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?
```

In [ ]:

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

In article < 65882@mimsy.umd.edu > mangoe@cs.umd.edu (Charley Wingate) writes:
>
>I've said 100 times that there is no "alternative" that should think you
>might have caught on by now.  And there is no "alternative", but the point
>is, "rationality" isn't an alternative either.  The problems of metaphysical
>and religious knowledge are unsolvable-- or I should say, humans cannot
>solve them.
How does that saying go: Those who say it can't be done shouldn't interrupt
those who are doing it.
Jim
```

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

In [ ]:

```
#https://stackoverflow.com/questions/48660547/how-can-i-extract-gpelocation-using-nltk-ne
-chunk
from nltk import word_tokenize, pos_tag, ne_chunk
from nltk import Tree
import regex as re
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')
from nltk import word_tokenize, pos_tag, ne_chunk
from nltk.chunk import tree2conlltags
import numpy as np
from tqdm import tqdm
import json
import tensorflow as tf
```

Text Preprocessing

In [ ]:

```
class Preprocess():
  def __init__(self,phrase):
    self.phrase = phrase

  def list_string(self,sample):
    string = ' '
    for i in sample:
      string+=' '+i
    return string

  def decontracted(self,sample):

    # specific
    phrase = re.sub("won\\'t", "will not", sample)
    phrase = re.sub("can\\'t", "can not", phrase)
    phrase = re.sub("shouldn\\'t","should not",phrase)

    # general
    phrase = re.sub("n\\'t", " not", phrase)
    phrase = re.sub("\\'re", " are", phrase)
    phrase = re.sub("\\'s", " is", phrase)
    phrase = re.sub("\\'d", " would", phrase)
    phrase = re.sub("\\'ll", " will", phrase)
    phrase = re.sub("\\'t", " not", phrase)
    phrase = re.sub("\\'ve", " have",phrase)
    phrase = re.sub("\\'m", " am",phrase)
    return phrase
  def underscore_remover(self,sample):
      string = ' '
      for i in sample.split():
        if len(i)>4:
          if i[0] == '_' and i[-1]=='_':
            i=i[1:-1]
        if len(i)>3:
          if i[1] == '_':
            i = i[2:]
        if len(i)>4:
          if i[2] == '_':
            i = i[3:]
        i = re.sub('.*_',' ',i)
        string+=' '+i
      return string
  def get_continuous_chunks(self,text, label):
    chunked = ne_chunk(pos_tag(word_tokenize(text)))
    prev = None
    continuous_chunk = []
    current_chunk = []

    for subtree in chunked:
        if type(subtree) == Tree and subtree.label() == label:
            current_chunk.append(" ".join([token for token, pos in subtree.leaves()]))
        if current_chunk:
            named_entity = " ".join(current_chunk)
            if named_entity not in continuous_chunk:
                continuous_chunk.append(named_entity)
                current_chunk = []
        else:
            continue
    #print('done')
    return continuous_chunk
  def lower_case(self,sample):
    string = ' '
    for i in sample.split():
      i = i.lower()
      if (len(i)>=15 or len(i)<=2):
        continue
      string+=' '+i
    return string

  def clean(self,sample):
    #subject_list = []
    mail_domain = []
    l = re.sub('\(.*\)',' ',sample)
    l = self.decontracted(l)
    #print(l)

    subject = re.findall('Subject:.*',l)
    subject = self.list_string(subject)
    subject = re.sub('[^A-Za-z_]',' ',subject)
    subject = self.lower_case(subject)
    subject = re.sub('subject',' ',subject)

    subject = self.lower_case(subject)
    #subject = subject.split()
    #subject = subject[1:]
    #subject_list.append(subject)

    l = re.sub('Subject:.*\\n',' ',l)
    l = re.sub('\\t',' ',l)
    #print(l)
    m = re.findall('@\S+',l)
    for i in range(len(m)):
      end = re.sub('@',' ',m[i])
      end = end.split('.')
      mail_domain+=end
    mail = ' '
    for i in mail_domain:
      i = re.sub('[^A-Za-z]',' ',i)
      i = re.sub('["\/\\:\-\?>\\t\\n]',' ',i)
      mail+=' '+i

    mail = re.sub('[^A-Za-z]',' ',mail)
    mail = self.lower_case(mail)

    l = re.sub('From:.*',' ',l)

    l = re.sub('.*writes:.*',' ',l)
    l = re.sub('\S+:',' ',l)
    l = re.sub('<.*>',' ',l)
    #print(l)
    l = re.sub('["\/\\:\-\?>\\t\\n]',' ',l)
    l = re.sub('\S+@\S+',' ',l)
# removing person names
    names = self.get_continuous_chunks(l,'PERSON')
    #print(names)
    places = self.get_continuous_chunks(l,'GPE')
```

```
        #print(places)
        #print(l)
        #print(names)
        #print(places)
        for name in names:
          name = self.lower_case(name)
          name = re.sub('[\/\\:\-\?>\\t\\n]',' ',name)
          name = re.sub('[^A-Za-z]',' ',name)
          name = self.underscore_remover(name)

          l = re.sub(name,' ',l)
       # print(l)
        for place in places:
          if len(place.split())>1:
            place = self.underscore_remover(place)
            place = re.sub('[\/\\:\-\?>\\t\\n]',' ',place)
            place = self.lower_case(place)
            l = re.sub(place,re.sub(' ','_',place),l)
        l = re.sub('[0-9]',' ',l)
        #print(l)
        l = self.underscore_remover(l) #removing works like _word_
        l = re.sub('[^A-Za-z_]',' ',l)
        l = self.lower_case(l)
        #print(l)
        return l,subject,mail
```

