196736      ['max_pooling1d[0][0]']

 conv1d_4 (Conv1D)               (None, 1000, 128)    196736      ['max_pooling1d[0][0]']

 conv1d_5 (Conv1D)               (None, 1000, 128)    196736      ['max_pooling1d[0][0]']

 concatenate_1 (Concatenate)     (None, 1000, 384)    0           ['conv1d_3[0][0]',
                                                                  'conv1d_4[0][0]',
                                                                  'conv1d_5[0][0]']

 max_pooling1d_1 (MaxPooling1D)  (None, 500, 384)     0           ['concatenate_1[0][0]']

 conv1d_6 (Conv1D)               (None, 499, 64)      49216       ['max_pooling1d_1[0][0]

 flatten (Flatten)               (None, 31936)        0           ['conv1d_6[0][0]']

 dropout (Dropout)               (None, 31936)        0           ['flatten[0][0]']

 dense (Dense)                   (None, 256)          8175872     ['dropout[0][0]']

 dense_1 (Dense)                 (None, 20)           5140        ['dense[0][0]']

=================================================================================================
Total params: 23,742,104
Trainable params: 8,974,804
Non-trainable params: 14,767,300
_________________________________________________________________________________________________
```

```
model_1.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')
```

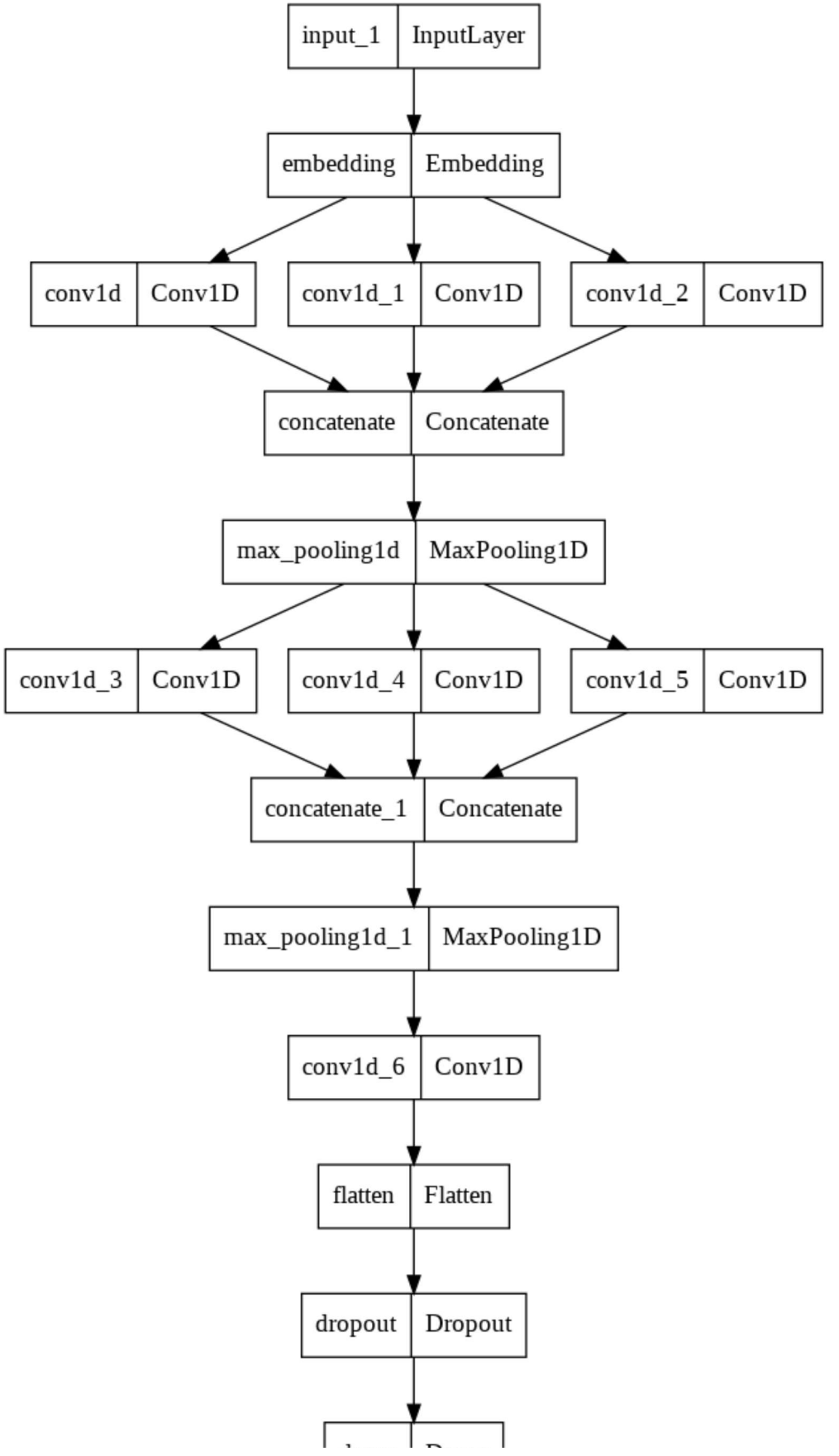
```
model_1.fit(X_train_padded_data, train_label ,validation_data=(X_test_padded_data, test_label)
```

