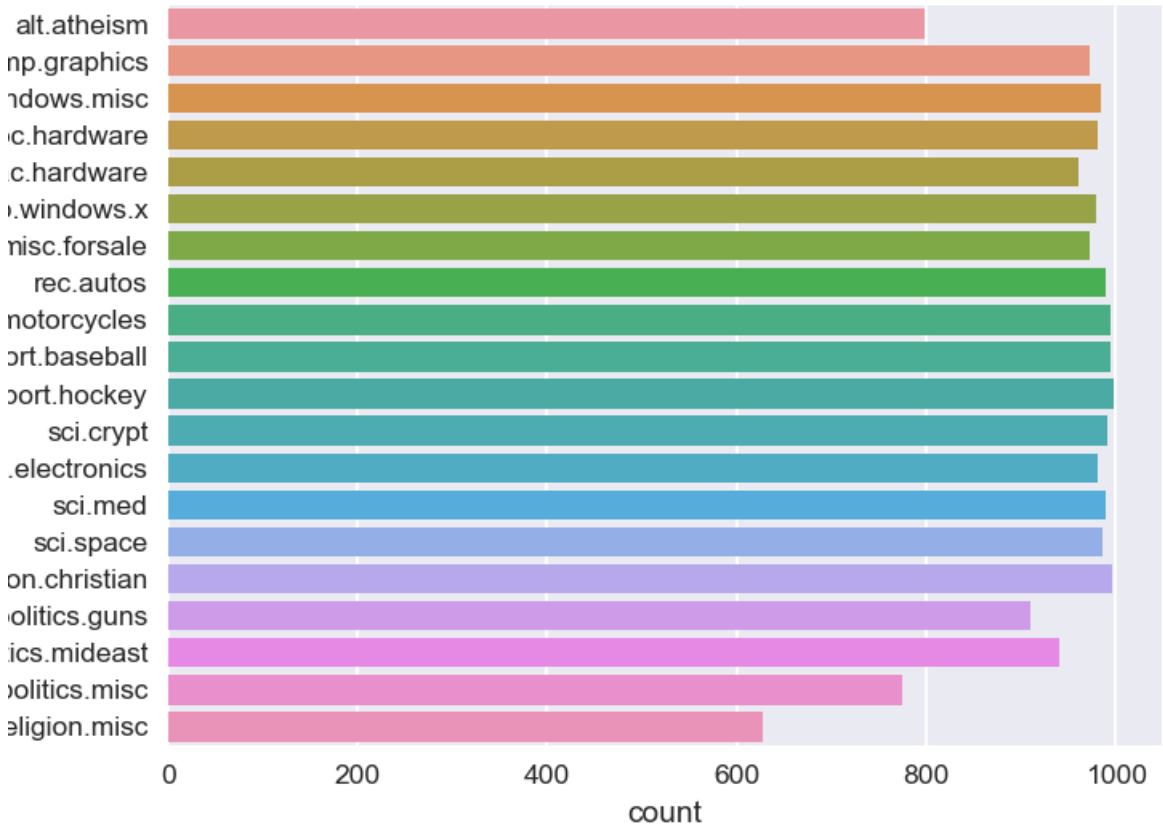
Text Classification:

Data

1. we have total of 20 types of documents(Text files) and total 18828 documents(text fil
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'Cl
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.

```
1 ### 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 t
after that remove the words whose length is less than or equal to 2 and also remove'com'
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,c
append all those into one list/array. (This will give length of 18828 sentences i.e one
Some sample output was shown below.

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

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

Eg: if we have sentance like "Subject: Re: Gospel Dating @ \r\r\n" --> You have to get "
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.

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-T

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

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 -->

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

```
#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

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

My name is Srikanth Varma --> [('My', 'PRP$'), ('name', 'NN'), ('is', 'VBZ'), Tree('P
```

We did chunking for above two lines and then We got one list where each word is mapped t
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 "Sr
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
"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.

Data collection

```
!wget --header="Host: doc-0c-2k-docs.googleusercontent.com" --header="User-Agent: Mozil
```

```
--2021-10-27 05:17:38--  https://doc-0c-2k-docs.googleusercontent.com/docs/securesc/t
Resolving doc-0c-2k-docs.googleusercontent.com (doc-0c-2k-docs.googleusercontent.com)
Connecting to doc-0c-2k-docs.googleusercontent.com (doc-0c-2k-docs.googleusercontent
HTTP request sent, awaiting response... 200 OK
Length: 19038123 (18M) [application/rar]
Saving to: ‘documents.rar’

documents.rar       100%[===================>]  18.16M  85.3MB/s    in 0.2s

2021-10-27 05:17:39 (85.3 MB/s) - ‘documents.rar’ saved [19038123/19038123]
```

```
!pip install unrar
!unrar x "documents.rar"
```

Email and subject preprocessing

```
def preprocess_email_subject(data):
  preprocessed_email = []
  preprocessed_subject = []
  preprocessed_data = []

  match = re.findall(r'[a-zA-Z0-9\.\-+_]+@[a-zA-Z0-9\.\-+_]+\.[a-z]+', data)
  for mail in match:
    mail = mail.split('@')[1]
    mail = mail.split('.')
    for val in mail:
      if len(val)>2 and val not in 'com':
        preprocessed_email.append(val)

  preprocessed_subject = []
  subject = re.findall(r'Subject:.+', data)[0].split(': ')[-1]
  subject = re.sub('[^a-zA-Z ]+', '', subject)
  preprocessed_subject.append(subject)

  data = re.sub(r'[a-zA-Z0-9\.\-+_]+@[a-zA-Z0-9\.\-+_]+\.[a-z]+', '', data)  # detectin
  data = re.sub(r'Subject:.+', '', data) # detecting and replacing 'Subject:' with blan
  data = re.sub(r'Write to:.+', '', data) # detecting and replacing 'Write to:' with bl
  data = re.sub(r'From:.+', '', data)  # detecting and replacing 'From:' with blank
  data = re.sub(r'<.+?>', ' ', data)  # detecting and replacing anything in between <>
  data = re.sub(r'\([^()]*\)', '', data) # detecting and replacing anything in between

  data = re.sub(r'[\n\r\t\v\f\-\\]', ' ', data)
```

```
    data = re.sub(r'[-\.+]', '', data) # replacing - with blank
    data = re.sub(r'\w+:\s?', '', data) # replacing words starting with colon