Preprocessing:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

In [ ]:

```
!pip install nltk
```

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

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

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

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

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

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

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

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

Code checking:

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

In [ ]:

```
with open('/content/alt.atheism_49960.txt', mode="r", encoding="utf-8", errors = 'ignore'
) as f:
  data = f.read()
Pre = Preprocess(data)
t,s,m = Pre.clean(data)
print('text:',t)
print('subject:',s)
print('mail:',m)
f.close()
```

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

In [ ]:

```
import pandas as pd
```

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 spli

t.

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

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

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

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

7. Use "categorical_crossentropy" as Loss.

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

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

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

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

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

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

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

In [ ]:

```
!pwd
!ls
!cd /content/drive/
```

In [ ]:

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

In [ ]:

```
import os
os.chdir('/content/drive/MyDrive/documents')
```

In [ ]:

```
!ls
```

Data classification

In [ ]:

In [ ]:

```python
# iterate through all file
label = []
data = []
def read_text_file(file_path):
  with open(file_path, mode="r", encoding="utf-8", errors = 'ignore') as f:
    return f.read()
for file in tqdm(os.listdir()):
    # Check whether file is in text format or not
    if file.startswith("talk.region.misc"):
        label.append(0)
        file_path = f"{'/content/drive/MyDrive/documents'}/{file}"
        # call read text file function
        data.append(read_text_file(file_path))
    if file.startswith("talk.politics.misc"):
        label.append(1)
        file_path = f"{'/content/drive/MyDrive/documents'}/{file}"
        # call read text file function
        data.append(read_text_file(file_path))
    if file.startswith("talk.politics.mideast"):
        label.append(2)
        file_path = f"{'/content/drive/MyDrive/documents'}/{file}"
        # call read text file function
        data.append(read_text_file(file_path))
    if file.startswith("talk.politics.guns"):
        label.append(3)
        file_path = f"{'/content/drive/MyDrive/documents'}/{file}"
        # call read text file function
        data.append(read_text_file(file_path))
    if file.startswith("soc.religion.christian"):
        label.append(4)
        file_path = f"{'/content/drive/MyDrive/documents'}/{file}"
        # call read text file function
        data.append(read_text_file(file_path))
    if file.startswith("sci.space"):
        label.append(5)
        file_path = f"{'/content/drive/MyDrive/documents'}/{file}"
        # call read text file function
        data.append(read_text_file(file_path))
    if file.startswith("sci.med"):
        label.append(6)
        file_path = f"{'/content/drive/MyDrive/documents'}/{file}"
        # call read text file function
        data.append(read_text_file(file_path))
    if file.startswith("sci.electronics"):
        label.append(7)
        file_path = f"{'/content/drive/MyDrive/documents'}/{file}"
        # call read text file function
        data.append(read_text_file(file_path))
    if file.startswith("sci.crypt"):
        label.append(8)
        file_path = f"{'/content/drive/MyDrive/documents'}/{file}"
        # call read text file function
        data.append(read_text_file(file_path))
    if file.startswith("rec.sport.hockey"):
        label.append(9)
        file_path = f"{'/content/drive/MyDrive/documents'}/{file}"
        # call read text file function
        data.append(read_text_file(file_path))
    if file.startswith("rec.sport.baseball"):
        label.append(10)
        file_path = f"{'/content/drive/MyDrive/documents'}/{file}"
        # call read text file function
        data.append(read_text_file(file_path))
    if file.startswith("rec.motorcycles"):
        label.append(11)
        file_path = f"{'/content/drive/MyDrive/documents'}/{file}"
        # call read text file function
        data.append(read_text_file(file_path))
    if file.startswith("rec.autos"):
        label.append(12)
        file_path = f"{'/content/drive/MyDrive/documents'}/{file}"
        # call read text file function
        data.append(read_text_file(file_path))
    if file.startswith("misc.forsale"):
        label.append(13)
        file_path = f"{'/content/drive/MyDrive/documents'}/{file}"
        # call read text file function
        data.append(read_text_file(file_path))
    if file.startswith("comp.windows.x"):
        label.append(14)
        file_path = f"{'/content/drive/MyDrive/documents'}/{file}"
        # call read text file function
        data.append(read_text_file(file_path))
    if file.startswith("comp.sys.mac.hardware"):
        label.append(15)
        file_path = f"{'/content/drive/MyDrive/documents'}/{file}"
        # call read text file function
        data.append(read_text_file(file_path))
    if file.startswith("comp.sys.ibm.pc.hardware"):
        label.append(16)
        file_path = f"{'/content/drive/MyDrive/documents'}/{file}"
        # call read text file function
        data.append(read_text_file(file_path))
    if file.startswith("comp.os.ms-windows.misc"):
        label.append(17)
        file_path = f"{'/content/drive/MyDrive/documents'}/{file}"
        # call read text file function
        data.append(read_text_file(file_path))
    if file.startswith("comp.graphics"):
        label.append(18)
        file_path = f"{'/content/drive/MyDrive/documents'}/{file}"
        # call read text file function
        data.append(read_text_file(file_path))
    if file.startswith("alt.atheism"):
        label.append(19)
        file_path = f"{'/content/drive/MyDrive/documents'}/{file}"
        # call read text file function
        data.append(read_text_file(file_path))
```

