Essentia TensorFlow Models for Audio and Music Processing on the Web

Albin Correya *
Music Technology Group
Universitat Pompeu Fabra
Barcelona, Spain
albin.a.correya@gmail.com

Pablo Alonso-Jiménez
Music Technology Group
Universitat Pompeu Fabra
Barcelona, Spain
pablo.alonso@upf.edu

Jorge Marcos-Fernández
Music Technology Group
Universitat Pompeu Fabra
Barcelona, Spain
jorge.marcos@upf.edu

Xavier Serra
Music Technology Group
Universitat Pompeu Fabra
Barcelona, Spain
xavier.serra@upf.edu

Dmitry Bogdanov
Music Technology Group
Universitat Pompeu Fabra
Barcelona, Spain
dmitry.bogdanov@upf.edu

ABSTRACT

Recent advances in web-based machine learning (ML) tools empower a wide range of application developers in both industrial and creative contexts. The availability of pre-trained ML models and JavaScript (JS) APIs in frameworks like *TensorFlow.js* enabled developers to use AI technologies without demanding domain expertise. Nevertheless, there is a lack of pre-trained models in web audio compared to other domains, such as text and image analysis. Motivated by this, we present a collection of open pre-trained *TensorFlow.js* models for music-related tasks on the Web. Our models currently allow for different types of music classification (e.g., genres, moods, danceability, voice or instrumentation), tempo estimation, and music feature embeddings. To facilitate their use, we provide a dedicated JS add-on module *essentia.js-model* within the *Essentia.js* library for audio and music analysis. It has a simple API, enabling end-to-end analysis from audio input to prediction results on web browsers and Node.js. Along with the Web Audio API and web workers, it can be also used to build real-time applications. We provide usage examples, discuss possible use-cases, and report benchmarking results.

1. INTRODUCTION

Nowadays, the Web is one of the most ubiquitous and thriving computing platforms with an ever-growing number of possible applications following the updates in Web standards. Web Audio is an intrinsic part of the next generation of applications for multimedia content creators, designers, and researchers, and music tutors, artists, and consumers. With the adoption of HTML5, the latest W3C Web Audio API specifications and the development of WebAssembly (WASM), modern web browsers became capable of more advanced audio processing, synthesis, and analysis. This has paved way for the development of new extensive JS software libraries for audio analysis and music information retrieval (MIR). Lately, *Essentia.js* [8] has been released by porting and extending one of the most common MIR libraries used in native applications to the Web [6]. There are also few other existing smaller-scale libraries, offering music audio analysis [7, 9, 10, 12], but *Essentia.js* provides the largest variety of music descriptors at this moment and allows high flexibility for custom analysis chains.

In addition, ML methods, especially deep learning for audio and music processing, allowed for innovative approaches that greatly complement the traditional signal processing methods but are not yet well-represented in the web compared to other domains such as text and image processing. Web ML frameworks with multiple computing back-ends like *TensorFlow.js*[1] and *ONNX.js*[2] have enabled the use of pre-trained ML models as black-box software systems in typical web software development workflows, which has helped application developers to leverage this new set of AI technologies. The TensorFlow ecosystem provides an easy-to-use tool to convert pre-trained ML models trained in Python or C++ into web targets.

Currently, TensorFlow Hub[3] provides many pre-trained models ready for deployment in JS applications, yet lacking most of the common audio problems. This is not surprising considering that many ML audio models require an intermediate representation of audio signal derived from spectral analysis as an input for inference (except for few models that operate on raw audio). And this input has to be the same as the one used when training the model (e.g., in Python) to produce the expected results. Essentia recently released a collection of pre-trained TensorFlow models for audio and music related tasks [2, 3]. These models are optimised for production and are trained with the audio representations computed using Essentia itself, which makes them an potential choice to be ported to *TensorFlow.js* models for the

*Currently at Moodagent A/S, Copenhagen, Denmark

Web Audio Conference WAC-2021, July 5–7, 2021, Barcelona, Spain.

