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 [1]:

```
import os
import re
import pickle
import datetime
from tqdm import tqdm
from time import time

import numpy as np
import pandas as pd

import plotly.express as px
import plotly.graph_objects as go
import matplotlib.pyplot as plt

import string
import nltk
from nltk import ne_chunk
from nltk.tree import Tree
from nltk.tag import pos_tag
from nltk import word_tokenize

from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

import tensorflow
from tensorflow.keras import Input
from tensorflow.keras import Model
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Conv1D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import MaxPool1D
from tensorflow.keras.layers import Embedding
from tensorflow.keras.layers import concatenate

from tensorflow.keras.callbacks import Callback
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import ReduceLROnPlateau

# from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers import Adam

from tensorflow.keras.utils import plot_model
from tensorflow.keras.utils import to_categorical

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

%load_ext tensorboard

import warnings
warnings.filterwarnings("ignore")

start_ = time()
```

```
2022-07-09 01:34:46.404941: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynami
c library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object file: No such file or direct
ory
2022-07-09 01:34:46.404971: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if y
ou do not have a GPU set up on your machine.
```

In [2]:

```
nltk.download('averaged_perceptron_tagger')
```

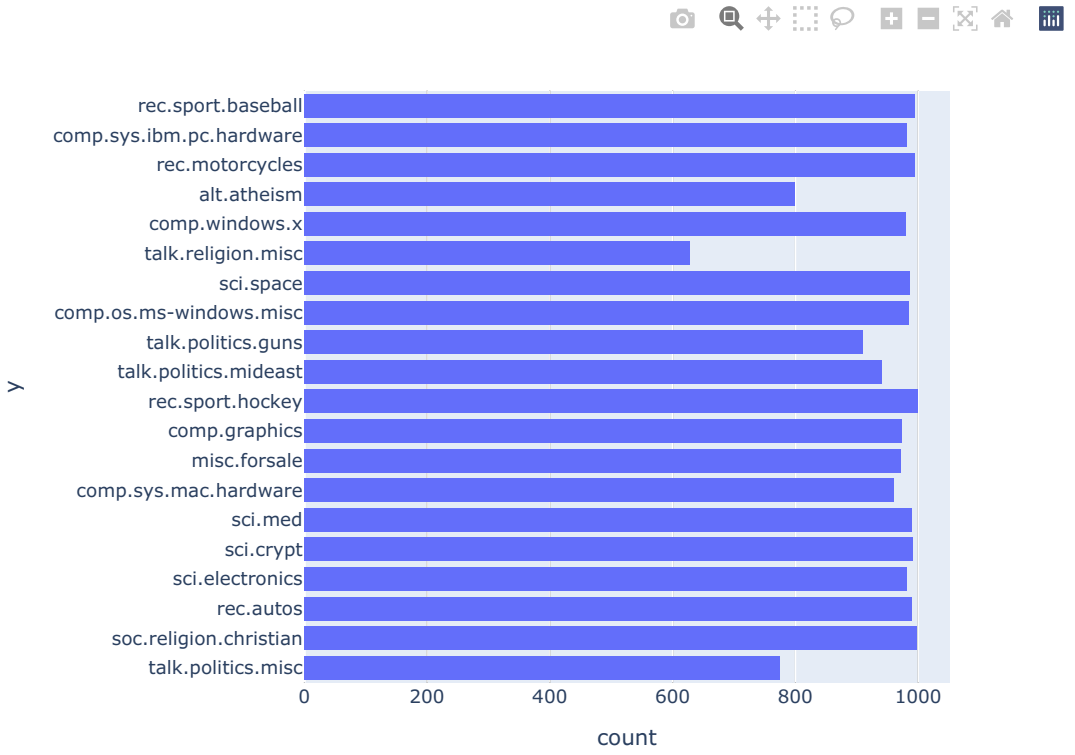
```
nltk.download('maxent_ne_chunker')
nltk.download('words')
nltk.download('punkt')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /home/jishnu/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /home/jishnu/nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to /home/jishnu/nltk_data...
[nltk_data]   Package words is already up-to-date!
[nltk_data] Downloading package punkt to /home/jishnu/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[2]: True

In [3]:
```
Main_Path = '/home/jishnu/AAIC/CNN_with_textdata'
ext_doc_path = Main_Path + '/extracted_documents'
```

In [4]:
```
file_name_list = [files for files in os.listdir(ext_doc_path)]

### count plot of all the class labels.
tg_class = []
for file in file_name_list:
    tg_class.append(file.split('_')[0])

# sns.countplot(y = tg_class)
px.histogram(y = tg_class)
```



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?

In [5]:

```
# https://docs.python.org/2.4/lib/standard-encodings.html
# Python Library Reference - Standard Encodings

encoders = ['ascii', 'big5', 'big5hkscs', 'cp037', 'cp424', 'cp437', 'cp500', 'cp737', 'cp775',
            'cp850', 'cp852', 'cp855', 'cp856', 'cp857', 'cp860', 'cp861', 'cp862', 'cp863',
            'cp864', 'cp865', 'cp866', 'cp869', 'cp874', 'cp875', 'cp932', 'cp949', 'cp950',
            'cp1006', 'cp1026', 'cp1140', 'cp1250', 'cp1251', 'cp1252', 'cp1253', 'cp1254',
            'cp1255', 'cp1256', 'cp1257', 'cp1258', 'euc_jp', 'euc_jis_2004', 'euc_jisx0213',
            'euc_kr', 'gb2312', 'gbk', 'gb18030', 'hz', 'iso2022_jp', 'iso2022_jp_1',
            'iso2022_jp_2', 'iso2022_jp_2004', 'iso2022_jp_3', 'iso2022_jp_ext', 'iso2022_kr',
            'latin_1', 'iso8859_2', 'iso8859_3', 'iso8859_4', 'iso8859_5', 'iso8859_6',
            'iso8859_7', 'iso8859_8', 'iso8859_9', 'iso8859_10', 'iso8859_13', 'iso8859_14',
            'iso8859_15', 'johab', 'koi8_r', 'koi8_u', 'mac_cyrillic', 'mac_greek',
            'mac_iceland', 'mac_latin2', 'mac_roman', 'mac_turkish', 'ptcp154', 'shift_jis',
            'shift_jis_2004', 'shift_jisx0213', 'utf_16', 'utf_16_be', 'utf_16_le', 'utf_7',
            'utf_8', 'base64_codec', 'bz2_codec', 'hex_codec', 'idna', 'mbcs', 'palmos',
            'punycode', 'quopri_codec', 'raw_unicode_escape', 'rot_13', 'string_escape',
            'undefined', 'unicode_escape', 'unicode_internal', 'uu_codec', 'zlib_codec']

valid_encoder_list = []
for enc_ in tqdm(encoders):
    for file_name in file_name_list:
        try:
            with open(os.path.join(ext_doc_path, file_name), mode='r', encoding= enc_) as f:
                for line in f:
                    pass
            valid_encoder_list.append(enc_)
        except:
            pass

temp_df = pd.DataFrame(pd.Series(valid_encoder_list).value_counts()).reset_index(drop = False)
temp_df.rename(columns={'index': 'Encodings_Codec', 0: 'Successful_Run'}, inplace=True)
temp_df[temp_df.Successful_Run == len(file_name_list)]
```

```
100%|██████████| 101/101 [00:31<00:00,  3.16it/s]
```

Out[5]:

	Encodings_Codec	Successful_Run
0	cp1252	18828
1	cp1006	18828
2	cp1140	18828
3	cp1250	18828
4	cp1251	18828
5	iso8859_10	18828
6	cp1254	18828
7	cp1256	18828
8	cp1258	18828
9	ptcp154	18828
10	mac_turkish	18828
11	mac_roman	18828
12	mac_latin2	18828
13	mac_iceland	18828
14	mac_greek	18828
15	mac_cyrillic	18828
16	koi8_u	18828
17	koi8_r	18828
18	iso8859_15	18828
19	iso8859_14	18828
20	latin_1	18828

21	iso8859_2	18828
22	iso8859_4	18828
23	iso8859_5	18828
24	iso8859_13	18828
25	cp1026	18828
26	iso8859_9	18828
27	palmos	18828
28	cp437	18828
29	cp850	18828
30	cp852	18828
31	cp855	18828
32	cp500	18828
33	cp860	18828
34	cp861	18828
35	cp862	18828
36	cp863	18828
37	cp737	18828
38	cp775	18828
39	cp037	18828
40	cp865	18828
41	cp866	18828
42	cp875	18828

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.

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

12. 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.

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

```
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)
```

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

In [6]:
```
def decontracted(phrase): # https://stackoverflow.com/a/47091490/4084039
    # specific
    phrase = re.sub(r"won\'t", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

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

    return phrase

def get_person_chunks(text, label):
    chunks = ne_chunk(pos_tag(word_tokenize(text)))

    person_chunk = []
    person_chunks = []

    for chunk in chunks:
        if (type(chunk) == Tree) and (chunk.label() == label):
            person_chunk.append(' '.join([tk for tk, pos in chunk]))

    return pd.Series(person_chunk).unique().tolist()


def get_gpe_chunks(text, label):
    chunks = ne_chunk(pos_tag(word_tokenize(text)))

    gpe_chunk = []
    gpe_chunk_without = []

    for chunk in chunks:
        if (type(chunk) == Tree) and (chunk.label() == label):
            gpe_chunk.append('_'.join([tk for tk, pos in chunk]))
            gpe_chunk_without.append(' '.join([tk for tk, pos in chunk]))

    return pd.Series(gpe_chunk).unique().tolist(), pd.Series(gpe_chunk_without).unique().tolist()
```

In [7]:
```
def preprocess(filename, encoder = 'mac_roman'):

    tg_class = filename.split('_')[0]

    raw_data = []
    with open(os.path.join(ext_doc_path, filename), mode='r', encoding= encoder) as file:
        raw_data.append(file.read())

    lists = []
    with open(os.path.join(ext_doc_path, filename), mode='r', encoding= encoder) as f:
        for line in f:
            inner_list = [line.strip() for line in line.split(' ')]
            line = ' '.join(inner_list)
            lists.append(line)

