Data-Driven Approaches for Tempo and Key Estimation of Music Recordings

Datengetriebene Verfahren für Tempo- und Tonart-Schätzung von Musikaufnahmen

Dissertation

Der Technischen Fakultät

der Friedrich-Alexander-Universität Erlangen-Nürnberg

zur

Erlangung des Doktorgrades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)

vorgelegt von

Hendrik Schreiber

aus

Soest

Als Dissertation genehmigt
von der Technischen Fakultät
der Friedrich-Alexander-Universität Erlangen-Nürnberg

Tag der mündlichen Prüfung:	1. Juli 2020
Vorsitzender des Promotionsorgans:	Prof. Dr.-Ing. Andreas Paul Fröba
1. Gutachter:	Prof. Dr. rer. nat. Meinard Müller
2. Gutachter:	Prof. Dr.-Ing. Sebastian Stober

Abstract

In recent years, we have witnessed the creation of large digital music collections, accessible, for example, via streaming services. Efficient retrieval from such collections, which goes beyond simple text searches, requires automated music analysis methods. Creating such methods is a central part of the research area Music Information Retrieval (MIR). In this thesis, we propose, explore, and analyze novel data-driven approaches for the two MIR analysis tasks *tempo* and *key estimation* for music recordings. Tempo estimation is often defined as determining the number of times a person would "tap" per time interval when listening to music. Key estimation labels music recordings with a chord name describing its tonal center, e.g., C major. Both tasks are well established in MIR research.

To improve tempo estimation, we focus mainly on shortcomings of existing approaches, particularly estimates on the wrong metrical level, known as *octave errors*. We first propose novel methods using digital signal processing and traditional feature engineering. We then re-formulate the signal-processing pipeline as a deep computational graph with trainable weights. This allows us to take a purely data-driven approach using supervised machine learning (ML) with convolutional neural networks (CNN). We find that the same kinds of networks can also be used for key estimation by changing the orientation of directional filters. To improve our understanding of these systems, we systematically explore network architectures for both global and local estimation, with varying depths and filter shapes, as well as different ways of splitting datasets for training, validation, and testing. In particular, we investigate the effects of learning on different splits of cross-version datasets, i.e., datasets that contain multiple recordings of the same pieces.

For training and evaluation the proposed data-driven approaches rely on curated datasets covering certain key and tempo ranges as well as genres. Datasets are therefore another focus of this work. Additionally to creating or deriving new datasets for both tasks, we evaluate the quality and suitability of popular tempo datasets and metrics, and conclude that there is ample room for improvement. To promote better, transparent evaluation, we propose new metrics and establish a large open and public repository containing evaluation code, reference annotations, and estimates.

Zusammenfassung

In den vergangenen Jahren sind große digitale Musiksammlungen entstanden, die – beispielsweise – über Streaming-Dienste einfach zugänglich sind. Ein effizientes Retrieval aus solchen Sammlungen, das über die simple Textsuche hinausgeht, erfordert automatisierte Musikanalysemethoden. Das Erforschen solcher Methoden ist ein zentraler Bestandteil des Forschungsgebiets Music Information Retrieval (MIR). In dieser Arbeit stellen wir neue datengetriebene Ansätze für die beiden MIR-Analyseaufgaben Tempo- und Tonart-Schätzung für Musikaufnahmen vor und analysieren sie. Dabei wird Tempo-Schätzung oft definiert als das Zählen der Male, die eine Person beim Hören von Musik pro Zeitintervall "klopfen" würde. Tonart-Schätzung weist Musikaufnahmen einen Akkordnamen zu, der den Klangmittelpunkt beschreibt, z.B. C-Dur. Beide Aufgaben sind in der MIR-Forschung fest verankert.

