```
import re
import os
from nltk import ne_chunk, pos_tag, word_tokenize
from nltk.tree import Tree
import pickle
from multiprocessing import Pool
import numpy as np
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from keras.initializers import glorot_normal
from keras.utils import to_categorical
import datetime
```

```
import tensorflow
print(tensorflow.__version__)
```

```
    2.4.0
```

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

```
    --2020-12-27 05:02:58--  http://nlp.stanford.edu/data/glove.6B.zip
    Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
    Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected.
    HTTP request sent, awaiting response... 302 Found
    Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
    --2020-12-27 05:02:58--  https://nlp.stanford.edu/data/glove.6B.zip
    Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
    HTTP request sent, awaiting response... 301 Moved Permanently
    Location: http://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
    --2020-12-27 05:02:58--  http://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
    Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
    Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:80... c
    HTTP request sent, awaiting response... 200 OK
    Length: 862182613 (822M) [application/zip]
    Saving to: ‘glove.6B.zip’

    glove.6B.zip        100%[===================>] 822.24M  2.02MB/s    in 6m 26s

    2020-12-27 05:09:25 (2.13 MB/s) - ‘glove.6B.zip’ saved [862182613/862182613]

    Archive:  glove.6B.zip
      inflating: glove.6B.50d.txt
      inflating: glove.6B.100d.txt
      inflating: glove.6B.200d.txt
      inflating: glove.6B.300d.txt
```

```
def get_emails(text):
    pattern = re.compile(r'[a-zA-Z0-9-_.]+@([a-zA-Z0-9-_]+\.)+\w{2,}')
    emails = pattern.finditer(text)
    text = re.sub(r'[a-zA-Z0-9-_.]+@([a-zA-Z0-9-_]+\.)+\w{2,}',' ',text)
    return (emails,text)
```

```
def extract_domain(domains):
    domain_list= [domain.group() for domain in domains]
    top_domain_set = set()
    for domain in domain_list:
        idx = domain.find('@')
        top_domains = domain[idx+1:].split('.')
```

```
            top_domains = [i for i in top_domains if len(i)>=3 and i.strip().lower() != 'com' ]
            top_domain_set = top_domain_set.union(set(top_domains))
        return list(top_domain_set)


    def extract_subject(text):
        subject = re.search(r'subject:.+', text,re.IGNORECASE)
        text = re.sub(r'subject:.+','', text, flags = re.IGNORECASE)
        subject = subject.group()

        idx = subject.find(':')
        subject= subject[idx+1:]
        replacements =[(r'Re:',''),(r'\r',''),(r'\n',''),(r'[^\w\* \.]','')]
        for look, replace in replacements:
            subject = re.sub(look, replace, subject, flags = re.IGNORECASE)

        return (subject.strip(),text)


    def clean(text):
        text = re.sub(r'From:(.+\n)+','', text, flags = re.IGNORECASE)
        text = re.sub(r'write.+to:(.+\n)+','',text, flags = re.IGNORECASE)
        text = re.sub(r'\n',' ',text, flags = re.IGNORECASE)
        text = re.sub(r'\t',' ',text, flags = re.IGNORECASE)
        text = re.sub(r'\\',' ',text, flags = re.IGNORECASE)
        text = re.sub(r'-',' ',text, flags = re.IGNORECASE)
        text = re.sub(r'\w+:',' ',text, flags = re.IGNORECASE)

        return text


    def expand(text):
        replacements =[(r"\'ve",' have'),(r"\'t",' not'),(r"'m",' am'),(r"\'re",' are'),(r"\'ll","
                       (r"\'s",' is')]

        for look, replace in replacements:
            text = re.sub(look, replace, text, flags = re.IGNORECASE)

        return text


    def remove_braces(start, end, text):
        i = 0
        count = 0
        stack = []

        for i in range(len(text)):

            if text[i]==start:
                if count == 0:
                    a = i
                count+=1
                stack.append(text[i])
                continue

            if text[i]==end:
                if count!=0:
                    count-=1
                    while stack[-1]!=start:
                        stack.pop()
                    stack.pop()
                    continue

            stack.append(text[i])
        stack = ''.join(stack)
        return stack


    def chunk(text):
        ms = list([])

        parse_tree = ne_chunk(pos_tag(word_tokenize(text)), binary=True)
        for elt in parse_tree:

            if isinstance(elt, Tree):
                if elt.label() != 'PERSON':
                    ms.append("_".join(w for w, t in elt))
            else:
                ms.append(elt[0])

        return ' '.join(ms)


    def strip(m):
        v = str(m.group(0))
        return v.strip('_')


    def clean_after_tagging(text):
        text = re.sub(r'\d',' ',text)
        text = re.subn(r'(\b_\w+_*\b|\b_*\w+_\b)', lambda x: x.group(0).strip('_'), text)
        text = re.subn(r'(\w{1,2})_\w+', lambda x: x.group(0)[x.group(0).index('_')+1:], text[0])
        text = re.sub(r'\b[a-zA-Z]{,2}\b', ' ', text[0])
        text = re.sub(r'\b[a-zA-Z]{15,}\b', ' ', text)
        text = re.sub(r'[^a-zA-Z_]', ' ', text)
        text = text.lower().strip()
        return text


    def cleaner(files):
        top_domains_list = list([])
        subject_list = list([])
        text_list = list([])

        for file in files[:-1]:
            with open(os.path.join('documents',file) ,'rb') as f:
                data = f.read().decode("latin-1")
                domains,text = get_emails(data)
                top_domains = extract_domain(domains)
                subject,text = extract_subject(text)
                text = clean(text)
                text = expand(text)
                text = remove_braces('(',')',text)
                text = remove_braces('<','>',text)
                text = chunk(text)
                text = clean_after_tagging(text)

                top_domains_list.append(' '.join(top_domains).lower())
                subject_list.append(subject)
                text_list.append(text)

        data_dict = {
                    'top_domains_list' : top_domains_list,
                    'subject_list' : subject_list,
                    'text_list' : text_list
                    }

        with open(os.path.join('temp_files',str(files[-1])+'.pickle'),'wb') as handle:
            pickle.dump(data_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)

        return
```

```
len(os.listdir('documents'))
```

```
ext_drive = 'documents/'
tfiles = os.listdir(ext_drive)
quart = int(len(tfiles)/4)
train1 = tfiles[:quart]
train2 = tfiles[quart:(2*quart)]
train3 = tfiles[(2*quart):(3*quart)]
train4 = tfiles[(3*quart):]

train1.append('first')
train2.append('second')
train3.append('third')
train4.append('fourth')

trains = [train1, train2, train3, train4]
p = Pool(4)
p.map(cleaner, trains)
```

