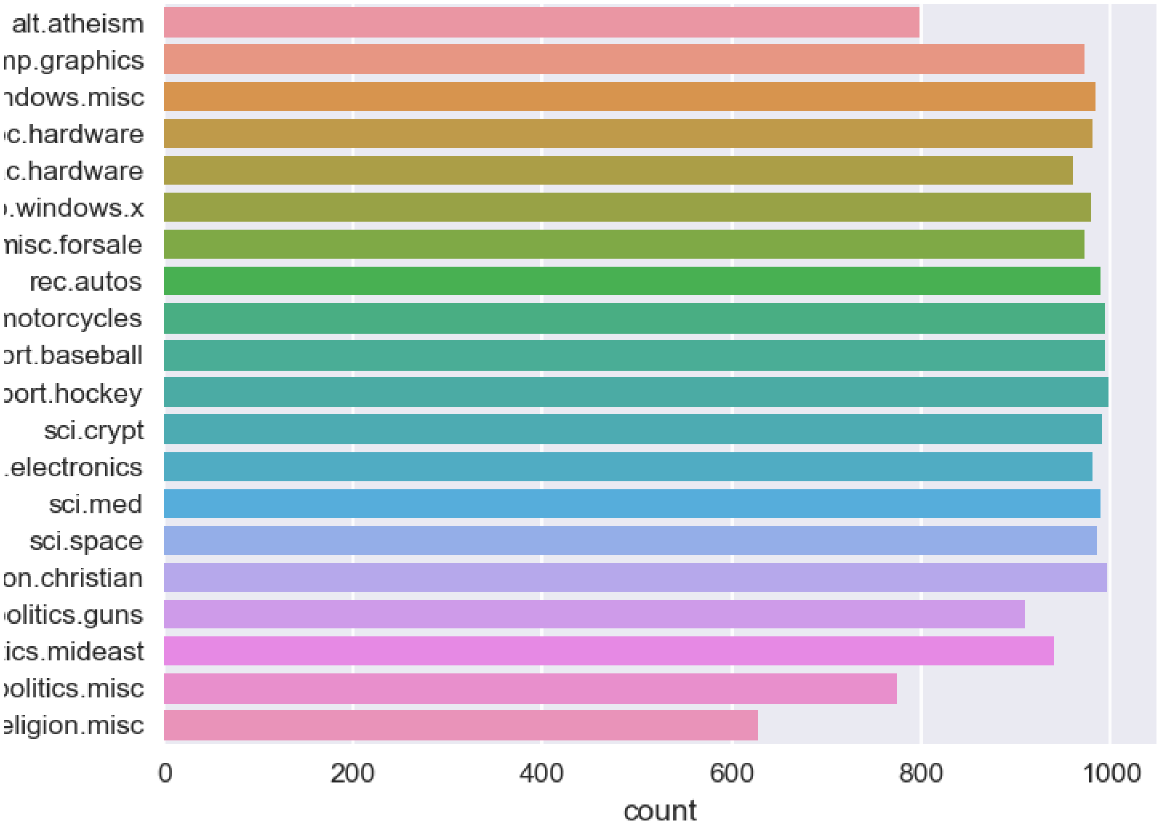
Text Classification:

Data

1. we have total of 20 types of documents(Text files) and total 18828 documents(text files).
2. You can download data from this link, in that you will get documents.rar folder.
If you unzip that, you will get total of 18828 documnets. document name is defined as'ClassLabel_DocumentNumberInThatLabel'.
so from document name, you can extract the label for that document.
4. Now our problem is to classify all the documents into any one of the class.
5. Below we provided count plot of all the labels in our data.

In [ ]:

```
### count plot of all the class labels.
```



Assignment:

sample document

```
Subject: A word of advice
From: jcopelan@nyx.cs.du.edu (The One and Only)

In article < 65882@mimsy.umd.edu > mangoe@cs.umd.edu (Charley Wingate) writes:
>
>I've said 100 times that there is no "alternative" that should think you
>might have caught on by now.  And there is no "alternative", but the point
```

```
>is, "rationality" isn't an alternative either.  The problems of metaphysical
>and religious knowledge are unsolvable-- or I should say, humans cannot
>solve them.

How does that saying go: Those who say it can't be done shouldn't interrupt
those who are doing it.

Jim
--
Have you washed your brain today?
```

Preprocessing:

useful links: http://www.pyregex.com/

1. Find all emails in the document and then get the text after the "@". and then split those texts by '.'
after that remove the words whose length is less than or equal to 2 and also remove'com' word and then combine those words by space.
In one doc, if we have 2 or more mails, get all.
Eg:[test@dm1.d.com, test2@dm2.dm3.com]-->[dm1.d.com, dm3.dm4.com]-->[dm1,d,com,dm2,dm3,com]-->[dm1,dm2,dm3]-->"dm1 dm2 dm3"
append all those into one list/array. (This will give length of 18828 sentences i.e one list for each of the document).
Some sample output was shown below.

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

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

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

In [ ]:

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

In [ ]:

```
len(preprocessed_email)
```

Out[ ]:

```
18828
```

In [ ]:

```
import os
import pandas as pd
import numpy as np
import re
import nltk
from tqdm import tqdm
```

In [ ]:

```
datalist = []
for filename in os.listdir('documents'):
    filetxt = open('documents/'+filename,'r')
    datalist.append( [ filetxt.read() , filename.split('_')[0] ] )
```

In [ ]:

```
df = pd.DataFrame(datalist)

df.columns = ['filetxt','classlabel']

print(df.head(10))
```

```
                                             filetxt   classlabel
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
5  From: I3150101@dbstu1.rz.tu-bs.de (Benedikt Ro...  alt.atheism
6  From: keith@cco.caltech.edu (Keith Allan Schne...  alt.atheism
7  From: I3150101@dbstu1.rz.tu-bs.de (Benedikt Ro...  alt.atheism
8  From: keith@cco.caltech.edu (Keith Allan Schne...  alt.atheism
9  From: keith@cco.caltech.edu (Keith Allan Schne...  alt.atheism
```

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

4. After you store it in the list, Replace those sentances in original text by space.

5. Delete all the sentances where sentence starts with **"Write to:"** or **"From:"**.
> In the above sample document check the 2nd line, we should remove that

6. Delete all the tags like "< anyword >"
> In the above sample document check the 4nd line, we should remove that "< 65882@mimsy.umd.edu >"

7. Delete all the data which are present in the brackets.
In many text data, we observed that, they maintained the explanation of sentence
or translation of sentence to another language in brackets so remove all those.
Eg: "AAIC-The course that gets you HIRED(AAIC - Der Kurs, der Sie anstellt)" --> "AAIC-The course that gets you HIRED"

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

8. Remove all the newlines('\n'), tabs('\t'), "-", "\".

9. Remove all the words which ends with ":".
Eg: "Anyword:"
> In the above sample document check the 4nd line, we should remove that "writes:"

10. Decontractions, replace words like below to full words.
please check the donors choose preprocessing for this
Eg: can't -> can not, 's -> is, i've -> i have, i'm -> i am, you're -> you are, i'll --> i will

There is no order to do point 6 to 10. but you have to get final output correctly

11. Do chunking on the text you have after above preprocessing.
Text chunking, also referred to as shallow parsing, is a task that
follows Part-Of-Speech Tagging and that adds more structure to the sentence.
So it combines the some phrases, named entities into single word.
So after that combine all those phrases/named entities by separating **"_"**.
And remove the phrases/named entities if that is a "Person".
You can use **nltk.ne_chunk** to get these.
Below we have given one example. please go through it.

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

In [ ]:

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

```
i am living in the New York --> [('i', 'NN'), ('am', 'VBP'), ('living', 'VBG'), ('in', 'IN'), ('the', 'DT'), Tree('GPE',
[('New', 'NNP'), ('York', 'NNP')])]

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

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

We did chunking for above two lines and then We got one list where each word is mapped to a
POS(parts of speech) and also if you see "New York" and "Srikanth Varma",
they got combined and represented as a tree and "New York" was referred as "GPE" and "Srikanth Varma"
was referred as "PERSON".
so now you have to Combine the "New York" with "_" i.e "New_York"
and remove the "Srikanth Varma" from the above sentence because it is a person.

13. Replace all the digits with space i.e delete all the digits.
> In the above sample document, the 6th line have digit 100, so we have to remove that.

14. After doing above points, we observed there might be few word's like
"_word_" (i.e starting and ending with the _), "_word" (i.e starting with the _),
"word_" (i.e ending with the _) remove the _ from these type of words.

15. We also observed some words like **"OneLetter_word"- eg: d_berlin,**
"TwoLetters_word" - eg: dr_berlin , in these words we remove the "OneLetter_" (d_berlin ==> berlin) and
"TwoLetters_" (de_berlin ==> berlin). i.e remove the words
which are length less than or equal to 2 after spliiting those words by "_".

16. Convert all the words into lower case and lowe case
and remove the words which are greater than or equal to 15 or less than or equal to 2.

17. replace all the words except "A-Za-z_" with space.

18. Now You got Preprocessed Text, email, subject. create a dataframe with those. Below are the columns of the df.

In [ ]:

```
data.columns
```

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

In [ ]:

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

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

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

In [ ]:

```
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"""

    #https://www.tutorialspoint.com/python_text_processing/python_extract_emails_from_text.htm
    #https://www.youtube.com/watch?v=K8L6KVGG-7o&ab_channel=CoreySchafer

    #finding all EMAILS
    emaillist = re.findall('[a-z0-9\.\-+_]+@[a-z0-9\.\-+_]+\.[a-z]+' , Input_Text)

    #getting all text after @
    dotlist = [email.split('@')[1] for email in emaillist]

    #list of each word
    distinctext = [ word.split('.') for word in dotlist ]

    #string of distinct extensions where length is greater than 2
    preprocessedEmail = ' '.join(set([ item for word in distinctext for item in word if len(item) >2 ]))

    #replacing all emails with blank
    for i in emaillist:
        Input_Text = Input_Text.replace(i,'')

    #Output Text after email
    Output_Text_after_email = Input_Text
```

```
subs = re.findall( r'Subject:+.*' , Output_Text_after_email)
subs_2 = [ re.sub(r'\w+:\s?',' ',s)  for s in subs]

#preprocessed Subject
preprocessedSubject = (' '.join([ re.sub(r'\s?[@\r\t]\s?',' ',s)  for s in subs_2])).strip()

for i in subs:
    Output_Text_after_email = Output_Text_after_email.replace(i,'')

Output_Text_after_subject = Output_Text_after_email

temp = re.sub( r'Write to:+.*' , ' ', Output_Text_after_subject)
temp = re.sub( r'From:+.*' , ' ', temp)
temp = re.sub( r'<+.*>' , ' ' ,temp)
temp = re.sub(r'[(]+[a-zA-Z0-9]*[)]' , ' ' ,temp)
temp = re.sub('[\t\n-]' , ' ', temp)
temp = temp.replace('\\',' ')
temp = re.sub(r'\w+:' , ' ', temp)

#Donors Choose
temp = re.sub(r"won't", "will not", temp)
temp = re.sub(r"can\'t", "can not", temp)
temp = re.sub(r"n\'t", " not", temp)
temp = re.sub(r"\'re", " are", temp)
temp = re.sub(r"\'s", " is", temp)
temp = re.sub(r"\'d", " would", temp)
temp = re.sub(r"\'ll", " will", temp)
temp = re.sub(r"\'t", " not", temp)
temp = re.sub(r"\'ve", " have", temp)
temp = re.sub(r"\'m", " am", temp)

#splitting words on _ and retaining the words with len > 2
temp = ' '.join( [k for w in temp.split() for k in w.split('_') if len(k) > 2] )

word = nltk.word_tokenize(temp)
pos_tag = nltk.pos_tag(word)
chunk = nltk.ne_chunk(pos_tag)

named_entities = []
person = []

for t in chunk.subtrees():
    if t.label() == 'GPE' :
        named_entities.append(list(t))
        elif t.label() == 'PERSON':
            person.append(list(t))

    #list of named entity and Person
    NE = [ " ".join(w for w, t in ele) for ele in named_entities ]
    P = [ " ".join(w for w, t in ele) for ele in person ]

    for i in NE:
        temp = temp.replace(i, '_'.join([w for w in i.split()]) )

    for i in P:
        temp = temp.replace(i,'')

    temp = re.sub(r"\d" ,' ', temp)

    #removing _ from beginning and ending of words
    temp = ' '.join([w.strip('_') for w in temp.split()])



    #converting to lower
    temp = temp.lower()

    temp = ' '.join( [w for w in temp.split() if len(w) < 15 and len(w) > 2] )

    temp = re.sub(r"[^a-zA-Z_\s]" , ' ' , temp)

    preprocessedText = temp

    return (preprocessedText,preprocessedSubject,preprocessedEmail )



classlabellst = []
featurelst = []

for idx , vals in tqdm(df.iterrows()):

    preprocessed_text,preprocessed_subject,preprocessed_emails = preprocess(vals.iat[0])
    text = vals.iat[0]
    classlabel = vals.iat[1]

    featurelst.append(preprocessed_text + ' ' + preprocessed_subject + ' ' + preprocessed_emails)
    classlabellst.append(classlabel)
```

```
final_df= pd.DataFrame()

final_df['feature'] = featurelst
final_df['doctype'] = classlabellst
```

In [ ]:

```
print (final_df.head(10))
```

```
                                             feature      doctype
0  archive alt atheism archive resources last dec...  alt.atheism
1  archive alt atheism archive introduction last ...  alt.atheism
2  article   well  has quite different  not neces...  alt.atheism
3      until kings become philosophers philosophe...  alt.atheism
4  article however  hate economic terrorism and p...  alt.atheism
5  article did not you say was created with perfe...  alt.atheism
6    the motto originated the star spangled banne...  alt.atheism
7  article  gregg jaeger    when they are victimi...  alt.atheism
8   reference line trimmed         there good dea...  alt.atheism
9  kmr  po cwru edu   then why people keep asking...  alt.atheism
```

In [ ]:

```
# final_df.to_pickle("./final_df.pkl")
```

In [ ]:

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

```
Mounted at /content/gdrive
```

In [ ]:

```
final_df = pd.read_csv("/content/gdrive/MyDrive/Colab Notebooks/final_dfcsv.csv")
```

In [ ]:

```
print(final_df.head(10))
```

```
   Unnamed: 0                                            feature      doctype
0           0  archive alt atheism archive resources last dec...  alt.atheism
1           1  archive alt atheism archive introduction last ...  alt.atheism
2           2  article   well  has quite different  not neces...  alt.atheism
3           3      until kings become philosophers philosophe...  alt.atheism
4           4  article however  hate economic terrorism and p...  alt.atheism
5           5  article did not you say was created with perfe...  alt.atheism
6           6    the motto originated the star spangled banne...  alt.atheism
7           7  article  gregg jaeger    when they are victimi...  alt.atheism
8           8   reference line trimmed         there good dea...  alt.atheism
9           9  kmr  po cwru edu   then why people keep asking...  alt.atheism
```

In [ ]:

```
import tensorflow as tf
import os
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras import applications
from tensorflow.keras.layers import Dense,Conv2D,MaxPool2D,Activation,Dropout,Flatten,GlobalAveragePooling2D,Conv1D, MaxP
from tensorflow.keras import Input
from tensorflow.keras import layers
import random as rn
from tensorflow.keras.models import Model
from keras.callbacks import Callback,EarlyStopping,ModelCheckpoint ,TensorBoard
import datetime
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.utils import np_utils
from keras.layers import Embedding
from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical
from sklearn.metrics import f1_score
```

MODEL 1

In [ ]:

```
x = final_df['feature']
y = final_df['doctype']

t = Tokenizer(filters='_')
t.fit_on_texts(x)
# print(t.word_index)
vocab_size = len(t.word_index) + 1
# print(vocab_size)
# integer encode the documents
encoded_docs = t.texts_to_sequences(x)
# print(len(encoded_docs[:1][0]))
# print(encoded_docs)
# pad documents to a max length of 4 words
max_length = 350
padded_docs = pad_sequences(encoded_docs, maxlen=max_length, padding='post')
# print(padded_docs[:1])

# https://stackoverflow.com/questions/56227671/how-can-i-one-hot-encode-a-list-of-strings-with-keras/56227965


from sklearn.model_selection import train_test_split

X_train , X_test , y_train , y_test = train_test_split(padded_docs, y, stratify = y , test_size = 0.25 , random_state=42)

print(len(X_train))
print(len(X_test))
```

```
y_train = np.array(y_train)
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)

y_test = np.array(y_test)
label_encoder = LabelEncoder()
y_test = label_encoder.fit_transform(y_test)
y_testf1 = label_encoder.fit_transform(y_test)

y_train = np_utils.to_categorical(y_train, num_classes=20)
y_test = np_utils.to_categorical(y_test, num_classes= 20)

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

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

In [ ]:

```
embeddings_index = dict()

f = open('/content/gdrive/MyDrive/Colab Notebooks/glove.6B.300d.txt')
for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
f.close()
print('Loaded %s word vectors.' % len(embeddings_index))

# create a weight matrix for words in training docs

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

```
Loaded 400000 word vectors.
```

In [ ]:

```
input_layer = Input(shape=(350,) ,dtype='int32')


embeddingLayer1 = Embedding(  input_dim = vocab_size , output_dim = 300, weights = [embedding_matrix] , trainable = False
conv1a = Conv1D( filters = 128, kernel_size = 3 , strides = 2, padding = 'same',activation='relu' )(embeddingLayer1)
conv2a = Conv1D( filters = 64, kernel_size = 3 ,strides = 2,  padding = 'same',activation='relu' )(embeddingLayer1)
conv3a = Conv1D( filters = 32, kernel_size = 3 ,strides = 2,  padding = 'same',activation='relu' )(embeddingLayer1)

concat_layera = layers.concatenate ( [conv1a , conv2a , conv3a ] )

pool1 = MaxPool1D (pool_size=5 , strides=2 , padding='same') (concat_layera)

conv1b = Conv1D( filters = 128, kernel_size = 3 , strides = 2, padding = 'same',activation='relu' )(pool1)
conv2b = Conv1D( filters = 64, kernel_size =3 , strides = 2, padding = 'same',activation='relu' )(pool1)
conv3b = Conv1D( filters = 32, kernel_size = 3 , strides = 2, padding = 'same',activation='relu')(pool1)

concat_layerb = layers.concatenate ([ conv1b , conv2b , conv3b ])

pool2 = MaxPool1D(pool_size=10 , strides=3, padding='same') (concat_layerb)

conv1c = Conv1D( filters = 32, kernel_size = 3 , strides = 2 ,padding = 'same',activation='relu' )(pool2)

flat1 = Flatten()(conv1c)

drop1 = Dropout(0.1)(flat1)

FC1 = Dense(units=256 ,activation='relu')(drop1)
Out = Dense(units=20,activation='softmax')(FC1)

finalmodel = Model(input_layer , Out)
finalmodel.summary()
```

```
Model: "model_25"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to                     
==================================================================================================
input_27 (InputLayer)           [(None, 350)]        0                                            
__________________________________________________________________________________________________
embedding_26 (Embedding)        (None, 350, 300)     25679100    input_27[0][0]                   
__________________________________________________________________________________________________
conv1d_175 (Conv1D)             (None, 175, 128)     115328      embedding_26[0][0]               
__________________________________________________________________________________________________
conv1d_176 (Conv1D)             (None, 175, 64)      57664       embedding_26[0][0]               
__________________________________________________________________________________________________
conv1d_177 (Conv1D)             (None, 175, 32)      28832       embedding_26[0][0]               
__________________________________________________________________________________________________
concatenate_50 (Concatenate)    (None, 175, 224)     0           conv1d_175[0][0]                 
                                                                 conv1d_176[0][0]                 
                                                                 conv1d_177[0][0]                 
__________________________________________________________________________________________________
max_pooling1d_50 (MaxPooling1D) (None, 88, 224)      0           concatenate_50[0][0]             
```

```
__________________________________________________________________________________________________
conv1d_178 (Conv1D)             (None, 44, 128)      86144       max_pooling1d_50[0][0]
__________________________________________________________________________________________________
conv1d_179 (Conv1D)             (None, 44, 64)       43072       max_pooling1d_50[0][0]
__________________________________________________________________________________________________
conv1d_180 (Conv1D)             (None, 44, 32)       21536       max_pooling1d_50[0][0]
__________________________________________________________________________________________________
concatenate_51 (Concatenate)    (None, 44, 224)      0           conv1d_178[0][0]
                                                                 conv1d_179[0][0]
                                                                 conv1d_180[0][0]
__________________________________________________________________________________________________
max_pooling1d_51 (MaxPooling1D) (None, 15, 224)      0           concatenate_51[0][0]
__________________________________________________________________________________________________
conv1d_181 (Conv1D)             (None, 8, 32)        21536       max_pooling1d_51[0][0]
__________________________________________________________________________________________________
flatten_25 (Flatten)            (None, 256)          0           conv1d_181[0][0]
__________________________________________________________________________________________________
dropout_25 (Dropout)            (None, 256)          0           flatten_25[0][0]
__________________________________________________________________________________________________
dense_51 (Dense)                (None, 256)          65792       dropout_25[0][0]
__________________________________________________________________________________________________
dense_52 (Dense)                (None, 20)           5140        dense_51[0][0]
==================================================================================================
Total params: 26,124,144
Trainable params: 445,044
Non-trainable params: 25,679,100
__________________________________________________________________________________________________
```

In [ ]:

```
class Metrics(Callback):

    def on_train_begin(self, logs={}):
        self.f1sc=[]


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

          val_predict = (np.argmax(finalmodel.predict(X_test), axis=-1))
          val_targ = y_testf1.astype(int)


          _val_f1 = f1_score(val_targ, val_predict , average='micro').round(4)
          self.f1sc.append(_val_f1)
          print("\nValidation F1Score : " , _val_f1,'\n')

metrics_custom = Metrics()
# adam = tf.keras.optimizers.Adam(learning_rate=0.001)
```

```
adam = tf.keras.optimizers.Adam(lr=1e-2, epsilon=1e-07)

filepath="/content/gdrive/MyDrive/Colab Notebooks/model_save/best_model_1.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath,save_freq = 'epoch', monitor='val_accuracy',  verbose=1, mode='max' ,save_

logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))

earlystop = EarlyStopping (monitor='val_accuracy', min_delta=0.001, patience=2, verbose=1 , mode = 'auto', restore_best_w

tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir,histogram_freq=1, write_graph=True)

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

finalmodel.fit(X_train , y_train,epochs=100,validation_data=(X_test , y_test),  callbacks=[earlystop,tensorboard_callback
```

```
Epoch 1/100
  3/442 [..............................] - ETA: 35s - loss: 3.0326 - accuracy: 0.1076 WARNING:tensorflow:Callback method
`on_train_batch_end` is slow compared to the batch time (batch time: 0.0120s vs `on_train_batch_end` time: 0.0255s). Chec
k your callbacks.
442/442 [==============================] - 7s 14ms/step - loss: 2.8800 - accuracy: 0.1085 - val_loss: 2.0569 - val_accura
cy: 0.3229

Epoch 00001: val_accuracy improved from -inf to 0.32292, saving model to /content/gdrive/MyDrive/Colab Notebooks/model_sa
ve/best_model_1.hdf5

Validation F1Score :  0.3229

Epoch 2/100
442/442 [==============================] - 6s 13ms/step - loss: 1.8573 - accuracy: 0.3742 - val_loss: 1.4659 - val_accura
cy: 0.5082

Epoch 00002: val_accuracy improved from 0.32292 to 0.50818, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_1.hdf5

Validation F1Score :  0.5082

Epoch 3/100
442/442 [==============================] - 6s 13ms/step - loss: 1.3304 - accuracy: 0.5467 - val_loss: 1.2288 - val_accura
cy: 0.5777

Epoch 00003: val_accuracy improved from 0.50818 to 0.57765, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_1.hdf5

Validation F1Score :  0.5777

Epoch 4/100
442/442 [==============================] - 6s 13ms/step - loss: 1.0826 - accuracy: 0.6334 - val_loss: 1.1163 - val_accura
cy: 0.6142
```

```
Epoch 00004: val_accuracy improved from 0.57765 to 0.61419, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_1.hdf5

Validation F1Score :  0.6142

Epoch 5/100
442/442 [==============================] - 6s 13ms/step - loss: 0.9332 - accuracy: 0.6913 - val_loss: 1.0309 - val_accura
cy: 0.6526

Epoch 00005: val_accuracy improved from 0.61419 to 0.65264, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_1.hdf5

Validation F1Score :  0.6526

Epoch 6/100
442/442 [==============================] - 6s 13ms/step - loss: 0.7857 - accuracy: 0.7420 - val_loss: 1.0157 - val_accura
cy: 0.6586

Epoch 00006: val_accuracy improved from 0.65264 to 0.65859, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_1.hdf5

Validation F1Score :  0.6586

Epoch 7/100
442/442 [==============================] - 6s 13ms/step - loss: 0.6619 - accuracy: 0.7836 - val_loss: 0.9597 - val_accura
cy: 0.6786

Epoch 00007: val_accuracy improved from 0.65859 to 0.67856, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_1.hdf5

Validation F1Score :  0.6786

Epoch 8/100
442/442 [==============================] - 6s 13ms/step - loss: 0.5810 - accuracy: 0.8106 - val_loss: 0.9539 - val_accura
cy: 0.6902

Epoch 00008: val_accuracy improved from 0.67856 to 0.69025, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_1.hdf5

Validation F1Score :  0.6902

Epoch 9/100
442/442 [==============================] - 6s 13ms/step - loss: 0.4950 - accuracy: 0.8421 - val_loss: 0.9475 - val_accura
cy: 0.6968

Epoch 00009: val_accuracy improved from 0.69025 to 0.69683, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_1.hdf5
```

```
Validation F1Score :  0.6968

Epoch 10/100
442/442 [==============================] - 6s 13ms/step - loss: 0.4102 - accuracy: 0.8726 - val_loss: 0.9321 - val_accura
cy: 0.7066

Epoch 00010: val_accuracy improved from 0.69683 to 0.70661, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_1.hdf5

Validation F1Score :  0.7066

Epoch 11/100
442/442 [==============================] - 6s 13ms/step - loss: 0.3313 - accuracy: 0.9006 - val_loss: 0.9228 - val_accura
cy: 0.7166

Epoch 00011: val_accuracy improved from 0.70661 to 0.71659, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_1.hdf5