    data = re.sub(r"can't", 'can not', data, flags=re.IGNORECASE)
    data = re.sub(r"'s", ' is', data)
    data = re.sub(r"i've", 'I have', data, flags=re.IGNORECASE)
    data = re.sub(r"i'm", 'I am', data, flags=re.IGNORECASE)
    data = re.sub(r"you're", 'you are', data)
    data = re.sub(r"i'll", 'I will', data, flags=re.IGNORECASE)

    preprocessed_data.append(data)

    return preprocessed_email, preprocessed_subject, preprocessed_data
```

```
from tqdm import tqdm
import os
import re

files = os.listdir('documents')

final_emails = []
final_subjects = []
final_data = []

for file in tqdm(files):
  with open('/content/documents/'+str(file), 'r+', encoding='ISO-8859-1') as f:
    emails = []
    subjects = []
    data = []

    file_as_str = f.read()
    em, sub, text = preprocess_email_subject(file_as_str)
    emails.append(em)
    subjects.append(sub)
    data.append(text)

  final_emails.append(emails)
  final_subjects.append(subjects)
  final_data.append(data)
```

```
100%|██████████| 18828/18828 [00:19<00:00, 961.71it/s]
```

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 c
Eg: if we have sentance like "Subject: Re: Gospel Dating @ \r\r\n" --> You have to get "
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.

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-T

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

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 -->

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

NLTK packages

```
import nltk

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

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping chunkers/maxent_ne_chunker.zip.
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Unzipping corpora/words.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

Chunking

```
from tqdm import tqdm

files = os.listdir('documents')

final_chunks = []
count=0
for file in tqdm(final_data):
  gpe = []
  person = []
  file_as_str = ''.join(str(file))

  parse_tree = nltk.ne_chunk(nltk.tag.pos_tag(file_as_str.split()))

  for t in parse_tree.subtrees():
    if t.label() == 'GPE': # if it's a place, e.g. Nueva York, store it in a special li
        gpe.append(t)
    if t.label() == 'PERSON':
        person.append(t) # if it's a person, e.g. Pablo Escobar, store it in a special
  gpe = [ " ".join(w for w, t in elt) for elt in gpe if isinstance(elt, nltk.Tree)] # g
  person = [ " ".join(w for w, t in elt) for elt in person if isinstance(elt, nltk.Tree

  gpe = ''.join(gpe)
  person = ''.join(person)

  gpe = re.sub('[^A-Za-z0-9 ]+', '', gpe) # if any character is not a word/digit/space,
  person = re.sub('[^A-Za-z0-9 ]+', '', person) # if any character is not a word/digit/

  replaced_gpe = re.sub(r' ', '_', gpe) # converting Nueva York to Nueva_York (e.g.)
```

```
  new_string = re.sub(gpe, replaced_gpe, file_as_str) # replacing Nueva York with Nueva
  new_string = re.sub(person, '', new_string) # replacing Pablo Escobar with blank (e.g

  final_chunks.append(new_string)
```

```
100%|██████████| 18828/18828 [27:03<00:00, 11.60it/s]
```

```
def final_cleaned_text():
  """FINALLYYYY"""
  cleaned_it_further = []

  for data in tqdm(final_chunks):

    data = re.sub(r'[0-9]+', '', str(data)) # removing all digits

    pre_underscore = [t for t in data.split() if t.startswith('_') and t.isalpha() == F
    post_underscore = [t for t in data.split() if t.endswith('_') and t.isalpha() == Fa
    clean_pre = [x.replace('_', '') for x in pre_underscore] # replacing words that sta
    clean_post = [y.replace('_', '') for y in post_underscore] # replacing words that e

    for i, j in zip(pre_underscore, clean_pre):
      split = data.split(' ')
      for word in split:
        if word == i:
          text = data.replace(i, j)
        else: None

    for a, b in zip(post_underscore, clean_post):
      data = text.replace(a, b)

    data = re.sub(r'\b\w{1,2}\b', '', data) # removing words less than or equal to 2
    data = re.sub(r'[^A-Za-z_]', ' ', data) # removing all special characters except _
    colon_finder = [i for i in data.split() if i.endswith(':')] # extracting words that

    for word_with_colon in colon_finder:
      data = data.replace(word_with_colon, '') # replacing words that end with ':' with
    data = re.sub('[^A-Za-z0-9_ ]+', '', data)

    data = re.sub('  +', ' ', data) # removing extra spacesdata
    data = data.lower()

    cleaned_it_further.append(data)

  return cleaned_it_further
```

```
final_preprocessed_text = final_cleaned_text()
```

```
100%|██████████| 18828/18828 [00:08<00:00, 2312.87it/s]
```

```
final_text = final_preprocessed_text
```

```
# with open('/content/mi hermano pablo.txt', 'wb') as f:
#    pickle.dump(final_preprocessed_text, f)
```

Run this for evalutaion

```
for index, value in enumerate(final_chunks):
  if 'schizophrenic' in value:
    print('\nINDEX: ', index)
    print('|nVALUE: ', value)
# it's 17480
```

```
!pip install rich
```

```
from rich.pretty import Pretty
from rich.panel import Panel
from rich import print
from rich.pretty import pprint

print(Panel('[italic red]Raw text:'))
pprint(str(final_data[17480]), expand_all=True)
print('-'*70)
print(Panel('[italic blue]Preprocessed_text:'))
pprint(final_preprocessed_text[17480])
```

```
╭──────────────────────────────────────────────────────────────────────────────╮
│ Raw text:                                                                    │
╰──────────────────────────────────────────────────────────────────────────────╯

'[[\'   Archive atheism/resources Alt atheism archive resources Last 11 December 1992

----------------------------------------------------------------------
╭──────────────────────────────────────────────────────────────────────────────╮
│ Preprocessed_text:                                                           │
```

