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.

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

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

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

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

```
!pip install tensorflow-addons
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting tensorflow-addons
  Downloading tensorflow_addons-0.19.0-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.1 MB)
     |████████████████████████████████| 1.1 MB 34.5 MB/s
Requirement already satisfied: packaging in /usr/local/lib/python3.8/dist-packages (from tensorflow-addons) (21.3)
Requirement already satisfied: typeguard>=2.7 in /usr/local/lib/python3.8/dist-packages (from tensorflow-addons) (2.7.1)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.8/dist-packages (from packaging->tensorflow-addon
Installing collected packages: tensorflow-addons
Successfully installed tensorflow-addons-0.19.0
```

```
import os
from tqdm import tqdm
import pickle
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer,one_hot
from tensorflow.keras.preprocessing.sequence import pad_sequences
import tensorflow_addons as tfa
import tensorflow as tf
from tensorflow.keras.regularizers import l2
import tensorflow as tf
from tensorflow.keras.layers import Input,Dense,Conv1D,concatenate,Embedding,Flatten,Dropout,BatchNormalization,MaxPool1D
from tensorflow.keras.models import Model
from tensorflow.keras.utils import plot_model
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras.callbacks import EarlyStopping
import datetime
from keras.initializers import HeUniform
```

```
### 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?
```

Saving...

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.

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

```
len(preprocessed_email)
```

3. Get subject of the text i.e. get the total lines where "Subject:" occur and remove
the word which are before the ":" remove the newlines, tabs, punctuations, any special chars.
Eg: if we have sentance like "Subject: Re: Gospel Dating @ \r\r\n" --> You have to get "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.
amed entities into single word.
ases/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.

Saving...

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

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

We did chunking for above two lines and then We got one list where each word is mapped to a
POS(parts of speech) and also if you see "New York" and "Srikanth Varma",
they got combined and represented as a tree and "New York" was referred as "GPE" and "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.

```
!unrar x 'drive/MyDrive/Document-classification-CNN/documents.rar'
```

```
Extracting  documents/talk.religion.misc_84360.txt                    OK
Extracting  documents/talk.religion.misc_84380.txt                    OK
Extracting  documents/talk.religion.misc_84395.txt                    OK
Extracting  documents/talk.religion.misc_84396.txt                    OK
Extracting  documents/talk.religion.misc_84397.txt                    OK
Extracting  documents/talk.religion.misc_84398.txt                    OK
Extracting  documents/talk.religion.misc_84399.txt                    OK
Extracting  documents/talk.religion.misc_84401.txt                    OK
Extracting  documents/talk.religion.misc_84414.txt                    OK
Extracting  documents/talk.religion.misc_84422.txt                    OK
Extracting  documents/talk.religion.misc_84423.txt                    OK
Extracting  documents/talk.religion.misc_84428.txt                    OK
Extracting  documents/talk.religion.misc_84429.txt                    OK
Extracting  documents/talk.religion.misc_84430.txt                    OK
Extracting  documents/talk.religion.misc_84431.txt                    OK
Extracting  documents/talk.religion.misc_84433.txt                    OK
Extracting  documents/talk.religion.misc_84434.txt                    OK
Extracting  documents/talk.religion.misc_84435.txt                    OK
Extracting  documents/talk.religion.misc_84436.txt                    OK
Extracting  documents/talk.religion.misc_84437.txt                    OK
Extracting  documents/talk.religion.misc_84438.txt                    OK
Extracting  documents/talk.religion.misc_84439.txt                    OK
                                    gion.misc_84440.txt               OK
                                    gion.misc_84441.txt               OK
                                    gion.misc_84442.txt               OK
Extracting  documents/talk.religion.misc_84443.txt                    OK
Extracting  documents/talk.religion.misc_84444.txt                    OK
Extracting  documents/talk.religion.misc_84445.txt                    OK
Extracting  documents/talk.religion.misc_84446.txt                    OK
Extracting  documents/talk.religion.misc_84447.txt                    OK
Extracting  documents/talk.religion.misc_84448.txt                    OK
Extracting  documents/talk.religion.misc_84449.txt                    OK
Extracting  documents/talk.religion.misc_84450.txt                    OK
Extracting  documents/talk.religion.misc_84451.txt                    OK
Extracting  documents/talk.religion.misc_84452.txt                    OK
Extracting  documents/talk.religion.misc_84506.txt                    OK
Extracting  documents/talk.religion.misc_84507.txt                    OK
Extracting  documents/talk.religion.misc_84508.txt                    OK
Extracting  documents/talk.religion.misc_84509.txt                    OK
Extracting  documents/talk.religion.misc_84510.txt                    OK
Extracting  documents/talk.religion.misc_84511.txt                    OK
Extracting  documents/talk.religion.misc_84538.txt                    OK
Extracting  documents/talk.religion.misc_84552.txt                    OK
Extracting  documents/talk.religion.misc_84553.txt                    OK
Extracting  documents/talk.religion.misc_84554.txt                    OK
Extracting  documents/talk.religion.misc_84555.txt                    OK
Extracting  documents/talk.religion.misc_84557.txt                    OK
Extracting  documents/talk.religion.misc_84558.txt                    OK
Extracting  documents/talk.religion.misc_84559.txt                    OK
Extracting  documents/talk.religion.misc_84560.txt                    OK
Extracting  documents/talk.religion.misc_84562.txt                    OK
Extracting  documents/talk.religion.misc_84563.txt                    OK
Extracting  documents/talk.religion.misc_84564.txt                    OK
Extracting  documents/talk.religion.misc_84565.txt                    OK
Extracting  documents/talk.religion.misc_84568.txt                    OK
Extracting  documents/talk.religion.misc_84569.txt                    OK
Extracting  documents/talk.religion.misc_84570.txt                    OK
All OK
```