Validation F1Score :  0.7166

Epoch 12/100
442/442 [==============================] - 6s 13ms/step - loss: 0.2709 - accuracy: 0.9239 - val_loss: 0.9397 - val_accura
cy: 0.7157

Epoch 00012: val_accuracy did not improve from 0.71659

Validation F1Score :  0.7157

Epoch 13/100
442/442 [==============================] - 6s 14ms/step - loss: 0.2162 - accuracy: 0.9458 - val_loss: 0.9341 - val_accura
cy: 0.7302

Epoch 00013: val_accuracy improved from 0.71659 to 0.73019, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_1.hdf5

Validation F1Score :  0.7302

Epoch 14/100
442/442 [==============================] - 6s 13ms/step - loss: 0.1753 - accuracy: 0.9541 - val_loss: 0.9628 - val_accura
cy: 0.7285

Epoch 00014: val_accuracy did not improve from 0.73019

Validation F1Score :  0.7285

Epoch 15/100
442/442 [==============================] - 6s 13ms/step - loss: 0.1377 - accuracy: 0.9646 - val_loss: 0.9848 - val_accura
cy: 0.7344
```

```
Epoch 00015: val_accuracy improved from 0.73019 to 0.73444, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_1.hdf5

Validation F1Score :  0.7344

Epoch 16/100
442/442 [==============================] - 6s 13ms/step - loss: 0.1080 - accuracy: 0.9743 - val_loss: 1.0215 - val_accura
cy: 0.7272

Epoch 00016: val_accuracy did not improve from 0.73444

Validation F1Score :  0.7272

Epoch 17/100
442/442 [==============================] - 6s 13ms/step - loss: 0.0835 - accuracy: 0.9845 - val_loss: 1.1101 - val_accura
cy: 0.7208
Restoring model weights from the end of the best epoch.

Epoch 00017: val_accuracy did not improve from 0.73444

Validation F1Score :  0.7344

Epoch 00017: early stopping
```

Out[ ]:

```
<tensorflow.python.keras.callbacks.History at 0x7f185863b630>
```

In [ ]:

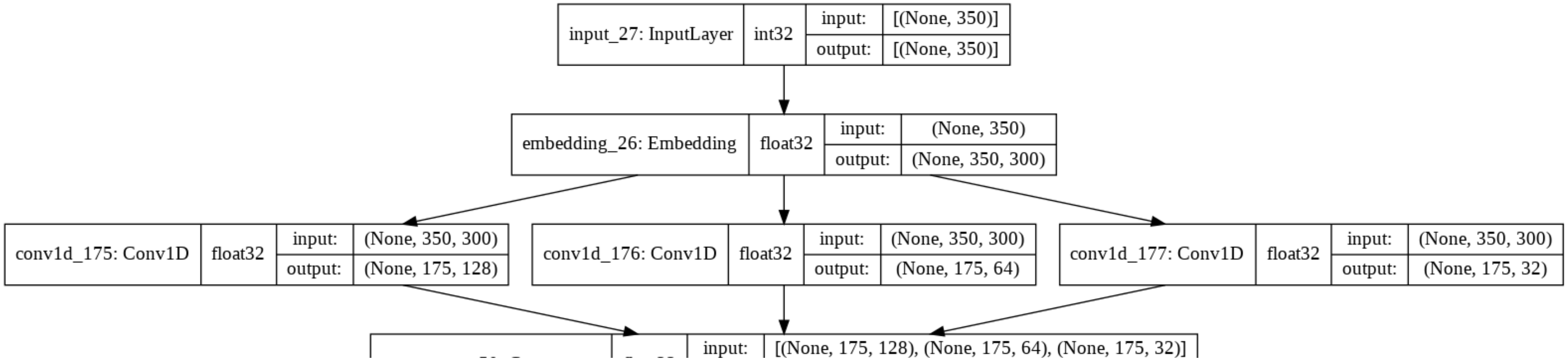
```
%load_ext tensorboard
%tensorboard --logdir logs
```

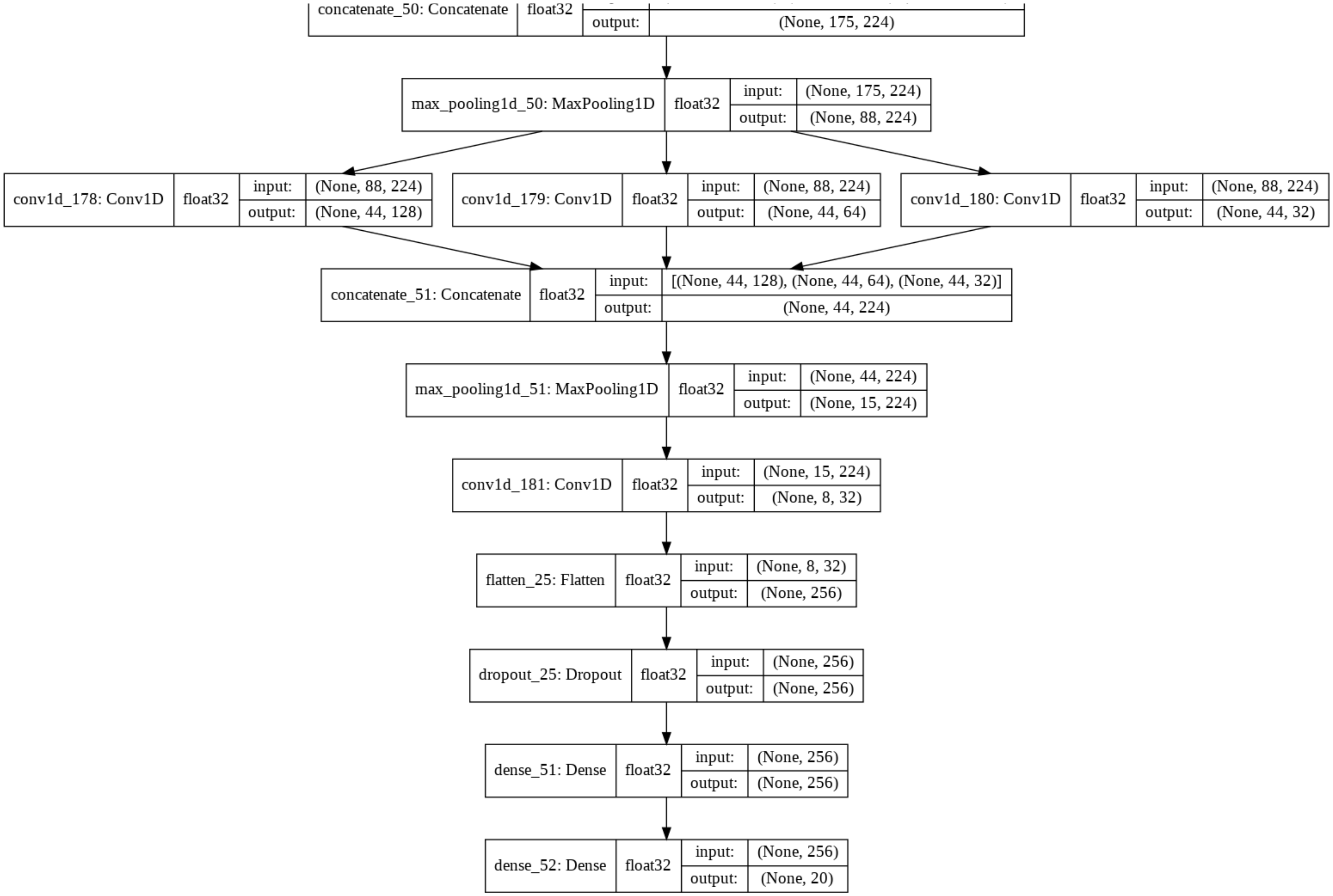
```
Output hidden; open in https://colab.research.google.com to view.
```

In [ ]:

```
tf.keras.utils.plot_model(
    finalmodel, to_file='model1.png', show_shapes=True, show_dtype=True,
    show_layer_names=True,dpi=96
)
```

Out[ ]:





MODEL 2

In [ ]:
```
x = final_df['feature']
y = final_df['doctype']

t = Tokenizer(filters='_' , char_level = True)
t.fit_on_texts(x)
```

In [ ]:

```
vocab_size =  len(t.word_index) + 1

encoded_docs = t.texts_to_sequences(x)

max_length = 350

padded_docs = pad_sequences(encoded_docs, maxlen=max_length, padding='post')

# https://stackoverflow.com/questions/56227671/how-can-i-one-hot-encode-a-list-of-strings-with-keras/56227965


from sklearn.model_selection import train_test_split

X_train , X_test , y_train , y_test = train_test_split(padded_docs, y, stratify = y , test_size = 0.25 , random_state=42)

print(len(X_train))
print(len(X_test))

y_train = np.array(y_train)
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)

y_test = np.array(y_test)
label_encoder = LabelEncoder()
y_test = label_encoder.fit_transform(y_test)
y_testf1 = label_encoder.fit_transform(y_test)

y_train = np_utils.to_categorical(y_train, num_classes=20)
y_test = np_utils.to_categorical(y_test, num_classes= 20)