Converting to a dataframe

```
from functools import reduce

final_emails = reduce(lambda x,y: x+y, final_emails)
final_subjects = reduce(lambda x,y: x+y, final_subjects)
```

```
def list_to_string(s):
    str1 = " "
    return (str1.join(s))

preprocessed_emails = []
preprocessed_subjects = []
```

```
for email in final_emails:
  preprocessed_emails.append(list_to_string(email))

for sub in final_subjects:
  preprocessed_subjects.append(list_to_string(sub))
```

```
print(len(preprocessed_emails))
print(len(preprocessed_subjects))
print(len(final_text))
```

```
18828
18828
18828
```

```
import pandas as pd

data = pd.DataFrame()

data['emails'] = preprocessed_emails
data['subjects'] = preprocessed_subjects
data['text'] = final_text

data['combined'] = data['emails'].astype('str') + ' ' + data['subjects'].astype('str')
```

```
# data.to_csv('final.csv')
```

```
file = os.listdir('documents')
y = []

for file in tqdm(files):
  name  = file.split('_')[0]
  y.append(str(name))
```

```
100%|██████████| 18828/18828 [00:00<00:00, 966435.64it/s]
```

```
from tensorflow.keras.layers import Embedding
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense, Input, Activation, Conv1D, Concatenate, MaxP
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import datetime
from tensorflow.keras.layers import TextVectorization
from sklearn.model_selection import train_test_split


X = data['combined']
y = y
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=
```

```
print('Max length of a string: ', X.map(len).max())
print('Min length of a string: ', X.map(len).min())
```

```
3 print('Mean length of a string: ', X.map(len).mean())
```

```
Max length of a string:  54105
Min length of a string:  9
Mean length of a string:  1183.7941895049926
```

```
1 from nltk.corpus import stopwords
2 stop = stopwords.words('english')
3
4 X = X.apply(lambda words: ' '.join(word.lower() for word in words.split() if word not i
```

```
1 from collections import Counter
2 Counter(" ".join(X).split()).most_common(250)
```

```
[('edu', 24633),
 ('would', 14297),
 ('one', 13643),
 ('article', 11382),
 ('people', 9153),
 ('like', 8850),
 ('know', 8089),
 ('get', 7805),
 ('think', 7005),
 ('also', 6709),
 ('use', 6241),
 ('time', 6165),
 ('could', 5652),
 ('good', 5620),
 ('new', 5456),
 ('well', 5400),
 ('even', 4960),
 ('may', 4872),
 ('god', 4808),
 ('way', 4721),
 ('two', 4656),
 ('make', 4642),
 ('see', 4632),
 ('much', 4557),
 ('many', 4485),
 ('first', 4461),
 ('right', 4314),
 ('say', 4200),
 ('anyone', 4098),
 ('want', 3994),
 ('system', 3956),
 ('need', 3776),
 ('used', 3761),
 ('said', 3734),
 ('work', 3504),
 ('really', 3422),
 ('problem', 3402),
 ('something', 3338),
 ('max', 3296),
 ('since', 3276),
 ('believe', 3272),
 ('please', 3265),
 ('back', 3247),
 ('going', 3165),
 ('still', 3107),
```

```
('find', 3010),
('windows', 2978),
('take', 2969),
('help', 2909),
('government', 2898),
('point', 2892),
('might', 2852),
('years', 2838),
('using', 2835),
('thanks', 2830),
('never', 2814),
('year', 2812),
('information', 2772),
('things', 2743),
```

Glove Vectors

```
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip -q glove.6B.zip
```

create folders: model_save and model_images

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.

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 c

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 mod

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 acceptab

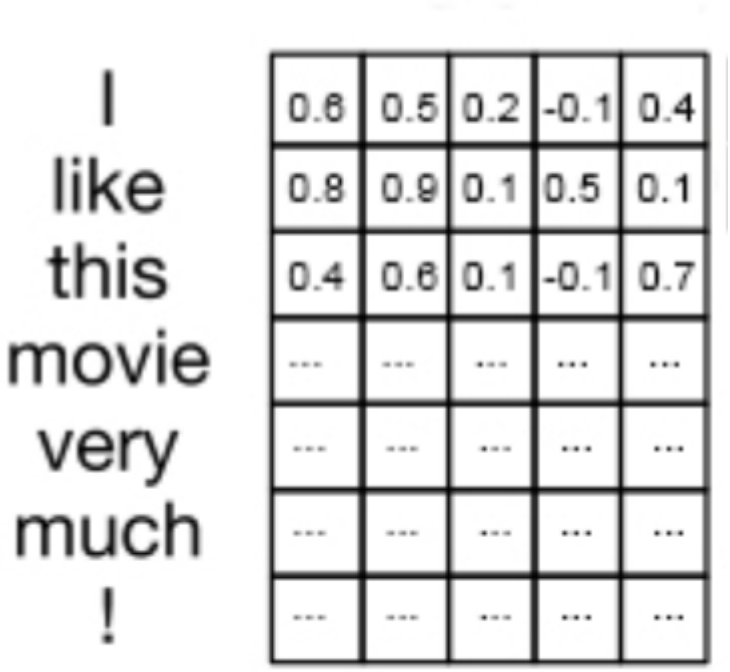
13. Try to use **Early Stopping** technique or any of the callback techniques that you did i

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
In the example we have considered d = 5, but in this assignment we will get d = dimensio
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 lay



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-network

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/

Embedding and Vectorization for M1

This code has been entirely referenced from here:
https://keras.io/examples/nlp/pretrained_word_embeddings/

```
from tensorflow.keras.layers import TextVectorization
import tensorflow as tf

vectorizer = TextVectorization(max_tokens=12000, output_sequence_length=1500)
# After experimenting numerous times with the data, I've found 12000 and 1500 to be the
text_ds = tf.data.Dataset.from_tensor_slices(X_train).batch(128)
vectorizer.adapt(text_ds)
```

```
voc = vectorizer.get_vocabulary()
word_index = dict(zip(voc, range(len(voc))))
```

```
import numpy as np


path_to_glove_file = os.path.join(
    os.path.expanduser("~"), "/content/glove.6B.300d.txt"
)

embeddings_index = {}
with open(path_to_glove_file) as f:
    for line in tqdm(f):
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep=" ")
        embeddings_index[word] = coefs

print("\nFound %s word vectors." % len(embeddings_index))
# source: https://keras.io/examples/nlp/pretrained_word_embeddings/
```

```
400000it [00:19, 20014.69it/s]
Found 400000 word vectors.
```

```
num_tokens = len(word_index) + 2 #should I use the padded data or word_index
embedding_dim = 300
hits = 0
misses = 0

# Prepare embedding matrix
embedding_matrix = np.zeros((num_tokens, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # Words not found in embedding index will be all-zeros.
        # This includes the representation for "padding" and "OOV"
        embedding_matrix[i] = embedding_vector
        hits += 1
    else:
        misses += 1
print("\nConverted %d words (%d misses)" % (hits, misses))
# source: https://keras.io/examples/nlp/pretrained_word_embeddings/
```

```
Converted 11440 words (560 misses)
```

```
from tensorflow.keras.layers import Embedding
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense, Input, Activation, Conv1D, Concatenate, MaxP
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import datetime
import keras