```
Epoch 1/11
111/111 [==============================] - ETA: 0s - loss: 2.8871 - accuracy: 0.0880

Epoch 1: val_loss improved from inf to 2.53054, saving model to model_save/best_model_
111/111 [==============================] - 1933s 17s/step - loss: 2.8871 - accuracy: 0
Epoch 2/11
111/111 [==============================] - ETA: 0s - loss: 2.5799 - accuracy: 0.1370

Epoch 2: val_loss improved from 2.53054 to 2.39782, saving model to model_save/best_mc
111/111 [==============================] - 1992s 18s/step - loss: 2.5799 - accuracy: 0
Epoch 3/11
111/111 [==============================] - ETA: 0s - loss: 2.4558 - accuracy: 0.1710

Epoch 3: val_loss improved from 2.39782 to 2.32977, saving model to model_save/best_mc
111/111 [==============================] - 2012s 18s/step - loss: 2.4558 - accuracy: 0
Epoch 4/11
111/111 [==============================] - ETA: 0s - loss: 2.0868 - accuracy: 0.2626

Epoch 4: val_loss improved from 2.32977 to 1.73130, saving model to model_save/best_mc
111/111 [==============================] - 1939s 17s/step - loss: 2.0868 - accuracy: 0
Epoch 5/11
111/111 [==============================] - ETA: 0s - loss: 1.6065 - accuracy: 0.4173

Epoch 5: val_loss improved from 1.73130 to 1.36046, saving model to model_save/best_mc
111/111 [==============================] - 2077s 19s/step - loss: 1.6065 - accuracy: 0
Epoch 6/11
111/111 [==============================] - ETA: 0s - loss: 1.2731 - accuracy: 0.5501

Epoch 6: val_loss improved from 1.36046 to 1.13064, saving model to model_save/best_mc
111/111 [==============================] - 2046s 18s/step - loss: 1.2731 - accuracy: 0
Epoch 7/11
111/111 [==============================] - ETA: 0s - loss: 0.9983 - accuracy: 0.6481

Epoch 7: val_loss improved from 1.13064 to 1.00113, saving model to model_save/best_mc
111/111 [==============================] - 2019s 18s/step - loss: 0.9983 - accuracy: 0
Epoch 8/11
111/111 [==============================] - ETA: 0s - loss: 0.8178 - accuracy: 0.7103

