Master thesis on Sound and Music Computing

Universitat Pompeu Fabra

Applications of Essentia on the web

Luis Joglar-Ongay

Supervisor: Dmitry Bogdanov

August 2020

Master thesis on Sound and Music Computing

Universitat Pompeu Fabra

Applications of Essentia on the web

Luis Joglar-Ongay

Supervisor: Dmitry Bogdanov

August 2020

Contents

"I never think of the future - it comes soon enough."

— Albert Einstein

"If I were not a physicist, I would probably be a musician.

I often think in music. I live my daydreams in music.

I see my life in terms of music."

— Albert Einstein

Acknowledgement

Thanks to my colleague and friend Alex Albàs, for our shared moments of work and passion. Maybe one day I can return the favour?

Thanks to my workmates, my friends and "the lads" for their support.

Thanks and congratulations to my fellow students Jorge and Alia for our ghost pomodoros that led us to success.

Thanks to all the people from the MTG for sharing their passion for music and technology, you all are an inspiration. Especially to my supervisor Dmitry and to Albin, Pablo and Xavier for our work together these past two years. I hope we can keep working and collaborating for a long time.

Thanks to my family for their help, interest, and support. Especially for their home catering delivery service.

Abstract

Essentia.js, a port of the popular Open-Source C++ library Essentia, comes to the web world to become the reference library to use together with the Web Audio API. Whilst there are many libraries for audio analysis and feature extraction in native computing languages, this Master thesis exposes the need of such a library in JavaScript, and proposes Essentia.js as the best option to cover those needs. The Music Information Retrieval community needs this tool to be able to develop software and research using web technologies, to continue evolving and stay in the state of the art.

Having compiled the C++ library into a JavaScript audio analysis library, a study on the efficiency of the library is carried out by benchmarking the execution of algorithms available in Essentia.js and comparing them to their equivalents from Meyda.js, a library written in JavaScript.

Also, an application designed to be integrated into a production environment is developed using Essentia.js to detect audio problems in music files uploaded to a website. Helping improve the efficiency of the quality control process for digital music distribution. The difficulties found in the process are enumerated and the solution developed is described and demonstrated.

These steps, described in this Master Thesis, are part of a bigger project by the Audio Signal Processing Lab at Music Technology Group of the Universitat Pompeu Fabra, to turn Essentia into Essentia.js and ensuring the quality of the library.

Keywords: Web Audio; Audio Analysis; Audio Problems

Chapter 1

Introduction

Web technologies are evolving every day providing higher capabilities and becoming one of the most used environments for software development. Javascript (from now on, also as JS) is the most used programming language in 2020 according to GitHub[1] and StackOverflow[2]. Audio Signal Processing (ASP) and Music Information Retrieval (MIR) development and research should follow this trend and jump into the web.

Thanks to the release and to the continued development of the Web Audio API the field of music and sound computing in the web is having a great evolution. It is the moment to bring into the game tools that allow developers and researchers to think of JavaScript and web technologies as any other language for their audio projects. Ideally, these tools should be developed to be as powerful as the ones already available in other languages such as C/C++ and Python.

All previous reasons motivated the exploration on how to use Essentia, an established open-source C++ library for music and audio analysis, description and synthesis developed by the Music Technology Group (MTG) of the Universitat Pompeu Fabra (UPF), as a JavaScript library. This would enable it to be used both in the web client or in other JS compliant platforms.

[1] `https://madnight.github.io/githut/`
[2] `https://insights.stackoverflow.com/survey/2020`

This Master thesis started alongside and as part of a bigger project by the Audio Signal Processing Lab (ASP lab) of the MTG during summer 2019 with the previously mentioned goal of having Essentia working as a JS library. Due to the nature of the project, the master thesis objective evolved along with the advances of the main project.

In this report, you will find descriptions of the steps researched by the author. Starting with the first steps into compiling Essentia from C++ to JS using Emscripten and WebAssembly technologies, done parallelly by the author and other members of the ASP lab. Continuing with a study of an application example using the audio problems algorithms of Essentia into a web application. Finalising with a benchmark study and comparison to Meyda, another audio feature extraction library written in JavaScript.

As part of the main project of porting Essentia into the web technologies, even though this is not part of this Master thesis report, the author collaborated with the dissemination of the project by doing a talk called "Essentia in the browser" in the Web Audio Conference in Norway in December 2019 [1]. Also collaborated in the accepted and pending publication paper for ISMIR 2020 [2], contributing to it mainly with the benchmarking and comparison with the Meyda library that will be elaborated on in this thesis.

It is important to understand that the JavaScript version of the Essentia library is still in beta and therefore there are still issues in its behaviour. For that reason, studying its performance and finding the cases that work and the ones where exceptions are thrown or there are any bugs is crucial at this stage. These processes and findings will be described and addressed together with the applications.

Chapter 2

State of the Art

Sound and Music Computing (SMC) is an area of research that started in the 1950s but did not gain much attention until the 1970s when the first research groups were born: the Center for Computer Research in Music and Acoustics (CCRMA) at Stanford University, California, USA and the Institute for Research and Coordination Acoustic/Music (IRCAM) in Paris, France [3]. Also in the 1970s the first association and the first journal were founded, the International Computer Music Association in 1974 and the Computer Music Journal in 1977. Sound and Music Computing combines scientific, technological and artistic views with the research goal of comprehending, modelling and synthesising sound and music using computational techniques [4].

It was not until the 1990s when the Music Information Retrieval (MIR) research started gaining its name as a separate sub-field of SMC. According to the Sound and Music Computing Network website[1] MIR is considered to be an area of application of the SMC research together with digital music instruments, music production, digital music libraries, interactive multimedia systems, auditory interfaces and augmented action and perception.

Other researchers define MIR in terms of a field where science, engineering and social studies come together to research "the extraction, analysis, and usage of information

[1] `http://www.smcnetwork.org/roadmap`

about any kind of music entity [...] on any representation level" [5], in other words, "understanding music understanding" [6].

In 2005 J. Stephen Downie wrote for the Annual Review of Information Science and Technology a chapter about what challenges Musical Information Retrieval faces [7]. He described the MIR research as the study for the extraction and inference of meaningful features from music [...] indexing of music using these features, and the development of different search and retrieval systems or methodologies such as music recommendation systems, classification of a large collection of music, content-based search or designs to navigate through music collections [8].

He also analysed the main challenges of the MIR research, naming the one he considered the main one as the Multifaceted Challenge because these different research facets are not mutually exclusive most of the time. He proposed a seven facets approach to describe the different elements MIR research is composed of *Pitch, Temporal, Harmonic, Timbral, Editorial, Textual, and Bibliographic.*

For Downie, this Multifaceted challenge together with the following four are the reasons why MIR research is so demanding and difficult. He named and described these challenges as:

- Multirepresentational Challenge: different ways to represent music
- Multicultural Challenge: different ways in which humans as societies understand and express musicality over history
- Multiexperiental Challenge: different ways in which humans as individuals experience music.
- Multidisciplinary Challenge: different approaches for MIR research

In 2014 Markus Schedl *et. al* published a study of the state of the art of MIR [9] in which they describe the trends of this field of science. He proposed the following division:

Applications MIR

- Music Retrieval
  - Audio identification
  - Audio alignment
  - Cover Song identification
  - Query by Humming and Query by Tapping
- Music Recommendation
- Music Playlist generation
- Music Browsing Interfaces
- Beyond Retraild

There is one thing though that all these trends in MIR have in common, also with other SMC research. As described at the beginning, SMC research combines scientific, technological and artistic methodologies to achieve their goals. It is the use of technologies to study, analyse and extract useful information from music and audio files that all these sciences have in common.

There are several approaches to get to this information, depending on the source used and the technologies involved. Most of these can be classified into three groups:

- Data source: these are based on external data and meta-data not directly extracted from the audio, such as databases and data sets.
- Feature representation: consist of the extraction of features from the audio data that represent characteristics of the audio or music.
- Statistics and machine learning: feeding the audio data directly or not much treated like the FFT into the ML or statistics system to extract results.

Computational MIR approaches typically use features and create models to describe music by one or more of the following categories of music perception: music content,

music context, user properties, and user context [9]. Music content refers to aspects that are encoded in the audio signal, while music context comprises factors that cannot be extracted directly from the audio but are nevertheless related to the music item, artist, or performer.

In this master thesis, the focus is on tools that enable the extraction of features from music content. There exist several technologies allowing this kind of operation. From libraries to use in proprietary software such as MATLAB to others in the form of plugins to use in software like Audacity or Sonic Visualizer like VAMP plugins. The main interest of this thesis is Open Source libraries for use in software developed in Python, C++ and JavaScript. These libraries are used both in research and in industrial applications. There are a few libraries of this kind, but this study is going to focus on the main two libraries used by the MIR community Librosa[2] and Essentia,[3] and the currently most used and recognised by the web audio community Meyda.[4]

2.1 Libraries and what are these used for

Two of the most used libraries in the MIR community are Librosa [10] and Essentia [11]. Librosa is written in Python whilst Essentia is written in C++ with bindings for Python, so both are very used in research Python developments.

The following study focuses on the topics researched making use of these libraries and how have these been used. For this survey, the top 50 most cited papers citing Librosa or Essentia in Google Scholar over the past 5 years have been reviewed.

This information can give us an overview of how researchers are using these types of libraries in recent research, and maybe we can extract some tendency for the coming years.

[2] `https://librosa.org/`
[3] `https://essentia.upf.edu/`
[4] `https://meyda.js.org/`

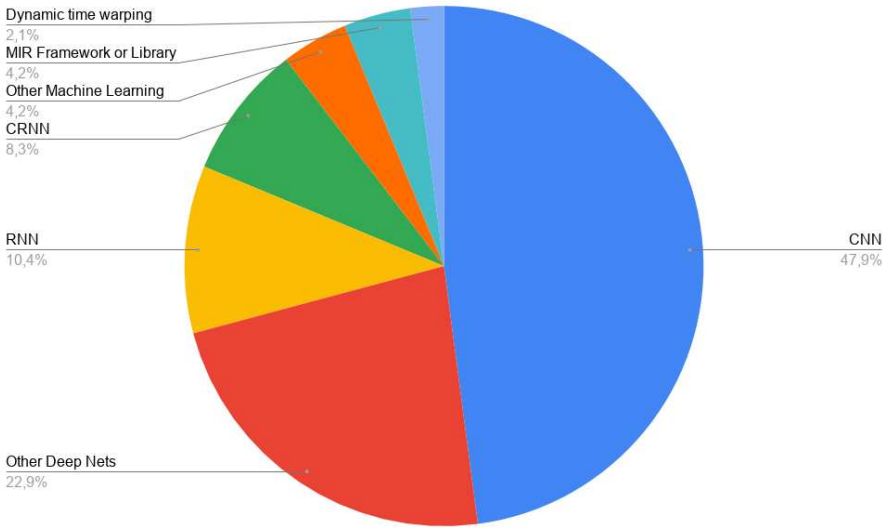


Figure 1: Distribution of techniques used for MIR research in papers using Librosa.

2.1.1 Librosa

Librosa [10] is an Open Source under ISC license Python library for audio and music processing presented in 2015 in the 14th Python in Science Conference. It is now in its 0.8.0 version, what suggests it is not yet ready for production deployment, even it being quite widely used in MIR research projects and has quite a big community with more than 3800 stars and more than 640 forks in its GitHub page.[5]

Reviewing the papers using Librosa, as shown in figure 1 seems quite clear that the most applied technology and techniques in MIR research over the past 5 years are Machine Learning procedures with a clear focus on Neural Networks. Librosa is mostly used to extract features from audio for this kind of experiment in more than 75% of the reviewed papers. References to the reviewed papers can be found in Annex A.

The most common use of the library is to extract features like Mel-spectrogram, Mel band energies, and constant-Q transform to feed them to deep neural networks. This technique is used for a variety of applications being the most popular ones: acoustic scene and music genre classification, sound event detection, and speech synthesis.

[5] `https://github.com/librosa/librosa` (accessed 24/08/2020)

(a) *Word cloud of Librosa algorithms used in the papers.*

(b) *Word cloud of the topics the papers research was about.*

Figure 2: Results of the review of the top 50 papers citing Librosa.

2.1.2 Essentia

Essentia [11] is an Open Source, under Affero GPL v3 license, C++ library for audio and music analysis, description and synthesis. It has Python bindings to be able to use it as a Python library. It was presented at the International Society for Music Information Retrieval in Curitiba in Brazil in 2013. The latest release is the version 2.0.1, having the version 2.1 in Beta. It has more than 1600 stars and more than 370 forks in its GitHub repository.[6] These numbers suggest that the community around Essentia is about half the size of Librosa. Although in this case on the website[7] there is a section with several companies using the library in production environments.

Examining the top 50 papers using Essentia as their library to achieve their research goals, the tendency over these past years to use machine learning methodologies in particular deep neural networks is confirmed. Although in this instance it is not as predominant as in the case of Librosa. Other methodologies and techniques are used when it comes to studies choosing Essentia as their tool. Several papers from this collection make use of high and low-level descriptors, as features to extract from audio, to be used in different ways depending on the research. Some papers make use

[6] `https://github.com/MTG/essentia` (accessed 24/08/2020)
[7] `https://essentia.upf.edu/`

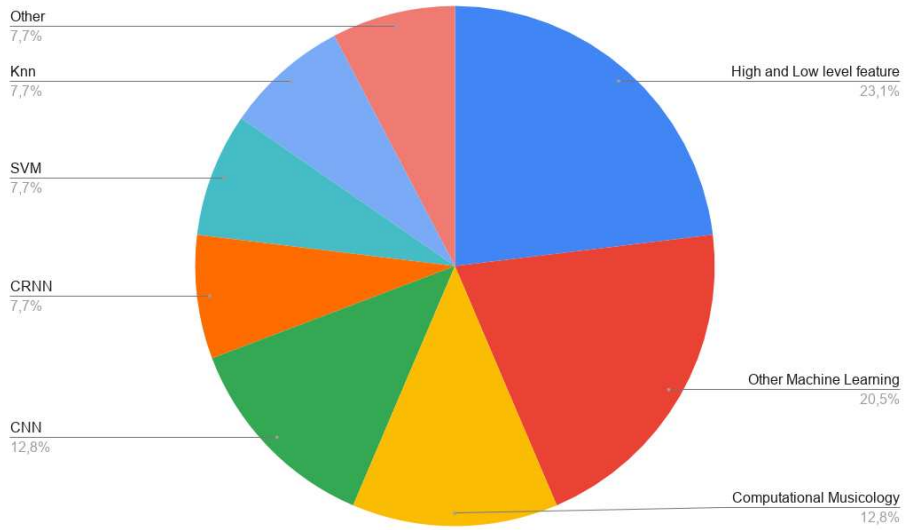


Figure 3: Distribution of techniques used for MIR research in papers using Essentia.

of computational musicology methodologies, other multi-dimensional spaces to find closer distances or knowledge graphs. The references for this 50 papers collection can be found in Annex B.

Figure 4a shows the word cloud of the Essentia algorithms used in these papers. It is noticeable how the variety is quite large compared to the word cloud in figure 2a for Librosa. Mel-spectrograms and MFCCs are still the most used ones to feed this information into the deep neural networks, but other algorithms like Melodia, pitchYinFFT, loudness or key extractor are also used.

Comparing the word clouds of the topics from figure 2b and figure 4b, it is easy to see how there are common topics, but it is also apparent how the variety of topics is wider in the case of papers using Essentia than in the case of papers using Librosa. Some of the main differences are papers presenting works on data sets of several kinds and plugins to be used in other applications or frameworks and libraries. Also, topics such as artificial reverberation and musical culture studies are quite distinct.

(a) *Word cloud of Essentia algorithms used in the papers.*

(b) *Word cloud of the topics the papers research was about.*

Figure 4: Results of the review of the top 50 papers citing Essentia.