```

```
embedding_layer = Embedding(
                            input_dim=num_tokens,
                            output_dim=embedding_dim,
                            embeddings_initializer=keras.initializers.Constant(embeddin
                            trainable=False
                            )
```

```
from tensorflow.keras.utils import to_categorical

X_train = vectorizer(np.array([[s] for s in X_train])).numpy()
X_test = vectorizer(np.array([[s] for s in X_test])).numpy()

y_train = pd.get_dummies(y_train).to_numpy() # using get_dummies instead of to_categori
y_test = pd.get_dummies(y_test).to_numpy()
```

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

```
(14121, 1500)
(14121, 20)
(4707, 1500)
(4707, 20)
```

Final Model 1

```
input_layer = keras.Input(shape=(1500, ))
embedded_sequences = embedding_layer(input_layer)

conv_layer1 = Conv1D(128, 5, activation="relu", padding='same')(embedded_sequences)
conv_layer2 = Conv1D(256, 5, activation="relu", padding='same')(embedded_sequences)
conv_layer3 = Conv1D(512, 5, activation="relu", padding='same')(embedded_sequences)

concatted_layer1 = Concatenate()([conv_layer1, conv_layer2, conv_layer3])

maxpool_layer1 = MaxPooling1D(5)(concatted_layer1)

conv_layer4 = Conv1D(128, 7, activation="relu", padding='same')(maxpool_layer1)
conv_layer5 = Conv1D(256, 7, activation="relu", padding='same')(maxpool_layer1)
conv_layer6 = Conv1D(512, 7, activation="relu", padding='same')(maxpool_layer1)

concatted_layer2 = Concatenate()([conv_layer4, conv_layer5, conv_layer6])

maxpool_layer2 = MaxPooling1D(7)(concatted_layer2)
conv_layer7 = Conv1D(128, 9, activation="relu", padding='same')(maxpool_layer2)

flatten = tf.keras.layers.Flatten()(conv_layer7)
flatten2 = tf.keras.layers.Flatten()(flatten)

dropout = Dropout(rate=0.2)(flatten2)
```

```
dense = Dense(512, activation="relu")(dropout)
dense2 = Dense(256, activation="relu")(dense) #added another dense layer here with size

filepath = "model_save/weights-{epoch:02d}-{val_acc:.4f}.hdf5"
earlystop = EarlyStopping(monitor='val_acc', patience=3, verbose=1, mode='max')
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_acc', verbose=1, save_best

log_dir="logs/fit/model_1/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1

output = Dense(20, activation='softmax', kernel_initializer=tf.keras.initializers.HeUni
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoa
```

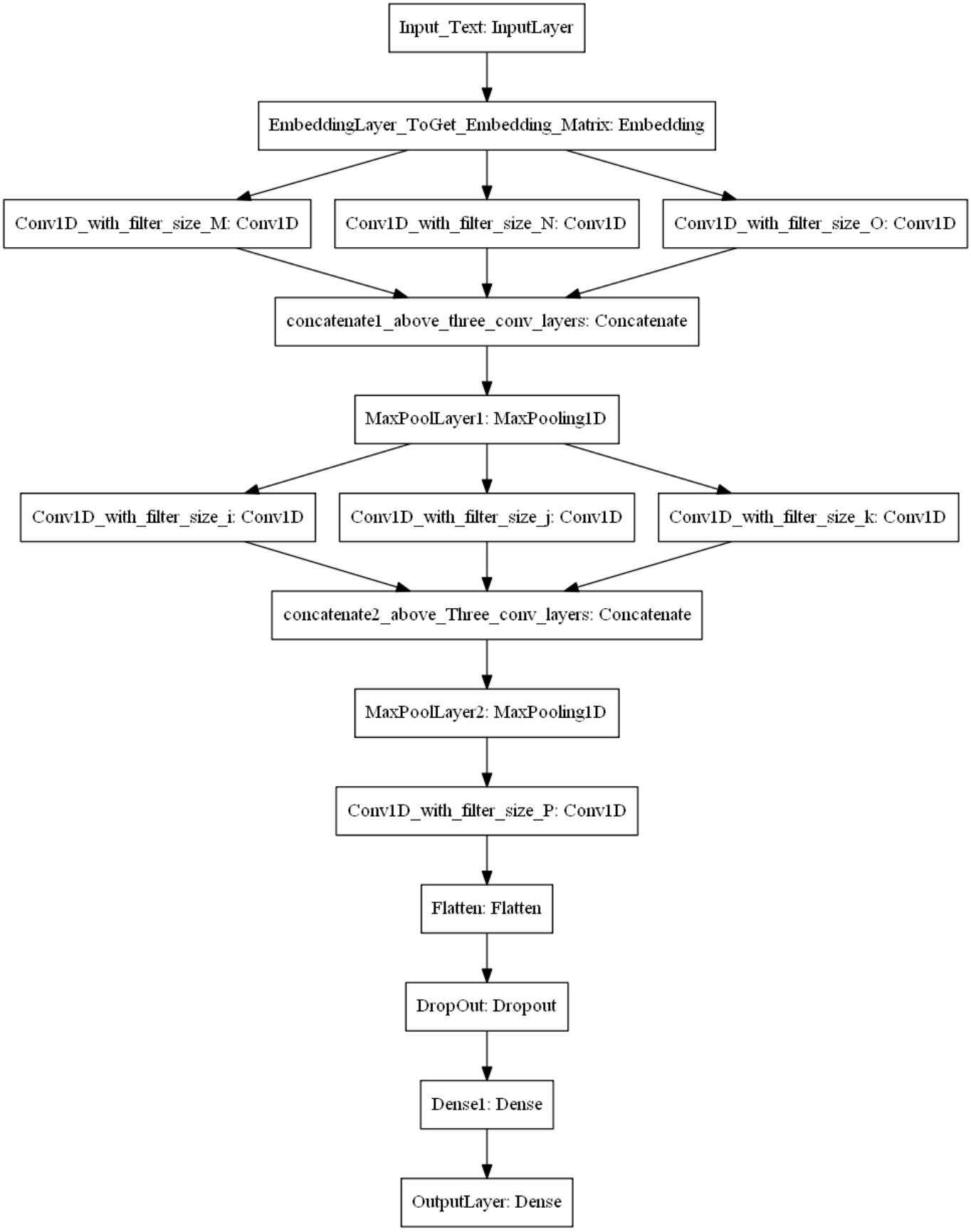
```
!pip install tensorflow_addons
```

```
Collecting tensorflow_addons
  Downloading tensorflow_addons-0.14.0-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010
     |████████████████████████████████| 1.1 MB 5.2 MB/s
Requirement already satisfied: typeguard>=2.7 in /usr/local/lib/python3.7/dist-packag
Installing collected packages: tensorflow-addons
Successfully installed tensorflow-addons-0.14.0
```

```
import tensorflow_addons as tfa

model = tf.keras.Model(inputs=input_layer, outputs=output)
dot_img_file = '/content/model_images/model_1.png'
tf.keras.utils.plot_model(model, to_file=dot_img_file, show_shapes=True)

micro_f1 = tfa.metrics.F1Score(num_classes=20, average='micro')
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["acc", micro_
model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test), callbacks=[che
```

```
Epoch 1/10
442/442 [==============================] - 212s 405ms/step - loss: 2.1048 - acc: 0.29

Epoch 00001: val_acc improved from -inf to 0.61653, saving model to model_save/weight
Epoch 2/10
442/442 [==============================] - 179s 405ms/step - loss: 0.9555 - acc: 0.66

Epoch 00002: val_acc improved from 0.61653 to 0.71744, saving model to model_save/wei
Epoch 3/10
442/442 [==============================] - 179s 405ms/step - loss: 0.5568 - acc: 0.81

Epoch 00003: val_acc improved from 0.71744 to 0.80051, saving model to model_save/wei
Epoch 4/10
442/442 [==============================] - 176s 399ms/step - loss: 0.3310 - acc: 0.89

Epoch 00004: val_acc improved from 0.80051 to 0.82515, saving model to model_save/wei
Epoch 5/10
442/442 [==============================] - 179s 404ms/step - loss: 0.2455 - acc: 0.92

Epoch 00005: val_acc did not improve from 0.82515
```

```
Epoch 6/10
442/442 [==============================] - 179s 404ms/step - loss: 0.1906 - acc: 0.94

Epoch 00006: val_acc did not improve from 0.82515
Epoch 7/10
442/442 [==============================] - 179s 404ms/step - loss: 0.1361 - acc: 0.96

Epoch 00007: val_acc improved from 0.82515 to 0.83854, saving model to model_save/wei
Epoch 8/10
442/442 [==============================] - 179s 404ms/step - loss: 0.1004 - acc: 0.97

Epoch 00008: val_acc did not improve from 0.83854
Epoch 9/10
442/442 [==============================] - 179s 404ms/step - loss: 0.0693 - acc: 0.98

Epoch 00009: val_acc did not improve from 0.83854
Epoch 10/10
442/442 [==============================] - 176s 399ms/step - loss: 0.0977 - acc: 0.97

Epoch 00010: val_acc did not improve from 0.83854
Epoch 00010: early stopping
<keras.callbacks.History at 0x7f1d4c8adc90>
```