Epoch 8: val_loss improved from 1.00113 to 0.95824, saving model to model_save/best_mc
111/111 [==============================] - 1954s 18s/step - loss: 0.8178 - accuracy: 0
Epoch 9/11
111/111 [==============================] - ETA: 0s - loss: 0.6985 - accuracy: 0.7530

Epoch 9: val_loss improved from 0.95824 to 0.88897, saving model to model_save/best_mc
111/111 [==============================] - 2019s 18s/step - loss: 0.6985 - accuracy: 0
Epoch 10/11
111/111 [==============================] - ETA: 0s - loss: 0.5873 - accuracy: 0.7966

Epoch 10: val_loss improved from 0.88897 to 0.84847, saving model to model_save/best_m
111/111 [==============================] - 1983s 18s/step - loss: 0.5873 - accuracy: 0
Epoch 11/11
111/111 [==============================] - ETA: 0s - loss: 0.4853 - accuracy: 0.8302

Epoch 11: val_loss did not improve from 0.84847
111/111 [==============================] - 1973s 18s/step - loss: 0.4853 - accuracy: 0
<keras.callbacks.History at 0x7fc2d2a40050>
```

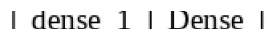
```
%tensorboard --logdir logs/fits
```




```
tf.keras.utils.plot_model(
    model_1,
    to_file='Model1 Text Classification.png',
    show_shapes=False,
    show_dtype=False,
    show_layer_names=True,
    rankdir='TB',
    expand_nested=False,
    dpi=96,
    layer_range=None,
    show_layer_activations=False
)
```


input_1 InputLayer
embedding Embedding
conv1d Conv1D
conv1d_1 Conv1D
conv1d_2 Conv1D
concatenate Concatenate
max_pooling1d MaxPooling1D
conv1d_3 Conv1D
conv1d_4 Conv1D
conv1d_5 Conv1D
concatenate_1 Concatenate
max_pooling1d_1 MaxPooling1D
conv1d_6 Conv1D
flatten Flatten
dropout Dropout

MODEL - 2 Using 1D Convolutions with Character Embedding




```
# Here we are treating every character as a token ---> char_level = True
token = Tokenizer(num_words=None, char_level=True, oov_token='UNK')
token.fit_on_texts(X_train)
```

Reference : Character Level Embedding Intution: https://youtu.be/CNY8VjJt-iQ

https://towardsdatascience.com/character-level-cnn-with-keras-50391c3adf33

https://towardsdatascience.com/the-definitive-guide-to-bidaf-part-2-word-embedding-character-embedding-and-contextual-c151fc4f05bb

Code Reference : https://towardsdatascience.com/character-level-cnn-with-keras-50391c3adf33

```
token.word_index
```

```
{'\x08': 42,
 '\x10': 41,
 ' ': 2,
 '-': 33,
 '0': 31,
 '1': 30,
 '2': 32,
 '3': 34,
 '4': 35,
 '5': 38,
 '6': 36,
 '7': 40,
 '8': 37,
 '9': 39,
 'UNK': 1,
 '_': 26,
 'a': 5,
 'b': 22,
 'c': 14,
 'd': 13,
 'e': 3,
 'f': 18,
 'g': 19,
 'h': 11,
 'i': 7,
 'j': 27,
 'k': 24,
 'l': 12,
 'm': 16,
 'n': 8,
 'o': 6,
 'p': 17,
 'q': 28,
 'r': 10,
 's': 9,
 't': 4,
 'u': 15,
 'v': 23,
 'w': 21,
 'x': 25,
 'y': 20,
 'z': 29}
```

```
# construct a new vocabulary
```

```
characters = "abcdefghijklmnopqrstuvwxyz0123456789,;.!?:'\"/\\|_@#$%^&*~`+-=<>()[]{}"
character_dict = {}
for i, char in enumerate(characters):
    character_dict[char] = i + 1

