madmom: a new Python Audio and Music Signal Processing Library

Sebastian Böck†, Filip Korzeniowski†, Jan Schlüter‡, Florian Krebs†, Gerhard Widmer†‡
† Department of Computational Perception, Johannes Kepler University Linz, Austria
‡ Austrian Research Institute for Artificial Intelligence (OFAI), Vienna, Austria

ABSTRACT

In this paper, we present *madmom*, an open-source audio processing and music information retrieval (MIR) library written in Python. *madmom* features a concise, *NumPy*-compatible, object oriented design with simple calling conventions and sensible default values for all parameters, facilitating fast prototyping of MIR applications. Prototypes can be seamlessly converted into callable processing pipelines through *madmom*'s concept of *Processors*, callable objects that run transparently on multiple cores. *Processors* can also be serialised, saved, and re-run to allow results to be easily reproduced anywhere.

Apart from low-level audio processing, *madmom* puts emphasis on musically meaningful features. Many of these incorporate machine learning techniques and *madmom* provides a module that implements some methods commonly used in MIR such as hidden Markov models and neural networks. Additionally, *madmom* comes with several state-of-the-art MIR algorithms for onset detection, beat, downbeat and meter tracking, tempo estimation, and chord recognition. These can easily be incorporated into bigger MIR systems or run as stand-alone programs.

1. INTRODUCTION

Music information retrieval (MIR) has become an emerging research area over the last 15 years. Especially audio-based MIR has become more and more important, since the amount of available audio data in the last years exploded beyond being manageable manually.

Most state-of-the-art audio-based MIR algorithms consist of two components: first, low-level features are extracted from the audio signal (*feature extraction* stage), and then the features are analysed (*feature analysis* stage) to retrieve the desired information. Most current MIR systems incorporate machine learning algorithms in the feature analysis stage, with neural networks currently being the most popular and successful ones [2, 3, 10, 18].

Numerous software libraries have been proposed over the years to facilitate research and development of applications in MIR. Some libraries concentrate on low-level feature extraction from audio signals, such as *Marsyas* [20], *YAAFE* [15] and *openSMILE* [8]. Others also include higher level feature extraction such as onset and beat detection as for example in the *MIRtoolbox* [14], *Essentia* [6] and *LibROSA* [16]. However, to our knowledge, there exist no library that also includes machine learning components (except *Marsyas* [20], which contains two classifiers), although machine learning components are crucial in current MIR applications.

Therefore, we propose *madmom*, a library that incorporates low-level feature extraction *and* high-level feature analysis based on machine learning methods. This allows the construction of the full processing chain within a single software framework, making it possible to build standalone programs without any dependency on other machine learning frameworks. Moreover, *madmom* comes with several state-of-the-art systems including their trained models, for example for onset detection [7, 18, 19], tempo estimation [3], beat [2, 10] and downbeat tracking [4, 13], and chord recognition [11, 12].

madmom is written in Python, which has become the language of choice for scientific computing for many people due to its free availability and its simplicity of use. The code is released under BSD license and pre-trained models are released under the CC BY-NC-SA 4.0 license.

1.1 Design and Functionality

1.1.1 Object-oriented programming

madmom follows an object-oriented programming (OOP) approach. We encapsulate everything in objects that are often designed as subclasses of *NumPy*'s *ndarray*, offering all array handling routines inherited from *NumPy* [21] with additional functionality. This compactly bundles data and meta-data (e.g. a *Spectrogram* and its *frame rate*) and simplifies meta-data handling for the user.

1.1.2 Rapid prototyping

madmom aims at minimising the turnaround time from a research idea to a software prototype. Thus, object instantiation is made as simple as possible: e.g., a *Spectrogram* object can be instantiated with a single line of code by only providing the path to an audio file. *madmom* automatically creates all objects in between using sensible default values.

1.1.3 Simple conversion into runnable programs

Once an audio processing algorithm is prototyped, the complete workflow should be easily transformable into a runnable standalone program with a consistent calling interface. This is implemented using *madmom*'s concept of *Processors*.

© Sebastian Böck, Filip Korzeniowski, Jan Schlüter, Florian Krebs, Gerhard Widmer. Licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0). **Attribution:** Sebastian Böck, Filip Korzeniowski, Jan Schlüter, Florian Krebs, Gerhard Widmer. "madmom: a new Python Audio and Music Signal Processing Library", Extended abstracts for the Late-Breaking Demo Session of the 17th International Society for Music Information Retrieval Conference, 2016.