[1] https://www.tensorflow.org/js
[2] https://github.com/microsoft/onnxjs
[3] https://tfhub.dev

web platform. However, using pre-trained models via ML libraries like *TensorFlow.js* directly can be cumbersome for a wide range of application developers, creative coders and artists since it demands some ML domain expertise. In order to avoid this overhead and facilitate inclusivity and usability of these tools, a few new JS abstraction libraries and tools were created with a user-friendly APIs [1, 4, 14]. Some of these tools were specifically designed with creative applications in mind.

In this paper, we present a collection of *TensorFlow.js* audio ML models for music processing along with a high-level add-on JS module *essentia.js-model* integrated into the *Essentia.js* library. This module allows developers to do end-to-end processing from audio input to the models' prediction results with a simple JS API. The rest of the paper is organized as follows. Section 2 briefly outlines the various components of *Essentia.js*. Section 3 presents the pre-trained models available in Essentia, which we have ported for *TensorFlow.js*. In Section 4 we describe a new add-on module for *Essenita.js* that we developed to facilitate using both libraries together with our models and provide example applications and benchmarking results. Finally, we conclude and discuss future work in Section 5.

2. ESSENTIA.JS

Essentia.js[4] is a JS library powered by a WASM back-end of the popular audio and music analysis library Essentia [6]. It provides an extensive collection of over 200 algorithms for typical sound and music analysis tasks, including spectral, tonal, and rhythmic characterization. The library is suitable for onset detection, beat tracking and tempo estimation, melody extraction, key and chord estimation, sound and music classification, cover song similarity, loudness metering, and audio problems detection, among other common tasks.

The core of the library is powered by the *Essentia WASM backend*, based on the Essentia C++ library, which is coupled with custom JS bindings and high-level JS API. All the algorithm methods are configurable similarly to Essentia's C++/Python API itself. The build tools provided with the library allow creating lightweight builds of the library, with only a few specific algorithms required for a particular application.

In addition to the core library, *Essentia.js* has a few add-on modules to facilitate common MIR tasks. In particular, *essentia.js-extractor* contains predefined feature extractors for common MIR tasks, computing BPM, beat positions, pitch, predominant melody, key, chords, chroma features, MFCC, etc. Also, *essentia.js-plot* provides helper functions for visualization of MIR features, allowing both real-time and offline plotting in a given HTML element.

See our paper introducing *Essentia.js* [8] and the online documentation for more details.

3. TENSORFLOW MODELS

Essentia models is a repository of pre-trained machine learning models publicly available under the Creative Commons BY-NC-ND 4.0 license[5] and intended to use within Essentia.[6] Many of them are classifiers for specific music audio annotation tasks. Other models are trained on large generic MIR datasets and can be used to extract feature embeddings. In [2,3], the authors give further implementation details and perform an extensive evaluation of the provided models.

For this work we focused on the following models for the tasks of auto-tagging [13], tempo estimation [15], and classification based on transfer learning [2,3]:

- Two auto-tagging models trained on Million Song Dataset (MSD) [5] and MagnaTagATune (MTT) [11] with activations for the top-50 tags in each taxonomy. The tags contain information related to the genre, instrumentation, mood or era of the music (e.g., rock, pop, alternative, indie, electronic, female, vocalists, dance, 00s, alternative rock, and jazz).
- The tempo models estimate the tempo of music ranging from 30 to 286 BPM. We included a variety of CNN architectures with different model sizes.
- Various classifiers for genre, mood, and other semantic categories trained using transfer learning. We used the above-mentioned auto-tagging models to extract embeddings that were then used as input features to train the classifiers. This technique allows leveraging the knowledge acquired on our larger datasets from auto-tagging for more specific classification tasks. Table 1 contains the classes available on each tasks.

