Text Classification:

Data

1. We have total of 20 types of documents(Text files) and total 18828 documents(text files).
2. Download data from this link
3. 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.

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 [1]:

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%
b&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2
www.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly
ttps%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

In [2]:

```
%tensorflow_version 2.x
import tensorflow as tf
print(tf.__version__)
```

```
TensorFlow 2.x selected.
2.1.0
```

In [0]:

```
import pandas as pd
import numpy as np
```

```python
import numpy as np
import os
import re
import string
import nltk
from tqdm import tqdm
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split
```

In [0]:

```python
# !unzip -uq "/content/drive/My Drive/Applied_AI/documents.zip" -d "/content/drive/My
Drive/Applied_AI/"
```

In [0]:

```python
#Extracting label of document and storing it with it's data in a dataframe

entries = os.listdir('/content/drive/My Drive/Applied_AI/documents/')
class_label = [each.split('_')[0] for each in entries]
email_text = [open("/content/drive/My Drive/Applied_AI/documents/"+entries[each], "r",
encoding='ISO-8859-1').read() for each in tqdm(range(0, len(entries)))]

data = pd.DataFrame(columns=['Filename', 'Email_Text', 'Class_Label'])
data['Filename'] = pd.Series(entries)
data['Email_Text'] = pd.Series(email_text)
data['Class_Label'] = pd.Series(class_label)

print("Number of unique classes: ", data.Class_Label.nunique())
print(data.head())
```

```
100%|██████████| 18828/18828 [00:46<00:00, 404.39it/s]
```

```
Number of unique classes:  20
                Filename  ...  Class_Label
0  alt.atheism_49960.txt  ...  alt.atheism
1  alt.atheism_51060.txt  ...  alt.atheism
2  alt.atheism_51119.txt  ...  alt.atheism
3  alt.atheism_51120.txt  ...  alt.atheism
4  alt.atheism_51121.txt  ...  alt.atheism

[5 rows x 3 columns]
```

In [0]:

```python
#Preprocessing Email addresses (@)

data['Preprocessed@'] = [re.findall('[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+', each) for
each in data['Email_Text'].values]
data['Preprocessed@'] = data['Preprocessed@'].apply(lambda x: [i.split('@', 1)[1] for i in x])
data['Preprocessed@'] = data['Preprocessed@'].apply(lambda x: [i.split('.') for i in x])
data['Preprocessed@'] = data['Preprocessed@'].apply(lambda x: [item for sublist in x for item in su
blist])
data['Preprocessed@'] = data['Preprocessed@'].apply(lambda x: [each.lower() for each in x])
data['Preprocessed@'] = data['Preprocessed@'].apply(lambda x: [each for each in x if (len(each)>2 a
nd each!='com')])
data['Preprocessed@'] = data['Preprocessed@'].apply(lambda x: ' '.join(x))

#Replace all the emails by space in the original text.
data['Preprocessed_Email'] = data['Email_Text'].apply(lambda x: re.sub('[a-zA-Z0-9_.+-]+@[a-zA-Z0-9
-]+\.[a-zA-Z0-9-.]+', '', x))
```

In [0]:

```python
# Get subject of the text and Preprocess it

punc = string.punctuation
data['Preprocessed_Subject'] = [re.findall(r'Subject:.*', each) for each in
data['Preprocessed_Email'].values]
data['Preprocessed_Subject'] = data['Preprocessed_Subject'].apply(lambda x: [str.strip(each.split('
:')[-1]) for each in x])
data['Preprocessed_Subject'] = data['Preprocessed_Subject'].apply(lambda x: [' '.join(''.join(e for
```

```
e in each if e not in punc).split()) for each in x])

#Replace all the emails subject line by space in the original text.
data['Preprocessed_Email'] = data['Preprocessed_Email'].apply(lambda x: re.sub(r'Subject:.*', '', x
))
```

In [0]:

```
#Further preprocessing of Email_Text

# a) Delete all the sentances where sentence starts with "Write to:" or "From:"
data['Preprocessed_Email'] = data['Preprocessed_Email'].apply(lambda x: re.sub(r'From:.*', '', x))
data['Preprocessed_Email'] = data['Preprocessed_Email'].apply(lambda x: re.sub(r'Write to:.*', '',
x))

# b) Delete all the tags like "< anyword >"
data['Preprocessed_Email'] = data['Preprocessed_Email'].apply(lambda x: re.sub(r'<.*?>', '', x))

# c) Remove all the newlines('\n'), tabs('\t'), "-", "\".
data['Preprocessed_Email'] = data['Preprocessed_Email'].apply(lambda x: re.sub(r'\n', '', x))
data['Preprocessed_Email'] = data['Preprocessed_Email'].apply(lambda x: re.sub(r'\t', '', x))
data['Preprocessed_Email'] = data['Preprocessed_Email'].apply(lambda x: re.sub(r'\\', '', x))
data['Preprocessed_Email'] = data['Preprocessed_Email'].apply(lambda x: re.sub(r'-', '', x))

# d) Remove all the words which ends with ":"
data['Preprocessed_Email'] = data['Preprocessed_Email'].apply(lambda x: re.sub(r'\S+:', '', x))

# e) Delete all the data which are present in the brackets.
data['Preprocessed_Email'] = data['Preprocessed_Email'].apply(lambda x: re.sub(r'\(.*?\)', '', x))

# f) Decontractions, replace words like below to full words.
def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase

data['Preprocessed_Email'] = data['Preprocessed_Email'].apply(lambda x: decontracted(x))
# g) Replace all the digits with space i.e delete all the digits.
data['Preprocessed_Email'] = data['Preprocessed_Email'].apply(lambda x: re.sub(r'\d+', '', x))

# h)Convert all the words into lower case and remove the words which are greater than or equal to
15 or less than or equal to 2.
data['Preprocessed_Email'] = data['Preprocessed_Email'].apply(lambda x: x.lower())
data['Preprocessed_Email'] = data['Preprocessed_Email'].apply(lambda x: re.sub('\b\w{15,}\b|\b\w{0,
2}\b', '', x))

# i) Removing extra spaces from Preprocessed_Email
data['Preprocessed_Text'] = data['Preprocessed_Text'].apply(lambda x: ' '.join(x.split()))
```

In [0]:

```
#Further Preprocessing of Subject and Email addresses
data['Preprocessed_Subject'] = data['Preprocessed_Subject'].apply(lambda x: ' '.join(x))

data['Preprocessed_Subject'] = data['Preprocessed_Subject'].apply(lambda x: re.sub(r'\d+', '', x))
data['Preprocessed_Subject'] = data['Preprocessed_Subject'].apply(lambda x: x.lower())
data['Preprocessed_Subject'] = data['Preprocessed_Subject'].apply(lambda x: re.sub('[^A-Za-z_ ]', '
', x))

data['Preprocessed@'] = data['Preprocessed@'].apply(lambda x: re.sub(r'\d+', '', x))
data['Preprocessed@'] = data['Preprocessed@'].apply(lambda x: x.lower())
data['Preprocessed@'] = data['Preprocessed@'].apply(lambda x: re.sub('[^A-Za-z_ ]', '', x))
```

In [1]:

```
#Perform text chunking

def chunking(input_str):

    chunks = nltk.ne_chunk(nltk.pos_tag(nltk.word_tokenize(input_str)), binary=False)
    final_string = ""

    for i in list(chunks):
        if ((type(i)==nltk.Tree) and i.label()=='GPE'):
            t = '_'.join(c[0] for c in i.leaves())
        elif ((type(i)==nltk.Tree) and i.label()=='PERSON'):
            continue
        elif ((type(i)==nltk.Tree)):
            t = ' '.join(c[0] for c in i.leaves())
        else:
            t=i[0]

        final_string = final_string+' '+t
        final_string = str.strip(final_string)

    return final_string

data['Preprocessed_Email'] = data['Preprocessed_Email'].apply(lambda x: chunking(x))
```