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

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

In [ ]:

```
embeddings_index = {}
f = open('/content/gdrive/MyDrive/Colab Notebooks/glove.840B.300d-char.txt')
for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
```

```
f.close()

print('Loaded %s word vectors.' % len(embeddings_index))

# create a weight matrix for words in training docs

embedding_matrix = np.zeros((vocab_size, 300))

for word, i in t.word_index.items():
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
                embedding_matrix[i] = embedding_vector
```

```
Loaded 94 word vectors.
```

In [ ]:

```
input_layer = Input(shape=(350,) ,dtype='int32')


embeddingLayer = Embedding(  input_dim = vocab_size , output_dim = 300, weights = [embedding_matrix] , trainable = False)

conv1 = Conv1D( filters = 128, kernel_size = 3 , strides = 2, padding = 'same',activation='relu' )(embeddingLayer)
conv2 = Conv1D( filters = 128, kernel_size = 3 , strides = 2, padding = 'same',activation='relu' )(conv1)

pool1 = MaxPool1D (pool_size=5 , strides=2 , padding='same') (conv2)

conv3 = Conv1D( filters = 128, kernel_size = 3 , strides = 2, padding = 'same',activation='relu' )(pool1)
conv4 = Conv1D( filters = 128, kernel_size =3 , strides = 2, padding = 'same',activation='relu' )(conv3)

pool2 = MaxPool1D(pool_size=10 , strides=3, padding='same') (conv4)

flat1 = Flatten()(pool2)
drop1 = Dropout(0.2)(flat1)

FC1 = Dense(units=256 ,activation='relu')(drop1)
Out = Dense(units=20,activation='softmax')(FC1)

finalmodel = Model(input_layer , Out)
finalmodel.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_3 (InputLayer)         [(None, 350)]             0         
_________________________________________________________________
embedding_1 (Embedding)      (None, 350, 300)          21900     
_________________________________________________________________
```

```
conv1d_4 (Conv1D)            (None, 175, 128)          115328    
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 88, 128)           49280     
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 44, 128)           0         
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 22, 128)           49280     
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 11, 128)           49280     
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 4, 128)            0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328    
_________________________________________________________________
dense_3 (Dense)              (None, 20)                5140      
=================================================================
Total params: 421,536
Trainable params: 399,636
Non-trainable params: 21,900
_________________________________________________________________
```

In [ ]:

```
class Metrics(Callback):

    def on_train_begin(self, logs={}):
        self.f1sc=[]


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

          val_predict = (np.argmax(finalmodel.predict(X_test), axis=-1))
          val_targ = y_testf1.astype(int)


          _val_f1 = f1_score(val_targ, val_predict , average='micro').round(4)
          self.f1sc.append(_val_f1)
          print("\nValidation F1Score : " , _val_f1,'\n')

metrics_custom = Metrics()
# adam = tf.keras.optimizers.Adam(learning_rate=0.001)

adam = tf.keras.optimizers.Adam(lr=1e-3, epsilon=1e-07)

filepath="/content/gdrive/MyDrive/Colab Notebooks/model_save/best_model_2.hdf5"
```

```
checkpoint = ModelCheckpoint(filepath=filepath,save_freq = 'epoch', monitor='val_accuracy',  verbose=1, mode='max' ,save_

logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))

earlystop = EarlyStopping (monitor='val_accuracy', min_delta=0.001, patience=2, verbose=1 , mode = 'auto', restore_best_w

tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir,histogram_freq=1, write_graph=True)

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

finalmodel.fit(X_train , y_train,epochs=100,validation_data=(X_test , y_test),  callbacks=[earlystop,tensorboard_callback
```

```
Epoch 1/100
  3/442 [..............................] - ETA: 28s - loss: 3.0006 - accuracy: 0.0747 WARNING:tensorflow:Callback method
`on_train_batch_end` is slow compared to the batch time (batch time: 0.0095s vs `on_train_batch_end` time: 0.0198s). Chec
k your callbacks.
442/442 [==============================] - 5s 10ms/step - loss: 2.9868 - accuracy: 0.0609 - val_loss: 2.9412 - val_accura
cy: 0.0833

Epoch 00001: val_accuracy improved from -inf to 0.08328, saving model to /content/gdrive/MyDrive/Colab Notebooks/model_sa
ve/best_model_2.hdf5

Validation F1Score :  0.0833

Epoch 2/100
442/442 [==============================] - 4s 10ms/step - loss: 2.9332 - accuracy: 0.0835 - val_loss: 2.9166 - val_accura
cy: 0.0943

Epoch 00002: val_accuracy improved from 0.08328 to 0.09433, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_2.hdf5

Validation F1Score :  0.0943

Epoch 3/100
442/442 [==============================] - 4s 9ms/step - loss: 2.8941 - accuracy: 0.0990 - val_loss: 2.8814 - val_accurac
y: 0.1060

Epoch 00003: val_accuracy improved from 0.09433 to 0.10601, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_2.hdf5

Validation F1Score :  0.106

Epoch 4/100
442/442 [==============================] - 4s 9ms/step - loss: 2.8538 - accuracy: 0.1128 - val_loss: 2.8615 - val_accurac
y: 0.1171

Epoch 00004: val_accuracy improved from 0.10601 to 0.11706, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_2.hdf5
```

```
Validation F1Score :  0.1171

Epoch 5/100
442/442 [==============================] - 4s 10ms/step - loss: 2.8099 - accuracy: 0.1309 - val_loss: 2.8425 - val_accura
cy: 0.1224

Epoch 00005: val_accuracy improved from 0.11706 to 0.12237, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_2.hdf5

Validation F1Score :  0.1224

Epoch 6/100
442/442 [==============================] - 4s 9ms/step - loss: 2.7745 - accuracy: 0.1341 - val_loss: 2.8380 - val_accurac
y: 0.1385

Epoch 00006: val_accuracy improved from 0.12237 to 0.13852, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_2.hdf5

Validation F1Score :  0.1385

Epoch 7/100
442/442 [==============================] - 4s 9ms/step - loss: 2.7072 - accuracy: 0.1548 - val_loss: 2.8050 - val_accurac
y: 0.1398

Epoch 00007: val_accuracy improved from 0.13852 to 0.13979, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_2.hdf5

Validation F1Score :  0.1398

Epoch 8/100
442/442 [==============================] - 4s 9ms/step - loss: 2.6450 - accuracy: 0.1744 - val_loss: 2.7702 - val_accurac
y: 0.1570

Epoch 00008: val_accuracy improved from 0.13979 to 0.15700, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_2.hdf5

Validation F1Score :  0.157

Epoch 9/100
442/442 [==============================] - 4s 9ms/step - loss: 2.5588 - accuracy: 0.2084 - val_loss: 2.7821 - val_accurac
y: 0.1617

Epoch 00009: val_accuracy improved from 0.15700 to 0.16167, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_2.hdf5

Validation F1Score :  0.1617

Epoch 10/100
```

```
442/442 [==============================] - 4s 10ms/step - loss: 2.4712 - accuracy: 0.2428 - val_loss: 2.8095 - val_accura
cy: 0.1649

Epoch 00010: val_accuracy improved from 0.16167 to 0.16486, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_2.hdf5

Validation F1Score :  0.1649

Epoch 11/100
442/442 [==============================] - 4s 10ms/step - loss: 2.4150 - accuracy: 0.2526 - val_loss: 2.8213 - val_accura
cy: 0.1700

Epoch 00011: val_accuracy improved from 0.16486 to 0.16996, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_2.hdf5

Validation F1Score :  0.17

Epoch 12/100
442/442 [==============================] - 4s 9ms/step - loss: 2.3108 - accuracy: 0.2864 - val_loss: 2.8474 - val_accurac
y: 0.1725

Epoch 00012: val_accuracy improved from 0.16996 to 0.17251, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_2.hdf5

Validation F1Score :  0.1725

Epoch 13/100
442/442 [==============================] - 4s 9ms/step - loss: 2.2240 - accuracy: 0.3128 - val_loss: 2.8675 - val_accurac
y: 0.1729

Epoch 00013: val_accuracy improved from 0.17251 to 0.17293, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_2.hdf5

Validation F1Score :  0.1729

Epoch 14/100
442/442 [==============================] - 4s 9ms/step - loss: 2.1448 - accuracy: 0.3308 - val_loss: 2.9396 - val_accurac
y: 0.1797

Epoch 00014: val_accuracy improved from 0.17293 to 0.17973, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
_save/best_model_2.hdf5

Validation F1Score :  0.1797

Epoch 15/100
442/442 [==============================] - 4s 9ms/step - loss: 2.0365 - accuracy: 0.3681 - val_loss: 2.9985 - val_accurac
y: 0.1836

Epoch 00015: val_accuracy improved from 0.17973 to 0.18356, saving model to /content/gdrive/MyDrive/Colab Notebooks/model
```

```
_save/best_model_2.hdf5

Validation F1Score :  0.1836

Epoch 16/100
442/442 [==============================] - 4s 9ms/step - loss: 1.9720 - accuracy: 0.3959 - val_loss: 3.0972 - val_accurac
y: 0.1759

Epoch 00016: val_accuracy did not improve from 0.18356

Validation F1Score :  0.1759

Epoch 17/100
442/442 [==============================] - 4s 9ms/step - loss: 1.8581 - accuracy: 0.4271 - val_loss: 3.1766 - val_accurac
y: 0.1763
Restoring model weights from the end of the best epoch.

Epoch 00017: val_accuracy did not improve from 0.18356

Validation F1Score :  0.1836

Epoch 00017: early stopping
```