Um die Tempo-Schätzung zu verbessern, konzentrieren wir uns hauptsächlich auf Defizite bestehender Ansätze, insbesondere Schätzungen auf der falschen metrischen Ebene, den sogenannten *Oktavfehlern*. Dazu schlagen wir zunächst neue Methoden vor, die sich der digitalen Signalverarbeitung und des traditionellen Feature-Engineerings bedienen. Anschließend formulieren wir die Signalverarbeitungspipeline in eine tiefe, graphenartige Rechenstruktur mit trainierbaren Parametern um. Dies ermöglicht uns einen rein datengetriebenen Ansatz unter Verwendung von überwachtem maschinellem Lernen (ML) mit neuronalen Netzen – insbesondere Convolutional Neural Networks (CNN). Wir stellen fest, dass durch das Ändern der Orientierung von gerichteten Filtern, die gleichen Arten von Netzwerken auch für die Tonart-Schätzung verwendet werden können. Um unser Verständnis dieser Systeme zu vertiefen, untersuchen wir systematisch Netzwerkarchitekturen für die globale und lokale Schätzung mit unterschiedlichen Tiefen und Filterformen sowie verschiedenen Datensatz-Splits für Training, Validierung und Test. Insbesondere betrachten wir, welche Auswirkungen das Lernen auf verschiedenen Splits von Cross-Version-Datensätzen hat. Dies sind Datensätze, die mehrere Aufnahmen derselben Stücke enthalten.

Für Training und Evaluation stützen sich die vorgeschlagenen datengetriebenen Ansätze auf kuratierte Datensätze, die bestimmte Tonart- und Tempobereiche sowie Genres abdecken. Ein weiterer Schwerpunkt dieser Arbeit liegt daher auf den Datensätzen selbst. Zusätzlich zum Erstellen oder Ableiten neuer Datensätze für beide o.g. Aufgaben evaluieren wir die Qualität und Eignung gängiger Tempo-Datensätze und -Metriken und kommen zu dem Schluss, dass es

Raum für Verbesserungen gibt. Um eine bessere, transparentere Evaluation zu fördern, schlagen wir daher neue Metriken vor und etablieren ein großes, offenes und öffentliches Repository mit Evaluationscode, Referenzannotationen und Schätzungen.

Contents

1. Introduction

A little more than a quarter century ago, in 1992, the world of music was changed forever. The audio coding standard MPEG-1, Layer III—later known as MP3—had been approved just a few months ago [120] and was now presented in a publication by Stoll and Brandenburg [178]. This event marked the beginning of a new digital music era, in which music became ubiquitously accessible to everyone. A few years later, in 1995, the now iconic filename extension `.mp3` was chosen and the Fraunhofer Society released the first MP3 software encoder named l3enc. In the same year, the first real-time software MP3 player followed (WinPlay3), and in April of 1997, Nullsoft released the now legendary MP3 player Winamp. By itself, this might not have been a remarkable event, but it coincided with the exponential growth of the Internet. Between 1990 and 1997, the number of Internet hosts grew exponentially from just 313 thousand to 26 million.[1] The availability of a suitable audio format, a cheap, world-spanning communication platform, and free music player software made CD-ripping and file-sharing possible. For the first time in history, music was truly at the users' fingertips.[2] This development gave rise to a new class of software: *personal music library systems*. While the ubiquitous Winamp initially did not offer any library functions, other music players like Microsoft's Windows Media Player 7 (WMP7, released 7/2000) and Apple's iTunes[3] (released 1/2001) quickly added browse and search functionalities. They did so by exploiting basic metadata tags, which had been embedded into MP3 files using the initially informal ID3 standard [83].

Just before the appearance of iTunes and WMP7, another important event happened. In 1998, the virtual DJ solution MJ was released [26, 190]. Besides ID3 tag support, it already featured pitch change, scratching, mixing, and basic content analysis in the form of *automatic beat detection*. MJ's developers later joined Native Instruments, the developer of the still successful DJ software Traktor.[4]

By the middle of the 2000s, digital music aficionados had thousands of tracks on their personal computers and portable music players, but little support for building playlists, or—more generally—determining song- or artist-similarity. Even though online services like Last.fm,[5] which has

[1] Source: Internet Systems Consortium (ISC), archived at `https://web.archive.org/web/20120518101749/https://www.isc.org/solutions/survey/history`

[2] Bill Gates [54]: "information at your fingertips"

[3] Based on SoundJam MP (1999)

