Birectional LSTM model for audio labeling with Keras

In this Kaggle kernel we will use the curated data from the "Freesound Audio Tagging 2019" competition to predict the labels of .wav files.

Table of contents

Data Description

From Kaggle's data page (https://www.kaggle.com/c/freesound-audio-tagging-2019/data) for the competition:

The curated subset is a small set of manually-labeled data from FSD.

Number of clips/class: 75 except in a few cases (where there are less)

Total number of clips: 4970

Avge number of labels/clip: 1.2

Total duration: 10.5 hours

The duration of the audio clips ranges from 0.3 to 30s due to the diversity of the sound categories and the preferences of Freesound users when recording/uploading sounds. It can happen that a few of these audio clips present additional acoustic material beyond the provided ground truth label(s).

Test Set:

The test set is used for system evaluation and consists of manually-labeled data from FSD. Since most of the train data come from YFCC, some acoustic domain mismatch between the train and test set can be expected. All the acoustic material present in the test set is labeled, except human error, considering the vocabulary of 80 classes used in the competition.

Columns:

fname: the audio file name, eg, 0006ae4e.wav *labels*: the audio classification label(s) (ground truth). Note that the number of labels per clip can be one, eg, Bark or more, eg, "Walk_and_footsteps,Slam".

Dependencies

In [1]:
```
# Dependencies
import numpy as np
import pandas as pd
import os
import librosa
import matplotlib.pyplot as plt
import gc
import time
from tqdm import tqdm, tqdm_notebook; tqdm.pandas() # Progress bar
from sklearn.metrics import label_ranking_average_precision_score
from sklearn.model_selection import train_test_split

# Machine Learning
import tensorflow as tf
from keras import backend as K
from keras.engine.topology import Layer
from keras import initializers, regularizers, constraints, optimizers, layer
s
from keras.layers import (Dense, Bidirectional, CuDNNLSTM, ELU,
                          Dropout, LeakyReLU, Conv1D, BatchNormalization)
from keras.models import Sequential
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping

# Path specifications
KAGGLE_DIR = '../input/'
train_curated_path = KAGGLE_DIR + 'train_curated/'
test_path = KAGGLE_DIR + 'test/'

# Set seed for reproducability
seed = 1234
np.random.seed(seed)
tf.set_random_seed(seed)

# File sizes and specifications
print('\n# Files and file sizes')
for file in os.listdir(KAGGLE_DIR):
    print('{}| {} MB'.format(file.ljust(30),
                             str(round(os.path.getsize(KAGGLE_DIR + file) /
1000000, 2))))

# For keeping time. GPU limit for this competition is set to 60 min.
t_start = time.time()
```

```
# Files and file sizes
train_curated.csv              | 0.14 MB
train_noisy.csv                | 0.58 MB
test                           | 0.04 MB
sample_submission.csv          | 0.19 MB
train_curated                  | 0.14 MB
train_noisy                    | 0.55 MB

Using TensorFlow backend.
```

Evaluation metric

From the [competition evaluation page (https://www.kaggle.com/c/freesound-audio-tagging-2019/overview/evaluation)](https://www.kaggle.com/c/freesound-audio-tagging-2019/overview/evaluation):

The task consists of predicting the audio labels (tags) for every test clip. Some test clips bear one label while others bear several labels. The predictions are to be done at the clip level, i.e., no start/end timestamps for the sound events are required.

The primary competition metric will be label-weighted [label-ranking average precision (https://scikit-learn.org/stable/modules/model_evaluation.html#label-ranking-average-precision)](https://scikit-learn.org/stable/modules/model_evaluation.html#label-ranking-average-precision) (lwlrap, pronounced "Lol wrap"). This measures the average precision of retrieving a ranked list of relevant labels for each test clip (i.e., the system ranks all the available labels, then the precisions of the ranked lists down to each true label are averaged). This is a generalization of the mean reciprocal rank measure (used in last year's edition of the competition) for the case where there can be multiple true labels per test item. The novel "label-weighted" part means that the overall score is the average over all the labels in the test set, where each label receives equal weight (by contrast, plain lrap gives each test item equal weight, thereby discounting the contribution of individual labels when they appear on the same item as multiple other labels).