Saving...

```
#File names in documents.rar file
file_names =os.listdir("/content/documents")

#Sorting file names and get labels by splitting
file_names = sorted(file_names)
labels=[]
for i in file_names:
  labels.append(i.split('_')[0])
```

```
print(len(labels))
```

```
18828
```

```
labels[0:10]
```

```
['alt.atheism',
 'alt.atheism',
 'alt.atheism',
 'alt.atheism',
 'alt.atheism',
 'alt.atheism',
```

```
       'alt.atheism',
       'alt.atheism',
       'alt.atheism',
       'alt.atheism']
```

```
len(set(labels))
```

```
20
```

```
#all_content contains all the contents of the file
all_content=[]
for idx,name in enumerate(file_names):
  with open("/content/documents/"+name,'r',encoding='ISO-8859-1') as f:
    all_content.append(f.read())
```

```
print(len(all_content))
```

```
18828
```

```
data=pd.DataFrame()
data['text'] = all_content
data['class'] = labels
data.head()
```

Saving...

	text	class
0	From: mathew <mathew@mantis.co.uk>\nSubject: A...	alt.atheism
1	From: mathew <mathew@mantis.co.uk>\nSubject: A...	alt.atheism
2	From: I3150101@dbstu1.rz.tu-bs.de (Benedikt Ro...	alt.atheism
3	From: mathew <mathew@mantis.co.uk>\nSubject: R...	alt.atheism
4	From: strom@Watson.Ibm.Com (Rob Strom)\nSubjec...	alt.atheism

```
# https://stackoverflow.com/a/47091490/4084039
#same function which was worked on Donors choosen datasets.
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
```

```
text = all_content[4]
text
```

```
'From: strom@Watson.Ibm.Com (Rob Strom)\nSubject: Re: [soc.motss, et al.] "Princeton
axes matching funds for Boy Scouts"\n\nIn article <N4HY.93Apr5120934@harder.ccr-p.id
a.org>, n4hy@harder.ccr-p.ida.org (Bob McGwier) writes:\n\n|> [1] HOWEVER, I hate ec
onomic terrorism and political correctness\n|> worse than I hate this policy.  \n\n
\n|> [2] A more effective approach is to stop donating\n|> to ANY organizating that
directly or indirectly supports gay rights issues\n|> until they end the boycott on
```

```
#Preprocessing subject text
def subject(text):
  each_sub = re.findall("Subject:.*",text)
  substitute_sub = re.sub("Subject: Re?",'',each_sub[0])
  preprocess_sub = re.sub('[^A-Za-z0-9]+', ' ',substitute_sub)
  #remove extra space
  preprocess_sub= re.sub(' +', ' ',preprocess_sub)

  return preprocess_sub
```

```
subject(text)
```

```
' soc motss et al Princeton axes matching funds for Boy Scouts '
```

```
#preprocessing mail text
def mail_text(text):
  all_words_from_mails=[]
  #https://stackoverflow.com/questions/17681670/extract-email-sub-strings-from-large-document
  list_of_mails_in_text = re.findall(r'[\w\.-]+@[\w\.-]+\.\w+', text)
  #print("list_of_mails_in_text", list_of_mails_in_text)
  for mail in list_of_mails_in_text:
    each_mail_words_after_at = mail.split('@')[-1].split('.')
    all_words_from_mails.extend(each_mail_words_after_at)
  processed_mail = ' '.join([word for word in all_words_from_mails if len(word)>2])
  return processed_mail


mail_text(text)
```

```
'Watson Ibm Com harder ccr-p ida org harder ccr-p ida org watson ibm com'
```

```
#reference: https://www.guru99.com/pos-tagging-chunking-nltk.html, https://stackoverflow.com/questions/48660547/how-can-i-extract-gpeloca

#In this mainly work on pos tags if the word belongs to parts of speech tag Person than we can replce with space otherwise append those w
def chunking(text):
  person = []
  gpe = []
  word = nltk.word_tokenize(text)
  pos_tag = nltk.pos_tag(word)
  chunk = nltk.ne_chunk(pos_tag)
```

Saving...

```
            if ele.label()=='PERSON':
              for w,p in ele:
                text=text.replace(w,"")
            else:
               strg=[]
               for word,pos in ele:
                 strg.append(word)
               strng="_".join(strg)
               for i in range(len(ele)):
                 if i==len(ele)-1:
                   text=text.replace(ele[i][0],strng)
                 else:
                   text=text.replace(ele[i][0],"")


  return text


chunking(text)
```

```
'From: strom@Watson.Ibm.Com ( )\nSubject: Re: [soc.motss, et al.] "Princeton axes ma
tching funds for  "\n\nIn article <N4HY.93Apr5120934@harder.ccr-p.ida.org>, n4hy@har
der.ccr-p.ida.org ( ) writes:\n\n|> [1] HOWEVER, I hate economic terrorism and polit
ical correctness\n|> worse than I hate this policy.  \n\n\n|> [2] A more effective a
pproach is to stop donating\n|> to ANY organizating that directly or indirectly supp
orts gay rights issues\n|> until they end the boycott on funding of scouts.  \n\nCan
```

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

```
def preprocessed_text(text):

  text=re.sub('[\w\.-]+@[\w\.-]+\.\w+', ' ',text)
  text=re.sub("Subject:.*\w+",'',text)
  #3. Delete all the sentances where sentence starts with "Write to:" or "From:".
  text=re.sub("From:.*?", ' ',text)
  text=re.sub("Write to:.*?",' ',text)

  # 4. Delete all the tags like "< anyword >"
  clean = re.compile('<.*?>')
  text=re.sub(clean,' ',text)

  # 5. Delete all the data which are present in the brackets.
  clean1 = re.compile('\(.*\)')
  text=re.sub(clean1,'',text)

  #6. Remove all the newlines('\n'), tabs('\t'), "-", "\".
  text= re.sub(r"[\n\t-]*", "", text)

  #Remove all the words which ends with ":".
  text= re.sub(r'\w+:\s?',' ',text)
  text= re.sub('[^A-Za-z0-9]+', ' ',text)
```

```
    #Decontraction of text
    text = decontracted(text)

    text = chunking(text)

    #Replace all the digits with space i.e delete all the digits.
    text= re.sub("[0-9]+","",text)
    text= re.sub(r"\b_([a-zA-z]+)_\b",r"\1",text)

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


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

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


preprocessed_text(text)
```

```
    'article however hate economic terrorism and political correctness worse than hate t
    his policy more effective approach stop donating any organizating that directly indi
    es until they end the boycott funding scouts can some
    tradiction between and strom ibm research saw box vor
```