	Task	Classes
genre	dortmund	alternative, blues, electronic, folk-country, funksoulrnb, jazz, pop, raphiphop, rock
	gtzan	blues, classic, country, disco, hip hop, jazz, metal, pop, reggae, rock
	rosamerica	classic, dance, hip hop, jazz, pop, rhythm and blues, rock, speech
mood	acoustic	acoustic, non acoustic
	aggressive	aggressive, non aggressive
	electronic	electronic, non electronic
	happy	happy, non happy
	party	party, non party
	relaxed	relaxed, non relaxed
	sad	sad, non sad
misc.	danceability	danceable, non danceable
	voice/instrum.	voice, instrumental
	gender	male, female
	tonal/atonal	atonal, tonal
	urbansound8k	air conditioner, car horn, children playing, dog bark, drilling, engine idling, gun shot, jackhammer, siren, street music
	fs-loop-ds	bass, chords, fx, melody, percussion

Table 1: Tasks used to train the transfer learning classifiers.

Table 2 compares the different architectures in terms of receptive field (seconds of audio required to perform a prediction), number of parameters, size in megabytes and purpose. Note that we only account for the feature extractor part of the transfer learning model, as the fully-connected classifiers are negligible in size. We considered a wide variety of model capabilities in terms of parameters so it is not expected that all the models are suitable for web deployment on computationally-weak devices.

[4]https://essentia.upf.edu/essentiajs
[5]https://creativecommons.org/licenses/by-nc-nd/4.0
[6]https://essentia.upf.edu/machine_learning.html

Model	RF (s)	Params.	Size (MB)	Purpose
MusiCNN	3	787K	3.1	AT/TL
VGG	3	605K	2.4	AT/TL
VGGish	1	62M	276	TL
TempoCNN	12	[27K-1.2M]	[0.1-4.7]	Tempo

Table 2: The Essentia models. RF: Receptive field, AT: Auto-tagging, TL: Transfer learning.

Figure 1 shows the activations produced by all the auto-tagging and classification taxonomies on the song Bohemian Rhapsody by the rock band Queen. It can be seen how some of the classes can be useful to describe the structure of the song. Note that the transfer learning classifiers need to activate an output even when none of the choices seem appropriate. Hence, we can find some incongruences, such as the label *ambient* from the *mood electronic* classifier. Even if it does not seem an adequate label, the classifier does not contain better choices.

3.0.1 *Essentia models in TensorFlow.js*

The ability to deploy client-side deep learning models is a attractive feature supported by a growing amount of frameworks. By the time we started this work the main options to go were *TensorFlow.js* and *ONNX.js*. We identified the following advantages of using *TensorFlow.js* in our case:

- It is the most actively maintained project with extensive documentation and example projects.
- It is part of the TensorFlow ecosystem, the same deep learning library used in Essentia, which is very convenient.
- It supports multiple backend options such as WebGL and WASM for inference on browsers or Node.js, which provides flexibility for future scenarios.
- It provides a conversion tool able to easily handle the format of existing Essentia models.

We used the *TensorFlow.js* converter[7] to port the models from frozen protocol buffers to the *TensorFlow.js* format. While in the frozen format the topology and weights are contained in the same binary file, *TensorFlow.js* models are defined in two files: a human-readable JSON file containing the topology and a binary file with the model weights. None of the weight quantization options offered by the converter were applied. The models take approximately the same data size after conversion.

We compared the activations generated by both the original and the converted models finding minimal numerical differences in the range of $1e^{-5}$. We have also seen similar differences when testing the original models under different computer architectures or TensorFlow versions. After a further inspection of prediction outcomes, we conclude that they are too small to alter the sense of the predictions in any case.

All the converted models are available for download on the Essentia website.[8] They can be used for inference on a wide variety of devices without any necessity for a dedicated GPU (similar to *TensorFlow.js*).