lets verify the shape of data

```
with open(os.path.join('documents','alt.atheism_49960.txt') ,'rb') as f:
    data = f.read().decode("latin-1")
    domains,text = get_emails(data)
    top_domains = extract_domain(domains)
    subject,text = extract_subject(text)
    text = clean(text)
    text = expand(text)
    text = remove_braces('(',')',text)
    text = remove_braces('<','>',text)
    text = chunk(text)
    text = clean_after_tagging(text)

    print(top_domains)
    print('='*50)
    print(subject)
    print('='*50)
    print(text)
```

```
['mantis', 'netcom']
==================================================
Alt.Atheism FAQ Atheist Resources
==================================================
archive   atheism   resources   alt   atheism   archive   resources   last      december
```

lets do modelling

```
len(data['top_domains_list'])
```

```
4707
```

```
X = list([]) # training data set
```

```
X = list([]) # training data set

files = ['first.pickle', 'second.pickle', 'third.pickle', 'fourth.pickle']

for file in files:
    with open(os.path.join('temp_files',file), 'rb') as f:
        data = pickle.load(f)
    for i in range(len(data['top_domains_list'])):
        mail = data['top_domains_list'][i]
        subject = data['subject_list'][i]
        text = data['text_list'][i]
        combined_text = ' '.join([mail, subject, text])
        X.append(combined_text)
```

```
len(X)
```

```
    18828
```

```
#char_indices = {'a':0,'b':1,'c':2,' ':3}
```

```
# def vectorize_sentences(data, char_indices):
#     X = []
#     for sentences in data:
#         x = [char_indices[w] for w in sentences]
#         x2 = np.eye(len(char_indices))[x]
#         print(x)
#         print(x2)
#         X.append(x2)
#     return (pad_sequences(X, maxlen=5))
```

```
# x=vectorize_sentences(['abc bc aa bbb cc','abc aa bc bbb cc'], char_indices)
```

```
#np.eye(3)[[0,1,2,0]]
```

```
Y_1 = [ y.split('_')[0].strip() for y in train1[:-1]]
Y_2 = [ y.split('_')[0].strip() for y in train2[:-1]]
Y_3 = [ y.split('_')[0].strip() for y in train3[:-1]]
Y_4 = [ y.split('_')[0].strip() for y in train4[:-1]]
```

```
Y = Y_1 + Y_2 +Y_3 +Y_4
```

```
with open(os.path.join('labels.pickle'), 'rb') as f:
        Y = pickle.load(f)
```

```
len(Y)
```

```
    18828
```

```
# we did train4[:-1] since
train4[-1]
```

```
le = preprocessing.LabelEncoder()
le.fit(Y)
```

```
    LabelEncoder()
```

```
def generate_tokenizer(expressions):
    tokenizer = Tokenizer(filters='!"#$%&()*+,-./:;<=>?@[\\]^``{|}~\t\n') #removing underscore
    tokenizer.fit_on_texts(expressions)
    return tokenizer


def max_length(lines):
    return max([len(s.split()) for s in lines])


def encode_text(tokenizer, lines, length):
    # integer encode
    encoded = tokenizer.texts_to_sequences(lines)
    # pad encoded sequences
    padded = pad_sequences(encoded, maxlen=length, padding='post')
    return padded


# lets define model
```