1.1.4 *Machine learning integration*

We aim at a seamless integration of machine learning methods without the need of any third party modules. We limit ourselves to testing capabilities (applying pre-trained models), since it is impossible to keep up with newly emerging training methods in the various machine learning domains. Models that have been trained in an external library should be easily be convertible to an internal *madmom* model format.

1.1.5 *State-of-the-art features*

Many existing libraries provide a huge variety of low-level features, but few musically meaningful high-level features. *madmom* tries to close this gap by offering high-quality state-of-the-art feature extractors for onsets, beats, downbeats, chords, tempo, etc.

1.1.6 *Reproducible research*

In order to foster reproducible research, we want to be able to save and load the specific settings used to obtain the results for a certain experiment. In *madmom* this is implemented using Python's own *pickle* functionality which allows to save an entire processing chain (including all settings) to a file.

1.1.7 *Few dependencies*

madmom is built on top of three excellent and wide-spread libraries: *NumPy* [21] provides all the array handling subroutines for *madmom*'s *data classes*. *SciPy* [9] provides optimised routines for the fast Fourier transform (FFT), linear algebra operations and sparse matrix representations. Finally, *Cython* [1] is used to speed up time critical parts of the library by automatically generating C code from a Python-like syntax and then compiling and linking it into extensions which can be transparently used from within Python. These libraries are the only installation and runtime dependencies of *madmom* besides the *Python* standard library itself, supported in version 2.7 as well as 3.3 and newer.

1.1.8 *Multi-core capability*

We designed *madmom* to be able to exploit the multi-core capabilities of modern computer architectures, by providing functionality to run several programs or *Processors* in parallel.

1.1.9 *Extensive documentation*

All source code files contain thorough documentation following the *NumPy* format. The complete API reference, instruction on how to build and install the library, as well as interactive *Jupyter* [17] notebooks can be found online at `http://madmom.readthedocs.io`. The documentation is built automatically with *Sphinx*.

1.1.10 *Open development process*

We follow an open development process and the source code and documentation of our project is publicly available on GitHub: `http://github.com/CPJKU/madmom`. To maintain high code quality, we use continuous integration testing via TravisCI, code quality tests via QuantifiedCode, and test coverage via Coveralls.

2. LIBRARY DESCRIPTION

In this section, we will describe the overall architecture of *madmom*, its packages as well as the provided standalone programs.

madmom's main API is composed of classes, but much of the functionality is implemented as functions (in turn used internally by the classes). This way, *madmom* offers the 'best of both worlds': concise interfaces exposed through classes, and detailed access to functionality through functions. In general, the classes can be split in two different types: the so called *data classes* and *processor classes*.

Data classes represent data entities such as audio signals or spectrograms. They are implemented as subclasses of *NumPy*'s *ndarray*, and thus offer all array handling routines inherited directly from *NumPy* (e.g., transposing or saving the data to file in either binary or human readable format). These classes are enriched by additional attributes and expose additional functionality via methods.

Processor classes exclusively store information on how to process data, i.e. how to transform one data class into another (e.g., from an (audio-)*Signal* into a *Spectrogram*). In order to build chains of transformations, each data class has its corresponding processor class, which implements this transformation. This enables a simple and fast conversion of algorithm prototypes to callable processing pipelines.

2.1 Standalone Programs

madmom comes with a set of standalone programs, covering many areas of MIR. The outstanding results in Table 1 highlight the state-of-the-art features *madmom* provides.

Table 1. Ranks of the programs included in *madmom* for the MIREX evaluations, results aggregated over all years (2006-2016).

Program	Task	Year	Rank
CNNOnsetDetector [18]	onset	2016	1
OnsetDetector [7]	onset	2013	2
BeatTracker [5]	beat MCK	2015	1
DBNBeatTracker [2]	beat SMC	2015	1
CRFBeatDetector [10]	beat MAZ	2015	1
DBNDownBeatTracker [4]	downbeat	2016	1
TempoDetector [3]	tempo	2015	1
CNNChordRecognition [12]	chord	2016	1

3. ACKNOWLEDGMENTS

This work is supported by the European Union Seventh Framework Programme FP7 / 2007-2013 through the GiantSteps project (grant agreement no. 610591) and the Austrian Science Fund (FWF) project Z159.

