In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
import re
import os
import pickle


from functools import partial
from tqdm import tqdm
tqdm = partial(tqdm, position=0, leave=True)

import tensorflow as tf
import tensorflow.keras as keras
from sklearn.preprocessing import OneHotEncoder,LabelEncoder
# from tensorflow.keras import utils
from keras.utils import np_utils

from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Input, Dense, Flatten, Embedding, Concatenate, Conv
1D, MaxPooling1D, Dropout
from tensorflow.keras.models import Sequential

from keras.utils.vis_utils import plot_model
from sklearn.metrics import  f1_score
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

import nltk
from nltk import ne_chunk, pos_tag, word_tokenize
from nltk.tree import Tree
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')

warnings.filterwarnings("ignore")
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.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.
```

In [2]:

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

Load Data and Preprocess

In [3]:

```
!gdown --id "1IKPfj6pBPe5wjx1GjRquxyLJ8VKdtebN"
```

```
Downloading...
From: https://drive.google.com/uc?id=1IKPfj6pBPe5wjx1GjRquxyLJ8VKdtebN
To: /content/documents.rar
19.0MB [00:00, 116MB/s]
```

In [68]:

```
!mkdir data
!unrar e documents.rar "data"
```

In [5]:

```
import chardet
rawdata = open("data/alt.atheism_49960.txt", 'rb').read()
print(chardet.detect(rawdata))
print("ENCODING: ",chardet.detect(rawdata)["encoding"])
```

```
{'encoding': 'ISO-8859-1', 'confidence': 0.73, 'language': ''}
ENCODING:  ISO-8859-1
```

In [6]:

```
text_list = []
class_label_list = []
for fileName in os.listdir("data"):
  class_label  = "".join(fileName.split("_")[:-1])
  class_label_list.append(class_label)
  with open("data/"+ fileName,"r",encoding="ISO-8859-1") as textFile:
    text_list.append(textFile.read())
```

In [7]:

```
data = pd.DataFrame(data={"text" : text_list, "class"  : class_label_list})
```

In [8]:

```
def getEmail(text):
  re_email = re.compile("([a-zA-Z0-9+._-]+@[a-zA-Z0-9._-]+)")
  emails = re.findall(re_email, text)
  email_list = []
  for email in emails:
    for word in email.split("@")[-1].split("."):
      if len(word) >= 2 and word != "com":
        email_list.append(word)

  email_removed = re.sub(re_email," ",text)
  return email_removed, " ".join(email_list)
```

In [9]:

```
def getSubject(text):
  re_subject = re.compile("^Subject: .*$",re.MULTILINE)
  sub = re.findall(re_subject,text)[0]
  sub_list = []

  for word in sub.split(":")[1:]:
    # only keeps alphabets and numbers
    word = re.sub('[^A-Za-z0-9\s]+', '', word)
    # remove \n \r
    # r'' indicates to treat string as raw string and ignore meaning on \n and \r
    word = re.sub(r'[\n\r\t]', '', word)
    sub_list.append(word)

  remove_subject = re.sub(re_subject, " ", text)
  return remove_subject, " ".join(sub_list)
```

In [10]:

```
def removeLines(text):
  re_from = re.compile("^From: .*$",re.MULTILINE)
  re_write = re.compile("^Write To: .*$",re.MULTILINE)


  removed_text = re.sub(re_from, "", text) # From:
  removed_text = re.sub(re_write, "", removed_text) # Write To:
  removed_text = re.sub("<.*>", "", removed_text) # < word >
  removed_text = re.sub("\(.*\)", "", removed_text) # ( word )
  removed_text = re.sub(r'[\n\r\t-/]', " ", removed_text) # remove \n \r
  removed_text = re.sub('\s+',' ',removed_text) # replace multiple spaces with one spac
e
  removed_text = re.sub('\w*:', '', removed_text) # remove words ending with : ex- tex
t:
  removed_text = re.sub('[^\w\s]','',removed_text)

  return removed_text
```

In [11]:

```
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
```

In [12]:

```
def initial_preprocess_data(data):
  email_list = []
  subject_list = []
  processed_text_list = []

  text ,email = getEmail(data)
  email_list = email

  text, subject = getSubject(text)
  subject_list= subject

  text = removeLines(text)
  text = decontracted(text)

  return (text, email_list, subject_list)
```

In [13]:

```
def chunking(text):
  parse_tree = nltk.ne_chunk(nltk.tag.pos_tag(text.split()))
  leaf_list = []

  for t in parse_tree.subtrees():
    leaf_list = list(t)
    break


  final_str = ""
  for x in leaf_list:
    if type(x) == Tree:
      if x.label() == 'PERSON': # ignore if Tree is PERSON
        continue
      final_str += "_".join([val for val,label in x.leaves()])
    else:
      final_str += x[0]
    final_str += " "

  return final_str.strip()
```

In [14]:

```
def process_after_chunking(data):
  data = re.sub('[0-9]+'," ",data) #remove all digits
  data = re.sub('\b_\S+_\b|\b_\S+\b|\b\S+_\b'," ",data) #remove words like _abc, abc_,
 _abc_