talk.region.misc talk.politics.misc talk.politics.mideast talk.politics.guns soc.religion.christian sci.space sci.med sci.electronics sci.crypt rec.sport.hockey rec.sport.baseball rec.motorcycles rec.autos misc.forsale comp.windows.x comp.sys.mac.hardware comp.sys.ibm.pc.hardware comp.os.ms-windows.misc comp.graphics alt.atheism

In [ ]:

```
len(label)
```

In [ ]:

```
from tqdm import tqdm
```

In [ ]:

```
preprocessed_subject = []
preprocessed_emails = []
preprocessed_text = []
for j in tqdm(np.arange(len(data))):
  p = Preprocess(data[j])
  text,subject,mail = p.clean(data[j])
  #text_chunk = tree2conlltags(ne_chunk(pos_tag(word_tokenize(text))))
  #sub_chunk = tree2conlltags(ne_chunk(pos_tag(word_tokenize(subject))))
  #mail_chunk = tree2conlltags(ne_chunk(pos_tag(word_tokenize(mail))))
  preprocessed_text.append(text)
  preprocessed_subject.append(subject)
  preprocessed_emails.append(mail)
```

In [ ]:

```
preprocessed = pd.DataFrame(preprocessed_text,columns=['text'])
```

In [ ]:

```
preprocessed['subject'] = preprocessed_subject
preprocessed['mail'] = preprocessed_emails
```

In [ ]:

```
len(preprocessed_text)
```

In [ ]:

```
preprocessed_total = []
```

In [ ]:

```
for i in range(len(preprocessed_text)):
  preprocessed_total.append(preprocessed_text[i]+preprocessed_subject[i]+preprocessed_em
ails[i])
```

In [ ]:

```
label = np.array(label)
```

In [ ]:

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
import re

from numpy import array
from keras.preprocessing.text import one_hot
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.embeddings import Embedding
```

In [ ]:

```
#https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
#https://stackoverflow.com/questions/49073673/include-punctuation-in-keras-tokenizer
to_exclude = '!"#$%&()*+-/:;<=>@[\\]^`{|}~\t\n'
max_word = 10000
max_length = 1000
t = Tokenizer(filters=to_exclude,num_words=max_word)
t.fit_on_texts(preprocessed_total)
# define documents
t.word_index
print('words found :',len(t.word_index))
vocab_size = len(t.word_index) + 1
encoded_docs = t.texts_to_sequences(preprocessed_total)
#encoded_test = t.texts_to_sequences(X_test)
#print(encoded_docs)


padded_docs = pad_sequences(encoded_docs,maxlen = max_length,padding='post')
#padded_test = pad_sequences(encoded_test,maxlen = max_length,padding='post')
print(padded_docs.shape)
#print(padded_test.shape)

#labels = tf.keras.utils.to_categorical(label, 20)
print(labels.shape)
```

Train Test Splitting

In [ ]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(padded_docs,label,test_size = 0.25,stratify=labels,random_state=42)
Y_train = tf.keras.utils.to_categorical(y_train)
Y_test = tf.keras.utils.to_categorical(y_test)
```

In [ ]:

```
print(X_train.shape,Y_train.shape)
print(X_test.shape,Y_test.shape)
```

In [ ]:

```
%load_ext tensorboard
# Clear any logs from previous runs
!rm -rf ./logs/
```