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.

```
preprocessed_text_data=[]
preprocessed_subject=[]
preprocessed_emails=[]

for i in tqdm(range(data.shape[0])):
  preprocessed_emails.append(mail_text(data['text'].values[i]))
  preprocessed_subject.append(subject(data['text'].values[i]))
  preprocessed_text_data.append(preprocessed_text(data['text'].values[i]))
```

```
    100%|██████████| 18828/18828 [18:12<00:00, 17.23it/s]
```

```
preprocessed_text_data[1]
```

```
    'mathew atheism pril begin pgp signed messge introduction theism mathew this article
    attempts provide general introduction atheism tried neutral possible regarding conte
    ntious issues youshould always remember that this document represents only one viewp
    oint encourage you read widely and draw your own conclusions somerelevant books are
    listed companion article provide sense cohesion and progression have presented this
    articleas imaginary conversation between atheist and theist thequestions asked the i
    maginary theist are questions which have been croppedup repeatedly alt atheism since
    the newsgroup was created some asked questions are answered companion article note t
```

```
data['preprocessed_subject'] = preprocessed_subject
data['preprocessed_emails'] = preprocessed_emails
data['preprocessed_text_data'] = preprocessed_text_data
data.head()
```

	text	class	preprocessed_subject	preprocessed_emails	preprocessed_text_data
0	From: mathew	alt.atheism	Subject Alt Atheism FAQ	mantis netcom com mantis	mathew atheism from religion fish

```
data.columns
```

```
Index(['text', 'class', 'preprocessed_subject', 'preprocessed_emails',
       'preprocessed_text_data'],
      dtype='object')
```

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

```
text                    From: perry@dsinc.com (Jim Perry)\nSubject: Re...
class                                                        alt.atheism
preprocessed_subject           Is Morality Constant was Re Biblical Rape
preprocessed_emails     dsinc com darkside osrhe uoknor edu okcforum o...
preprocessed_text_data  this response originally fell into bit bucket ...
Name: 400, dtype: object
```

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.

Saving...

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.

```
data['text'] = data['preprocessed_emails']+data['preprocessed_subject']+data['preprocessed_text_data']
data=data['text']
data=pd.DataFrame(data)
data['class']=labels
data.head()
```

	text	class
0	mantis netcom com mantisSubject Alt Atheism FA...	alt.atheism

```
X_data=data.drop('class',axis=1)
y_data=data['class']

# train test split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X_data,y_data,test_size=0.25,stratify=y_data)


y_train.head()
```

```
9772          rec.sport.hockey
17728       talk.politics.misc
4385     comp.sys.mac.hardware
17999       talk.politics.misc
1340             comp.graphics
Name: class, dtype: object
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```



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

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


print('max length of text : ',max(length_of_text))
```

```
max length of text :  1
```

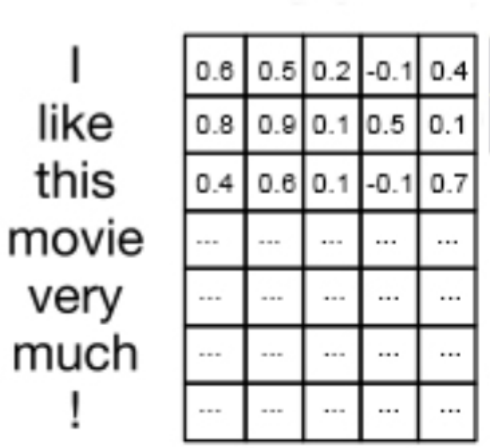
```
import pickle


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



print(y_train.shape)
print(y_test.shape)
```