  #remove one letter words : eg=>d_berlin
  clean_str = ""
  for word in data.split(" "):
    if len(word.split("_")) == 1:
      clean_str += word+" "
      continue

    temp_str = [w for w in word.split("_") if len(w)>2]
    clean_str += "_".join(temp_str)+" "

  clean_str = clean_str.strip()

  # convert all words into lower case
  # remove len(word)>=15 and len(word)<=2
  res_str = ""
  for word in clean_str.split(" "):
    word = word.lower()
    if len(word)>=15 or len(word)<=2:
      word = ""
    res_str += word+" "

  res_str = res_str.strip()

  # replace all the words except "A-Za-z_" with space
  clean_sentence = ""
  for word in res_str.split(" "):
    match_list = re.findall("^[A-Za-z_]+$",word)
    if len(match_list) == 0:
      continue
    else:
      clean_sentence += word + " "

  clean_sentence = clean_sentence.strip()

  return clean_sentence
```

In [62]:

```
def preprocess(input_text):
    """Do all the Preprocessing as shown above and
    return a tuple contain preprocess_email,preprocess_subject,preprocess_text for that
Text_data"""
    text, processed_email_list, processed_subject_list = initial_preprocess_data(input_
text)
    text = chunking(text)
    clean_input_text = process_after_chunking(text)


    return (processed_email_list,processed_subject_list,clean_input_text)
```

Test for file : alt.atheism_49960

In [65]:

```
fileName = "alt.atheism_49960.txt"
with open("data/"+ fileName,"r",encoding="ISO-8859-1") as textFile:
    text = textFile.read()
    test_mail, test_sub, test_text = preprocess(text)
    print("EMAIL: ", test_mail)
    print("SUBJECT: ", test_sub)
    print("TEXT: ", test_text)
```

```
EMAIL:  mantis co uk netcom mantis co uk
SUBJECT:   AltAtheism FAQ  Atheist Resources
TEXT:  archive atheism resources alt atheism archive resources last decemb
er usa freedom from religion foundation darwin fish bumper stickers and as
sorted other atheist paraphernalia are available from the the write ffrf b
ox madison evolution designs evolution designs sell the darwin fish fish s
ymbol like the ones christians stick their cars but with feet and the word
written inside the deluxe moulded plastic fish postpaid the write evolutio
n designs north hollywood people the area can get from try mailing for net
people who directly the price per fish american atheist press aap publish 
various atheist books critiques the bible lists biblical contradictions an
d one such book the bible_handbook and foote press isbn edition bible cont
radictions absurdities atrocities immoralities contains ball the aap based
the king version the press box prometheus books_sell books including write
east amherst street york alternate address prometheus books african americ
ans for humanism organization promoting black secular humanism and uncover
ing the history black freethought they publish quarterly newsletter aah ex
aminer write norm allen for box united press association national secular 
society street holloway road london london british society lamb red lion s
quare london fax the publish the freethinker monthly magazine founded germ
any ibka der und berlin germany ibka publish und zur zeit politisches jour
nal der und ibka postfach berlin germany for atheist books write ucherdien
st der hannover germany books_fiction thomas disch the story the ultimate 
proof that exists all characters and events are fictitious any similarity 
living dead gods well walter canticle for leibowitz one gem this post atom
ic doomsday novel the monks who spent their lives copying blueprints from 
filling the sheets paper with ink and leaving white lines and letters edga
r pangborn atomic doomsday novel set clerical states the church for exampl
e forbids that anyone produce describe use any substance containing atoms 
philip dick wrote many philosophical and thought provoking short stories a
nd novels his stories are bizarre times but very approachable wrote mainly
but wrote about people truth and religion rather than technology although 
often believed that had met some sort god remained sceptical amongst his n
ovels the following are some fallible alien deity summons group earth craf
tsmen and women remote planet raise giant cathedral from beneath the ocean
s when the deity begins demand faith from the earthers pot healer unable c
omply polished ironic and amusing novel maze for its description technolog
y based religion valis the schizophrenic hero searches for the hidden myst
eries gnostic christianity after reality fired into his brain pink laser b
eam unknown but possibly divine origin accompanied his dogmatic and dismis
sively atheist friend and assorted other odd characters the god invades ma
king young woman pregnant she returns from another star system unfortunate
ly she terminally ill and must assisted dead man whose brain wired hour ea
sy listening music margaret atwood the handmaid story based the premise th
at the congress mysteriously assassinated and quickly take charge the nati
on set right again the book the diary woman life she tries live under the 
new christian theocracy women right own property revoked and their bank ac
counts are closed sinful luxuries are outlawed and the radio only used for
readings from the bible crimes are punished doctors who performed legal ab
ortions the old world are hunted down and hanged writing style difficult g
et used first but the tale grows more and more chilling goes various autho
rs the bible this somewhat dull and rambling work has often been criticize
d however probably worth reading only that you know what all the fuss abou
t exists many different versions make sure you get the one true version fi
ction peter rosa_vicars christ_bantam press although seems even catholic t
his very enlighting history papal immoralities adulteries fallacies etc ge
rman gottes erste dunkle seite des knaur michael martin philosophical just
ification temple university press philadelphia usa detailed and scholarly 
justification atheism contains outstanding appendix defining terminology a
nd usage this tendentious area both for negative atheism the non belief th
e existence god and also for positive atheism the belief the non existence
```