# Use character_dict to replace the tk.word_index
# Here we are replacing the character tokens which is generated from train data with our vocab
token.word_index = character_dict.copy()
# Add 'UNK' to the vocabulary
# oov_token if given, it will be added to word_index and used to replace out-of-vocabulary wor
token.word_index[token.oov_token] = max(character_dict.values()) + 1

token.word_index
```

```
      , : 37,
      '-': 59,
      '.': 39,
      '/': 45,
      '0': 27,
      '1': 28,
      '2': 29,
      '3': 30,
      '4': 31,
      '5': 32,
      '6': 33,
      '7': 34,
      '8': 35,
      '9': 36,
      ':': 42,
      ';': 38,
      '<': 61,
      '=': 60,
      '>': 62,
      '?': 41,
      '@': 49,
      'UNK': 69,
      '[': 65,
      '\\': 46,
      ']': 66,
      '^': 53,
      '_': 48,
      '`': 57,
      'a': 1,
      'b': 2,
      'c': 3,
      'd': 4,
      'e': 5,
      'f': 6,
      'g': 7,
      'h': 8,
      'i': 9,
      'j': 10,
      'k': 11,
      'l': 12,
      'm': 13,
      'n': 14,
      'o': 15,
      'p': 16,
      'q': 17,
      'r': 18,
      's': 19,
      't': 20,
      'u': 21,
      'v': 22,
      'w': 23,
      'x': 24,
```

```
 'y': 25,
 'z': 26,
 '{': 67,
 '|': 47,
 '}': 68,
 '~': 56}
```

```
print(len(token.word_index))
vocab_length = len(token.word_index)
```

```
69
```

```
# Transforms each text in texts to a sequence of integers.
# Converting each character of text to a sequence of integer.
encoded_train_data = token.texts_to_sequences(X_train)
encoded_test_data = token.texts_to_sequences(X_test)
```

```
# padding
#max_length=2000
# taking maximum length of 2000 for padding
max_length = 2000
X_train_padded_data = pad_sequences(encoded_train_data,padding='post',maxlen=max_length)
print(X_train_padded_data.shape)
X_test_padded_data= pad_sequences(encoded_test_data ,padding='post',maxlen=max_length)
print(X_test_padded_data.shape)
```

```
(14121, 2000)
(4707, 2000)
```

```
# We are using one hot encoded vector to reprent this 69 character
# Because Keras use 0 for PAD, we add a zero vector to represent PAD.
# Creating Embedding Matrix
embedding_weights=[]
embedding_weights.append(np.zeros(vocab_length))

for char,i in token.word_index.items():
  onehot=np.zeros(vocab_length)
  onehot[i-1]=1
  embedding_weights.append(onehot)
embedding_weights=np.array(embedding_weights)
print(embedding_weights)
print(embedding_weights.shape)
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 1. 0. 0.]
 [0. 0. 0. ... 0. 1. 0.]
 [0. 0. 0. ... 0. 0. 1.]]
(70, 69)
```

```
# Callbacks to save best model weights
# Callback to save the model
#Saving the model at every epoch if your validation accuracy is improved from previous epoch
filepath="model_save/best_model_2.hdf5" # path
```

```
checkpoint_model_save = ModelCheckpoint(filepath=filepath, monitor='val_loss',  verbose=1, sav
# Reference : https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ModelCheckpoint

# Callback that  print micro F1 score
# Here we are inherting the parent class Callback
# we are writing our custom callbacks in a child class
# In this subclass we are writing custom callbacks to print f1_score and auc_score at the end
class F1_score(tf.keras.callbacks.Callback):
    def __init__(self,x_test,y_test):
       self.x_test=x_test
       self.y_test=y_test
    def on_epoch_end(self, epoch, logs={}):
      y_pred = (np.asarray(self.model.predict(self.x_test))).round()
      y = self.y_test
      f1_score_val = f1_score(y, y_pred.round() , average = "micro") # micro f1 score
      print(" f1_score :",f1_score_val)