# 1 Find all emails
    preprocessed_email = []
    for idx, line in enumerate(lists):
        if '@' in line:
            # \S Matches any character which is not a whitespace character.
            for email in re.findall(r'@(\S+)', line):
                for e_id in email.split('.'):
                    e_id = re.sub(r'<|>', '', e_id)
                    if (e_id.lower() != 'com') & (len(e_id) >2):
                        preprocessed_email.append(e_id.lower())

#2 Replace all the emails by space
            if re.search(r'@\S+', line):
                lists[idx] = re.sub(r'\b\S+@\S+\b', ' ',line)

#3 Get subject of the text
    # https://www.computerhope.com/jargon/s/specchar.htm
    subject_list =[]
    for idx, line in enumerate(lists):
        if re.findall(r'[S|s]ubject:', line):
            subject_list.append(' '.join(re.findall(r'\w+', line[9:].replace('Re:', ''))).strip())

#4 Replace those sentances in original text by space
            lists[idx] = ' '

#5 Delete all the sentances where sentence starts with "Write to:" or "From:"
    for idx, line in enumerate(lists):
        if re.findall(r'^From:|^Write to:', line):
            lists[idx] = ' '
```

```
    #6 Delete all the tags like "< anyword >"
        for idx, line in enumerate(lists):
            if re.findall(r'<.*?>', line):
                lists[idx] = re.sub(r'<.*?>', '', line).strip()

    #7 Delete all the data which are present in the brackets
        for idx, line in enumerate(lists):
            if re.findall(r'\(.*\)', line):
                lists[idx] = re.sub(r'\(.*\)', '', line)

    #8 Remove all the newlines('\n'), tabs('\t'), "-", "\"
        for idx, line in enumerate(lists):
            if re.findall(r'[\t\n\-\\]', line):
                lists[idx] = re.sub(r'[\t\n\-\\]', '', line)

    #9 Remove all the words which ends with ":"
        for idx, line in enumerate(lists):
            if re.findall(r':', line):
                lists[idx] = re.sub(r'\w+:', '', line)

    #10 Decontractions
        for idx, line in enumerate(lists):
            lists[idx] = decontracted(line)

    #11 Chunking
        whole_text = ' '.join(lists).strip()
        whole_text = re.sub(r'\s+', ' ', whole_text)
        person_chunk = get_person_chunks(whole_text, 'PERSON')
        gpe_chunk, gpe_chunk_without = get_gpe_chunks(whole_text, 'GPE')

    #12 Remove PERSON and adding '_' to GPE
        for chunk in person_chunk:
            whole_text = whole_text.replace(chunk, '')
        for i in range(len(gpe_chunk)):
            whole_text = whole_text.replace(gpe_chunk_without[i], gpe_chunk[i])

    #13 Replace all the digits with space i.e delete all the digits
        whole_text = re.sub(r'\d', '', whole_text)

    #14 Remove the '_' from starting and ending of a word
        whole_text = re.sub(r'_\b', '', whole_text) # From end of a word
        whole_text = re.sub(r'\s_', ' ', whole_text) # From start of a word

    #15 Remove words which are length <= 2 after spliiting those words by '_'
        for word in re.findall(r'\w+_\w+', whole_text):
            if (len(word.split('_')[0]) <= 2) and (len(word.split('_')[1]) > 0):
                whole_text = whole_text.replace(word, word.split('_')[1])

    #16 Convert all the words into lower case, remove words with length >=15 and <=2
        whole_text = re.sub(r'\s+', ' ', whole_text).lower()
        whole_text = ' '.join([word for word in whole_text.split() if (len(word) < 15) and (len(word) > 2)])

    #17 replace all the words except "A-Za-z_" with space
        whole_text = re.sub(r'[^A-Za-z_ ]', '', whole_text)

        return raw_data, preprocessed_email, subject_list, whole_text, tg_class
```

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.

In [8]:

```
data, email, sub, tex, cl = preprocess('alt.atheism_49960.txt')

print(f'Email : \t {email}\n')
print('-'*80)
print(f'\nSubject : \t {sub}\n')
print('-'*80)
print(f'\nText : \t\t {tex}')
```

```
Email :          ['mantis', 'netcom', 'mantis']

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

Subject :        ['Alt Atheism FAQ Atheist Resources']

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

Text :           resources december atheist resources addresses atheist organizations usa freedom from religion f
oundation fish bumper stickers and assorted other atheist paraphernalia are available from the freedom from relig
ion foundation the us evolution designs evolution designs sell the fish fish symbol like the ones stick their car
```

s but with feet and the word written inside the deluxe moulded plastic fish postpaid the us people the san franci
sco bay area can get fish from try mailing for net people who directly the price per fish american atheist press
aap publish various atheist books critiques the bible lists biblical and on one such book the bible handbook wp a
nd gw american atheist press pp isbn edition bible absurdities atrocities contains the bible contradicts itself a
ap based the king version the bible austin prometheus books sell books including holy horrors alternate address p
rometheus books for humanism organization promoting black secular humanism and uncovering the history black freet
hought they publish quarterly newsletter aah examiner press association national secular society street holloway
road london london british humanist association south place ethical society lamb wcr red lion square london wcr f
ax the national secular society publish the freethinker monthly magazine founded ev bund der und germany ibka pub
lish miz materialien und zur zeit politisches journal der und ibka ev mizvertrieb postfach germany for atheist bo
oks write ibdk bucherdienst der germany books fiction thomas disch the claus compromise short story the ultimate
proof that exists all characters and events are fictitious any similarity living dead gods uh well walter miller
canticle for leibowitz one gem this post atomic doomsday novel the monks who spent their lives copying blueprints
from saint leibowitz filling the sheets paper with ink and leaving white lines and letters edgar pangborn davy po
st atomic doomsday novel set clerical states the church for example forbids that anyone produce describe use any
substance containing atoms philip dick wrote many philosophical and short stories and novels his stories are biza
rre times but very approachable wrote mainly sf but wrote about people truth and religion rather than technology
although often believed that had met some sort remained sceptical amongst his novels the following are some galac
tic pothealer fallible alien deity summons group craftsmen and women remote planet raise giant cathedral from ben
eath the oceans when the deity begins demand faith from the earthers pothealer unable comply polished ironic and
amusing novel maze death noteworthy for its description religion valis the schizophrenic hero searches for the hi
dden mysteries gnostic ity after reality fired into his brain pink laser beam unknown but possibly divine origin
accompanied his dogmatic and dismissively atheist friend and assorted other odd characters the divine invasion in
vades making young woman pregnant she returns from another star system unfortunately she terminally ill and must
assisted dead man whose brain wired hour easy listening music margaret atwood the handmaid story based the premis
e that the congress mysteriously assassinated and quickly take charge the nation set right again the book the dia
ry woman life she tries live under the new theocracy women right own property revoked and their bank accounts are
closed sinful luxuries are outlawed and the radio only used for readings from the bible crimes are punished docto
rs who performed legal abortions the old world are hunted down and hanged writing style difficult get used first
but the tale grows more and more chilling goes on various authors the bible this somewhat dull and rambling work
has often been criticized however probably worth reading only that you will know what all the fuss about exists m
any different versions make sure you get the one true version peter rosa vicars christ although seems even cathol
ic this very enlighting history papal immoralities adulteries fallacies etc german gottes erste dunkle seite des
droemerknaur michael martin philosophical philadelphia usa detailed and scholarly justification atheism contains
outstanding appendix defining terminology and usage this tendentious area argues both for negative atheism ie the
nonbelief the existence god and also for positive atheism the belief the nonexistence god includes great refutati
ons the most challenging arguments for god particular attention paid refuting contempory theists such and swinbur
ne pages isbn the case against ity comprehensive critique ity which considers the best contemporary defences ity
and demonstrates that they are unsupportable andor incoherent pages isbn james turner the johns hopkins universit
y press baltimore md usa subtitled the origins unbelief america examines the way which unbelief became mainstream
alternative worldview focusses the period and while considering france and britain the emphasis american and part
icularly new_england developments neither religious history secularization atheism is rather the intellectual his
tory the fate single idea the belief that exists pages isbn george seldes the great thoughts antine books new_yor
k usa dictionary quotations different kind concentrating statements and writings which explicitly implicitly pres
ent the person philosophy and worldview includes obscure opinions from many people for some popular observations
traces the way which various people expressed and twisted the idea over the centuries quite number the quotations
are derived from cardiff what religion and views religion pages isbn the existence oxford this book the second vo
lume trilogy that began with the coherence theism this work swinburne attempts construct series inductive argumen
ts for the existence his arguments which are somewhat tendentious and rely upon the imputation late century weste
rn values and aesthetics which supposedly simple can conceived were decisively rejected the miracle theism the re
vised edition the existence swinburne includes appendix which makes somewhat incoherent attempt rebut mackie the
miracle theism oxford this volume contains comprehensive review the principal arguments for and against the exist
ence ranges from the classical philosophical positions descartes anselm al through the moral arguments newman kan
t and the recent restatements the classical theses and swinburne also addresses those positions which push the co
ncept beyond the realm the rational such those kierkegaard and well replacements for such axiarchism the book del
ight read less formalistic and better written than works and refreshingly direct when compared with the handwavin
g swinburne haught holy illustrated history religious murder and madness prometheus books looks religious persecu
tion from ancient times the present day and not only library congress catalog card number norm allen jr african a
merican anthology see the listing for african americans for humanism above gordon stein an anthology atheism and
rationalism prometheus books anthology covering wide range subjects including the devil and morality and the hist
ory freethought comprehensive bibliography edmund cohen the mind the prometheus books study why people become and
what effect has them net resources there small mailbased archive server mantiscouk which carries archives old art
icles and assorted other files for more information send mail saying help send atheismindex and will mail back re
ply mathew

In [9]:

```
raw_data, preprocessed_email, subject_list, whole_text, tg_clas = [], [], [], [], []

for file in tqdm(file_name_list):
    raw, email, subj, text, tg_class = preprocess(file)

    raw_data.append(' '.join(raw))
    preprocessed_email.append(' '.join(email))
    subject_list.append(' '.join(subj))
    whole_text.append(text)
    tg_clas.append((tg_class))

data = pd.DataFrame({'text' : raw_data, 'class_' : tg_clas, 'preprocessed_text' : whole_text,
             'preprocessed_subject' : subject_list , 'preprocessed_emails' : preprocessed_email,
            })
```

```
100%|██████████| 18828/18828 [1:01:15<00:00,  5.12it/s]
```

In [10]:

```
data.columns
```

Out[10]:
```
Index(['text', 'class_', 'preprocessed_text', 'preprocessed_subject',
       'preprocessed_emails'],
      dtype='object')
```

In [11]:
```
data.shape
```

Out[11]:
```
(18828, 5)
```

In [12]:
```
data.head()
```

Out[12]:

	text	class_	preprocessed_text	preprocessed_subject	preprocessed_emails
0	From: gsh7w@fermi.clas.Virginia.EDU (Greg Henn...	talk.politics.misc	article there big difference between running o...	Why not concentrate on child molesters	fermi clas virginia edu optilink optilink virg...
1	From: kramersc@expert.cc.purdue.edu (Scott Kra...	soc.religion.christian	article how much better get wisdom than gold c...	Daily Verse	expert purdue edu athos rutgers edu gvg47 gvg tek
2	From: aas7@po.CWRU.Edu (Andrew A. Spencer)\nSu...	rec.autos	previous article in article not familiar with ...	RFI Art of clutchless shifting	cwru edu ccwf utexas edu psuvm psu edu psuvm p...
3	From: schuch@phx.mcd.mot.com (John Schuch)\nSu...	sci.electronics	article how does the radio electronics free in...	Radio Electronics Free information card	phx mcd mot acsu buffalo edu ubvmsb buffalo edu
4	From: holthaus@news.weeg.uiowa.edu (James R. H...	sci.crypt	what the status cruptology for private citizen...	Cryptology in the world	news weeg uiowa edu uiowa edu

In [13]:
```
data['preprocessed_data'] = data['preprocessed_text'] + ' ' + data['preprocessed_subject'] + ' ' \
                            + data['preprocessed_emails']

data.head()
```

Out[13]:

	text	class_	preprocessed_text	preprocessed_subject	preprocessed_emails	preprocessed_data
0	From: gsh7w@fermi.clas.Virginia.EDU (Greg Henn...	talk.politics.misc	article there big difference between running o...	Why not concentrate on child molesters	fermi clas virginia edu optilink optilink virg...	article there big difference between running o...
1	From: kramersc@expert.cc.purdue.edu (Scott Kra...	soc.religion.christian	article how much better get wisdom than gold c...	Daily Verse	expert purdue edu athos rutgers edu gvg47 gvg tek	article how much better get wisdom than gold c...
2	From: aas7@po.CWRU.Edu (Andrew A. Spencer)\nSu...	rec.autos	previous article in article not familiar with ...	RFI Art of clutchless shifting	cwru edu ccwf utexas edu psuvm psu edu psuvm p...	previous article in article not familiar with ...
3	From: schuch@phx.mcd.mot.com (John Schuch)\nSu...	sci.electronics	article how does the radio electronics free in...	Radio Electronics Free information card	phx mcd mot acsu buffalo edu ubvmsb buffalo edu	article how does the radio electronics free in...
4	From: holthaus@news.weeg.uiowa.edu (James R. H...	sci.crypt	what the status cruptology for private citizen...	Cryptology in the world	news weeg uiowa edu uiowa edu	what the status cruptology for private citizen...

In [14]:
```
# Saving data as Pickle file

with open('cnn_data.pickle', 'wb') as file:
    pickle.dump(data, file)

cleaning_ = time()
print(f'Total Time : {round((cleaning_ - start_)/60, 2)} Min')
```

```
Total Time : 61.83 Min
```

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.

In [1]:

```
import os
import re
import pickle
import datetime
from tqdm import tqdm
from time import time

import numpy as np
import pandas as pd

import plotly.express as px
import plotly.graph_objects as go
import matplotlib.pyplot as plt

import string
import nltk
from nltk import ne_chunk
from nltk.tree import Tree
from nltk.tag import pos_tag
from nltk import word_tokenize

from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

import tensorflow
from tensorflow.keras import Input
from tensorflow.keras import Model
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Conv1D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import MaxPool1D
from tensorflow.keras.layers import Embedding
from tensorflow.keras.layers import concatenate

from tensorflow.keras.callbacks import Callback
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import ReduceLROnPlateau

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.utils import plot_model
from tensorflow.keras.utils import to_categorical

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
%load_ext tensorboard

import warnings
warnings.filterwarnings("ignore")

start_ = time()
```

```
2022-07-09 10:02:36.513747: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynami
c library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object file: No such file or direct
ory
2022-07-09 10:02:36.513790: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if y
ou do not have a GPU set up on your machine.
```

In [2]:
```
modeling_1 = time()

# Loading Pickle data file

with open('cnn_data.pickle', 'rb') as file:
    data = pickle.load(file)
```

In [3]:
```
Y = data.class_
X = data.drop('class_', axis=1)
```

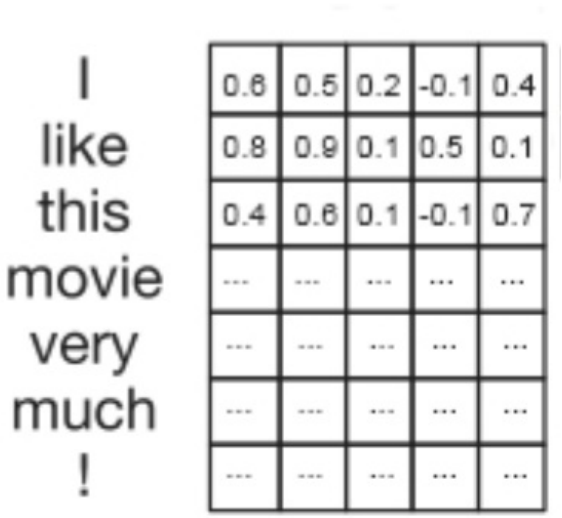
In [4]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.25, stratify = Y,
                                                    random_state = 42)
```

In [5]:
```
# https://stackoverflow.com/a/61550151

label_encoder = LabelEncoder()

label_vec_train = label_encoder.fit_transform(y_train)
y_train_labeled = to_categorical(label_vec_train, num_classes = 20)

label_vec_test = label_encoder.transform(y_test)
y_test_labeled = to_categorical(label_vec_test, num_classes = 20)

print(y_test_labeled.shape)
```

```
(4707, 20)
```

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

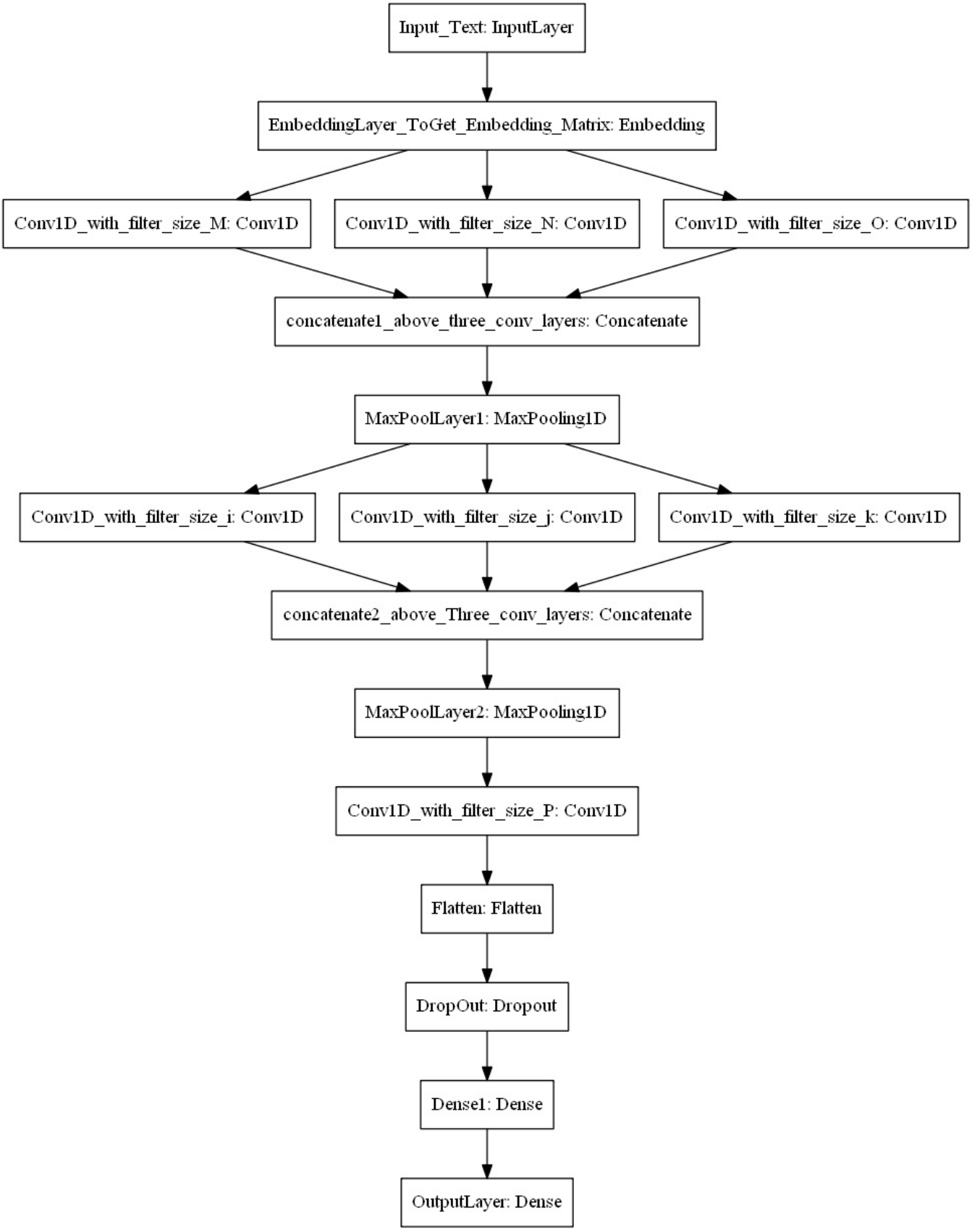


Ref: https://i.imgur.com/kiVQuk1.png

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.