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

```
with open('drive/MyDrive/Document-classification-CNN/glove_vectors','rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

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

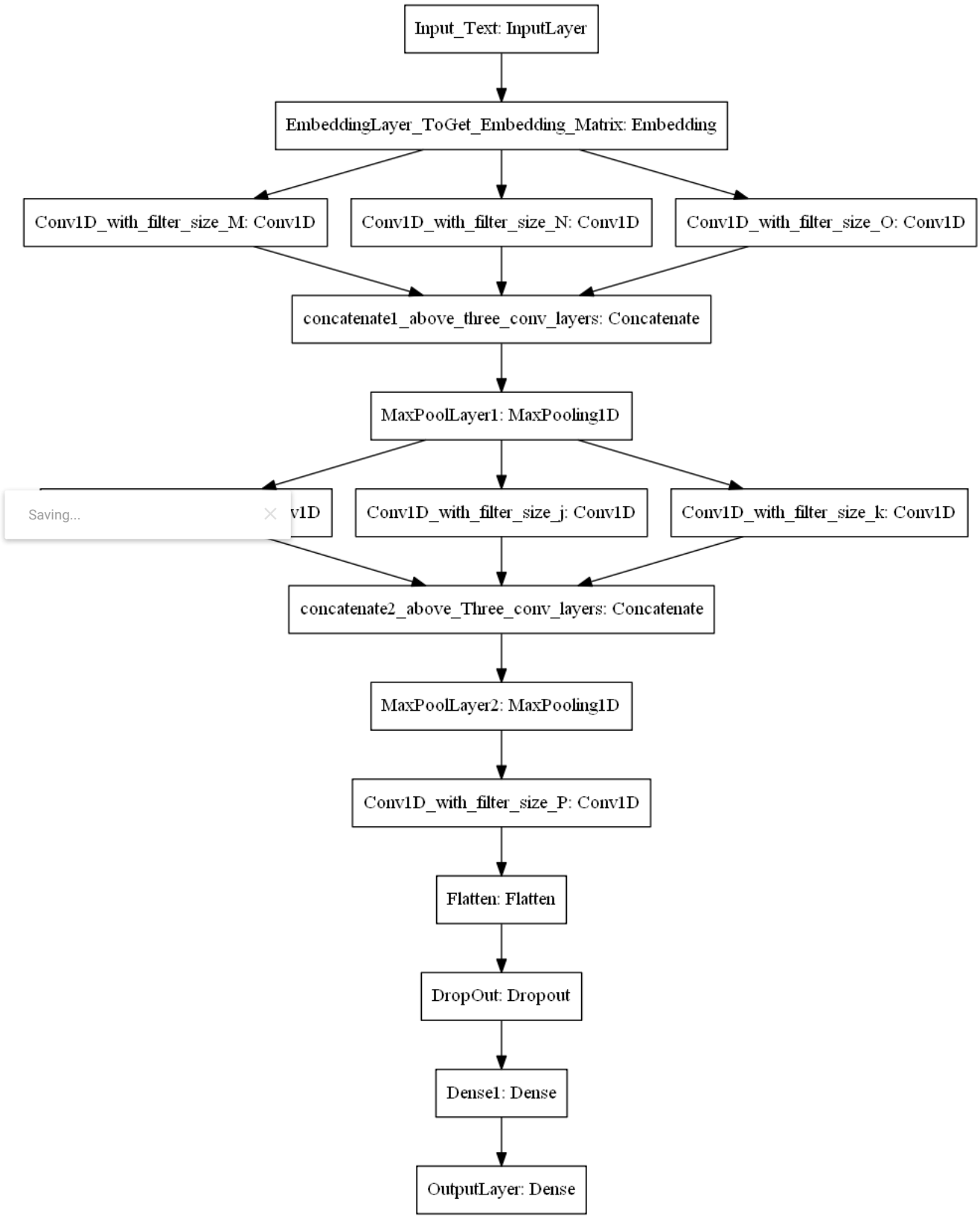
Reference:

https://stackoverflow.com/a/43399308/4084039

https://missinglink.ai/guides/keras/keras-conv1d-working-1d-convolutional-neural-networks-keras/

Saving...

- 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.

```
tokenizer= Tokenizer(filters='!"#$%&()*+,-./:;<=>?@[\\]^`{|}~\t\n')
tokenized_text = tokenizer.fit_on_texts(list(X_train['text']))
vocab_size=len(tokenizer.word_index)+1
print("uniques words:{}".format(vocab_size))
```

```
uniques words:144074
```

```
train_sequences=tokenizer.texts_to_sequences(list(X_train['text']))
print(len(train_sequences[0]))
#took padding length as 250 because average length of sequences were 250
train_padded_sequences=pad_sequences(train_sequences,maxlen=250)
print(len(train_padded_sequences[0]))
```

```
58
250
```

```
test_sequences=tokenizer.texts_to_sequences(list(X_test['text']))
print(len(test_sequences[0]))
test_padded_sequences=pad_sequences(test_sequences,maxlen=250)
print(len(test_padded_sequences[0]))
```

```
161
250
```



```
test_padded_sequences.shape
```

```
(4707, 250)
```

```
embedding_matrix = np.zeros((vocab_size, 300))
for word, i in tokenizer.word_index.items():
    embedding_vector = model.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

```
!rm -rf ./model1_logs/
```

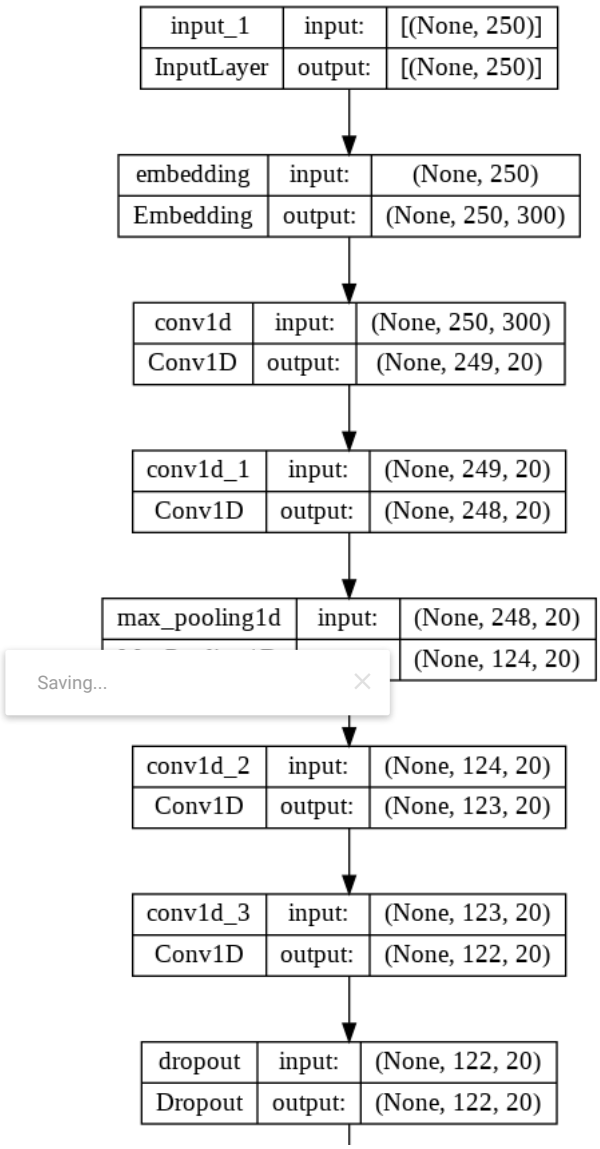
```
tf.keras.backend.clear_session()

initializer = tf.keras.initializers.he_normal
micro_f1_score = tfa.metrics.F1Score(num_classes=20,average='micro')

input=Input(shape=(250,))
embedding_layer=Embedding(vocab_size_1,300,weights=[embedding_matrix_1],trainable=False,input_length=250)(input)

conv_1=Conv1D(filters=20,kernel_size=2,activation='relu',kernel_initializer=initializer)(embedding_layer)
conv_2=Conv1D(filters=20,kernel_size=2,activation='relu',kernel_initializer=initializer)(conv_1)


maxpool_1=MaxPool1D(pool_size=2,strides=2)(conv_2)

