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.

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

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.

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

```python
print("i am living in the New York -->", list(chunks))
print(" ")
print("-"*50)
print(" ")
print("My name is Mridul Mazumdar -->", 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 "Mridul Mazumdar",
they got combined and represented as a tree and "New York" was referred as "GPE" and "Mridul Mazumdar" was referred as "PERSON".
so now you have to Combine the "New York" with "_" i.e "New_York"
and remove the "Mridul Mazumdar" 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.

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.

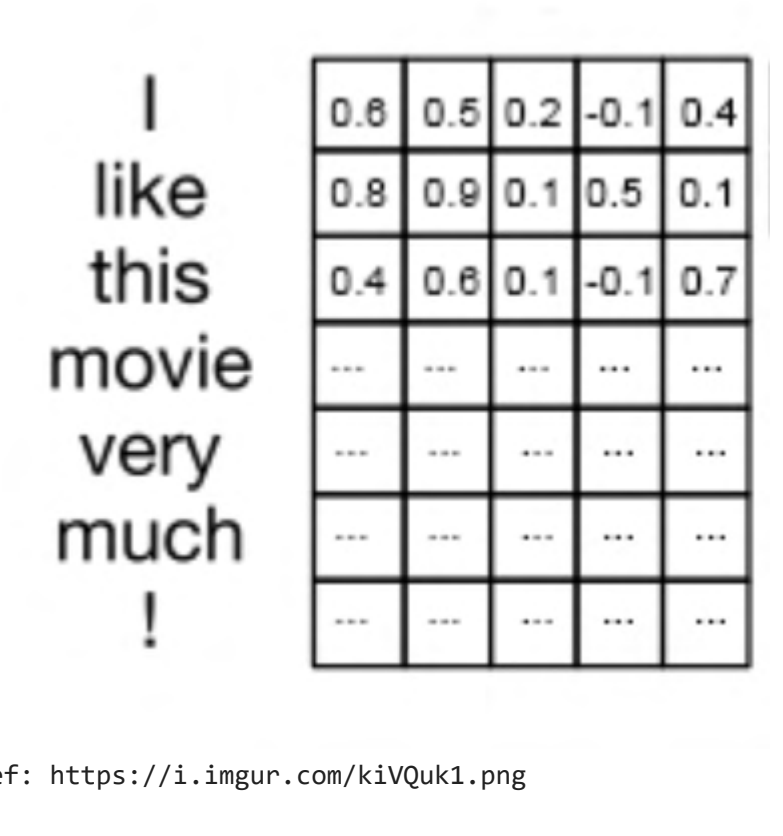
12. You can add some layer to our architecture but you **deletion** of layer is not acceptable.

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

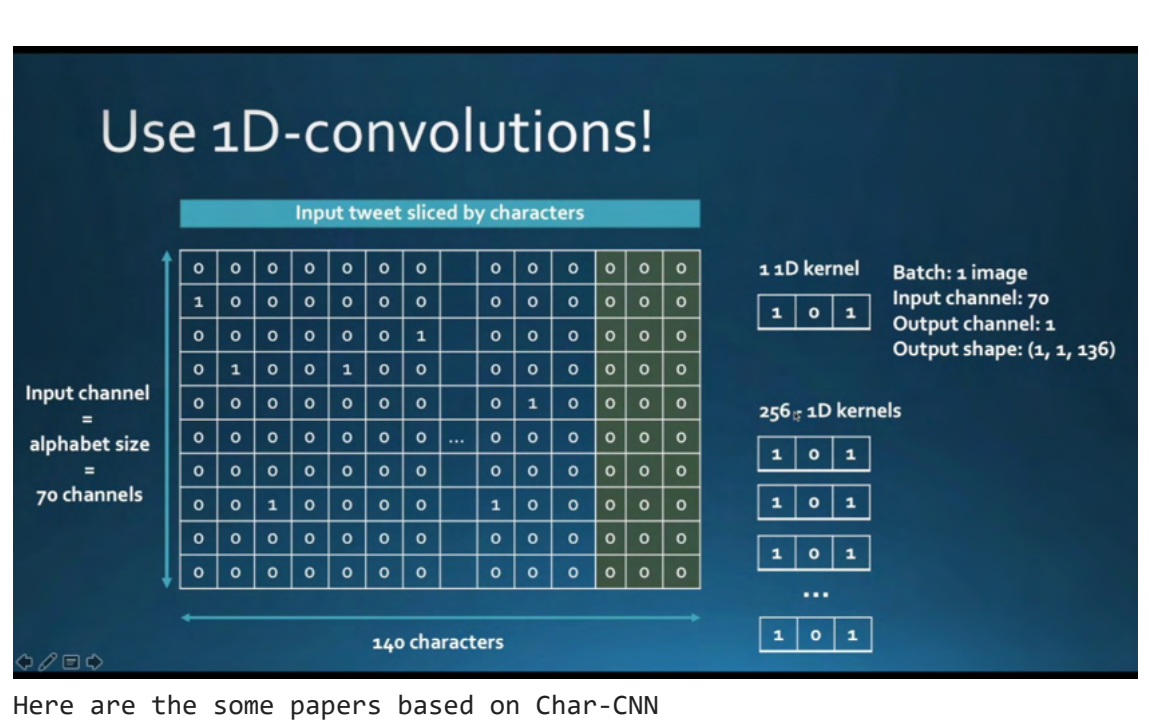


Ref: https://i.imgur.com/kiVQuk1.png

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

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/

Model-2 : Using 1D convolutions with character embedding



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