Acceptance criteria

Model	Accuracy
Model 1	70 %
Model 2	10 %

In [6]:
```
# https://towardsdatascience.com/nlp-preparing-text-for-deep-learning-model-using-tensorflow2-461428138657

t = Tokenizer(filters='!"#$%&()*+,-/:;<=>?@[\\]^`{|}~\t\n', oov_token='<OOV>')
t.fit_on_texts(X_train['preprocessed_data'])

#Sequencing

sequencing_docs_train= t.texts_to_sequences(X_train['preprocessed_data'])
sequencing_docs_test= t.texts_to_sequences(X_test['preprocessed_data'])
```

In [7]:
```
max_seq_len = []

for i in sequencing_docs_train:
    max_seq_len.append(len(i))

print(f'Maximum sequence / padding size : {max(max_seq_len)}')
```

```
Maximum sequence / padding size : 11646
```

In [8]:
```
# https://www.kaggle.com/code/mirhyun0508/basic-eda-cleaning-and-glove-copy?scriptVersionId=97097810&cellId=68

embedding_dict = {}

with open('glove/glove.6B.100d.txt') as file:
    for line in file:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embedding_dict[word] = coefs

print(f'Vocab Size : {int(len(embedding_dict) / 1000)}K')
```

```
Vocab Size : 400K
```

In [9]:
```
num_words = len(t.word_index) + 1
print(f'Number of words in the Vocab : {num_words}')

embedding_matrix = np.zeros((num_words, 100))

for word, i in t.word_index.items():
    word_vec = embedding_dict.get(word)

    if word_vec is not None:
        embedding_matrix[i] = word_vec
```

```
Number of words in the Vocab : 93985
```

Choosing `MAXLEN` as **1050** after lots of iterations. Larger length takes more time to execute the codes.

In [10]:
```
MAXLEN = 1050

padding_doc_train = pad_sequences(sequencing_docs_train, maxlen = MAXLEN, padding = 'post')
padding_doc_test = pad_sequences(sequencing_docs_test, maxlen = MAXLEN, padding = 'post')
```

GloVe: Global Vectors for Word Representation

https://nlp.stanford.edu/projects/glove/ : Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download)

https://nlp.stanford.edu/data/glove.6B.zip

https://kaggle.com/datasets/rtatman/glove-global-vectors-for-word-representation

https://kaggle.com/code/shahules/basic-eda-cleaning-and-glove

In [11]:
```
class f1_score_(Callback):

    def __init__(self, test_x, test_y):
        self.x = test_x
        self.y = test_y

    def on_epoch_end(self, epoch, logs = {}):

        y_pred = (np.array(self.model.predict(self.x))).round()
        f1 = f1_score(self.y, y_pred, average = 'micro')
        print(f' MicroF1_Test:{round(f1, 3)}')
```

In [12]:
```
filpath = 'model_save/epo_{epoch:02d}-accu_{val_accuracy:.4f}.hdf5'
model_check_point = ModelCheckpoint(filpath, monitor = 'val_accuracy', patience = 1, verbose = 1)
```

In [13]:
```
tensorflow.random.set_seed(42)
tensorflow.keras.backend.clear_session()

input_ = Input(shape = (MAXLEN,), dtype = 'int32')
embed = Embedding(input_dim = num_words, output_dim = 100, input_length = MAXLEN, trainable = True)(input_)

x1 = Conv1D(3, 7, kernel_initializer = 'glorot_uniform', activation = 'relu')(embed)
x2 = Conv1D(3, 8, kernel_initializer = 'glorot_uniform', activation = 'relu')(embed)
x3 = Conv1D(3, 6, kernel_initializer = 'glorot_uniform', activation = 'relu')(embed)

concat_1 = concatenate([x1, x2, x3], axis = 1)

max_p1 = MaxPool1D(pool_size = 2)(concat_1)

x4 = Conv1D(3, 9, kernel_initializer = 'glorot_uniform', activation = 'relu')(max_p1)
x5 = Conv1D(3, 4, kernel_initializer = 'glorot_uniform', activation = 'relu')(max_p1)
x6 = Conv1D(3, 5, kernel_initializer = 'glorot_uniform', activation = 'relu')(max_p1)

concat_2 = concatenate([x4, x5, x6], axis =1)

max_p2 = MaxPool1D(pool_size = 2)(concat_2)

drop_1 = Dropout(0.5)(max_p2)

max_p3 = MaxPool1D(pool_size = 2)(drop_1)

con1 = Conv1D(3, 12, kernel_initializer = 'glorot_uniform', activation = 'relu')(max_p3)

flat = Flatten()(con1)

drop_2 = Dropout(0.5)(flat)

dense_1 = Dense(100,activation='relu')(drop_2)

out_ = Dense(20, activation='softmax', kernel_initializer='random_uniform')(dense_1)

model_1 = Model(inputs = input_, outputs = out_)

model_1.summary()
```

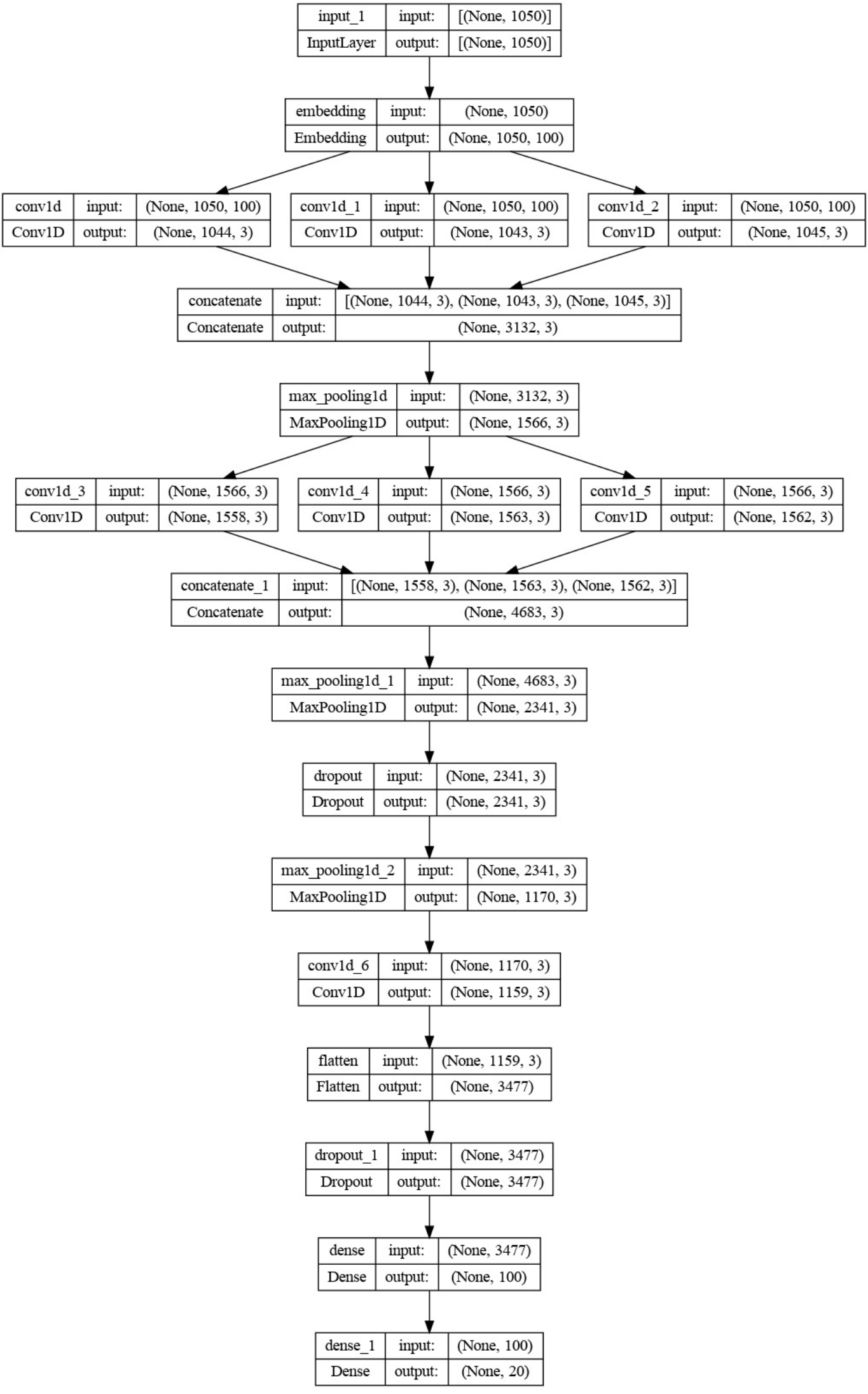
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 1050)]            0         
```