[7]https://github.com/tensorflow/tfjs
[8]https://essentia.upf.edu/models

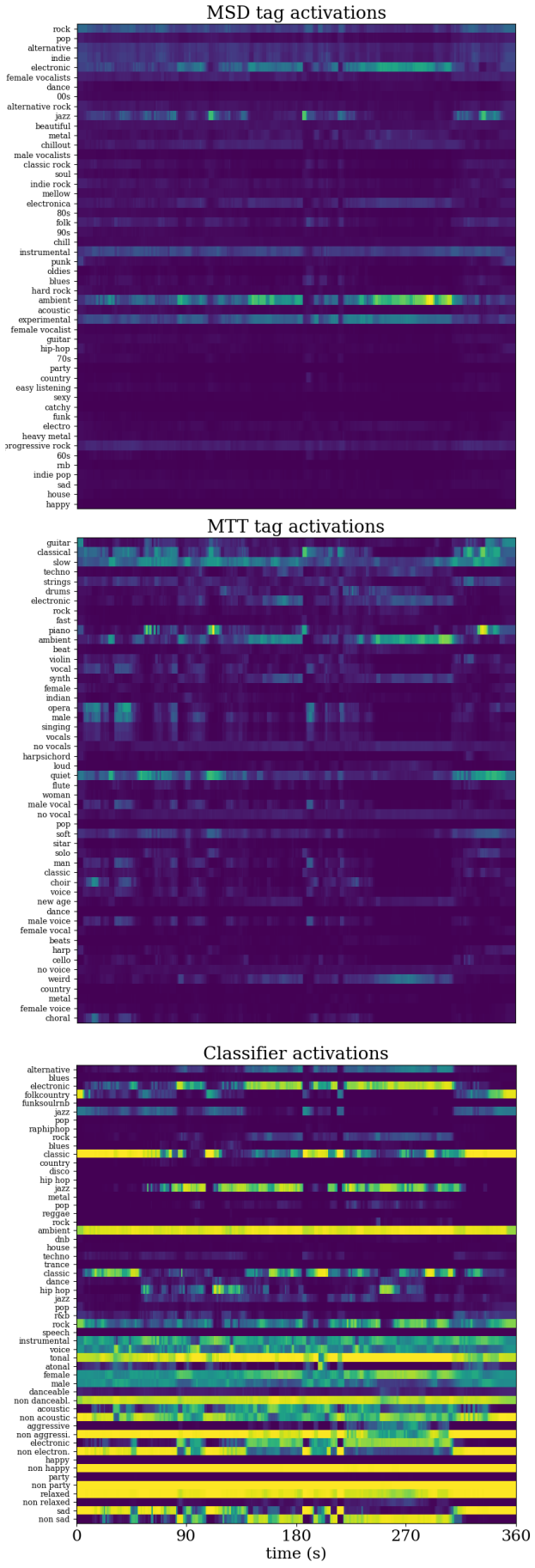

Figure 1: Activations for the MSD and MTT auto-tagging taxonomies and for all the transfer learning classifiers.

```
// Import Essentia WASM backend
import {EssentiaWASM} from './essentia-wasm';
import {EssentiaTFInputExtractor} from
'./essentia.js-model.es.js';

// Instantiate feature extractor for MusiCNN-based models
const extractor = new EssentiaTFInputExtractor(EssentiaWASM, '
    musicnn');

// Load a mono audio file from a given URL using Web Audio API
const audioURL = "https://freesound.org/data/previews
    /328/328857_230356-lq.mp3";
const audioContext = new AudioContext();
const audioBuffer = await extractor.getAudioBufferFromURL(
    audioURL, audioContext);

// Downsample audio to required sample rate
const audio = extractor.downsampleAudioBuffer(audioBuffer);

// Compute mel-spectrogram
let inputFeature = extractor.computeFrameWise(audio);
```

Listing 1: Example of offline audio feature extraction for the MusiCNN-based models using *Essentia.js-model* via ES6 style imports.