2.1.3 Meyda

Meyda [12] is an Open Source library, under MIT license, for audio feature extraction written in JavaScript. It can be used together with the Web Audio API or in plain JavaScript, and both, offline or in real-time if used with the Web Audio API. It was presented in the first Web Audio API Conference in 2015 in Paris. The latest version is 4.3.1 and has 680 stars and 62 forks in its GitHub repository.[8] The community does not seem too big compared to Essentia and Librosa, one reason could be that JavaScript and the Web Audio API are not yet very common technologies for the MIR research community. It is quite known in the Web Audio community though, having its own channel in the web audio slack,[9] a community of developers with 939 registered users.[10] There, authors, users and collaborators can discuss any issues related to the library.

To review the papers using Meyda, a search at Google Scholar[11] for papers citing Meyda was done. It is quite astonishing to see how little the MIR community is using web technologies. Only 20 papers appear in the Google Scholar search.

[8] `https://github.com/meyda/meyda`
[9] `https://web-audio-slackin.herokuapp.com`
[10] as of August 26, 2020
[11] `https://scholar.google.com/`

From those, 15 only cite Meyda as an option of a library written in JavaScript for audio analysis and feature extraction. Some are papers presenting other JS libraries, others just reviews studying the state of art. A couple of these also criticize some aspects of Meyda like that it is not modular nor easy to integrate with other JavaScript modules [13] or the fact that it only has low-level descriptors [14].

From the other papers, three of them are ML-based and the only use they do of Meyda is to extract MFCC to feed the models.

Another of the papers introduces a library written in JS for detecting synthesized sounds [15]. In this case, Meyda is used to extract 17 audio features. The paper only specifies two of them: the power spectrum and MFCC. The last one presents an audience participatory sound art performance titled Precipitate. This work uses Meyda to extract the features: *"clarity, turbidity, pitchedness, strength, spectral centroid, and loudness"* [16]. The latest two are low-level descriptors developed in Meyda but the paper does not describe how the first four high-level descriptors are defined or extracted.

The reference to all these 20 papers can be found in Appendix C.

There are of course other JS libraries, but none of them is better either in performance or in number of features than Meyda. Specifically, the next better known JS library for feature extraction is JS-Xtract[17]. The GitHub repository of this library[12] has been paralysed without any activity whatsoever for more than two years,[13] indicating that there is no interest in continuing its development and support. Only 12 papers citing this library appear in Google Scholar. For those reasons, this master thesis will only compare Essentia.js to Meyda. In a further chapter, the comparison will go into discussing some implementation details and also a benchmarking analysis.

[12] `https://github.com/nickjillings/js-xtract`
[13] accessed on 27-08-2020

2.2 Why Essentia in the Browser

The reviews of Essentia and Librosa show how there is a big research activity in MIR topics, with a growing tendency to use Machine Learning (ML) approaches although having also a variety of methods outside ML. For those reasons having comprehensive and easy to use libraries is important, to allow the community to focus on their research instead of on the tools.

It is quite clear though, that for the MIR community there is a need to cover when it comes to web technologies. While Meyda [12] and JS-Xtract [17] are at the moment, to the best of the author's knowledge, the most used and extensive JavaScript libraries available for audio feature extraction, they offer a very limited collection of algorithms.[14] There is then the need for a comprehensive, modular, and lightweight library, easy to use and integrate into any JS existing platform and framework.That is what Essentia.js aims to cover.

From day one, the Web Audio API was designed to allow developers to write their code in JS accessing the audio data both offline and in real-time. To do so offline the *decodeAudioData* function from the *AudioContext* object provides an *AudioBuffer* with the PCM data of the audio file. This buffer can then call the *AudioBuffer.getChannelData()* function to access the samples of a given channel passed as a parameter [18].

For real-time audio processing development, the *ScriptProcessorNode* was the designed solution. This node works like any other node in the web audio API by connecting them in a node chain. The main particularity of this node is that it gives access to the audio data to the developer to apply any processing algorithms coding directly in JS. This node is now deprecated as it worked running the JS audio processing code in the main UI thread. This can lead to issues in the performance and sound glitches, drop-outs, and stuttering [19]. To fix this, the W3C Audio Working Group[15] proposed the *Audio Worklet* [20] as a solution. It allows the audio

[14] As of August 2020, Meyda only has 20 MIR algorithms.
[15] `https://www.w3.org/2011/audio/`

processing code to run in a separate audio thread while there are bidirectional open communications between the two threads.

The web audio community is stronger every day and the continuous evolution of the Web Audio API confirms so. Several websites are using these technologies as the core of their business like Soundtrap.[16] Others are integrating their services with these technologies like Yamaha[17] and Dolby.[18] However, there is still a tendency to develop musical web services having their audio processing in the backend side using technologies such as Java, C/C++ or Python and then sending the information to the browser. Some examples would be Spotify API(formerly Echonest),[19] Gracenote (Sony),[20] ACR Cloud[21] or Freesound API [21] and AcousticBrainz [22].

Changing this paradigm from doing the computation process in the server to the browser can offer great advantages to web services, as this way the intense part of the computation process is distributed in the web browsers of the users and it is not using any CPU power nor memory from the server.

For these reasons, Essentia.js is a strong candidate to be a reference library for the web audio and MIR communities. It offers a library modular and very easy to use, with a comprehensive set of algorithms for audio and music feature extraction, processing and even synthesis in the web technologies ecosystem, working both on the browser together with the Web Audio API or in any JS-based server.

The main ideas of the Essentia.js project together with a short demo were included in a talk done by the author at the fifth Web Audio Conference in Norway in December 2019, receiving great interest from the assistants.

Full details on the design, architecture and steps for the compilation process to turn Essentia into Essentia.js can be found in a paper pending publication by Albin Correya *et. al* [2].

[16a] urlhttps://www.soundtrap.com/
[17] `https://soundmondo.yamahasynth.com/`
[18] `https://developer.dolby.com/platforms/html5/code/`
[19] `https://developer.spotify.com/documentation`
[20] `http://devapi.gracenote.com/timeline/`
[21] `https://docs.acrcloud.com/docs/acrcloud/`

Chapter 3

Methodology

As explained in the previous chapter Essentia has a big potential of becoming a key tool for audio processing and MIR software in the web technologies world. The first step to achieve having Essentia working with JavaScript was to be able to compile its C++ base code using Emscripten [23] to have it as a WebAssembly binary [24] together with the glue code in JavaScript to be able to call the functions.

Once the compilation of Essentia was working, the next steps were to be considered. After some consideration, the following two were chosen:

- Use of the library by developing examples and real software to test its usability and reliability.
- Benchmarking common MIR use cases to test it's efficiency and compare it to other JavaScript libraries, in this case, Meyda.

These two points were considered from the perspective of testing the new implementation of the library to find where it works as desired and where there were flaws that need careful thought and improvement.

In this chapter, you can find the methodology used for each of these steps.

3.1 How to have Essentia in the Browser

As mentioned before the technologies used for the compilation process of Essentia into Essentia.js are Emscripten, to compile into WebAssembly and Embind to create the bindings.

Emscripten [23] is a toolchain that allows compiling C and C++ code to JavaScript, more specifically ams.js which is a subset of JS corresponding almost to machine instructions, or to WebAssembly making it possible to run native code on the web at "near-native speed".

WebAssembly [24] (or Wasm) is an open standard designed by the W3C and implemented and maintained by Mozilla, Microsoft, Google, Apple, Fastly, Intel, and Red Hat. It is developed to enable high-performance applications usually written in native languages into the web. It defines a binary instruction format and a programming language, being this the fourth language to run natively in web browsers together with HTML, CSS and JavaScript. Its main use is as a portable compiler, so you can have code written in languages like C, C++, Rust or Go among others and deploy them on web pages.

The first step of this thesis involved the research on how to effectively use the Emscripten compiler to be able to have Essentia compiled into a wasm binary to, afterwards, through the bindings done with Embind access from JavaScript the C++ functions. The figure 5 illustrates the process to follow to compile Essentia into a JS library

This research was carried out in parallel by Albin Correya[1] a member of the ASP Lab of the MTG. The final way to compile Essentia.js was developed by Albin. For that reason in this report, this research is not explained with full details other than some difficulties the author found in the process that were finally fixed or worked around. The instructions on how to compile Essentia into Wasm can be found in the Essentia.js documentation website.[2]

[1] https://github.com/albincorreya

[2] https://mtg.github.io/essentia.js/docs/api/tutorial-2.%20Building%20from%20Source.html

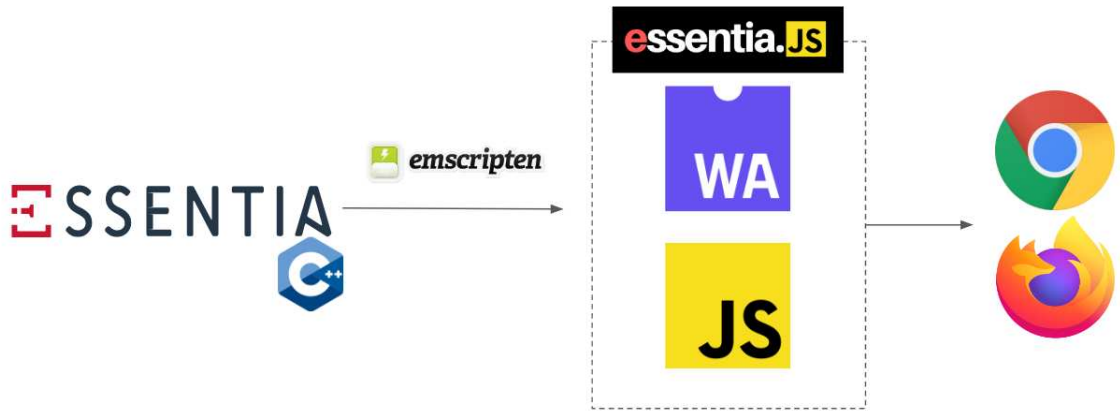


Figure 5: Basic flow diagram of the process to have Essentia in the browser

3.1.1 Compilation issues

Path to emcc

At the beginning of the research, due to a lack of acknowledgement of the compilation process of Essentia both native and with Emscripten, the configure process [25] was failing to find the compiler.

The first trial to attempt to fix this issue was to specify the GCC path to the compiler, doing it as shown in listing 3.1.

```
emconfigure sh -c './waf configure CXX=/usr/bin/g++-7 CC=/usr/bin/
    gcc-7 --prefix=$EMSCRIPTEN/system/local/ --build-static --
    lightweight=fftw --emscripten'
```

Listing 3.1: emconfigure command with paths to gcc7.

Of course, this was not the solution because the emcc compiler, this being the Emscripten compiler, needs to be used. This was fixed by giving access to the emcc path to emscripten adding it to the batch profile.

The argument to '-O'

There was another issue with the 'wscript' configuration file. When compiling the error in listing 3.2 appeared.

The '-O' option is to set compilation optimizations. The compiler tries to reduce code size and execution time. The arguments *'0'*, *'01'*, *'02'* and *'03'* set different levels of opti-

```
cc1plus: error: argument to '-O' should be a non-negative integer,
    'g', 's' or 'fast'
```

Listing 3.2: Error thrown about argument -O.

mization from lower to higher[3], *'s'* optimizes for size, *'g'* optimizes debugging experience and *'fast'* enables all 3 optimizations and other ones that are not valid for all standard-compliant programs.

So the issue was that the argument *'z'* wasn't recognized by Emconfigure. No documentation was found on the argument *'z'*.

To fix it the argument *'-Oz'* was changed by *'-O3'*. In the final version though, it is back to the original value, due to the following case fixing also this issue.

Fail to compile LLVM bitcode

The next error appeared during the compilation process. At this point, there was an issue compiling the code into LLVM bitcode. The message was the one shown in listing 3.3

```
ERROR root: compiler frontend failed to generate LLVM bitcode,
    halting
```

Listing 3.3: Error thrown when failing to build LLVM bitcode.

Researching over the internet similar errors, it was suggested that there was something not properly configured in Essentia compilation files.

Finally, this was fixed by Albin Correya, a member of the ASP Lab of the MTG, by updating the waf version used to compile Essentia. And by adding to the .waf file the command ./waf-light –tools=c_emscripten . As can be seen in the Essentia repository's issue number 806.[4]

3.1.2 FFT libraries

FFTW[5] is a C library for computing the discrete Fourier transform (DFT) in one or more dimensions. Following the Essentia compilation guide[6] FFTW is a dependency of Essentia.

[3] https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html
[4] https://github.com/MTG/essentia/issues/806
[5] fftw.org
[6] https://essentia.upf.edu/installing.html

The command in the listing 3.4 would be the one used to compile Essentia with Emscripten using FFTW as FFT library.

```
emconfigure sh -c './waf configure --prefix=$EMSCRIPTEN/system/
    local/ --build-static --lightweight=fftw --emscripten'
```

Listing 3.4: Emconfigure command with fftw.

When trying to compile Essentia with that command to have Emscripten using FFTW it does not work. It is needed to change the FFT library to KissFFT as shown in listing 3.5

```
emconfigure sh -c './waf configure --prefix=$EMSCRIPTEN/system/
    local/ --build-static --lightweight= --fft=KISS --emscripten'
```

Listing 3.5: Emconfigure command with fftKISS.

3.1.3 KEEPALIVE

While exploring options to compile Essentia with Emscripten, an option provided by the latter was considered by the author. This option consists of adding a tag "KEEPALIVE"[7] next to the function definition target, forcing this way LLVM to not dead-code-eliminate the function, exporting it then to the functions available to be called from JS [26].

This option has as a pro that avoids the need of having to create bindings for all the functions between the compiled wasm and JS. As a big con though, it would mean having to modify the entire C++ library, adding the tag KEEPALIVE to every function we would want to use in JS.

While testing this option the errors shown in the listing 3.6 where thrown by the compiler:

Before finding a solution, an alternative working way was found leaving this option without a complete exploration, as it was considered worst due to having to modify too much the C++ library.

3.2 Benchmark and comparison

It was considered very important for the project to be able to measure the performance of the algorithms compiled into Essentia.js. As the main idea of having Essentia ported

[7] https://emscripten.org/docs/api_reference/emscripten.h.html?#c.EMSCRIPTEN_KEEPALIVE

```
shared:WARNING: object /tmp/emscripten_temp_iXqdK5_archive_contents
   /algorithm.cpp.1.o is not a valid object file for emscripten,
   cannot link
shared:WARNING: object /tmp/emscripten_temp_iXqdK5_archive_contents
   /essentia.cpp.1.o is not a valid object file for emscripten,
   cannot link

error: undefined symbol: _ZN8essentia4initEv
error: undefined symbol: _ZN8essentia8shutdownEv
error: undefined symbol:
   _ZN8essentia15EssentiaFactoryINS_8standard9AlgorithmEE9_instanceE

[...]
```

Listing 3.6: Error when compiling with KEEPALIVE.

into the web was to be able to use it in web pages using the capabilities of the browser a website was built to measure its behaviour and efficiency across different browsers and devices. This way only by accessing a website the measurements could be run.

Benchmarking website

The library used for the benchmarking is Bencharck.js.[8] This library allows high-resolution time measurements and provides statistical results. It has one dependency: lodash.js[9] a library that provides utilities to work with arrays, numbers, objects, strings, etc. Platform.js[10] a library that provides information about the JS platform the code is run into will be needed if this information needs to be added to benchmark data. It is not used in the current version of the code but will be added, as this information is very useful and currently annotated manually.

The library works by creating suites of tests to measure. A suite for each candidate algorithm is created. In the case of algorithms to be compared against its equivalent from Meyda, each library algorithm has a test of a suite.

So basically by using the method 'add' from the Suite object, a test is created where the code we want to measure is the one inside the function sent as a parameter. A suite developed to measure the energy algorithm is shown in the listing 3.7.