Metrics = F1_score(x_test = X_test_padded_data , y_test = test_label)


# !pip install -q tf-nightly-2.0-preview
# if you want to use the tf2.0 please uncomment the above line
# Load the TensorBoard notebook extension

# there are other ways of doing this: https://www.dlology.com/blog/quick-guide-to-run-tensorbo
%load_ext tensorboard
# Clear any logs from previous runs
!rm -rf ./logs/

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


# Enabling Tensorboard Callback
import os
import datetime
log_dir = os.path.join("logs",'fits', datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard= tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1,write_graph=True)
```

Model-2 Architechture

```
#Model 2
from tensorflow.keras import regularizers
vocab_size= vocab_length + 1 # total number of words in our vocab


input = Input(shape=(X_train_padded_data.shape[1],))

embedding_layer = Embedding(vocab_size, 69, weights=[embedding_weights], input_length=2000, tr
layer1=Conv1D(128,kernel_size=2, activation='relu',padding='same',kernel_initializer='he_norma
layer2=Conv1D(128,kernel_size=2, activation='relu',padding='same',kernel_initializer='he_norma
layer3 = MaxPooling1D(pool_size = 2 , strides = 2)(layer2)
Dropout_layer1 = Dropout(0.3)(layer3)
layer4=Conv1D(128,kernel_size=2, activation='relu',padding='same',kernel_initializer='he_norma
layer5 = MaxPooling1D(pool_size = 2 , strides = 2)(layer4)
flatten_layer = Flatten()(layer5)
Dropout_layer2 = Dropout(0.3)(flatten_layer)

dense_layer1=Dense(128, activation='relu')(Dropout_layer2)
dense_layer2=Dense(128, activation='relu')(Dropout_layer2)
```

```
output_layer=Dense(20, activation='softmax')(dense_layer1)


model_2= keras.Model(input, output_layer)
model_2.summary()
```

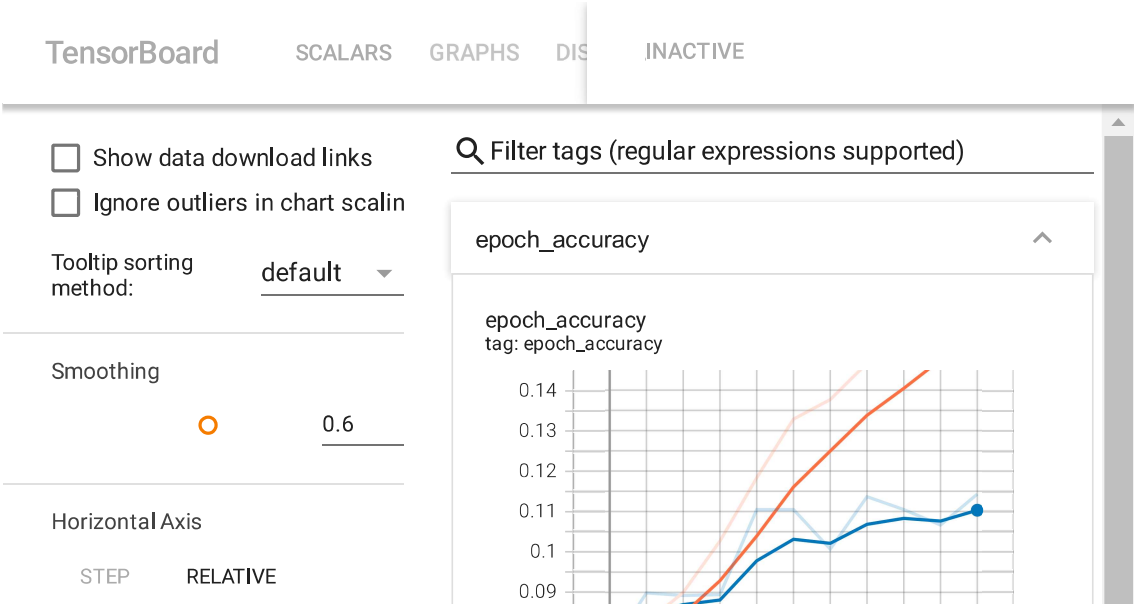
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 2000)]            0

 embedding (Embedding)       (None, 2000, 69)          4830

 conv1d (Conv1D)             (None, 2000, 128)         17792

 conv1d_1 (Conv1D)           (None, 2000, 128)         32896

 max_pooling1d (MaxPooling1D  (None, 1000, 128)        0
 )

 dropout (Dropout)           (None, 1000, 128)         0

 conv1d_2 (Conv1D)           (None, 1000, 128)         32896

 max_pooling1d_1 (MaxPooling  (None, 500, 128)         0
 1D)

 flatten (Flatten)           (None, 64000)             0

 dropout_1 (Dropout)         (None, 64000)             0

 dense (Dense)               (None, 128)               8192128

 dense_2 (Dense)             (None, 20)                2580

=================================================================
Total params: 8,283,122
Trainable params: 8,278,292
Non-trainable params: 4,830
_________________________________________________________________
```

```
model_2.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')
model_2.fit(X_train_padded_data, train_label ,validation_data=(X_test_padded_data, test_label)
```

