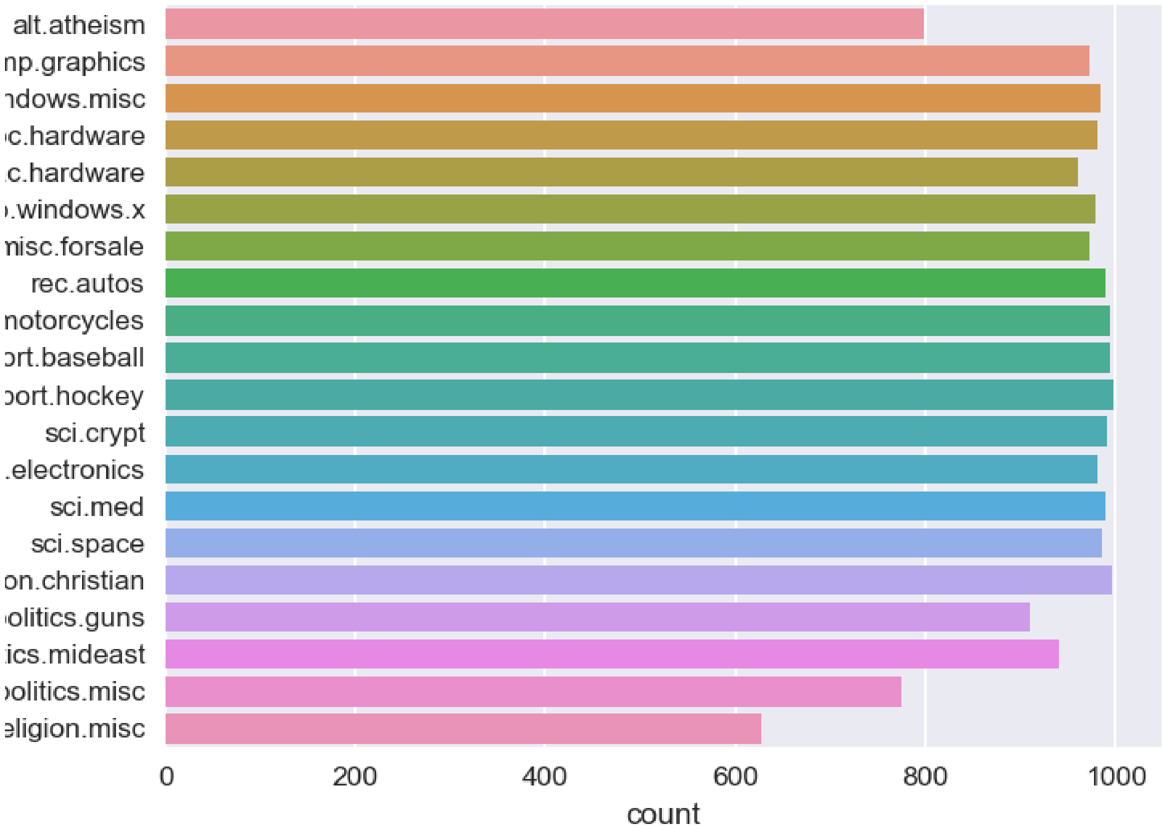
Text Classification:

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'ClassLabel_DocumentNumberInThatLabel'.
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.
```

In [ ]:

```
### 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 texts by '.'
after that remove the words whose length is less than or equal to 2 and also remove'com' word and then combine those words by space.
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]--> [dm1,dm2,dm3]-->"dm1 dm2 dm3"
append all those into one list/array. (This will give length of 18828 sentences i.e one list for each of the document).
Some sample output was shown below.

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

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

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

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

Out[ ]:
```
array(['juliet caltech edu',
       'coding bchs edu newsgate sps mot austlcm sps mot austlcm sps mot com  dna bchs edu',
       '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)
```

In [ ]:
```
len(preprocessed_email)
```

Out[ ]:
```
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 chars.
Eg: if we have sentance like "Subject: Re: Gospel Dating @ \r\r\n" --> You have to get "Gospel Dating"
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.edu >"

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 course that gets you HIRED"

> 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 will

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

In [ ]:
```
#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', 'IN'), ('the', 'DT'), Tree('GP
```

```
E', [('New', 'NNP'), ('York', 'NNP')])]

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

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

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 "Srikanth Varma" was referred as "PERSON".
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 ==> berlin) and
"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.

In [ ]:
```
data.columns
```

```
Index(['text', 'class', 'preprocessed_text', 'preprocessed_subject',
```

```
       'preprocessed_emails'],
      dtype='object')
```

In [ ]:
```python
data.iloc[400]
```

```
text                   From: arc1@ukc.ac.uk (Tony Curtis)\r\r\r\nSubj...
class                                                       alt.atheism
preprocessed_text      said re is article if followed the quoting rig...
preprocessed_subject                                  christian morality is
preprocessed_emails                                     ukc mac macalstr edu
Name: 567, dtype: object
```

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

In [ ]:
```python
def preprocess(Input_Text):
    """Do all the Preprocessing as shown above and
    return a tuple contain preprocess_email,preprocess_subject,preprocess_text for that Text_data"""
    return (list_of_preproessed_emails,subject,text)
```

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 column. use
that column to model.

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 the
previous assignments.

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 shown Below.
In the example we have considered d = 5, but in this assignment we will get d = dimension of Word vectors we are using.
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

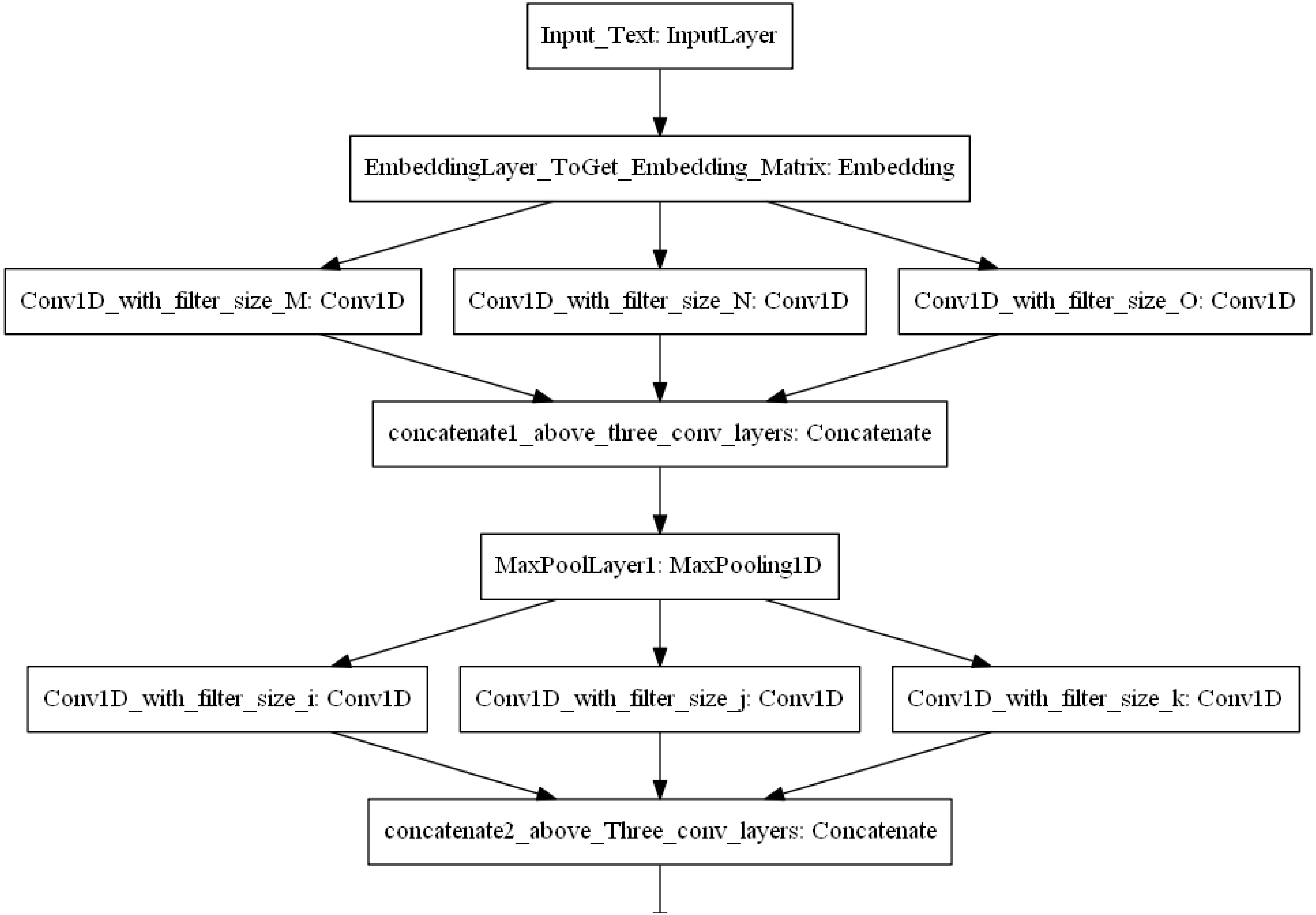
I	0.6	0.5	0.2	-0.1	0.4
like	0.8	0.9	0.1	0.5	0.1
this	0.4	0.6	0.1	-0.1	0.7
movie	...	...	...	...	...
very	...	...	...	...	...
much	...	...	...	...	...
!	...	...	...	...	...

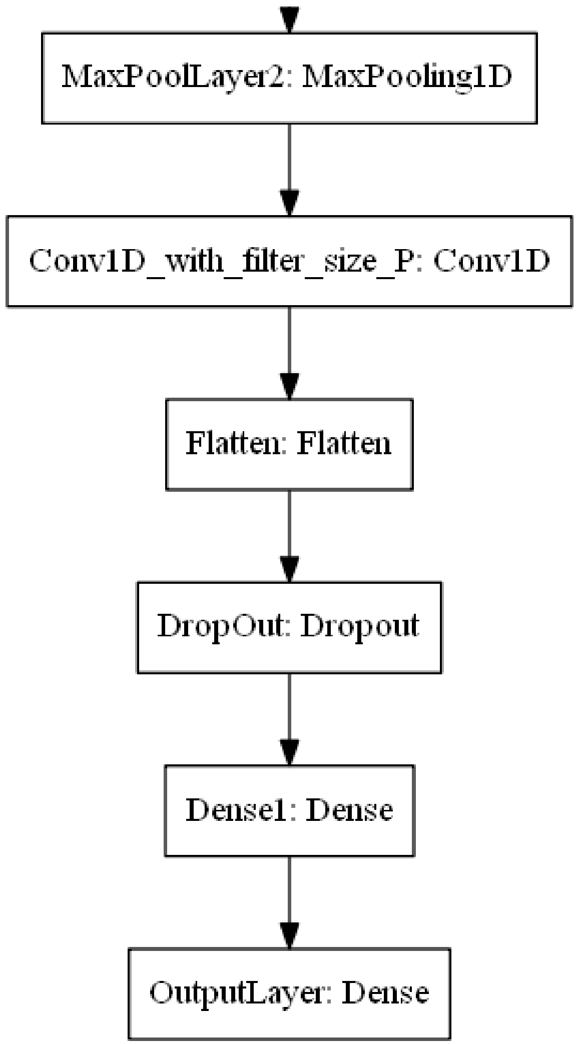
Ref: https://i.imgur.com/kiVQuk1.png

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

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/



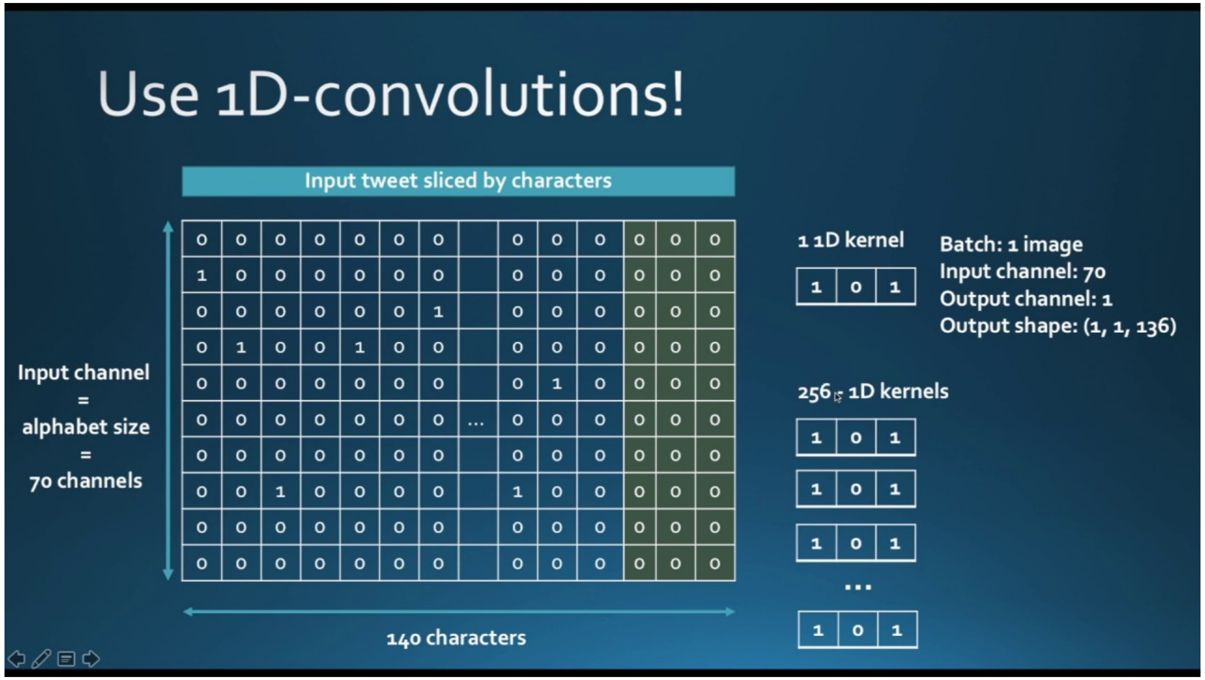


ref: 'https://i.imgur.com/fv1GvFJ.png'