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

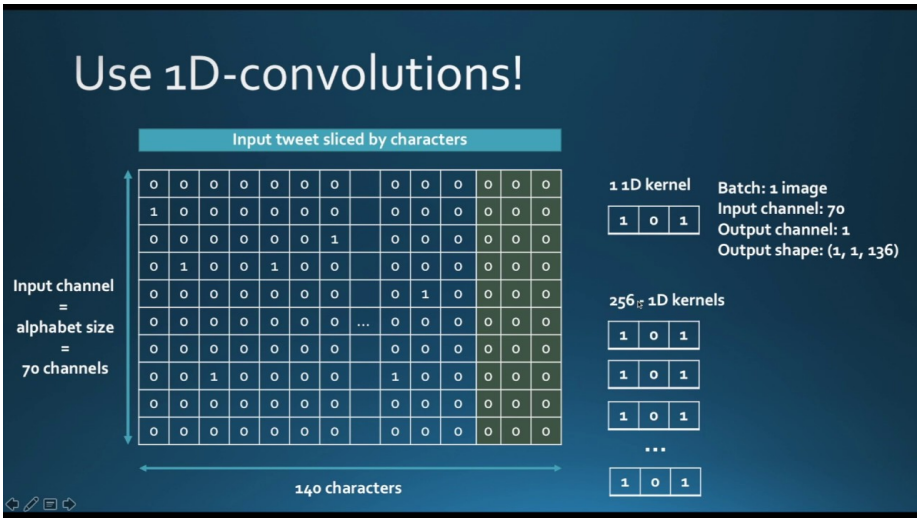
1. all are Conv1D layers with any number of filter and filter sizes, there is no restric
2. use concatenate layer is to concatenate all the filters/channels.

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 p
(Only recommendation if you have less computing power)

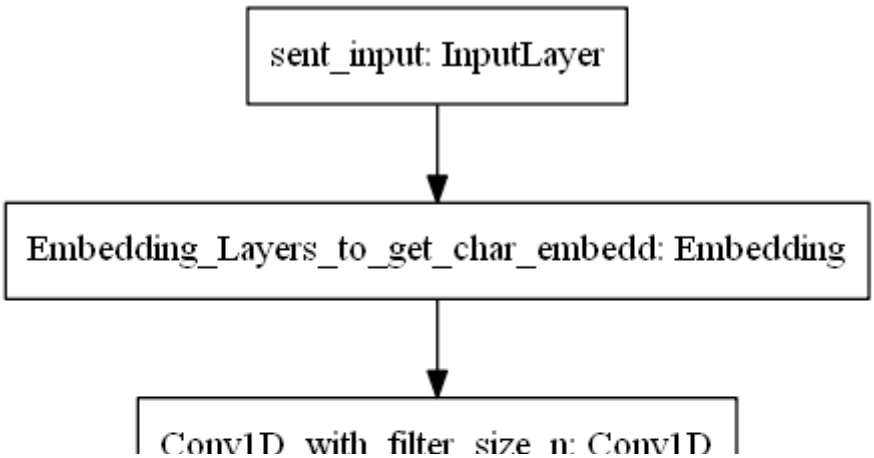
5. You can use any number of layers after the Flatten Layer.