Prepare train data and test data for model 1

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25, random_state=42, str
```

```
tokenizer = generate_tokenizer(X_train)
```

```
length = max_length(X_train)
length
```

```
    8991
```

```
vocab_size = len(tokenizer.word_index)+1
vocab_size
```

```
    92443
```

```
X_train = encode_text(tokenizer, X_train, length)
```

```
X_test = encode_text(tokenizer, X_test, length)
```

```
X_train.shape
```

```
    (14121, 8991)
```

```
X_test.shape
```

```
    (4707, 8991)
```

```
y_train = le.transform(y_train)
y_test = le.transform(y_test)
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

prepare glove embeddings

```
# load glove embeddings


embeddings_index = {}
f = open(os.path.join('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))
```

```
     Found 400000 word vectors.
```

```
# initialize embedding matrix


embedding_matrix = np.zeros((len(tokenizer.word_index) + 1, 100))
for word, i in tokenizer.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.
        embedding_matrix[i] = embedding_vector
```

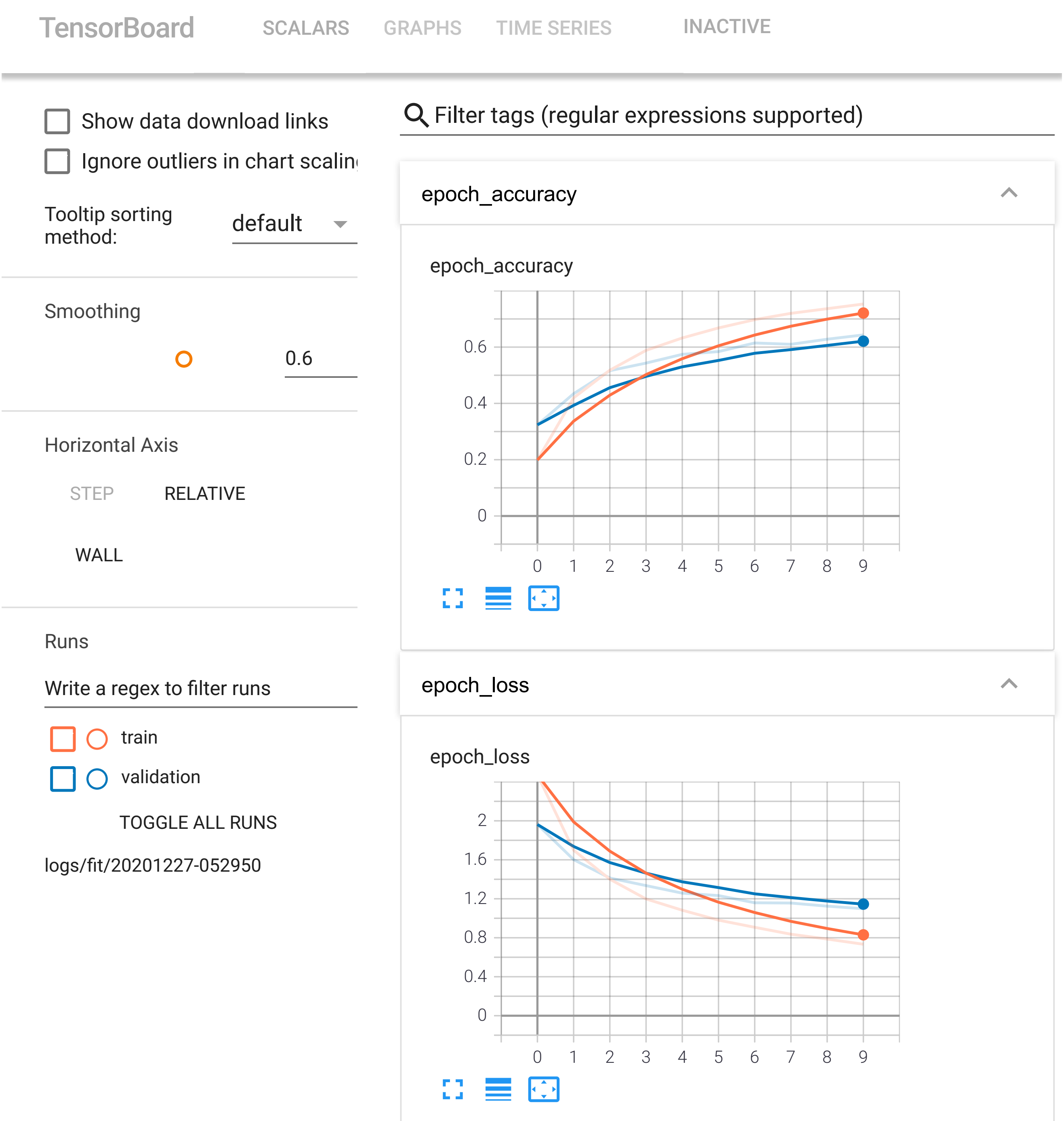
```
#importing layers
from keras.layers import Input
from keras.layers import Embedding
from keras.layers.merge import concatenate
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers import Dense
from keras.utils import to_categorical
from keras.utils.vis_utils import plot_model
from keras.models import Model
from keras.optimizers import Adam
from keras.callbacks import Callback, LearningRateScheduler, TensorBoard, EarlyStopping, Model
from sklearn.metrics import roc_auc_score, roc_curve, f1_score
from keras.regularizers import l2


class Customcallback(Callback):

    def __init__(self, x_test, y_test, model_file_path):
        self.x_test  = x_test
        self.y_test = y_test
        self.prev_val_acc = 0
        self.num_val_acc_dec = 0
        self.model_dir = model_file_path
        self.acc_reduced = 0

    def on_epoch_end(self, epoch, logs={}):

        y_pred = self.model.predict(self.x_test)
        act_labels = np.argmax(self.y_test, axis=1 )
        pred_labels = np.argmax(y_pred, axis=1 )
        auc = roc_auc_score(self.y_test, y_pred, average='micro', multi_class='ovr')
        f1 = f1_score(act_labels, pred_labels, average='micro')
        f1 = f1_score(act_labels, pred_labels, average='micro')
        print('\n Epoch:{}, f1 score :{} , auc :{}'.format(epoch+1, f1, auc))


    def modify_lr_rate(self, epoch, lr):
        '''
            this function modifies learning rate
        '''

        if self.acc_reduced: # REDUCE LR BY 10% IF ACCURACY HAS DECREASED
            lr = lr - (0.1*lr)
            print('lr decreased by 10%:{}'.format(lr))