[8] https://benchmarkjs.com/
[9] https://lodash.com/
[10] https://github.com/bestiejs/platform.js

```
suite.add('Meyda#ENERGY', () => {
  for (let i = 0; i < audioBuffer.length/HOP_SIZE; i++) {
    Meyda.bufferSize = FRAME_SIZE;
    let frame = audioBuffer.getChannelData(0).slice(HOP_SIZE*i,
       HOP_SIZE*i + FRAME_SIZE);
    let lastFrame;
    if (frame.length !== FRAME_SIZE) {
      lastFrame = new Float32Array(FRAME_SIZE);
      audioBuffer.copyFromChannel(lastFrame, 0, HOP_SIZE*i);
      frame = lastFrame;
    }
      Meyda.extract(['energy'], frame);
  }
}, options)
.add('Essentia#ENERGY', () => {
  switch(frameMode){
    case "vanilla":
      for (let i = 0; i < audioBuffer.length/HOP_SIZE; i++) {
        let frame = audioBuffer.getChannelData(0).slice(HOP_SIZE*i,
            HOP_SIZE*i + FRAME_SIZE);
        if (frame.length !== FRAME_SIZE) {
          let lastFrame = new Float32Array(FRAME_SIZE);
          audioBuffer.copyFromChannel(lastFrame, 0, HOP_SIZE*i);
          frame = lastFrame;
        }
        essentia.Energy(essentia.arrayToVector(frame));
      }
      break;
    case "essentia":
      const frames = essentia.FrameGenerator(audioBuffer.
         getChannelData(0), FRAME_SIZE, HOP_SIZE);
      for (var i = 0; i < frames.size(); i++){
        essentia.Energy(frames.get(i));
      }
      break;
  }}, options)
```

Listing 3.7: Code for the Energy algorithms.

Another interesting feature of the benchmark.js library is that there are several 'on' events to bind functions to. The following 'on' events are used in the website code:

- 'cycle': to send log messages to the console to be able to control the progress of the suite.
- 'start': to add the class 'is-loading' to the button so in the UI is shown that a process is going on.
- 'complete': here is done everything that needs to be done once the suite is finished. Print in the UI the fastest algorithm, remove the 'is-loading' class from the button, show the table with the results the library returns, plot the violin distribution using plotly.js,[11] and download the results in a JSON file.

Finally, to run the suite the function 'run' is called. Once the run function is called, the system executes the tests in the suite as many cycles as is set, in this case just one. One of the most important features in benchmarck.js is that it is designed to provide significant information [27]. For instance, it is designed to do the minimum number of repetitions of each test per cycle to achieve a result with a per cent uncertainty of 1% or less.

In the UI of the website, there are some options to choose.

- Repetitions. It is optional. If it is not filled in the application works with the default behaviour of the library. The library doesn't have an option to customise the number of repetitions per cycle. Nevertheless, after researching if there was a way to do it, an answer in StackOverflow [28] provided the solution. In the listing 3.8 you can see how it works. By setting the *'mintime'* and *'maxtime'* to a negative number and setting *'initCount'* to 1 then *'minSamples'* become the number of repetitions. This is not recommended even by the same people proposing the method, but in our case, we wanted to be able to compare the algorithms using the same behaviour than the benchmark library used in python for the native version of Essentia.

[11] `https://plotly.com/javascript/`

```
const options = repetitions ?
        {
            minSamples: repetitions,
            initCount: 1,
            minTime: -Infinity,
            maxTime: -Infinity,
        }
        : {};
```

Listing 3.8: Options to set the number of repetions per cycle in benchmarck.js.

- Audio Duration. The website provides four samples with different duration to do the test.
- Upload Audio. With this option, a custom audio is uploaded. The previous Audio Duration option is ignored.
- Download Results. If checked, once the suite is done the results are downloaded to a JSON file.
- Use the Essentia FrameCutter algorithm. Whether to use the frame cutter algorithm or a JS for loop to divide the buffer in frames.

Another test case was using Node.js.[12] Node.js is an open-source JavaScript runtime framework to execute JS outside the browser[29]. It is built on Chrome's V8 JavaScript engine. It is commonly used as a server-side technology for web applications.

Benchmarking Node.js was considered because it is a very common platform to develop JavaScript modules and applications outside of the traditional web browser. The same Benchmark.js library and the same approach as explained was used for the test cases implemented to run in Node.js.

The code for this benchmarking website is open-source under license Afero GPL and can be found in the MTG repository.[13] Node.js test cases can be found and executed from the folder *'node'*. A website is available[14] to replicate the test suites.

[12] https://nodejs.org/en/
[13] https://github.com/MTG/essentia.js-benchmarks
[14] https://mtg.github.io/essentia.js-benchmarks/

Real-time

The benchmarking of the library using it in real-time was also considered. After spending some time designing the system and starting the code implementation, the author realised it was not worth it as it wouldn't provide new information. Real-time algorithm implementation in Web Audio API works by sending each frame of audio to the *ScriptNodeProcessor* (or to the *Audio Worklet*) for the required process to be done and sending the results through the chain to the destination. If all this process is done in less time than the time between frames, the real-time audio will work fine without glitches or delays.

Due to that nature, benchmarking any algorithm used in real-time will result in measuring the time the song lasts instead of how much the algorithm takes to process the song. The only way to measure the effective time the algorithm is working is to measure the time for each frame and add them, but that would be the same as a benchmark in offline mode.

To ensure that the algorithms can work in real-time, the only measurement needed is with a frame. If the time needed to process a frame when it arrives at the *ScriptNodeProcessor* (or the *Audio Worklet*) and returns the values is smaller than the time between one sample and the next one, then the algorithm can be used in real-time.

That is what Hugh Rawlinson *et al.* [12] did to justify that Meyda algorithms could be used in real-time. *"[...], we ran benchmarking on Meyda to confirm that it was running faster than real-time audio on a relatively standard device."*

Issues in iOS

When executing the benchmarking in iOS it didn't work. There wasn't any apparent error, so debugging was needed. To be able to debug a browser in iOS from a Linux computer with Chrome or Firefox, it is needed to install a proxy called iOS WebKit Debug Proxy[15] that listens to the usbmuxd daemon over a WebSocket connection, allowing bidirectional communication.

Once the debug system is working, the error in the listing 3.9 message is:

After researching what this error could be caused by, it seems that the browsers in iOS have issues with the Audio Context from the Web Audio API and therefore the applica-

[15] `https://github.com/google/ios-webkit-debug-proxy`

```
Unhandled Promise Rejection: TypeError: Not enough arguments
```

Listing 3.9: Error thrown form iOS.

tions using the standard Audio Context do not work. There is though, a library called standardized audio context[16] that recreates the Web Audio API whilst fixing the errors in iOS. It supports the following browsers: Chrome v81+, Edge v81+, Firefox v70+, Opera v68+ and Safari v12.1+, as of August 2020.

Once this library was installed and the Web Audio API was working, another issue was found. In this case the debugging previously described wasn't giving any information and the behaviour in the iOS device was that after a while executing a test suite the web page refreshed itself.

In this case, it was needed to connect the iOS device to a Mac computer to debug the website using safari in both devices. Doing so the error in listing 3.10 was found.

```
Error] emscripten_realloc_buffer: Attempted to grow heap from
    741277696 bytes to 842006528 bytes, but got error: Error: Out of
     memory
  emscripten_realloc_buffer (essentia-wasm.web.js:27:136622)
  _emscripten_resize_heap (essentia-wasm.web.js:27:137466)
  wasm-stub
  <?>.wasm-function[7493]
[...]
  <?>.wasm-function[7602]
  wasm-stub
  (funcio anonima) (essentia-wasm.web.js:27:32815)
  dynCall_iiiiii_210 (Script anonim 1 (linia 3))
  EssentiaJS$FrameGenerator (Script anonim 2 (linia 10))
  (funcio anonima) (suite_energy.js:66)
[...]
  run (benchmark.js:2114)
  execute (benchmark.js:860)
  (funcio anonima) (lodash.min.js:26)
```

Listing 3.10: Error: Out of memory.

This error was caused then by the browser running out of memory to do the benchmarking process. Thanks to the option "Repetitions" were the user can set the number of repetitions to be carried out for each test per cycle, the benchmarking in iOS could be done. It was done doing only 5 repetitions per algorithm. These results are not statistically as significant

[16] https://github.com/chrisguttandin/standardized-audio-context

as the rest of cases, but at least they provide an accurate enough idea of the times for the browsers in iOS.

3.3 Application using Essentia.js

Once Essentia has been successfully ported to Essentia.js, being now a JS library to use in the web, starting to use it by integrating it in web applications is a key next step for the project. It is very important to start utilising the library in real cases to find out to what extent it is reliable and whether there are points to fix and improve.

3.3.1 Detection of Audio Problems in Music

The idea of this part of the project is to build an application that could fit the requirements of a minimum viable product (MVP) for a project for SonoSuite. SonoSuite (or SNS) is a company that offers a Software as a Service in the form of a web platform for digital music distribution. The web application allows its users to create and manage their music catalogue, creating their albums and uploading their music. Then they can distribute their releases to several digital service providers (DSP) such as Spotify, Apple Music, Youtube Music and many others around the world. The service SNS offers also includes the management of the royalties generated from the music.

It is of critical importance to ensure that the music sent to the channels is within the music industry quality standards. For this reason, SonoSuite has a Quality Control (QC) department in charge of ensuring that. The methodology used for the Quality Control agents is nothing better than manual operations, therefore it is very time consuming and prone to mistakes. To improve this situation and to provide a helping tool to the QC agents, SonoSuite started to look for automatization of the process.

With that goal in mind, along 2018 and 2019 SonoSuite and the MTG collaborated in a research project intending to develop algorithms in C++ as part of Essentia that allowed the detection of audio problems in music files. The results of this project end up being presented in a paper called Automatic Detection of Audio Problems for Quality Control in Digital Music Distribution [30] at the 146th Audio Engineering Society Convention in Dublin in March 2019.

Now that these algorithms are available in Essentia and therefore in Essentia.js a new opportunity arises for SonoSuite. Following the change in paradigm explained in section 2.2, thanks to having these algorithms ready to be used in the clients' browser, now the audio problems can be detected before the songs are sent to the server. This way the platform can inform the client right away if their song has any issue they need to take care of before uploading the audio master to be checked by the QC team. This will help fasten the process both for the SNS team and for the clients.

The specifications for the MVP would be:

- Render the waveform of the audio file
- Process some audio problems algorithms for the audio file in the browser
- Highlight the zones where audio problems have been found
- A legend explaining the relation between colours of the highlighted areas and the audio problems.

Implementation of the application:

- UI of the application: The application is very simple, it consists on a section at the top where the audio waveform will be rendered once the song is loaded and just under it a buttons section with 4 buttons: play/pause, stop, loop, and audio problems. Implementation of the UI can be seen in the figure 6.
- Design and usability of the application: The application is designed to be a simple music player with the possibility of creating a loop in any segment of the song.
  - The button "Play/Pause" starts and holds the reproduction of the song. After a pause, the song resumes from where it was stopped.
  - The "Stop" button ends the reproduction of the music and resets the playing position to the beginning of the track.
  - The "Loop" button enables the loop reproduction option. To set the two points "from" and "to" of the loop, just by double-clicking in the waveform a vertical red line will indicate each location.

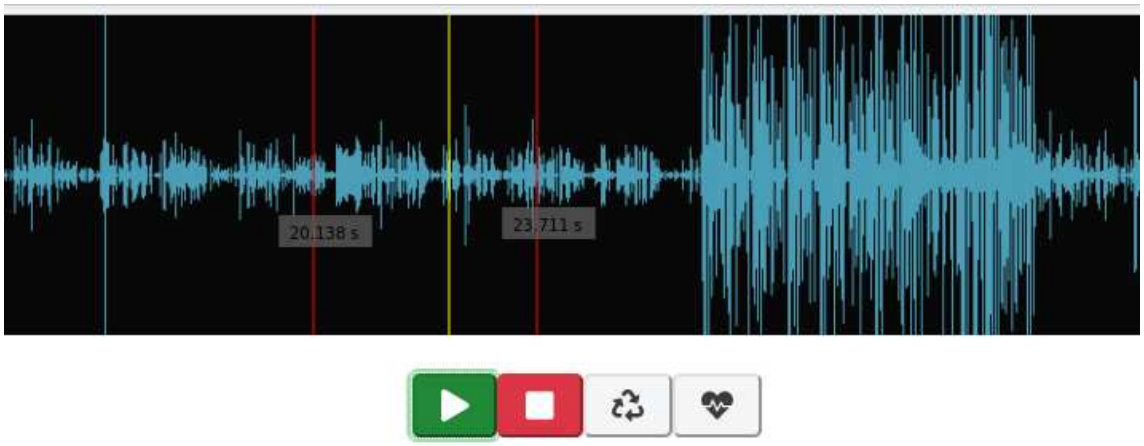


Figure 6: UI of the application to detect audio problems

  - The "Audio Problems Analysis" button will trigger the Essentia.js algorithms to detect whether there are issues in this track and mark them in the waveform.

- Loading and using Essentia.js: there are different ways of using Essentia.js. The project documentation website[17] describes very well how you can get Essentia working in your application whatever JS environment you are using. There are also the steps needed to compile Essentia into Essentia.js.

This application uses Essentia.js as a full JS library. It also uses another way to have Essentia algorithms in the browser. This second way is with custom extractors. This option will be explained with more detail in the following section. Essentia.js can be loaded into the browser to make use of the entire library by loading the files through the HTML script tags as in the listing 3.11.

```
<script src="https://unpkg.com/essentia.js@0.0.9-dev/dist/essentia-
    wasm.web.js"></script>
<script src="https://unpkg.com/essentia.js@0.0.9-dev/dist/essentia.
    js-core.js"></script>
```

Listing 3.11: HTML script tags to load Essentia.js from a CDN.

Then as in the listing 3.12 an instance of Essentia.js can be created from a promise.

And finally, the algorithms can be used as shown in the example in listing 3.13.

[17] https://mtg.github.io/essentia.js/docs/api/

```
EssentiaModule().then( (EssentiaWasmModule)=> {
    this.essentia = new Essentia(EssentiaWasmModule);
});
```

Listing 3.12: Load Essentia Module and create Essentia instance

```
const frames = this.essentia.FrameGenerator(trackBuffer.
   getChannelData(0), FRAME_SIZE, HOP_SIZE);
for (let i = 0; i < frames.size(); i++) {
  let frame_windowed = this.essentia.Windowing(frames.get(i), true,
      FRAME_SIZE);
  // SNR
  snrResults.push(this.essentia.SNR(frame_windowed['frame']));
}
```

Listing 3.13: Example using Essentia.js algorithms

The application will then execute all the algorithms for the audio problems. If any anomaly is found, the zone where these are will be highlighted by rendering a colour area using the Canvas API on top of the waveform.

The code for the application is Open-Source under license Afero GPL and can be found in a GitHub Repository.[18] A demo website is available as well.[19]

3.3.2 Custom extractors

A custom extractor is a different way of using Essentia in the browser. Instead of loading the entire library, in this case a custom use of the library is programmed in C++ and then compiled with Emscripten. This is a very interesting way to use Essentia.js as loading the full library can be overdoing if there is a specific need.

In the GitHub page, although it is still in the 'dev' branch there is a good explanation of how to build a custom extractor.[20]

For this master thesis, custom extractors have an extra functionality. As will be discussed in the results section, some of the algorithms to detect audio quality problems do not work in the way Essentia.js is compiled. For more details into this, please read section 4.2.

The solution to be able to use these algorithms is to create a custom extractor for them.

[18] https://github.com/ljoglar/wave-form-marker
[19] https://ljoglar.github.io/wave-form-marker/
[20] https://github.com/MTG/essentia.js/tree/dev/src/cpp/custom (as of 29/08/2020)

To facilitate this process an automation process, to create the C++ code needed to be able to compile these functions into their own custom extractor, has been developed. This automation has been done using the Python file generation library cog.[21] The system is built in the form of a template folder with the structure shown in the figure 7. These files have at the top a JSON object with the configuration needed to create the C++ code.

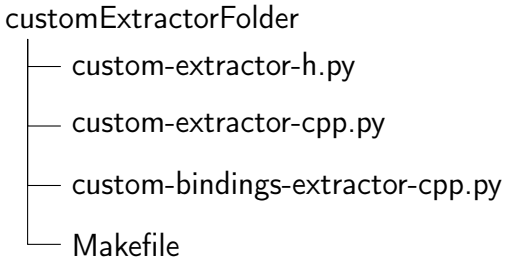


Figure 7: Folder structure of the template.