```
embedding (Embedding)          (None, 1050, 100)    9398500     ['input_1[0][0]']

conv1d (Conv1D)                (None, 1044, 3)      2103        ['embedding[0][0]']

conv1d_1 (Conv1D)              (None, 1043, 3)      2403        ['embedding[0][0]']

conv1d_2 (Conv1D)              (None, 1045, 3)      1803        ['embedding[0][0]']

concatenate (Concatenate)      (None, 3132, 3)      0           ['conv1d[0][0]',
                                                                 'conv1d_1[0][0]',
                                                                 'conv1d_2[0][0]']

max_pooling1d (MaxPooling1D)   (None, 1566, 3)      0           ['concatenate[0][0]']

conv1d_3 (Conv1D)              (None, 1558, 3)      84          ['max_pooling1d[0][0]']

conv1d_4 (Conv1D)              (None, 1563, 3)      39          ['max_pooling1d[0][0]']

conv1d_5 (Conv1D)              (None, 1562, 3)      48          ['max_pooling1d[0][0]']

concatenate_1 (Concatenate)    (None, 4683, 3)      0           ['conv1d_3[0][0]',
                                                                 'conv1d_4[0][0]',
                                                                 'conv1d_5[0][0]']

max_pooling1d_1 (MaxPooling1D)  (None, 2341, 3)     0           ['concatenate_1[0][0]']

dropout (Dropout)              (None, 2341, 3)      0           ['max_pooling1d_1[0][0]']

max_pooling1d_2 (MaxPooling1D)  (None, 1170, 3)     0           ['dropout[0][0]']

conv1d_6 (Conv1D)              (None, 1159, 3)      111         ['max_pooling1d_2[0][0]']

flatten (Flatten)              (None, 3477)         0           ['conv1d_6[0][0]']

dropout_1 (Dropout)            (None, 3477)         0           ['flatten[0][0]']

dense (Dense)                  (None, 100)          347800      ['dropout_1[0][0]']

dense_1 (Dense)                (None, 20)           2020        ['dense[0][0]']

==================================================================================================
Total params: 9,754,911
Trainable params: 9,754,911
Non-trainable params: 0
__________________________________________________________________________________________________
2022-07-09 10:02:47.532362: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynami
c library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open shared object file: No such file or directory
2022-07-09 10:02:47.532390: W tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to cuInit: UNKNOWN
ERROR (303)
2022-07-09 10:02:47.532407: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] kernel driver does not app
ear to be running on this host (ubuntu): /proc/driver/nvidia/version does not exist
2022-07-09 10:02:47.532972: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimi
zed with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critica
l operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

In [14]:

```
plot_model(model_1, to_file='model_1.png', show_shapes = True)
```

Out[14]:


input_1 | input: | [(None, 1050)]
InputLayer | output: | [(None, 1050)]
embedding | input: | (None, 1050)
Embedding | output: | (None, 1050, 100)
conv1d | input: | (None, 1050, 100)
Conv1D | output: | (None, 1044, 3)
conv1d_1 | input: | (None, 1050, 100)
Conv1D | output: | (None, 1043, 3)
conv1d_2 | input: | (None, 1050, 100)
Conv1D | output: | (None, 1045, 3)
concatenate | input: | [(None, 1044, 3), (None, 1043, 3), (None, 1045, 3)]
Concatenate | output: | (None, 3132, 3)
max_pooling1d | input: | (None, 3132, 3)
MaxPooling1D | output: | (None, 1566, 3)
conv1d_3 | input: | (None, 1566, 3)
Conv1D | output: | (None, 1558, 3)
conv1d_4 | input: | (None, 1566, 3)
Conv1D | output: | (None, 1563, 3)
conv1d_5 | input: | (None, 1566, 3)
Conv1D | output: | (None, 1562, 3)
concatenate_1 | input: | [(None, 1558, 3), (None, 1563, 3), (None, 1562, 3)]
Concatenate | output: | (None, 4683, 3)
max_pooling1d_1 | input: | (None, 4683, 3)
MaxPooling1D | output: | (None, 2341, 3)
dropout | input: | (None, 2341, 3)
Dropout | output: | (None, 2341, 3)
max_pooling1d_2 | input: | (None, 2341, 3)
MaxPooling1D | output: | (None, 1170, 3)
conv1d_6 | input: | (None, 1170, 3)
Conv1D | output: | (None, 1159, 3)
flatten | input: | (None, 1159, 3)
Flatten | output: | (None, 3477)
dropout_1 | input: | (None, 3477)
Dropout | output: | (None, 3477)
dense | input: | (None, 3477)
Dense | output: | (None, 100)
dense_1 | input: | (None, 100)
Dense | output: | (None, 20)

In [15]:

```
EPOCH = 25

optimizer = Adam(learning_rate = 0.01)

log_dir = 'logs/model_1/' + datetime.datetime.now().strftime('%Y%m%d_%H%M')
tensorboard_callback = TensorBoard(log_dir = log_dir, histogram_freq = 1, write_graph = True)

early_stopping = EarlyStopping(monitor = 'accuracy', patience = 2, verbose = 1)
epoch_1 = ReduceLROnPlateau(monitor = 'accuracy', factor = 0.9, verbose = 1, patience = 1)

F1score = f1_score_(padding_doc_test, y_test_labeled)

callback_list = [tensorboard_callback, early_stopping, epoch_1, F1score]

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

history_1 = model_1.fit(padding_doc_train, y_train_labeled, epochs = EPOCH,
                  validation_data = (padding_doc_test, y_test_labeled),
                  verbose = 1, callbacks = callback_list)
```