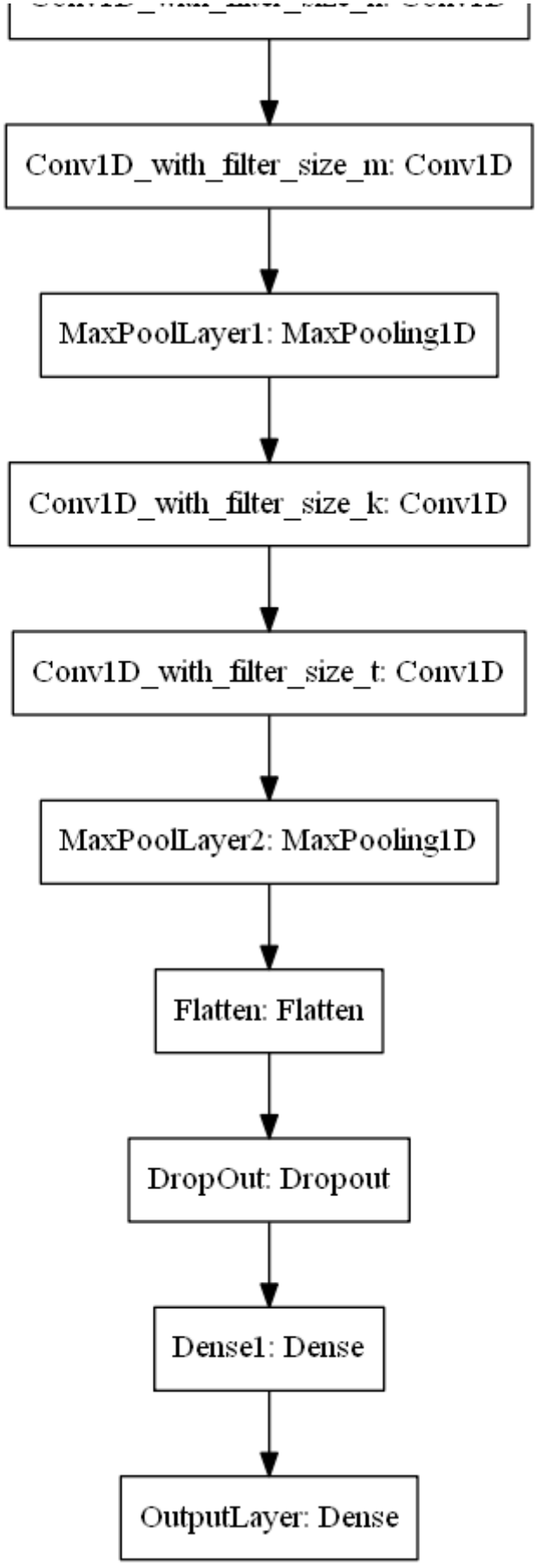
Model-2 : Using 1D convolutions with character embedding



Here are the some papers based on Char-CNN

1. Xiang Zhang, Junbo Zhao, Yann LeCun. Character-level Convolutional Networks for Te
2. Yoon Kim, Yacine Jernite, David Sontag, Alexander M. Rush. Character-Aware Neural
3. Shaojie Bai, J. Zico Kolter, Vladlen Koltun. An Empirical Evaluation of Generic Co
4. Use the pratrained char embeddings https://github.com/minimaxir/char-embeddings/bl





Embedding for M2

before running this, run `'Email and Subject preprocessing'` and `'Converting to a dataframe'` .

```
import numpy as np
import os

file_path = "/content/glove.840B.300d-char.txt"

vectors = {}
with open(file_path, 'r') as f:
    for line in f:
        line_split = line.strip().split(" ")
        vec = np.array(line_split[1:], dtype=float)
        word = line_split[0]

        for char in word:
            if ord(char) < 128:
                if char in vectors:
                    vectors[char] = (vectors[char][0] + vec,
                                     vectors[char][1] + 1)
                else:
                    vectors[char] = (vec, 1)

base_name = os.path.splitext(os.path.basename(file_path))[0] + '-char.txt'
with open(base_name, 'w') as f2:
    for word in vectors:
        avg_vector = np.round(
            (vectors[word][0] / vectors[word][1]), 6).tolist()
        f2.write(word + " " + " ".join(str(x) for x in avg_vector) + "\n")
```

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

```
path_to_glove_file = os.path.join(
    os.path.expanduser("~"), "/content/glove.840B.300d-char-char.txt"
)

embeddings_index = {}
with open(path_to_glove_file) as f:
    for line in tqdm(f):
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep=" ")
        embeddings_index[word] = coefs

print("\nFound %s word vectors." % len(embeddings_index))
# source: https://keras.io/examples/nlp/pretrained_word_embeddings/
```

```
94it [00:00, 8333.82it/s]
Found 94 word vectors.
```

```
num_tokens = len(char_dict) + 2 #should I use the padded data or word_index
```

```
embedding_dim = 300
hits = 0
misses = 0

# Prepare embedding matrix
embedding_matrix = np.zeros((num_tokens, embedding_dim))
for word, i in char_dict.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # Words not found in embedding index will be all-zeros.
        # This includes the representation for "padding" and "OOV"
        embedding_matrix[i] = embedding_vector
        hits += 1
    else:
        misses += 1
print("\nConverted %d words (%d misses)" % (hits, misses))
# source: https://keras.io/examples/nlp/pretrained_word_embeddings/
```

```
Converted 27 words (0 misses)
```

```
from tensorflow.keras.layers import Embedding
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense, Input, Activation, Conv1D, Concatenate, MaxP
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import datetime
import keras
import tensorflow as tf

embedding_layer_char = Embedding(
                            input_dim=num_tokens,
                            output_dim=embedding_dim,
                            weights=[embedding_matrix],
                            embeddings_initializer=keras.initializers.Constant(embeddin
                            trainable=False
                            )
```

Vectorizing for M2

```
from tensorflow.keras.layers import TextVectorization
import tensorflow as tf

vectorizer = TextVectorization(max_tokens=100, output_sequence_length=1500)
# using 5000 bc the sequence length at 98th percentile is 4438 and is 7029 at the 99th
text_ds = tf.data.Dataset.from_tensor_slices(X_train).batch(128)
vectorizer.adapt(text_ds)
```

```
from tensorflow.keras.utils import to_categorical

X_train = vectorizer(np.array([[s] for s in X_train])).numpy()
X_test = vectorizer(np.array([[s] for s in X_test])).numpy()

```

```
y_train = pd.get_dummies(y_train).to_numpy() # using get_dummies instead of to_categori
y_test = pd.get_dummies(y_test).to_numpy()
```