            return lr

        if (epoch+1)%3 == 0: # REDUCE ACCURACY BY 5% IF EPOCH IS MULTIPLE OF 3
            lr = lr-(0.05*lr)
            print('lr decreased by 5%:{} '.format(lr))
            return lr

        return lr


inputs = Input(shape=(length,))

embedding_layer = Embedding(vocab_size,
                            100,
                            weights=[embedding_matrix],
                            input_length=length,
                            trainable=False)(inputs)

conv1 = Conv1D(filters=32, kernel_size=9, kernel_initializer = glorot_normal(seed=3), activati
conv2 = Conv1D(filters=32, kernel_size=8, kernel_initializer = glorot_normal(seed=3), activati
conv3 = Conv1D(filters=32, kernel_size=7, kernel_initializer = glorot_normal(seed=3), activati

merged = concatenate([conv1, conv2, conv3], axis=1)
pool1 = MaxPooling1D(pool_size=2)(merged)

conv4 = Conv1D(filters=16, kernel_size=6, kernel_initializer = glorot_normal(seed=3), activati
conv5 = Conv1D(filters=16, kernel_size=5, kernel_initializer = glorot_normal(seed=3), activati
conv6 = Conv1D(filters=16, kernel_size=4, kernel_initializer = glorot_normal(seed=3), activati

merged2 = concatenate([conv4, conv5, conv6], axis=1)
pool2 = MaxPooling1D(pool_size=2)(merged2)

conv7 = Conv1D(filters=8, kernel_size=2,kernel_initializer = glorot_normal(seed=3), activation

flat1 = Flatten()(conv7)
drop1 = Dropout(0.5)(flat1)
dense1 = Dense(100,kernel_initializer = glorot_normal(seed=3), activation='relu')(drop1)
outputs = Dense(20, kernel_initializer = glorot_normal(seed=3), activation='softmax')(dense1)

model1 = Model(inputs= inputs, outputs=outputs)
plot_model(model1, show_shapes=True, to_file='multichannel.png')
model1.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.0001), metrics=['accuracy'

#print(model1.summary())

#plot_model(model1, show_shapes=True, to_file='multichannel.png')


%load_ext tensorboard


report_auc_f1 = Customcallback(X_test, y_test, "models/")
```

```
early_stop = EarlyStopping(
    monitor='val_loss', min_delta=0, patience=2, verbose=0,
    mode='auto', baseline=None, restore_best_weights=False
)
#lr_obj = Learning_rate(0.07)
#lrschedule = LearningRateScheduler(report_auc_f1.modify_lr_rate, verbose=1)
model_dir = 'models/'
log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = TensorBoard(log_dir=log_dir,histogram_freq=0, write_graph=True,write_gr
model_save_checkpoint = ModelCheckpoint(os.path.join('models', 'best_model_1.h5'), verbose=1,

history = model1.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1,
                     validation_data=(X_test, y_test),
           callbacks=[report_auc_f1, tensorboard_callback, model_save_checkpoint])