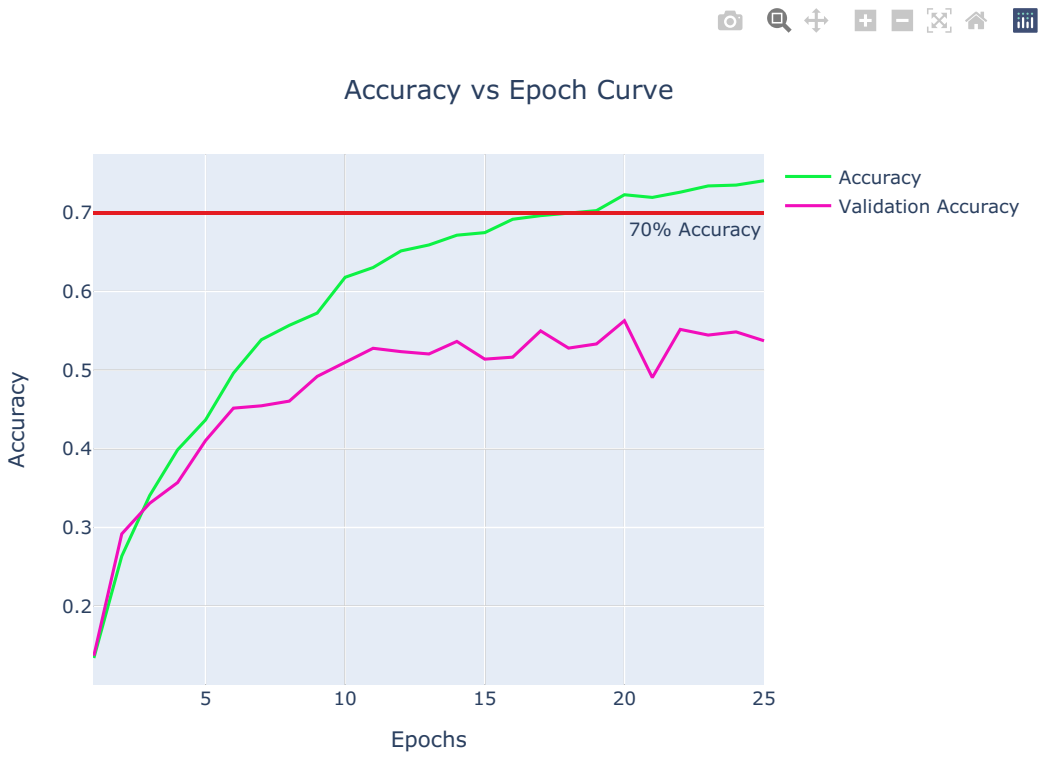
```
Epoch 1/25
442/442 [==============================] - ETA: 0s - loss: 2.6728 - accuracy: 0.1342

2022-07-09 10:04:04.813971: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 2255640000 exceeds 10% of free system memory.

148/148 [==============================] - 3s 18ms/step
 MicroF1_Test:0.025
442/442 [==============================] - 80s 180ms/step - loss: 2.6728 - accuracy: 0.1342 - val_loss: 2.6638 - val_accuracy: 0.1370 - lr: 0.0100
Epoch 2/25
442/442 [==============================] - ETA: 0s - loss: 2.1386 - accuracy: 0.2636

2022-07-09 10:05:22.713497: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 2255640000 exceeds 10% of free system memory.

148/148 [==============================] - 3s 20ms/step
 MicroF1_Test:0.156
442/442 [==============================] - 78s 177ms/step - loss: 2.1386 - accuracy: 0.2636 - val_loss: 2.0295 - val_accuracy: 0.2919 - lr: 0.0100
Epoch 3/25
442/442 [==============================] - ETA: 0s - loss: 1.8456 - accuracy: 0.3406
2022-07-09 10:06:41.238644: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 2255640000 exceeds 10% of free system memory.

148/148 [==============================] - 3s 20ms/step
 MicroF1_Test:0.199
442/442 [==============================] - 79s 178ms/step - loss: 1.8456 - accuracy: 0.3406 - val_loss: 1.8667 - val_accuracy: 0.3308 - lr: 0.0100
Epoch 4/25
442/442 [==============================] - ETA: 0s - loss: 1.6450 - accuracy: 0.3988

2022-07-09 10:07:59.808225: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 2255640000 exceeds 10% of free system memory.
148/148 [==============================] - 3s 20ms/step
 MicroF1_Test:0.253
442/442 [==============================] - 79s 178ms/step - loss: 1.6450 - accuracy: 0.3988 - val_loss: 1.8531 - val_accuracy: 0.3571 - lr: 0.0100
Epoch 5/25
442/442 [==============================] - ETA: 0s - loss: 1.5157 - accuracy: 0.4367

2022-07-09 10:09:18.705296: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 2255640000 exceeds 10% of free system memory.

148/148 [==============================] - 3s 20ms/step
 MicroF1_Test:0.358
442/442 [==============================] - 79s 178ms/step - loss: 1.5157 - accuracy: 0.4367 - val_loss: 1.6822 - val_accuracy: 0.4105 - lr: 0.0100
Epoch 6/25
148/148 [==============================] - 3s 20ms/step
 MicroF1_Test:0.44
442/442 [==============================] - 80s 182ms/step - loss: 1.3517 - accuracy: 0.4961 - val_loss: 1.5996 - val_accuracy: 0.4517 - lr: 0.0100
Epoch 7/25
148/148 [==============================] - 3s 20ms/step
 MicroF1_Test:0.423
442/442 [==============================] - 79s 179ms/step - loss: 1.2546 - accuracy: 0.5386 - val_loss: 1.5426 - val_accuracy: 0.4546 - lr: 0.0100
Epoch 8/25
148/148 [==============================] - 3s 20ms/step
 MicroF1_Test:0.472
442/442 [==============================] - 78s 177ms/step - loss: 1.1861 - accuracy: 0.5569 - val_loss: 1.5988 -
```

```
val_accuracy: 0.4606 - lr: 0.0100
Epoch 9/25
148/148 [==============================] - 3s 20ms/step
 MicroF1_Test:0.499
442/442 [==============================] - 78s 177ms/step - loss: 1.1292 - accuracy: 0.5725 - val_loss: 1.4685 -
val_accuracy: 0.4920 - lr: 0.0100
Epoch 10/25
148/148 [==============================] - 3s 20ms/step
 MicroF1_Test:0.523
442/442 [==============================] - 78s 176ms/step - loss: 1.0066 - accuracy: 0.6179 - val_loss: 1.4585 -
val_accuracy: 0.5092 - lr: 0.0100
Epoch 11/25
148/148 [==============================] - 3s 20ms/step
 MicroF1_Test:0.549
442/442 [==============================] - 78s 177ms/step - loss: 0.9561 - accuracy: 0.6303 - val_loss: 1.4654 -
val_accuracy: 0.5277 - lr: 0.0100
Epoch 12/25
148/148 [==============================] - 3s 21ms/step
 MicroF1_Test:0.548
442/442 [==============================] - 80s 180ms/step - loss: 0.9103 - accuracy: 0.6514 - val_loss: 1.5254 -
val_accuracy: 0.5235 - lr: 0.0100
Epoch 13/25
148/148 [==============================] - 3s 20ms/step
 MicroF1_Test:0.534
442/442 [==============================] - 79s 178ms/step - loss: 0.8786 - accuracy: 0.6590 - val_loss: 1.4996 -
val_accuracy: 0.5205 - lr: 0.0100
Epoch 14/25
148/148 [==============================] - 3s 20ms/step
 MicroF1_Test:0.552
442/442 [==============================] - 78s 178ms/step - loss: 0.8554 - accuracy: 0.6714 - val_loss: 1.4687 -
val_accuracy: 0.5364 - lr: 0.0100
Epoch 15/25
148/148 [==============================] - 3s 20ms/step
 MicroF1_Test:0.528
442/442 [==============================] - 78s 177ms/step - loss: 0.8638 - accuracy: 0.6747 - val_loss: 1.5457 -
val_accuracy: 0.5139 - lr: 0.0100
Epoch 16/25
148/148 [==============================] - 3s 20ms/step
 MicroF1_Test:0.533
442/442 [==============================] - 79s 178ms/step - loss: 0.8058 - accuracy: 0.6917 - val_loss: 1.5987 -
val_accuracy: 0.5165 - lr: 0.0100
Epoch 17/25
148/148 [==============================] - 3s 20ms/step
 MicroF1_Test:0.571
442/442 [==============================] - 78s 176ms/step - loss: 0.7928 - accuracy: 0.6963 - val_loss: 1.6182 -
val_accuracy: 0.5498 - lr: 0.0100
Epoch 18/25
148/148 [==============================] - 3s 20ms/step
 MicroF1_Test:0.544
442/442 [==============================] - 78s 177ms/step - loss: 0.7915 - accuracy: 0.7000 - val_loss: 1.6342 -
val_accuracy: 0.5279 - lr: 0.0100
Epoch 19/25
148/148 [==============================] - 3s 20ms/step
 MicroF1_Test:0.553
442/442 [==============================] - 78s 176ms/step - loss: 0.7742 - accuracy: 0.7026 - val_loss: 1.5649 -
val_accuracy: 0.5332 - lr: 0.0100
Epoch 20/25
148/148 [==============================] - 3s 20ms/step
 MicroF1_Test:0.588
442/442 [==============================] - 78s 176ms/step - loss: 0.7367 - accuracy: 0.7228 - val_loss: 1.7243 -
val_accuracy: 0.5628 - lr: 0.0100
Epoch 21/25
442/442 [==============================] - ETA: 0s - loss: 0.7456 - accuracy: 0.7194
Epoch 21: ReduceLROnPlateau reducing learning rate to 0.008999999798834325.
148/148 [==============================] - 3s 20ms/step
 MicroF1_Test:0.505
442/442 [==============================] - 78s 176ms/step - loss: 0.7456 - accuracy: 0.7194 - val_loss: 1.7138 -
val_accuracy: 0.4905 - lr: 0.0100
Epoch 22/25
148/148 [==============================] - 3s 20ms/step
 MicroF1_Test:0.575
442/442 [==============================] - 78s 177ms/step - loss: 0.7262 - accuracy: 0.7261 - val_loss: 1.6759 -
val_accuracy: 0.5517 - lr: 0.0090
Epoch 23/25
148/148 [==============================] - 3s 20ms/step
 MicroF1_Test:0.562
442/442 [==============================] - 78s 177ms/step - loss: 0.7055 - accuracy: 0.7340 - val_loss: 1.7220 -
val_accuracy: 0.5445 - lr: 0.0090
Epoch 24/25
148/148 [==============================] - 3s 20ms/step
 MicroF1_Test:0.573
442/442 [==============================] - 78s 177ms/step - loss: 0.7025 - accuracy: 0.7351 - val_loss: 1.8342 -
val_accuracy: 0.5485 - lr: 0.0090
Epoch 25/25
148/148 [==============================] - 3s 21ms/step
 MicroF1_Test:0.559
442/442 [==============================] - 79s 178ms/step - loss: 0.6807 - accuracy: 0.7407 - val_loss: 1.6930 -
val_accuracy: 0.5373 - lr: 0.0090
```

In [16]:
```
model_1.save_weights('best_model_1.h5')
```

In [17]:
```
# https://plotly.com/python/line-charts/

leng_ = len(history_1.history['accuracy']) + 1

fig = go.Figure()

fig.add_trace(go.Scatter(y = history_1.history['accuracy'], name = 'Accuracy',
                         line_color = '#0DF244', x = list(range(1, leng_))))
fig.add_trace(go.Scatter(y = history_1.history['val_accuracy'], name = 'Validation Accuracy',
                         line_color = '#F20DBB', x = list(range(1, leng_))))

fig.update_layout(title = {'text': 'Accuracy vs Epoch Curve', 'y':0.9, 'x':0.5,
                         'xanchor': 'center', 'yanchor': 'top'}, xaxis_title = 'Epochs',
                         yaxis_title = 'Accuracy')

fig.add_hline(y=0.70, annotation_text = '70% Accuracy',
              annotation_position = 'bottom right', line_color = '#E41B22')

print(f"Accuracy of Model_1 : {round(history_1.history['accuracy'][-1], 2) * 100}%")

fig.show()
```

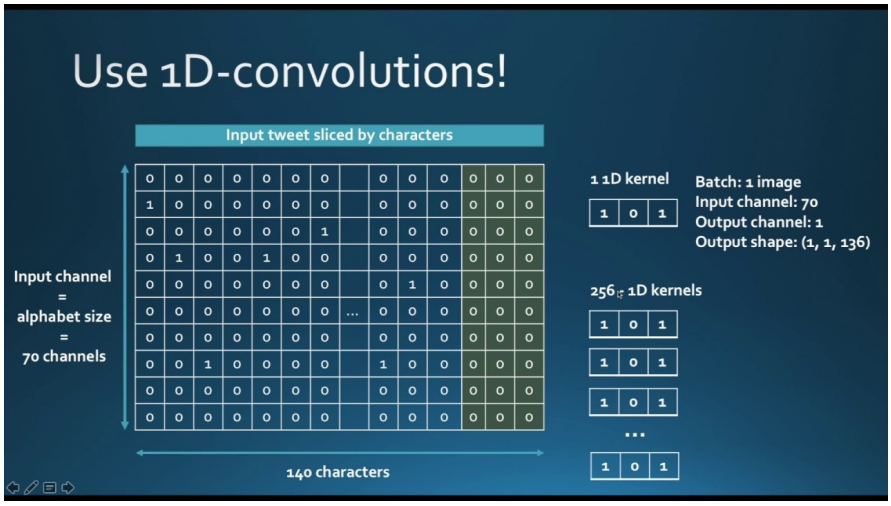
```
Accuracy of Model_1 : 74.0%
```



In [18]:
```
model_1 = time()

print(f'Model_1 Time : {round((model_1 - start_) / 60, 2)} Min')
```