An example of the configuration needed to create an extractor for the "StartStopSilence" Essentia's algorithm is shown in listing 3.14, listing 3.15, and listing 3.16 followed by the structure explained for any other custom extractor.

```
extractors = {
    'StartStopSilenceExtractor':{
        'params': 'const int frameSize=512, const int hopSize=256',
        'algorithms': ['FrameCutter', 'StartStopSilence'],
        'compute_return': 'int'
    }
}
```

Listing 3.14: Configuration for custom-extractor-h.py.

The values for the arguments params, algorithms and compute-return of the file *custom-extractor-h.py* as shown in listing 3.14 are:

- Name of the Custom Extractor
  - params: String with the parameters to pass to the constructor and configure functions
  - algorithms: Array with the names of the Essentia functions to be used in the extractor
  - compute return: String with the variable type that the compute function will return

```
extractors = {
    'StartStopSilenceExtractor':{
       'params': 'const int frameSize, const int hopSize',
       'algorithms': {
            'FrameCutter': {
                'params': ["frameSize", "frameSize", "hopSize", "
                   hopSize", "startFromZero","true"],
                'inputs': {'signal': 'std::vector<float>'},
                'outputs': {'frame': 'std::vector<Real>'}
            },
            'StartStopSilence': {
                'params': [],
                'inputs': {'frame': 'std::vector<Real>'},
                'outputs':{'startFrame': 'int', 'stopFrame': 'int'}
            }
       },
       'compute_return': 'int'
    }
}
```

Listing 3.15: Configuration for custom-extractor-cpp.py.

The values for the arguments of the JSON configuration of the file *custom-extractor-cpp.py* as shown in listing 3.15 are:

The first element is the name of the Custom Extractor.

- params: String with the parameters to pass to the constructor and configure functions
- algorithms: Algorithm name (following Essentia documentation)
  - params: Array with pairs of param name (following Essentia documentation) and variable name or value hardcoded
  - inputs: Dictionary with pairs input name (following Essentia documentation) and a string with the variable type of that input
  - outputs: Dictionary with pairs output name (following Essentia documentation) and a string with the variable type of that output
- compute return: String with the variable type that the compute function will return

[21] `https://nedbatchelder.com/code/cog/`

```
extractors = {'StartStopSilenceExtractor': {'params': 'int, int'}}
```

Listing 3.16: Configuration for custom-bindings-extractor-cpp.py.

The values for the argument params of the file *custom-bindings-extractor.py* as shown in listing 3.16 are:

- Name of the Custom Extractor
  - params: String with variable types of constructor parameters

Once these files are filled in with the configuration, the next step is to call the cog library to create the C++ files. By calling the commands in the listing 3.17 in the terminal from inside the template folder, the C++ files for the custom extractor will be created.

```
cog -d -o custom-extractor.h custom_extractor_h.py
cog -d -o custom-extractor.cpp custom-extractor-cpp.py
cog -d -o custom-bindings-extractor.cpp custom-bindings-
    extractor_cpp.py
```

Listing 3.17: Command to create the C++ files.

The last step is now to compile the C++ files to create the JS module and wasm binary to upload to the website to use the custom extractor. By following the instructions in the GitHub page to compile custom extractors, the final files structure will be like the shown in figure 8.

Once the files are created it is recommendable to not load the module in a synchronous way using the main JS thread. Otherwise, Chrome will throw the error shown in figure 9

```
<script src="/assets/essentia-custom/saturationExtractor/essentia-
    custom-extractor.web.js"></script>

EssentiaWASM().then( (EssentiaWasmModule)=> {
  this.essentiaSaturationExtractor = new EssentiaWasmModule.
     SaturationDetectorExtractor(512, 256);
  console.log(this.essentiaSaturationExtractor);
});
```

Listing 3.18: Code to load the custom extractor to a web application.

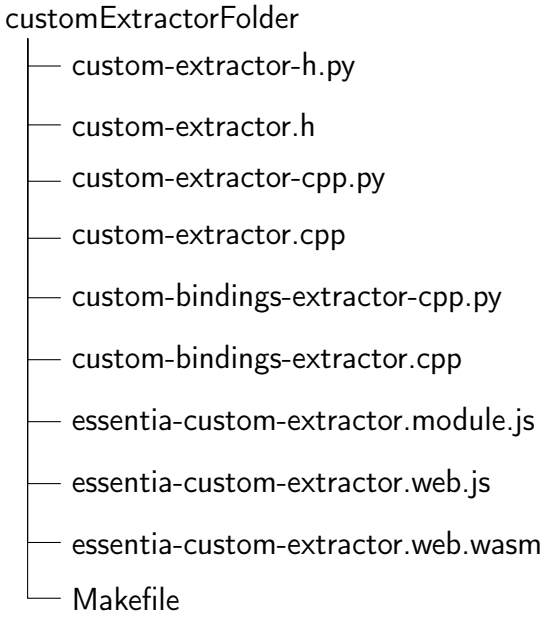


Figure 8: Folder structure after compiling the custom extractor.

Figure 9: Error thrown in Chrome when loading the custom extractor synchronously.

The best way to upload the custom extractor to the web page is by storing locally (or on a CDN or similar service) the files *essentia-custom-extractor.web.js* and *essentia-custom-extractor.web.wasm* and loading them through the HTML script tags. Then, loading the module and creating and instance of the extractor using a promise as shown in the listing 3.18.

The code for the custom extractor automation can be found in the Essentia.js repository fork of the author,[22] as it is not yet merged into the main repository.[23]

[22] `https://github.com/ljoglar/essentia.js`
[23] in the branch *custom_saturation_detector*

Chapter 4

Results

This chapter presents the results of the elements developed following the explanations on the section Methodology. On one hand the benchmarking of the algorithms of Essentia.js and the comparison to Meyda.js, and on the other hand, the results of the application built to detect audio problems in music files to improve the quality control process for digital music distribution.

4.1 Benchmark and comparison

The benchmarking was done for common MIR audio features on various devices and browsers, and on node.js. The analysis times (also called execution times) for the feature extraction were measured using the Benchmarks.js library. In the case where both libraries have the same algorithms the two execution times are compared. In other cases where Meyda.js doesn't have the algorithms, only the times for Essentia.js are displayed. As a base comparison element, also the times for Essentia in Python have been measured, using the equivalent code in Essentia and Essentia.js. The benchmarking of Python implementation was done using the library *pytest* with the *benchmark extension.*[1]

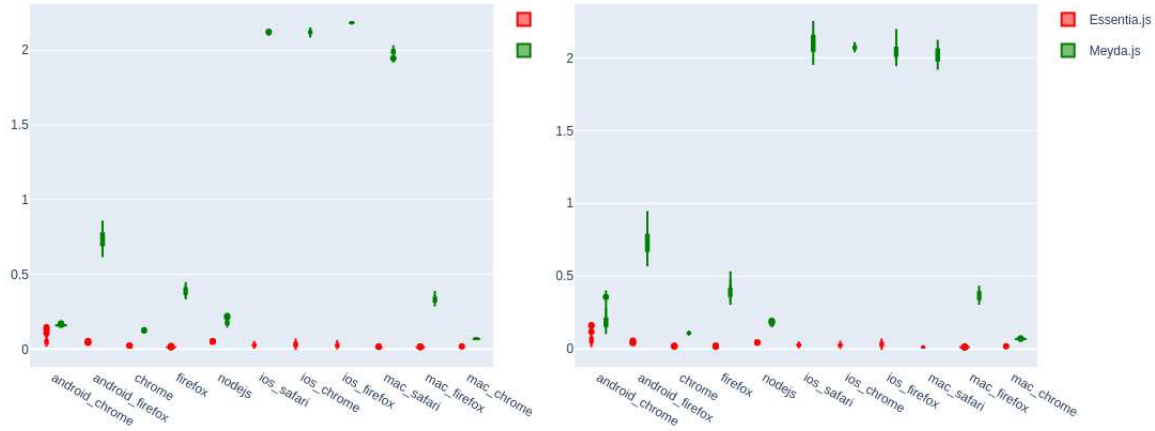
[1] `https://pytest-benchmark.readthedocs.io/en/latest/`

The following environments have been used for executing the tests:

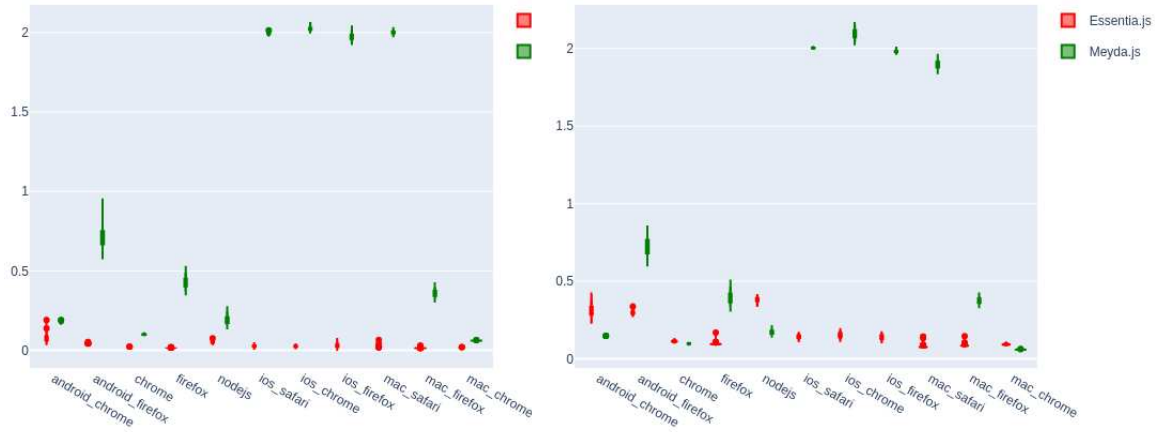
- Linux with Chrome 84.0.4147.89 run with disabled extensions.
- Linux with Firefox 78.0.2 in private browsing mode.
- Linux with Node.js v.13.13.0.
- Android with Chrome 84.0.4147.89 in incognito mode.
- Android with Firefox Nightly 200727 06:00
- MacOS with Chrome 84.0.4147.135 run in incognito mode.
- MacOS with Firefox 79.0 in private browsing mode.
- MacOS with Safari v.13.1.2 (13609.3.5.1.5)
- iOS with Chrome 85.0.4183.72 in incognito mode.
- iOS with Firefox Focus v.28.0
- iOS with Safari 13.1

The specifications of the devices are:

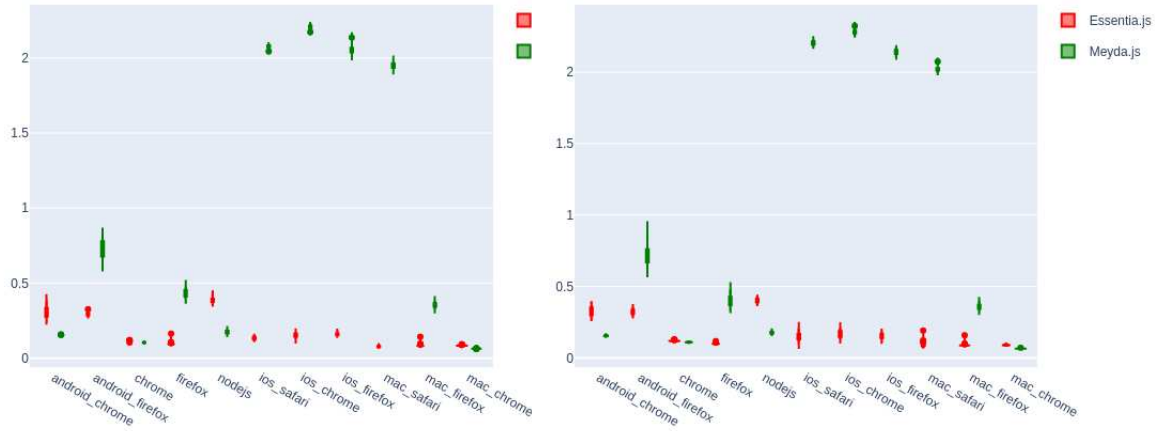
- The Linux computer is a 2017 DELL XPS-15 with a 2.8GHz x 8 Intel Core i7-7700HQ processor, 16GB of RAM and as O.S. Ubuntu 19.04.
- The mobile phone is a Xiaomi Redmi Note 7 Pro with a Snapdragon Octa-core 1.7 GHz processor and 6GB RAM, with O.S. Android 9 (LineageOS 16).
- The Mac is a MacBook Pro (Retina, 15-inch, Mid 2014) with 2.8GHz quad-core Intel Core i7 and 16GB RAM, with MacOS 10.13.6 (High Sierra)
- The iOS is an iPhone 8 with a processor A11 (Hexa-core) and 2 GB RAM with iOS 13.5.1



(a) *Energy* (b) *RMS*

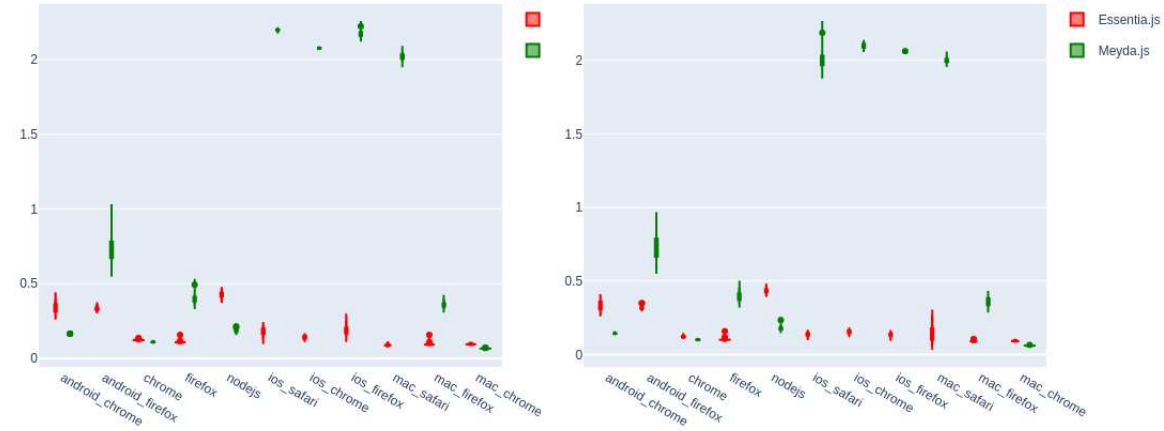


(c) *Zero Crossing Rate* (d) *Amplitude Spectrum*

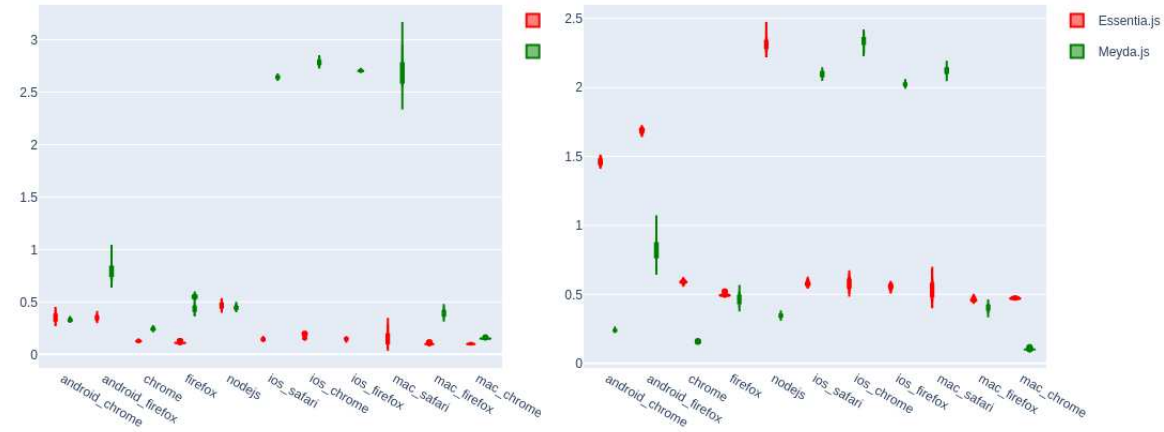


(e) *Power Spectrum* (f) *Spectral Centroid*

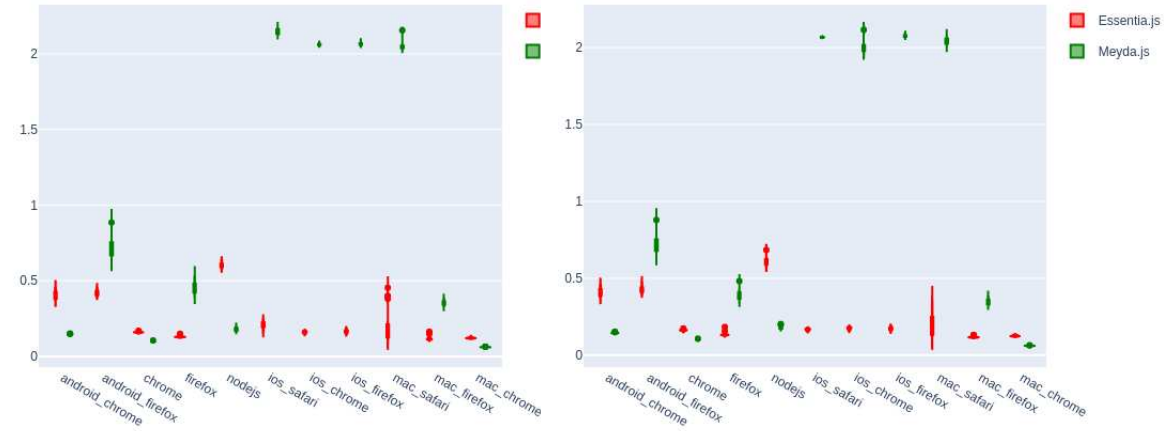
Figure 10: Distribution of execution times (seconds) per platform