conv_4=Conv1D(filters=20,kernel_size=2,activation='relu',kernel_initializer=initializer)(maxpool_1)
conv_5=Conv1D(filters=20,kernel_size=2,activation='relu',kernel_initializer=initializer)(conv_4)


dropout_2=Dropout(rate=0.2)(conv_5)
bn_2=BatchNormalization()(dropout_2)

maxpool_2=MaxPool1D(pool_size=2,strides=2)(bn_2)


flatten=Flatten()(maxpool_2)
dropout=Dropout(rate=0.2)(flatten)

fc=Dense(30,activation='relu',kernel_initializer=initializer)(dropout)
op=Dense(20,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_normal)(fc)

model_1=Model(input,op)
optimizer=tf.keras.optimizers.Adam()
model_1.compile(optimizer=optimizer,loss='categorical_crossentropy',metrics=['accuracy',micro_f1_score])


Early_stop=EarlyStopping(monitor='val_accuracy',min_delta=0.0001,patience=3,verbose=1)

log_dir="model1_logs/fit/"+ datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
```

```
tensorboard=TensorBoard(log_dir=log_dir,histogram_freq=1)

filepath="best_model_1.h5"
checkpoint=ModelCheckpoint(filepath,monitor='val_accuracy',verbose=1,mode='auto',save_best_only=True)

model_1.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 250)]             0

 embedding (Embedding)       (None, 250, 300)          12000

 conv1d (Conv1D)             (None, 249, 20)           12020

 conv1d_1 (Conv1D)           (None, 248, 20)           820

 max_pooling1d (MaxPooling1D  (None, 124, 20)          0
 )

 conv1d_2 (Conv1D)           (None, 123, 20)           820

 conv1d_3 (Conv1D)           (None, 122, 20)           820

 dropout (Dropout)           (None, 122, 20)           0

                             None, 122, 20)            80

 max_pooling1d_1 (MaxPooling  (None, 61, 20)           0
 1D)

 flatten (Flatten)           (None, 1220)              0

 dropout_1 (Dropout)         (None, 1220)              0

 dense (Dense)               (None, 30)                36630

 dense_1 (Dense)             (None, 20)                620

=================================================================
Total params: 63,810
Trainable params: 51,770
Non-trainable params: 12,040
_________________________________________________________________
```