Out[ ]:

```
<tensorflow.python.keras.callbacks.History at 0x7f0f2ae4e7f0>
```

In [ ]:

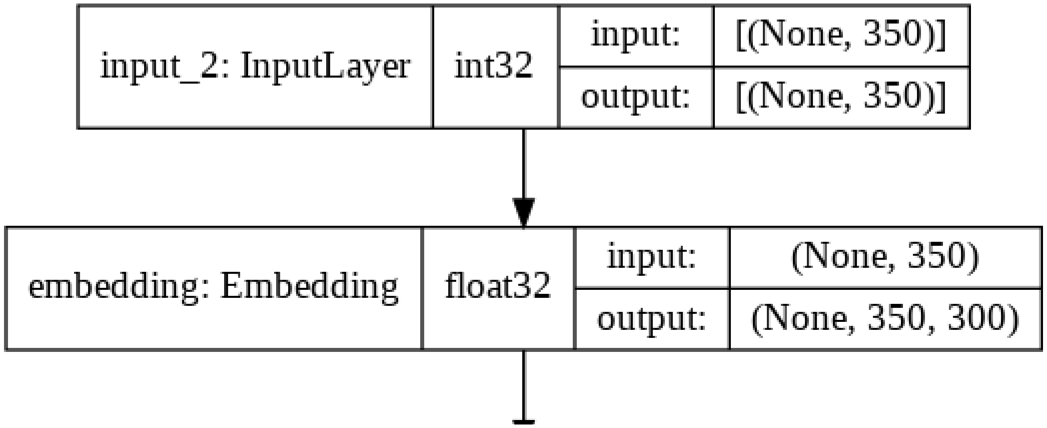
```
%load_ext tensorboard
%tensorboard --logdir logs
```

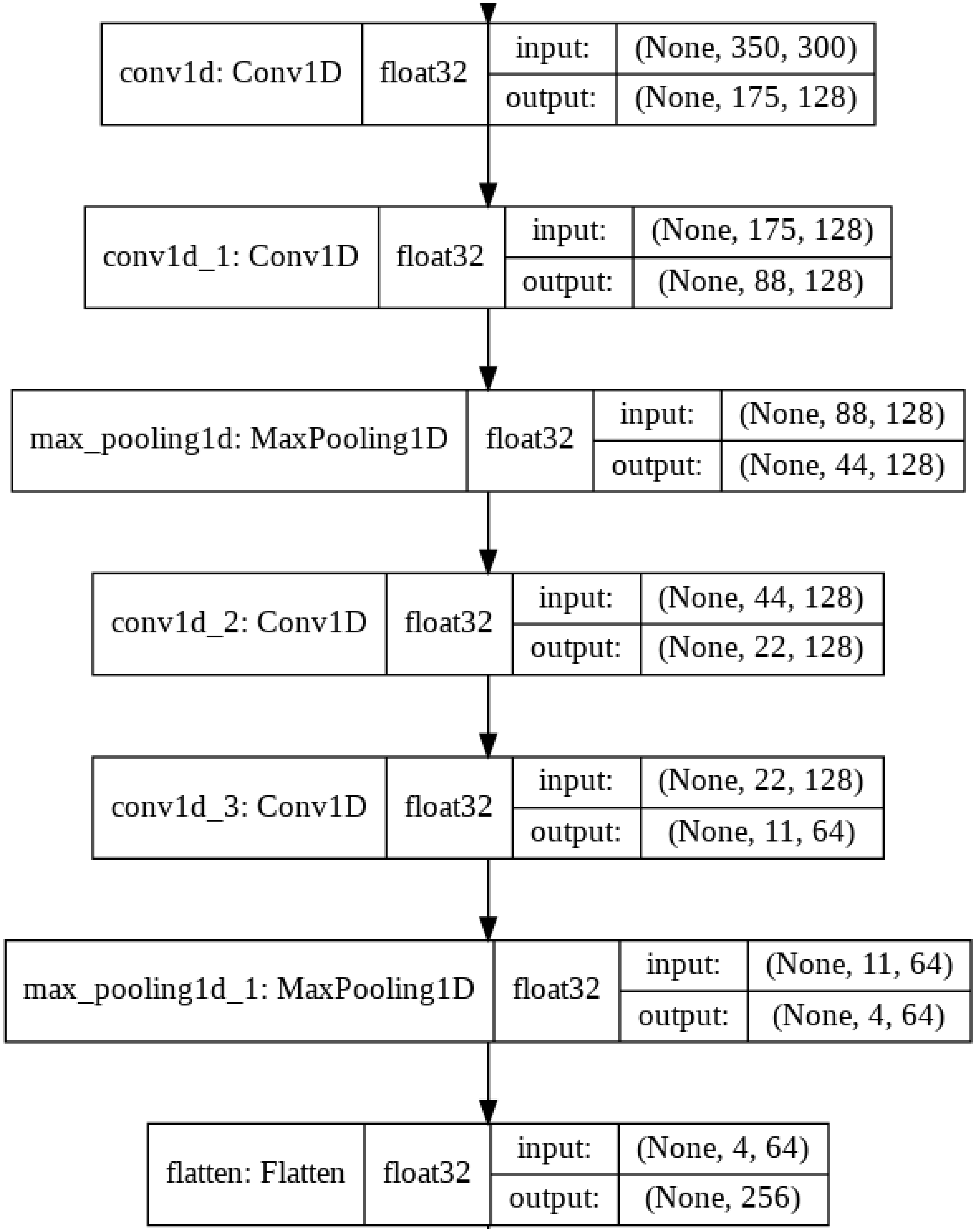
```
Output hidden; open in https://colab.research.google.com to view.
```

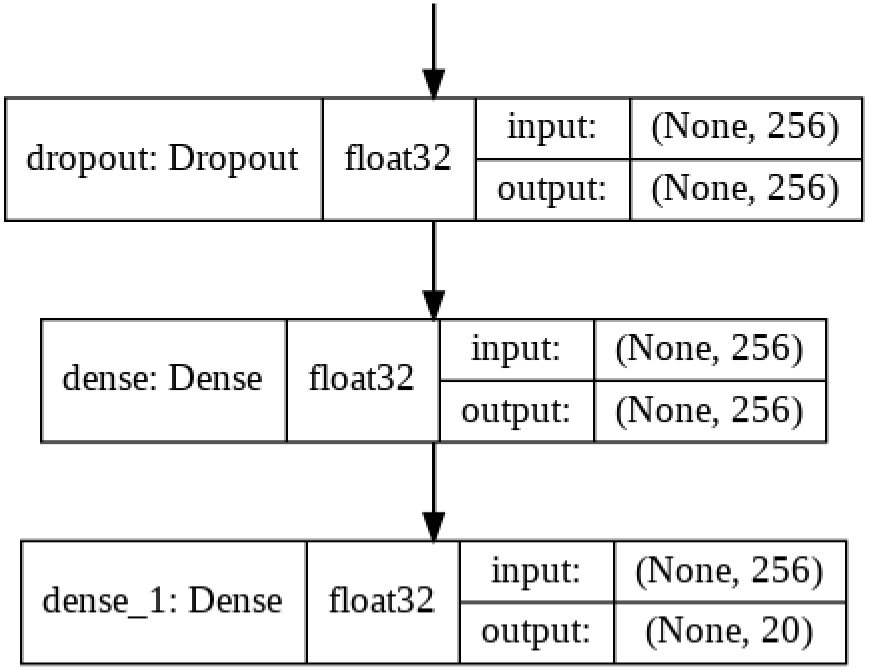
In [ ]:

```
tf.keras.utils.plot_model(
    finalmodel, to_file='model2.png', show_shapes=True, show_dtype=True,
    show_layer_names=True,dpi=96
)
```

Out[ ]:




conv1d: Conv1D
float32
input:
(None, 350, 300)
output:
(None, 175, 128)
conv1d_1: Conv1D
float32
input:
(None, 175, 128)
output:
(None, 88, 128)
max_pooling1d: MaxPooling1D
float32
input:
(None, 88, 128)
output:
(None, 44, 128)
conv1d_2: Conv1D
float32
input:
(None, 44, 128)
output:
(None, 22, 128)
conv1d_3: Conv1D
float32
input:
(None, 22, 128)
output:
(None, 11, 64)
max_pooling1d_1: MaxPooling1D
float32
input:
(None, 11, 64)
output:
(None, 4, 64)
flatten: Flatten
float32
input:
(None, 4, 64)
output:
(None, 256)



Code checking:

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

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

Training The models to Classify:

```
1. Combine "preprocessed_text", "preprocessed_subject", "preprocessed_emails" into one column. use that
column to model.

2. Now Split the data into Train and test. use 25% for test also do a stratify split.

3. Analyze your text data and pad the sequnce if required.
Sequnce length is not restricted, you can use anything of your choice.
```

you need to give the reasoning

4. Do Tokenizer i.e convert text into numbers. please be careful while doing it.
if you are using tf.keras "Tokenizer" API, it removes the **"_"**, but we need that.