4. ESSENTIA.JS-MODEL

To use our pre-trained models in *TensorFlow.js*, one would have to implement the exact audio representations needed by the models as an input, which requires some development effort and domain knowledge. Models based on different CNN architectures expect different types and resolutions of input spectrogram representations for inference. Yet, in a regular web development workflow, many users won't necessarily need to know these specifics. We therefore developed *essentia.js-model*, an add-on JS module for *Essentia.js*. It combines both feature extraction using *Essentia.js* and model inference using *TensorFlow.js*. The APIs for achieving both of these processes are decoupled to allow more complex use-cases (for example, doing feature extraction and inference sessions in separate web workers). The detailed API documentation of the module is available online.[9]

4.1 Getting started

In this section, we outline several usage examples and application scenarios for getting started with *essentia.js-model*. The library can be imported into a web application using the following methods:

- **HTML <script> tag**. The simplest way to use *essentia.js-model* module is by using it with the HTML *<script>* tag. Note that, this will run your model inference on the main UI thread.
- **NPM/Yarn**. It can be also accessed from NPM by installing the latest version of *Essentia.js* with the command `npm install essentia.js` or `yarn add essentia.js`.
- **ES6 class imports**. *essentia.js-model* can be also imported using the ES6 class style imports in JS using the builds distributed on Github releases[10] or on NPM. This allows users to integrate the code into any modern JS framework. Listing 1 and 2 show an example of using ES6 style imports of *essentia.js-model*.
- **CDN**. We provide CDN links for instantly serving the these builds online using free third-party web services.

[9]https://essentia.upf.edu/essentiajs
[10]https://github.com/MTG/essentia.js/releases

```
// Import essentia.js-model and tensorflow.js
import {TensorflowMusiCNN} from './essentia.js-model.es.js';
import * as tf from "@tensorflow/tfjs";

// Path where the tfjs models are stored
const modelURL = "./autotagging/msd/msd-musicnn-1/model.json";

// Create an instance of EssentiaTensorflowJSModel
const musiCNN = new TensorflowMusiCNN(tf, modelURL);

// Promise for initializing the model
await musiCNN.initialize();

// Run model inference on the given feature input
let prediction = await musiCNN.predict(inputFeature);
```

Listing 2: Example of inference of MusiCNN-based models from the feature input computed in Listing 1 using *Essentia.js-model* via ES6 style imports.

4.1.1 Input feature extraction

The proposed module provides an interface for feature extraction via the *EssentiaTFInputExtractor* class. This class helps the user ensure the correct type and size of input audio feature representation matching the desired models of choice. Its constructor is created by passing the *EssentiaWASM* import from the Essentia WASM backend shipped with *Essentia.js* and choosing your target extractor type. The *compute* method of the class computes the feature representation for a given audio frame. Listing 1 shows an example of using the class for an offline feature extraction task with the MusiCNN-based models.

4.1.2 Inference

The proposed module provides its model inference functionalities through the classes *TensorflowMusiCNN*, *TensorflowVGGish* and *TensorflowTempoCNN* for models with the MusiCNN, VGGish and TempoCNN architectures respectively. Each of these classes' constructor is created by passing a global import of *TensorFlow.js* package and path to where the pre-trained model is stored (can be both an URL or a local file path). The *predict* method returns the output of the inference session as a promise for a given input feature representation which is pre-computed by *EssentiaTFInputExtractor*. Listing 2 shows an example of using *TensorflowMusiCNN*.

4.2 Applications

There are a lot of potential web applications that can be built with *essentia.js-model*. The library, along with the pre-trained models, provides algorithms for typical sound and music analysis tasks such as music auto-tagging, tempo estimation, genre identification, and mood classification, to mention a few. We show some starter web application examples for the above-mentioned use-cases in our online documentation.[11] Besides, these models can also be used for transfer learning tasks, using the model output as features to train a new ML model.

Web applications with real-time analysis can be built by leveraging *AudioWorklet* for audio feature extraction and *WebWorker* for model inference. We have put together a minimal code example[12] using these web technologies to

[11]https://mtg.github.io/essentia.js/docs/api/
[12]https://glitch.com/edit/#!/essentia-js-models-rt?path=README.md

perform real-time inference with an auto-tagging MusiCNN model. We also provide this as a live web example.[13] For JS server applications, TensorFlow has a dedicated wrapper *tfjs-node* with direct bindings to the TensorFlow C API, which can be used in Node.js run-time applications. Note that not all of these models we provide are suitable for real-time inference on web browsers, though.