In [0]:

```
#Preprocessing of Underscore Words
def underscore(input_str):

    output_str = input_str

    list_1 = re.findall(r'[a-z]+_[a-z]+|_[a-z]+_|_[a-z]+|[a-z]+_', input_str) #Find all occurances
of _ words
    list_2 = [list(filter( lambda a: a != '', each.split('_'))) for each in list_1]

    for (every_1, every_2) in zip(list_1,list_2):
        if (len(every_2)==1):
            output_str = output_str.replace(every_1, every_2[0])
        elif (len(every_2)==2):
            output_str = output_str.replace(every_1, max(every_2, key=len))
        else:
            pass

    return output_str

data['Preprocessed_Email'] = data['Preprocessed_Email'].apply(lambda x: underscore(x))
```

In [0]:

```
# Replace all the words except "A-Za-z_" with space
data['Preprocessed_Email'] = data['Preprocessed_Email'].apply(lambda x: re.sub('[^A-Za-z_ ]', '', x
))
```

In [0]:

```
data.rename(columns={'Preprocessed@':'Preprocessed_Emails',
'Preprocessed_Email':'Preprocessed_Text'}, inplace=True)
```

CODE CHECKING

In [9]:

```
#Preprocessing output for alt.atheism_49960.txt doc

print("*"*55 + 'Preprocessed_Emails' + "*"*53)
print(data[data['Filename']=='alt.atheism_49960.txt']['Preprocessed_Emails'].values)
print("-"*127)
```

```
print("*"*55 + 'Preprocessed_Subject' + "*"*52)
print(data[data['Filename']=='alt.atheism_49960.txt']['Preprocessed_Subject'].values[0])
print("-"*127)

print("*"*55 + 'Preprocessed_Text' + "*"*55)
print(data[data['Filename']=='alt.atheism_49960.txt']['Preprocessed_Text'].values[0])
print("-"*127)
```

```
*******************************************************Preprocessed_Emails************************
************************
['mantis netcom mantis']
-------------------------------------------------------------------------------------------------------
-----------------------
*******************************************************Preprocessed_Subject***********************
************************
atheist resources
-------------------------------------------------------------------------------------------------------
-----------------------
*******************************************************Preprocessed_Text**************************
************************
december atheist resources addresses of atheist organizations usafreedom from religion
foundationdarwin fish bumper stickers and assorted other atheist paraphernalia areavailable from t
he freedom from religion foundation in the evolution designsevolution designs sell the darwin fish
it is a fish symbol like the oneschristians stick on their cars but with feet and the word darwin
writteninside the deluxe moulded d plastic fish is postpaid in the us ca people in the san francis
co bay area can get darwin fish from lynn gold try mailing for net people who go to lynn directly
theprice is per fishamerican atheist pressaap publish various atheist books critiques of the bible
lists ofbiblical contradictions and so on one such book the bible handbook by wp ball and gw foote
american atheist press pp isbn nd edition bible contradictions absurdities atrocities immoralities
contains ball the biblecontradicts itself aap based on the king james version of the bible cameron
road austin tx prometheus bookssell books including haught is holy horrors an alternate address pr
ometheus books glenn drive buffalo ny africanamericans for humanisman organization promoting black
secular humanism and uncovering the history ofblack freethought they publish a quarterly
newsletter aah examiner buffalo ny united kingdomrationalist press association national secular so
ciety islington high street holloway roadlondon n ew london n nl british humanist association sout
h place ethical society lamb is conduit passage conway halllondon wcr rh red lion square london wc
r rlfax the national secular society publish the freethinker a monthly magazinefounded in
germanyibka evinternationaler bund der konfessionslosen und atheistenpostfach d berlin germanyibka
publish a miz mizvertrieb postfach d berlin germanyfor atheist books write ibdk internationaler b
ucherdienst der konfessionslosenpostfach d hannover books fictionthomas m disch the santa claus co
mpromise short story the ultimate proof that santa exists all characters and events are fictitious
any similarity to living or dead gods uh well walter m miller jr a canticle for leibowitz one gem
in this post atomic doomsday novel is the monks who spent their livescopying blueprints from saint
leibowitz filling the sheets of paper withink and leaving white lines and lettersedgar pangborn da
vy post atomic doomsday novel set in clerical states the church for example forbids that anyone pr
oduce describe or use any substance containing atoms philip k dickphilip k dick dick wrote many ph
ilosophical and thoughtprovoking short stories and novels his stories are bizarre at times but ver
y approachablehe wrote mainly sf but he wrote about people truth and religion rather
thantechnology although he often believed that he had met some sort of god heremained sceptical am
ongst his novels the following are of some galactic pothealer a fallible alien deity summons a gro
up of earth craftsmen and women to aremote planet to raise a giant cathedral from beneath the
oceans when thedeity begins to demand faith from the earthers pothealer joe fernwright isunable to
comply a polished ironic and amusing novel a maze of death noteworthy for its description of a tec
hnologybased religion valis the schizophrenic hero searches for the hidden mysteries of
gnosticchristianity after reality is fired into his brain by a pink laser beam ofunknown but
possibly divine origin he is accompanied by his dogmatic anddismissively atheist friend and assort
ed other odd characters the divine invasion god invades earth by making a young woman pregnant as
she returns fromanother star system unfortunately she is terminally ill and must beassisted by a d
ead man whose brain is wired to hour easy listening musicmargaret atwood the handmaid is tale a st
ory based on the premise that the us congress is mysteriouslyassassinated and fundamentalists quic
kly take charge of the nation to set it right again the book is the diary of a woman is life as sh
e tries to liveunder the new christian theocracy women is right to own property is revoked and the
ir bank accounts are closed sinful luxuries are outlawed and theradio is only used for readings fr
om the bible crimes are doctors who performed legal abortions in the old world arehunted down and
hanged atwood is writing style is difficult to get used toat first but the tale grows more and mor
e chilling as it goes onvarious authors the bible this somewhat dull and rambling work has often b
een criticized however itis probably worth reading if only so that you will know what all the fuss
isabout it exists in many different versions so make sure you get the onetrue version books nonfic
tionpeter de rosa vicars of christ bantam press although de rosa seems to be christian or even cat
holic this is a veryenlighting history of papal immoralities adulteries fallacies etcmichael a
philosophical justification temple university press philadelphia usaa detailed and scholarly
justification of atheism contains an outstandingappendix defining terminology and usage in this te
ndentiousarea argues both for negative atheism and also for positive atheism includes great refuta
tions of the mostchallenging arguments for god particular attention is paid to refutingcontempory
theists such as platinga and swinburne pages isbn the case against christianity temple university
```

```
pressa comprehensive critique of christianity in which he considersthe best contemporary defences
of christianity and demonstrates that they are unsupportable andor incoherent pages isbn james
turner without god without creed the johns hopkins university press baltimore md usasubtitled the
origins of unbelief in america examines the way in whichunbelief became a mainstream
alternativeworldview focusses on the period and while considering franceand britain the emphasis i
s on american and particularly new englanddevelopments neither a religious history of
secularization or atheism without god without creed is rather the intellectual history of the
fateof a single idea the belief that god exists pages isbn x george seldes the great thoughts ball
antine books new york usaa dictionary of quotations of a different kind concentrating on
statementsand writings which explicitly or implicitly present the person is philosophyand
worldview includes obscure opinions from manypeople for some popular observations traces the way i
n which variouspeople expressed and twisted the idea over the centuries quite a number ofthe
quotations are derived from cardiff is what great men think of religion and noyes views of
religion pages isbn xrichard swinburne the existence of god clarendon paperbacks oxfordthis book i
s the second volume in a trilogy that began with the coherence oftheism and was concluded with fai
th and reason in thiswork swinburne attempts to construct a series of inductive arguments for thee
xistence of god his arguments which are somewhat tendentious and relyupon the imputation of late t
h century western christian values andaesthetics to a god which is supposedly as simple as can be
conceived weredecisively rejected in mackie is the miracle of theism in the revisededition of the
existence of god swinburne includes an appendix in which hemakes a somewhat incoherent attempt to
rebut mackiej l mackie the miracle of theism oxfordthis volume contains a comprehensive review of
the principalarguments for and against the existence of god it ranges from the
classicalphilosophical positions of descartes anselm berkeley hume et al throughthe moral
arguments of newman kant and sidgwick to the recent restatementsof the classical theses by plantin
ga and swinburne it also addresses thosepositions which push the concept of god beyond the realm o
f the rational such as those of kierkegaard kung and philips as well as replacements forgod such a
s lelie is axiarchism the book is a delight to read lessformalistic and better written than martin
is works and refreshingly directwhen compared with the handwaving of swinburnejames a haught holy
an illustrated history of religious murder and madness prometheus bookslooks at religious
persecution from ancient times to the present day andnot only by christianslibrary of congress cat
alog card number norm r allen jr african american an anthology see the listing for african america
ns for humanism abovegordon stein an anthology of atheism and rationalism prometheus booksan antho
logy covering a wide range of subjects including the devil eviland morality and the history of fre
ethought comprehensive bibliographyedmund d cohen the mind of the biblebeliever prometheus booksa
study of why people become christian fundamentalists and what effect ithas on them net
resourcesthere is a small mailbased archive server at mantiscouk which carriesarchives of old
altatheismmoderated articles and assorted other files formore information send mail to saying help
send atheismindexand it will mail back a replymathew
----------------------------------------------------------------------------------------------------
------------------------
```