In [ ]:

```
!pwd
!cd /content/
os.chdir('/content')
!ls
```

Pretrained Glove Model

In [ ]:

```
#https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
# load the whole embedding into memory
embeddings_index = dict()
f = open('glove.6B.100d.txt')
for line in f:
 values = line.split()
 word = values[0]
 coefs = np.array(values[1:], dtype='float32')
 embeddings_index[word] = coefs
f.close()
print('Loaded %s word vectors.' % len(embeddings_index))
```

In [ ]:

```
#https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
# create a weight matrix for words in training docs
embedding_matrix = np.zeros((vocab_size, 100))
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
```

In [ ]:

```
import numpy
from keras.models import Sequential
from keras.layers.convolutional import Conv1D
from keras.layers import Input
from keras.layers import Concatenate
from keras.layers import MaxPooling1D
from keras.layers import Dropout
from keras.layers import Dense
from keras.layers import Flatten
from tensorflow.keras.models import Model
from sklearn.model_selection import train_test_split
from sklearn.metrics import auc
from tensorflow.keras.callbacks import LearningRateScheduler
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import roc_auc_score
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import EarlyStopping
from sklearn import metrics
from tensorflow.keras.callbacks import LearningRateScheduler
import numpy as np # importing numpy for numerical computation
from itertools import combinations
import os
import tensorflow as tf
import datetime
from tensorflow.keras.utils import plot_model
from tensorflow.keras import metrics
```

Call Backs

In [ ]:

```
from sklearn.metrics import recall_score
class LossHistory(tf.keras.callbacks.Callback):
    def __init__(self,validation_data):
      self.x_test = validation_data[0]
      self.y_test = validation_data[1]
    def on_train_begin(self, logs={}):
        ## on begin of training, we are creating a instance varible called history
        ## it is a dict with keys [loss, acc, val_loss, val_acc]
        self.history={'loss': [],'accuracy': [],'val_loss': [],'val_accuracy': [],'auc':
[]}

    def on_epoch_end(self, epoch, logs={}):
        true_positives=0
        ## on end of each epoch, we will get logs and update the self.history dict
        loss = logs.get('loss')
        if loss is not None:
           if np.isnan(loss) or np.isinf(loss):
             print("Invalid loss and terminated at epoch {}".format(epoch))
             self.model.stop_training = True
        #Terminate if model weights are None
        self.history['loss'].append(loss)
        self.history['accuracy'].append(logs.get('accuracy'))
        if logs.get('val_loss', -1) != -1:
            self.history['val_loss'].append(logs.get('val_loss'))
        if logs.get('val_accuracy', -1) != -1:
            self.history['val_accuracy'].append(logs.get('val_accuracy'))
        # we can get a list of all predicted values at the end of the epoch
        # we can use these predicted value and the true values to calculate any custom ev
aluation score if it is needed for our model
        # Here we are taking log of all true positives and then taking average of it
        y_pred = self.model.predict(self.x_test)
        y_label_pred=np.argmax(y_pred,axis=1)
        #print(y_label_pred)
       # y_pred_prob = y_pred[:,1]
        recall = recall_score(y_test, y_label_pred, average='micro')
        precision = precision_score(y_test, y_label_pred, average='micro')

        #a = pd.DataFrame([y_test,y_pred], columns = ('y','y_pred'))
        custom_score = (2*recall*precision)/(recall+precision)
        #fpr, tpr, thresholds = metrics.roc_curve(y,y_label_pred)

        #we can also calcualte predefined metrics such as precison, recall, etc. using ca
llbacks
        #auc = metrics.auc(fpr, tpr)
        #auc = roc_auc_score(y_test,y_pred_prob)
        #self.history['auc'].append(auc)
        print(' F1: ',np.round(custom_score,5))

history_own=LossHistory(validation_data=[X_test,y_test])
```

Learning Rate Scheduler Function

In [ ]:

```
def changeLearningRate(epoch,lr):
    val = history_own.history['val_accuracy']
    for i in range(len(val)-1):
      if val[i+1] < val[i]:
        lr = 0.9*lr
    if (epoch+1)%3 == 0 :
        lr = 0.95*lr
    return lr
```