4.3 Benchmarking

We measured the performance of the models across different platforms: Node.js, Chrome and Firefox browsers (since they are amongst the most widely used[14]) and, as a baseline for comparison, the standard Essentia C++ models using its Python bindings. We carried out the reported tests on a Macintosh machine running macOS 10.15.7, with a 2.2 GHz 6-core i7 CPU and 16 GB of RAM. For each model, we ran three benchmark functions: time spent on feature extraction (using the algorithms corresponding to each model's architecture), on inference, and on the entire end-to-end process, i.e. feature extraction and inference. All tests were performed on a one-minute *WAV* audio file (since this is closer to a real use case than shorter fragments), re-sampled to 16 kHz, and mixed down to mono. Note that browser tests were performed on the main UI thread, and we did not benchmark the models' real-time capabilities. We tested two auto-tagging architectures, MusiCNN and VGG, and four transfer-learning classifiers, trained for genre and mood classification using pre-trained MusiCNN and VGGish models.

As a baseline we tested the C++ implementation of the algorithms (using Python bindings), running each of the test functions (feature extraction, inference, and end-to-end) 50 times and taking the average of the time measured using Python's `time.perf_counter()`. The same approach was taken to perform the tests on Node.js and browsers (using `Date.now()`), except that the average of 10 repetitions was taken instead of 50, since running the models for 1min of audio on the Javascript platforms took longer than the baseline. All browser benchmarks were done using the *TensorFlow.js* WebGL backend, which is selected by default on desktop devices. Figure 2 shows the results of these tests. Overall the end-to-end process takes from 7.1 to 15.7 times slower than on the native platform, taking up to 18.2 seconds to analyze a 60-second audio input. Feature extraction is consistently slower than inference across all benchmarks, which may be due to the fact that tensor operations can be run in parallel whereas feature extraction is not. We repeated these benchmarks using the WASM *TensorFlow.js* backend which, unlike the WebGL backend, does not execute tensor operations on the device's GPU: with this backend, inference is on average 12 times slower on Chrome, and 90 times slower on Firefox. We are unsure what could be the causes of such discrepancies between Chrome and Firefox.

5. CONCLUSIONS

We have presented a collection of pre-trained ML models ported for *TensorFlow.js* and a new *Essentia.js* add-on module for their easy use in Web applications requiring music/audio analysis. These models address some of the common music classification tasks, tempo estimation, and extraction of music feature embeddings, some of them available for real-time applications. The new addition of ML models expands the functionality of *Essentia.js* even further, allowing for many industrial and creative use-cases in Web Audio.

In our future work, we will focus on adding more pre-trained models for different MIR use-cases. We also aim to develop interesting web applications that go beyond typical MIR tasks to attract and build a diverse user community. For better portability, we will also consider creating the models in the *ONNX* format.