5. code the model's (Model-1, Model-2) as discussed below
and try to optimize that models.

6. For every model use predefined Glove vectors.
Don't train any word vectors while Training the model.

7. Use "categorical_crossentropy" as Loss.

8. Use **Accuracy and Micro Avgeraged F1 score** as your as Key metrics to evaluate your model.

9. Use Tensorboard to plot the loss and Metrics based on the epoches.

10. Please save your best model weights in to **'best_model_L.h5' (L = 1 or 2)**.

11. You are free to choose any Activation function, learning rate, optimizer.
But have to use the same architecture which we are giving below.

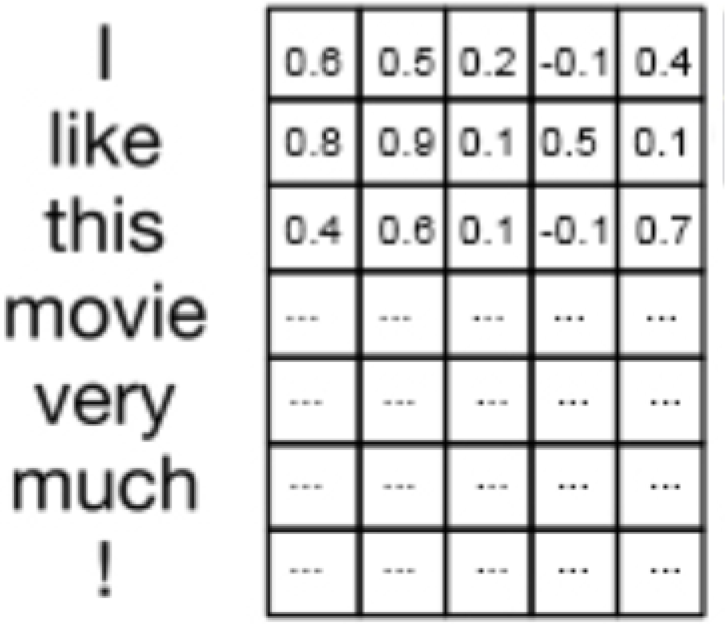
12. You can add some layer to our architecture but you **deletion** of layer is not acceptable.

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

14. For Every model save your model to image (Plot the model) with shapes
and inlcude those images in the notebook markdown cell,
upload those imgages to Classroom. You can use "plot_model"
please refer this if you don't know how to plot the model with shapes.

Model-1: Using 1D convolutions with word embeddings

Encoding of the Text --> For a given text data create a Matrix with Embedding layer as shown Below.
In the example we have considered d = 5, but in this assignment we will get d = dimension of Word vectors we are using.
i.e if we have maximum of 350 words in a sentence and embedding of 300 dim word vector,
we result in 350*300 dimensional matrix for each sentance as output after embedding layer



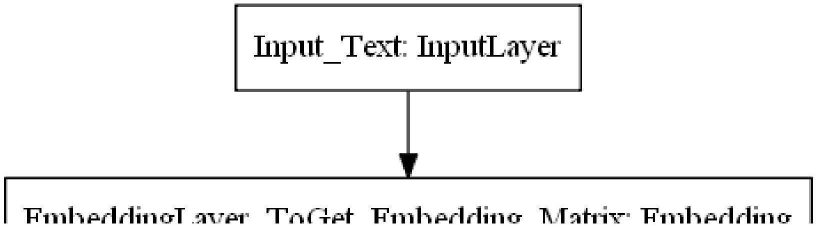
I	0.6	0.5	0.2	-0.1	0.4
like	0.8	0.9	0.1	0.5	0.1
this	0.4	0.6	0.1	-0.1	0.7
movie	...	...	...	...	...
very	...	...	...	...	...
much	...	...	...	...	...
!	...	...	...	...	...

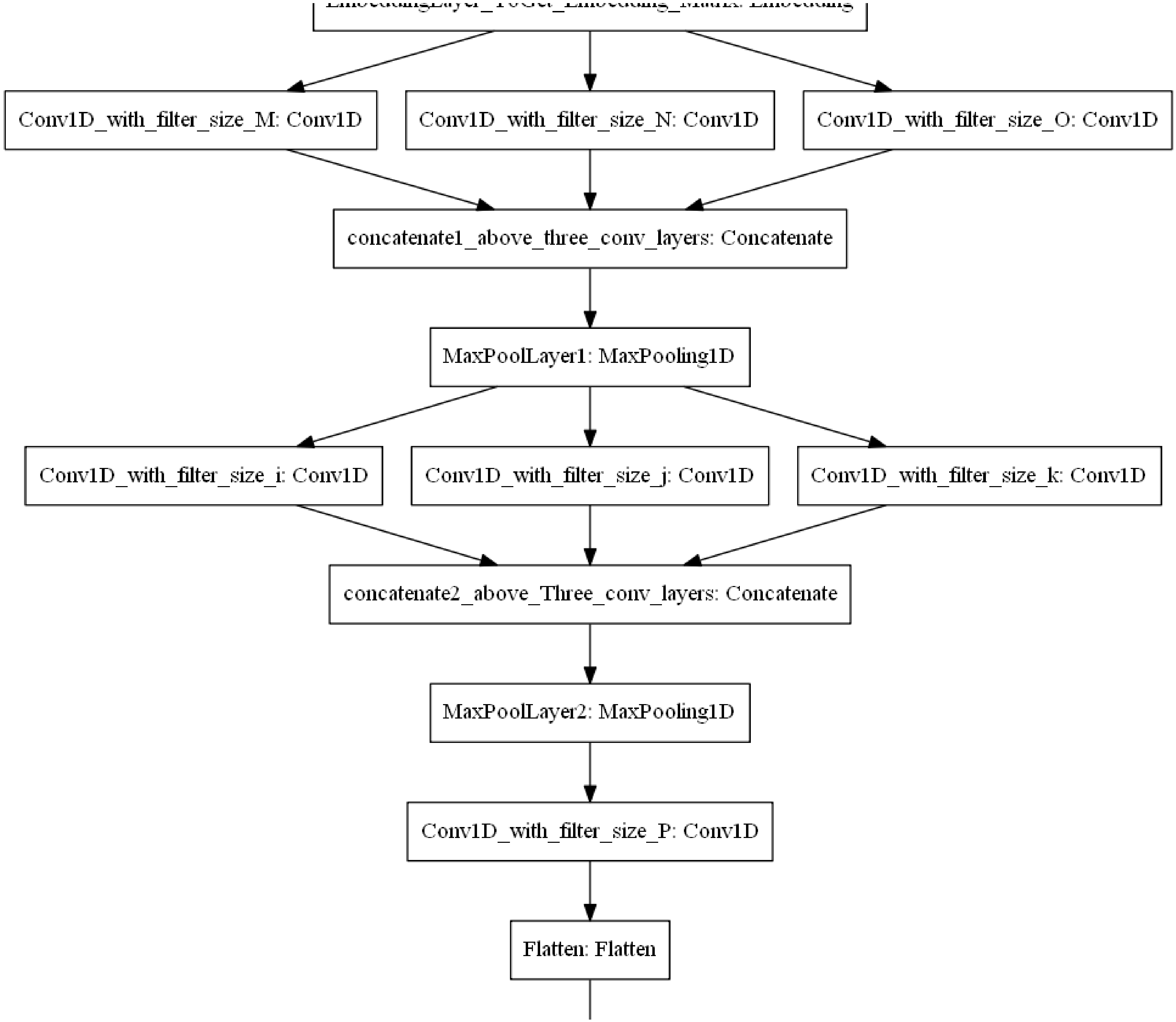
Ref: https://i.imgur.com/kiVQuk1.png

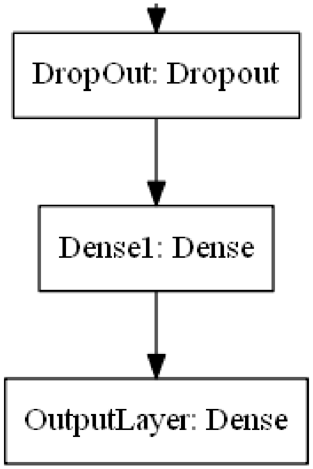
Reference:
https://stackoverflow.com/a/43399308/4084039
https://missinglink.ai/guides/keras/keras-conv1d-working-1d-convolutional-neural-networks-keras/