Model 1

In [ ]:

```
max_length = 1000
input_text = Input(shape=(1000,),dtype = 'int32',name = "input_text")
#print(input_text.shape)
Embedding_layer = Embedding(vocab_size,100,input_length=max_length, weights=[embedding_matrix],trainable=False,name="Embedding_layer")(input_text)
#print(Embedding_layer.shape)
#print(Embedding_layer.shape)
conv_1d_with_size_3 = Conv1D(filters=3, kernel_size=2, padding='same', kernel_initializer='normal',activation='relu',name = "conv_1d_with_size_8")(Embedding_layer)
conv_1d_with_size_4 = Conv1D(filters=4, kernel_size=2, padding='same', kernel_initializer='normal',activation='relu',name = "conv_1d_with_size_4")(Embedding_layer)
conv_1d_with_size_5 = Conv1D(filters=5, kernel_size=2, padding='same',kernel_initializer='normal', activation='relu',name = "conv_1d_with_size_2")(Embedding_layer)
concatenated1 = Concatenate(name = "concatenated1_above_3_conv_layers")([conv_1d_with_size_3,conv_1d_with_size_4,conv_1d_with_size_5])
maxpool_layer1 = MaxPooling1D(pool_size = 3,name = "MaxPoolLayer1")(concatenated1)
#print(maxpool_layer1.shape)
conv_1d_with_size_3_ = Conv1D(filters=3, kernel_size=2, padding='same',kernel_initializer='normal', activation='relu',name = "conv_1d_with_size_12_")(maxpool_layer1)
conv_1d_with_size_4_ = Conv1D(filters=4, kernel_size=2, padding='same',kernel_initializer='normal' ,activation='relu',name="conv_1d_with_size_4_")(maxpool_layer1)
conv_1d_with_size_5_ = Conv1D(filters=5, kernel_size=2, padding='same',kernel_initializer='normal', activation='relu',name = "conv_1d_with_size_2_")(maxpool_layer1)
concatenated2 = Concatenate(name = "concatenated2_above_3_conv_layers")([conv_1d_with_size_3_,conv_1d_with_size_4_,conv_1d_with_size_5_])
maxpool_layer2 = MaxPooling1D(pool_size = 2,name = "MaxPoolLayer2")(concatenated2)
#print(maxpool_layer2.shape)
conv_1d_with_size_16_ = Conv1D(filters=16, kernel_size=3, padding='same', activation='relu',name = "con_with_filter_size_16")(maxpool_layer2)
flatten = Flatten(name="Flatten")(conv_1d_with_size_16_)
#print(flatten.shape)
dropout = Dropout(0.5,name = "Dropout")(flatten)
dense1 = Dense(50,name = "Dense")(dropout)
outputlayer = Dense(20,activation = 'softmax',name = "outoutLayer")(dense1)
#print(outputlayer.shape)
```

In [ ]:

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

#model

model = Model(inputs=input_text,outputs=outputlayer)

#callbacks

history_own=LossHistory(validation_data=[X_test,y_test])

filepath="model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_loss',  verbose=1, save_best_only=True, mode='auto')

lrschedule = LearningRateScheduler(changeLearningRate)
#earltstopping
earlystop = EarlyStopping(monitor='val_loss',patience=2)
```

```
#adam optimizer

optimizer = tf.keras.optimizers.SGD(learning_rate = 0.1 , momentum=0.1)#callbacks

callback_list = [history_own,lrschedule,earlystop,checkpoint,tensorboard_callback]
model.compile(optimizer=optimizer, loss='categorical_crossentropy',metrics=['accuracy'])
```

Model 1 Summary

In [ ]:

```
model.summary()
plot_model(model)
```

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

In [ ]:

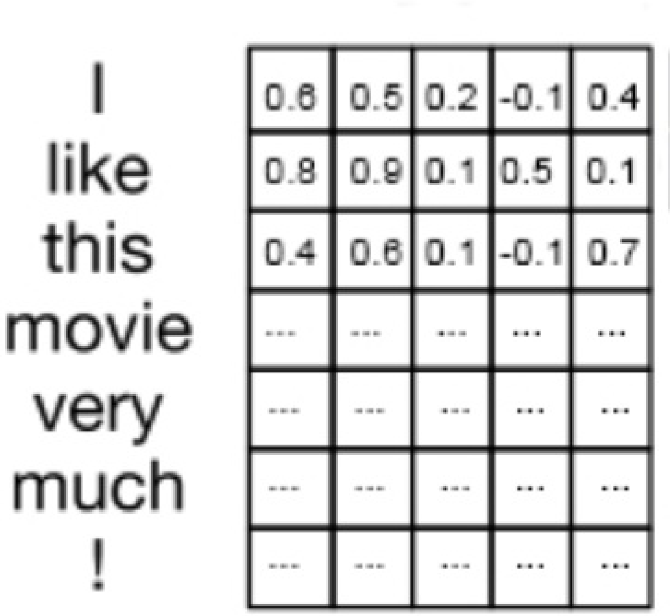
```
model.fit(X_train,Y_train,epochs=25, validation_data=(X_test,Y_test), batch_size=128, callbacks=callback_list)
```

In [ ]:

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

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	...	...	...	...	...
!	...	...	...	...	...

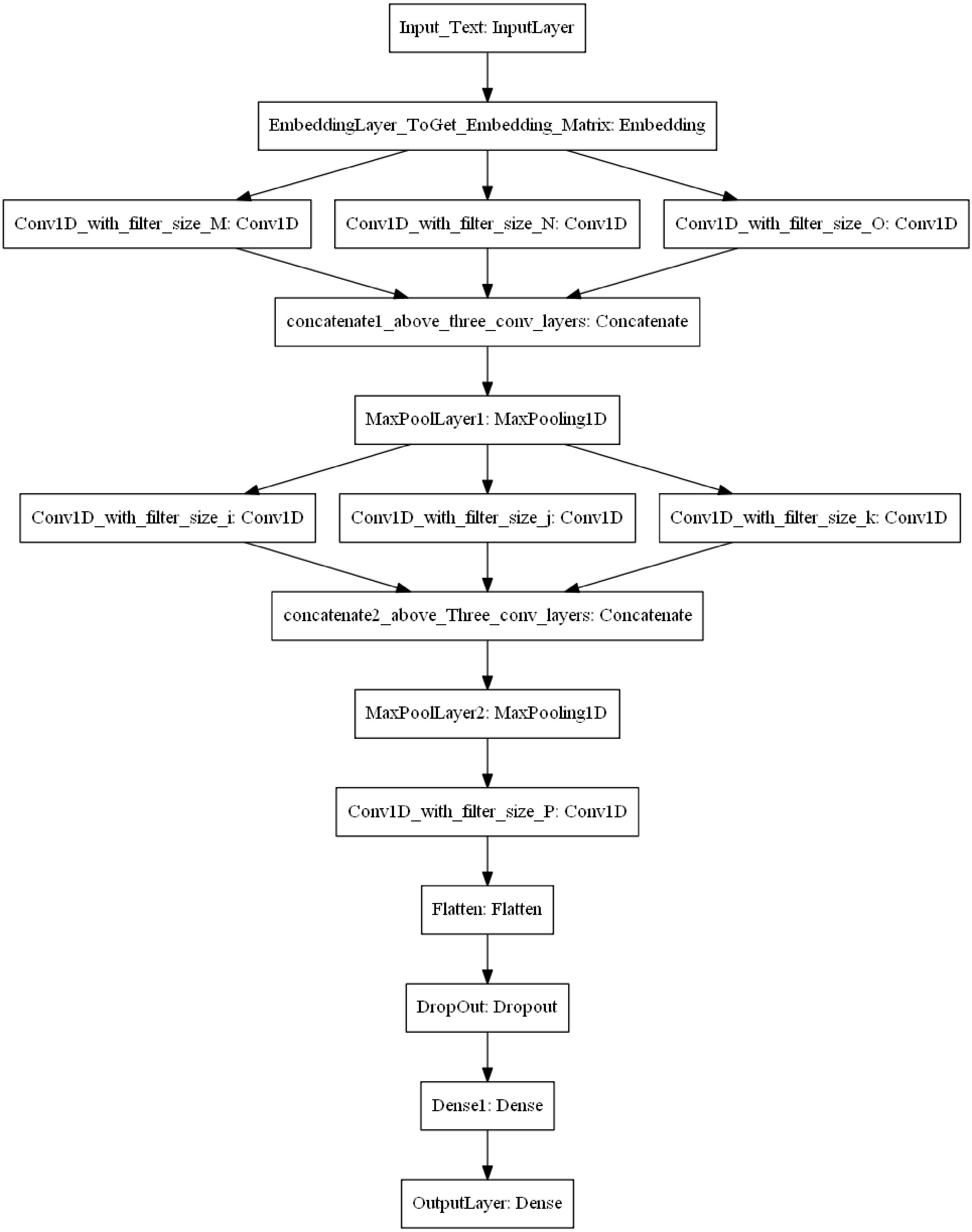
```
Ref: https://i.imgur.com/kiVQuk1.png
```

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/


Input_Text: InputLayer
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
DropOut: Dropout
Dense1: Dense
OutputLayer: Dense

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

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.

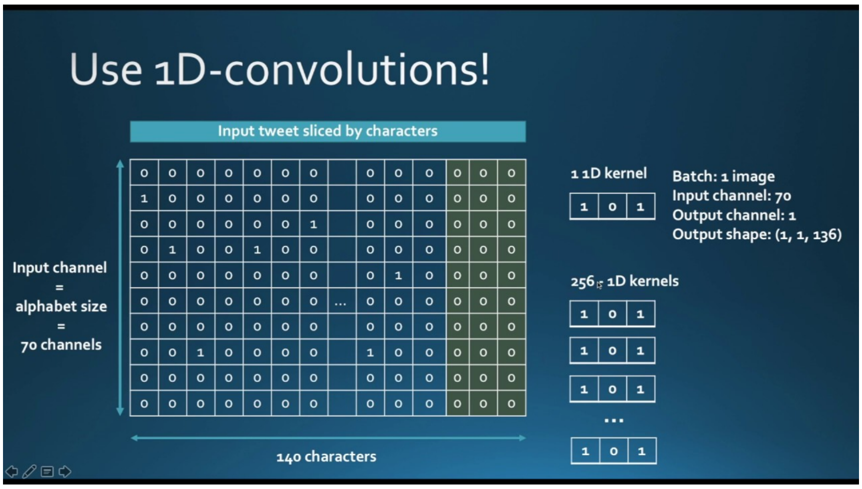
4. Don't use more than 16 filters in one Conv layer becuase it will increase the no of params.
(Only recommendation if you have less computing power)

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

In [ ]:

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
```