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



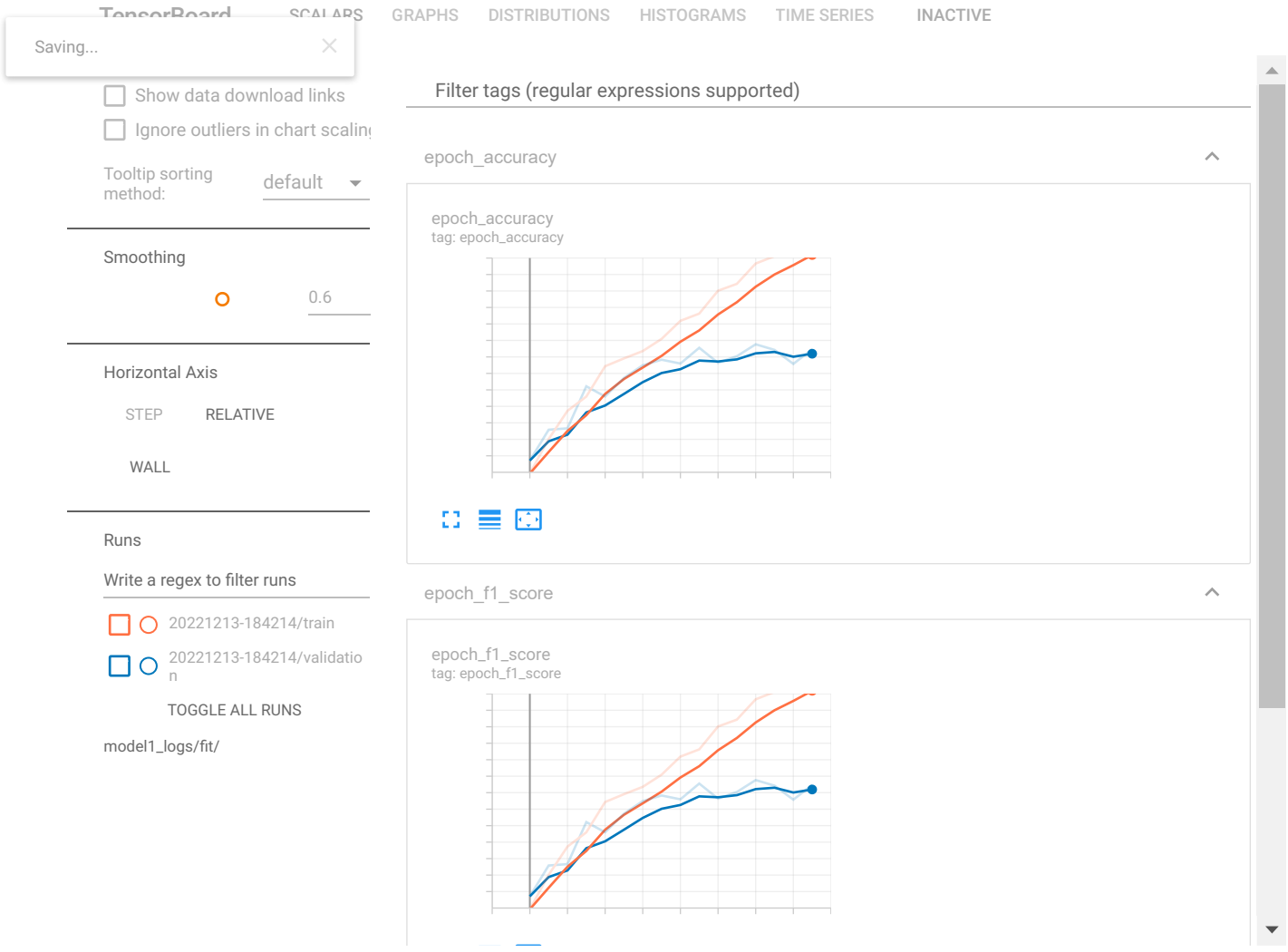

```
model_1.fit(train_padded_sequences,y_train,epochs=50,validation_data=(test_padded_sequences,y_test),callbacks=[Early_stop,tensorboard,che
```

```
Epoch 1/50
439/442 [============================>.] - ETA: 0s - loss: 2.9981 - accuracy: 0.0692 - f1_score: 0.0692
Epoch 1: val_accuracy improved from -inf to 0.07712, saving model to best_model_1.h5
442/442 [==============================] - 4s 8ms/step - loss: 2.9978 - accuracy: 0.0693 - f1_score: 0.0693 - val_loss: 2.9317 -
Epoch 2/50
436/442 [============================>.] - ETA: 0s - loss: 2.8960 - accuracy: 0.0897 - f1_score: 0.0897
Epoch 2: val_accuracy improved from 0.07712 to 0.09581, saving model to best_model_1.h5
442/442 [==============================] - 3s 7ms/step - loss: 2.8955 - accuracy: 0.0901 - f1_score: 0.0901 - val_loss: 2.8711 -
Epoch 3/50
436/442 [============================>.] - ETA: 0s - loss: 2.8084 - accuracy: 0.1074 - f1_score: 0.1074
Epoch 3: val_accuracy improved from 0.09581 to 0.09666, saving model to best_model_1.h5
442/442 [==============================] - 3s 6ms/step - loss: 2.8090 - accuracy: 0.1073 - f1_score: 0.1073 - val_loss: 2.9623 -
Epoch 4/50
433/442 [============================>.] - ETA: 0s - loss: 2.7637 - accuracy: 0.1150 - f1_score: 0.1150
Epoch 4: val_accuracy improved from 0.09666 to 0.12216, saving model to best_model_1.h5
442/442 [==============================] - 3s 7ms/step - loss: 2.7638 - accuracy: 0.1159 - f1_score: 0.1159 - val_loss: 2.7628 -
Epoch 5/50
437/442 [============================>.] - ETA: 0s - loss: 2.7385 - accuracy: 0.1341 - f1_score: 0.1341
Epoch 5: val_accuracy did not improve from 0.12216
442/442 [==============================] - 3s 7ms/step - loss: 2.7386 - accuracy: 0.1343 - f1_score: 0.1343 - val_loss: 2.8245 -
Epoch 6/50
441/442 [============================>.] - ETA: 0s - loss: 2.7134 - accuracy: 0.1390 - f1_score: 0.1390
Epoch 6: val_accuracy improved from 0.12216 to 0.12726, saving model to best_model_1.h5
442/442 [==============================] - 3s 7ms/step - loss: 2.7135 - accuracy: 0.1391 - f1_score: 0.1391 - val_loss: 2.7581 -
Epoch 7/50
439/442 [============================>.] - ETA: 0s - loss: 2.6928 - accuracy: 0.1435 - f1_score: 0.1435
Epoch 7: val_accuracy improved from 0.12726 to 0.13491, saving model to best_model_1.h5
442/442 [==============================] - 3s 6ms/step - loss: 2.6929 - accuracy: 0.1435 - f1_score: 0.1435 - val_loss: 2.7392 -
Epoch 8/50
440/442 [============================>.] - ETA: 0s - loss: 2.6641 - accuracy: 0.1512 - f1_score: 0.1512
Epoch 8: val_accuracy improved from 0.13491 to 0.13830, saving model to best_model_1.h5
442/442 [==============================] - 3s 6ms/step - loss: 2.6646 - accuracy: 0.1509 - f1_score: 0.1509 - val_loss: 2.7416 -
Epoch 9/50
440/442 [============================>.] - ETA: 0s - loss: 2.6425 - accuracy: 0.1620 - f1_score: 0.1620
Epoch 9: val_accuracy did not improve from 0.13830
442/442 [==============================] - 3s 7ms/step - loss: 2.6426 - accuracy: 0.1619 - f1_score: 0.1619 - val_loss: 2.7559 -
```