```python
from tqdm import tqdm
import pandas as pd
import numpy as np
from bs4 import BeautifulSoup
import re
import os
import nltk
```

```python
files=os.listdir('documents/')
files
```

```python
labels=[]
for i in tqdm(os.listdir('documents/')):
    labels.append(i.split('_')[0])
```

```python
def decontracted(phrase):
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase

def preprocess(input_path):
    files=os.listdir(path=input_path)
    preprocessed_emails=[]
    preprocessed_subjects=[]
    preprocessed_text=[]
    for x in tqdm(files):
        text=open(input_path+x,'r').read()
        e=re.findall(r"[a-z0-9\.\-+_]+@[a-z0-9\.\-+_]+\.[a-z]+",text)
        pre=[]
        for i in e:
            z=i.split('@')[1].split('.')
            for j in z:
                if len(j)>=2:
                    pre.append(j)
        preprocessed_emails.append(' '.join(pre))
        preprocessed_subjects.append(text.split('Subject:')[1].split('\n')[0])

        text1=open(input_path+x,'r')
        pre_text=''
        for sentence in text1.readlines():
            if not (sentence.startswith('Write to:') or
                    sentence.startswith('From:') or
                    sentence.startswith('Subject:')):
                pre_text+=sentence
        pre_text=decontracted(pre_text)
        pre_text=re.sub("<.*?>"," ",pre_text)
        pre_text=re.sub("\n"," ",pre_text)
        pre_text=re.sub("\t"," ",pre_text)
        pre_text=re.sub("-"," ",pre_text)
        pre_text=re.sub("/"," ",pre_text)
        pre_text=re.sub(">"," ",pre_text)
        pre_text=re.sub("\'"," ",pre_text)
        pre_text=re.sub("\d+","",pre_text)
        pre_text=[i for i in pre_text.split('_') if len(i)>2]
        pre_text=' '.join(x for x in pre_text)
        pre_text=[word for word in pre_text.split() if not word.endswith(':')]
        pre_text=' '.join(x.lower() for x in pre_text)
        pre_text=re.sub("[^A-Za-z_]+"," ",pre_text)
        pre_text=re.sub("\(.*\)","",pre_text)
        #pre_text=[word for word in pre_text.split(' ') if (len(word)<15 or len(word)>2)]
        #pre_text=' '.join(x for x in pre_text)
        pre_text=re.sub(r'\b\w{1,2}\b','',pre_text)


        preprocessed_text.append(pre_text)

    return preprocessed_emails,preprocessed_subjects,preprocessed_text
```

```python
path='documents/'

preprocessed=preprocess(path)
```

```python
preprocessed_emails=preprocessed[0]
preprocessed_subjects=preprocessed[1]
preprocessed_text=preprocessed[2]
text=[open(path+i,'r').read() for i in files]
```

```python
open('documents/alt.atheism_49960.txt','r').read()
```

```python
preprocessed_text[0]
```

```python
data={'text':text,
      'preprocessed_emails':preprocessed_emails,
      'preprocessed_subjects':preprocessed_subjects,
      'preprocessed_text':preprocessed_text,
      'class':labels}

df=pd.DataFrame(data)
```

```python
df.to_csv('doc_CNN.csv',index=False)
```

```python
df=pd.read_csv('doc_CNN.csv').dropna()
df.reset_index(drop=True,inplace=True)
df
```