The formula for label-ranking average precision (LRAP) is as follows:

$$LRAP(y, \hat{f}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} \frac{1}{||y_i||_0} \sum_{j:y_{ij}=1} \frac{|\mathcal{L}_{ij}|}{\text{rank}_{ij}}$$

Happily, the evaluation metric is provided by Kaggle and can be found in this [Google Colab file (https://colab.research.google.com/drive/1AgPdhSp7ttY18O3fEoHOQKlt_3HJDLi8#scrollTo=52LPXQNPppex)](https://colab.research.google.com/drive/1AgPdhSp7ttY18O3fEoHOQKlt_3HJDLi8#scrollTo=52LPXQNPppex).

In [2]:
```
def calculate_overall_lwlrap_sklearn(truth, scores):
    """Calculate the overall lwlrap using sklearn.metrics.lrap."""
    # sklearn doesn't correctly apply weighting to samples with no labels, s
o just skip them.
    sample_weight = np.sum(truth > 0, axis=1)
    nonzero_weight_sample_indices = np.flatnonzero(sample_weight > 0)
    overall_lwlrap = label_ranking_average_precision_score(
        truth[nonzero_weight_sample_indices, :] > 0,
        scores[nonzero_weight_sample_indices, :],
        sample_weight=sample_weight[nonzero_weight_sample_indices])
    return overall_lwlrap
```

Helper Functions and Preprocessing

I got the inspiration for most of the preprocessing steps and the attention layer from [this Kaggle kernel (https://www.kaggle.com/chewzy/gru-w-attention-baseline-model-curated)](https://www.kaggle.com/chewzy/gru-w-attention-baseline-model-curated).

In [3]:
```
n_classes = 80

def split_and_label(rows_labels, n_classes):
    '''
    Retrieves a list of all the relevant classes. This is necessary due to
    the multi-labeling of the initial csv file.
    '''
    row_labels_list = []
    for row in rows_labels:
        row_labels = row.split(',')
        labels_array = np.zeros((n_classes))
        for label in row_labels:
            index = label_mapping[label]
            labels_array[index] = 1
        row_labels_list.append(labels_array)
    return row_labels_list
```

In [4]:
```
# Load in data
df = pd.read_csv(KAGGLE_DIR + 'train_curated.csv')
test_df = pd.read_csv(KAGGLE_DIR + 'sample_submission.csv')

# Retrieve labels
label_columns = test_df.columns[1:]
label_mapping = dict((label, index) for index, label in enumerate(label_colu
mns))
for col in label_columns:
    df[col] = 0
df[label_columns] = split_and_label(df['labels'], n_classes)
df['num_labels'] = df[label_columns].sum(axis=1)
```

In [5]:
```
# Check dataframes
print('Training dataframe:')
display(df.head(3))
print('Testing dataframe:')
test_df.head(3)
```

```
Training dataframe:
```

	fname	labels	Accelerating_and_revving_and_vroom	Accordion	Acoustic_guitar	Appla
0	0006ae4e.wav	Bark	0.0	0.0	0.0	
1	0019ef41.wav	Raindrop	0.0	0.0	0.0	
2	001ec0ad.wav	Finger_snapping	0.0	0.0	0.0	

```
Testing dataframe:
```

Out[5]:

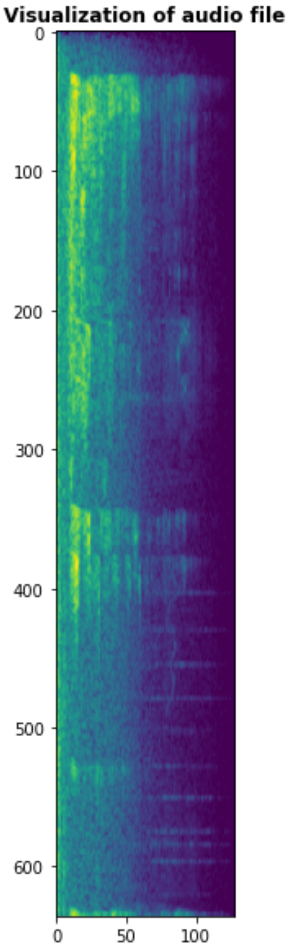
	fname	Accelerating_and_revving_and_vroom	Accordion	Acoustic_guitar	Applause	Bark	Bass_
0	000ccb97.wav	0	0	0	0	0	
1	0012633b.wav	0	0	0	0	0	
2	001ed5f1.wav	0	0	0	0	0	