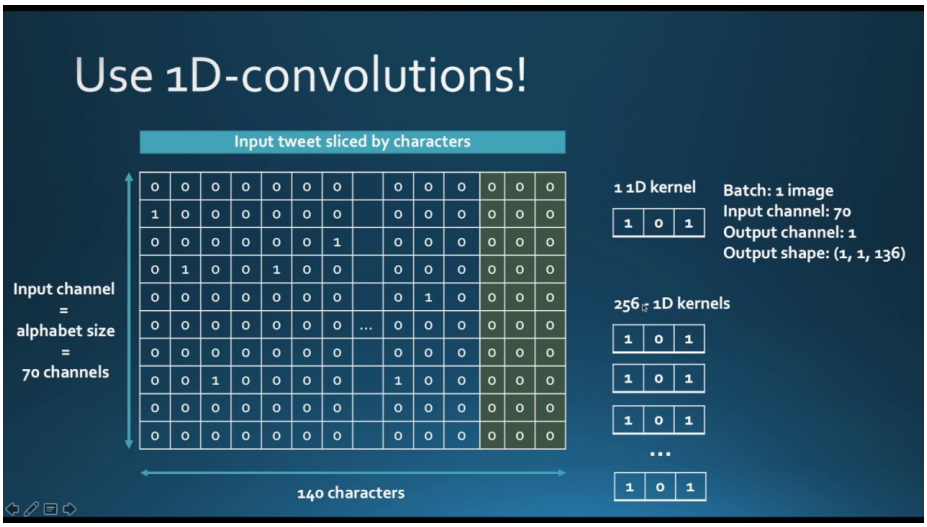
```
Epoch 10/50
433/442 [============================>.] - ETA: 0s - loss: 2.6170 - accuracy: 0.1659 - f1_score: 0.1659
Epoch 10: val_accuracy improved from 0.13830 to 0.14553, saving model to best_model_1.h5
442/442 [==============================] - 3s 7ms/step - loss: 2.6162 - accuracy: 0.1663 - f1_score: 0.1663 - val_loss: 2.7464 -
Epoch 11/50
435/442 [============================>.] - ETA: 0s - loss: 2.5923 - accuracy: 0.1797 - f1_score: 0.1797
Epoch 11: val_accuracy did not improve from 0.14553
442/442 [==============================] - 3s 7ms/step - loss: 2.5907 - accuracy: 0.1802 - f1_score: 0.1802 - val_loss: 2.7875 -
Epoch 12/50
440/442 [============================>.] - ETA: 0s - loss: 2.5653 - accuracy: 0.1844 - f1_score: 0.1844
Epoch 12: val_accuracy did not improve from 0.14553
442/442 [==============================] - 3s 7ms/step - loss: 2.5656 - accuracy: 0.1843 - f1_score: 0.1843 - val_loss: 2.7723 -
Epoch 13/50
436/442 [============================>.] - ETA: 0s - loss: 2.5379 - accuracy: 0.1966 - f1_score: 0.1966
Epoch 13: val_accuracy improved from 0.14553 to 0.14765, saving model to best_model_1.h5
442/442 [==============================] - 3s 7ms/step - loss: 2.5378 - accuracy: 0.1967 - f1_score: 0.1967 - val_loss: 2.7467 -
Epoch 14/50
435/442 [============================>.] - ETA: 0s - loss: 2.5225 - accuracy: 0.2016 - f1_score: 0.2016
Epoch 14: val_accuracy did not improve from 0.14765
```

```
%load_ext tensorboard
%tensorboard --logdir model1_logs/fit/
```

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



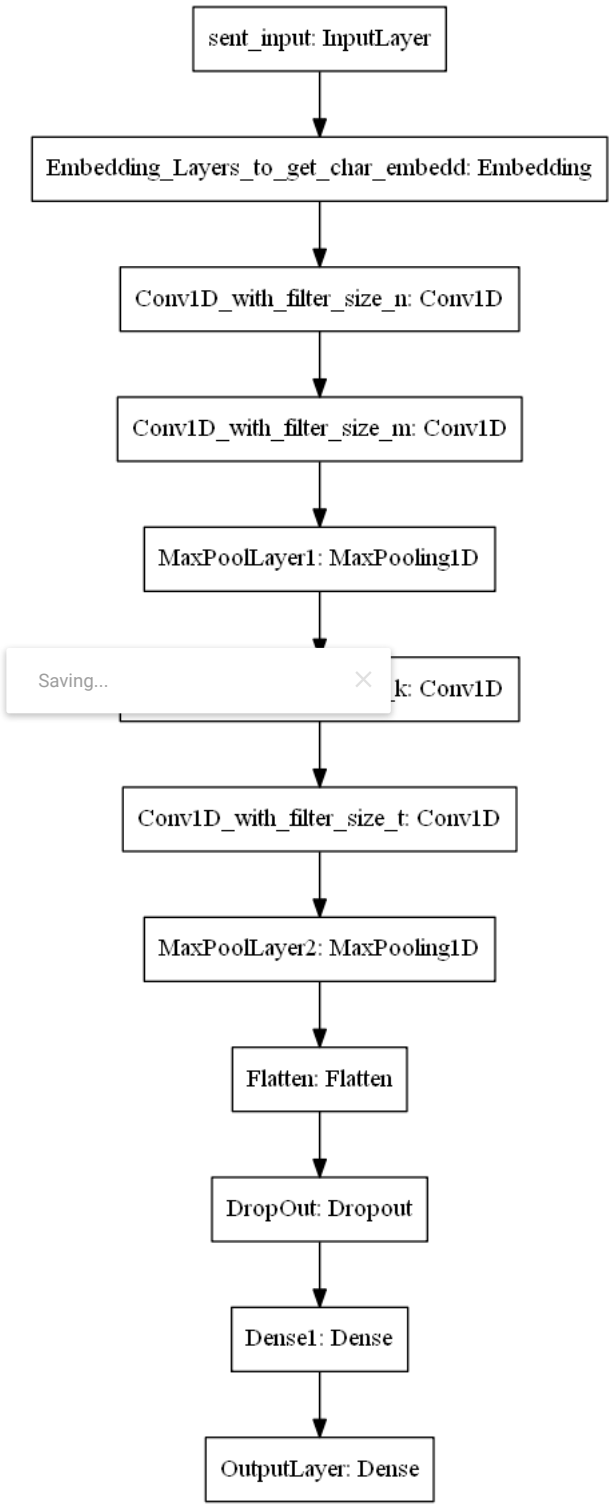
Model-2 : Using 1D convolutions with character embedding



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. ...d Sontag, Alexander M. Rush. Character-Aware Neural Language Models. AAAI 2016
3. .../ladlen Koltun. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequ
4. Use the pratrained char embeddings https://github.com/minimaxir/char-embeddings/blob/master/glove.840B.300d-char.txt

Saving...




```
tokenizer_1=Tokenizer(char_level=True,filters='!"#$%&()*+,-./:;<=>?@[\\]^`{|}~\t\n')#character level tokenization
tokenized_text_1=tokenizer_1.fit_on_texts(list(X_train['text']))
vocab_size_1=len(tokenizer_1.word_index)+1
print("uniques words:{}".format(vocab_size_1))
```

```
uniques words:40
```

```
train_sequences_1=tokenizer_1.texts_to_sequences(list(X_train['text']))
print(len(train_sequences_1[0]))
train_padded_sequences_1=pad_sequences(train_sequences_1,maxlen=250)
print(len(train_padded_sequences_1[0]))
```

```
360
250
```

```
test_sequences_1=tokenizer_1.texts_to_sequences(list(X_test['text']))
print(len(test_sequences_1[0]))
test_padded_sequences_1=pad_sequences(test_sequences_1,maxlen=250)
print(len(test_padded_sequences_1[0]))
```

```
1152
250
```

```
path = "drive/MyDrive/Document-classification-CNN/glove.840B.300d.txt"
```

```
#getting character embeddings
with open(path,'r') as f:
  embedd_dict={}
  file=f.readlines()
  for i in range(len(file)):
     embedd_dict[file[i][0]]=file[i][1:].split(" ")[1:]
```

```
embedding_matrix_1= np.zeros((vocab_size_1, 300))
for word, i in tokenizer_1.word_index.items():
    embedding_vector = embedd_dict.get(word)
    if embedding_vector is not None:
        embedding_matrix_1[i] = embedding_vector
```

```
!rm -rf ./model2_logs/
```