```
god includes great refutations the most challenging arguments for god part
icular attention paid refuting contempory theists such and swinburne pages
isbn the case_against christianity temple university press comprehensive c
ritique christianity which considers the best contemporary defences christ
ianity and demonstrates that they are unsupportable and incoherent pages i
sbn james turner_without god without creed the press baltimore usa subtitl
ed the origins unbelief america examines the way which unbelief became mai
nstream alternative world view focusses the period and while considering f
rance and britain the emphasis american and particularly new_england devel
opments neither religious history secularization atheism without rather th
e intellectual history the fate single idea the belief that exists pages i
sbn george seldes the great thoughts usa dictionary quotations different k
ind concentrating statements and writings which explicitly implicitly pres
ent the person philosophy and world view includes obscure opinions from ma
ny people for some popular observations traces the way which various peopl
e expressed and twisted the idea over the centuries quite number the quota
tions are derived from what religion and religion pages isbn richard swinb
urne the existence this book the second volume trilogy that began with the
coherence theism this work swinburne attempts construct series inductive a
rguments for the existence god his arguments which are somewhat tendentiou
s and rely upon the imputation late century western christian values and a
esthetics god which supposedly simple can conceived were decisively reject
ed mackie the miracle theism the revised edition the existence god_swinbur
ne includes appendix which makes somewhat incoherent attempt rebut mackie 
the miracle theism oxford this volume contains comprehensive review the pr
incipal arguments for and against the existence god ranges from the classi
cal philosophical positions through the moral arguments newman_kant and th
e recent restatements the classical theses and swinburne also addresses th
ose positions which push the concept god beyond the realm the rational suc
h those and well replacements for god such axiarchism the book delight rea
d less formalistic and better written than works and refreshingly direct w
hen compared with the hand waving james and religious persecution from anc
ient times the present day and not only number norm allen african american
the listing for for humanism above gordon stein anthology atheism and anth
ology covering wide range subjects including the devil_evil and morality a
nd the history freethought comprehensive bibliography edmund cohen the min
d the study why people become christian and what effect has them net_resou
rces there small mail based archive server mantis which carries archives o
ld alt atheism moderated articles and assorted other files for more inform
ation send mail saying help send atheism index and will mail back reply ma
thew
```

Build New dataframe from cleaned data

In [66]:

```
text_list = []
email_list = []
subject_list = []

for row in tqdm(data["text"].values):
  email, sub, txt = preprocess(row)
  text_list.append(txt)
  email_list.append(email)
  subject_list.append(sub)
```

```
100%|██████████| 18828/18828 [20:58<00:00, 14.96it/s]
```

In [ ]:

```
df = pd.DataFrame({"text":data["text"].values, "class":data["class"].values, "preprocessed_text" : text_list, "preprocessed_email":email_list,"preprocessed_subject":subject_list})
```

Model 1 : Word Encoding

Load Data and Tokenize

In [19]:

```
print(df.columns)
# concat into one column for training
df["train_text"] = df["preprocessed_text"] + df["preprocessed_email"]+ df["preprocessed_subject"]

X = df["train_text"].values
y = df["class"].values

oneEncoder = OneHotEncoder().fit(y.reshape(-1,1))
y = oneEncoder.transform(y.reshape(-1,1)).toarray()
# le.inverse_transform()

X_train, X_test, y_train, y_test = train_test_split(X,y ,test_size=0.25, random_state = 0, stratify=y)

print("X Train : (%dx%d) "%( len(X_train), len(X_train[0])))
print("Y Train : (%dx%d) "%( len(y_train), len(y_train[0])))
```

```
Index(['text', 'class', 'preprocessed_text', 'preprocessed_email',
       'preprocessed_subject'],
      dtype='object')
X Train : (14121x867)
Y Train : (14121x20)
```

In [20]:

```
# df["train_text"]
```

In [21]:

```
#use tf.tokenizer : remove '_' from filters as we need words joined by _ (new_york)
tokenizer = Tokenizer(lower=True, split=' ',filters='!"#$%&()*+,-./:;<=>?@[\\]^`{|}~\t\n')
tokenizer.fit_on_texts(X_train)

# Encode training data sentences into sequences
train_sequences = tokenizer.texts_to_sequences(X_train)
test_sequences = tokenizer.texts_to_sequences(X_test)
```

In [22]:

```
vocab_size = len(tokenizer.word_index) + 1
print("Learned Vocab has size : ",vocab_size)
maxlen = max([len(x) for x in train_sequences])
print("Maximum len of words in train_data is: ", maxlen)
```

```
Learned Vocab has size :  104876
Maximum len of words in train_data is:  8791
```

In [23]:

```
# Pad the sequences based on maxLen
train_padded = pad_sequences(train_sequences, padding='post', truncating='post', maxlen
=maxlen)
test_padded = pad_sequences(test_sequences, padding='post', truncating='post', maxlen=m
axlen)
```

Create Embedding Matrix

In [24]:

```
#Load GLove vector
!gdown --id "1MRsz-18c7i0nOMwYRG-CBbE65FLh58EM"
```

```
Downloading...
From: https://drive.google.com/uc?id=1MRsz-18c7i0nOMwYRG-CBbE65FLh58EM
To: /content/glove_vectors
128MB [00:01, 104MB/s]
```

In [25]:

```
with open('glove_vectors', 'rb') as f:
  glove_vector = pickle.load(f)
  glove_words = set(glove_vector.keys())
```

In [26]:

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

In [27]:

```
print("Shape of Embedding Matrix: %d x %d"%(len(embedding_matrix),len(embedding_matrix[
0])))
```

```
Shape of Embedding Matrix: 104876 x 300
```

Build -> Model 1

In [28]:

```
class Metrics_Callback(tf.keras.callbacks.Callback):
  def __init__(self,x_val,y_val):
    self.x_val = x_val
    self.y_val = y_val

  def on_train_begin(self, logs={}):
    self.history = {"micro_f1":[]}

  def on_epoch_end(self, epoch, logs={}):
    # self.model.predict_classes
    y_pred = self.model.predict(self.x_val)

    y_true_labels = [np.argmax(x) for x in self.y_val]
    y_pred_labels = [np.argmax(x) for x in y_pred]
    micro_f1_s = f1_score(y_true_labels,y_pred_labels, average='micro')
    self.history["micro_f1"].append(micro_f1_s)
```

In [29]:

```
input = Input(shape = (maxlen))
embedding_layer = Embedding(vocab_size, 300, weights=[embedding_matrix], input_length=maxlen, trainable=False,name="Embedding_Layer")(input)

branch_a = Conv1D(filters=4, kernel_size=9, activation= 'relu',name="Branch_A")(embedding_layer)
branch_b = Conv1D(filters=8, kernel_size=9, activation= 'relu',name="Branch_B")(embedding_layer)
branch_c = Conv1D(filters=16, kernel_size=9, activation= 'relu',name="Branch_C")(embedding_layer)
concat1 = Concatenate(axis=2,name="Concat")([branch_a, branch_b, branch_c])

max_pool_1 = MaxPooling1D(pool_size=10,strides=1, padding='valid')(concat1)

branch_a_2 = Conv1D(filters=8, kernel_size=9, activation= 'relu')(max_pool_1)
branch_b_2 = Conv1D(filters=4, kernel_size=9, activation= 'relu')(max_pool_1)
branch_c_2 = Conv1D(filters=2, kernel_size=9, activation= 'relu')(max_pool_1)

concat2 = Concatenate()([branch_a_2, branch_b_2, branch_c_2])

max_pool_2 = MaxPooling1D(pool_size=10,strides=1, padding='valid')(concat2)

conv_filter = Conv1D(filters=32, kernel_size=9, activation= 'relu')(max_pool_2)

flatten = Flatten()(conv_filter)

drop = Dropout(0.4)(flatten)

dense1 = Dense(40,activation='relu',kernel_initializer=tf.keras.initializers.he_normal())

output = Dense(20, activation='softmax')(drop)
```

In [30]:

```
model = keras.Model(input,output)
```

In [31]:

```
model.summary()
```

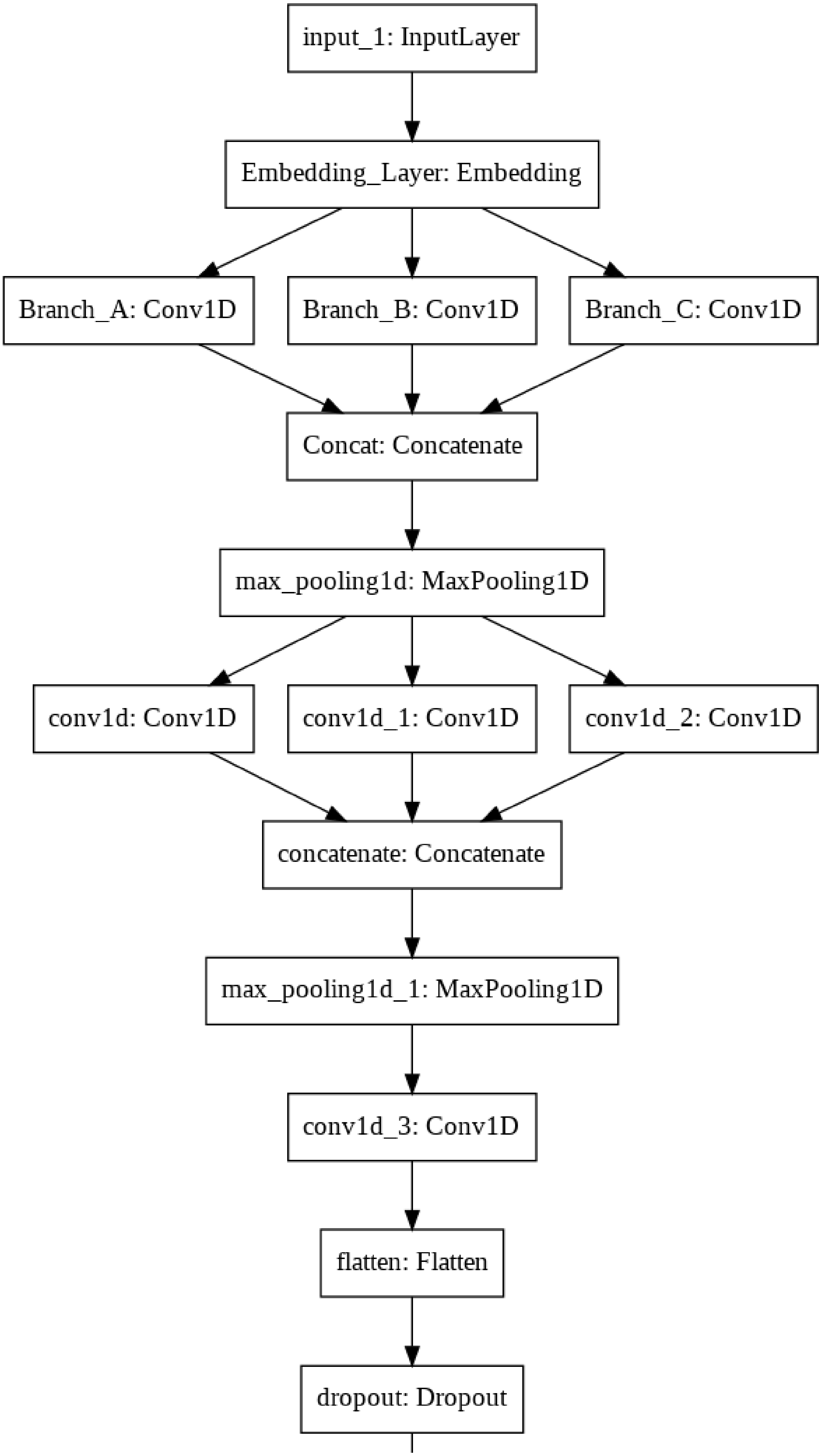
```
Model: "model"
________________________________________________________________________
__________________________
Layer (type)                    Output Shape         Param #     Connect
ed to
========================================================================
==========================
input_1 (InputLayer)            [(None, 8791)]       0
________________________________________________________________________
__________________________
Embedding_Layer (Embedding)     (None, 8791, 300)    31462800    input_1
[0][0]
________________________________________________________________________
__________________________
Branch_A (Conv1D)               (None, 8783, 4)      10804       Embeddi
ng_Layer[0][0]
________________________________________________________________________
__________________________
Branch_B (Conv1D)               (None, 8783, 8)      21608       Embeddi
ng_Layer[0][0]
________________________________________________________________________
__________________________
Branch_C (Conv1D)               (None, 8783, 16)     43216       Embeddi
ng_Layer[0][0]
________________________________________________________________________
__________________________
Concat (Concatenate)            (None, 8783, 28)     0           Branch_
A[0][0]
                                                                 Branch_
B[0][0]
                                                                 Branch_
C[0][0]
________________________________________________________________________
__________________________
max_pooling1d (MaxPooling1D)    (None, 8774, 28)     0           Concat
[0][0]
________________________________________________________________________
__________________________
conv1d (Conv1D)                 (None, 8766, 8)      2024        max_poo
ling1d[0][0]
________________________________________________________________________
__________________________
conv1d_1 (Conv1D)               (None, 8766, 4)      1012        max_poo
ling1d[0][0]
________________________________________________________________________
__________________________
conv1d_2 (Conv1D)               (None, 8766, 2)      506         max_poo
ling1d[0][0]
________________________________________________________________________
__________________________
concatenate (Concatenate)       (None, 8766, 14)     0           conv1d
[0][0]
                                                                 conv1d_
1[0][0]
                                                                 conv1d_
2[0][0]
________________________________________________________________________
__________________________
max_pooling1d_1 (MaxPooling1D)  (None, 8757, 14)     0           concate
nate[0][0]
________________________________________________________________________
```

```
__________________________
conv1d_3 (Conv1D)               (None, 8749, 32)     4064        max_poo
ling1d_1[0][0]
________________________________________________________________________
__________________________
flatten (Flatten)               (None, 279968)       0           conv1d_
3[0][0]
________________________________________________________________________
__________________________
dropout (Dropout)               (None, 279968)       0           flatten
[0][0]
________________________________________________________________________
__________________________
dense_1 (Dense)                 (None, 20)           5599380     dropout
[0][0]
========================================================================
==========================
Total params: 37,145,414
Trainable params: 5,682,614
Non-trainable params: 31,462,800
________________________________________________________________________
__________________________
```

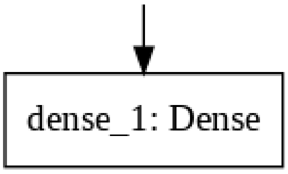
In [ ]:

```
plot_model(model)
```

Out[ ]:



input_1: InputLayer
Embedding_Layer: Embedding
Branch_A: Conv1D
Branch_B: Conv1D
Branch_C: Conv1D
Concat: Concatenate
max_pooling1d: MaxPooling1D
conv1d: Conv1D
conv1d_1: Conv1D
conv1d_2: Conv1D
concatenate: Concatenate
max_pooling1d_1: MaxPooling1D
conv1d_3: Conv1D
flatten: Flatten
dropout: Dropout



In [32]:

```
Metrics = Metrics_Callback(test_padded,y_test)

EarlyStop = EarlyStopping(monitor='accuracy',mode='max')

filePath = "best_model_1.h5"
model_checkpoint_callback = ModelCheckpoint(filepath=filePath,save_best_only=True,monit
or='val_accuracy',mode='max')

!rm -rf logs/*
log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1
, write_graph=True)
```

In [33]:

```
model.compile(optimizer="adam",
              loss="categorical_crossentropy",
              metrics=["accuracy"])
```

In [34]:

```
model.fit(train_padded,y_train,
  epochs=10,
  validation_data=(test_padded,y_test),
  callbacks = [Metrics, EarlyStop, model_checkpoint_callback, tensorboard_callback])
```

```
Epoch 1/10
442/442 [==============================] - 126s 268ms/step - loss: 2.5556
- accuracy: 0.1871 - val_loss: 1.6482 - val_accuracy: 0.4483
Epoch 2/10
442/442 [==============================] - 118s 267ms/step - loss: 1.3164
- accuracy: 0.5447 - val_loss: 1.2525 - val_accuracy: 0.6125
Epoch 3/10
442/442 [==============================] - 118s 268ms/step - loss: 0.8579
- accuracy: 0.7005 - val_loss: 1.1925 - val_accuracy: 0.6584
Epoch 4/10
442/442 [==============================] - 118s 268ms/step - loss: 0.6004
- accuracy: 0.7891 - val_loss: 1.1475 - val_accuracy: 0.6877
Epoch 5/10
442/442 [==============================] - 119s 268ms/step - loss: 0.4760
- accuracy: 0.8310 - val_loss: 1.2408 - val_accuracy: 0.6909
Epoch 6/10
442/442 [==============================] - 118s 268ms/step - loss: 0.3767
- accuracy: 0.8720 - val_loss: 1.3851 - val_accuracy: 0.6913
Epoch 7/10
442/442 [==============================] - 118s 268ms/step - loss: 0.4587
- accuracy: 0.8521 - val_loss: 1.4809 - val_accuracy: 0.7143
Epoch 8/10
442/442 [==============================] - 118s 268ms/step - loss: 0.2655
- accuracy: 0.9103 - val_loss: 1.4058 - val_accuracy: 0.7289
Epoch 9/10
442/442 [==============================] - 118s 267ms/step - loss: 0.2079
- accuracy: 0.9314 - val_loss: 1.5852 - val_accuracy: 0.7270
Epoch 10/10
442/442 [==============================] - 118s 268ms/step - loss: 0.1819
- accuracy: 0.9402 - val_loss: 1.6568 - val_accuracy: 0.7166
```

Out[34]:

```
<tensorflow.python.keras.callbacks.History at 0x7f14c9ff36d0>
```

In [35]:

```
Metrics.history
```

Out[35]:

```
{'micro_f1': [0.4482685362226471,
  0.6124920331421287,
  0.6583811344805609,
  0.6876991714467814,
  0.6908859145952836,
  0.6913108136817506,
  0.7142553643509666,
  0.7289143828340768,
  0.7270023369449755,
  0.7165923093265348]}
```

In [37]:

```
# %tensorboard --logdir logs/fit
```

Model 2 : Character Encoding

Load Data and Tokenize

In [38]:

```
print(df.columns)
# concat into one column for training
df["train_text"] = df["preprocessed_text"] + df["preprocessed_email"]+ df["preprocessed
_subject"]

X = df["train_text"].values
y = df["class"].values

oneEncoder = OneHotEncoder().fit(y.reshape(-1,1))
y = oneEncoder.transform(y.reshape(-1,1)).toarray()
# le.inverse_transform()

X_train, X_test, y_train, y_test = train_test_split(X,y ,test_size=0.25, random_state =
0, stratify=y)

print("X Train : (%dx%d) "%( len(X_train), len(X_train[0])))
print("Y Train : (%dx%d) "%( len(y_train), len(y_train[0])))
```

```
Index(['text', 'class', 'preprocessed_text', 'preprocessed_email',
       'preprocessed_subject', 'train_text'],
      dtype='object')
X Train : (14121x867)
Y Train : (14121x20)
```

In [39]:

```
tkn = Tokenizer(lower=True, char_level=True, split=' ', filters='!"#$%&()*+,-./:;<=>?@[
\\]^`{|}~\t\n')

# update tkn.word_index
alphabet = "abcdefghijklmnopqrstuvwxyz_"
char_dict = {}
for i, char in enumerate(alphabet):
    char_dict[char] = i

# Use char_dict to replace the tk.word_index
tkn.word_index = char_dict.copy()
```

In [40]:

```
# text to int sequence
train_sequences_char = tkn.texts_to_sequences(X_train)
test_sequences_char = tkn.texts_to_sequences(X_test)

max_len_char = max([len(x) for x in train_sequences_char])

# Pad the sequences based on maxLen
train_padded_char = pad_sequences(train_sequences_char, padding='post', truncating='pos
t', maxlen=int(max_len_char/1.5))
test_padded_char = pad_sequences(test_sequences_char, padding='post', truncating='post'
, maxlen=int(max_len_char/1.5))
```

In [41]:

```
len(train_padded_char[0])
```

Out[41]:

```
31392
```

Create Embedding Matrix

In [42]:

```
#download char-embeddings glove
!wget https://raw.githubusercontent.com/minimaxir/char-embeddings/master/glove.840B.300
d-char.txt
```

```
--2021-02-25 10:15:32--  https://raw.githubusercontent.com/minimaxir/char-
embeddings/master/glove.840B.300d-char.txt
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.19
9.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.19
9.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 265233 (259K) [text/plain]
Saving to: ‘glove.840B.300d-char.txt’

glove.840B.300d-cha 100%[===================>] 259.02K  --.-KB/s    in 0.0
3s

2021-02-25 10:15:32 (7.76 MB/s) - ‘glove.840B.300d-char.txt’ saved [26523
3/265233]
```

In [43]:

```
# Ref : https://stackoverflow.com/questions/37793118/load-pretrained-glove-vectors-in-p
ython
def loadGloveModel(fileName):
    f = open(fileName,'r')
    gloveModel = {}
    for line in f:
        line_words = line.split(" ")
        word = line_words[0]
        wordEmbedding = np.array([float(value) for value in line_words[1:]])
        gloveModel[word] = wordEmbedding
    print(len(gloveModel)," words loaded!")
    return gloveModel
```

In [44]:

```
char_glove = loadGloveModel("glove.840B.300d-char.txt")
print(len(char_glove["a"]))
```

```
94  words loaded!
300
```

In [45]:

```
char_embedding_matrix = np.zeros((27, 300))
for word, i in tkn.word_index.items():
        embedding_vector = char_glove.get(word)
        if embedding_vector is not None:
                char_embedding_matrix[i] = embedding_vector
```

Build -> Model 2

In [ ]:

```
print(max_len_char)
print(int(max_len_char/1.5))
```

```
47088
31392
```

In [46]:

```
input = Input(shape = (int(max_len_char/1.5)))
embedding_layer = Embedding(27, 300, weights=[char_embedding_matrix], input_length=int(
max_len_char/1.5), trainable=False)(input)

branch_a = Conv1D(filters=16, kernel_size=8, activation= 'relu')(embedding_layer)
branch_b = Conv1D(filters=16, kernel_size=12, activation= 'relu')(branch_a)
max_pool_1 = MaxPooling1D()(branch_b)

branch_a_2 = Conv1D(filters=16, kernel_size=12, activation= 'relu')(max_pool_1)
branch_b_2 = Conv1D(filters=16, kernel_size=12, activation= 'relu')(branch_a_2)
max_pool_2 = MaxPooling1D()(branch_b_2)

flatten = Flatten()(max_pool_2)
drop = Dropout(0.5)(flatten)
dense1 = Dense(128,activation='relu')(drop)
output = Dense(20, activation='softmax')(dense1)
```

In [47]:

```
model2 = keras.Model(input,output)
```

In [48]:

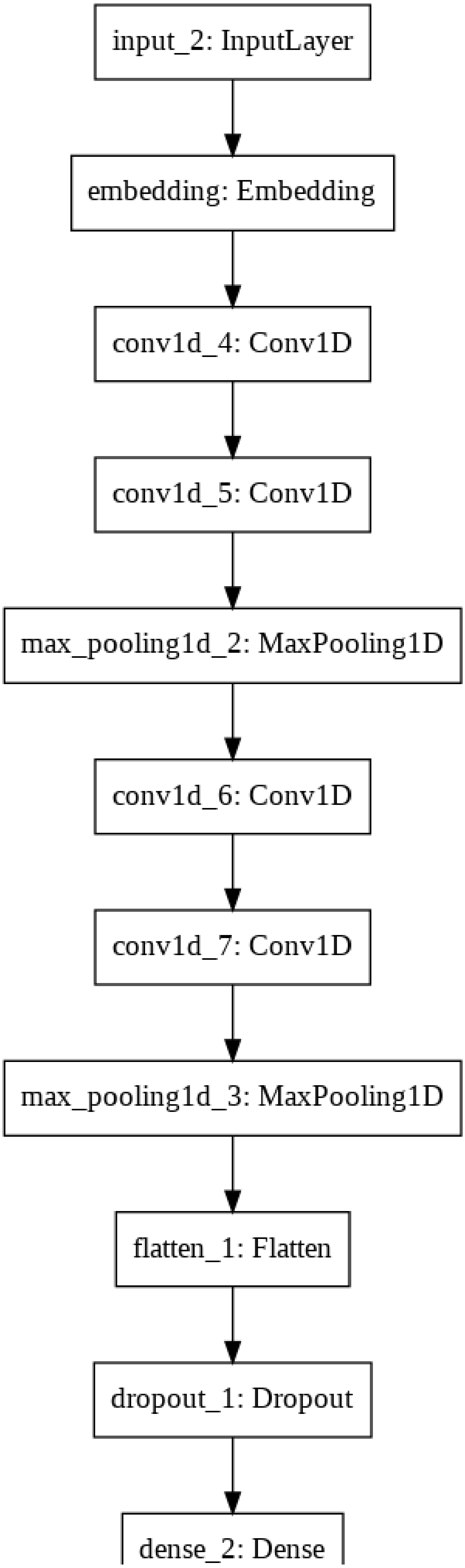
```
model2.summary()
```

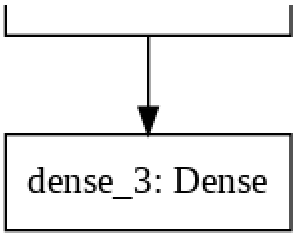
```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_2 (InputLayer)         [(None, 31392)]           0         
_________________________________________________________________
embedding (Embedding)        (None, 31392, 300)        8100      
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 31385, 16)         38416     
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 31374, 16)         3088      
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 15687, 16)         0         
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 15676, 16)         3088      
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 15665, 16)         3088      
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 7832, 16)          0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 125312)            0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 125312)            0         
_________________________________________________________________
dense_2 (Dense)              (None, 128)               16040064  
_________________________________________________________________
dense_3 (Dense)              (None, 20)                2580      
=================================================================
Total params: 16,098,424
Trainable params: 16,090,324
Non-trainable params: 8,100
_________________________________________________________________
```

In [57]:

```
plot_model(model2)
```

Out[57]:


input_2: InputLayer
embedding: Embedding
conv1d_4: Conv1D
conv1d_5: Conv1D
max_pooling1d_2: MaxPooling1D
conv1d_6: Conv1D
conv1d_7: Conv1D
max_pooling1d_3: MaxPooling1D
flatten_1: Flatten
dropout_1: Dropout
dense_2: Dense



In [50]:

```
Metrics = Metrics_Callback(test_padded_char,y_test)

# EarlyStop = EarlyStopping(monitor='accuracy',mode='max')

filePath = "best_model_2.h5"
model_checkpoint_callback = ModelCheckpoint(filepath=filePath,save_best_only=True,monit
or='val_accuracy',mode='max')

!rm -rf logs/*
log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1
, write_graph=True)
```

In [51]:

```
model2.compile(optimizer="adam",
              loss="categorical_crossentropy",
              metrics=["accuracy"])
```

In [52]:

```
model2.fit(train_padded_char,y_train,
  epochs=10,
  validation_data=(test_padded_char,y_test),
  callbacks = [Metrics, model_checkpoint_callback, tensorboard_callback])
```

```
Epoch 1/10
442/442 [==============================] - 262s 589ms/step - loss: 3.0026
- accuracy: 0.0482 - val_loss: 2.9914 - val_accuracy: 0.0529
Epoch 2/10
442/442 [==============================] - 261s 590ms/step - loss: 2.9914
- accuracy: 0.0528 - val_loss: 2.9905 - val_accuracy: 0.0529
Epoch 3/10
442/442 [==============================] - 261s 590ms/step - loss: 2.9906
- accuracy: 0.0521 - val_loss: 2.9904 - val_accuracy: 0.0531
Epoch 4/10
442/442 [==============================] - 260s 589ms/step - loss: 2.9920
- accuracy: 0.0502 - val_loss: 2.9903 - val_accuracy: 0.0527
Epoch 5/10
442/442 [==============================] - 261s 590ms/step - loss: 2.9897
- accuracy: 0.0532 - val_loss: 2.9903 - val_accuracy: 0.0527
Epoch 6/10
442/442 [==============================] - 260s 589ms/step - loss: 2.9893
- accuracy: 0.0513 - val_loss: 2.9904 - val_accuracy: 0.0529
Epoch 7/10
442/442 [==============================] - 261s 590ms/step - loss: 2.9900
- accuracy: 0.0475 - val_loss: 2.9903 - val_accuracy: 0.0529
Epoch 8/10
442/442 [==============================] - 260s 589ms/step - loss: 2.9905
- accuracy: 0.0520 - val_loss: 2.9903 - val_accuracy: 0.0527
Epoch 9/10
442/442 [==============================] - 261s 590ms/step - loss: 2.9896
- accuracy: 0.0510 - val_loss: 2.9904 - val_accuracy: 0.0527
Epoch 10/10
442/442 [==============================] - 260s 588ms/step - loss: 2.9910
- accuracy: 0.0518 - val_loss: 2.9903 - val_accuracy: 0.0529
```

Out[52]:

```
<tensorflow.python.keras.callbacks.History at 0x7f14c9b46e90>
```

In [53]:

```
Metrics.history
```

Out[53]:

```
{'micro_f1': [0.052899936265137025,
  0.052899936265137025,
  0.053112385808370514,
  0.052687486721903556,
  0.052687486721903556,
  0.052899936265137025,
  0.052899936265137025,
  0.052687486721903556,
  0.052687486721903556,
  0.052899936265137025]}
```

In [55]:

```
%tensorboard --logdir logs/fit
#
```

In [67]:

```
!jupyter nbconvert --to html "/content/Document_Classification.ipynb"
```

```
[NbConvertApp] Converting notebook /content/Document_Classification.ipynb
to html
[NbConvertApp] Writing 973401 bytes to /content/Document_Classification.ht
ml
```

In [ ]:

```

```