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

In [ ]:

```
#https://towardsdatascience.com/character-level-cnn-with-keras-50391c3adf33
to_exclude = '!"#$%&()*+-/:;<=>@[\\]^`{|}~\t\n'
text = preprocessed_total
tk = Tokenizer(filters=to_exclude,num_words=None, char_level=True, oov_token='UNK')
tk.fit_on_texts(preprocessed_total)
# define documents
print(tk.word_index)
```

```
#print(t.word_index)
vocab_size_ = len(tk.word_index) + 1
encoded_docs_ = tk.texts_to_sequences(preprocessed_total)
#print(encoded_docs)

max_length_ = 29
padded_docs_model2 = pad_sequences(encoded_docs_, maxlen=max_length_,padding='post')
print(padded_docs_model2)
```

In [ ]:

```
from sklearn.model_selection import train_test_split
X_train_,X_test_,y_train_,y_test_ = train_test_split(padded_docs_model2,label,test_size =
0.25,stratify=labels,random_state=42)
Y_train_ = tf.keras.utils.to_categorical(y_train)
Y_test_ = tf.keras.utils.to_categorical(y_test)
```

In [ ]:

```
alphabet = "abcdefghijklmnopqrstuvwxyz_"
char_dict = {}
for i, char in enumerate(alphabet):
    char_dict[char] = i + 1

# Use char_dict to replace the tk.word_index
tk.word_index = char_dict.copy()
# Add 'UNK' to the vocabulary
tk.word_index[tk.oov_token] = max(char_dict.values()) + 1
```

In [ ]:

```
!pwd
os.chdir('/content')
```

In [ ]:

```
#https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
# load the whole embedding into memory
embeddings_index = dict()
f = open('glove.840B.300d-char.txt')
for line in f:
 values = line.split()
 word = values[0]
 coefs = np.array(values[1:], dtype='float32')
 embeddings_index[word] = coefs
f.close()
print('Loaded %s word vectors.' % len(embeddings_index))
```

In [ ]:

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

In [ ]:

```
%load_ext tensorboard
# Clear any logs from previous runs
!rm -rf ./logs/
```

In [ ]:

```
sent_input = Input(shape=29,name = "sent_input")
#print(input_text.shape)
Embedding_layer = Embedding(vocab_size,300,input_length=max_length, weights=[embedding_m
atrix],trainable=False,name="Embedding_layer")(sent_input)
#print(Embedding_layer.shape)
conv_1d_with_size_5 = Conv1D(filters=5, kernel_size=2, padding='same',kernel_initializer
='normal',activation='relu',name = "conv_1d_with_size_8")(Embedding_layer)
conv_1d_with_size_4 = Conv1D(filters=4, kernel_size=2, padding='same', kernel_initialize
r='normal',activation='relu',name = "conv_1d_with_size_4")(conv_1d_with_size_5)
#conv_1d_with_size_2 = Conv1D(filters=2, kernel_size=2, padding='same', activation='relu'
,name = "conv_1d_with_size_2")(Embedding_layer)
#concatenated1 = Concatenate(name = "concatenated1_above_3_conv_layers")([conv_1d_with_si
ze_8,conv_1d_with_size_4,conv_1d_with_size_2])
maxpool_layer1 = MaxPooling1D(pool_size = 2,name = "MaxPoolLayer1")(conv_1d_with_size_4)
#print(maxpool_layer1.shape)
conv_1d_with_size_5_ = Conv1D(filters=5, kernel_size=2, padding='same',kernel_initialize
r='normal', activation='relu',name = "conv_1d_with_size_12_")(maxpool_layer1)
conv_1d_with_size_4_ = Conv1D(filters=4, kernel_size=2, padding='same', kernel_initializ
er='normal',activation='relu',name="conv_1d_with_size_4_")(conv_1d_with_size_5_)
#conv_1d_with_size_2_ = Conv1D(filters=2, kernel_size=2, padding='same', activation='relu
',name = "conv_1d_with_size_2_")(maxpool_layer1)
#concatenated2 = Concatenate(name = "concatenated2_above_3_conv_layers")([conv_1d_with_si
ze_8_,conv_1d_with_size_4_,conv_1d_with_size_2_])
maxpool_layer2 = MaxPooling1D(pool_size = 2,name = "MaxPoolLayer2")(conv_1d_with_size_4_
)
#print(maxpool_layer2.shape)
#conv_1d_with_size_16_ = Conv1D(filters=16, kernel_size=2, padding='same', activation='re
lu',name = "con_with_filter_size_16")(maxpool_layer2)
flatten = Flatten(name="Flatten")(maxpool_layer2)
#print(flatten.shape)
dropout = Dropout(0.2,name = "Dropout")(flatten)
dense1 = Dense(100,name = "Dense")(dropout)
outputlayer = Dense(20,name = "outoutLayer",activation='softmax')(dense1)
```