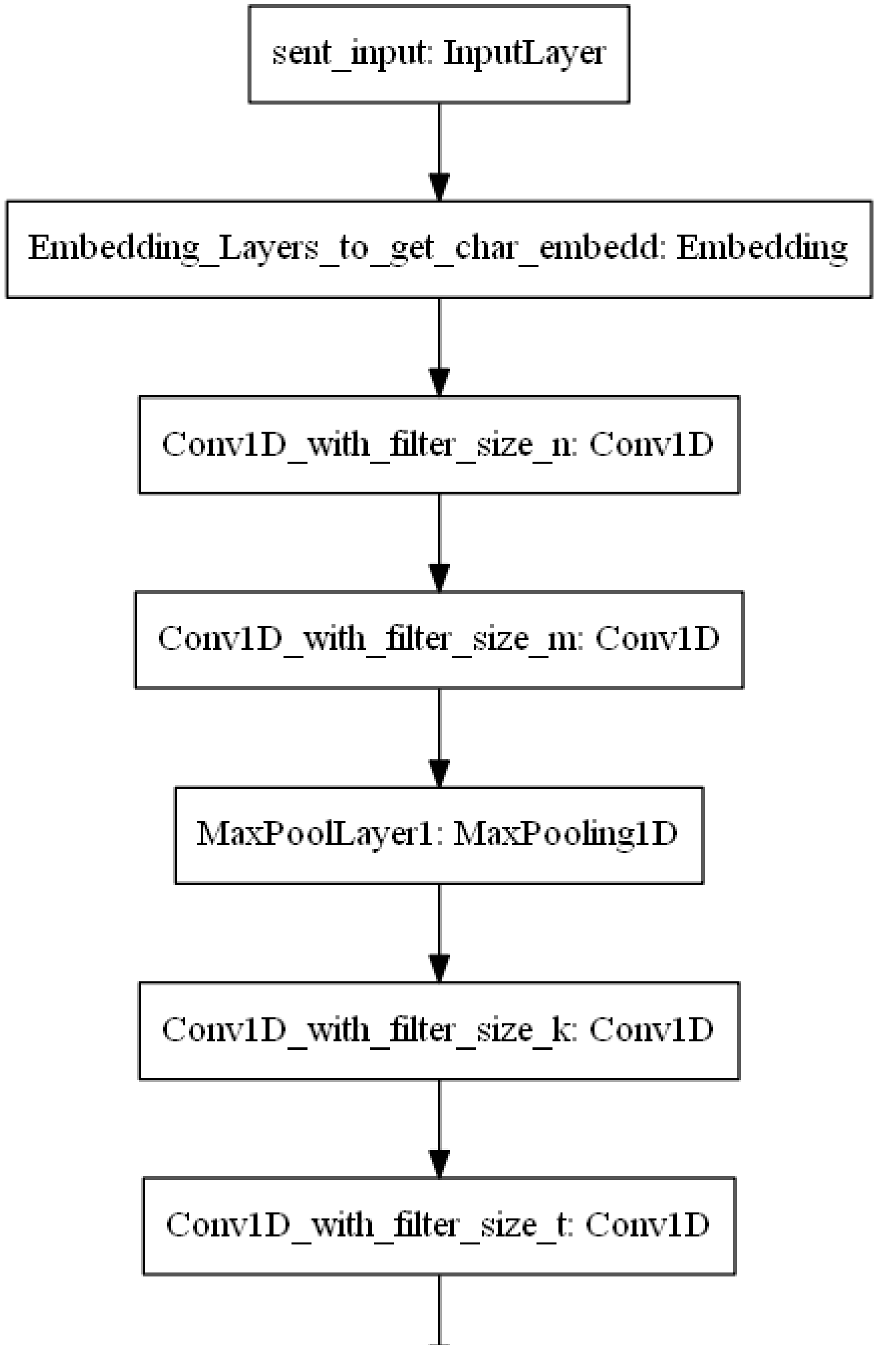
```
1. all are Conv1D layers with any number of filter and filter sizes, there is no restriction on this.

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

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

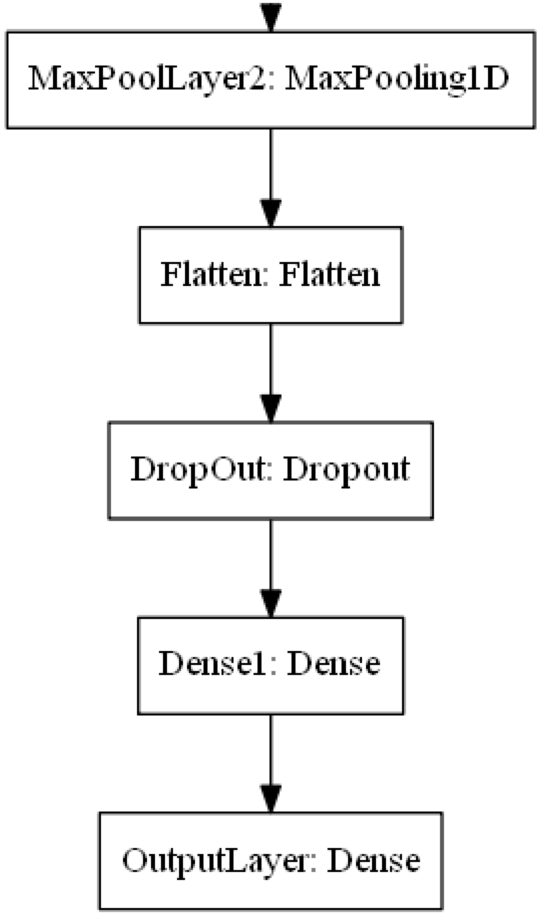
```
4. Don't use more than 16 filters in one Conv layer becuase it will increase the no of params.
( Only recommendation if you have less computing power )

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 Classification.NIPS 2015
2. Yoon Kim, Yacine Jernite, David Sontag, Alexander M. Rush. Character-Aware Neural Language Models. AAAI 2016
3. Shaojie Bai, J. Zico Kolter, Vladlen Koltun. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling
4. Use the pratrained char embeddings https://github.com/minimaxir/char-embeddings/blob/master/glove.840B.300d-char.txt


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



In [ ]:

```

```

In [ ]:

```
from nltk.chunk import ne_chunk
from nltk import pos_tag
from nltk.tokenize import word_tokenize
from nltk import Tree
from nltk import RegexpParser
import re
import numpy as np
import os
```

```
import pandas as pd
from tqdm import tqdm
```

Decontraction

In [ ]:
```
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
o="It's"
decontracted(o)
```

Out[ ]:
```
'It is'
```

In [ ]:
```
def email_extractor(input_text):
    pattern = re.compile("@")
    path="documents/"
    ppp=[]
    words=[]
    at_word=[]
    for line in open(input_text):
        for match in re.finditer(pattern, line):
            line = re.sub(r'\<','',line)
            line = re.sub(r'\>','',line)
            line = re.sub(r'\,','',line)
            words.extend(line.split())
    for word in words:
        if '@' in word:
```

```
            at_word.append(word)
    split_at_words=[]
    for word in at_word:
        split_at_words.append(word.split('@'))
    op=[]
    for i in split_at_words:
        op.append(i[-1])
    pp=[]

    for i in op:
        tt= i.split('.')
        for j in tt:
                if not j == 'com' and len(j)>2:
                    pp.append(j)
    ppp.append(pp)
    for ii in ppp:
        strng=" "
        tt=strng.join(ii)
        email=tt.lower() #email is list which contains preprocessed email
    return(email)
```

In [ ]:
```
def subject_extractor(input_text):
    #--------------SUBJECT EXTRACTION ----------------

    path="documents/"


    with open(input_text) as f:
        for j,line in enumerate(f):
            if 'Subject:' in line:
                line = re.sub(r'\w*:','',line)
                line = re.sub(r'\W',' ',line)
                line.strip()
                subject=line.lower() #subject is list which contains preprocessed subject
                break
    return(subject)
```

In [ ]:
```
def text_extractor(input_text):
    #--------------TEXT EXTRACTION ----------------
    with open(input_text) as fp:
```

```
        content=fp.read()
    with open(input_text) as f:
        content = decontracted(content)

        for j,line in enumerate(f):
            line = decontracted(line)

            if 'From:' in line:
                content=content.replace(line,"")
            if 'Subject:' in line:
                content=content.replace(line,"")
            if 'Write to:' in line:
                content=content.replace(line,"")
            if '@' in line:
                content=content.replace(line,"")

            words = line.split()
            for word in words:
                if word.endswith(':'):
                    content=content.replace(line,"")
                    break
        words = content.split()
        cleaned_words=[]
            #d_berlin ==> berlin ,de_berlin ==> berlin
        for ii in words:
            if '_' in ii:
                words_with_underscore = ii.split('_')
                if len(words_with_underscore[0])>2 and len(words_with_underscore[1])>2:
                    cleaned_words.append(ii)
                else:
                    for kk in words_with_underscore:
                        if len(kk)>2:
                            cleaned_words.append(kk)
            else:
                cleaned_words.append(ii)
        a=" "
        content=a.join(cleaned_words)
        content=re.sub("\d", "",content) # remove digits
        content = re.sub(r'\w*:',' ',content) # remove words ending with :
        content=re.sub("[\(\[].*?[\)\]]", " ", content) # remove words inside brackets
        content=re.sub("[\<\[].*?[\>\]]", "",content) # remove tags
        content = re.sub(r'\w\w_',' ',content)
```

```
        content = re.sub(r'\w_',' ',content)
        content = re.sub(r'[\n,\t,\\,\.,\-,\$,(,\",\',\<,\>,\/,\|]',' ',content) #remove special characters other tha
        content = content.lower() #converting to lowercase
        words1 = content.split()
            #Removing words with leength less than or equal to 2 or greater than or equal to 15
        cleaned_words=[]
        for i in words1:
            if len(i)>=2 and len(i)<=15:
                cleaned_words.append(i)
        a=' '
        content=a.join(cleaned_words)

    text= content#text is list which contains preprocessed text
    return(text)
```

Shallow Parsing

In [ ]:
```
def chunk_treat_names_places(text):
    token = list(ne_chunk(pos_tag(word_tokenize(text))))
    chunks = r"""Nouns : {<NNP><NN.*><.*>*<NN.*>}
                    }<VB.?|JJ.?|DT|IN|TO>+{"""
    chunkParser = RegexpParser(chunks)
    chunked = chunkParser.parse(token)
    for i in chunked:
        if type(i)==Tree:
            if i.label() == "GPE":
                j = i.leaves()
                if len(j)>1:   #if new_delhi or bigger name
                    gpe = "_".join([term for term,pos in j])
                    text  = re.sub(rf'{j[1][0]}',gpe,text, flags=re.MULTILINE)            #replacing delhi with new
                    text = re.sub(rf'\b{j[0][0]}\b',"",text, flags=re.MULTILINE)       #deleting new, \b is important
            if i.label()=="PERSON":            # deleting Ramesh
                for term,pog in i.leaves():
                    text = re.sub(term,"",text, flags=re.MULTILINE)

    return str(text)
```