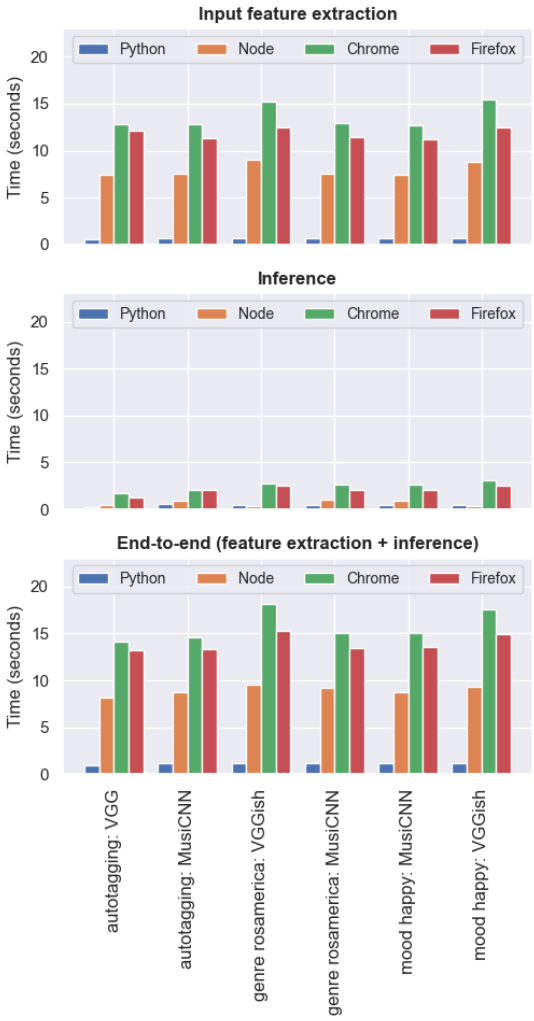

Figure 2: Per-model performance in seconds for each of the tested platforms averaged over 50 runs for the Python baseline and 10 runs for Node.js and browsers. Results for 1min WAV audio file.

6. REFERENCES

[1] ml5.js: Friendly machine learning for the Web. https://ml5js.org. Accessed: 2021-03-15.

[2] P. Alonso-Jiménez, D. Bogdanov, J. Pons, and X. Serra. Tensorflow audio models in Essentia. In

[13] https://mtg.github.io/essentia.js/examples/

[14] https://gs.statcounter.com/browser-market-share#yearly-2019-2020-bar

IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2020), 2020.

[3] P. Alonso-Jiménez, D. Bogdanov, and X. Serra. Deep embeddings with Essentia models. In *International Society for Music Information Retrieval Conference (ISMIR 2020) Late Breaking Demo*, 2020.

[4] F. Bernardo, C. Kiefer, and T. Magnusson. An audioworklet-based signal engine for a live coding language ecosystem. In *International Web Audio Conference (WAC 2019)*, 2019.

[5] T. Bertin-Mahieux, D. P. W. Ellis, B. Whitman, and P. Lamere. The Million Song Dataset. In *International Society for Music Information Retrieval Conference (ISMIR 2011)*, 2011.

[6] D. Bogdanov, N. Wack, E. Gómez, S. Gulati, P. Herrera, O. Mayor, G. Roma, J. Salamon, J. Zapata, and X. Serra. Essentia: An audio analysis library for music information retrieval. In *International Society for Music Information Retrieval Conference (ISMIR 2013)*, 2013.

[7] N. Collins and S. Knotts. A JavaScript musical machine listening library. Technical Report, 2019.

[8] A. Correya, D. Bogdanov, L. Joglar-Ongay, and X. Serra. Essentia.js: a JavaScript library for music and audio analysis on the web. In *International Society for Music Information Retrieval Conference (ISMIR 2020)*, 2020.

[9] J. Fiala, N. Segal, and H. A. Rawlinson. Meyda: an audio feature extraction library for the Web Audio API. In *Web Audio Conference (WAC 2015)*, 2015.

[10] N. Jillings, J. Bullock, and R. Stables. JS-Xtract: A realtime audio feature extraction library for the Web. In *International Society for Music Information Retrieval Conference (ISMIR 2016) Late Breaking Demo*, 2016.

[11] E. Law, K. West, M. I. Mandel, M. Bay, and J. S. Downie. Evaluation of algorithms using games: The case of music tagging. In *International Society for Music Information Retrieval Conference (ISMIR 2009)*, 2009.

[12] B. Matuszewski and N. Schnell. Lfo–a graph-based modular approach to the processing of data streams. In *Web Audio Conference (WAC 2017)*, 2017.

[13] J. Pons and X. Serra. musicnn: Pre-trained convolutional neural networks for music audio tagging. In *International Society for Music Information Retrieval Conference (ISMIR 2019) Late Breaking Demo*.

[14] A. Roberts, C. Hawthorne, and I. Simon. Magenta.js: A JavaScript API for augmenting creativity with deep learning. In *Joint Workshop on Machine Learning for Music (ICML)*, 2018.

[15] H. Schreiber and M. Müller. Musical tempo and key estimation using convolutional neural networks with directional filters. In *Sound and Music Computing Conference (SMC 2019)*, 2019.