In [6]:
```
# Preprocessing parameters
sr = 44100 # Sampling rate
duration = 5
hop_length = 347 # to make time steps 128
fmin = 20
fmax = sr // 2
n_mels = 128
n_fft = n_mels * 20
samples = sr * duration
```

In [7]:
```
def read_audio(path):
    '''
    Reads in the audio file and returns
    an array that we can turn into a melspectogram
    '''
    y, _ = librosa.core.load(path, sr=44100)
    # trim silence
    if 0 < len(y): # workaround: 0 length causes error
        y, _ = librosa.effects.trim(y)
    if len(y) > samples: # long enough
        y = y[0:0+samples]
    else: # pad blank
        padding = samples - len(y)
        offset = padding // 2
        y = np.pad(y, (offset, samples - len(y) - offset), 'constant')
    return y

def audio_to_melspectrogram(audio):
    '''
    Convert to melspectrogram after audio is read in
    '''
    spectrogram = librosa.feature.melspectrogram(audio,
                                                 sr=sr,
                                                 n_mels=n_mels,
                                                 hop_length=hop_length,
                                                 n_fft=n_fft,
                                                 fmin=fmin,
                                                 fmax=fmax)
    return librosa.power_to_db(spectrogram).astype(np.float32)

def read_as_melspectrogram(path):
    '''
    Convert audio into a melspectrogram
    so we can use machine learning
    '''
    mels = audio_to_melspectrogram(read_audio(path))
    return mels

def convert_wav_to_image(df, path):
    X = []
    for _,row in tqdm_notebook(df.iterrows()):
        x = read_as_melspectrogram('{}/{}'.format(path[0],
                                                  str(row['fname'])))
        X.append(x.transpose())
    return X

def normalize(img):
    '''
    Normalizes an array
    (subtract mean and divide by standard deviation)
    '''
    eps = 0.001
    if np.std(img) != 0:
        img = (img - np.mean(img)) / np.std(img)
    else:
        img = (img - np.mean(img)) / eps
    return img

def normalize_dataset(X):
    '''
    Normalizes list of arrays
    (subtract mean and divide by standard deviation)
    '''
    normalized_dataset = []
    for img in X:
        normalized = normalize(img)
        normalized_dataset.append(normalized)
    return normalized_dataset
```

In [8]:
```
# Preprocess dataset and create validation sets
X = np.array(convert_wav_to_image(df, [train_curated_path]))
X = normalize_dataset(X)
Y = df[label_columns].values
x_train, x_val, y_train, y_val = train_test_split(X, Y, test_size=0.1, rando
m_state=seed)
```

In [9]:
```
# Visualize an melspectogram example
plt.figure(figsize=(15,10))
plt.title('Visualization of audio file', weight='bold')
plt.imshow(X[0]);
```



Modeling

My main inspiration for this architecture has been [this paper (https://arxiv.org/pdf/1602.05875v3.pdf)](https://arxiv.org/pdf/1602.05875v3.pdf).