In [ ]:
```
def preprocess(file):
    """Do all the Preprocessing as shown above and
    return a tuple contain preprocess_email,preprocess_subject,preprocess_text for that Text_data"""
```

```
    path="documents/"
    import os
    path = "documents/"
    if type(file) is list:
        text=[]
        pre_email=[]
        pre_subject=[]
        pre_text=[]
        n = len(file)
        clas=[]
        for i in tqdm(range(n)):
            with open(path+file[i]) as fp:
                text.append(fp.read())
            a = file[i].split('_')
            clas.append(a[0])
            pre_email.append(email_extractor(path+file[i]))
            pre_subject.append(subject_extractor(path+file[i]))
            text_pre=text_extractor(path+file[i])
            pre_text.append(chunk_treat_names_places(str(text_pre)))

    elif type(file) is str:
        with open(path+file) as fp:
            text=fp.read()
        a = file.split('_')
        clas = a[0]
        #print(type(pre_email))
        pre_email=str(email_extractor(path+file))
        pre_subject=subject_extractor(path+file)
        text_pre=text_extractor(path+file)
        pre_text=chunk_treat_names_places(str(text_pre))
    else:
        print("Give valid input to the function")
        exit()
    return(text,clas,pre_email,pre_subject,pre_text)
```

In [ ]:

```
path = "documents/"
files=os.listdir(path)
text,clas,preprocessed_email,preprocessed_subject,preprocessed_text = preprocess(files)
```

```
100%|██████████| 18828/18828 [44:51<00:00,  6.99it/s]
```

In [ ]:

```
path = "documents/"
```

```
files=os.listdir(path)
text,clas,preprocessed_email,preprocessed_subject,preprocessed_text = preprocess(files)
```

```
100%|██████████| 18828/18828 [50:23<00:00,  6.23it/s]
```

In [ ]:

```
import pandas as pd
df = pd.DataFrame()
df['text'] = text
df['class'] = clas
df['preprocessed email'] = preprocessed_email
df['preprocessed subject'] = preprocessed_subject
df['preprocessed text'] = preprocessed_text
df.head()
```

Out[ ]:

	text	class	preprocessed email	preprocessed subject	preprocessed text
0	From: mathew <mathew@mantis.co.uk>\nSubject: A...	alt.atheism	mantis netcom mantis	alt atheism atheist resources	atheist resources addresses of atheist organiz...
1	From: mathew <mathew@mantis.co.uk>\nSubject: A...	alt.atheism	mantis mantis mantis	alt atheism introduction to atheism	begin pgp signed message an introduction to at...
2	From: I3150101@dbstu1.rz.tu-bs.de (Benedikt Ro...	alt.atheism	dbstu1 tu-bs mimsy umd edu umd edu	gospel dating	well john has quite different not necessarily ...
3	From: mathew <mathew@mantis.co.uk>\nSubject: R...	alt.atheism	mantis kepler unh edu	university violating separation of church st...	recently ras have been ordered it is some sort...
4	From: strom@Watson.Ibm.Com (Rob Strom)\nSubjec...	alt.atheism	watson ibm com harder ccr-p ida org harder ccr...	soc motss et al princeton axes matching...	however hate economic terrorism and political ...

In [ ]:

```
df.to_csv("preprocessed.csv")
```

In [ ]:

```
text,classs,pre_email,pre_subject,pre_text=preprocess("alt.atheism_49960.txt")
```

In [ ]:

```
classs
```

Out[ ]:

```
'alt.atheism'
```

In [ ]:

```
pre_email
```

Out[ ]:

```
'mantis netcom mantis'
```

In [ ]:
```
pre_subject
```

Out[ ]:
```
' alt atheism  atheist resources '
```

In [ ]:
```
pre_text
```

Out[ ]:
```
'atheist resources addresses of atheist organizations usa freedom from religion foundation darwin fish bumper stick
ers and assorted other atheist paraphernalia are available from the freedom from religion foundation in the us evol
ution designs evolution designs sell the darwin fish it is fish symbol like the ones christians stick on their cars
but with feet and the word darwin written inside the deluxe moulded plastic fish is postpaid in the us ca people in
the san francisco bay area can get darwin fish from lynn gold price is per fish american atheist press aap publish
various atheist books critiques of the bible lists of the bible handbook by ball and foote american atheist press p
p isbn nd edition bible contradictions contradicts itself aap based on the king james version of the bible promethe
us books sell books including haught is holy horrors prometheus books glenn drive buffalo ny african americans for
humanism an organization promoting black secular humanism and uncovering the history of black freethought they publ
ish quarterly newsletter aah examiner buffalo ny united kingdom rationalist press association national secular soci
ety islington high street holloway road london ew london nl british humanist association south place ethical societ
y lamb is conduit passage conway hall london wcr rh red lion square london wcr rl fax the national secular society
publish the freethinker monthly magazine founded in germany ibka internationaler bund der und atheisten postfach be
rlin germany miz miz vertrieb postfach berlin germany ibdk internationaler ucherdienst der postfach hannover german
y books fiction thomas disch the santa claus compromise short story the ultimate proof that santa exists all charac
ters and events are fictitious any similarity to living or dead gods uh well walter miller jr canticle for leibowit
z one gem in this post atomic doomsday novel is the monks who spent their lives copying blueprints from saint leibo
witz filling the sheets of paper with ink and leaving white lines and letters edgar pangborn davy post atomic dooms
day novel set in clerical states the church for example forbids that anyone produce describe or use any substance c
ontaining atoms philip dick philip dick dick wrote many philosophical and thought provoking short stories and novel
s his stories are bizarre at times but very approachable he wrote mainly sf but he wrote about people truth and rel
igion rather than technology although he often believed that he had met some sort of god he galactic pot healer fal
lible alien deity summons group of earth craftsmen and women to remote planet to raise giant cathedral from beneath
the oceans when the deity begins to demand faith from the earthers pot healer joe fernwright is unable to comply po
lished ironic and amusing novel maze of death noteworthy for its description of technology based religion valis the
schizophrenic hero searches for the hidden mysteries of gnostic christianity after reality is fired into his brain
by pink laser beam of unknown but possibly divine origin he is accompanied by his dogmatic and dismissively atheist
friend and assorted other odd characters the divine invasion god invades earth by making young woman pregnant as sh
e returns from another star system unfortunately she is terminally ill and must be assisted by dead man whose brain
is wired to hour easy listening music margaret atwood the handmaid is tale story based on the premise that the us c
ongress is mysteriously assassinated and fundamentalists quickly take charge of the nation to set it right again th
e book is the diary of woman is life as she tries to live under the new christian theocracy women is right to own p
roperty is revoked and their bank accounts are closed; sinful luxuries are outlawed and the radio is only used for
readings from the bible crimes are punished hunted down and hanged atwood is writing style is difficult to get used
to at first but the tale grows more and more chilling as it goes on various authors the bible this somewhat dull an
d rambling work has often been criticized however it is probably worth reading if only so that you will know what a
```

```
ll the fuss is about it exists in many different versions so make sure you get the one true version books non ficti
on peter de rosa vicars of christ bantam press although de rosa seems to be christian or even catholic this is very
enlighting history of papal immoralities adulteries fallacies etc droemer knaur michael martin philadelphia usa det
ailed and scholarly justification of atheism contains an outstanding appendix defining terminology and usage in thi
s tendentious area argues both for negative atheism and also for positive atheism includes great refutations of the
most challenging arguments for god; particular attention is paid to refuting contempory theists such as platinga an
d swinburne pages isbn the case against christianity temple university press comprehensive critique of christianity
in which he considers the best contemporary defences of christianity and demonstrates that they are unsupportable a
nd or incoherent pages isbn james turner without god without creed the johns hopkins university press baltimore md
usa subtitled the origins of unbelief in america examines the way in which unbelief became mainstream alternative w
orld view focusses on the period and while considering france and britain the emphasis is on american and particula
rly new england developments neither religious history of secularization or atheism without god without creed is ra
ther the intellectual history of the fate of single idea the belief that god exists pages isbn george seldes the gr
eat thoughts ballantine books new york usa dictionary of quotations of different kind concentrating on statements a
nd writings which explicitly or implicitly present the person is philosophy and world view includes obscure opinion
s from many people for some popular observations traces the way in which various people expressed and twisted the i
dea over the centuries quite number of the quotations are derived from cardiff is what great men think of religion
and noyes views of religion pages isbn richard swinburne the existence of god clarendon paperbacks oxford this book
is the second volume in trilogy that began with the coherence of theism and was concluded with faith and reason in
this work swinburne attempts to construct series of inductive arguments for the existence of god his arguments whic
h are somewhat tendentious and rely upon the imputation of late th century western christian values and aesthetics
to god which is supposedly as simple as can be conceived were decisively rejected in mackie is the miracle of theis
m in the revised edition of the existence of god swinburne includes an appendix in which he makes somewhat incohere
nt attempt to rebut mackie mackie the miracle of theism oxford this volume contains comprehensive review of the pri
ncipal arguments for and against the existence of god it ranges from the classical philosophical positions of desca
rtes anselm berkeley hume et al through the moral arguments of newman kant and sidgwick to the recent restatements
of the classical theses by plantinga and swinburne it also addresses those positions which push the concept of god
beyond the realm of the rational such as those of kierkegaard kung and philips as well as replacements for god such
as lelie is axiarchism the book is delight to read less formalistic and better written than martin is works and ref
reshingly direct when compared with the hand waving of swinburne james haught prometheus books looks at religious p
ersecution from ancient times to the present day and not only by christians library of congress catalog card number
norm allen jr see the listing for african americans for humanism above gordon stein an anthology of atheism and rat
ionalism prometheus books an anthology covering wide range of subjects including the devil evil and morality and th
e history of freethought comprehensive bibliography edmund cohen the mind of the bible believer prometheus books st
udy of why people become christian fundamentalists and what effect it has on them net resources there is small mail
based archive server at mantis co uk which carries archives of old alt atheism moderated articles and assorted othe
r files for help send atheism index and it will mail back reply mathew'
```

In [1]:
```python
from google.colab import drive
drive.mount("/content/drive/")
```

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

In [7]:
```python
import numpy as np
import scipy
```

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import roc_auc_score,roc_curve,auc
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.preprocessing import OneHotEncoder
from tensorflow.keras.preprocessing.text import Tokenizer
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

In [3]:
```python
import pandas as pd
data = pd.read_csv("/content/drive/MyDrive/preprocessed.csv")
```

In [4]:
```python
data=data.drop('Unnamed: 0',axis=1)
data.head()
```

Out[4]:

	text	class	preprocessed email	preprocessed subject	preprocessed text
0	From: mathew <mathew@mantis.co.uk>\nSubject: A...	alt.atheism	mantis netcom mantis	alt atheism atheist resources	atheist resources addresses of atheist organiz...
1	From: mathew <mathew@mantis.co.uk>\nSubject: A...	alt.atheism	mantis mantis mantis	alt atheism introduction to atheism	begin pgp signed message an introduction to at...
2	From: I3150101@dbstu1.rz.tu-bs.de (Benedikt Ro...	alt.atheism	dbstu1 tu-bs mimsy umd edu umd edu	gospel dating	well john has quite different not necessarily ...
3	From: mathew <mathew@mantis.co.uk>\nSubject: R...	alt.atheism	mantis kepler unh edu	university violating separation of church st...	recently ras have been ordered it is some sort...
4	From: strom@Watson.Ibm.Com (Rob Strom)\nSubjec...	alt.atheism	watson ibm com harder ccr-p ida org harder ccr...	soc motss et al princeton axes matching...	however hate economic terrorism and political ...