[4] `https://www.native-instruments.com/`

[5] `https://www.last.fm/`

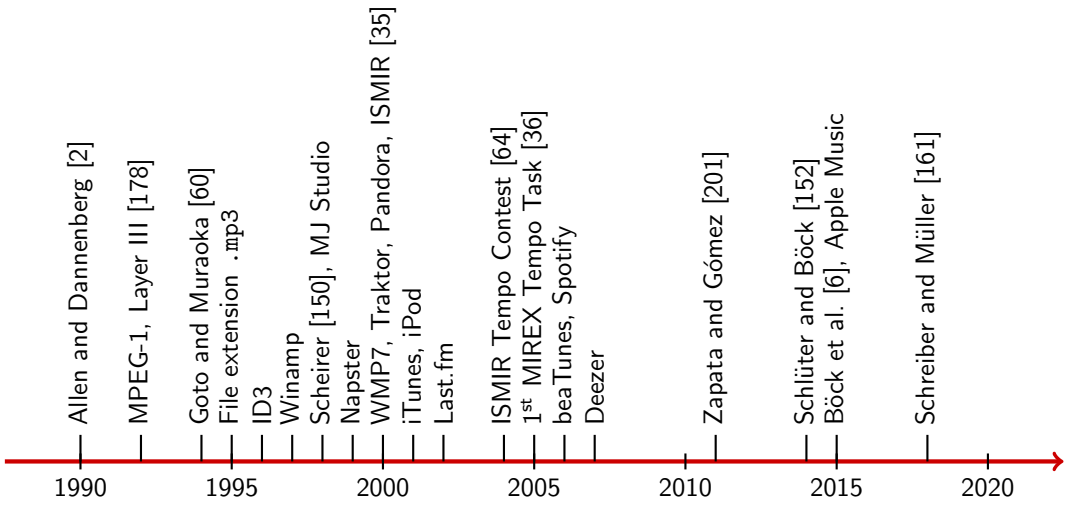


Figure 1.1.: Selection of important events and publications in beat tracking, tempo estimation, music research, and music software development. The tempo and beat tracking publications dated 2014 and later use DNNs as their main component.

pioneered collaborative filtering in the music domain, were already available, querying a local music library for tracks relevant to an information need more complex than a simple genre or artist search, was still very difficult. The desktop software beaTunes[6] ['biːtuːnz] released by tagtraum industries[7] in 7/2006 attempted to fill this niche (Appendix D). It supported users in correcting erroneous textual metadata (artist, title, genre, ...) and computing additional content-based annotations (tempo, timbre, key, ...). The resulting rich metadata could then be used as input to a user-customizable song-similarity function, which allowed building playlists of matching songs, thus attempting to solve a classic Music Information Retrieval (MIR) problem.

Parallel to the developments in the commercial world, digital music processing started to attract scientific attention. In the 1990s, early automatic beat tracking and tempo estimation methods were published [2, 60, 150], and before the turn of the millennium, an academic community formed around the intersection of library and information science, cognitive science, digital signal processing, and musicology. This led to the foundation of the International Society for Music Information Retrieval (ISMIR)[8] in 2000 [35]. To this day, the society is an international forum for research on the organization of music-related data. Many of the tasks beaTunes attempted to solve emerged as its core research topics—among them *music recommendation*, *similarity*, *tempo estimation*, and *key estimation*. During the 2000s, the MIR research community approached these tasks mostly using signal-processing techniques, handcrafted features, heuristics, and shallow machine learning (ML) methods [76]. Then, like many other scientific disciplines, MIR experienced a fundamental shift from feature engineering to automatic feature learning. Recent

[6] `https://www.beatunes.com/`
[7] Owned and operated by the author of this thesis.
[8] `http://www.ismir.net/`

solutions increasingly rely on deep ML architectures [119], and by now, deep neural networks (DNN) have become a widely accepted tool in MIR research.

The described commercial as well as scientific developments (see also Figure 1.1) form the context of this thesis. It revolves in particular around the tasks of *automatic tempo estimation* and, to a lesser degree, *automatic key estimation* for MIR systems like beaTunes. We loosely define tempo estimation as the attempt to determine the number of times a listener would "tap" to a beat with his or her foot per time interval [150, 33]. Key estimation tries to find "a set of pitch relationships that establish a note—or, better, a chord—as a tonal center" [142, p. 43], e.g., C major or D minor.