score = model1.evaluate(X_test, y_test, verbose=0)
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard`
Epoch 1/10
442/442 [==============================] - 64s 128ms/step - loss: 2.7865 - accuracy: 0.12

 Epoch:1, f1 score :0.32377310388782665 , auc :0.8860007932656377

Epoch 00001: val_loss improved from inf to 1.96263, saving model to models/best_model_1.h
Epoch 2/10
442/442 [==============================] - 57s 129ms/step - loss: 1.7763 - accuracy: 0.39

 Epoch:2, f1 score :0.43445931591247083 , auc :0.9289030642118834

Epoch 00002: val_loss improved from 1.96263 to 1.60143, saving model to models/best_model
Epoch 3/10
442/442 [==============================] - 57s 128ms/step - loss: 1.4298 - accuracy: 0.50

 Epoch:3, f1 score :0.5162523900573613 , auc :0.9451969699786843

Epoch 00003: val_loss improved from 1.60143 to 1.41302, saving model to models/best_model
Epoch 4/10
442/442 [==============================] - 57s 128ms/step - loss: 1.2108 - accuracy: 0.58

 Epoch:4, f1 score :0.5432334820480136 , auc :0.9511011124682673

Epoch 00004: val_loss improved from 1.41302 to 1.33521, saving model to models/best_model
Epoch 5/10
442/442 [==============================] - 57s 129ms/step - loss: 1.0925 - accuracy: 0.62

 Epoch:5, f1 score :0.574251115360102 , auc :0.9565530386224661

Epoch 00005: val_loss improved from 1.33521 to 1.25774, saving model to models/best_model
Epoch 6/10
442/442 [==============================] - 57s 129ms/step - loss: 0.9848 - accuracy: 0.66

 Epoch:6, f1 score :0.5840237943488421 , auc :0.958375832803433

Epoch 00006: val_loss improved from 1.25774 to 1.23283, saving model to models/best_model
Epoch 7/10
442/442 [==============================] - 57s 129ms/step - loss: 0.9129 - accuracy: 0.69

 Epoch:7, f1 score :0.6146165285744636 , auc :0.9632042631508417

Epoch 00007: val_loss improved from 1.23283 to 1.15892, saving model to models/best_model
Epoch 8/10
442/442 [==============================] - 57s 128ms/step - loss: 0.8298 - accuracy: 0.72

 Epoch:8, f1 score :0.609942638623327 , auc :0.9635366299412572

Epoch 00008: val_loss improved from 1.15892 to 1.15633, saving model to models/best_model
Epoch 9/10
442/442 [==============================] - 56s 128ms/step - loss: 0.7911 - accuracy: 0.73
```

```
 Epoch:9, f1 score :0.6280008497981729 , auc :0.9657106180665408

Epoch 00009: val_loss improved from 1.15633 to 1.12420, saving model to models/best_model
Epoch 10/10
442/442 [==============================] - 57s 128ms/step - loss: 0.7265 - accuracy: 0.75

 Epoch:10, f1 score :0.6432972169109836 , auc :0.966962291822615
```

```
%tensorboard --logdir logs/fit/20201227-052950  --port=8047
```