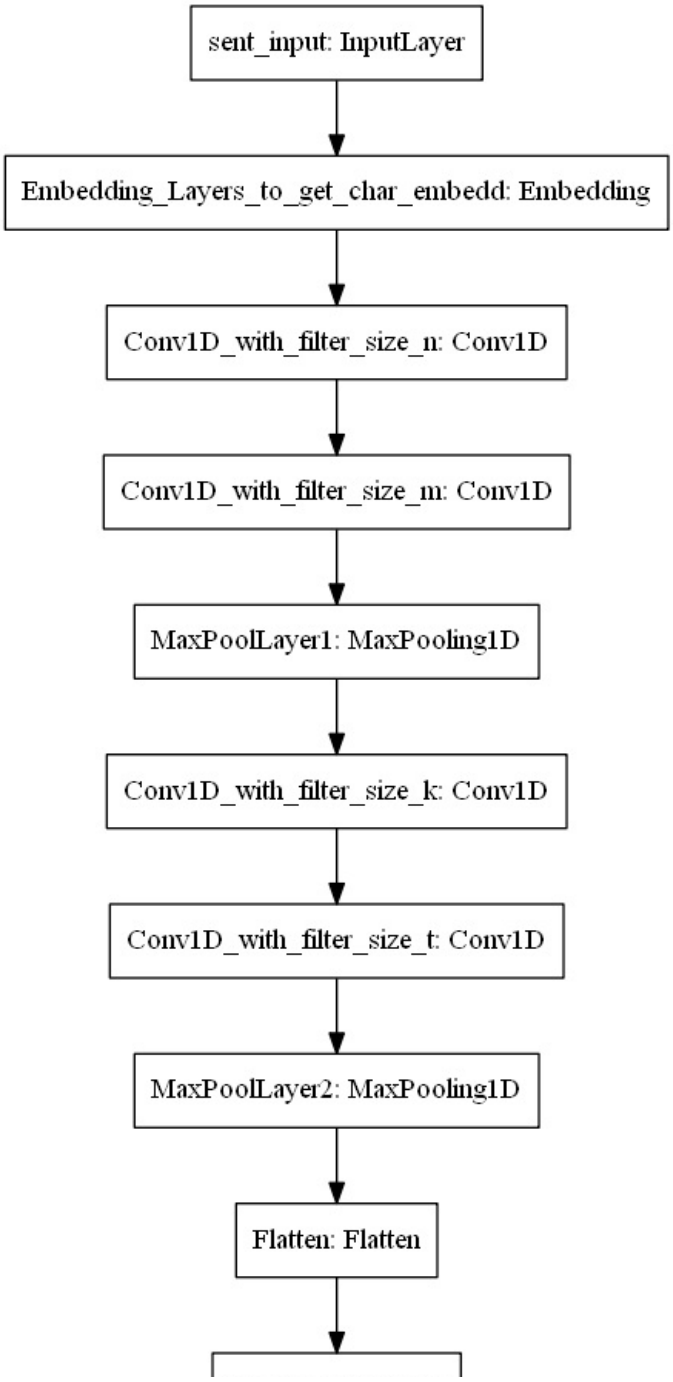
```
Model_1 Time : 32.89 Min
```

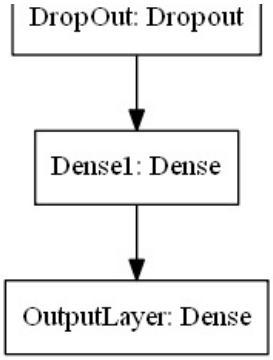
Model-2 : Using 1D convolutions with character embedding



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





In [19]:
```
# https://towardsdatascience.com/character-level-cnn-with-keras-50391c3adf33

tocken = Tokenizer(filters = '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n', char_level = True, oov_token = 'UNK')

tocken.fit_on_texts(X_train['preprocessed_data'])

alphas = string.ascii_lowercase

char_dict = {}
for i, char in enumerate(alphas):
    if char not in char_dict:
         char_dict[char] = i +1

# tocken.word_index = char_dict.copy()
tocken.word_index = char_dict
tocken.word_index[tocken.oov_token] = max(char_dict.values()) + 1
```

In [20]:
```
# Convert string to index

sequencing_chardocs_train = tocken.texts_to_sequences(X_train['preprocessed_data'])
sequencing_chardocs_test= tocken.texts_to_sequences(X_test['preprocessed_data'])

len(sequencing_chardocs_test)
```

Out[20]:
```
4707
```

In [21]:
```
max_seq_len = []
for i in sequencing_chardocs_train:
    max_seq_len.append(len(i))
print(f'Maximum sequence / padding size : {max(max_seq_len)}')

char1_num_words = len(tocken.word_index) + 1
print(f'Number of characters : {char1_num_words}')
```

```
Maximum sequence / padding size : 49625
Number of characters : 28
```

Choosing `MAXLEN` as **1500** after lots of iterations. Larger length takes more time to execute the codes.

Also `np.percentile(max_seq_len, 82)` ie, 82th percentile value of `max_seq_len` is **1492**

In [22]:
```
MAXLEN = 1500

padding_chardoc_train = pad_sequences(sequencing_chardocs_train, maxlen = MAXLEN, padding = 'post')
padding_chardoc_test = pad_sequences(sequencing_chardocs_test, maxlen = MAXLEN, padding = 'post')
```

In [23]:
```
# https://towardsdatascience.com/character-level-cnn-with-keras-50391c3adf33

embed_weights = []
embed_weights.append(np.zeros(char1_num_words))

for char, i in tocken.word_index.items():
    one_hot = np.zeros(char1_num_words)
    one_hot[i-1] = 1
    embed_weights.append(one_hot)

embed_weights = np.array(embed_weights)

print(embed_weights.shape)
```

```
(28, 28)
```

In [24]:

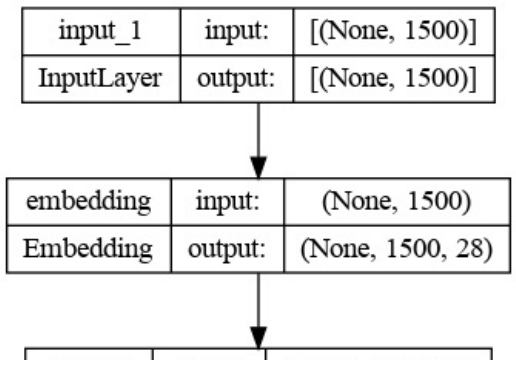
```
tensorflow.random.set_seed(42)
tensorflow.keras.backend.clear_session()

input_ = Input(shape = (MAXLEN,), dtype = 'int32')
embed = Embedding(input_dim = char1_num_words, output_dim = 28, input_length = MAXLEN, trainable = True)(input_)


conv1 = Conv1D(4, 11, kernel_initializer = 'glorot_uniform', activation = 'relu')(embed)
conv2 = Conv1D(4, 9, kernel_initializer = 'glorot_uniform', activation = 'relu')(conv1)

max_p1 = MaxPool1D(pool_size = 3)(conv2)

conv3 = Conv1D(6, 8, kernel_initializer = 'glorot_uniform', activation = 'relu')(max_p1)
conv4 = Conv1D(6, 7, kernel_initializer = 'glorot_uniform', activation = 'relu')(conv3)

max_p2 = MaxPool1D(pool_size = 3)(conv4)

flat = Flatten()(max_p2)

drop_1 = Dropout(0.2)(flat)

dense_2 = Dense(32,activation='tanh')(drop_1)

out_ = Dense(20, activation='softmax')(dense_2)

model_2 = Model(inputs = input_, outputs = out_)

model_2.summary()
```

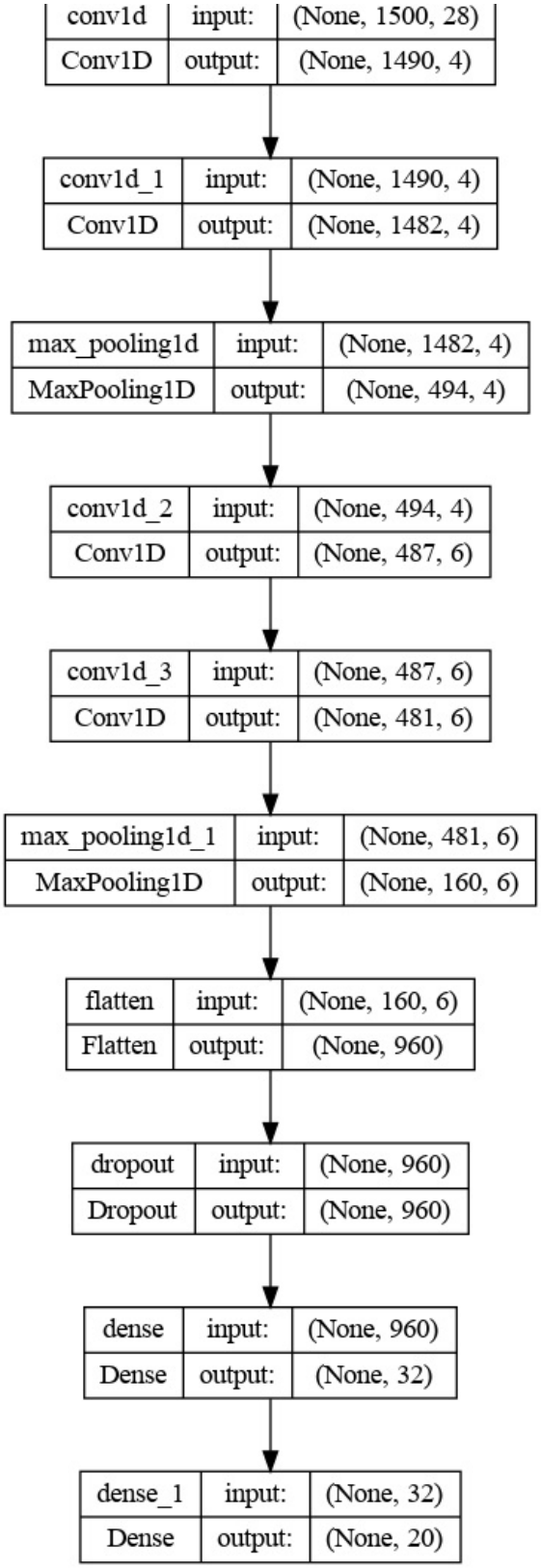
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 1500)]            0

 embedding (Embedding)       (None, 1500, 28)          784

 conv1d (Conv1D)             (None, 1490, 4)           1236

 conv1d_1 (Conv1D)           (None, 1482, 4)           148

 max_pooling1d (MaxPooling1D  (None, 494, 4)           0
 )

 conv1d_2 (Conv1D)           (None, 487, 6)            198

 conv1d_3 (Conv1D)           (None, 481, 6)            258

 max_pooling1d_1 (MaxPooling  (None, 160, 6)           0
 1D)

 flatten (Flatten)           (None, 960)               0

 dropout (Dropout)           (None, 960)               0

 dense (Dense)               (None, 32)                30752

 dense_1 (Dense)             (None, 20)                660

=================================================================
Total params: 34,036
Trainable params: 34,036
Non-trainable params: 0
_________________________________________________________________
```

In [25]:

```
plot_model(model_2, to_file='model_2.png', show_shapes = True)
```

Out[25]:





In [26]:

```
EPOCH = 15

optimizer = Adam(learning_rate = 0.01, epsilon = 1e-10)

F1score = f1_score_(padding_chardoc_test, y_test_labeled)

early_stopping = EarlyStopping(monitor = 'accuracy', patience = 3, verbose = 1)
epoch_1 = ReduceLROnPlateau(monitor = 'accuracy', factor = 0.9, verbose = 1, patience = 1)

log_dir = 'logs/model_2/' + datetime.datetime.now().strftime('%Y%m%d_%H%M')
tensorboard_callback = TensorBoard(log_dir = log_dir, histogram_freq = 1, write_graph = True)

callback_list = [tensorboard_callback, early_stopping, epoch_1, F1score] #F1score

model_2.compile(optimizer = optimizer, loss = 'categorical_crossentropy', metrics = ['accuracy'])

history_2 = model_2.fit(padding_chardoc_train, y_train_labeled, epochs = EPOCH,
                  validation_data = (padding_chardoc_test, y_test_labeled),
                  verbose = 1, callbacks = callback_list)
```

```
Epoch 1/15
148/148 [==============================] - 1s 7ms/step
 MicroF1_Test:0.0
442/442 [==============================] - 17s 37ms/step - loss: 2.9575 - accuracy: 0.0751 - val_loss: 2.9426 - v
al_accuracy: 0.0765 - lr: 0.0100
Epoch 2/15
148/148 [==============================] - 1s 7ms/step
 MicroF1_Test:0.0
442/442 [==============================] - 16s 36ms/step - loss: 2.9468 - accuracy: 0.0790 - val_loss: 2.9222 - v
al_accuracy: 0.0852 - lr: 0.0100
Epoch 3/15
148/148 [==============================] - 1s 7ms/step
 MicroF1_Test:0.005
442/442 [==============================] - 16s 36ms/step - loss: 2.8624 - accuracy: 0.0882 - val_loss: 2.8283 - v
al_accuracy: 0.0824 - lr: 0.0100
Epoch 4/15
148/148 [==============================] - 1s 7ms/step
 MicroF1_Test:0.004
442/442 [==============================] - 16s 36ms/step - loss: 2.7691 - accuracy: 0.0927 - val_loss: 2.7319 - v
al_accuracy: 0.1071 - lr: 0.0100
Epoch 5/15
148/148 [==============================] - 1s 7ms/step
 MicroF1_Test:0.0
442/442 [==============================] - 16s 35ms/step - loss: 2.7399 - accuracy: 0.0994 - val_loss: 2.7342 - v
al_accuracy: 0.1028 - lr: 0.0100
Epoch 6/15
148/148 [==============================] - 1s 7ms/step
 MicroF1_Test:0.012
442/442 [==============================] - 15s 35ms/step - loss: 2.7154 - accuracy: 0.1115 - val_loss: 2.7284 - v
al_accuracy: 0.1100 - lr: 0.0100
Epoch 7/15
441/442 [============================>.] - ETA: 0s - loss: 2.6892 - accuracy: 0.1106
Epoch 7: ReduceLROnPlateau reducing learning rate to 0.008999999798834325.
148/148 [==============================] - 1s 7ms/step
 MicroF1_Test:0.006
442/442 [==============================] - 15s 35ms/step - loss: 2.6892 - accuracy: 0.1107 - val_loss: 2.6809 - v
al_accuracy: 0.1120 - lr: 0.0100
Epoch 8/15
148/148 [==============================] - 1s 7ms/step
 MicroF1_Test:0.006
442/442 [==============================] - 16s 35ms/step - loss: 2.6654 - accuracy: 0.1209 - val_loss: 2.6796 - v
al_accuracy: 0.1147 - lr: 0.0090
Epoch 9/15
148/148 [==============================] - 1s 7ms/step
 MicroF1_Test:0.017
442/442 [==============================] - 16s 35ms/step - loss: 2.6475 - accuracy: 0.1253 - val_loss: 2.6591 - v
al_accuracy: 0.1147 - lr: 0.0090
Epoch 10/15
148/148 [==============================] - 1s 7ms/step
 MicroF1_Test:0.029
442/442 [==============================] - 15s 35ms/step - loss: 2.6418 - accuracy: 0.1301 - val_loss: 2.6740 - v
al_accuracy: 0.1256 - lr: 0.0090
Epoch 11/15
148/148 [==============================] - 1s 7ms/step
 MicroF1_Test:0.037
442/442 [==============================] - 16s 35ms/step - loss: 2.6250 - accuracy: 0.1397 - val_loss: 2.6490 - v
al_accuracy: 0.1307 - lr: 0.0090
Epoch 12/15
441/442 [============================>.] - ETA: 0s - loss: 2.6105 - accuracy: 0.1397
Epoch 12: ReduceLROnPlateau reducing learning rate to 0.008099999651312828.
148/148 [==============================] - 1s 7ms/step
 MicroF1_Test:0.057
442/442 [==============================] - 16s 35ms/step - loss: 2.6105 - accuracy: 0.1397 - val_loss: 2.6358 - v
al_accuracy: 0.1381 - lr: 0.0090
Epoch 13/15
148/148 [==============================] - 1s 7ms/step
 MicroF1_Test:0.048
442/442 [==============================] - 16s 35ms/step - loss: 2.5979 - accuracy: 0.1401 - val_loss: 2.6654 - v
al_accuracy: 0.1281 - lr: 0.0081
Epoch 14/15
148/148 [==============================] - 1s 7ms/step
 MicroF1_Test:0.048
442/442 [==============================] - 16s 36ms/step - loss: 2.5927 - accuracy: 0.1445 - val_loss: 2.6295 - v
al_accuracy: 0.1394 - lr: 0.0081
Epoch 15/15
148/148 [==============================] - 1s 7ms/step
 MicroF1_Test:0.053
442/442 [==============================] - 16s 36ms/step - loss: 2.5794 - accuracy: 0.1505 - val_loss: 2.6232 - v
al_accuracy: 0.1419 - lr: 0.0081
```

In [27]:

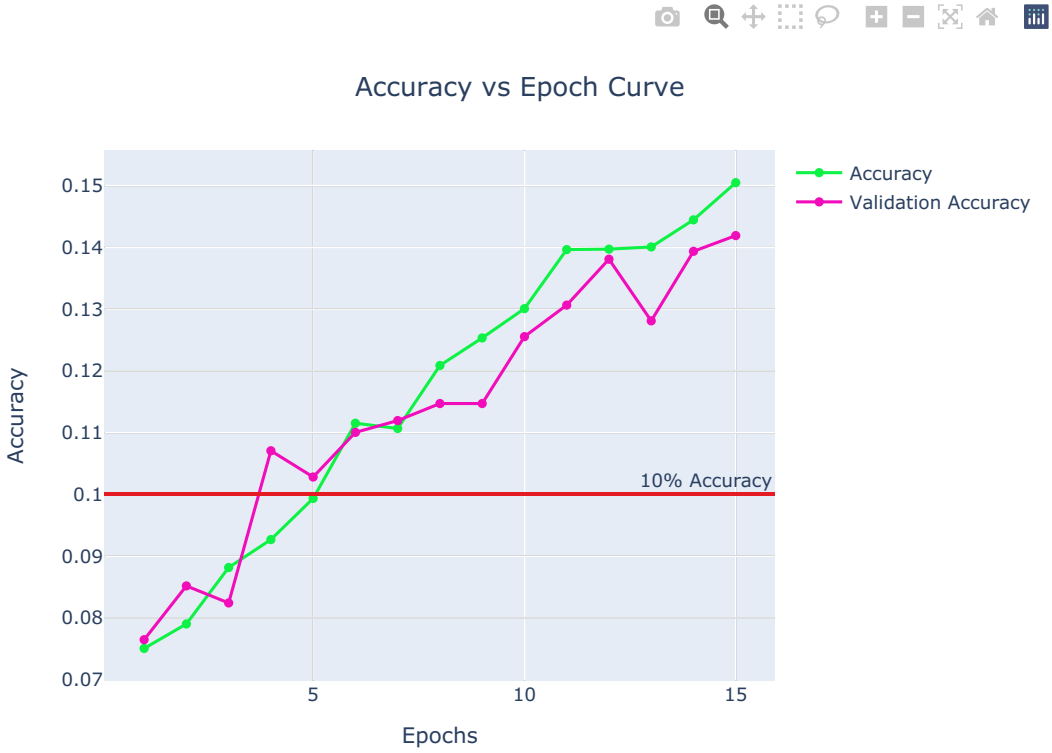
```
model_2.save_weights('best_model_2.h5')
```

In [28]:

```
# https://plotly.com/python/line-charts/
```

```
leng_ = len(history_2.history['accuracy']) + 1

fig = go.Figure()

fig.add_trace(go.Scatter(y = history_2.history['accuracy'], name = 'Accuracy',
                         line_color = '#0DF244', x = list(range(1, leng_))))
fig.add_trace(go.Scatter(y = history_2.history['val_accuracy'], name = 'Validation Accuracy',
                         line_color = '#F20DBB', x = list(range(1, leng_))))

fig.update_layout(title = {'text': 'Accuracy vs Epoch Curve', 'y':0.9, 'x':0.5,
                         'xanchor': 'center', 'yanchor': 'top'}, xaxis_title = 'Epochs',
                         yaxis_title = 'Accuracy')

fig.add_hline(y=0.10, annotation_text = '10% Accuracy',
                         annotation_position = 'top right', line_color = '#E41B22')

print(f"Accuracy of Model_2 : {round(history_2.history['accuracy'][-1], 2) * 100}%")

fig.show()
```

```
Accuracy of Model_2 : 15.0%
```



In [29]:

```
modeling_end = time()
print(f'Model_2 Time : {round((modeling_end - model_1)/60, 2)} Min')
print(f'Total Time : {round((modeling_end - start_)/60, 2)} Min')
```

```
Model_2 Time : 4.02 Min
Total Time : 36.91 Min
```

In [30]:

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

Cannot open file://:6006/: Path is a directory

Failed to load URL file://:6006/.

QtNetwork Error 202