How EMBEDDING LAYER WORKS

Go through this blog, if you have any doubt on using predefined Embedding values in Embedding layer - https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/



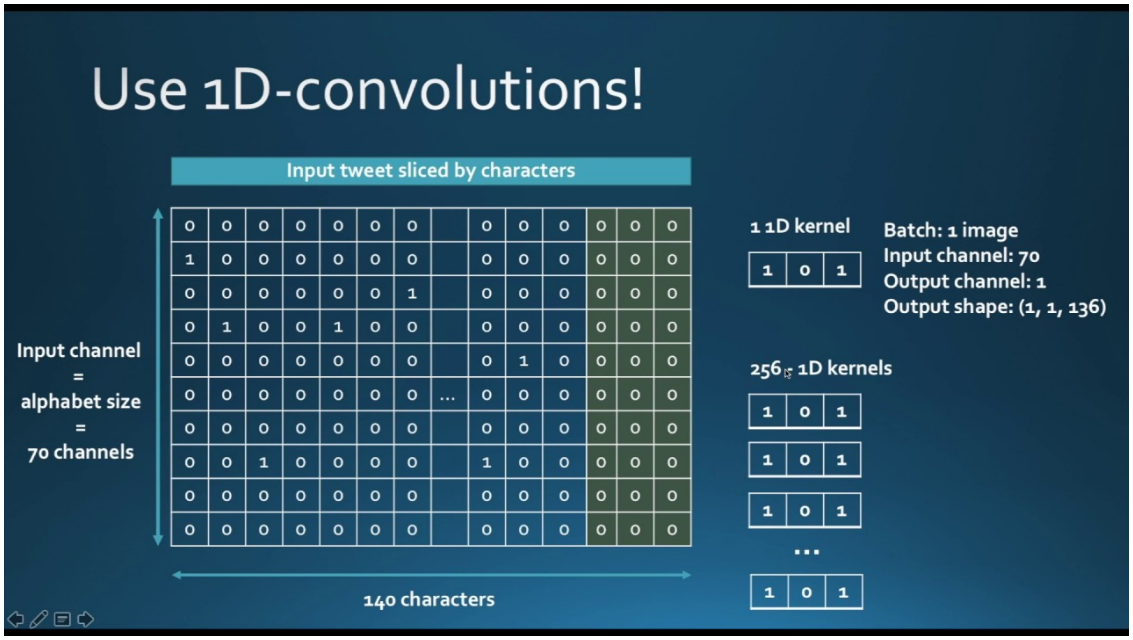

EmbeddingLayer_ToGet_Embedding_Matrix: Embedding
Conv1D_with_filter_size_M: Conv1D
Conv1D_with_filter_size_N: Conv1D
Conv1D_with_filter_size_O: Conv1D
concatenate1_above_three_conv_layers: Concatenate
MaxPoolLayer1: MaxPooling1D
Conv1D_with_filter_size_i: Conv1D
Conv1D_with_filter_size_j: Conv1D
Conv1D_with_filter_size_k: Conv1D
concatenate2_above_Three_conv_layers: Concatenate
MaxPoolLayer2: MaxPooling1D
Conv1D_with_filter_size_P: Conv1D
Flatten: Flatten



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

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

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

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

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

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

Model-2 : Using 1D convolutions with character embedding



Use 1D-convolutions!

Input tweet sliced by characters

Input channel = alphabet size = 70 channels

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

140 characters

1 1D kernel

1	0	1

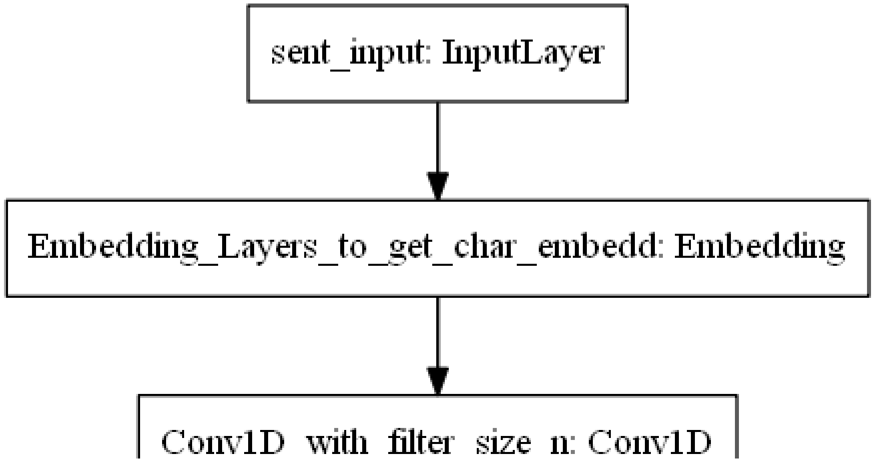
Batch: 1 image
Input channel: 70
Output channel: 1
Output shape: (1, 1, 136)

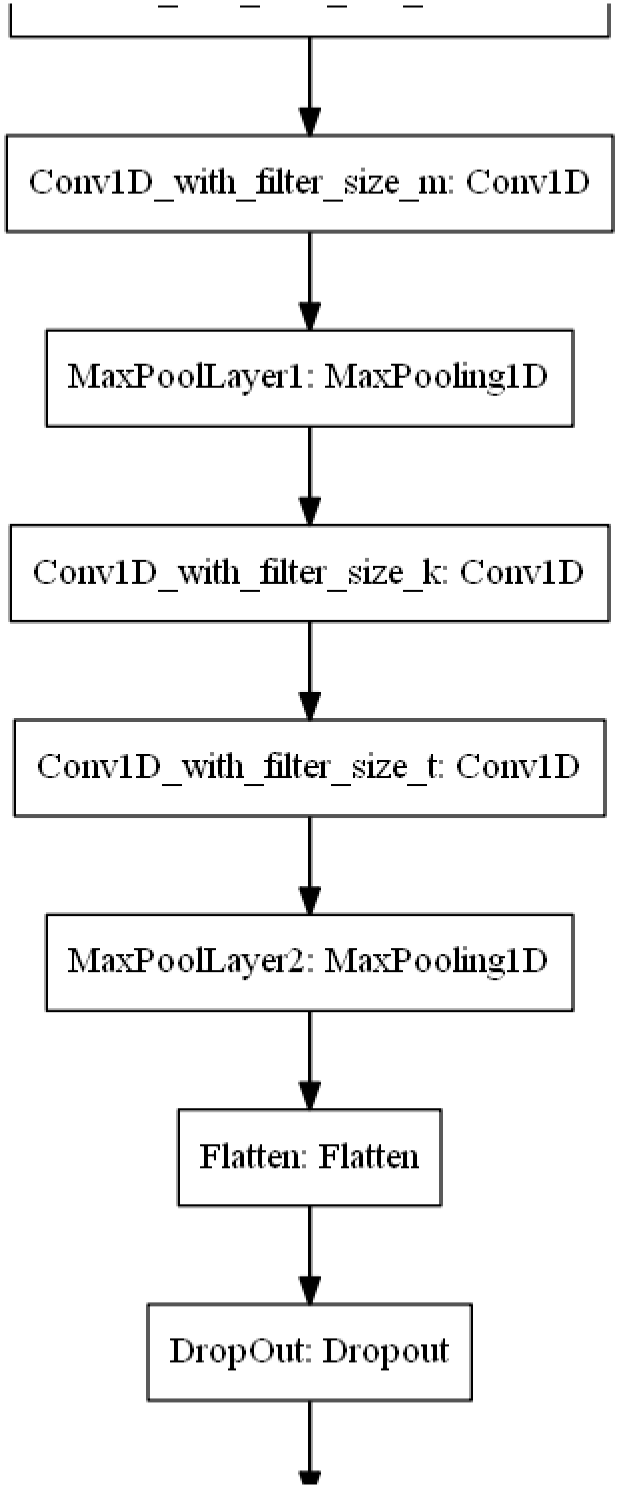
256 1D kernels

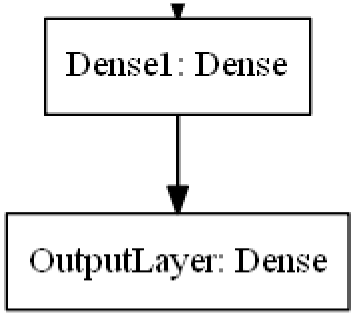
1	0	1
1	0	1
1	0	1
...		
1	0	1

Here are the some papers based on Char-CNN

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


sent_input: InputLayer
Embedding_Layers_to_get_char_embedd: Embedding
Conv1D_with_filter_size_n: Conv1D


Conv1D_with_filter_size_m: Conv1D
MaxPoolLayer1: MaxPooling1D
Conv1D_with_filter_size_k: Conv1D
Conv1D_with_filter_size_t: Conv1D
MaxPoolLayer2: MaxPooling1D
Flatten: Flatten
DropOut: Dropout


Dense1: Dense
OutputLayer: Dense