In [5]:
```python
combined_columns=[]
```

```
from tqdm import tqdm
for i in tqdm(range(data.shape[0])):
  combined_columns.append(str(data['preprocessed email'][i])+str(data['preprocessed subject'][i])+str(data['preproces

combined_columns = pd.DataFrame(combined_columns)
```

```
100%|██████████| 18828/18828 [00:00<00:00, 52512.30it/s]
```

In [ ]:
```
y = data['class']
X = combined_columns

# train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,stratify=y)
#X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

In [ ]:
```
from nltk.tokenize import word_tokenize
from sklearn.preprocessing import OneHotEncoder
import pickle
import nltk
nltk.download('punkt')
train = X_train[0]
train = train.apply(word_tokenize)


tokenizer = Tokenizer(num_words= 10000)
tokenizer = Tokenizer(num_words= 10000)
tokenizer.fit_on_texts(train)
sequences = tokenizer.texts_to_sequences(train)
length_sequences = list()


for i in sequences:
    length_sequences.append(len(i))
max_length = 1000


from tensorflow.keras.preprocessing.sequence import pad_sequences
train = pad_sequences(sequences,maxlen = max_length,padding='post')
vocab_length_train = len(tokenizer.word_index)

with open('/content/drive/MyDrive/data/glove_vectors', 'rb') as f:
    model = pickle.load(f)
```

```
    glove_words =  set(model.keys())
total_words = vocab_length_train + 1
skipped_words = 0
embedding_dim = 300
embedding_matrix = np.zeros((total_words, embedding_dim))
for word, index in tokenizer.word_index.items():
    try:
        embedding_vector = model[word]
    except:
        skipped_words = skipped_words+1
    pass
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector



encoder = OneHotEncoder()
y_train_ohe = encoder.fit_transform(np.array(y_train).reshape(-1,1))
y_test_ohe = encoder.fit_transform(np.array(y_test).reshape(-1,1))
y_train_ohe1 = scipy.sparse.csr_matrix.todense(y_train_ohe)
y_test_ohe1 = scipy.sparse.csr_matrix.todense(y_test_ohe)
val = X_test[0]
val = val.apply(word_tokenize)


sequences_val = tokenizer.texts_to_sequences(val)
val = pad_sequences(sequences_val,maxlen = max_length,padding='post')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [ ]:

```
vocab_length_train=len(tokenizer.word_index)
text_input= tf.keras.Input(shape=(max_length,),dtype='int32',name="input_seq_total_text_data")
text_embedding = layers.Embedding(input_dim = len(embedding_matrix), output_dim = 300, input_length=max_length,weight

conv1_m = layers.Conv1D(16,3,activation='relu')(text_embedding)
conv1_n = layers.Conv1D(14,3,activation='relu')(text_embedding)
conv1_o = layers.Conv1D(12,3,activation='relu')(text_embedding)

x = layers.concatenate([conv1_m,conv1_n,conv1_o])
x = layers.MaxPool1D(2)(x)
```

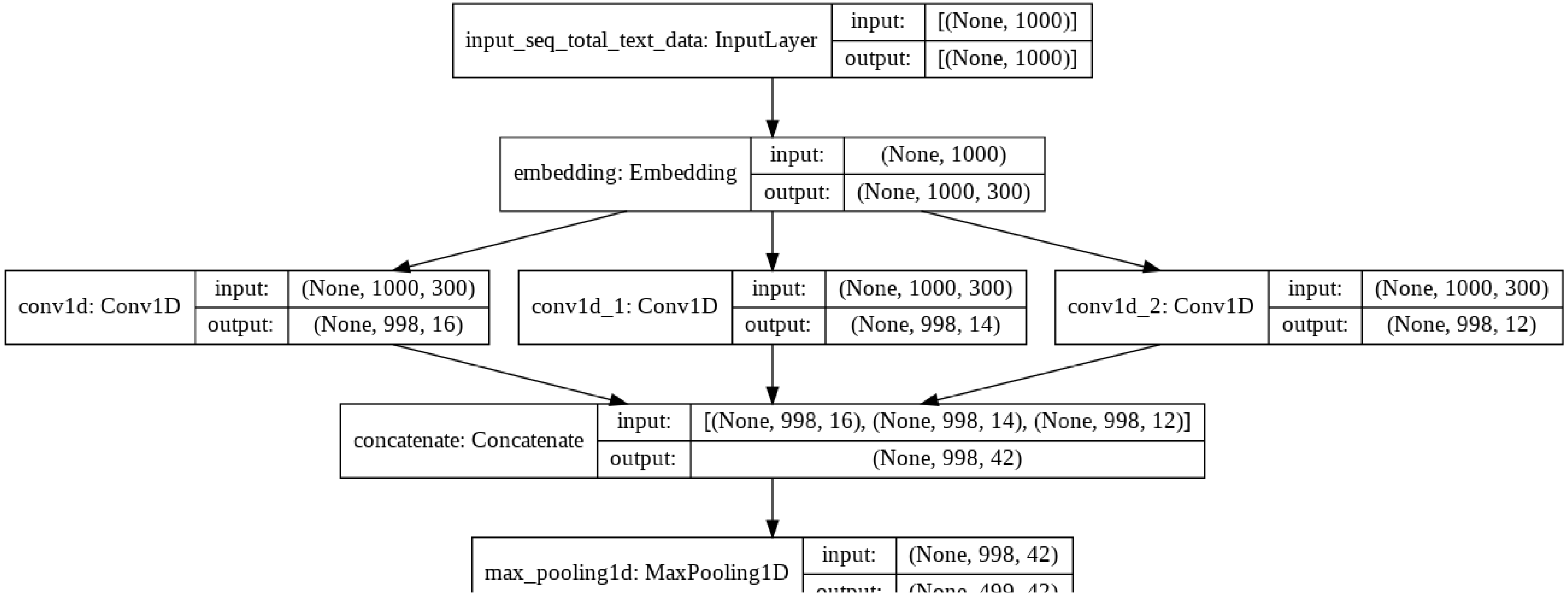
```
conv1_i = layers.Conv1D(15,3,activation='relu')(x)
conv1_j = layers.Conv1D(13,3,activation='relu')(x)
conv1_k = layers.Conv1D(11,3,activation='relu')(x)

x = layers.concatenate([conv1_i,conv1_j,conv1_k])
x = layers.MaxPool1D()(x)
x = layers.Conv1D(16,8,activation='relu')(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(128)(x)
x = layers.Dense(20,activation='softmax',name="output")(x)
model = keras.Model(
    inputs=[text_input],
    outputs=[x],
)
model.summary()
```

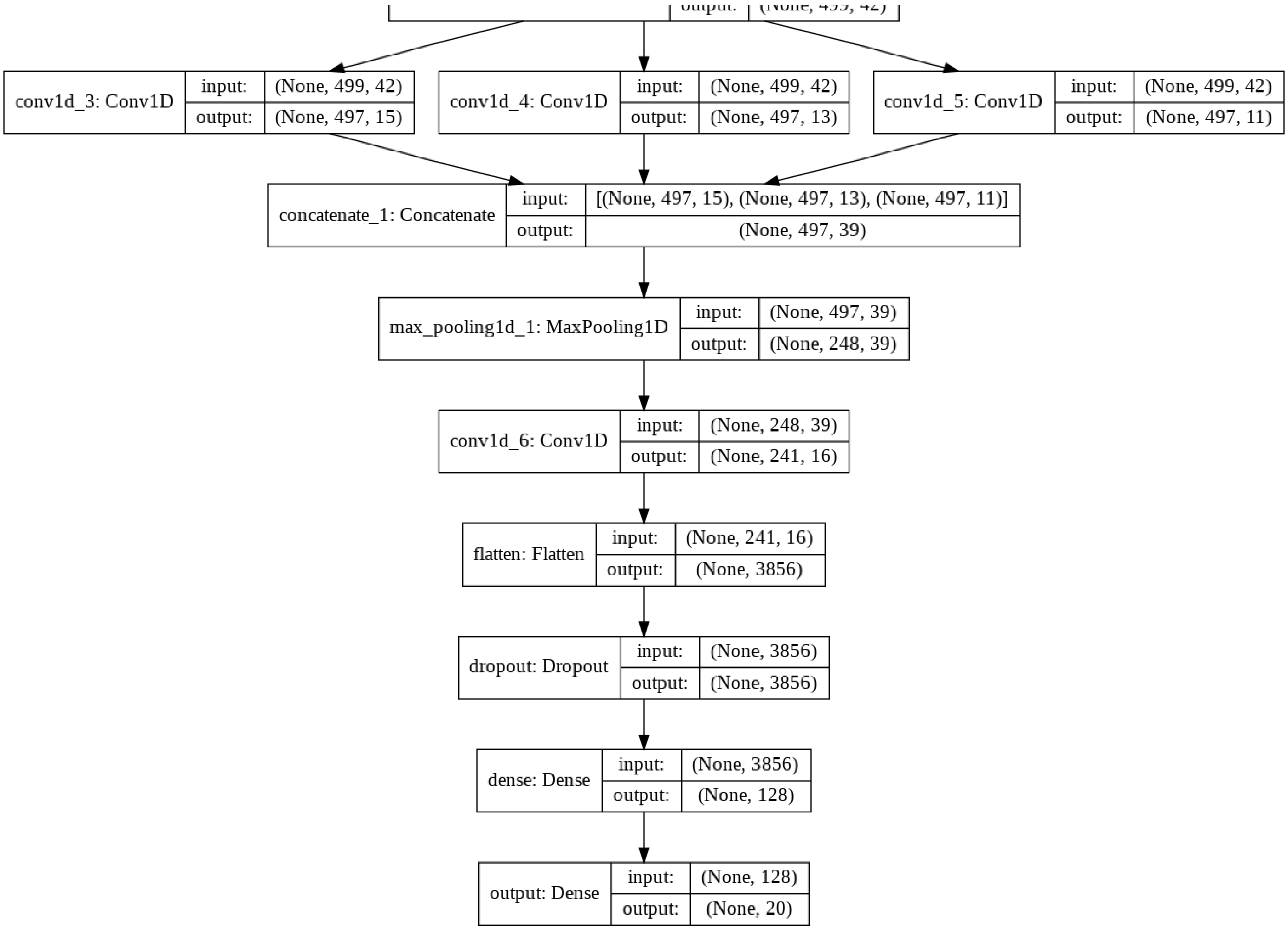
```
Model: "model"

________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================================
input_seq_total_text_data (Inpu [(None, 1000)]       0

________________________________________________________________________________________________
embedding (Embedding)           (None, 1000, 300)    26987700    input_seq_total_text_data[0][0]

________________________________________________________________________________________________
conv1d (Conv1D)                 (None, 998, 16)      14416       embedding[0][0]

________________________________________________________________________________________________
conv1d_1 (Conv1D)               (None, 998, 14)      12614       embedding[0][0]

________________________________________________________________________________________________
conv1d_2 (Conv1D)               (None, 998, 12)      10812       embedding[0][0]

________________________________________________________________________________________________
concatenate (Concatenate)       (None, 998, 42)      0           conv1d[0][0]
                                                                 conv1d_1[0][0]
                                                                 conv1d_2[0][0]

________________________________________________________________________________________________
max_pooling1d (MaxPooling1D)    (None, 499, 42)      0           concatenate[0][0]

________________________________________________________________________________________________
conv1d_3 (Conv1D)               (None, 497, 15)      1905        max_pooling1d[0][0]

________________________________________________________________________________________________
conv1d_4 (Conv1D)               (None, 497, 13)      1651        max_pooling1d[0][0]

________________________________________________________________________________________________
conv1d_5 (Conv1D)               (None, 497, 11)      1397        max_pooling1d[0][0]

________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 497, 39)      0           conv1d_3[0][0]
```

```
                                                                 conv1d_4[0][0]
                                                                 conv1d_5[0][0]
__________________________________________________________________________________________________
max_pooling1d_1 (MaxPooling1D)  (None, 248, 39)      0           concatenate_1[0][0]
__________________________________________________________________________________________________
conv1d_6 (Conv1D)               (None, 241, 16)      5008        max_pooling1d_1[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 3856)         0           conv1d_6[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 3856)         0           flatten[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 128)          493696      dropout[0][0]
__________________________________________________________________________________________________
output (Dense)                  (None, 20)           2580        dense[0][0]
==================================================================================================
Total params: 27,531,779
Trainable params: 544,079
Non-trainable params: 26,987,700
__________________________________________________________________________________________________
```

In [ ]:
```python
keras.utils.plot_model(model,show_shapes=True)
```

Out[ ]:






```
from sklearn import preprocessing
```

In [ ]:
```
label_encoder = preprocessing.LabelEncoder()
y_train_enc= label_encoder.fit_transform(y_train)
y_test_enc= label_encoder.transform(y_test)

from keras.utils import np_utils
ytr = np_utils.to_categorical(y_train_enc,20)
yte = np_utils.to_categorical(y_test_enc,20)
```