In [10]:
```
class Attention(Layer):
    def __init__(self, step_dim,
                 W_regularizer=None, b_regularizer=None,
                 W_constraint=None, b_constraint=None,
                 bias=True, **kwargs):
        self.supports_masking = True
        self.init = initializers.get('glorot_uniform')
        self.W_regularizer = regularizers.get(W_regularizer)
        self.b_regularizer = regularizers.get(b_regularizer)
        self.W_constraint = constraints.get(W_constraint)
        self.b_constraint = constraints.get(b_constraint)
        self.bias = bias
        self.step_dim = step_dim
        self.features_dim = 0
        super(Attention, self).__init__(**kwargs)

    def build(self, input_shape):
        assert len(input_shape) == 3

        self.W = self.add_weight((input_shape[-1],),
                                 initializer=self.init,
                                 name='{}_W'.format(self.name),
                                 regularizer=self.W_regularizer,
                                 constraint=self.W_constraint)
        self.features_dim = input_shape[-1]

        if self.bias:
            self.b = self.add_weight((input_shape[1],),
                                     initializer='zero',
                                     name='{}_b'.format(self.name),
                                     regularizer=self.b_regularizer,
                                     constraint=self.b_constraint)
        else:
            self.b = None
        self.built = True

    def compute_mask(self, input, input_mask=None):
        return None

    def call(self, x, mask=None):
        features_dim = self.features_dim
        step_dim = self.step_dim

        eij = K.reshape(K.dot(K.reshape(x, (-1, features_dim)),
                        K.reshape(self.W, (features_dim, 1))), (-1, step_di
m))
        if self.bias:
            eij += self.b
        eij = K.tanh(eij)
        a = K.exp(eij)
        if mask is not None:
            a *= K.cast(mask, K.floatx())
        a /= K.cast(K.sum(a, axis=1, keepdims=True) + K.epsilon(), K.floatx
())
        a = K.expand_dims(a)
        weighted_input = x * a
        return K.sum(weighted_input, axis=1)

    def compute_output_shape(self, input_shape):
        return input_shape[0],  self.features_dim
```

In [11]:

```
# Neural network model
input_shape = (636,128)
optimizer = Adam(0.005, beta_1=0.1, beta_2=0.001, amsgrad=True)
n_classes = 80

model = Sequential()
model.add(Bidirectional(CuDNNLSTM(256, return_sequences=True), input_shape=i
nput_shape))
model.add(Attention(636))
model.add(Dropout(0.2))
model.add(Dense(400))
model.add(ELU())
model.add(Dropout(0.2))
model.add(Dense(n_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer=optimizer,
              metrics=['acc'])
```

```
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/pyt
hon/framework/op_def_library.py:263: colocate_with (from tensorflow.python.fr
amework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/keras/backend/
tensorflow_backend.py:3445: calling dropout (from tensorflow.python.ops.nn_op
s) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - k
eep_prob`.
```