Final Model 2

```
input_layer = keras.Input(shape=(1500, ))
embedded_sequences = embedding_layer_char(input_layer)

conv_layer1 = Conv1D(100, 3, activation="relu", padding='same')(embedded_sequences)
conv_layer2 = Conv1D(100, 5, activation="relu", padding='same')(conv_layer1)

maxpool_layer1 = MaxPooling1D(5)(conv_layer2)

conv_layer3 = Conv1D(100, 5, activation="relu", padding='same')(maxpool_layer1)
conv_layer4 = Conv1D(100, 9, activation="relu", padding='same')(conv_layer3)

maxpool_layer2 = MaxPooling1D(7)(conv_layer4)

flatten = tf.keras.layers.Flatten()(maxpool_layer2)

dropout = Dropout(rate=0.5)(flatten)
dense = Dense(94, activation="relu")(dropout)
dense2 = Dense(94, activation="relu")(dense)


filepath = "model_save2/weights-{epoch:02d}-{val_acc:.4f}.hdf5"
earlystop = EarlyStopping(monitor='val_acc', patience=3, verbose=1, mode='max')
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_acc', verbose=1, save_best

log_dir="logs2/fit/model_1/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1

output = Dense(20, activation='softmax', kernel_initializer=tf.keras.initializers.HeUni
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoa
```

```
import tensorflow_addons as tfa

model = tf.keras.Model(inputs=input_layer, outputs=output)
dot_img_file = '/content/model_images/model_2.png'
tf.keras.utils.plot_model(model, to_file=dot_img_file, show_shapes=True)

micro_f1 = tfa.metrics.F1Score(num_classes=20, average='micro')
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["acc", micro_
model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test), callbacks=[che
```

```
 - 29s 63ms/step - loss: 2.9444 - acc: 0.0787 - f1_score: 0.0787 - val_loss: 2.9224 -

to 0.08795, saving model to model_save2/weights-01-0.0880.hdf5
```

```
 - 25s 57ms/step - loss: 2.9097 - acc: 0.0931 - f1_score: 0.0931 - val_loss: 2.8945 -

95 to 0.09560, saving model to model_save2/weights-02-0.0956.hdf5

 - 25s 56ms/step - loss: 2.8724 - acc: 0.1100 - f1_score: 0.1100 - val_loss: 2.8636 -

60 to 0.11621, saving model to model_save2/weights-03-0.1162.hdf5

 - 25s 57ms/step - loss: 2.8479 - acc: 0.1210 - f1_score: 0.1210 - val_loss: 2.8572 -

21 to 0.12110, saving model to model_save2/weights-04-0.1211.hdf5

 - 27s 60ms/step - loss: 2.8293 - acc: 0.1268 - f1_score: 0.1268 - val_loss: 2.8464 -

10 to 0.12747, saving model to model_save2/weights-05-0.1275.hdf5

 - 27s 60ms/step - loss: 2.8088 - acc: 0.1348 - f1_score: 0.1348 - val_loss: 2.8670 -

m 0.12747

 - 27s 60ms/step - loss: 2.7891 - acc: 0.1372 - f1_score: 0.1372 - val_loss: 2.8525 -

m 0.12747

 - 25s 56ms/step - loss: 2.7626 - acc: 0.1470 - f1_score: 0.1470 - val_loss: 2.8629 -

47 to 0.12853, saving model to model_save2/weights-08-0.1285.hdf5

 - 25s 56ms/step - loss: 2.7335 - acc: 0.1571 - f1_score: 0.1571 - val_loss: 2.8725 -

53 to 0.12874, saving model to model_save2/weights-09-0.1287.hdf5

 - 27s 60ms/step - loss: 2.6955 - acc: 0.1651 - f1_score: 0.1651 - val_loss: 2.8696 -

74 to 0.13087, saving model to model_save2/weights-10-0.1309.hdf5
90>
```