In [ ]:
```
#https://towardsdatascience.com/neural-network-with-tensorflow-how-to-stop-training-using-callback-5c8d575c18a9
class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('val_accuracy') > 0.70):
      print("\nReached 70+ validation accuracy, so stopping training!!" )
      self.model.stop_training = True
```

In [ ]:
```
from sklearn.metrics import f1_score
class metrics(tf.keras.callbacks.Callback):
  def __init__(self,validationx,valy):
    super(metrics, self).__init__()
    self.validationx = validationx
    self.valy = valy

  def on_epoch_end(self, epoch, logs={}):
    y_pred = (np.asarray(self.model.predict(self.validationx))).round()
    y_true = (np.squeeze(np.asarray(self.valy)))
    val_f1 = f1_score(y_true, y_pred,average='samples')
    print("F1 Score : "+str(val_f1))
    return
```

In [ ]:
```
import datetime
import os
os.mkdir("/content/m1logss")
log_dir = "/content/m1logss/model_log"+datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1)
```

In [ ]:
```
#compile
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.001,momentum=0.9),loss='categorical_crossentropy',met

train_input = train
val_input = val
```

```
train_output = ytr
val_output = yte
metrics_callback = metrics(val_input,yte)

#train
tf.keras.backend.clear_session()
history=model.fit(train_input,train_output,batch_size=100,epochs=100,validation_data=(val_input,yte),callbacks=[myCal
```

```
Epoch 1/100
142/142 [==============================] - 11s 70ms/step - loss: 3.0547 - accuracy: 0.0611 - val_loss: 2.9516 - val_a
ccuracy: 0.0822
F1 Score : 0.0
Epoch 2/100
142/142 [==============================] - 10s 69ms/step - loss: 2.9513 - accuracy: 0.0788 - val_loss: 2.9165 - val_a
ccuracy: 0.0967
F1 Score : 0.0
Epoch 3/100
142/142 [==============================] - 10s 69ms/step - loss: 2.9028 - accuracy: 0.1009 - val_loss: 2.7861 - val_a
ccuracy: 0.1321
F1 Score : 0.0006373486297004461
Epoch 4/100
142/142 [==============================] - 10s 69ms/step - loss: 2.7539 - accuracy: 0.1226 - val_loss: 2.5392 - val_a
ccuracy: 0.1557
F1 Score : 0.0019120458891013384
Epoch 5/100
142/142 [==============================] - 10s 69ms/step - loss: 2.5421 - accuracy: 0.1584 - val_loss: 2.4005 - val_a
ccuracy: 0.1961
F1 Score : 0.005948587210537497
Epoch 6/100
142/142 [==============================] - 10s 70ms/step - loss: 2.4258 - accuracy: 0.1856 - val_loss: 2.2712 - val_a
ccuracy: 0.2243
F1 Score : 0.008922880815806247
Epoch 7/100
142/142 [==============================] - 10s 70ms/step - loss: 2.2827 - accuracy: 0.2097 - val_loss: 2.2570 - val_a
ccuracy: 0.2243
F1 Score : 0.017420862545145528
Epoch 8/100
142/142 [==============================] - 10s 70ms/step - loss: 2.1993 - accuracy: 0.2429 - val_loss: 2.0978 - val_a
ccuracy: 0.2836
F1 Score : 0.034629275547057574
Epoch 9/100
142/142 [==============================] - 10s 70ms/step - loss: 2.0748 - accuracy: 0.2792 - val_loss: 2.0030 - val_a
ccuracy: 0.2981
```

```
F1 Score : 0.06904610155088167
Epoch 10/100
142/142 [==============================] - 10s 70ms/step - loss: 2.0382 - accuracy: 0.2875 - val_loss: 1.8438 - val_a
ccuracy: 0.3639
F1 Score : 0.09092840450393032
Epoch 11/100
142/142 [==============================] - 10s 70ms/step - loss: 1.8712 - accuracy: 0.3516 - val_loss: 1.8523 - val_a
ccuracy: 0.3546
F1 Score : 0.10261312938177183
Epoch 12/100
142/142 [==============================] - 10s 70ms/step - loss: 1.8040 - accuracy: 0.3664 - val_loss: 1.7703 - val_a
ccuracy: 0.3786
F1 Score : 0.11026131293817719
Epoch 13/100
142/142 [==============================] - 10s 70ms/step - loss: 1.6963 - accuracy: 0.3989 - val_loss: 1.8417 - val_a
ccuracy: 0.3627
F1 Score : 0.1504142766093053
Epoch 14/100
142/142 [==============================] - 10s 70ms/step - loss: 1.6624 - accuracy: 0.4121 - val_loss: 1.7652 - val_a
ccuracy: 0.3871
F1 Score : 0.14807733163373699
Epoch 15/100
142/142 [==============================] - 10s 70ms/step - loss: 1.6100 - accuracy: 0.4293 - val_loss: 1.5811 - val_a
ccuracy: 0.4413
F1 Score : 0.18908009347779903
Epoch 16/100
142/142 [==============================] - 10s 70ms/step - loss: 1.6075 - accuracy: 0.4326 - val_loss: 1.5447 - val_a
ccuracy: 0.4546
F1 Score : 0.21691098364138517
Epoch 17/100
142/142 [==============================] - 10s 70ms/step - loss: 1.5036 - accuracy: 0.4678 - val_loss: 1.5203 - val_a
ccuracy: 0.4619
F1 Score : 0.22073507541958784
Epoch 18/100
142/142 [==============================] - 10s 70ms/step - loss: 1.4739 - accuracy: 0.4835 - val_loss: 1.6174 - val_a
ccuracy: 0.4364
F1 Score : 0.24452942426173785
Epoch 19/100
142/142 [==============================] - 10s 71ms/step - loss: 1.4705 - accuracy: 0.4798 - val_loss: 1.4412 - val_a
ccuracy: 0.4942
F1 Score : 0.28616953473550033
Epoch 20/100
142/142 [==============================] - 10s 70ms/step - loss: 1.3863 - accuracy: 0.5150 - val_loss: 1.3892 - val_a
ccuracy: 0.5158
F1 Score : 0.30592734225621415
```

```
Epoch 21/100
142/142 [==============================] - 10s 70ms/step - loss: 1.3592 - accuracy: 0.5196 - val_loss: 1.3914 - val_a
ccuracy: 0.5167
F1 Score : 0.2948799660080731
Epoch 22/100
142/142 [==============================] - 10s 71ms/step - loss: 1.3082 - accuracy: 0.5380 - val_loss: 1.3949 - val_a
ccuracy: 0.5139
F1 Score : 0.3322710856171659
Epoch 23/100
142/142 [==============================] - 10s 70ms/step - loss: 1.2822 - accuracy: 0.5440 - val_loss: 1.3479 - val_a
ccuracy: 0.5390
F1 Score : 0.3511790949649458
Epoch 24/100
142/142 [==============================] - 10s 70ms/step - loss: 1.2753 - accuracy: 0.5563 - val_loss: 1.3382 - val_a
ccuracy: 0.5430
F1 Score : 0.3577650308051838
Epoch 25/100
142/142 [==============================] - 10s 70ms/step - loss: 1.2570 - accuracy: 0.5560 - val_loss: 1.3629 - val_a
ccuracy: 0.5358
F1 Score : 0.3496919481623115
Epoch 26/100
142/142 [==============================] - 10s 70ms/step - loss: 1.2313 - accuracy: 0.5767 - val_loss: 1.3347 - val_a
ccuracy: 0.5366
F1 Score : 0.37327384746122794
Epoch 27/100
142/142 [==============================] - 10s 70ms/step - loss: 1.1795 - accuracy: 0.5921 - val_loss: 1.2639 - val_a
ccuracy: 0.5721
F1 Score : 0.39048226046314
Epoch 28/100
142/142 [==============================] - 10s 70ms/step - loss: 1.1413 - accuracy: 0.6046 - val_loss: 1.2833 - val_a
ccuracy: 0.5658
F1 Score : 0.38899511366050565
Epoch 29/100
142/142 [==============================] - 10s 71ms/step - loss: 1.1427 - accuracy: 0.6028 - val_loss: 1.2711 - val_a
ccuracy: 0.5711
F1 Score : 0.41002761844062036
Epoch 30/100
142/142 [==============================] - 10s 71ms/step - loss: 1.1169 - accuracy: 0.6189 - val_loss: 1.1987 - val_a
ccuracy: 0.6004
F1 Score : 0.4393456554068409
Epoch 31/100
142/142 [==============================] - 10s 70ms/step - loss: 1.0729 - accuracy: 0.6301 - val_loss: 1.1778 - val_a
ccuracy: 0.6063
F1 Score : 0.4506054811982154
Epoch 32/100
```

```
142/142 [==============================] - 10s 70ms/step - loss: 1.0626 - accuracy: 0.6322 - val_loss: 1.1909 - val_a
ccuracy: 0.6023
F1 Score : 0.44933078393881454
Epoch 33/100
142/142 [==============================] - 10s 71ms/step - loss: 1.0353 - accuracy: 0.6443 - val_loss: 1.1576 - val_a
ccuracy: 0.6131
F1 Score : 0.4627151051625239
Epoch 34/100
142/142 [==============================] - 10s 71ms/step - loss: 1.0126 - accuracy: 0.6516 - val_loss: 1.2302 - val_a
ccuracy: 0.5893
F1 Score : 0.4518801784576163
Epoch 35/100
142/142 [==============================] - 10s 70ms/step - loss: 1.0406 - accuracy: 0.6443 - val_loss: 1.1321 - val_a
ccuracy: 0.6286
F1 Score : 0.4731251327809645
Epoch 36/100
142/142 [==============================] - 10s 70ms/step - loss: 0.9936 - accuracy: 0.6598 - val_loss: 1.1285 - val_a
ccuracy: 0.6286
F1 Score : 0.4818355640535373
Epoch 37/100
142/142 [==============================] - 10s 70ms/step - loss: 0.9862 - accuracy: 0.6583 - val_loss: 1.1562 - val_a
ccuracy: 0.6231
F1 Score : 0.4894837476099426
Epoch 38/100
142/142 [==============================] - 10s 70ms/step - loss: 0.9690 - accuracy: 0.6686 - val_loss: 1.1160 - val_a
ccuracy: 0.6359
F1 Score : 0.4894837476099426
Epoch 39/100
142/142 [==============================] - 10s 70ms/step - loss: 0.9548 - accuracy: 0.6728 - val_loss: 1.0976 - val_a
ccuracy: 0.6365
F1 Score : 0.5058423624389208
Epoch 40/100
142/142 [==============================] - 10s 70ms/step - loss: 0.9356 - accuracy: 0.6836 - val_loss: 1.0731 - val_a
ccuracy: 0.6486
F1 Score : 0.5141278946250265
Epoch 41/100
142/142 [==============================] - 10s 70ms/step - loss: 0.9154 - accuracy: 0.6842 - val_loss: 1.1503 - val_a
ccuracy: 0.6231
F1 Score : 0.48226046314000426
Epoch 42/100
142/142 [==============================] - 10s 71ms/step - loss: 0.9276 - accuracy: 0.6889 - val_loss: 1.1077 - val_a
ccuracy: 0.6427
F1 Score : 0.5073295092415552
Epoch 43/100
142/142 [==============================] - 10s 70ms/step - loss: 0.9171 - accuracy: 0.6876 - val_loss: 1.0888 - val_a
```

```
ccuracy: 0.6452
F1 Score : 0.5302740599107711
Epoch 44/100
142/142 [==============================] - 10s 70ms/step - loss: 0.8742 - accuracy: 0.7029 - val_loss: 1.0704 - val_a
ccuracy: 0.6535
F1 Score : 0.5249628213299341
Epoch 45/100
142/142 [==============================] - 10s 70ms/step - loss: 0.8322 - accuracy: 0.7228 - val_loss: 1.0440 - val_a
ccuracy: 0.6586
F1 Score : 0.5400467388995114
Epoch 46/100
142/142 [==============================] - 10s 69ms/step - loss: 0.8540 - accuracy: 0.7109 - val_loss: 1.0733 - val_a
ccuracy: 0.6592
F1 Score : 0.5466326747397493
Epoch 47/100
142/142 [==============================] - 10s 69ms/step - loss: 0.8415 - accuracy: 0.7159 - val_loss: 1.1020 - val_a
ccuracy: 0.6603
F1 Score : 0.5485447206288506
Epoch 48/100
142/142 [==============================] - 10s 69ms/step - loss: 0.8059 - accuracy: 0.7300 - val_loss: 1.1312 - val_a
ccuracy: 0.6552
F1 Score : 0.5425961334183131
Epoch 49/100
142/142 [==============================] - 10s 69ms/step - loss: 0.8093 - accuracy: 0.7316 - val_loss: 1.1105 - val_a
ccuracy: 0.6554
F1 Score : 0.5364350966645421
Epoch 50/100
142/142 [==============================] - 10s 69ms/step - loss: 0.8159 - accuracy: 0.7281 - val_loss: 1.0537 - val_a
ccuracy: 0.6633
F1 Score : 0.5549182069258551
Epoch 51/100
142/142 [==============================] - 10s 69ms/step - loss: 0.7875 - accuracy: 0.7361 - val_loss: 1.0036 - val_a
ccuracy: 0.6900
F1 Score : 0.5753133630762693
Epoch 52/100
142/142 [==============================] - 10s 69ms/step - loss: 0.7546 - accuracy: 0.7431 - val_loss: 1.0171 - val_a
ccuracy: 0.6898
F1 Score : 0.5714892712980667
Epoch 53/100
142/142 [==============================] - 10s 69ms/step - loss: 0.7739 - accuracy: 0.7370 - val_loss: 1.0567 - val_a
ccuracy: 0.6788
F1 Score : 0.5687274272360314
Epoch 54/100
142/142 [==============================] - 10s 69ms/step - loss: 0.7423 - accuracy: 0.7561 - val_loss: 1.1212 - val_a
ccuracy: 0.6616
```

```
F1 Score : 0.5625663904822604
Epoch 55/100
142/142 [==============================] - 10s 70ms/step - loss: 0.7631 - accuracy: 0.7438 - val_loss: 1.0579 - val_a
ccuracy: 0.6724
F1 Score : 0.5727639685574676
Epoch 56/100
142/142 [==============================] - 10s 69ms/step - loss: 0.7296 - accuracy: 0.7534 - val_loss: 1.0384 - val_a
ccuracy: 0.6730
F1 Score : 0.573401317187168
Epoch 57/100
142/142 [==============================] - 10s 69ms/step - loss: 0.7135 - accuracy: 0.7578 - val_loss: 1.0307 - val_a
ccuracy: 0.6839
F1 Score : 0.5687274272360314
Epoch 58/100
142/142 [==============================] - 10s 70ms/step - loss: 0.7201 - accuracy: 0.7634 - val_loss: 1.0520 - val_a
ccuracy: 0.6788
F1 Score : 0.5814743998300403
Epoch 59/100
142/142 [==============================] - 10s 69ms/step - loss: 0.6933 - accuracy: 0.7633 - val_loss: 1.0114 - val_a
ccuracy: 0.6987
F1 Score : 0.6039940514127895
Epoch 60/100
142/142 [==============================] - 10s 69ms/step - loss: 0.6836 - accuracy: 0.7767 - val_loss: 0.9698 - val_a
ccuracy: 0.7060

Reached 70+ validation accuracy, so stopping training!!
F1 Score : 0.6050562991289569
```