☐ Show data download links

☐ Ignore outliers in chart scalin

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis
STEP [✓ RELATIVE] WALL

Runs

Write a regex to filter runs

☐ ○ train

☐ ○ validation

TOGGLE ALL RUNS

logs/fit/20201227-052950

Prepare train data and test data for model 2

```
alphabets = 'abcdefghijklmnopqrstuvwxyz0123456789-,;.!?:’"/\|_@#$%^& ~‘+=<>()[]{}'
```

```
len(set(alphabets))
```

```
68
```

```
tk = Tokenizer(num_words= len(set(alphabets)), char_level= True, oov_token='UNK')
```

```
char_dict = {}
for i, char in enumerate(set(alphabets)):
    char_dict[char] = i+1
tk.word_index = char_dict.copy()
tk.word_index[tk.oov_token] = max(char_dict.values()) + 1
```

```
tk.word_index.keys()
```

```
dict_keys(['%', '#', 'd', 'n', ' ', '>', 'f', 'w', '_', 'i', 'b', '?', '-', '{', '‘', ']'
```

```
len(tk.word_index)
```

```
69
```

```
X[0]
```

```
'mit cornell mindlink athena newshub yorku mnemosyne nyx ariel edu Jack Morris article
article          article                       has   jack   lost      bit      his   edg
e     what      the   worst   start   jamorris   has    had                     jack   l
ost   his   edge   about     years    ago     and    has    had    only   one   above
average   year      the   last               again   goes      prove   that          bett
er         good   than    lucky     you   can        count       good   tomorrow     lucky
seems          prone      bad   starts                 hey   valentine        don   not
see   boston   with   any   world   series   rings      their     fingers     oooooo
```

Now if we look at our sentences there are irregular number of spaces in between words, we want the words to be separated by single space. So, we will trasform the setences accordingly and tokenize them.

lets first split the data in train ad test set

```
X = [ ' '.join(text.split()) for text in X ]
```

```
X[0]
```

```
'mit cornell mindlink athena newshub yorku mnemosyne nyx ariel edu Jack Morris article a
rticle article has jack lost bit his edge what the worst start jamorris has had jack los
t his edge about years ago and has had only one above average year the last again goes p
rove that better good than lucky you can count good tomorrow lucky seems prone bad start
s hey valentine don not see boston with any world series rings their fingers oooooo chea
p shot damn morris now has three and probably the hall fame his future who cares had two
them before came toronto and the jays had signed viola instead morris would have been fr
ank who won and got the ring and would his way this year too therefore would have say to
```

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25, random_state=42, str
```

```
X_train = encode_text(tk, X_train, 1014)
X_test = encode_text(tk, X_test, 1014)
```

```
y_train = le.transform(y_train)
y_test = le.transform(y_test)
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

lets prepare character embeddings

```
embeddings_index = {}
f = open(os.path.join('glove_char_weights.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))
```

```
Found 94 word vectors.
```

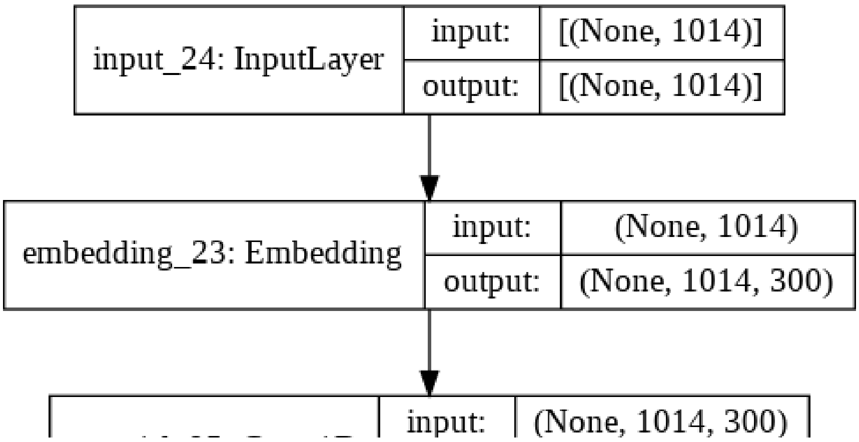
```
embedding_matrix = np.zeros((len(tk.word_index) + 1, 300))
for word, i in tk.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector
```