Preparations before Modelling

In [0]:

```
#Preprocessed_Union is the column that contains merged values of Preprocessed_Emails,
Preprocessed_Subject and Preprocessed_Text

data = data.assign(Preprocessed_Union = data['Preprocessed_Emails'].astype(str)+ ' ' +data['Preproc
essed_Subject'].astype(str)+' '+data['Preprocessed_Text'].astype(str))
```

In [6]:

```
#Splitting data into train and test
X = data['Preprocessed_Union'].values
y = data['Class_Label'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, stratify=y, random_state=
42)

print("Shape of Train Set:", X_train.shape, y_train.shape)
print("Shape of Test Set:", X_test.shape, y_test.shape)
```

```
Shape of Train Set: (14121,) (14121,)
Shape of Test Set: (4707,) (4707,)
```

In [7]:

```
#Tokenization of text data to numbers
tokenizer_obj = tf.keras.preprocessing.text.Tokenizer(num_words=5000, filters='!"#$%&()*+,-./:;<=>
?@[\\]^`{|}~\t\n') # _ has been removed from filters hence it will be preserved
tokenizer_obj.fit_on_texts(X_train)

word_index = tokenizer_obj.word_index
```

```
print('Found %s unique tokens.' % len(word_index))

#Encoded Documents
train_sequences = tokenizer_obj.texts_to_sequences(X_train)
test_sequences = tokenizer_obj.texts_to_sequences(X_test)

print("Train Sequences Length", len(train_sequences))
print("Test Sequences Length", len(test_sequences))

#Selecting max_length of words in a mail
print("Around 96 percentile of mails have length of words less than ",
np.percentile(pd.Series(train_sequences).apply(lambda x: len(x)), 96))
```

```
Found 171578 unique tokens.
Train Sequences Length 14121
Test Sequences Length 4707
Around 96 percentile of mails have length of words less than  604.0
```

Explanation for Sequence Length

1) We have observed here that around 90 percent of sentences have length less than 604 hence we have taken MAX_SEQUENCE_LENGTH to be 600. Words after this length will be trimmed off.

In [8]:

```
#Padding of Word Sequences
MAX_SEQUENCE_LENGTH = 600
vocab_size = len(word_index)+1
train_sequences_pad = tf.keras.preprocessing.sequence.pad_sequences(train_sequences, maxlen=MAX_SEQ
UENCE_LENGTH)
test_sequences_pad = tf.keras.preprocessing.sequence.pad_sequences(test_sequences, maxlen=MAX_SEQUE
NCE_LENGTH)
print("Shape of padded train sequences: ", train_sequences_pad.shape)
print("Shape of padded test sequences: ", test_sequences_pad.shape)
```

```
Shape of padded train sequences:  (14121, 600)
Shape of padded test sequences:  (4707, 600)
```

In [0]:

```
#One-Hot Encoding Output Variable
from sklearn.preprocessing import OneHotEncoder
encoder_onehot = OneHotEncoder()
encoder_onehot.fit(y_train.reshape(-1, 1))
y_train_encoded = encoder_onehot.transform(y_train.reshape(-1, 1)).toarray()
y_test_encoded = encoder_onehot.transform(y_test.reshape(-1, 1)).toarray()
```

In [10]:

```
#Preparing Embedding Layer using Glove vector (100 dimension)

# Loading Glove embedding layer
embeddings_index = {}
f = open('/content/drive/My Drive/Applied_AI/glove.6B.100d.txt')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

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


#*--*create a weight matrix for words in training docs*--*
embedding_matrix = np.zeros((vocab_size, 100))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

```
Found 400000 word vectors.
```

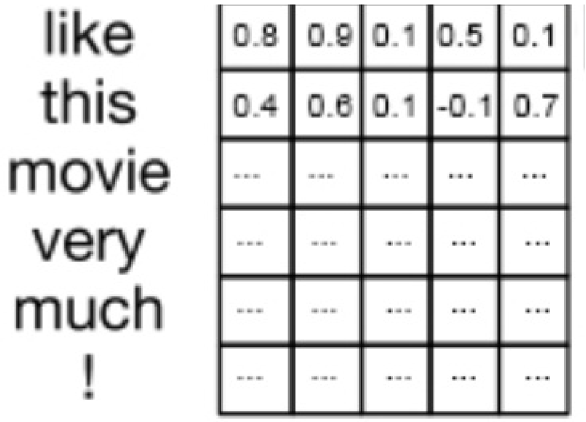
In [12]:

```python
#Early Stopping, Saving best model and Micro Averaged F1 Score callbacks

#*--*Save your model at every epoch if your accuracy is improved from previous epoch.*--*
filepath="/content/drive/My Drive/Applied_AI/CNN_Model_Save/best_model_1.h5"
checkpoint = tf.keras.callbacks.ModelCheckpoint(filepath=filepath, monitor='accuracy',  verbose=1, save_best_only=True, save_weights_only=True, mode='auto')

#*--*Early Stopping*--*
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='accuracy', min_delta=0, patience=2, verbose=1, mode='auto')

#*--*Using tensorboard to analyze models*--*
log_dir=r"logs/fit/Callbacks/Model_1_Try_1"
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1, write_graph=True,write_grads=True)

#*--*Micro Average F1-Score*--*
class custom_Callback(tf.keras.callbacks.Callback):

    def on_train_begin(self, logs={}):
        self.history={'loss': [], 'acc': [], 'val_loss': [], 'val_acc': [], 'val_f1' : [], 'val_AUC' : []}
        self.train_data = train_sequences_pad
        self.train_target = y_train_encoded
        self.validation_data = test_sequences_pad
        self.validation_target = y_test_encoded

    def on_epoch_end(self, epoch, logs={}):
        self.history['loss'].append(logs.get('loss'))
        self.history['acc'].append(logs.get('acc'))
        if logs.get('val_loss', -1) != -1:
            self.history['val_loss'].append(logs.get('val_loss'))
        if logs.get('accuracy', -1) != -1:
            self.history['val_acc'].append(logs.get('accuracy'))

