Essentia TensorFlow Models for Audio and Music Processing on the Web

Albin Correya ∗ Pablo Alonso-Jiménez Jorge Marcos-Fernández
Music Technology Group Music Technology Group Music Technology Group
Universitat Pompeu Fabra Universitat Pompeu Fabra Universitat Pompeu Fabra
Barcelona, Spain Barcelona, Spain Barcelona, Spain
albin.a.correya@gmail.com pablo.alonso@upf.edu jorge.marcos@upf.edu

Xavier Serra Dmitry Bogdanov
Music Technology Group Universitat Pompeu Fabra
Universitat Pompeu Fabra Universitat Pompeu Fabra
Barcelona, Spain Barcelona, Spain
xavier.serra@upf.edu 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 a growing number of 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 and ONNX.js2 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 Hub3 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

1https://www.tensorflow.org/js
2https://github.com/microsoft/onnxjs
3https://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 Essentia.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\(^3\) 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 task.

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

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

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.

---

3https://essentia.upf.edu/essentia.js
5https://creativecommons.org/licenses/by-nc-nd/4.0
6https://essentia.upf.edu/machine_learning.html
Table 2: The Essentia models. RF: Receptive field, AT: Auto-tagging, TL: Transfer learning.

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

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 an 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 $10^{-5}$. 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 { EssentialTFInputExtractor } from 'essentia-js-model.es.js';

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

// Load a mono audio file from a given URL using Web Audio API
const audioURL = 'https://freesound.org/data/previews/30652856-lq.mp3';
const audioContext = new AudioContext();
const audioBuffer = await extractor.getAudioBufferFromURL(audioURL, audioContext);

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

// Comute 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.

4.1.1 Input feature extraction

The proposed module provides an interface for feature extraction via the EssentialTFInputExtractor 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 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 EssentiaTensorflowWASMS. 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

\(^9\)https://essentia.upf.edu/essentia.js

\(^10\)https://github.com/MTG/essentia.js/releases

\(^11\)https://mtg.github.io/essentia.js/docs/api/

\(^12\)https://glitch.com/edit/#!/essentia-js-models-rt?path=README.md
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.

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.

6. REFERENCES