```python
from tqdm import tqdm
def chunk(string):
    parse_tree=nltk.ne_chunk(nltk.pos_tag(nltk.word_tokenize(string)))
    person_entities = []
    location_entities=[]

    for t in parse_tree.subtrees():
        if t.label()=='PERSON' :
            person_entities.append(t)
        if t.label()=='GPE':
            location_entities.append(t)

    people=[" ".join(w.lower() for w, t in elt) for elt in person_entities if isinstance(elt, nltk.Tree)]
    location=[" ".join(w.lower() for w, t in elt) for elt in location_entities if isinstance(elt, nltk.Tree)]
    location_re=["_".join(w.lower() for w, t in elt) for elt in location_entities if isinstance(elt, nltk.Tree)

    return people,location,location_re

for string in tqdm(range(len(df['preprocessed_text']))):
    chunks=chunk(df['text'][string])
    people=chunks[0]
    location=chunks[1]
    location_sub=chunks[2]
    for i in people:
        df['preprocessed_text'][string]=df['preprocessed_text'][string].replace(i,"")

    for (i,j) in zip(location,location_sub):
        df['preprocessed_text'][string]=df['preprocessed_text'][string].replace(i,j)
```

```python
df.to_csv('new_doc_CNN.csv',index=False)
```

```python
df2=pd.read_csv('/content/drive/MyDrive/Document_Class_CNN/new_doc_CNN.csv')
df2
```

```python
df2['preprocessed_text'][0]
```

```python
df2['data'] = df2[df2.columns[1:4]].apply(
    lambda x: ','.join(x.dropna().astype(str)),
    axis=1
)


from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
le.fit(df2['class'])
finallabels=le.transform(df2['class'])
len(finallabels)
```

```python
from sklearn.model_selection import train_test_split
X=df2['data']
Y=finallabels
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.25,stratify=Y)
```

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer,text_to_word_sequence
from tensorflow.keras.callbacks import ReduceLROnPlateau,TensorBoard,EarlyStopping,ModelCheckpoint
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.preprocessing import sequence
from sklearn.metrics import f1_score
from datetime import datetime
import warnings
warnings.filterwarnings("ignore")
```

```python
tokenizer=Tokenizer(filters='!"#$%&()*+,-./:;<=>?@[\\]^`{|}~\t\n')
num_classes=20
max_words = 5000

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(df2['data'])
x_train = tokenizer.texts_to_sequences(x_train)
x_test = tokenizer.texts_to_sequences(x_test)

max_length=1000
x_train_final=sequence.pad_sequences(x_train,maxlen=max_length)
x_test_final=sequence.pad_sequences(x_test,maxlen=max_length)

x_train_final.shape
```

```python
word_index=tokenizer.word_index
len(word_index)
```

```python
#https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html
embeddings_index = {}
f = open('/content/drive/MyDrive/Document_Class_CNN/glove.6B.300d.txt',encoding='utf8')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

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


embedding_matrix = np.zeros((len(word_index)+1, 300))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

embedding_matrix
```