        #Calculating and appending f1_score
        val_predict = (np.asarray(self.model.predict(self.validation_data))).round()
        val_targ = self.validation_target
        train_predict = (np.asarray(self.model.predict(self.train_data))).round()
        train_targ = self.train_target
        _train_f1 = f1_score(train_targ, train_predict, average='micro')
        _val_f1 = f1_score(val_targ, val_predict, average='micro')
        self.history['val_f1'].append(_val_f1)
        print("\nF1 Score Train: %f " %(_train_f1))
        print("\nF1 Score Validation: %f " %(_val_f1))

custom=custom_Callback()
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.
```

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 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



In [0]:

```
#Creating Embedding Layer
embedding_layer = tf.keras.layers.Embedding(vocab_size,
                                            100,
                                            weights=[embedding_matrix],
                                            input_length=MAX_SEQUENCE_LENGTH,
                                            trainable=False)
```

In [14]:

```
def create_model():

    input_shape = tf.keras.layers.Input(shape=(MAX_SEQUENCE_LENGTH,))
    embedded_sequences = embedding_layer(input_shape)

    tower_1 = tf.keras.layers.Conv1D(64, 5, activation='relu')(embedded_sequences) #Kernel Size(M)
= 3
    tower_2 = tf.keras.layers.Conv1D(64, 7, activation='relu')(embedded_sequences) #Kernel Size(N)
= 5
    tower_3 = tf.keras.layers.Conv1D(64, 9, activation='relu')(embedded_sequences) #Kernel Size(O)
= 7

    concat = tf.keras.layers.concatenate([tower_1, tower_2, tower_3], axis=1)
    max_pool = tf.keras.layers.MaxPooling1D(9)(concat)#9

    tower_1a = tf.keras.layers.Conv1D(64, 5, activation='relu')(max_pool) #Kernel Size(i) = 5
    tower_2b = tf.keras.layers.Conv1D(64, 7, activation='relu')(max_pool) #Kernel Size(j) = 7
    tower_3c = tf.keras.layers.Conv1D(64, 9, activation='relu')(max_pool) #Kernel Size(k) = 9

    concat2 = tf.keras.layers.concatenate([tower_1a, tower_2b, tower_3c], axis=1)
    max_pool2 = tf.keras.layers.MaxPooling1D(9)(concat2)#9

    convP = tf.keras.layers.Conv1D(64, 9, activation='relu')(max_pool2) #Kernel Size(P) = 9
    flatten = tf.keras.layers.Flatten()(convP)
    dropout = tf.keras.layers.Dropout(0.7)(flatten) #Taking Dropout Rate = 0.2

    dense = tf.keras.layers.Dense(128, activation='relu')(dropout) #128
    preds = tf.keras.layers.Dense(20, activation='softmax')(dense)

    model_created = tf.keras.models.Model(input_shape, preds)
    return model_created

#Calling create_model method and printing summary of model
model = create_model()
print(model.summary())

#Compiling the model and fitting it on the train data
optimizer_adam = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(loss='categorical_crossentropy', optimizer=optimizer_adam, metrics=['accuracy'])
model.fit(train_sequences_pad, y_train_encoded, validation_data=(test_sequences_pad, y_test_encoded
), nb_epoch=5, batch_size=64, callbacks=[checkpoint, early_stopping, tensorboard_callback, custom],
use_multiprocessing=False)
```

```
WARNING:tensorflow:Large dropout rate: 0.7 (>0.5). In TensorFlow 2.x, dropout() uses dropout rate
```

```
instead of keep_prob. Please ensure that this is intended.
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 600)]        0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 600, 100)     17157900    input_1[0][0]
__________________________________________________________________________________________________
conv1d (Conv1D)                 (None, 596, 64)      32064       embedding[0][0]
__________________________________________________________________________________________________
conv1d_1 (Conv1D)               (None, 594, 64)      44864       embedding[0][0]
__________________________________________________________________________________________________
conv1d_2 (Conv1D)               (None, 592, 64)      57664       embedding[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 1782, 64)     0           conv1d[0][0]
                                                                 conv1d_1[0][0]
                                                                 conv1d_2[0][0]
__________________________________________________________________________________________________
max_pooling1d (MaxPooling1D)    (None, 198, 64)      0           concatenate[0][0]
__________________________________________________________________________________________________
conv1d_3 (Conv1D)               (None, 194, 64)      20544       max_pooling1d[0][0]
__________________________________________________________________________________________________
conv1d_4 (Conv1D)               (None, 192, 64)      28736       max_pooling1d[0][0]
__________________________________________________________________________________________________
conv1d_5 (Conv1D)               (None, 190, 64)      36928       max_pooling1d[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 576, 64)      0           conv1d_3[0][0]
                                                                 conv1d_4[0][0]
                                                                 conv1d_5[0][0]
__________________________________________________________________________________________________
max_pooling1d_1 (MaxPooling1D)  (None, 64, 64)       0           concatenate_1[0][0]
__________________________________________________________________________________________________
conv1d_6 (Conv1D)               (None, 56, 64)       36928       max_pooling1d_1[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 3584)         0           conv1d_6[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 3584)         0           flatten[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 128)          458880      dropout[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 20)           2580        dense[0][0]
==================================================================================================
Total params: 17,877,088
Trainable params: 719,188
Non-trainable params: 17,157,900
__________________________________________________________________________________________________
None
WARNING:tensorflow:The `nb_epoch` argument in `fit` has been renamed `epochs`.
Train on 14121 samples, validate on 4707 samples
Epoch 1/5
WARNING:tensorflow:Large dropout rate: 0.7 (>0.5). In TensorFlow 2.x, dropout() uses dropout rate
instead of keep_prob. Please ensure that this is intended.
WARNING:tensorflow:Large dropout rate: 0.7 (>0.5). In TensorFlow 2.x, dropout() uses dropout rate
instead of keep_prob. Please ensure that this is intended.
14080/14121 [============================>.] - ETA: 0s - loss: 2.5597 - accuracy: 0.1608
Epoch 00001: accuracy improved from -inf to 0.16104, saving model to /content/drive/My
Drive/Applied_AI/CNN_Model_Save/best_model_1.h5

F1 Score Train: 0.192647

F1 Score Validation: 0.196078
14121/14121 [==============================] - 149s 11ms/sample - loss: 2.5584 - accuracy: 0.1610
- val_loss: 1.9969 - val_accuracy: 0.3038
Epoch 2/5
14080/14121 [============================>.] - ETA: 0s - loss: 1.5367 - accuracy: 0.4578
Epoch 00002: accuracy improved from 0.16104 to 0.45825, saving model to /content/drive/My
Drive/Applied_AI/CNN_Model_Save/best_model_1.h5

F1 Score Train: 0.564412

F1 Score Validation: 0.522014
14121/14121 [==============================] - 146s 10ms/sample - loss: 1.5352 - accuracy: 0.4583
- val_loss: 1.2215 - val_accuracy: 0.5842
Epoch 3/5
14080/14121 [============================>.] - ETA: 0s - loss: 1.0366 - accuracy: 0.6415
Epoch 00003: accuracy improved from 0.45825 to 0.64167, saving model to /content/drive/My
```

```
Drive/Applied_AI/CNN_Model_Save/best_model_1.h5

F1 Score Train: 0.702128

F1 Score Validation: 0.623768
14121/14121 [==============================] - 143s 10ms/sample - loss: 1.0373 - accuracy: 0.6417
- val_loss: 1.0427 - val_accuracy: 0.6490
Epoch 4/5
14080/14121 [============================>.] - ETA: 0s - loss: 0.7691 - accuracy: 0.7332
Epoch 00004: accuracy improved from 0.64167 to 0.73330, saving model to /content/drive/My
Drive/Applied_AI/CNN_Model_Save/best_model_1.h5