(a) *Spectral Flatness* (b) *Spectral RollOff*

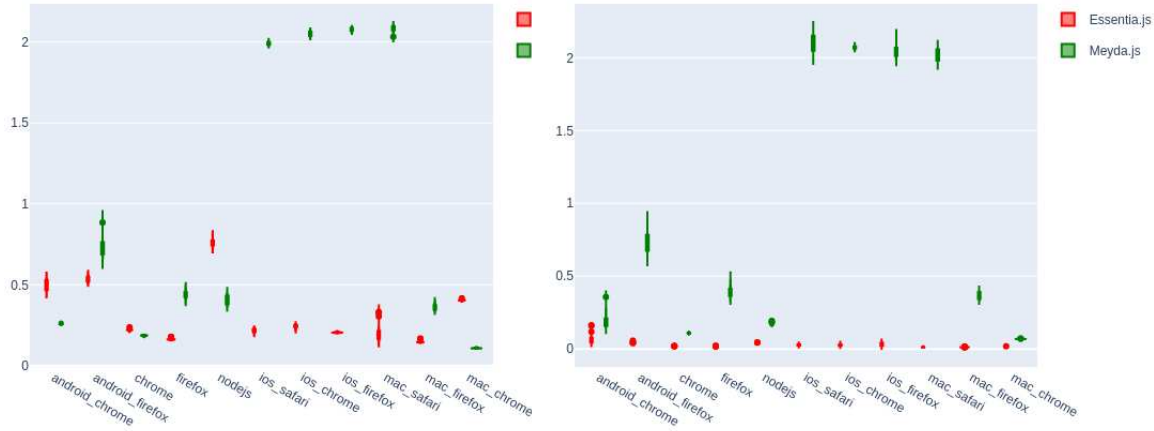


(c) *Spectral Kurtosis, Skewness, Spread* (d) *MFCC*



(e) *Loudness* (f) *Perceptual Spread*

Figure 11: Distribution of execution times (seconds) per platform



(a) *Chroma* (b) *All time domain and spectrum algorithms (12)*

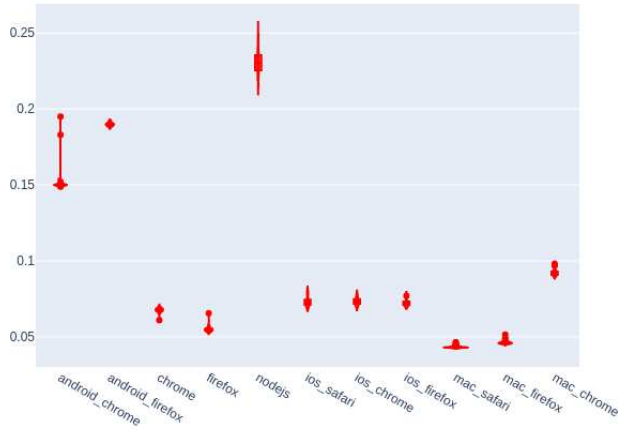
Figure 12: Distribution of execution times (seconds) per platform

The times measured correspond to the entire processing chain, from the moment the entire audio data is received until the moment the last algorithm finishes the analysis. All the experiments are done using a 5-second audio segment as input. The same analysis has been done with longer audio files, but the results are not shown as these do not provide new information, having the execution times a quite linear relationship to the audio segment duration.

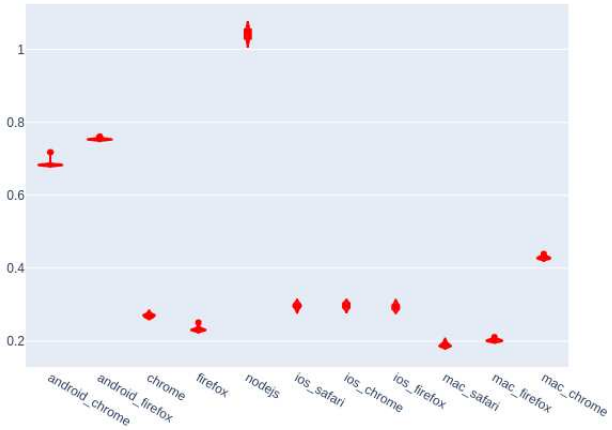
Figure 10, figure 11, figure 12 (14 plots) show the distribution of execution times for each algorithm depending on the platform and browser comparing the distribution of execution times for Essentia.js and Meyda.js. It is very interesting to see how Essentia.js is in most of the algorithms faster and very much consistent across platforms compared to Meyda.js. Being the latter very slow in Safari with macOS and iOS.

Meyda.js is a bit faster, although by a small difference in most algorithms in Chrome, except for Chrome in iOS, and in node.js. Results for node.js will be analysed later.

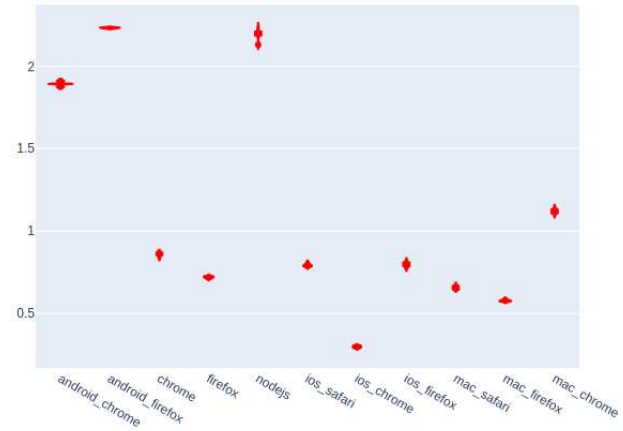
In the case of the figure 12b Essentia.js is faster in all the cases, suggesting that is more efficient when it comes to executing a complex chain of algorithms than Meyda.js.



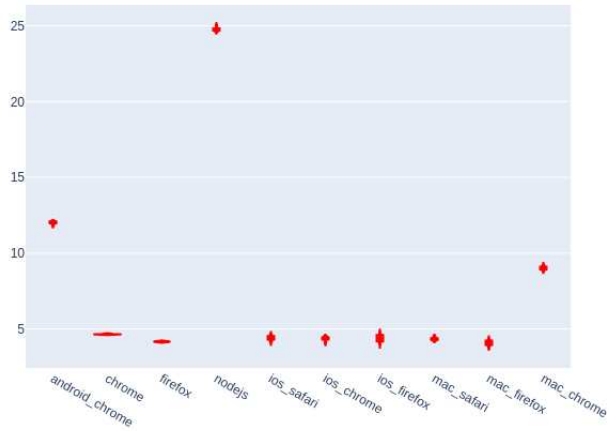
(a) *Onset*



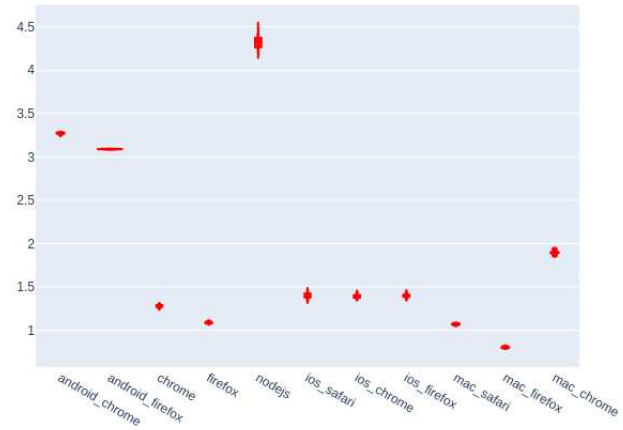
(b) *Super Flux*



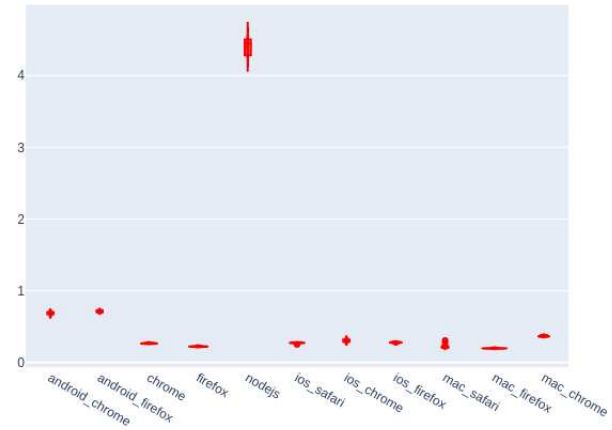
(c) *Beats Zapata*



(d) *Yin*

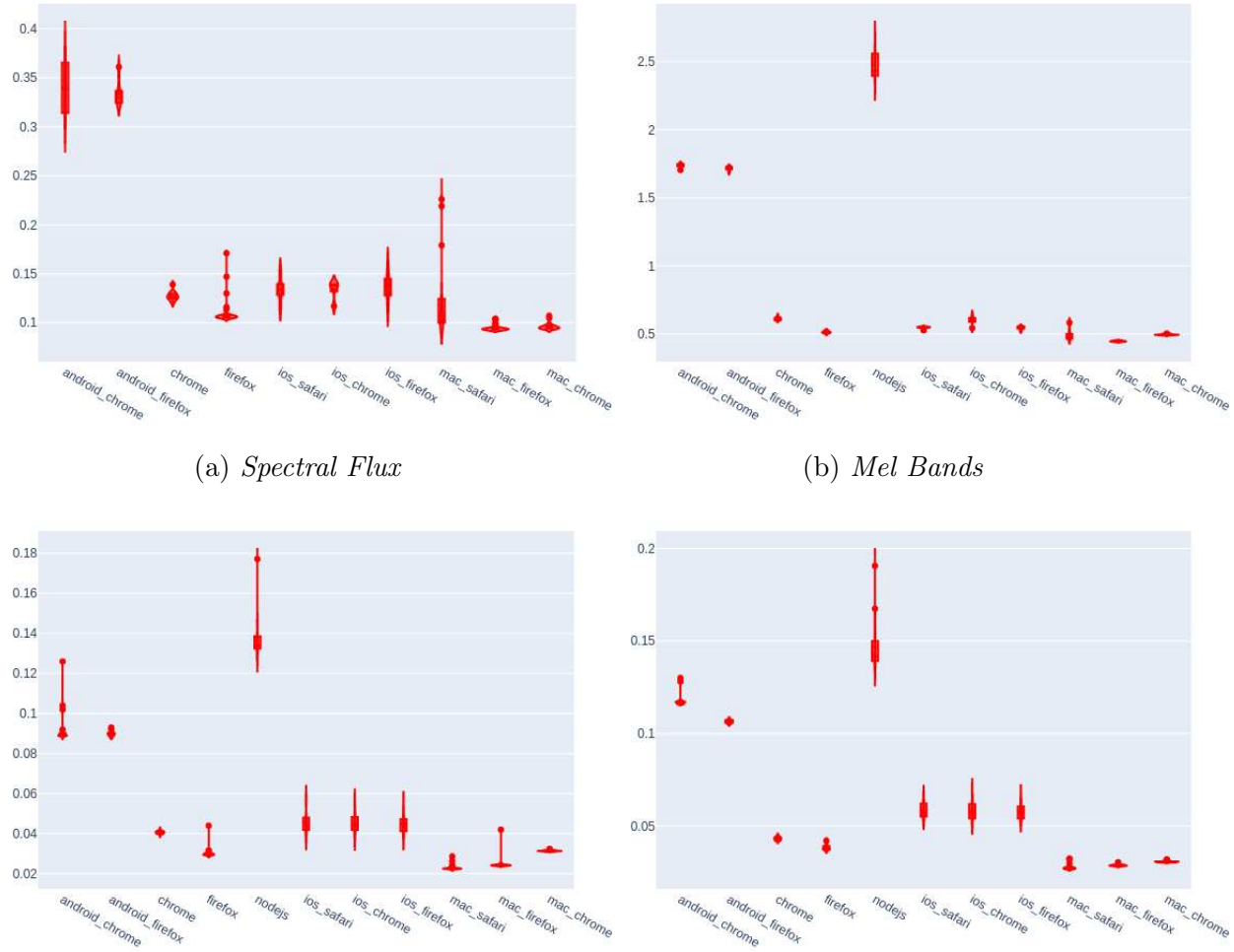


(e) *Pyin*



(f) *Pyin fft*

Figure 13: Distribution of execution times (seconds) per platform



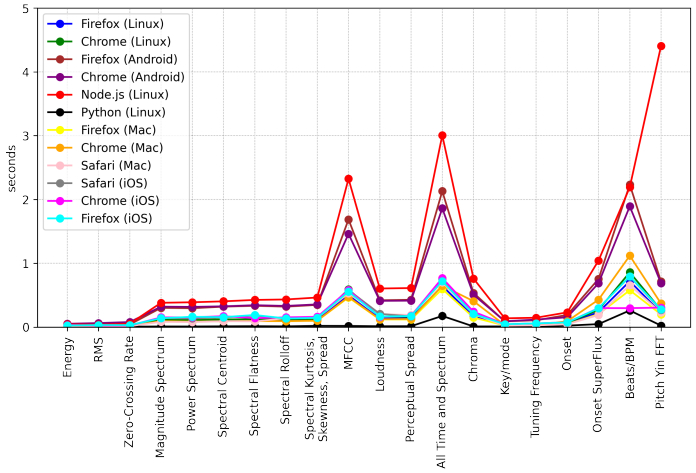
(a) *Spectral Flux* (b) *Mel Bands*

(c) *Key Extractor* (d) *Tuning Frequency*

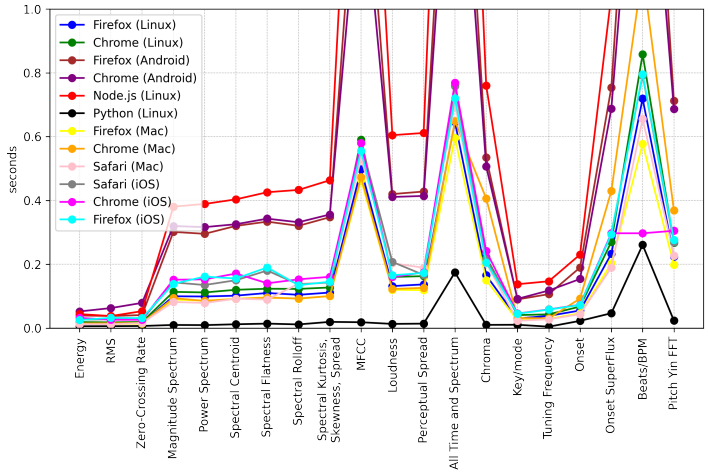
Figure 14: Distribution of execution times (seconds) per platform

In figure 13 and figure 14 are shown the distribution times depending on the environment of a few algorithms that are not implemented in Meyda.js. It is important to take into account when looking at these plots that the time scale is not fixed, it depends on the times of execution of the algorithms.