In [ ]:

```
model.save("'best_model_1.h5'")
```

```
INFO:tensorflow:Assets written to: 'best_model_1.h5'/assets
```

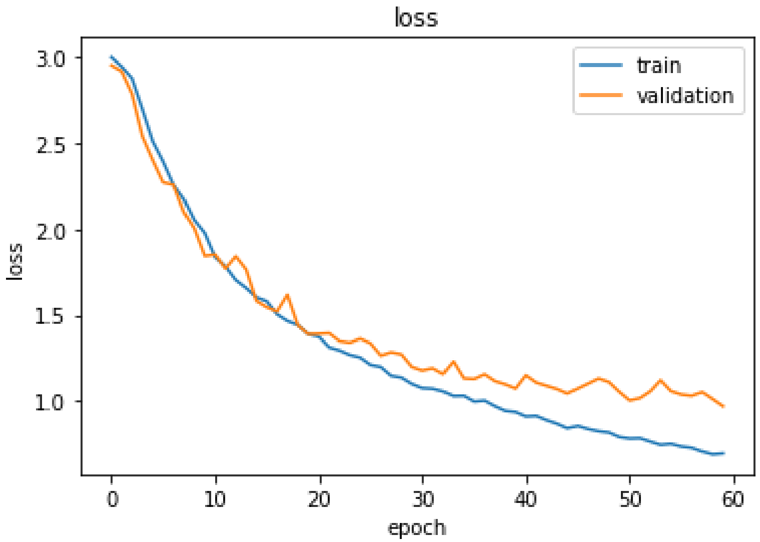
In [ ]:

```
saved_model = keras.models.load_model("'best_model_1.h5'")
score = saved_model.evaluate(val, yte, verbose=1)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
148/148 [==============================] - 2s 16ms/step - loss: 0.9698 - accuracy: 0.7060
Test loss: 0.9697709679603577
Test accuracy: 0.7059698104858398
```

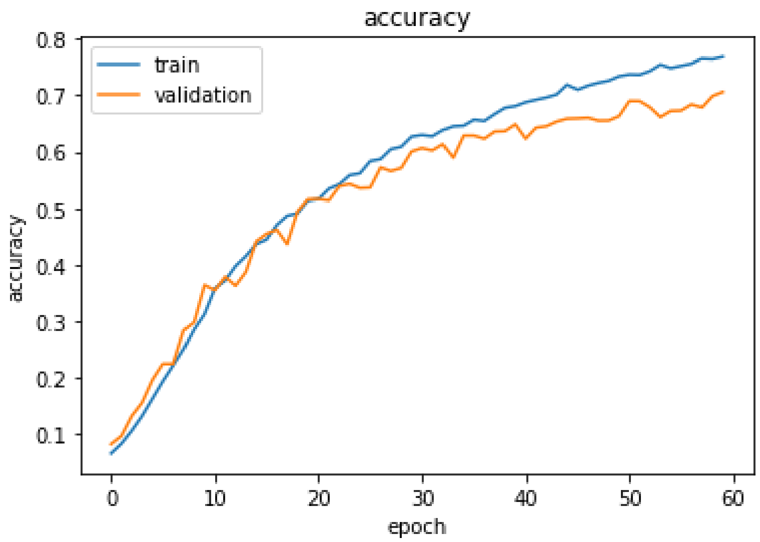
Loss Curve

In [ ]:
```
import matplotlib.pyplot as plt
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("loss")
plt.xlabel("epoch")
plt.ylabel("loss")
plt.legend(['train','validation'])
plt.show()
```



Accuracy Curve

In [ ]:
```
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title("accuracy")
plt.xlabel("epoch")
plt.ylabel("accuracy")
plt.legend(['train','validation'])
plt.show()
```



In [ ]:

Model 2

In [8]:
```
x = combined_columns
y = data['class']
x_train, x_test, y_train , y_test = train_test_split(x, y, test_size=0.25,random_state=30, stratify = y)
```

In [9]:
```
tokenizer_char = Tokenizer(char_level=True,filters='!"#$%&()*+,-./:;<=>?@[\\]^`{|}~\t\n')
tokenizer_char.fit_on_texts(x_train[0])
sequences2 = tokenizer_char.texts_to_sequences(x_train[0])
sequences2_test = tokenizer_char.texts_to_sequences(x_test[0])
length_sequences2 = list()
for i in sequences2:
    length_sequences2.append(len(i))
max_length2 = max(length_sequences2)
vocab = len(tokenizer_char.word_index)


x_train_pad = pad_sequences(sequences2,maxlen = max_length2,padding='post')
x_test_pad = pad_sequences(sequences2_test,maxlen = max_length2,padding='post')
```

In [25]:

```
import tensorflow
input_layer_2 = tensorflow.keras.Input(shape=(max_length2,),dtype='int32')
embedding_2 = layers.Embedding(input_dim = vocab+1, output_dim = 4, input_length = max_length2,trainable=True)(input_

conv1D_a = layers.Conv1D(filters=16, kernel_size=5, activation='relu',kernel_initializer = tensorflow.keras.initializ
conv1D_b = layers.Conv1D(filters=16, kernel_size=5, activation='relu',kernel_initializer = tensorflow.keras.initializ

pool1 = layers.MaxPooling1D(pool_size=2,strides=None, padding="valid")(conv1D_b)

conv1D_c = layers.Conv1D(filters=16, kernel_size=3, activation='relu',kernel_initializer = tensorflow.keras.initializ
conv1D_d = layers.Conv1D(filters=16, kernel_size=3, activation='relu',kernel_initializer = tensorflow.keras.initializ
conv1D_e = layers.Conv1D(filters=16, kernel_size=3,activation='relu',kernel_initializer = tensorflow.keras.initialize

pool2 = layers.MaxPooling1D(pool_size=3,strides=None, padding="valid")(conv1D_e)
conv1D_f = layers.Conv1D(filters=4, kernel_size=1, activation='relu',kernel_initializer = tensorflow.keras.initialize
flatten_1 = layers.Flatten()(conv1D_f)

dropout1 = layers.Dropout(0.2)(flatten_1)

dens1 = layers.Dense(560,activation='tanh',kernel_initializer = tensorflow.keras.initializers.glorot_normal(2))(dropo
dropout2 = layers.Dropout(0.3)(dens1)

dens2 = layers.Dense(130,activation='tanh')(dropout2)
dropout3 = layers.Dropout(0.25)(dens2)

dens3 = layers.Dense(68,activation='relu',kernel_initializer = tensorflow.keras.initializers.he_uniform(50))(dropout3
dropout4 = layers.Dropout(0.2)(dens3)

output = layers.Dense(20,activation='softmax',kernel_initializer=tensorflow.keras.initializers.glorot_normal(seed=0))
model2 = keras.Model(inputs=input_layer_2,outputs=output)

print(model2.summary())
```

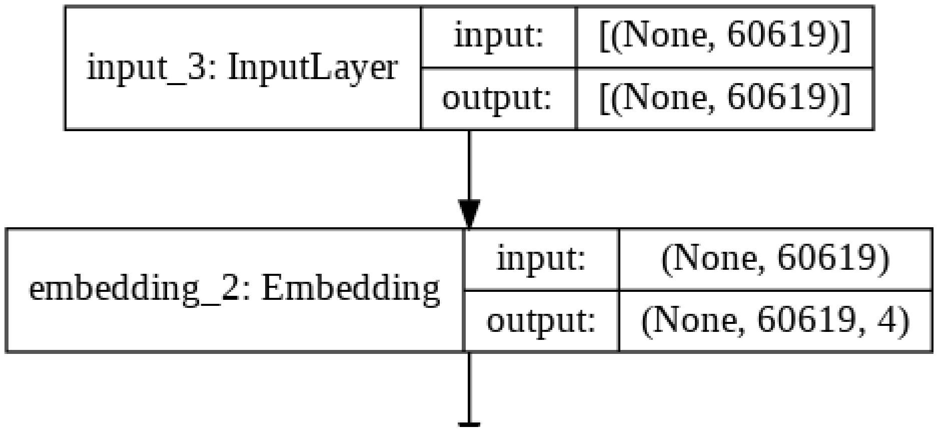
```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 60619)]           0
_________________________________________________________________
embedding_2 (Embedding)      (None, 60619, 4)          360
_________________________________________________________________
conv1d_10 (Conv1D)           (None, 60615, 16)         336
```