F1 Score Train: 0.831650

F1 Score Validation: 0.714389
14121/14121 [==============================] - 143s 10ms/sample - loss: 0.7687 - accuracy: 0.7333
- val_loss: 0.9031 - val_accuracy: 0.7077
Epoch 5/5
14080/14121 [============================>.] - ETA: 0s - loss: 0.5826 - accuracy: 0.7988
Epoch 00005: accuracy improved from 0.73330 to 0.79874, saving model to /content/drive/My
Drive/Applied_AI/CNN_Model_Save/best_model_1.h5

F1 Score Train: 0.869641

F1 Score Validation: 0.731210
14121/14121 [==============================] - 142s 10ms/sample - loss: 0.5830 - accuracy: 0.7987
- val_loss: 0.8999 - val_accuracy: 0.7215
```

Out[14]:

```
<tensorflow.python.keras.callbacks.History at 0x7ff4948abe10>
```

In [15]:

```
scores = model.evaluate(test_sequences_pad, y_test_encoded, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
y_label = (np.asarray(model.predict(test_sequences_pad))).round()
print("F1 SCore: ", f1_score(y_test_encoded, y_label, average='micro'))
```

```
Accuracy: 72.15%
F1 SCore:  0.7312104893467573
```

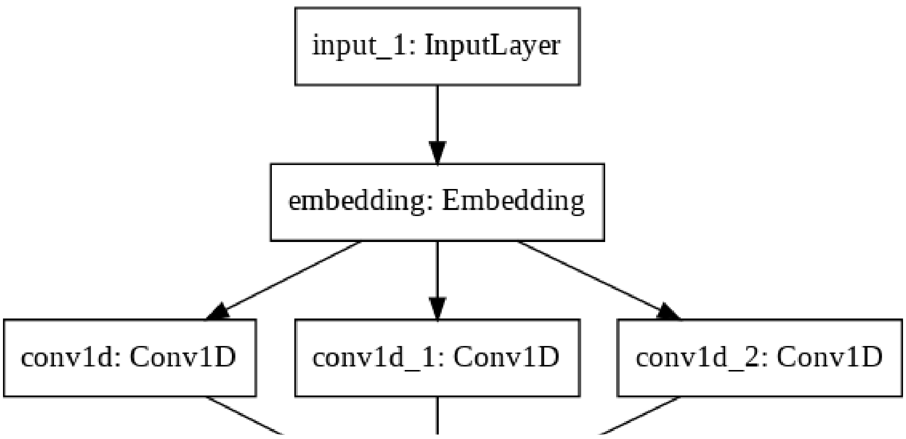
Observations

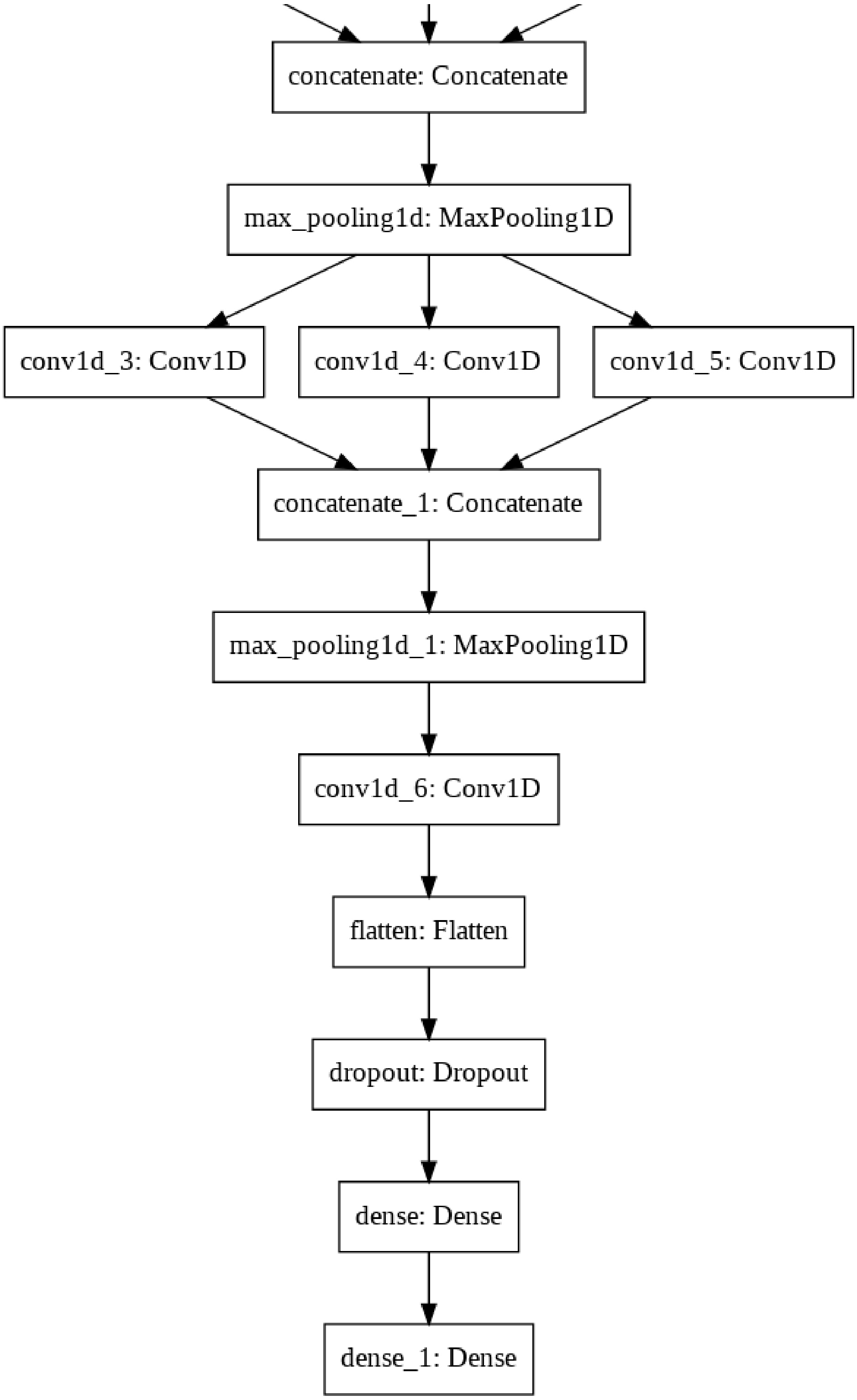
With word embeddings we are getting much better results than character embeddings as it is giving an acuracy of almost 73% which is pretty good also F1 score is also pretty good around 0.74

In [16]:

```
#Plot the architecture of the model

# from  import plot_model
tf.keras.utils.plot_model(model, to_file='/content/drive/My
Drive/Applied_AI/CNN_Model_Save/Model_1.png')
```

Out[16]:

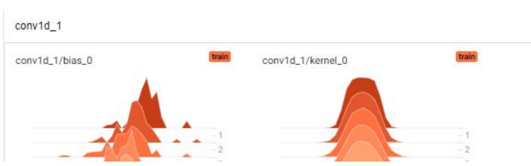


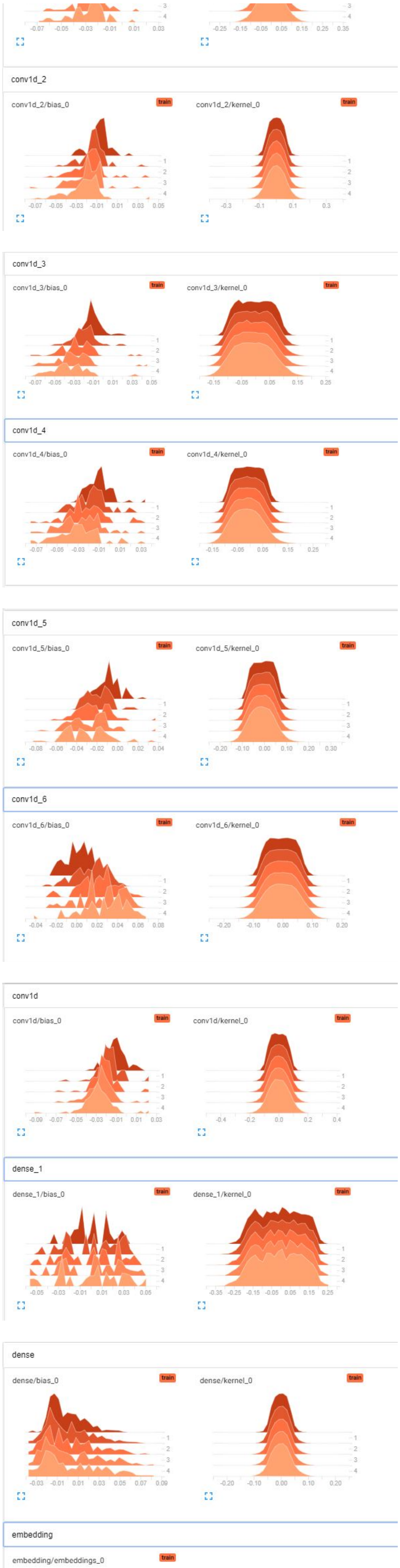


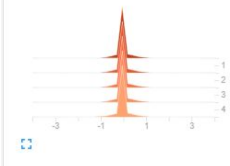
In [17]:

```
#Visualization via Tensorboard

%reload_ext tensorboard
%tensorboard --logdir "logs/fit/Callbacks/Model_1_Try_1"
```

Visualization of the histograms provided by Tensorboard for Model 1




conv1d_2
conv1d_2/bias_0
train
conv1d_2/kernel_0
train
conv1d_3
conv1d_3/bias_0
train
conv1d_3/kernel_0
train
conv1d_4
conv1d_4/bias_0
train
conv1d_4/kernel_0
train
conv1d_5
conv1d_5/bias_0
train
conv1d_5/kernel_0
train
conv1d_6
conv1d_6/bias_0
train
conv1d_6/kernel_0
train
conv1d
conv1d/bias_0
train
conv1d/kernel_0
train
dense_1
dense_1/bias_0
train
dense_1/kernel_0
train
dense
dense/bias_0
train
dense/kernel_0
train
embedding
embedding/embeddings_0
train



Observations

Tensorboard gives us this privelage of looking at these histograms which inform us about how the distribution of our tensors have changed over epochs.

Since we are looking at histograms for CNN hence we have 2 plots for each convolution layer i.e. one for bias and other for kernel values of our convolution operator for each layer over 5 epochs.

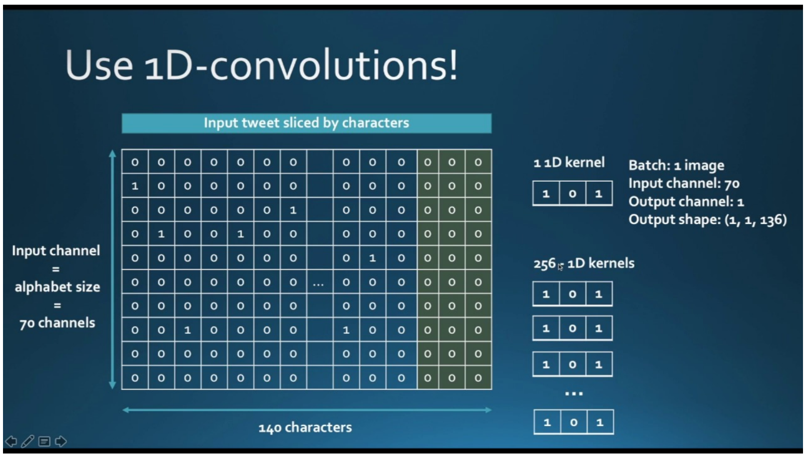
1) Here we can observe that distribution of our kernel values is changing little bit with after every layer hence we can say that our convolution layer are learning different aspect of data in each layer.

2) We have somewhat similar to normal distribution for our kernel values with different variance between conv1d_1 and conv1d_2 layer.

3) conv1d_3 and conv1d_4 are having almost uniform distribution where the probability of kernel values lying in a particular range is equally likely.

4) If we look at our embedding layer, there is no change happening there since we are used pre trained glove vectors hence no learning is happening there over epochs.

Model-2 : Using 1D convolutions with character embedding



In [34]:

```
#Tokenization of text data characters to numbers
tokenizer_obj_char = tf.keras.preprocessing.text.Tokenizer(num_words=None, filters='!"#$%&()*+,-./
:;<=>?@[\\]^`{|}~\t\n', char_level=True, oov_token='UNK') # _ has benn removed from filters hence
it will be preserved
tokenizer_obj_char.fit_on_texts(X_train)

word_index = tokenizer_obj_char.word_index
print('Found %s unique tokens.' % len(word_index))

#Encoded Documents
train_sequences = tokenizer_obj_char.texts_to_sequences(X_train)
test_sequences = tokenizer_obj_char.texts_to_sequences(X_test)

print("Train Sequences Length", len(train_sequences))
print("Test Sequences Length", len(test_sequences))

#Selecting max_length of words in a mail
print("Around 90 percentile of mails have length of words less than ",
np.percentile(pd.Series(train_sequences).apply(lambda x: len(x)), 90))
```

```
np.percentile(pd.Series(train_sequences).apply(lambda x: len(x)), 90))
```

```
Found 29 unique tokens.
Train Sequences Length 14121
Test Sequences Length 4707
Around 90 percentile of mails have length of words less than  2471.0
```

Explanation for Sequence Length

1) We have observed here that around 90 percent of sentences have length less than 2471 hence we have taken MAX_SEQUENCE_LENGTH to be 2500. Words after this length will be trimmed off.

In [35]:

```
#Padding of Char Sequences
MAX_SEQUENCE_LENGTH = 2500
vocab_size = len(word_index)+1
train_sequences_pad = tf.keras.preprocessing.sequence.pad_sequences(train_sequences, maxlen=MAX_SEQ
UENCE_LENGTH)
test_sequences_pad = tf.keras.preprocessing.sequence.pad_sequences(test_sequences, maxlen=MAX_SEQUE
NCE_LENGTH)
print("Shape of padded train sequences: ", train_sequences_pad.shape)
print("Shape of padded test sequences: ", test_sequences_pad.shape)
```

```
Shape of padded train sequences:  (14121, 2500)
Shape of padded test sequences:  (4707, 2500)
```

In [36]:

```
#Character Embedding Preparatation using Glove vector (300 dimension)

# Loading Glove embedding layer
embeddings_index = {}
f = open('/content/drive/My Drive/Applied_AI/glove-840B-300d-char_embed.txt')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

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


#*--*create a weight matrix for characters in training docs*--*
embedding_matrix = np.zeros((vocab_size, 300))
for char, i in word_index.items():
    embedding_vector = embeddings_index.get(char)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

#Creating Embedding Layer
embedding_layer = tf.keras.layers.Embedding(vocab_size,
                                            300,
                                            weights=[embedding_matrix],
                                            input_length=MAX_SEQUENCE_LENGTH,
                                            trainable=False)
```

```
Found 94 word vectors.
```

In [37]:

```
#Early Stopping and Saving best model callbacks

#*--*Save your model at every epoch if your accuracy is improved from previous epoch.*--*
filepath="/content/drive/My Drive/Applied_AI/CNN_Model_Save/best_model_2.h5"
checkpoint = tf.keras.callbacks.ModelCheckpoint(filepath=filepath, monitor='accuracy',  verbose=1,
save_best_only=True, save_weights_only=True, mode='auto')

#*--*Early Stopping*--*
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='accuracy', min_delta=0, patience=2, verb
ose=1, mode='auto')
```

```
#*--*Using tensorboard to analyze models*--*
log_dir=r"logs/fit/Callbacks/Model_2_Try_2"
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1, write_grap
h=True,write_grads=True)

#*--*Micro Average F1-Score*--*
class custom_Callback_2(tf.keras.callbacks.Callback):

    def on_train_begin(self, logs={}):
        self.history={'loss': [], 'acc': [], 'val_loss': [], 'val_acc': [], 'val_f1' : [],
'val_AUC' : []}
        self.train_data = train_sequences_pad
        self.train_target = y_train_encoded
        self.validation_data = test_sequences_pad
        self.validation_target = y_test_encoded

    def on_epoch_end(self, epoch, logs={}):
        self.history['loss'].append(logs.get('loss'))
        self.history['acc'].append(logs.get('acc'))
        if logs.get('val_loss', -1) != -1:
            self.history['val_loss'].append(logs.get('val_loss'))
        if logs.get('accuracy', -1) != -1:
            self.history['val_acc'].append(logs.get('accuracy'))

        #Calculating and appending f1_score and AUC score
        val_predict = (np.asarray(self.model.predict(self.validation_data))).round()
        val_targ = self.validation_target
        train_predict = (np.asarray(self.model.predict(self.train_data))).round()
        train_targ = self.train_target
        _train_f1 = f1_score(train_targ, train_predict, average='micro')
        _val_f1 = f1_score(val_targ, val_predict, average='micro')
        self.history['val_f1'].append(_val_f1)
        print("\nF1 Score Train: %f " %(_train_f1))
        print("\nF1 Score Validation: %f " %(_val_f1))

custom=custom_Callback_2()
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.
```

In [38]:

```
def create_model():

    input_shape = tf.keras.layers.Input(shape=(MAX_SEQUENCE_LENGTH,))
    embedded_sequences = embedding_layer(input_shape)

    convN = tf.keras.layers.Conv1D(32, 3, activation='relu')(embedded_sequences) #Kernel Size(N) =
3
    convM = tf.keras.layers.Conv1D(32, 5, activation='relu')(convN) #Kernel Size(M) = 5
    max_pool = tf.keras.layers.MaxPooling1D(5)(convM)
    convK = tf.keras.layers.Conv1D(32, 3, activation='relu')(max_pool) #Kernel Size(K) = 3
    convT = tf.keras.layers.Conv1D(32, 5, activation='relu')(convK) #Kernel Size(T) = 5
    max_pool2 = tf.keras.layers.MaxPooling1D(5)(convT)
    flatten = tf.keras.layers.Flatten()(max_pool2)
    dropout = tf.keras.layers.Dropout(0.5)(flatten) #Taking Dropout Rate = 0.5
    dense = tf.keras.layers.Dense(128, activation='relu')(dropout)
    preds = tf.keras.layers.Dense(20, activation='softmax')(dense)

    model_created = tf.keras.models.Model(input_shape, preds)
    return model_created

#Calling create_model method and printing summary of model
model2 = create_model()
print(model2.summary())

#Compiling the model and fitting it on the train data
optimizer_adam = tf.keras.optimizers.Adam(learning_rate=0.001)
model2.compile(loss='categorical_crossentropy', optimizer=optimizer_adam, metrics=['accuracy'])
model2.fit(train_sequences_pad, y_train_encoded, nb_epoch=5, batch_size=64, validation_data=(test_s
equences_pad, y_test_encoded), callbacks=[checkpoint, early_stopping, tensorboard_callback, custom]
, use_multiprocessing=False)
```

```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 2500)]            0
_________________________________________________________________
embedding_3 (Embedding)      (None, 2500, 300)         9000
_________________________________________________________________
conv1d_15 (Conv1D)           (None, 2498, 32)          28832
_________________________________________________________________
conv1d_16 (Conv1D)           (None, 2494, 32)          5152
_________________________________________________________________
max_pooling1d_6 (MaxPooling1 (None, 498, 32)           0
_________________________________________________________________
conv1d_17 (Conv1D)           (None, 496, 32)           3104
_________________________________________________________________
conv1d_18 (Conv1D)           (None, 492, 32)           5152
_________________________________________________________________
max_pooling1d_7 (MaxPooling1 (None, 98, 32)            0
_________________________________________________________________
flatten_3 (Flatten)          (None, 3136)              0
_________________________________________________________________
dropout_3 (Dropout)          (None, 3136)              0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               401536
_________________________________________________________________
dense_7 (Dense)              (None, 20)                2580
=================================================================
Total params: 455,356
Trainable params: 446,356
Non-trainable params: 9,000
_________________________________________________________________
None
WARNING:tensorflow:The `nb_epoch` argument in `fit` has been renamed `epochs`.
Train on 14121 samples, validate on 4707 samples
Epoch 1/5
14080/14121 [============================>.] - ETA: 0s - loss: 2.9360 - accuracy: 0.0811
Epoch 00001: accuracy improved from -inf to 0.08116, saving model to /content/drive/My
Drive/Applied_AI/CNN_Model_Save/best_model_2.h5

F1 Score Train: 0.000000

F1 Score Validation: 0.000000
14121/14121 [==============================] - 202s 14ms/sample - loss: 2.9357 - accuracy: 0.0812
- val_loss: 2.9225 - val_accuracy: 0.0909
Epoch 2/5
14080/14121 [============================>.] - ETA: 0s - loss: 2.9178 - accuracy: 0.0920
Epoch 00002: accuracy improved from 0.08116 to 0.09192, saving model to /content/drive/My
Drive/Applied_AI/CNN_Model_Save/best_model_2.h5

F1 Score Train: 0.002546

F1 Score Validation: 0.001274
14121/14121 [==============================] - 200s 14ms/sample - loss: 2.9179 - accuracy: 0.0919
- val_loss: 2.9138 - val_accuracy: 0.0928
Epoch 3/5
14080/14121 [============================>.] - ETA: 0s - loss: 2.8963 - accuracy: 0.0981
Epoch 00003: accuracy improved from 0.09192 to 0.09829, saving model to /content/drive/My
Drive/Applied_AI/CNN_Model_Save/best_model_2.h5

F1 Score Train: 0.008878

F1 Score Validation: 0.005923
14121/14121 [==============================] - 201s 14ms/sample - loss: 2.8958 - accuracy: 0.0983
- val_loss: 2.9054 - val_accuracy: 0.1037
Epoch 4/5
14080/14121 [============================>.] - ETA: 0s - loss: 2.8593 - accuracy: 0.1177
Epoch 00004: accuracy improved from 0.09829 to 0.11756, saving model to /content/drive/My
Drive/Applied_AI/CNN_Model_Save/best_model_2.h5

F1 Score Train: 0.012380

F1 Score Validation: 0.008029
14121/14121 [==============================] - 202s 14ms/sample - loss: 2.8593 - accuracy: 0.1176
- val_loss: 2.8818 - val_accuracy: 0.1103
Epoch 5/5
14080/14121 [============================>.] - ETA: 0s - loss: 2.7949 - accuracy: 0.1391
```

```
Epoch 00005: accuracy improved from 0.11756 to 0.13894, saving model to /content/drive/My
Drive/Applied_AI/CNN_Model_Save/best_model_2.h5

F1 Score Train: 0.019620

F1 Score Validation: 0.008863
14121/14121 [==============================] - 204s 14ms/sample - loss: 2.7949 - accuracy: 0.1389
- val_loss: 2.8728 - val_accuracy: 0.1188
```

Out[38]:

```
<tensorflow.python.keras.callbacks.History at 0x7ff489038668>
```

In [39]:

```
scores = model2.evaluate(test_sequences_pad, y_test_encoded, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
y_label = (np.asarray(model2.predict(test_sequences_pad))).round()
print("F1 SCore: ", f1_score(y_test_encoded, y_label, average='micro'))
```

```
Accuracy: 11.88%
F1 SCore:  0.008862629246676515
```

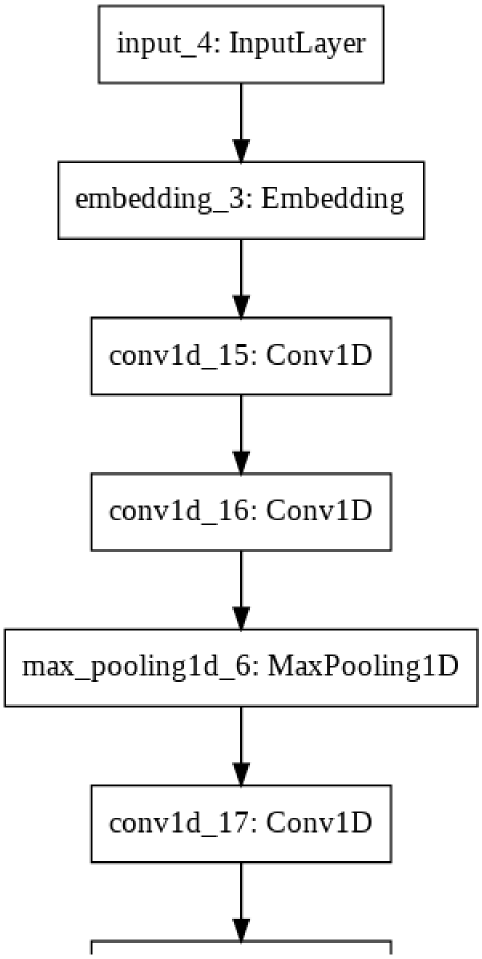
Observations

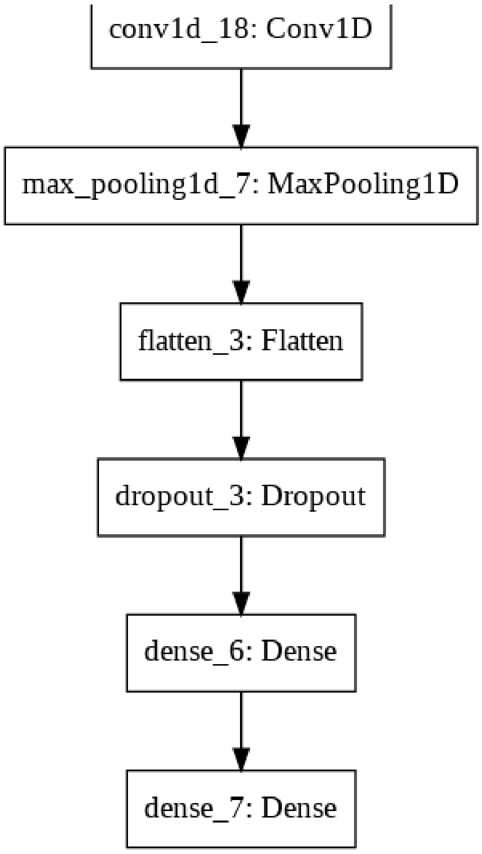
With character embedding we are not getting good results as it is giving an aacuracy of only 11.88% which is pretty less also F1 score is not that good.

In [40]:

```
#Plot the architecture of the model

# from  import plot_model
tf.keras.utils.plot_model(model2, to_file='/content/drive/My
Drive/Applied_AI/CNN_Model_Save/Model_2.png')
```

Out[40]:

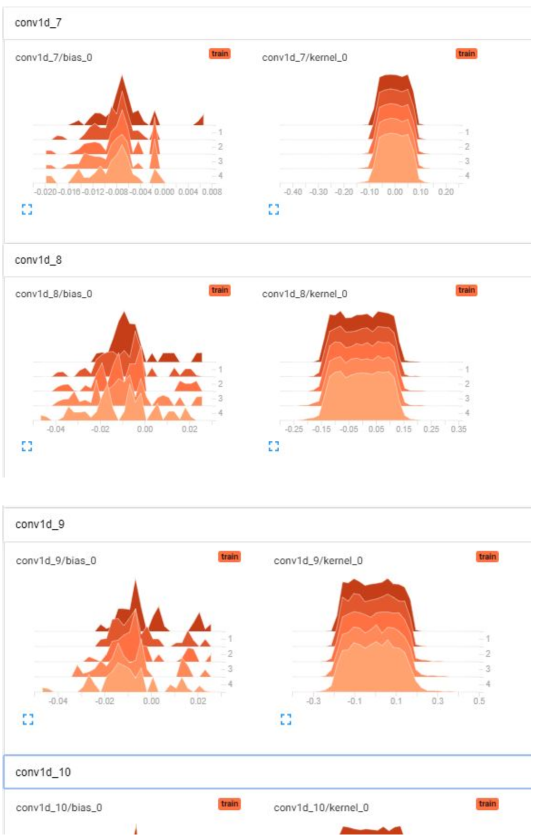


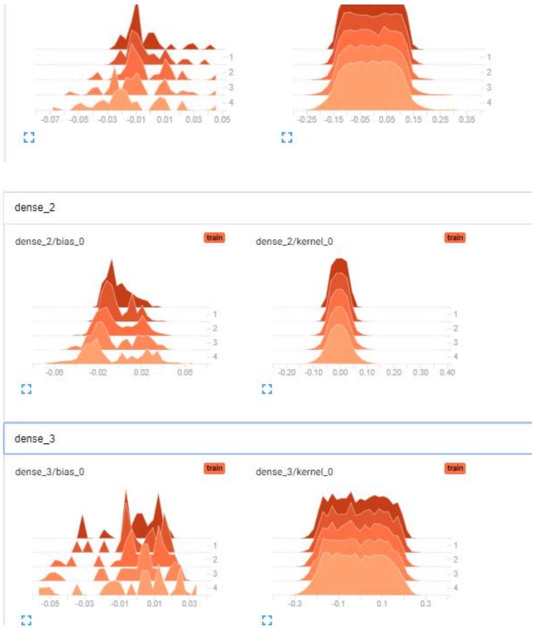


In [41]:

```
#Visualization via Tensorboard

%reload_ext tensorboard
%tensorboard --logdir "logs/fit/Callbacks/Model_2_Try_2"
```

Visualization of the histograms provided by Tensorboard for Model 2





Observations

Tensorboard gives us this privelage of looking at these histograms which inform us about how the distribution of our tensors have changed over epochs.

Since we are looking at histograms for CNN hence we have 2 plots for each convolution layer i.e. one for bias and other for kernel values of our convolution operator for each layer over 5 epochs.

1) Here we can observe that distribution of our kernel values is changing little bit with after every layer hence we can say that our convolution layer are learning different aspect of data in each layer.

2) We have almost uniform distribution for conv1d_7, conv1d_8, conv1d_9 and conv1d_10 which means that kernel values in these layers have equally likely probability to lie in the range [-0.10, +0.10], [-0.15, +0.15], etc respectively.

3) Since we are oberving plateaus for almost every convolution layer for kernel values which indiactes that not much learning is happening across layers and hence our model is not performing good.

4) dense_2 layer seems to have a normal distribution for kernel values.

5) Bias values seems to have no distinguished distribution rather bias values are more scattered across.

6) If we look at our embedding layer, there is no change happening there since we are used pre trained glove vectors hence no learning is happening there over epochs.