lets perform modelling now

```
inputs = Input(shape=(1014,))

embedding_layer = Embedding(len(tk.word_index)+1,
                            300,
                            weights=[embedding_matrix],
                            input_length=1014,
                            trainable=False)(inputs)

conv1 = Conv1D(filters=128, kernel_size=8, kernel_initializer = glorot_normal(seed=3), activat
conv2 = Conv1D(filters=128, kernel_size=8, kernel_initializer = glorot_normal(seed=3), activat

pool1 = MaxPooling1D(pool_size=2)(conv2)

conv3 = Conv1D(filters=64, kernel_size=4, kernel_initializer = glorot_normal(seed=3), activati
conv4 = Conv1D(filters=64, kernel_size=4, kernel_initializer = glorot_normal(seed=3), activati

pool2 = MaxPooling1D(pool_size=2)(conv4)

flat1 = Flatten()(pool2)
drop1 = Dropout(0.5)(flat1)

dense1 = Dense(100, kernel_initializer = glorot_normal(seed=3), activation='relu')(drop1)
drop2 = Dropout(0.2)(dense1)

outputs = Dense(20 ,kernel_initializer = glorot_normal(seed=3) ,activation='softmax')(drop2)

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

plot_model(model2, show_shapes=True, to_file='multichannel.png')

model2.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.0001), metrics=['accuracy'

print(model2.summary())

plot_model(model2, show_shapes=True, to_file='multichannel.png')
```

```
    plot_model(model2, show_shapes=True, to_file='multichannel.png')
```

```
Model: "model_22"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_24 (InputLayer)        [(None, 1014)]            0         
_________________________________________________________________
embedding_23 (Embedding)     (None, 1014, 300)         21000     
_________________________________________________________________
conv1d_95 (Conv1D)           (None, 1007, 128)         307328    
_________________________________________________________________
conv1d_96 (Conv1D)           (None, 1000, 128)         131200    
_________________________________________________________________
max_pooling1d_46 (MaxPooling (None, 500, 128)          0         
_________________________________________________________________
conv1d_97 (Conv1D)           (None, 497, 64)           32832     
_________________________________________________________________
conv1d_98 (Conv1D)           (None, 494, 64)           16448     
_________________________________________________________________
max_pooling1d_47 (MaxPooling (None, 247, 64)           0         
_________________________________________________________________
flatten_23 (Flatten)         (None, 15808)             0         
_________________________________________________________________
dropout_38 (Dropout)         (None, 15808)             0         
_________________________________________________________________
dense_43 (Dense)             (None, 100)               1580900   
_________________________________________________________________
dropout_39 (Dropout)         (None, 100)               0         
_________________________________________________________________
dense_44 (Dense)             (None, 20)                2020      
=================================================================
Total params: 2,091,728
Trainable params: 2,070,728
Non-trainable params: 21,000
_________________________________________________________________
None
```