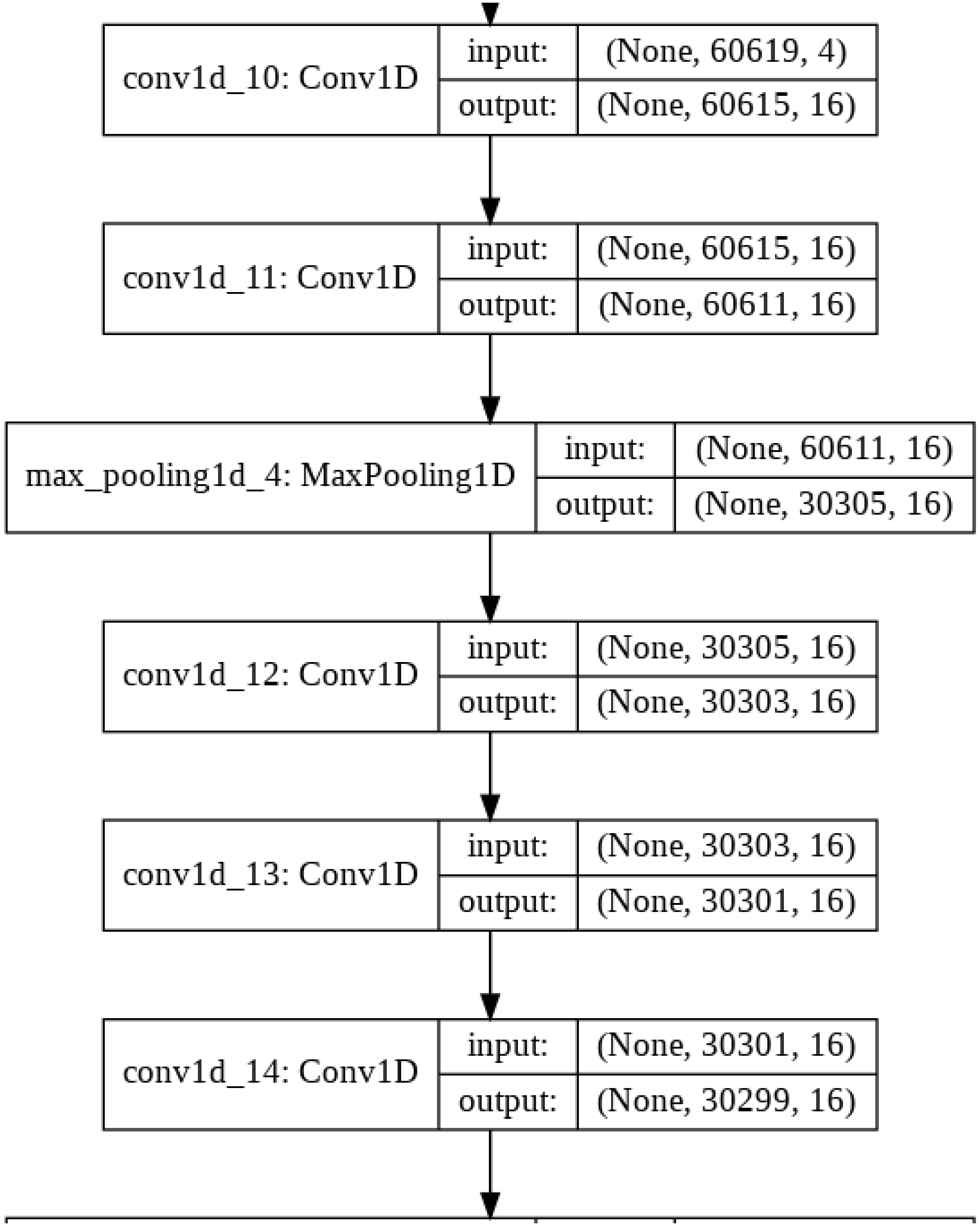
```
_________________________________________________________________
conv1d_11 (Conv1D)           (None, 60611, 16)         1296      
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 30305, 16)         0         
_________________________________________________________________
conv1d_12 (Conv1D)           (None, 30303, 16)         784       
_________________________________________________________________
conv1d_13 (Conv1D)           (None, 30301, 16)         784       
_________________________________________________________________
conv1d_14 (Conv1D)           (None, 30299, 16)         784       
_________________________________________________________________
max_pooling1d_5 (MaxPooling1 (None, 10099, 16)         0         
_________________________________________________________________
conv1d_15 (Conv1D)           (None, 10099, 4)          68        
_________________________________________________________________
flatten_2 (Flatten)          (None, 40396)             0         
_________________________________________________________________
dropout_5 (Dropout)          (None, 40396)             0         
_________________________________________________________________
dense_5 (Dense)              (None, 560)               22622320  
_________________________________________________________________
dense_8 (Dense)              (None, 20)                11220     
=================================================================
Total params: 22,637,952
Trainable params: 22,637,952
Non-trainable params: 0
_________________________________________________________________
None
```

In [27]:

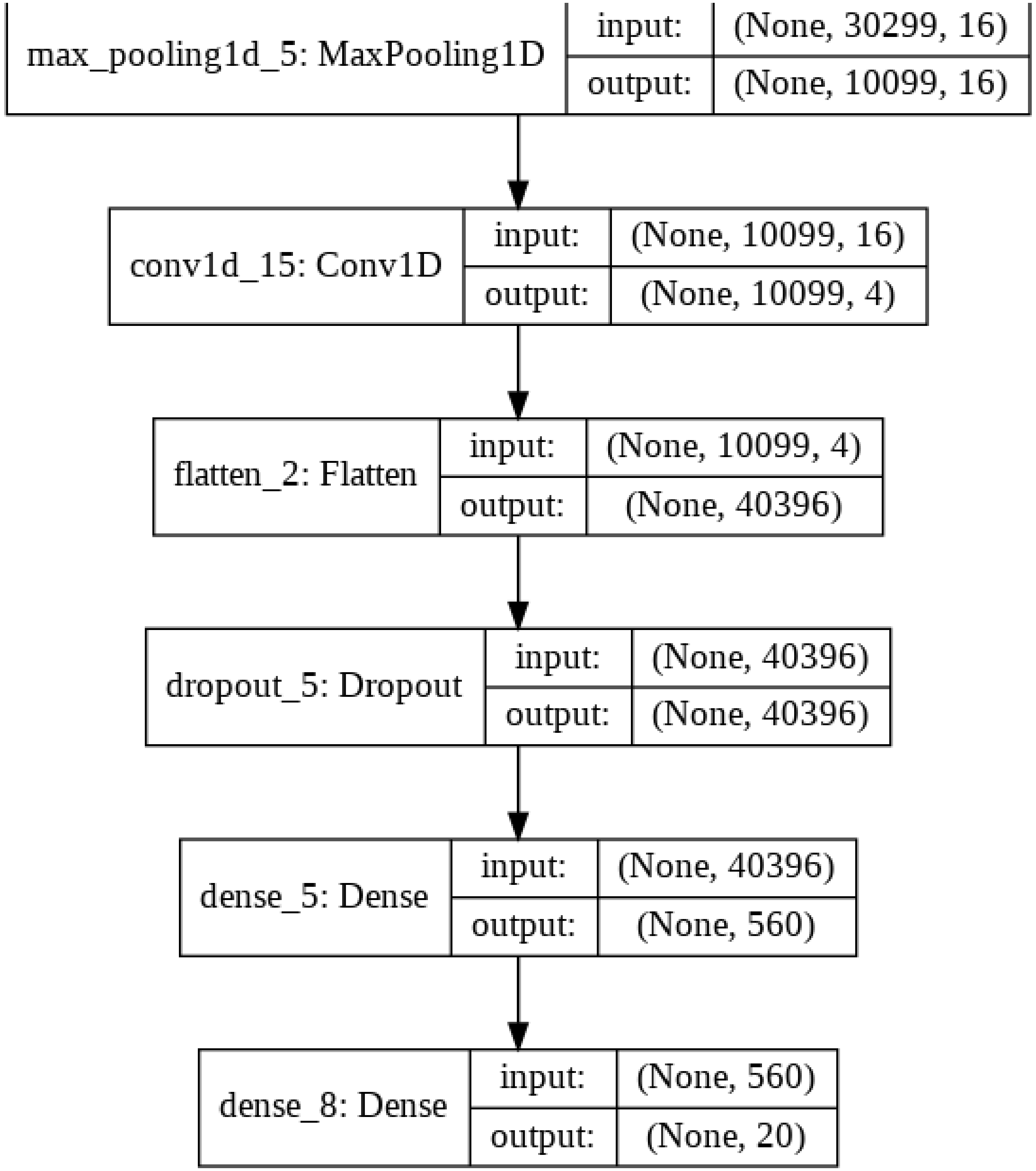
```
keras.utils.plot_model(model2,show_shapes=True)
```

Out[27]:





conv1d_10: Conv1D
input: (None, 60619, 4)
output: (None, 60615, 16)
conv1d_11: Conv1D
input: (None, 60615, 16)
output: (None, 60611, 16)
max_pooling1d_4: MaxPooling1D
input: (None, 60611, 16)
output: (None, 30305, 16)
conv1d_12: Conv1D
input: (None, 30305, 16)
output: (None, 30303, 16)
conv1d_13: Conv1D
input: (None, 30303, 16)
output: (None, 30301, 16)
conv1d_14: Conv1D
input: (None, 30301, 16)
output: (None, 30299, 16)



In [16]:

```
#https://towardsdatascience.com/neural-network-with-tensorflow-how-to-stop-training-using-callback-5c8d575c18a9
class myCallback(tf.keras.callbacks.Callback):
```

```
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('val_accuracy') > 0.10):
      print("\nReached 10+ validation accuracy, so stopping training!!" )
      self.model.stop_training = True
```

In [17]:

```
from sklearn.metrics import f1_score
class metrics(tf.keras.callbacks.Callback):
  def __init__(self,validationx,valy):
    super(metrics, self).__init__()
    self.validationx = validationx
    self.valy = valy

  def on_epoch_end(self, epoch, logs={}):
    y_pred = (np.asarray(self.model.predict(self.validationx))).round()
    y_true = (np.squeeze(np.asarray(self.valy)))
    val_f1 = f1_score(y_true, y_pred,average='samples')
    print("F1 Score : "+str(val_f1))
    return
```

In [33]:

```
import datetime
import os
os.mkdir("/content/m2__logss")
log_dir = "/content/m2__logss/model_log"+datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1)
```