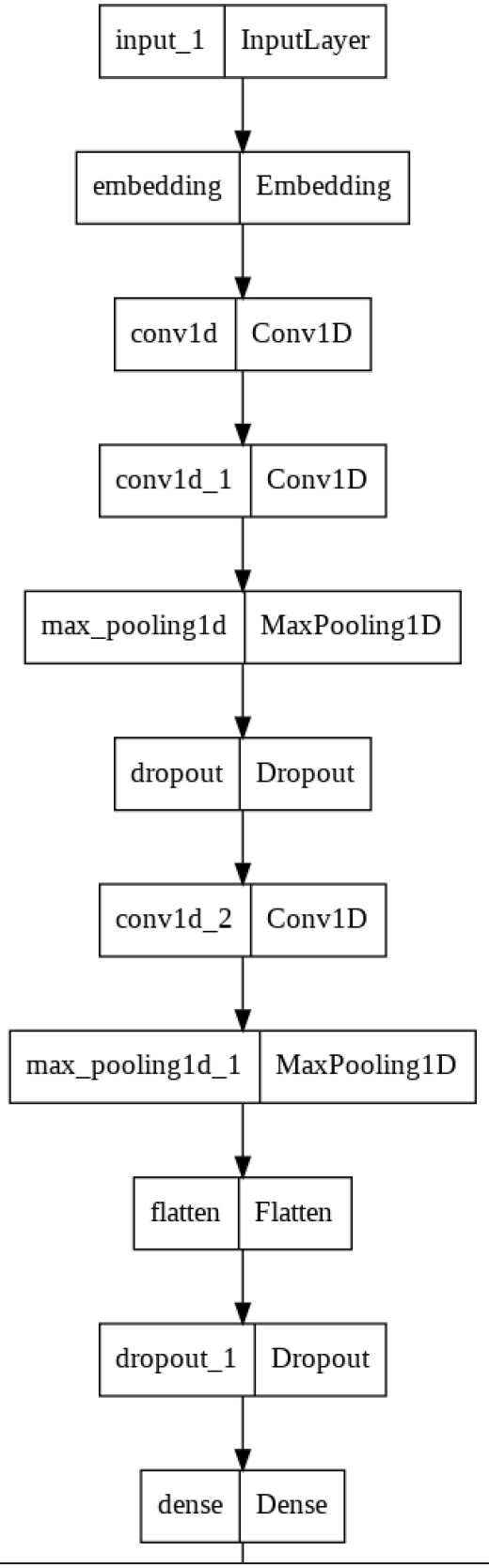
```
Epoch 1/11
111/111 [==============================] - ETA: 0s - loss: 7.5299 - accuracy: 0.0729 f

Epoch 1: val_loss improved from inf to 5.04627, saving model to model_save/best_model_
111/111 [==============================] - 27s 142ms/step - loss: 7.5299 - accuracy: 0
Epoch 2/11
110/111 [============================>.] - ETA: 0s - loss: 4.0956 - accuracy: 0.0826 f

Epoch 2: val_loss improved from 5.04627 to 3.48179, saving model to model_save/best_mc
111/111 [==============================] - 15s 135ms/step - loss: 4.0937 - accuracy: 0
Epoch 3/11
110/111 [============================>.] - ETA: 0s - loss: 3.2390 - accuracy: 0.0899 f

Epoch 3: val_loss improved from 3.48179 to 3.08760, saving model to model_save/best_mc
111/111 [==============================] - 15s 136ms/step - loss: 3.2387 - accuracy: 0
Epoch 4/11
110/111 [============================>.] - ETA: 0s - loss: 3.0036 - accuracy: 0.1028 f

Epoch 4: val_loss improved from 3.08760 to 2.97834, saving model to model_save/best_mc
```

```
111/111 [==============================] - 14s 130ms/step - loss: 3.0037 - accuracy: 0
Epoch 5/11
110/111 [============================>.] - ETA: 0s - loss: 2.9173 - accuracy: 0.1183 f

Epoch 5: val_loss improved from 2.97834 to 2.92633, saving model to model_save/best_mc
111/111 [==============================] - 14s 127ms/step - loss: 2.9171 - accuracy: 0
Epoch 6/11
110/111 [============================>.] - ETA: 0s - loss: 2.8738 - accuracy: 0.1328 f

Epoch 6: val_loss improved from 2.92633 to 2.91281, saving model to model_save/best_mc