In [ ]:

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

#model

model2 = Model(inputs=sent_input,outputs=outputlayer)

#callbacks

history_own=LossHistory(validation_data=[X_test_,y_test_])

filepath="model_save_/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_loss',  verbose=1, save_bes
t_only=True, mode='auto')

#earltstopping
earlystop = EarlyStopping(monitor='val_loss',patience=2)
#adam optimizer
optimizer = tf.keras.optimizers.Adam()
#callbacks
callback_list = [history_own,earlystop,checkpoint,tensorboard_callback]
model2.compile(optimizer=optimizer, loss='categorical_crossentropy',metrics=['accuracy']
)
```

In [ ]:

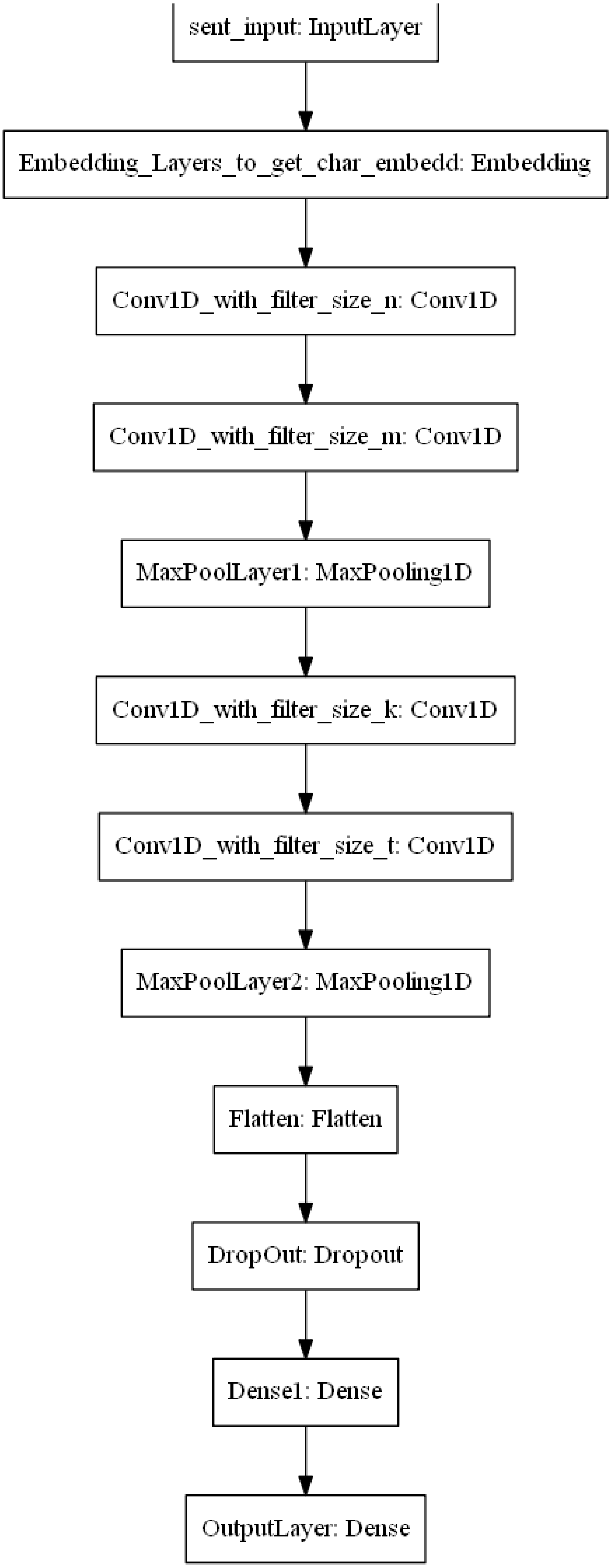
```
model2.summary()
plot_model(model2)
```

In [ ]:

```
model2.fit(X_train_,Y_train_,epochs=10, validation_data=(X_test_,Y_test_), batch_size=16
, callbacks=callback_list)
```

In [ ]:

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


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