Audio Mosaicing with the Freesound API and Essentia

David Bedoya
Universitat Pompeu Fabra
josedavid.bedoya01@estudiant.upf.edu

March 29, 2020

The main goal of this project is to create an audio mosaic using the Freesound API to search and download audios, and Essentia to analyze them. The process is divided into three notebooks available at https://github.com/ffont/amplab-freesound. The following sections describe the reasons for the changes made in each of these notebooks.

1 Create source sound collection

This first notebook creates the collection of sounds that is later used as source material for the audio mosaicing application. The collection intended in this project is a set of percussion instrument samples of the Beijing Opera. In the `query_freesound` function it is important to change the value of the `group_by_pack` parameter to 0, this is done so as not to collapse the results belonging to sounds of the same pack into single entries in the results list. 151 sounds are found and stored, the following query is the only one set.

```
freesound_queries = [
    {
        'query': 'beijin-opera dataset',
        'filter': None,
        'num_results': 200,
    },
]
```

2 Analyze source collection and target file

This second notebook analyzes the collection of sounds compiled in the previous notebook and which is then used as the source collection in the audio mosaic code. The `analyze_sound` function is used here to extract the features of the source collection samples, its `frameSize` parameter is set to `None`, this is to analyze the whole audio as a single frame. Originally, this function extracts `loudness` and `MFCC` coefficients. Additionally, `Flux`[1], `SpectralCentroidTime`[2] and `RollOf`[3] features have been added in this project. These additional features have been shown to be good for the classification of percussion instruments of the Beijing Opera[4].

```
    # Extract other features here and add to 'frame_output' dictionary
    flux_algo = estd.Flux()
    flux = flux_algo(frame)
    frame_output['flux'] = flux / len(frame)

    spectral_centroid_time_algo = estd.SpectralCentroidTime()
    spectral_centroid_time = spectral_centroid_time_algo(frame)
    frame_output['spectral_centroid_time'] = spectral_centroid_time

    rolloff_algo = estd.RollOff()
    rolloff = rolloff_algo(frame)
    frame_output['rolloff'] = rolloff
```

For analysis of the target sound file, the `analyze_target_sound` function is provided, which extracts the features in frames matching beat positions instead of using equally-spaced frames. This is accomplished through a `OnsetDetection`[5] function with the `hfc` method. Fig. 1 shows the first 4 seconds of the target sound, the vertical red lines mark the boundaries of the frames.

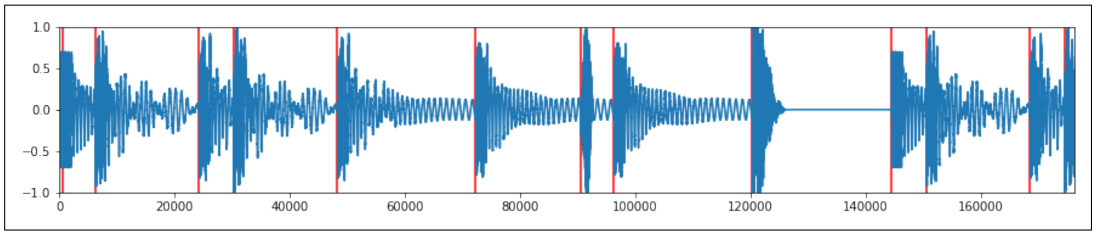

Figure 1: Target audio file (first 4 seconds).

[1]https://essentia.upf.edu/reference/std_Flux.html
[2]https://essentia.upf.edu/reference/std_SpectralCentroidTime.html
[3]https://essentia.upf.edu/reference/std_RollOff.html
[4]This statement is based on a project carried out in the MIR course
[5]https://essentia.upf.edu/reference/std_OnsetDetection.html

```
# Onset detection
# Phase 1: compute the onset detection function
od = estd.OnsetDetection(method='hfc')

# Let's also get the other algorithms we will need, and a pool to store the results
w = estd.Windowing(type = 'hann')
fft = estd.FFT() # this gives us a complex FFT
c2p = estd.CartesianToPolar() # and this turns it into a pair (magnitude, phase)
pool = estd.essentia.Pool()

# Computing onset detection functions.
for frame in estd.FrameGenerator(audio, frameSize = frame_size, hopSize = hop_size):
    mag, phase, = c2p(fft(w(frame)))
    pool.add('features.hfc', od(mag, phase))

# Phase 2: compute the actual onsets samples
onsets = estd.Onsets()
onsets_hfc = onsets(essentia.array([ pool['features.hfc'] ]), [ 1 ])
onsets_hfc_samples = [int(round(i*44100)) for i in onsets_hfc]

# Calculate the start and end samples for each audio frame
frame_start_end_samples = zip(onsets_hfc_samples[:-1], onsets_hfc_samples[1:])
```

3 Reconstruct target file with source collection frames

This final notebook performs audio mosaicing to construct a new version of the target file by using audio frames chosen from the source collection. The loaded features of csv files are scaled with a standard scaler[6], which improves the performance of the similarity algorithm used here.

```
# Scale features
features = ['loudness', 'flux', 'spectral_centroid_time', 'rolloff',
        'mfcc_0', 'mfcc_1', 'mfcc_2', 'mfcc_3', 'mfcc_4', 'mfcc_5', 'mfcc_6',
        'mfcc_7', 'mfcc_8', 'mfcc_9', 'mfcc_10', 'mfcc_11', 'mfcc_12']
scaler = preprocessing.StandardScaler()
df_target[features] = scaler.fit_transform(df_target[features])
```

The `chose_frame_from_source_collection` function is changed so as not to retrieve the most similar frame, but a random frame of those returned by the function `find_similar_frames`.

```
# Choose a similar frame at random
similar_frame = similar_frames[ np.random.randint(n_neighbours_to_find) ]
```

[6]https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

During reconstruction, each target frame is replaced with a whole sample from the source collection. If the length of the target frame is less than that of the source sound, the surplus audio is mixed in the following frames. Due to these surpluses, it may be necessary to add samples to the `generated_audio` array for some of the last frames.

```python
# Get the audio segment corresponding to the 'similar_frame'
similar_frame_audio = get_audio_file_segment(similar_frame['path'],
                                             similar_frame['start_sample'],
                                             similar_frame['end_sample'])

# Mix audio segment with the reconstructed audio array
frame_length = target_frame['start_sample'] + len(similar_frame_audio)
if target_frame['start_sample'] + len(similar_frame_audio) > len(generated_audio):
    generated_audio = np.append(generated_audio,
                                np.zeros(frame_length - len(generated_audio)))
generated_audio[target_frame['start_sample']:frame_length] += similar_frame_audio
```