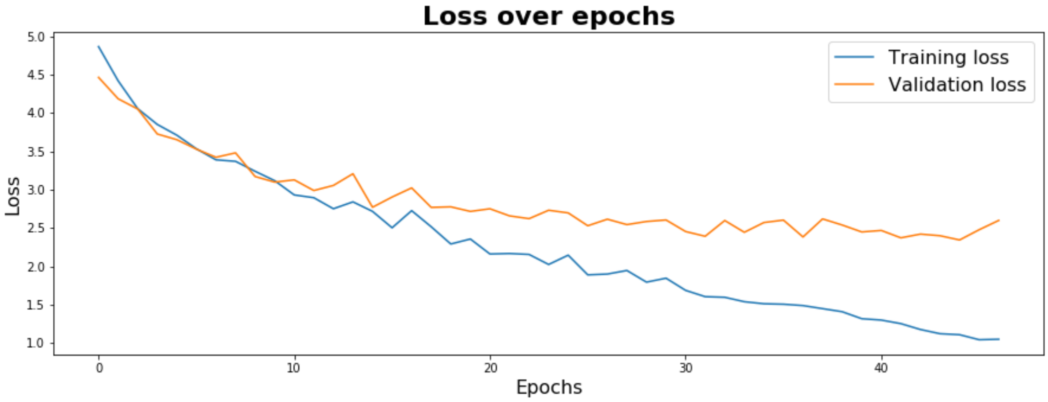
In [12]:
```
# Train model
es = EarlyStopping(monitor='val_acc', mode='max', verbose=1, patience=10)
hist = model.fit(np.array(x_train),
          y_train,
          batch_size=1024,
          epochs=500,
          validation_data=(np.array(x_val), y_val),
          callbacks = [es])
```

```
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/pyt
hon/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is d
eprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 4473 samples, validate on 497 samples
Epoch 1/500
4473/4473 [==============================] - 12s 3ms/step - loss: 4.8638 - ac
c: 0.0483 - val_loss: 4.4624 - val_acc: 0.0986
Epoch 2/500
4473/4473 [==============================] - 6s 1ms/step - loss: 4.4178 - ac
c: 0.1008 - val_loss: 4.1836 - val_acc: 0.1167
Epoch 3/500
4473/4473 [==============================] - 6s 1ms/step - loss: 4.0545 - ac
c: 0.1567 - val_loss: 4.0502 - val_acc: 0.1529
Epoch 4/500
4473/4473 [==============================] - 6s 1ms/step - loss: 3.8491 - ac
c: 0.1827 - val_loss: 3.7245 - val_acc: 0.1811
Epoch 5/500
4473/4473 [==============================] - 6s 1ms/step - loss: 3.7089 - ac
c: 0.2077 - val_loss: 3.6501 - val_acc: 0.1972
Epoch 6/500
4473/4473 [==============================] - 6s 1ms/step - loss: 3.5312 - ac
c: 0.2397 - val_loss: 3.5270 - val_acc: 0.2274
Epoch 7/500
4473/4473 [==============================] - 6s 1ms/step - loss: 3.3873 - ac
c: 0.2676 - val_loss: 3.4207 - val_acc: 0.2314
Epoch 8/500
4473/4473 [==============================] - 6s 1ms/step - loss: 3.3682 - ac
c: 0.2707 - val_loss: 3.4783 - val_acc: 0.2455
Epoch 9/500
4473/4473 [==============================] - 6s 1ms/step - loss: 3.2380 - ac
c: 0.2862 - val_loss: 3.1699 - val_acc: 0.3058
Epoch 10/500
4473/4473 [==============================] - 6s 1ms/step - loss: 3.1157 - ac
c: 0.3163 - val_loss: 3.0972 - val_acc: 0.2978
Epoch 11/500
4473/4473 [==============================] - 6s 1ms/step - loss: 2.9296 - ac
c: 0.3588 - val_loss: 3.1261 - val_acc: 0.2817
Epoch 12/500
4473/4473 [==============================] - 6s 1ms/step - loss: 2.8936 - ac
c: 0.3541 - val_loss: 2.9871 - val_acc: 0.3099
Epoch 13/500
4473/4473 [==============================] - 6s 1ms/step - loss: 2.7495 - ac
c: 0.3906 - val_loss: 3.0536 - val_acc: 0.3038
Epoch 14/500
4473/4473 [==============================] - 6s 1ms/step - loss: 2.8394 - ac
c: 0.3651 - val_loss: 3.2061 - val_acc: 0.2857
Epoch 15/500
4473/4473 [==============================] - 6s 1ms/step - loss: 2.7153 - ac
c: 0.3937 - val_loss: 2.7690 - val_acc: 0.3763
Epoch 16/500
4473/4473 [==============================] - 6s 1ms/step - loss: 2.5014 - ac
c: 0.4420 - val_loss: 2.9024 - val_acc: 0.3461
Epoch 17/500
4473/4473 [==============================] - 6s 1ms/step - loss: 2.7248 - ac
c: 0.3957 - val_loss: 3.0215 - val_acc: 0.3501
Epoch 18/500
4473/4473 [==============================] - 6s 1ms/step - loss: 2.5144 - ac
c: 0.4384 - val_loss: 2.7673 - val_acc: 0.3682
Epoch 19/500
4473/4473 [==============================] - 6s 1ms/step - loss: 2.2901 - ac
c: 0.4851 - val_loss: 2.7746 - val_acc: 0.3964
Epoch 20/500
4473/4473 [==============================] - 6s 1ms/step - loss: 2.3551 - ac
c: 0.4681 - val_loss: 2.7149 - val_acc: 0.4145
Epoch 21/500
4473/4473 [==============================] - 6s 1ms/step - loss: 2.1597 - ac
```