These show that the platforms are quite consistent on their behaviour with respect to the times of executions of the algorithms, as all the plots have a similar shape indepently of is a fast algorithm as *Onset* in figure 13a or a slow one as *Pithc Yin* in figure 13d. Clearly node.js has the worst times followed by the browsers in Android.



(a) *Full time scale*

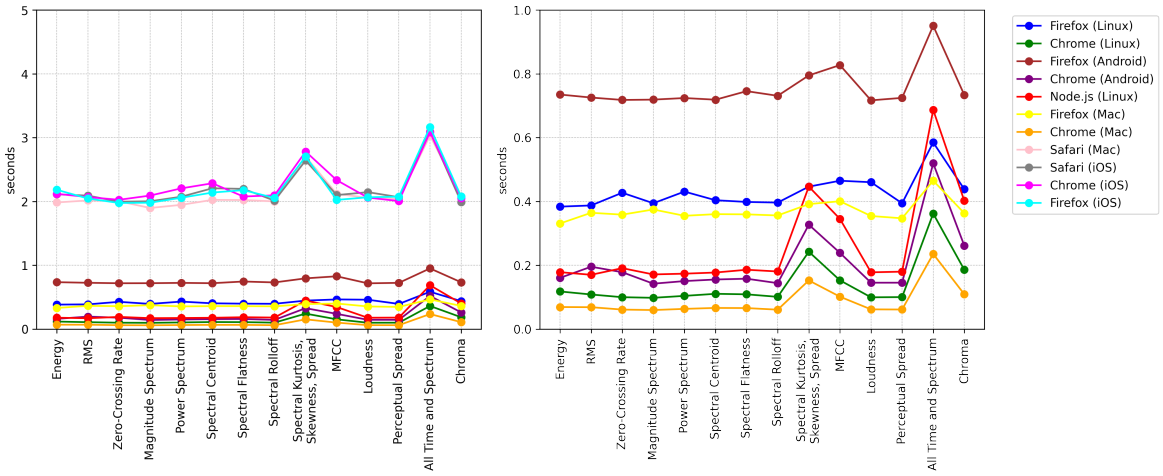


(b) *Time scale between 0s and 1s*

Figure 15: Mean execution times (seconds) for Essentia algorithms

The plots in figure 15 and figure 16 show a comparison of the mean execution times for each algorithm depending on the environment executed. The first two figures show Essentia.js algorithms and the latter the equivalents if exist for Meyda.js.

In this case is easier to see how Essentia.js is more consistent having most algorithms with mean times under the 0.2s except for the platforms FirefoxAndroid, ChromeAndroid and Node.js, and four tests MFCCs, All Time and Spectrum, Onset SuperFlux and Beats/BPM. It is expected though that these algorithms take more time, except MFCC, which is something to study further, to understand the reason for this peak.



(a) *Full time scale* (b) *Time scale between 0s and 1s*

Figure 16: Mean execution times (seconds) for Meyda algorithms

About Firefox and Chrome in Android, there is not a clear explanation of why these are slower and it would be interesting to analyse further, but it is clear that the processor has less computing capabilities and also Android is a slower OS than Ubuntu and macOS.

Surprisingly, the results for iOS are very good for Essentia.js, having the mean times close to the computers running Linux and macOS. Even more surprising when looking at the results for Meyda.js where the three browsers in iOS together with Safari in macOS get the worst times.

It is important to mention that to be able to have values for the benchmarking on iOS it was necessary to set the option repetitions to 5. Otherwise, if more repetitions were done, the benchmarking failed to throw an *"out of memory"* error. It would be interesting to continue exploring how Essentia.js behaves on iOS devices, as optimizations may be needed.

Finally, node.js is a special case that needs consideration. It was very surprising seeing the results for this platform. It was not expected to have results so different from the browsers in the same computer, specifically Chrome as the JavaScript base engine is the same. Web Assembly and the WASM standard could be a reason to take into account, as is a relatively new technology under active development and with many proposals still being considered[2], such as SIMD optimizations and multi-thread capabilities.

[2] `https://webassembly.org/roadmap/`

Another point to consider when it comes to node.js is to study how it works concerning the garbage collection,[3] to avoid having any memory leak in our implementation.

4.2 Application using Essentia.js

A website application was built using Essentia.js algorithms to be able to detect audio problems in music files to improve the quality control process for digital music distribution. This way if a problem is found in the music a user uploads to the web, almost immediate feedback is provided informing the user about the issues in the track for her to take action on them.

Details about the design and the UI of the application are explained in the section *Methodology*. While implementing the application, issues were found in some Essentia.js algorithms.

The following list looks over the algorithms intended to be used in the application, explaining whether an issue was found for each of them and the solutions carried out. All these algorithms are included in Essentia and documentation can be found in the website.[4]

LoudnessEBUR128 This algorithm expects the two channels of a stereo file. At the moment Essentia.js does not support this kind of object. The solution for this is, as explained in it's GitHub issue[5] to develop a custom wrapper that allows sending the two channels separately. This still needs to be released in the main build of Essentia.js, for that reason it is not implemented in the application. Once it is released it will be integrated.

FalseStereoDetector This algorithm is similar to the previous one, as it also expects as input the data for the two channels of a stereo file. As in the previous case, this is not available yet. The solution would be the same as for the LoudnessEBUR128: create a custom wrapper that allows sending the data of each channel separately.

StartStopCut and SNR. These algorithms are working fine. The first one detects if there are any cuts at the beginning or the end of the audio file and the second one evaluates the signal to noise ratio returning three values per frame: instant SNR, averaged SNR, and spectral SNR.

[3] `https://blog.risingstack.com/node-js-at-scale-node-js-garbage-collection`
[4] `https://essentia.upf.edu/algorithms_reference.html`
[5] `https://github.com/MTG/essentia.js/issues/28`

HumDetector, TruePeakDetector and GapsDetector when executing them a compilation error is thrown. The type of error can be seen in the listing 4.1. This algorithms need to be studied to find why the error is thrown.

```
essentia.HumDetector(this.essentia.arrayToVector(trackBuffer.
    getChannelData(0)));
essentia.TruePeakDetector(trackBufferData);
essentia.GapsDetector(frames.get(i));

essentia-wasm.web.js:27 Compiled code throwing an exception,
    10071176,27820,440
essentia-wasm.web.js:27 Compiled code throwing an exception,
    6083720,27820,440
```

Listing 4.1: Compilation Wasm error.

StartStopSilence, SaturationDetector, DiscontinuityDetector, ClickDetector and NoiseBurstDetector. All these algorithms have an internal counter to know the number of frames that have passed. This way the return values take into account the position in the track. Due to a decision in the design of Essentia.js, every time a new sample gets into the algorithm the count resets losing track of the position in the audio file, therefore the return values are not valid. To be able to use these algorithms a custom extractor for them need to be created as explained in section 3.3.2 *Custom Extractor*.

Chapter 5

Futher work and conclusions

The work presented in this Master thesis is just a part of the work of a bigger project by the Audio Processing Lab at the Music Technology Group of the Universitat Pompeu Fabra to turn the C++ library Essentia into a reference library in the Web Audio world. Providing this way with a reliable and comprehensive audio analysis JavaScript library for Music Information Retrieval and Sound and Music Computing research on the web.

For this goal to be achieved there is still a long path to follow, although the first steps into it are already done as demonstrated in this thesis. The library can now be used, even in some cases is needed to build a workaround like a custom extractor because the main algorithm in the library has some kind of issue.

5.1 Further Work

One of the most important steps to be done is thorough testing of the JS version. Starting by unit testing and also integration testing using audio samples with known results over the algorithms, this way ensuring the algorithms work as they should.

Regarding the parts of the project shown in this thesis, there is work to do derivated from the benchmarking. Research into why some algorithms take unexpectedly long to execute like in the case of the MFCC. It would be also interesting to take a deeper look at why node.js is the slowest environment. This research could lead to some optimisation and refinement important to improve the efficiency of the library.

As mentioned in the results the initial analysis done with different audio lengths seem to lead to the conclusion that the relationship between the length of the audio and the execution time of the algorithms is linear. Nevertheless, ensuring that this is the case with a thorough test case design for the purpose would be also an interesting research. There could be things like garbage collection issues or other elements that differ in behaviour from C++ to JS, offuscated.

Another interesting task to do, related to the algorithms used in the Audio Problems detection application is to fix the design issue of having config and run in the same step. As explained that causes issues in some algorithms that have an internal count and with this design it resets every new sample arrives. At the moment the only way of working with these algorithms is to create a custom extractor.

5.2 Conclusions

This Master thesis reports a few elements that were researched by the author as part of the project of transforming the library Essentia into a modular, easy-to-use, reliable, and comprehensive audio analysis JavaScript library to use in the web.

A quite thorough analysis of the trends in MIR research using music and audio analysis and feature extraction libraries over the past five years has been done, coming to the conclusion that while Machine Learning technologies are clearly the most used at the moment, there is a wide range of topics being researched and there is a lack of tools allowing these researchers to get their studies into the web world.

That is the hole Essentia.js wants to cover. The initial context and steps of the research into how to compile Essentia into Essentia.js using Emscripten to get a WebAssembly compiled code together with a JS binding code has been discussed leaving the final solution and instructions on how to do it out of the scope of this thesis. It can be found in the pending publication paper [2] and in the website of Essentia.js.[1]

A benchmark study has been carried out, building a website application to execute the algorithms and measure the execution times with the library Benchmark.js. Measurements of several algorithms of Essentia.js have been done in a few environments finding

[1] `https://mtg.github.io/essentia.js/`

unexpected results. Also a comparison to equivalent algorithms from another JS library for audio analysis has been carried out. Again the results where quite dependent on the browser and device executing the tests. A full analysis of the results is done getting to the conclusion that while Meyda.js is faster than Essentia.js in some environments Essentia.js is quite fast and more consistent across environments.

Finally an application with the goal of having Essentia.js working in a production system was built. The application consists on a system that allows uploading a song and prints the waveform in a simple player. Once the song is loaded a button can call a set of algorithms from Essentia to analyse the song and check if it has any audio problems such as saturation, clipping, cuts and silences among others. During the development process issues with the Essentia.js algorithms were found, having to look for possible solutions. A common solution for a few algorithms was to create a custom extractor, for that reason an automatisation of this process was developed.

List of Figures

Abbreviations

1. **Afero GPL** Affero General Public License
2. **API** Application programming interface
3. **ASP** Audio Signal Processing
4. **BPM** Beats per minute
5. **CDN** Content delivery network
6. **CPU** Central processing unit
7. **CSS** Cascading style sheets
8. **DSP** Digital service providers
9. **FFTW** Fastest Fourier Transform in the West
10. **GCC** GNU compiler collection
11. **DFT** Discrete Fourier transform
12. **HTML** HyperText Markup Language
13. **ISC license** Internet Systems Consortium license
14. **ISMIR** International Society of Music Information Retrieval
15. **JS** JavaScript
16. **JSON** JavaScript Object Notation
17. **LLVM** Low level virtual machine
18. **MFCC** Mel Frequency Cepstral Coefficients

19. **MVP** Minimum viable product
20. **ML** Machine learning
21. **MIR** Music information retrieval
22. **MTG** Music technology group
23. **OS** Operative system
24. **PCM** Pulse-code modulation
25. **QC** Quality control
26. **SNR** Signal-to-noise ratio
27. **SNS** SonoSuite
28. **SMC** Sound and music computing
29. **UI** User interface
30. **UPF** Universitat Pompeu Fabra
31. **WASM** WebAssembly

Listings

Bibliography

[1] Joglar-Ongay, L., Correya, A., Alonso-Jiménez, P., Serra, X. & Bogdanov, D. Essentia in the browser. In Xambó, A., Martín, S. R. & Roma, G. (eds.) *Proceedings of the International Web Audio Conference*, WAC '19, 114–115 (NTNU, Trondheim, Norway, 2019).

[2] Correya, A., Bogdanov, D., Joglar-Ongay, L. & Serra, X. Essentia.js: A JavaScript library for music and audio analysis on the Web. In *Proceedings of the 21st International Society for Music Information Retrieval Conference, ISMIR 2020* (2020). URL `https://github.com/MTG/essentia.js`.

[3] Sound and music computing. `https://en.wikipedia.org/wiki/Sound_and_music_computing` (2020). Accessed: 2020-08-22.

[4] Sound and Music Computing Network. `http://www.smcnetwork.org/roadmap#context`. Accessed: 2020-08-22.

[5] Schedl, M., Schedl, M., Knees, P., Pohle, T. & Widmer, G. Towards an automatically generated music information system via web content mining. *IN: EUROPEAN CONFERENCE ON INFORMATION RETRIEVAL* 585—-590 (2008). URL `https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.211.8930`.

[6] Herrera Boyer, P. *MIRages: an account of music audio extractors, semantic description and context-awareness, in the three ages of MIR.* Ph.D. thesis, Universitat Pompeu Fabra, Barcelona (2018). URL `https://zenodo.org/record/1882316`.

[7] Downie, J. S. Music information retrieval. *Annual Review of Information Science and Technology* **37**, 295–340 (2005). URL `http://doi.wiley.com/10.1002/aris.1440370108`.

[8] Downie, J. S. The scientific evaluation of music information retrieval systems: Foundations and future. *Computer Music Journal* **28**, 12–23 (2004). URL `https://doi.org/10.1162/014892604323112211`. `https://doi.org/10.1162/014892604323112211`.

[9] Schedl, M., Gómez, E. & Urbano, J. Music information retrieval: Recent developments and applications (2014). URL `http://www.nowpublishers.com/articles/foundations-and-trends-in-information-retrieval/INR-042`.

[10] Mcfee, B. *et al.* librosa: Audio and Music Signal Analysis in Python. In *PROC. OF THE 14th PYTHON IN SCIENCE CONF* (2015). URL `https://www.youtube.com/watch?v=MhOdbtPhbLU`.

[11] Bogdanov, D. *et al.* Essentia: An audio analysis library for music information retrieval. In *Proceedings of the 14th International Society for Music Information Retrieval Conference, ISMIR 2013*, 493–498 (2013). URL `http://github.com/MTG/gaia`.

[12] Rawlinson, H., Segal, N. & Fiala, J. Meyda: an audio feature extraction library for the Web Audio API *. In *Web Audio Conference* (Goldsmiths, University of London, Paris, 2015). URL `https://github.com/bmcfee/librosa`.

[13] Matuszewski, B. & Schnell, N. LFO-A Graph-based Modular Approach to the Processing of Data Streams. Tech. Rep. (2017). URL `https://github.com/ircam-jstools/parameters`.

[14] Collins, N. & Knotts, S. A Javascript Musical Machine Listening Library (2019). URL `http://composerprogrammer.com/research/MMLLfinal.pdf`.

[15] Kurihara, K., Itaya, A., Uemura, A., Kitahara, T. & Nagao, K. Picognizer: A javascript library for detecting and recognizing synthesized sounds. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10714 LNCS, 339–359 (Springer Verlag, 2018).

[16] Beverley, J. PRECIPITATE: Distributed Machine Listening for Participatory Weather Resonification. Tech. Rep.

[17] Jillings, N. & Bullock, J. JS-XTRACT: A REALTIME AUDIO FEATURE EXTRACTION LIBRARY FOR THE WEB. Tech. Rep. URL `www.w3.org/TR/workers/.`

[18] Joglar-Ongay, L., Dewey, C. & Wakefield, J. Implementation of Faster than Real Time Audio Analysis for Use with Web Audio API: An FFT Case Study (2016).