111/111 [==============================] - 15s 137ms/step - loss: 2.8736 - accuracy: 0
Epoch 7/11
110/111 [============================>.] - ETA: 0s - loss: 2.8426 - accuracy: 0.1379 f

Epoch 7: val_loss did not improve from 2.91281
111/111 [==============================] - 14s 126ms/step - loss: 2.8428 - accuracy: 0
Epoch 8/11
110/111 [============================>.] - ETA: 0s - loss: 2.8247 - accuracy: 0.1467 f

Epoch 8: val_loss did not improve from 2.91281
111/111 [==============================] - 14s 127ms/step - loss: 2.8248 - accuracy: 0
Epoch 9/11
110/111 [============================>.] - ETA: 0s - loss: 2.8086 - accuracy: 0.1504 f

Epoch 9: val_loss did not improve from 2.91281
111/111 [==============================] - 14s 129ms/step - loss: 2.8089 - accuracy: 0
Epoch 10/11
110/111 [============================>.] - ETA: 0s - loss: 2.7815 - accuracy: 0.1580 f

Epoch 10: val_loss did not improve from 2.91281
111/111 [==============================] - 15s 137ms/step - loss: 2.7813 - accuracy: 0
Epoch 11/11
110/111 [============================>.] - ETA: 0s - loss: 2.7648 - accuracy: 0.1678 f

Epoch 11: val_loss did not improve from 2.91281
111/111 [==============================] - 14s 126ms/step - loss: 2.7648 - accuracy: 0
<keras.callbacks.History at 0x7f2fd003a610>
```

```
%tensorboard --logdir logs/fits
```




```
# Plotting the image
tf.keras.utils.plot_model(
    model_2,
    to_file='Model1 Text Classification.png',
    show_shapes=False,
    show_dtype=False,
    show_layer_names=True,
    rankdir='TB',
    expand_nested=False,
    dpi=96,
    layer_range=None,
    show_layer_activations=False
)
```



✓ 1s completed at 10:40 AM