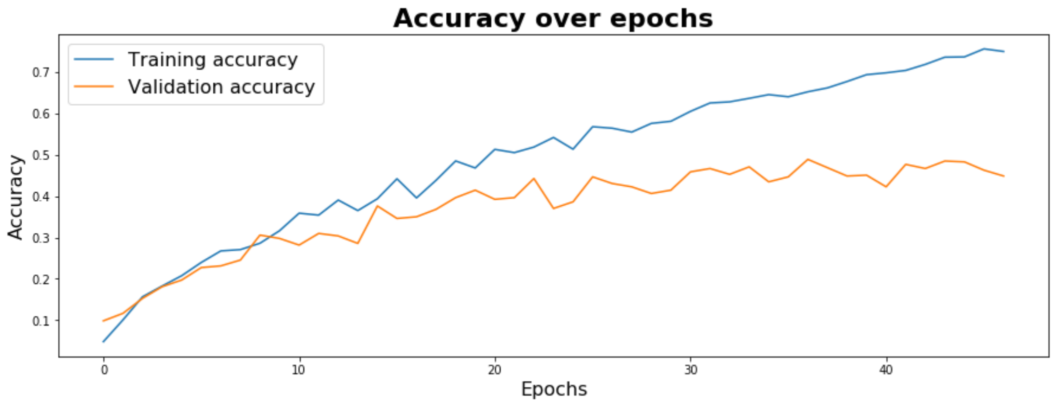
Visualization and Evaluation

Simple visualizations to keep track of the loss and accuracy over the epochs.

In [13]:
```
# Visualize loss
loss = hist.history['loss']
val_loss = hist.history['val_loss']
stopped_epoch = es.stopped_epoch
epochs = range(stopped_epoch+1)

plt.figure(figsize=(15,5))
plt.plot(epochs, loss)
plt.plot(epochs, val_loss)
plt.title('Loss over epochs', weight='bold', fontsize=22)
plt.xlabel('Epochs', fontsize=16)
plt.ylabel('Loss', fontsize=16)
plt.legend(['Training loss', 'Validation loss'], fontsize=16)
plt.show()
```



In [14]:
```
# Visualize Accuracy
acc = hist.history['acc']
val_acc = hist.history['val_acc']
epochs = range(stopped_epoch+1)

plt.figure(figsize=(15,5))
plt.plot(epochs, acc)
plt.plot(epochs, val_acc)
plt.title('Accuracy over epochs', weight='bold', fontsize=22)
plt.xlabel('Epochs', fontsize=16)
plt.ylabel('Accuracy', fontsize=16)
plt.legend(['Training accuracy', 'Validation accuracy'], fontsize=16)
plt.show()
```



Training accuracy LWLRAP score:

In [15]:
```
# Make predictions for training set and validation set
y_train_pred = model.predict(np.array(x_train))
y_val_pred = model.predict(np.array(x_val))
train_lwlrap = calculate_overall_lwlrap_sklearn(y_train, y_train_pred)
val_lwlrap = calculate_overall_lwlrap_sklearn(y_val, y_val_pred)

# Check training and validation LWLRAP score
print('Training LWLRAP : {}'.format(round(train_lwlrap,4)))
print('Validation LWLRAP : {}'.format(round(val_lwlrap,4)))
```

```
Training LWLRAP : 0.8752
Validation LWLRAP : 0.6121
```

Predictions and submission

Preprocess the test set, make predictions and store them as a csv file for our submission.

In [16]:
```
# Prepare test set
X_test = np.array(convert_wav_to_image(test_df, [test_path]))
X_test = normalize_dataset(X_test)
# Make predictions
predictions = model.predict(np.array(X_test))
# Save predictions in a csv file
test_df[label_columns] = predictions
test_df.to_csv('submission.csv', index=False)
```

Final checks

Lastly, we check if the submission format is correct and if we are under the one hour limit of GPU time.

In [17]:
```
# Check submission format
display(test_df.head())

# Check if we are under one hour of GPU time
t_finish = time.time()
total_time = round((t_finish-t_start)/3600, 4)
print('Kernel runtime = {} hours ({} minutes)'.format(total_time,
                                                      int(total_time*60)))
```

	fname	Accelerating_and_revving_and_vroom	Accordion	Acoustic_guitar	Applause	Bark
0	000ccb97.wav	0.000161	0.000009	7.580730e-05	2.552045e-04	0.001516
1	0012633b.wav	0.091847	0.000155	8.812740e-05	5.745049e-05	0.001546
2	001ed5f1.wav	0.000046	0.000007	1.403562e-05	5.386537e-05	0.000021
3	00294be0.wav	0.000003	0.000001	8.072470e-07	7.080239e-08	0.000127
4	003fde7a.wav	0.000021	0.000233	3.527369e-06	4.100578e-06	0.000005

```
Kernel runtime = 0.2056 hours (12 minutes)
```

If you like this Kaggle kernel, feel free to give an upvote and leave a comment! I will try to implement your suggestions in this kernel!