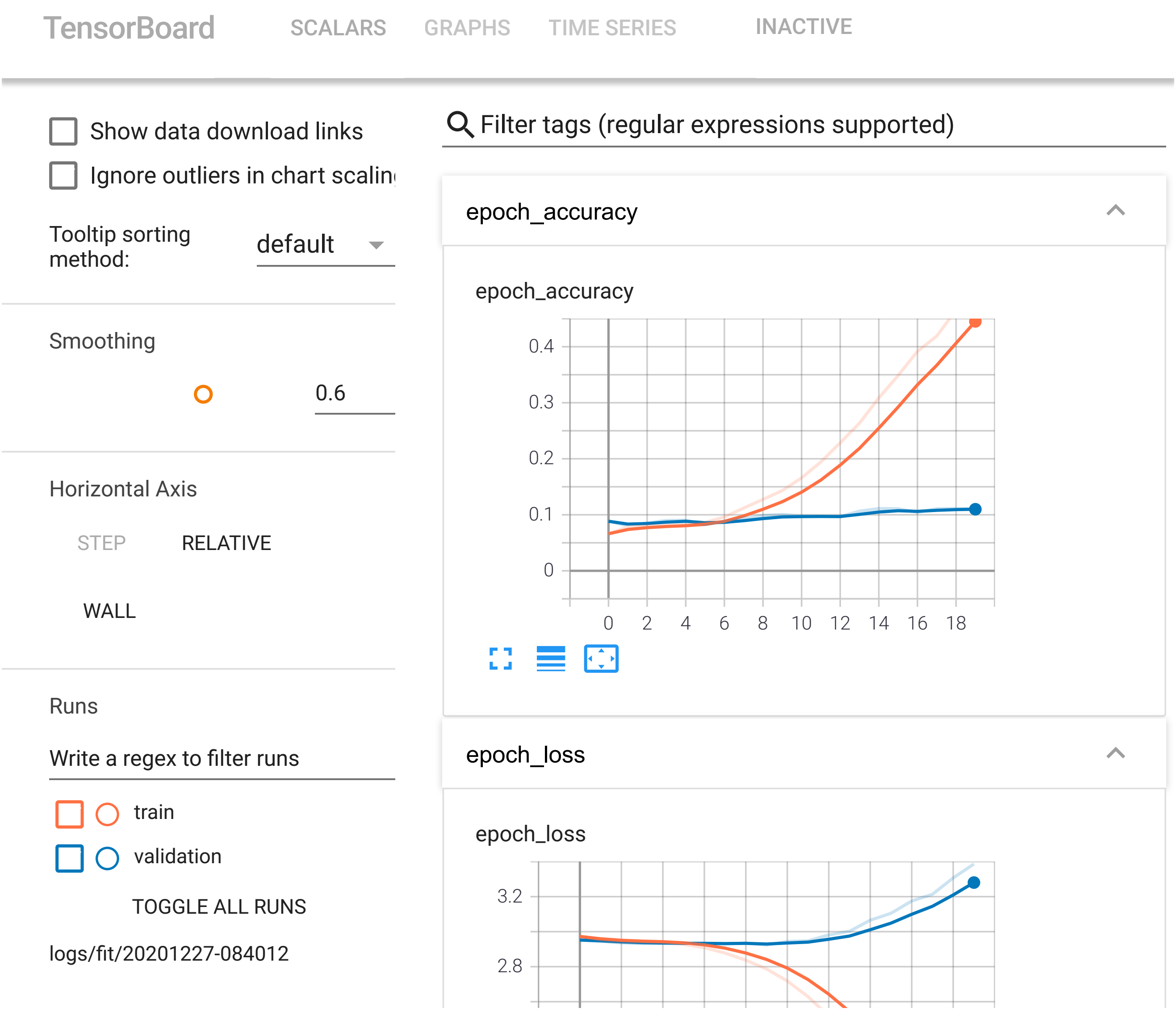


```
report_auc_f1 = Customcallback(X_test, y_test, "models/")
early_stop = EarlyStopping(
    monitor='val_loss', min_delta=0, patience=2, verbose=0,
    mode='auto', baseline=None, restore_best_weights=False)


log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = TensorBoard(log_dir=log_dir,histogram_freq=0, write_graph=True,write_gr
model_save_checkpoint = ModelCheckpoint(os.path.join('models', 'best_model_2.h5'), verbose=1,


history = model2.fit(X_train, y_train, epochs=20, batch_size=128, verbose=1,validation_split =
           callbacks=[report_auc_f1, tensorboard_callback, model_save_checkpoint])
score = model2.evaluate(X_test, y_test, verbose=0)
```

```
89/89 [==============================] - 11s 127ms/step - loss: 2.9368 - accuracy: 0.0829

 Epoch:5, f1 score :0.08370512003399193 , auc :0.5993725914804234
```

```
Epoch 00005: val_loss did not improve from 2.93201
Epoch 6/20
89/89 [==============================] - 11s 127ms/step - loss: 2.9356 - accuracy: 0.0822

 Epoch:6, f1 score :0.08604206500956023 , auc :0.599352551625485

Epoch 00006: val_loss improved from 2.93201 to 2.93097, saving model to models/best_model
Epoch 7/20
89/89 [==============================] - 11s 128ms/step - loss: 2.9105 - accuracy: 0.0917

 Epoch:7, f1 score :0.08604206500956023 , auc :0.6019096273760248

Epoch 00007: val_loss did not improve from 2.93097
Epoch 8/20
89/89 [==============================] - 11s 128ms/step - loss: 2.8780 - accuracy: 0.1114

 Epoch:8, f1 score :0.0839175695772254 , auc :0.6049839955413842

Epoch 00008: val_loss improved from 2.93097 to 2.92994, saving model to models/best_model
Epoch 9/20
89/89 [==============================] - 11s 129ms/step - loss: 2.8317 - accuracy: 0.1312

 Epoch:9, f1 score :0.09199065222009772 , auc :0.6034893456707289

Epoch 00009: val_loss did not improve from 2.92994
Epoch 10/20
89/89 [==============================] - 11s 129ms/step - loss: 2.7798 - accuracy: 0.1433

 Epoch:10, f1 score :0.09985128531973657 , auc :0.6089895031187571

Epoch 00010: val_loss improved from 2.92994 to 2.92361, saving model to models/best_model
Epoch 11/20
89/89 [==============================] - 11s 129ms/step - loss: 2.7256 - accuracy: 0.1696

 Epoch:11, f1 score :0.10218823029530485 , auc :0.6137582509963372

Epoch 00011: val_loss did not improve from 2.92361
Epoch 12/20
89/89 [==============================] - 11s 128ms/step - loss: 2.6357 - accuracy: 0.1894

 Epoch:12, f1 score :0.10813681750584236 , auc :0.6118705992834288

Epoch 00012: val_loss did not improve from 2.92361
Epoch 13/20
89/89 [==============================] - 11s 128ms/step - loss: 2.4996 - accuracy: 0.2372

 Epoch:13, f1 score :0.09836413851710218 , auc :0.6068466615745576

Epoch 00013: val_loss did not improve from 2.92361
Epoch 14/20
89/89 [==============================] - 11s 128ms/step - loss: 2.4198 - accuracy: 0.2557

 Epoch:14, f1 score :0.10516252390057362 , auc :0.6146105636532033

Epoch 00014: val_loss did not improve from 2.92361
Epoch 15/20
```

```
%tensorboard --logdir logs/fit/20201227-084012  --port=8047
```



□ Show data download links
□ Ignore outliers in chart scalin

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis
STEP RELATIVE
WALL

Runs
Write a regex to filter runs
□ ○ train
□ ○ validation
TOGGLE ALL RUNS
logs/fit/20201227-084012

```
print('accuracy of model 2 is', score[1])
```

```
accuracy of model 2 is 0.11068621277809143
```