Saving...

```
early_stop=EarlyStopping(monitor='val_accuracy',min_delta=0.0001,patience=5,verbose=1)

log_dir="model2_logs/fit/"+ datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard=TensorBoard(log_dir=log_dir,histogram_freq=1)

filepath="best_model_2.h5"
checkpoint=ModelCheckpoint(filepath,monitor='val_accuracy',verbose=1,mode='auto',save_best_only=True)
```

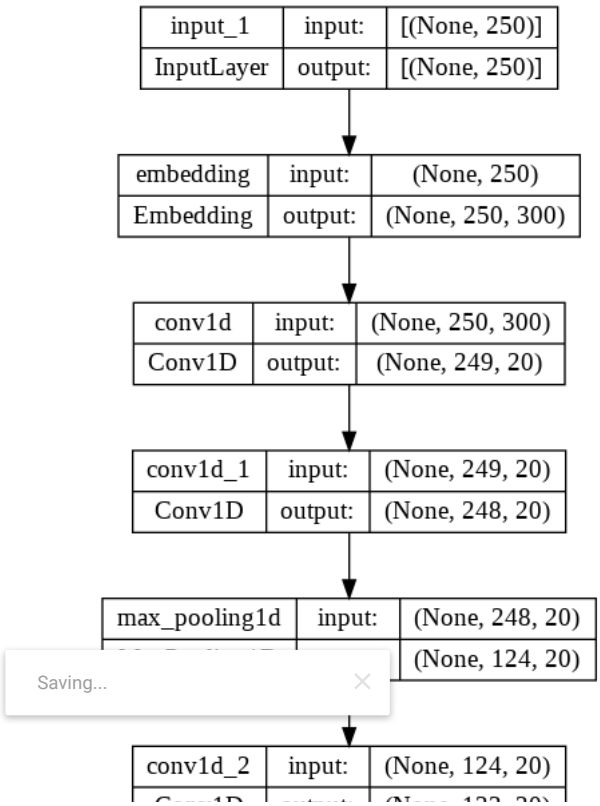
```
tf.keras.backend.clear_session()

initializer = tf.keras.initializers.he_normal
micro_f1_score = tfa.metrics.F1Score(num_classes=20,average='micro')

input=Input(shape=(250,))
embedding_layer=Embedding(vocab_size_1,300,weights=[embedding_matrix_1],trainable=False,input_length=250)(input)

conv_1=Conv1D(filters=20,kernel_size=2,activation='relu',kernel_initializer=initializer)(embedding_layer)
conv_2=Conv1D(filters=20,kernel_size=2,activation='relu',kernel_initializer=initializer)(conv_1)


maxpool_1=MaxPool1D(pool_size=2,strides=2)(conv_2)

conv_4=Conv1D(filters=20,kernel_size=2,activation='relu',kernel_initializer=initializer)(maxpool_1)
conv_5=Conv1D(filters=20,kernel_size=2,activation='relu',kernel_initializer=initializer)(conv_4)


dropout_2=Dropout(rate=0.2)(conv_5)
bn_2=BatchNormalization()(dropout_2)

maxpool_2=MaxPool1D(pool_size=2,strides=2)(bn_2)


flatten=Flatten()(maxpool_2)
dropout=Dropout(rate=0.2)(flatten)

fc=Dense(30,activation='relu',kernel_initializer=initializer)(dropout)
op=Dense(20,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_normal)(fc)

model_1=Model(input,op)
optimizer=tf.keras.optimizers.Adam()
model_1.compile(optimizer=optimizer,loss='categorical_crossentropy',metrics=['accuracy',micro_f1_score])
```

```
model_1.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 250)]             0

 embedding (Embedding)       (None, 250, 300)          12000
```

```
conv1d (Conv1D)              (None, 249, 20)           12020

conv1d_1 (Conv1D)            (None, 248, 20)           820

max_pooling1d (MaxPooling1D  (None, 124, 20)           0
)

conv1d_2 (Conv1D)            (None, 123, 20)           820

conv1d_3 (Conv1D)            (None, 122, 20)           820

dropout (Dropout)            (None, 122, 20)           0

batch_normalization (BatchN  (None, 122, 20)           80
ormalization)

max_pooling1d_1 (MaxPooling  (None, 61, 20)            0
1D)

flatten (Flatten)            (None, 1220)              0

dropout_1 (Dropout)          (None, 1220)              0

dense (Dense)                (None, 30)                36630

dense_1 (Dense)              (None, 20)                620

=================================================================
Saving...
Non-trainable params: 12,040
_________________________________________________________________
```

```
plot_model(model_1,'image_1.png',show_shapes=True)
```



Saving...

```
model_1.fit(train_padded_sequences_1,y_train,epochs=100,validation_data=(test_padded_sequences_1,y_test),callbacks=[Early_stop,tensorboar
```

```
434/442 [============================>.] - ETA: 0s - loss: 2.5767 - accuracy: 0.1567 - f1_score: 0.1567
Epoch 12: val_accuracy did not improve from 0.13087
442/442 [==============================] - 3s 6ms/step - loss: 2.5757 - accuracy: 0.1566 - f1_score: 0.1566 - val_loss: 2.8488 -
```

✓ 0s completed at 12:17 AM