Model1

```python
model1=Sequential()
model1.add(keras.layers.Embedding(len(word_index)+1,
                                  300,
                                  input_length=max_length,
                                  weights=[embedding_matrix],
                                  trainable=False))


I=layers.Input(model1.output.shape[1:])
c1=layers.Conv1D(32,5,activation='relu',kernel_initializer='he_uniform')(I)
c2=layers.Conv1D(32,4,activation='relu',kernel_initializer='he_uniform')(I)
c3=layers.Conv1D(32,3,activation='relu',kernel_initializer='he_uniform')(I)

concat=layers.concatenate([c1,c2,c3],axis=1)

mp=layers.MaxPooling1D(3)(concat)
c4=layers.Conv1D(64,5,activation='relu',kernel_initializer='he_uniform')(mp)
c5=layers.Conv1D(64,4,activation='relu',kernel_initializer='he_uniform')(mp)
c6=layers.Conv1D(64,3,activation='relu',kernel_initializer='he_uniform')(mp)

concat2=layers.concatenate([c4,c5,c6],axis=1)

mp2=layers.MaxPooling1D(3)(concat2)
c7=layers.Conv1D(128,3,activation='relu',kernel_initializer='he_uniform')(mp2)
fl=layers.Flatten()(c7)
dr=layers.Dropout(0.4)(fl)
d=layers.Dense(64,activation='relu',kernel_initializer='he_uniform')(dr)
pred=layers.Dense(20,activation='softmax')(d)

model1.add(keras.Model(I,pred))
model1.summary()


model1.compile(optimizer=keras.optimizers.Adam(lr=0.001),
               loss='sparse_categorical_crossentropy',metrics=['acc'])
```

```python
tf.keras.utils.plot_model(
    model1, to_file='model1.png', show_shapes=False, show_dtype=False,
    show_layer_names=True, rankdir='TB', expand_nested=True, dpi=96
)
```

```python
class PerformanceEval(keras.callbacks.Callback):
  def on_epoch_end(self,epoch,logs={}):
    y_true=keras.utils.to_categorical(y_test,num_classes=20)
    y_pred=self.model.predict(x_test_final).round()
    print(' f1 score: ',f1_score(y_true,y_pred,average='micro'))


early_stopping=EarlyStopping(monitor='val_acc',patience=3,verbose=1,mode='max')

log_dir='/content/drive/MyDrive/Document_Class_CNN/Model1/'+datetime.now().strftime("%Y%m%d-%H%M%S")

tensorboard_callback=TensorBoard(log_dir=log_dir,
                                 histogram_freq=1,
                                 write_graph=True,
                                 write_images=True)


model_dir="/content/drive/MyDrive/Document_Class_CNN/Model1_best/best_model_1.h5"

model1_save=ModelCheckpoint(model_dir,monitor='val_acc',save_best_only='True',mode='max')


model1.fit(x_train_final,y_train,
           epochs=10,
           batch_size=6,
          validation_data=(x_test_final,y_test),
          callbacks=[early_stopping,
                     tensorboard_callback,
                     PerformanceEval(),
                     model1_save])
```

```python
%load_ext tensorboard
%tensorboard --logdir /content/drive/MyDrive/Document_Class_CNN/Model1/20210217-054351/
```

```python
model1_best=keras.models.load_model(model_dir)
y_pred=model1_best.predict(x_test_final,y_test.any()).round()

y_true=keras.utils.to_categorical(y_test,num_classes=20)

print(' model1 best f1 score: ',f1_score(y_true,y_pred,average='micro'))
```

```python
#https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html
embeddings_index = {}
f = open('/content/drive/MyDrive/Document_Class_CNN/glove.840B.300d-char2.txt',encoding='utf8')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

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



embedding_matrix = np.zeros((len(word_index)+1, 300))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
embedding_matrix.shape
```

Model2

```python
model2=Sequential()
model2.add(keras.layers.Embedding(len(word_index)+1,
                                  300,
                                  input_length=max_length,
                                  weights=[embedding_matrix],
                                  trainable=False))

I=layers.Input(model2.output.shape[1:])
c1=layers.Conv1D(32,5,activation='relu',kernel_initializer='he_uniform')(I)
c1=layers.Conv1D(32,4,activation='relu',kernel_initializer='he_uniform')(c1)
m=layers.MaxPooling1D(4)(c1)
c1=layers.Conv1D(64,5,activation='relu',kernel_initializer='he_uniform')(m)
c1=layers.Conv1D(64,4,activation='relu',kernel_initializer='he_uniform')(c1)
m=layers.MaxPooling1D(4)(c1)
f=layers.Flatten()(m)
d=layers.Dropout(0.4)(f)
d=layers.Dense(128,activation='relu',kernel_initializer='he_uniform')(d)
out=layers.Dense(20,activation='softmax')(d)

model2.add(keras.Model(I,out))

model2.compile('adam',loss='sparse_categorical_crossentropy',metrics=['acc'])

model2.summary()
```

```python
tf.keras.utils.plot_model(
    model2, to_file='model2.png', show_shapes=False, show_dtype=False,
    show_layer_names=True, rankdir='TB', expand_nested=True, dpi=96
)
```

```python
class PerformanceEval(keras.callbacks.Callback):
  def on_epoch_end(self,epoch,logs={}):
    y_true=keras.utils.to_categorical(y_test,num_classes=20)
    y_pred=self.model.predict(x_test_final).round()
    print(' f1 score: ',f1_score(y_true,y_pred,average='micro'))


log_dir='/content/drive/MyDrive/Document_Class_CNN/Model2/'+datetime.now().strftime("%Y%m%d-%H%M%S")

tensorboard_callback=TensorBoard(log_dir=log_dir,
                                 histogram_freq=1,
                                 write_graph=True,
                                 write_images=True)


model_dir="/content/drive/MyDrive/Document_Class_CNN/Model2_best/best_model_2.h5"

model2_save=ModelCheckpoint(model_dir,monitor='val_acc',save_best_only='True',mode='max')


model2.fit(x_train_final,y_train,
           epochs=10,
          validation_data=(x_test_final,y_test),
          callbacks=[tensorboard_callback,
                     PerformanceEval(),
                     model2_save])
```

```python
%load_ext tensorboard
%tensorboard --logdir /content/drive/MyDrive/Document_Class_CNN/Model2/20210218-091833/
```