[19] W3C Technical Architecture Group, "Web Audio API Design Review". `https://github.com/w3ctag/design-reviews/blob/master/2013/07/WebAudio.md`. Accessed: 2020-08-26.

[20] Choi, H. Audioworklet: the Future of Web Audio. In *ICMC* (2018).

[21] Fonseca, E. *et al.* FREESOUND DATASETS: A PLATFORM FOR THE CREATION OF OPEN AUDIO DATASETS. In *International Society for Music In- formation Retrieval Conference (ISMIR)*, 486–493 (2017).

[22] Porter, A., Bogdanov, D., Kaye, R., Tsukanov, R. & Serra, X. ACOUSTICBRAINZ: A COMMUNITY PLATFORM FOR GATHERING MUSIC INFORMATION OBTAINED FROM AUDIO. In *International Society for Music Information Retrieval Conference (ISMIR 2015)* (2015). URL `https://github.com/metabrainz/`.

[23] Zakai, A. Emscripten: an llvm-to-javascript compiler. In *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2011)*, 301–312 (2011).

[24] Haas, A. *et al.* Bringing the web up to speed with webassembly. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2017)*, 185–200 (2017).

[25] Emscripten Building Projects. `https://emscripten.org/docs/compiling/Building-Projects.html`. Accessed: 2020-08-20.

[26] Emscripten and WebAssembly. `https://fsan.github.io/post/emscripten_and_webassembly/`. Accessed: 2020-08-30.

[27] Bynens, M. & Dalton, J.-D. Bulletproof JavaScript benchmarks. `https://calendar.perfplanet.com/2010/bulletproof-javascript-benchmarks/` (2010). Accessed: 2020-08-20.

[28] Define number of cycles - Benchmark.js. `https://stackoverflow.com/questions/32629779/define-number-of-cycles-benchmark-js` (2016). Accessed: 2020-08-20.

[29] Tilkov, S. & Vinoski, S. Node.js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing* **14**, 80–83 (2010).

[30] Alonso-Jiménez, P., Joglar-Ongay, L., Serra, X. & Bogdanov, D. Automatic detection of audio problems for quality control in digital music distribution. In *Audio Engineering Society Convention 146* (2019). URL `http://www.aes.org/e-lib/browse.cfm?elib=20338`.

Appendix A

Papers Citing Librosa

The following bibiliography are the articles reviewed for the section 2.1.1. These are the top 50 most cited papers citing Librosa during the past 5 years.

Bibliography

[1] Fonseca, E. *et al.* ACOUSTIC SCENE CLASSIFICATION USING A CONVOLUTIONAL NEURAL NETWORK ENSEMBLE AND NEAREST NEIGHBOR FILTERS (2017).

[2] Alsouda, Y., Pllana, S. & Kurti, A. A Machine Learning Driven IoT Solution for Noise Classification in Smart Cities (2018). URL `http://arxiv.org/abs/1809.00238`. `1809.00238`.

[3] Ashis Pati, K., Gururani, S. & Lerch, A. I. Assessment of Student Music Performances Using Deep Neural Networks. *mdpi.com* URL `www.mdpi.com/journal/applsci`.

[4] Böck, S., Korzeniowski, F., Schlüter, J., Krebs, F. & Widmer, G. Madmom: A new python audio and music signal processing library. In *MM 2016 - Proceedings of the 2016 ACM Multimedia Conference*, 1174–1178 (Association for Computing Machinery, Inc, New York, New York, USA, 2016). URL `http://dl.acm.org/citation.cfm?doid=2964284.2973795`. `1605.07008`.

[5] Cakir, E., Adavanne, S., Parascandolo, G., Drossos, K. & Virtanen, T. Convolutional recurrent neural networks for bird audio detection. In *25th European Signal Processing Conference, EUSIPCO 2017*, vol. 2017-January, 1744–1748 (Institute of Electrical and Electronics Engineers Inc., 2017).

[6] Cakir, E., Parascandolo, G., Heittola, T., Huttunen, H. & Virtanen, T. Convolutional Recurrent Neural Networks for Polyphonic Sound Event Detection. *IEEE/ACM Transactions on Audio Speech and Language Processing* **25**, 1291–1303 (2017). 1702.06286.

[7] Choi, K., Fazekas, G. & Sandler, M. Towards Playlist Generation Algorithms Using RNNs Trained on Within-Track Transitions. *CEUR Workshop Proceedings* **1618** (2016). URL http://arxiv.org/abs/1606.02096. 1606.02096.

[8] Choi, K., Fazekas, G. & Sandler, M. Explaining Deep Convolutional Neural Networks on Music Classification (2016). URL http://arxiv.org/abs/1607.02444. 1607.02444.

[9] Choi, K., Fazekas, G., Sandler, M. & Cho, K. Convolutional recurrent neural networks for music classification. In *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2392–2396 (Institute of Electrical and Electronics Engineers Inc., 2017). 1609.04243.

[10] Chou, S.-Y., Jang, J.-S. R. & Yang, Y.-H. FRAMECNN: A WEAKLY-SUPERVISED LEARNING FRAMEWORK FOR FRAME-WISE ACOUSTIC EVENT DETECTION AND CLASSIFICATION. Tech. Rep. (2017). URL https://github.com/librosa.

[11] Chou, S.-Y., Jang, J.-S. R. & Yang, Y.-H. Learning to Recognize Transient Sound Events Using Attentional Supervision. Tech. Rep. (2017).

[12] Colonel, J., Curro, C. & Keene, S. Improving Neural Net Autoencoders for Music Synthesis. Tech. Rep. URL http://github.com/JTColonel/ann{_}synth.

[13] Dorfer, M. *et al.* ACOUSTIC SCENE CLASSIFICATION WITH FULLY CONVOLUTIONAL NEURAL NETWORKS AND I-VECTORS Technical Report. Tech. Rep. URL https://www.kaggle.com/c/dcase2018-task1a-.

[14] Dorfer, M. & Widmer, G. TRAINING GENERAL-PURPOSE AUDIO TAGGING NETWORKS WITH NOISY LABELS AND ITERATIVE SELF-VERIFICATION. In *Detection and Classification of Acoustic Scenes and Events 2018* (Surrey, 2018). URL http://dcase.community/documents/challenge2018/technical{_}reports/DCASE2018{_}Dorfer{_}999.pdf.

[15] Fonseca, E., Plakal, M., Font, F., Ellis, D. P. W. & Serra, X. Audio tagging with noisy labels and minimal supervision 69–73 (2019). URL http://arxiv.org/abs/1906.02975. 1906.02975.

[16] Fricke, K. R. & Herzberg, P. Y. Personality and self-reported preference for music genres and attributes in a German-speaking sample. *Journal of Research in Personality* **68**, 114–123 (2017).

[17] Hung, Y.-N. & Yang, Y.-H. Frame-level Instrument Recognition by Timbre and Pitch. *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018* 135–142 (2018). URL http://arxiv.org/abs/1806.09587. 1806.09587.

[18] Juvela, L. *et al.* Speech Waveform Synthesis from MFCC Sequences with Generative Adversarial Networks. In *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 2018-April, 5679–5683 (Institute of Electrical and Electronics Engineers Inc., 2018). 1804.00920.

[19] Kim, D. H., Lee, M. K., Choi, D. Y. & Song, B. C. Multi-modal emotion recognition using semi-supervised learning and multiple neural networks in the wild. In *ICMI 2017 - Proceedings of the 19th ACM International Conference on Multimodal Interaction*, vol. 2017-January, 529–535 (Association for Computing Machinery, Inc, New York, New York, USA, 2017). URL http://dl.acm.org/citation.cfm?doid=3136755.3143005.

[20] Kim, J., Urbano, J., Liem, C., And, A. H. N. C. & undefined 2020. One deep music representation to rule them all? A comparative analysis of different representation learning strategies. *Springer* URL https://link.springer.com/article/10.1007/s00521-019-04076-1.

[21] Kumar, A. & Raj, B. Deep CNN Framework for Audio Event Recognition using Weakly Labeled Web Data (2017). URL http://arxiv.org/abs/1707.02530. 1707.02530.

[22] Kunze, J. *et al.* Transfer Learning for Speech Recognition on a Budget 168–177 (2017). URL http://arxiv.org/abs/1706.00290. 1706.00290.

[23] Latorre, J. *et al.* EFFECT OF DATA REDUCTION ON SEQUENCE-TO-SEQUENCE NEURAL TTS. Tech. Rep. URL https://ieeexplore.ieee.org/abstract/document/8682168/. 1811.06315v2.

[24] Lerch, A., Gururani, S. & Summers, C. INSTRUMENT ACTIVITY DETECTION IN POLYPHONIC MUSIC USING DEEP NEURAL NETWORKS Text Book on Audio Content Analysis View project Special Issue "Machine Learning Applied to Music/Audio Signal Processing" View project INSTRUMENT ACTIVITY DETECTION IN POLYPHONIC MUSIC USING DEEP NEURAL NETWORKS. Tech. Rep. (2019). URL https://www.researchgate.net/publication/332621784.

[25] Lorenzo-Trueba, J. *et al.* Towards achieving robust universal neural vocoding. In *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, vol. 2019-September, 181–185 (International Speech Communication Association, 2019). URL https://arxiv.org/abs/1811.06292v2. 1811.06292.

[26] Lostanlen, V., Salamon, J., Farnsworth, A., Kelling, S. & Bello, J. P. BIRDVOX-FULL-NIGHT: A DATASET AND BENCHMARK FOR AVIAN FLIGHT CALL DETECTION. Tech. Rep. URL https://wp.nyu.edu/birdvox/birdvox-full-night.

[27] Lukic, Y., Vogt, C., Durr, O. & Stadelmann, T. Speaker identification and clustering using convolutional neural networks. In *IEEE International Workshop on Machine Learning for Signal Processing, MLSP*, vol. 2016-November (IEEE Computer Society, 2016).

[28] Mcfee, B. *et al.* Open-Source Practices for Music Signal Processing Research Recommendations for transparent, sustainable, and reproducible audio research. *IEEE SIgnal ProcESSIng MagazInE* (2019). URL `https://redis.io`.

[29] McFee, B., Nieto, O., Farbood, M. M. & Bello, J. P. Evaluating hierarchical structure in music annotations. *Frontiers in Psychology* **8** (2017).

[30] Mishra, S., Sturm, B. L. & Dixon, S. LOCAL INTERPRETABLE MODEL-AGNOSTIC EXPLANATIONS FOR MUSIC CONTENT ANALYSIS. Tech. Rep. URL `https://code.soundsoftware.ac.uk/projects/SoundLIME`.

[31] Nieto, O. & Bello, J. P. SYSTEMATIC EXPLORATION OF COMPUTATIONAL MUSIC STRUCTURE RESEARCH. Tech. Rep. URL `https://github.com/urinieto/msaf`.

[32] Oramas, S., Barbieri, F., Nieto, O. & Serra, X. Multimodal Deep Learning for Music Genre Classification (2018). URL `https://doi.org/10.5334/tismir.10`.

[33] Oramas, S., Nieto, O., Barbieri, F. & Serra, X. Multi-label Music Genre Classification from Audio, Text, and Images Using Deep Features. *Proceedings of the 18th International Society for Music Information Retrieval Conference, ISMIR 2017* 23–30 (2017). URL `http://arxiv.org/abs/1707.04916`. `1707.04916`.

[34] Oramas, S., Nieto, O., Sordo, M. & Serra, X. A deep multimodal approach for cold-start music recommendation. In *ACM International Conference Proceeding Series*, vol. Part F130153, 32–37 (Association for Computing Machinery, New York, New York, USA, 2017). URL `http://dl.acm.org/citation.cfm?doid=3125486.3125492`. `1706.09739`.

[35] Panwar, S., Das, A., of ..., M. R. . t. S. & undefined 2017. A deep learning approach for mapping music genres. *ieeexplore.ieee.org* URL https://ieeexplore.ieee.org/abstract/document/7994970/.

[36] Pui Tang, C. *et al.* Music Genre classification using a hierarchical Long Short Term Memory (LSTM) model. Tech. Rep. (2018). URL https://doi.org/10.475/123{_}4.

[37] Raffel, C. Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching (2016).

[38] Raffel, C. & Ellis, D. P. W. OPTIMIZING DTW-BASED AUDIO-TO-MIDI ALIGNMENT AND MATCHING. Tech. Rep. URL https://ieeexplore.ieee.org/abstract/document/7471641/.

[39] Raffel, C. & Ellis, D. P. W. PRUNING SUBSEQUENCE SEARCH WITH ATTENTION-BASED EMBEDDING. Tech. Rep. URL https://ieeexplore.ieee.org/abstract/document/7471736/.

[40] Schindler, A., Lidy, T. & Rauber, A. Comparing Shallow versus Deep Neural Network Architectures for Automatic Music Genre Classification. Tech. Rep. URL http://ceur-ws.org.

[41] Su, Y., Zhang, K., Wang, J. & Madani, K. Environment Sound Classification Using a Two-Stream CNN Based on Decision-Level Fusion. *mdpi.com* URL www.mdpi.com/journal/sensors.

[42] Tang, T., Jia, J. & Mao, H. Dance with melody: An LSTM-autoencoder Approach to Music-oriented Dance Synthesis. In *MM 2018 - Proceedings of the 2018 ACM Multimedia Conference*, 1598–1606 (Association for Computing Machinery, Inc, New York, New York, USA, 2018). URL http://dl.acm.org/citation.cfm?doid=3240508.3240526.

[43] Thickstun, J., Harchaoui, Z. & Kakade, S. Learning Features of Music from Scratch. *5th International Conference on Learning Representations, ICLR 2017*

- *Conference Track Proceedings* (2016). URL http://arxiv.org/abs/1611.09827. 1611.09827.

[44] Valenti, M., Diment, A., Parascandolo, G., Squartini, S. & Virtanen, T. DCASE 2016 ACOUSTIC SCENE CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS. Tech. Rep. (2016).

[45] Valenti, M., Squartini, S., Diment, A., Parascandolo, G. & Virtanen, T. A convolutional neural network approach for acoustic scene classification. In *Proceedings of the International Joint Conference on Neural Networks*, vol. 2017-May, 1547–1554 (Institute of Electrical and Electronics Engineers Inc., 2017).

[46] Vesperini, F., Gabrielli, L., Principi, E., Squartini, S. & Member, S. Polyphonic Sound Event Detection by using Capsule Neural Networks. Tech. Rep. (2018). URL https://ieeexplore.ieee.org/abstract/document/8654643/. 1810.06325v4.

[47] Wyse, L. Audio Spectrogram Representations for Processing with Convolutional Neural Networks (2017). URL http://arxiv.org/abs/1706.09559. 1706.09559.

[48] Zeinali, H., Burget, L. & Cernocky, J. Convolutional Neural Networks and x-vector Embedding for DCASE2018 Acoustic Scene Classification Challenge (2018). URL http://arxiv.org/abs/1810.04273. 1810.04273.

[49] Zhao, J., Mao, X., Processing, L. C. I. S. & undefined 2018. Learning deep features to recognise speech emotion using merged deep CNN. *IET* URL https://digital-library.theiet.org/content/journals/10.1049/iet-spr.2017.0320.

[50] Zhao, J., Mao, X. & Chen, L. Speech emotion recognition using deep 1D & 2D CNN LSTM networks. *Biomedical Signal Processing and Control* **47**, 312–323 (2019).

Appendix B

Papers Citing Essentia

The following bibiliography are the articles reviewed for the section 2.1.2. These are the top 50 most cited papers citing Essentia during the past 5 years.

Bibliography

[1] Srinivasamurthy, A. & Serra Casals, X. A Data-driven Bayesian Approach to Automatic Rhythm Analysis of Indian Art Music. Tech. Rep. (2016). URL `http://www.upf.edu`.

[2] Chourdakis, E. T. A Machine-Learning Approach to Application of Intelligent Artificial Reverberation. *Journal of the Audio Engineering Society* **65** (2017). URL `https://doi.org/10.17743/jaes.2016.0069`.

[3] Picas, O. R. *et al.* A real-time system for measuring sound goodness in instrumental sounds. Tech. Rep. URL `www.aes.org`.