4. REFERENCES

[1] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. Seljebotn, and K. Smith. Cython: The Best of Both Worlds. *Computing in Science Engineering*, 13(2), 2011.

[2] S. Böck, F. Krebs, and G. Widmer. A Multi-model Approach to Beat Tracking considering Heterogeneous Music Styles. In *Proc. of the 15th Int. Society for Music Information Retrieval Conf. (ISMIR)*, 2014.

[3] S. Böck, F. Krebs, and G. Widmer. Accurate Tempo Estimation based on Recurrent Neural Networks and Resonating Comb Filters. In *Proc. of the 16th Int. Society for Music Information Retrieval Conf. (ISMIR)*, 2015.

[4] S. Böck, F. Krebs, and G. Widmer. Joint Beat and Downbeat Tracking with Recurrent Neural Networks. In *Proc. of the 17th Int. Society for Music Information Retrieval Conf. (ISMIR)*, 2016.

[5] S. Böck and M. Schedl. Enhanced Beat Tracking with Context-Aware Neural Networks. In *Proc. of the 14th Int. Conf. on Digital Audio Effects (DAFx)*, 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 open source library for sound and music analysis. In *In Proc. of ACM Multimedia*, 2013.

[7] F. Eyben, S. Böck, B. Schuller, and A. Graves. Universal Onset Detection with Bidirectional Long Short-Term Memory Neural Networks. In *Proc. of the 11th Int. Society for Music Information Retrieval Conf. (ISMIR)*, 2010.

[8] F. Eyben, F. Weninger, F. Gross, and B. Schuller. Recent Developments in openSMILE, the Munich Open-Source Multimedia Feature Extractor. In *In Proc. of ACM Multimedia*, Barcelona, Spain, 2013.

[9] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2016-05-20].

[10] F. Korzeniowski, S. Böck, and G. Widmer. Probabilistic Extraction of Beat Positions from a Beat Activation Function. In *Proc. of the 15th Int. Society for Music Information Retrieval Conf. (ISMIR)*, 2014.

[11] F. Korzeniowski and G. Widmer. Feature learning for chord recognition: The deep chroma extractor. In *Proc. of the 17th Int. Society for Music Information Retrieval Conf. (ISMIR)*, 2016.

[12] F. Korzeniowski and G. Widmer. A fully convolutional deep auditory model for musical chord recognition. In *Proc. of the IEEE Int. Workshop on Machine Learning for Signal Processing (MLSP)*, 2016.

[13] F. Krebs, S. Böck, and G. Widmer. Rhythmic Pattern Modeling for Beat and Downbeat Tracking in Musical Audio. In *Proc. of the 14th Int. Society for Music Information Retrieval Conf. (ISMIR)*, 2013.

[14] O. Lartillot and P. Toiviainen. A Matlab toolbox for musical feature extraction from audio. In *Proc. of the 10th Int. Conf. on Digital Audio Effects (DAFx)*, 2007.

[15] B. Mathieu, S. Essid, T. Fillon, and J. Prado. YAAFE, an easy to use and efficient audio feature extraction software. In *Proc. of the 11th Int. Society for Music Information Retrieval Conf. (ISMIR)*, 2010.

[16] B. McFee, C. Raffel, D. Liang, D. Ellis, M. McVicar, E. Battenberg, and O. Nieto. librosa: Audio and Music Signal Analysis in Python. In *Proc. of the 14th Python in Science Conf. (SCIPY)*, 2015.

[17] F. Pérez and B. E. Granger. IPython: A System for Interactive Scientific Computing. *Computing in Science Engineering*, 9(3), 2007.

[18] J. Schlüter and S. Böck. Musical Onset Detection with Convolutional Neural Networks. In *Proceedings of the 6th International Workshop on Machine Learning and Music*, Prague, Czech Republic, 2013.

[19] J. Schlüter and S. Böck. Improved musical onset detection with convolutional neural networks. In *Proc. of the 39th Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, 2014.

[20] G. Tzanetakis and P. Cook. MARSYAS: a framework for audio analysis. *Organised Sound*, 4, 2000.

[21] S. van der Walt, S. C. Colbert, and G. Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science Engineering*, 13(2), 2011.