In [36]:

```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
y_train_enc= label_encoder.fit_transform(y_train)
y_test_enc= label_encoder.transform(y_test)

from keras.utils import np_utils
ytr = np_utils.to_categorical(y_train_enc,20)
yte = np_utils.to_categorical(y_test_enc,20)

model2.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.001, momentum=0.9),loss='categorical_crossentropy',r
metrics_callback = metrics(x_test_pad,yte)
history=model2.fit(x_train_pad,ytr,validation_data=(x_test_pad,yte), batch_size=100,epochs=100,callbacks=[myCallback(
```

```
Epoch 1/100
142/142 [==============================] - 56s 388ms/step - loss: 2.9892 - accuracy: 0.0587 - val_loss: 2.9699 - val_
accuracy: 0.0697
```

```
F1 Score : 0.0
Epoch 2/100
142/142 [==============================] - 55s 390ms/step - loss: 2.9579 - accuracy: 0.0701 - val_loss: 2.9240 - val_
accuracy: 0.0865
F1 Score : 0.0
Epoch 3/100
142/142 [==============================] - 55s 388ms/step - loss: 2.9320 - accuracy: 0.0836 - val_loss: 2.9222 - val_
accuracy: 0.0788
F1 Score : 0.0
Epoch 4/100
142/142 [==============================] - 55s 387ms/step - loss: 2.9261 - accuracy: 0.0857 - val_loss: 2.9189 - val_
accuracy: 0.0814
F1 Score : 0.0
Epoch 5/100
142/142 [==============================] - 55s 390ms/step - loss: 2.9150 - accuracy: 0.0867 - val_loss: 2.9218 - val_
accuracy: 0.0914
F1 Score : 0.0
Epoch 6/100
142/142 [==============================] - 55s 390ms/step - loss: 2.9120 - accuracy: 0.0939 - val_loss: 2.9217 - val_
accuracy: 0.0822
F1 Score : 0.0
Epoch 7/100
142/142 [==============================] - 56s 393ms/step - loss: 2.9161 - accuracy: 0.0870 - val_loss: 2.9160 - val_
accuracy: 0.0807
F1 Score : 0.0
Epoch 8/100
142/142 [==============================] - 55s 391ms/step - loss: 2.9077 - accuracy: 0.0950 - val_loss: 2.9173 - val_
accuracy: 0.0918
F1 Score : 0.0
Epoch 9/100
142/142 [==============================] - 55s 390ms/step - loss: 2.9088 - accuracy: 0.0929 - val_loss: 2.9166 - val_
accuracy: 0.0833
F1 Score : 0.0
Epoch 10/100
142/142 [==============================] - 55s 389ms/step - loss: 2.9038 - accuracy: 0.0942 - val_loss: 2.9156 - val_
accuracy: 0.0888
F1 Score : 0.0
Epoch 11/100
142/142 [==============================] - 55s 389ms/step - loss: 2.9014 - accuracy: 0.0919 - val_loss: 2.9163 - val_
accuracy: 0.0888
F1 Score : 0.0
Epoch 12/100
142/142 [==============================] - 55s 387ms/step - loss: 2.8992 - accuracy: 0.0927 - val_loss: 2.9161 - val_
accuracy: 0.0873
F1 Score : 0.0
```

```
Epoch 13/100
142/142 [==============================] - 55s 387ms/step - loss: 2.9049 - accuracy: 0.0933 - val_loss: 2.9143 - val_
accuracy: 0.0814
F1 Score : 0.0
Epoch 14/100
142/142 [==============================] - 55s 387ms/step - loss: 2.9002 - accuracy: 0.0937 - val_loss: 2.9142 - val_
accuracy: 0.0809
F1 Score : 0.00021244954323348204
Epoch 15/100
142/142 [==============================] - 55s 389ms/step - loss: 2.8974 - accuracy: 0.0923 - val_loss: 2.9136 - val_
accuracy: 0.0820
F1 Score : 0.00021244954323348204
Epoch 16/100
142/142 [==============================] - 54s 384ms/step - loss: 2.8882 - accuracy: 0.1001 - val_loss: 2.9131 - val_
accuracy: 0.0812
F1 Score : 0.00021244954323348204
Epoch 17/100
142/142 [==============================] - 54s 381ms/step - loss: 2.8970 - accuracy: 0.0967 - val_loss: 2.9167 - val_
accuracy: 0.0869
F1 Score : 0.00021244954323348204
Epoch 18/100
142/142 [==============================] - 54s 380ms/step - loss: 2.8893 - accuracy: 0.0949 - val_loss: 2.9131 - val_
accuracy: 0.0856
F1 Score : 0.00021244954323348204
Epoch 19/100
142/142 [==============================] - 54s 380ms/step - loss: 2.8893 - accuracy: 0.0996 - val_loss: 2.9132 - val_
accuracy: 0.0852
F1 Score : 0.0004248990864669641
Epoch 20/100
142/142 [==============================] - 54s 383ms/step - loss: 2.8812 - accuracy: 0.1019 - val_loss: 2.9130 - val_
accuracy: 0.0854
F1 Score : 0.0004248990864669641
Epoch 21/100
142/142 [==============================] - 54s 381ms/step - loss: 2.8721 - accuracy: 0.1119 - val_loss: 2.9135 - val_
accuracy: 0.0888
F1 Score : 0.0004248990864669641
Epoch 22/100
142/142 [==============================] - 54s 381ms/step - loss: 2.8620 - accuracy: 0.1108 - val_loss: 2.9132 - val_
accuracy: 0.0907
F1 Score : 0.0006373486297004461
Epoch 23/100
142/142 [==============================] - 54s 383ms/step - loss: 2.8533 - accuracy: 0.1167 - val_loss: 2.9169 - val_
accuracy: 0.0907
F1 Score : 0.0008497981729339282
Epoch 24/100
```

```
142/142 [==============================] - 54s 381ms/step - loss: 2.8440 - accuracy: 0.1240 - val_loss: 2.9207 - val_
accuracy: 0.0871
F1 Score : 0.0006373486297004461
Epoch 25/100
142/142 [==============================] - 54s 384ms/step - loss: 2.8325 - accuracy: 0.1317 - val_loss: 2.9240 - val_
accuracy: 0.0909
F1 Score : 0.0008497981729339282
Epoch 26/100
142/142 [==============================] - 55s 385ms/step - loss: 2.8186 - accuracy: 0.1373 - val_loss: 2.9301 - val_
accuracy: 0.0918
F1 Score : 0.0012746972594008922
Epoch 27/100
142/142 [==============================] - 55s 386ms/step - loss: 2.8023 - accuracy: 0.1395 - val_loss: 2.9295 - val_
accuracy: 0.0892
F1 Score : 0.0012746972594008922
Epoch 28/100
142/142 [==============================] - 55s 389ms/step - loss: 2.7794 - accuracy: 0.1477 - val_loss: 2.9443 - val_
accuracy: 0.0856
F1 Score : 0.0021244954323348204
Epoch 29/100
142/142 [==============================] - 55s 387ms/step - loss: 2.7660 - accuracy: 0.1596 - val_loss: 2.9427 - val_
accuracy: 0.0965
F1 Score : 0.0012746972594008922
Epoch 30/100
142/142 [==============================] - 55s 390ms/step - loss: 2.7608 - accuracy: 0.1563 - val_loss: 2.9435 - val_
accuracy: 0.0956
F1 Score : 0.0019120458891013384
Epoch 31/100
142/142 [==============================] - 56s 393ms/step - loss: 2.7455 - accuracy: 0.1631 - val_loss: 2.9482 - val_
accuracy: 0.0918
F1 Score : 0.0021244954323348204
Epoch 32/100
142/142 [==============================] - 55s 388ms/step - loss: 2.7302 - accuracy: 0.1676 - val_loss: 2.9594 - val_
accuracy: 0.0926
F1 Score : 0.0031867431485022306
Epoch 33/100
142/142 [==============================] - 55s 390ms/step - loss: 2.7182 - accuracy: 0.1646 - val_loss: 2.9614 - val_
accuracy: 0.0931
F1 Score : 0.0029742936052687486
Epoch 34/100
142/142 [==============================] - 55s 389ms/step - loss: 2.7006 - accuracy: 0.1760 - val_loss: 2.9649 - val_
accuracy: 0.0884
F1 Score : 0.0025493945188017845
Epoch 35/100
142/142 [==============================] - 55s 387ms/step - loss: 2.6977 - accuracy: 0.1812 - val_loss: 2.9625 - val_
```

```
accuracy: 0.0914
F1 Score : 0.0036116422349691947
Epoch 36/100
142/142 [==============================] - 55s 388ms/step - loss: 2.6873 - accuracy: 0.1799 - val_loss: 2.9676 - val_
accuracy: 0.0943
F1 Score : 0.0029742936052687486
Epoch 37/100
142/142 [==============================] - 55s 391ms/step - loss: 2.6977 - accuracy: 0.1740 - val_loss: 3.0061 - val_
accuracy: 0.0918
F1 Score : 0.004248990864669641
Epoch 38/100
142/142 [==============================] - 55s 389ms/step - loss: 2.6830 - accuracy: 0.1798 - val_loss: 2.9881 - val_
accuracy: 0.0892
F1 Score : 0.0038240917782026767
Epoch 39/100
142/142 [==============================] - 55s 389ms/step - loss: 2.6680 - accuracy: 0.1852 - val_loss: 2.9759 - val_
accuracy: 0.0882
F1 Score : 0.0023369449755683024
Epoch 40/100
142/142 [==============================] - 55s 389ms/step - loss: 2.6656 - accuracy: 0.1824 - val_loss: 2.9935 - val_
accuracy: 0.0894
F1 Score : 0.0038240917782026767
Epoch 41/100
142/142 [==============================] - 55s 390ms/step - loss: 2.6523 - accuracy: 0.1856 - val_loss: 2.9934 - val_
accuracy: 0.0933
F1 Score : 0.0036116422349691947
Epoch 42/100
142/142 [==============================] - 55s 388ms/step - loss: 2.6521 - accuracy: 0.1889 - val_loss: 3.0167 - val_
accuracy: 0.0937
F1 Score : 0.004036541321436159
Epoch 43/100
142/142 [==============================] - 55s 389ms/step - loss: 2.6422 - accuracy: 0.1923 - val_loss: 3.0125 - val_
accuracy: 0.0984
F1 Score : 0.004248990864669641
Epoch 44/100
142/142 [==============================] - 55s 389ms/step - loss: 2.6383 - accuracy: 0.1926 - val_loss: 2.9962 - val_
accuracy: 0.0977
F1 Score : 0.004248990864669641
Epoch 45/100
142/142 [==============================] - 55s 390ms/step - loss: 2.6221 - accuracy: 0.1953 - val_loss: 3.0106 - val_
accuracy: 0.0965
F1 Score : 0.004036541321436159
Epoch 46/100
142/142 [==============================] - 55s 389ms/step - loss: 2.6131 - accuracy: 0.1966 - val_loss: 3.0174 - val_
accuracy: 0.0996
```

```
F1 Score : 0.0038240917782026767
Epoch 47/100
142/142 [==============================] - 55s 391ms/step - loss: 2.6229 - accuracy: 0.1998 - val_loss: 3.0240 - val_
accuracy: 0.0967
F1 Score : 0.004886339494370087
Epoch 48/100
142/142 [==============================] - 55s 390ms/step - loss: 2.6046 - accuracy: 0.2009 - val_loss: 3.0318 - val_
accuracy: 0.0994
F1 Score : 0.004673889951136605
Epoch 49/100
142/142 [==============================] - 55s 389ms/step - loss: 2.5932 - accuracy: 0.2085 - val_loss: 3.0597 - val_
accuracy: 0.1005

Reached 10+ validation accuracy, so stopping training!!
F1 Score : 0.005098789037603569
```

In [37]:

```
model2.save("'best_model_2.h5'")
```

```
INFO:tensorflow:Assets written to: 'best_model_2.h5'/assets
```

In [38]:

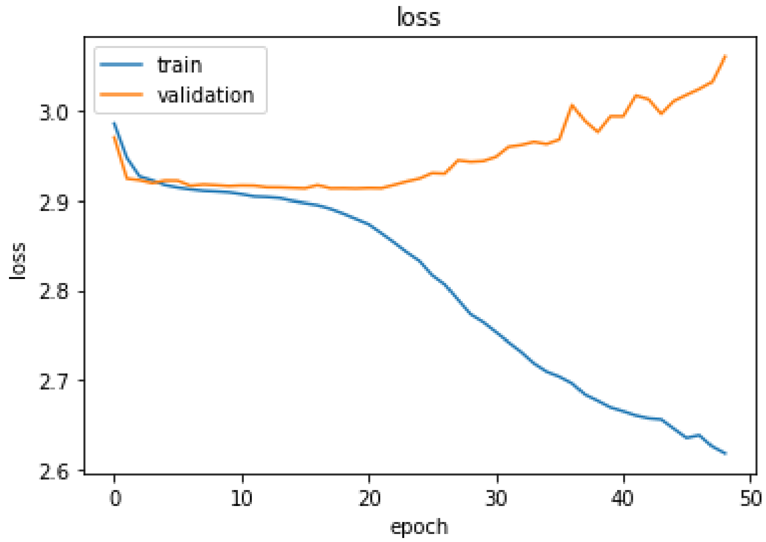
```
saved_model = keras.models.load_model("'best_model_2.h5'")
score = saved_model.evaluate(x_test_pad, yte, verbose=1)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
148/148 [==============================] - 6s 42ms/step - loss: 3.0597 - accuracy: 0.1005
Test loss: 3.059715747833252
Test accuracy: 0.10048863291740417
```

Loss curve

In [39]:

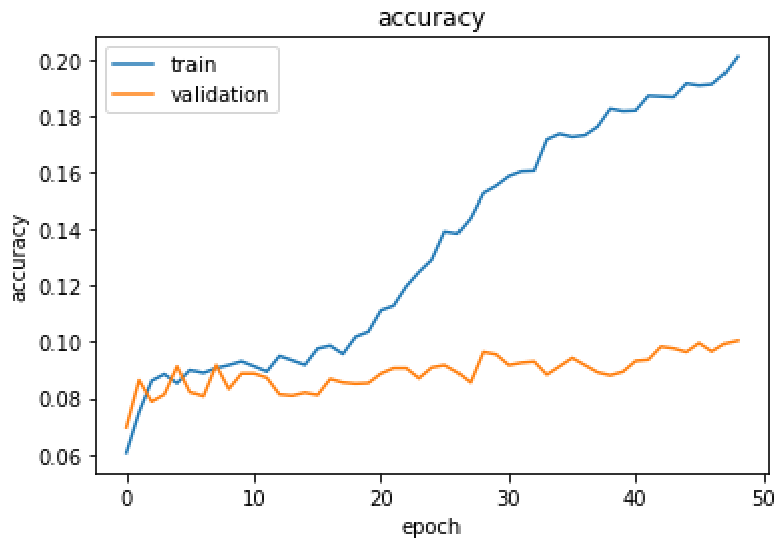
```
import matplotlib.pyplot as plt
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("loss")
plt.xlabel("epoch")
plt.ylabel("loss")
plt.legend(['train','validation'])
plt.show()
```



Accuracy Lose

In [40]:

```
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title("accuracy")
plt.xlabel("epoch")
plt.ylabel("accuracy")
plt.legend(['train','validation'])
plt.show()
```


accuracy
train
validation
0.20
0.18
0.16
0.14
0.12
0.10
0.08
0.06
accuracy
0
10
20
30
40
50
epoch

```
In [ ]:
```