[4] Rao, A., Huynh, E., reviews in . . . , T. R. I. & undefined 2018. Acoustic Methods for Pulmonary Diagnosis. *ieeexplore.ieee.org* URL `https://ieeexplore.ieee.org/abstract/document/8514011/`.

[5] Fonseca, E. *et al.* Acoustic scene classification by ensembling gradient boosting machine and convolutional neural networks. Tech. Rep. (2017). URL `https://github.com/Microsoft/LightGBM`.

[6] Porter, A., Bogdanov, D., Kaye, R., Tsukanov, R. & Serra, X. ACOUSTICBRAINZ: A COMMUNITY PLATFORM FOR GATHERING MUSIC INFORMATION OBTAINED FROM AUDIO. In *International Society for Music Information Retrieval Conference (ISMIR 2015)* (2015). URL `https://github.com/metabrainz/`.

[7] Abdul, A., Chen, J., Liao, H.-Y. & Chang, S.-H. An Emotion-Aware Personalized Music Recommendation System Using a Convolutional Neural Networks Approach. *mdpi.com* URL `www.mdpi.com/journal/applsci`.

[8] Gulati, S., Serrà, J. & Serra, X. AN EVALUATION OF METHODOLOGIES FOR MELODIC SIMILARITY IN AUDIO RECORDINGS OF INDIAN ART MUSIC. Tech. Rep. URL `http://nema.lis.illinois.edu/nema{_}out/`.

[9] Liem, C. C. S. & Hanjalic, A. COMPARATIVE ANALYSIS OF ORCHESTRAL PERFORMANCE RECORDINGS: AN IMAGE-BASED APPROACH. Tech. Rep. URL `http://phenicx.upf.edu`.

[10] Gulati, S. & Serra Casals, X. Computational Approaches for Melodic Description in Indian Art Music Corpora. Tech. Rep. (2017). URL `http://www.upf.edu/dtic`.

[11] Ono, J., Sikansi, F., . . . , D. C. . t. S. & undefined 2015. Concentric radviz: Visual exploration of multi-task classification. *ieeexplore.ieee.org* URL `https://ieeexplore.ieee.org/abstract/document/7314560/`.

[12] Kroher, N., Díaz-Báñez, J. M., Mora, J. & Gómez, E. Corpus COFLA: A research corpus for the computational study of flamenco music. *Journal on Computing and Cultural Heritage* **9** (2016). `1510.04029`.

[13] Augello, A. *et al.* Creation and cognition for humanoid live dancing. *Elsevier* URL `https://www.sciencedirect.com/science/article/pii/S092188901630584X`.

[14] Donahue, C., Lipton, Z. C. & Mcauley, J. Dance Dance Convolution. Tech. Rep. (2017). URL `https://github.com/chrisdonahue/ddc`.

[15] Ganguli, K. K., Gulati, S., Serra, X. & Rao, P. DATA-DRIVEN EXPLORATION OF MELODIC STRUCTURES IN HINDUSTANI MUSIC. Tech. Rep. URL http://www.music-ir.org/mirex/wiki/2011.

[16] Salamon, J. & Bello, J. P. Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification. Tech. Rep. URL https://github.com/justinsalamon/UrbanSound8K-JAMS. 1608.04363v2.

[17] Vaiciukynas, E., Verikas, A., Gelzinis, A., One, M. B. P. & undefined 2017. Detecting Parkinson's disease from sustained phonation and speech signals. *ncbi.nlm.nih.gov* URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5628839/.

[18] McNamara, Q., Vega, A. D. L., ACM, T. Y. P. o. t. r. & undefined 2017. Developing a comprehensive framework for multimodal feature extraction. *dl.acm.org* URL https://dl.acm.org/doi/abs/10.1145/3097983.3098075.

[19] Tralie, C. J. Early MFCC and HPCP fusion for robust cover song identification. In *Proceedings of the 18th International Society for Music Information Retrieval Conference, ISMIR 2017*, 294–301 (International Society for Music Information Retrieval, 2017). 1707.04680.

[20] Chen, Y.-P., Su, L. & Yang, Y.-H. ELECTRIC GUITAR PLAYING TECHNIQUE DETECTION IN REAL-WORLD RECORDINGS BASED ON F0 SEQUENCE PATTERN RECOGNITION. Tech. Rep. URL http://play.riffstation.com/.

[21] Maestre, E., Papiotis, P., Marchini, M., . . . , Q. L. I. & undefined 2017. Enriched multimodal representations of music performances: Online access and visualization. *ieeexplore.ieee.org* URL https://ieeexplore.ieee.org/abstract/document/7849104/.

[22] Bello, J. P. & Salamon, J. Feature Learning with Deep Scattering for Urban Sound Analysis. *ieeexplore.ieee.org* URL https://www.researchgate.net/publication/278019931.

[23] Fonseca, E. *et al.* FREESOUND DATASETS: A PLATFORM FOR THE CREATION OF OPEN AUDIO DATASETS. In *International Society for Music Information Retrieval Conference (ISMIR)*, 486–493 (2017).

[24] Salamon, J., Bello, J. P., Farnsworth, A. & Kelling, S. FUSING SHALLOW AND DEEP LEARNING FOR BIOACOUSTIC BIRD SPECIES CLASSIFICATION. Tech. Rep. URL https://ieeexplore.ieee.org/abstract/document/7952134/.

[25] Alemi, O., Françoise, J., Networks, P. P. & undefined 2017. GrooveNet: Real-time music-driven dance movement generation using artificial neural networks. *pdfs.semanticscholar.org* URL https://pdfs.semanticscholar.org/3790/e9d649ecc61e8d3ea0b16cfe0b96379cc34d.pdf.

[26] Angulo, I., Giraldo, S. & Ramirez, R. HEXAPHONIC GUITAR TRANSCRIPTION AND VISUALISATION. Tech. Rep. (2016). URL https://repositori.upf.edu/handle/10230/43212.

[27] Hanke, M., Dinga, R., Häusler, C., . . . , J. G. & undefined 2015. High-resolution 7-Tesla fMRI data on the perception of musical genres – an extension to the studyforrest dataset. *f1000research.com* URL https://f1000research.com/articles/4-174.

[28] Rao, A., Chu, S., Batlivala, N., journal of . . . , S. Z. I. & undefined 2018. Improved detection of lung fluid with standardized acoustic stimulation of the chest. *ieeexplore.ieee.org* URL https://ieeexplore.ieee.org/abstract/document/8443414/.

[29] Turchet, L., Fischione, C., Essl, G., Access, D. K. I. & undefined 2018. Internet of musical things: Vision and challenges. *ieeexplore.ieee.org* URL https://ieeexplore.ieee.org/abstract/document/8476543/.

[30] Choi, K., Joo, D. & Kim, J. Kapre: On-GPU Audio Preprocessing Layers for a Quick Implementation of Deep Neural Network Models with Keras (2017). URL http://arxiv.org/abs/1706.05781. 1706.05781.

[31] Faraldo, Á., Gómez, E., Jordà, S., On, P. H. E. C. & undefined 2016. Key estimation in electronic dance music. *Springer* URL https://link.springer.com/chapter/10.1007/978-3-319-30671-1{_}25.

[32] Xu, Y., Kong, Q., Wang, W. & Plumbley, M. D. LARGE-SCALE WEAKLY SUPERVISED AUDIO CLASSIFICATION USING GATED CONVOLUTIONAL NEURAL NETWORK. Tech. Rep. URL https://github.com/yongxuUSTC/dcase2017{_}task4{_}. 1710.00343v1.

[33] McFee, B., Raffel, C., Liang, D., of the ..., D. E. P. & undefined 2015. librosa: Audio and music signal analysis in python. *academia.edu* URL http://www.academia.edu/download/40296500/librosa.pdf.

[34] Böck, S., Korzeniowski, F., Schlüter, J., Krebs, F. & Widmer, G. Madmom: A new python audio and music signal processing library. In *MM 2016 - Proceedings of the 2016 ACM Multimedia Conference*, 1174–1178 (Association for Computing Machinery, Inc, 2016).

[35] Vicente, J., Gil, A., ..., A. d. L. R. .... C., Artificial & undefined 2018. Moodsically. Personal music management tool with automatic classification of emotions. *Springer* URL https://link.springer.com/chapter/10.1007/978-3-319-94649-8{_}14.

[36] Grekow, J. Music Emotion Maps in Arousal-Valence Space. *Springer* 697–706 (2016). URL https://hal.inria.fr/hal-01637515.

[37] Cheng, Z., (TOIS), J. S. A. T. o. I. S. & undefined 2016. On effective location-aware music recommendation. *dl.acm.org* URL https://dl.acm.org/doi/abs/10.1145/2846092.

[38] Fricke, K., Yorck Herzberg, P., Fricke, K. R. & Herzberg, P. Y. Personality and Self-reported Preference for Music Genres and Attributes in a German-speaking Sample Persönlichkeitspsychologie View project QoLISSY View project Personality and self-reported preference for music genres and attributes in a

German-speaking. *Article in Journal of Research in Personality* (2017). URL http://dx.doi.org/10.1016/j.jrp.2017.01.001.

[39] Schnell, N., Schwarz, D., Larralde, J., Borghesi, R. & Pipo, R. B. PiPo, A Plugin Interface for Afferent Data Stream Processing Modules A Plugin Interface for Afferent Data Stream Processing Modules PIPO, A PLUGIN INTERFACE FOR AFFERENT DATA STREAM PROCESSING MODULES. Tech. Rep. (2017). URL http://www.ladpsa.org.

[40] Bayle, Y., Hanna, P. & Robine, M. SATIN: A persistent musical database for music information retrieval. In *ACM International Conference Proceeding Series*, vol. Part F1301 (Association for Computing Machinery, 2017).

[41] Salamon, J., Macconnell, D., Cartwright, M., Li, P. & Bello, J. P. SCAPER: A LIBRARY FOR SOUNDSCAPE SYNTHESIS AND AUGMENTATION. Tech. Rep. URL http://urbansed.weebly.com/.

[42] Oramas, S., Ostuni, V. C., Di Noia, T., Serra, X. & Di Sciascio, E. Sound and music recommendation with knowledge graphs. *ACM Transactions on Intelligent Systems and Technology* **8** (2016).

[43] Nieto, O. & Bello, J. P. SYSTEMATIC EXPLORATION OF COMPUTATIONAL MUSIC STRUCTURE RESEARCH. Tech. Rep. URL https://github.com/urinieto/msaf.

[44] Serra, X. The Computational Study of a Musical Culture through Its Digital Traces. Tech. Rep. (2017). URL https://muse.jhu.edu/article/675804.

[45] Bogdanov, D., Porter, A., Urbano, J. & Schreiber, H. The MediaEval 2017 AcousticBrainz Genre Task: Content-based Music Genre Recognition from Multiple Sources. Tech. Rep. (2017). URL https://multimediaeval.github.io/2017-AcousticBrainz-Genre-Task/data/.

[46] Pellegrini, T. & Barrière, V. Time-continuous Estimation of Emotion in Music with Recurrent Neural Networks Open Archive TOULOUSE Archive Ouverte

(OATAO) Time-continuous estimation of emotion in music with recurrent neural networks. Tech. Rep. (2015). URL http://www.multimediaeval.org/mediaeval2015/.

[47] Gulati, S., Serrà, J., Ganguli, K. K., Entürkententürk, S. S. & Serra, X. TIME-DELAYED MELODY SURFACES FOR R AGA RECOGNITION. Tech. Rep. URL https://github.com/MTG/essentia.

[48] Salamon, J. *et al.* Towards the automatic classification of avian flight calls for bioacoustic monitoring. *ncbi.nlm.nih.gov* URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5120805/.

[49] Knees, P. *et al.* TWO DATA SETS FOR TEMPO ESTIMATION AND KEY DETECTION IN ELECTRONIC DANCE MUSIC ANNOTATED FROM USER CORRECTIONS. Tech. Rep. URL http://blog.dubspot.com/endo-harmonic-mixing-key-detection-.

[50] Salamon, J. & Bello, J. P. *UNSUPERVISED FEATURE LEARNING FOR URBAN SOUND CLASSIFICATION*. URL http://www.freesound.org.

Appendix C

Papers Citing Meyda

The following bibiliography are the articles reviewed for the section 2.1.3. These are the 18 papers citing Meyda found in Google Scholar.

Bibliography

[1] Bernstein, A. & Taylor, B. Gendy.js: A Web Audio Module for Dynamic Stochastic Synthesis. Tech. Rep. (2017). URL http://qmro.qmul.ac.uk/xmlui/handle/123456789/26163.

[2] Beverley, J. PRECIPITATE: Distributed Machine Listening for Participatory Weather Resonification. Tech. Rep.

[3] Chaki, J. Pattern analysis based acoustic signal processing: a survey of the state-of-art. *International Journal of Speech Technology* 1–43 (2020). URL https://doi.org/10.1007/s10772-020-09681-3.

[4] Collins, N. & Knotts, S. A Javascript Musical Machine Listening Library (2019). URL http://composerprogrammer.com/research/MMLLfinal.pdf.

[5] Jillings, N. & Bullock, J. JS-XTRACT: A REALTIME AUDIO FEATURE EXTRACTION LIBRARY FOR THE WEB. Tech. Rep. URL www.w3.org/TR/workers/.

[6] Karadoğan, C. & Köktürk, M. *AUDIO BASED CLASSIFICATION OVER MUSICAL PRODUCTION PERIOD*. Ph.D. thesis, İstanbul Technical University, Istanbul (2019). URL https://polen.itu.edu.tr/handle/11527/18262.

[7] Kurihara, K., Itaya, A., Uemura, A., Kitahara, T. & Nagao, K. Picognizer: A javascript library for detecting and recognizing synthesized sounds. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10714 LNCS, 339–359 (Springer Verlag, 2018).

[8] Lakka, E., Malamos, A. G., Pavlakis, K. G. & Ware, J. A. Spatial Sound Rendering – A Survey - Dialnet (2018). URL https://dialnet.unirioja.es/servlet/articulo?codigo=6901757.

[9] Lee, J. Honkling: In-Browser Personalization for Ubiquitous Keyword Spotting. Tech. Rep. (2019). URL https://uwspace.uwaterloo.ca/handle/10012/15343.

[10] Lee, J., Tang, R. & Lin, J. JavaScript Convolutional Neural Networks for Keyword Spotting in the Browser: An Experimental Analysis (2018). URL http://arxiv.org/abs/1810.12859. 1810.12859.

[11] Matuszewski, B. & Schnell, N. LFO-A Graph-based Modular Approach to the Processing of Data Streams. Tech. Rep. (2017). URL https://github.com/ircam-jstools/parameters.

[12] Moffat, D. J. Evaluation of Synthesised Sound Effects. Tech. Rep. (2019).

[13] Rawlinson, H., Segal, N. & Fiala, J. Meyda: an audio feature extraction library for the Web Audio API *. In *Web Audio Conference* (Goldsmiths, University of London, Paris, 2015). URL https://github.com/bmcfee/librosa.

[14] Roma, G., Xambó, A. & Freeman, J. Loop-aware Audio Recording for the Web. Tech. Rep. (2017). URL http://freesound.org.

[15] Thompson, L., Cannam, C. & Sandler, M. Piper: Audio Feature Extraction in Browser and Mobile Applications. Tech. Rep. (2017). URL http://github.com/piper-audio/.

[16] Won Lee, S., Taylor, B. & Essl, G. Interactive Music on the Web. In Filimowicz, M. (ed.) *Foundations in Sound Design for Interactive Media: A Multidisciplinary*, 200–228 (Routledge, 2019). URL https://www.routledge.com/Foundations-in-Sound-Design-for-Interactive-Media-A-Multidisciplinary/Filimowicz/p/book/9781138093942.

[17] Xambó, A., Green, O., Tremblay, P. A. & Roma, G. A Javascript Library for Flexible Visualization of Audio Descriptors. Tech. Rep. (2018). URL https://www.vamp-plugins.org.

[18] Zbyszyński, M. *et al.* Write once run anywhere revisited: machine learning and audio tools in the browser with C++ and emscripten. Tech. Rep. (2017). URL http://www.wekinator.org/.