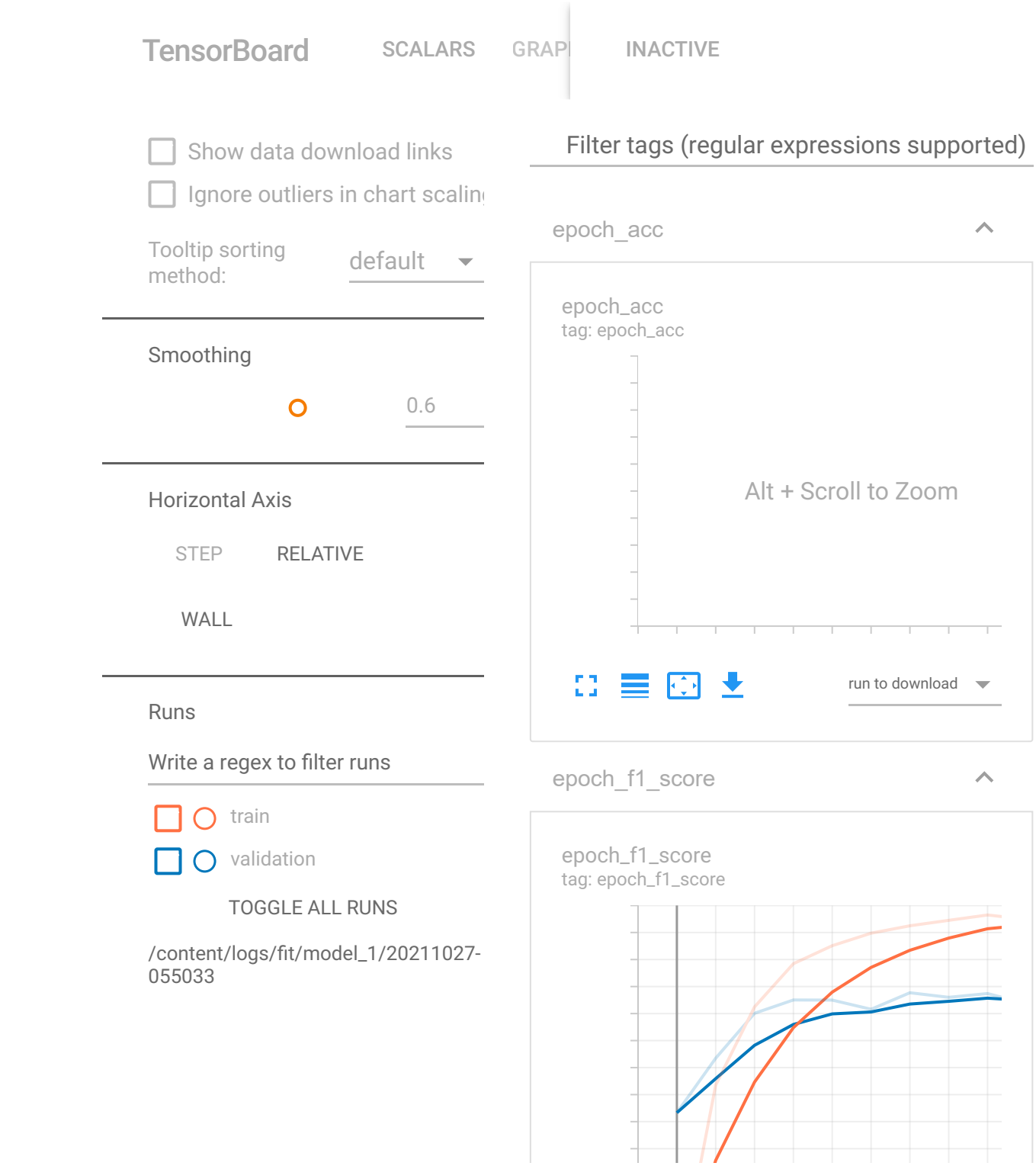
TensorBoard visualisations

Model 1

```
1 %load_ext tensorboard
2 !kill 197
```

```
/bin/bash: line 0: kill: (197) - No such process
```

```
1 %tensorboard --logdir /content/logs/fit/model_1/20211027-055033
```

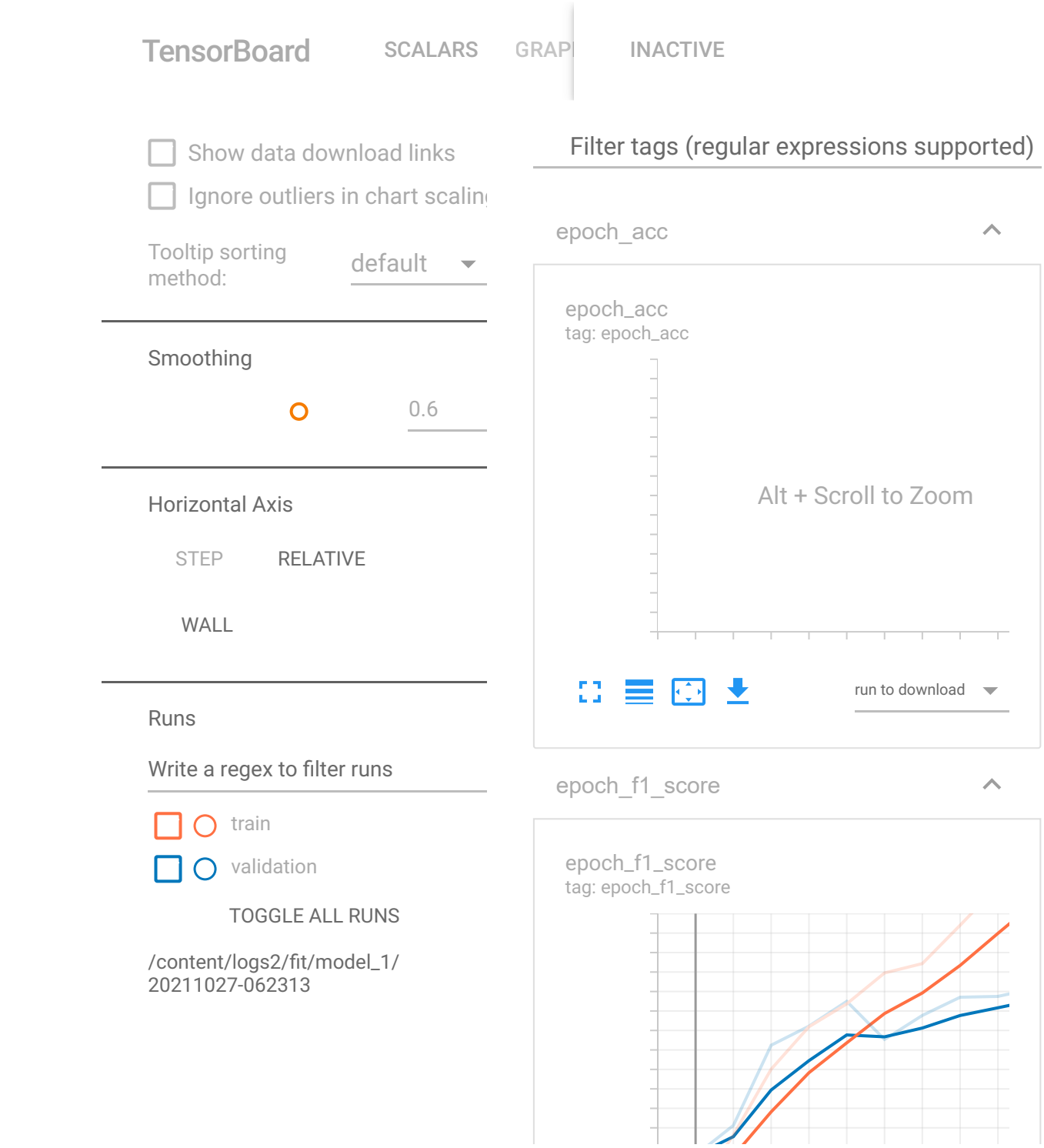


Model 2

```
1 !kill 197
```

```
    /bin/bash: line 0: kill: (197) - No such process
```

```
1 %tensorboard --logdir /content/logs2/fit/model_1/20211027-062313
```



Result

```
 1 from rich.console import Console
 2 from rich.table import Table
 3
 4 table = Table(title="Resultados después de cansarse del Hiper tuning")
 5
 6 table.add_column("Model", justify="right", style="cyan", no_wrap=True)
 7 table.add_column("Validation Accuracy", style="magenta")
 8 table.add_column("Validation F1", justify="right", style="green")
 9 table.add_column("Validation Loss", justify="right", style="red")
10
11
```

```
table.add_row("Model 1", "0.8300", "0.8301", "0.8549")
table.add_row("Model 2", "0.1309", "0.1309", "2.6955")

console = Console()
console.print(table)
```

Resultados después de cansarse del Hiper tuning

Model	Validation Accuracy	Validation F1	Validation Loss
Model 1	0.8300	0.8301	0.8549
Model 2	0.1309	0.1309	2.6955

✓ 0s completed at 12:24 PM