This thesis reflects the larger developments described in the brief history outline above in the sense that it starts out with using engineered features and simple models, and then moves towards straightforward signal representations and deeper ML models, showcasing *data-driven* approaches. Concretely, we first lay some digital signal-processing foundations and discuss how to build and improve conventional tempo estimation systems using onset signal strength functions and periodicity detection. Then we explore how such systems can be built using convolutional neural networks (CNN) and proceed to apply the same techniques to the related task of musical key detection. Of central importance for all presented approaches is data. It drives both training and evaluation of ML models, and helps us overcome the main challenges of automatic tempo estimation, in particular, non-constant local tempi, metrical estimation errors also known as octave errors, and the limitless variety of music genres, rhythms, and playing techniques. Therefore, dataset creation, analysis, and improvement constitute a large part of this work.

Using ML techniques and suitable data, we can build ever better estimation systems until we have to ask ourselves if we have "solved" the task. But since *use cases* often play a minor role in publications about new MIR methods, it is difficult to determine what constitutes a solution. Part of this thesis is therefore dedicated to critically discuss tempo estimation system evaluation. After all, we seek to better understand how to build systems that are relevant and useful to the user.

1.1. Structure of this Thesis

This thesis consists of eight main chapters.

In Chapter 2, we begin with explaining some fundamentals of digital signal processing (DSP) for music recordings. We then proceed with describing a basic tempo estimation system built on a spectral novelty function, which is analyzed for periodicities using the discrete Fourier transform, and then interpreted with special emphasis on harmonics. The presented system illustrates the

task of tempo estimation itself and serves as a baseline for the following chapters. We further introduce the two main evaluation metrics used throughout this thesis and provide an informal overview of important datasets.

In Chapter 3, we present two different approaches to correcting the metrical level of tempo estimates, also known as fixing the octave error. The first method, presented in Section 3.1, exploits a simple, approximately linear relationship between a single global feature, average spectral novelty, and listener perception of musical tempo. The second method, presented in Section 3.2, uses more sophisticated means and approaches the problem from a different direction. Using supervised learning with random forests [14] we predict algorithm-specific tempo estimation errors. In a post-processing step, these predictions can then be used to correct an algorithm's tempo estimates. While being straightforward and relying only on a small number of features, our proposed method significantly increases accuracy.

Leaving the world of traditional digital signal processing, we present a single-step tempo estimation system based solely on a convolutional neural network (CNN) in Chapter 4. Contrary to other systems, which typically first identify onsets or beats and then derive a tempo, the tempo is estimated directly from a conventional mel-spectrogram. This is achieved by framing tempo estimation as a multi-class classification problem using a network architecture that is inspired by DSP approaches. The system's CNN has been trained with the union of three datasets covering a large variety of genres and tempi using problem-specific data augmentation. Two of the three datasets are novel and have been released for research purposes. As input the system requires only 11.9 s of audio and is suitable for local as well as global tempo estimation. When used as a global estimator, it performs as well as or better than other state-of-the-art algorithms. Especially the exact estimation of tempo without tempo octave confusion is significantly improved. As local estimator it can be used to identify and visualize tempo drift in musical performances.

We cannot further improve the state of the art without proper evaluation, which in the case of tempo estimation relies on correctly annotated datasets. In Chapter 5, we therefore investigate a peculiarity: relative to other datasets, state-of-the-art tempo estimation algorithms perform poorly on the *GiantSteps Tempo* dataset for electronic dance music (EDM). To investigate why, we conducted a large-scale, crowdsourced experiment. In the collected data we observed significant tempo ambiguities, which we attribute to annotator subjectivity and tempo instability. As a further contribution, we constructed new annotations consisting of tempo distributions for each track. Using these annotations, we re-evaluated three recent tempo estimation systems achieving significantly improved results. The main conclusions of this investigation are that current tempo estimation systems perform better than previously thought and that evaluation quality needs to be improved.

As the logical next step, we present a thorough evaluation of current tempo estimation evaluation practices in Chapter 6. We discuss presumed and actual applications for global tempo estimation,

the pros and cons of commonly used metrics, and the suitability of popular datasets. To guide future research, we present results of a survey among domain experts that investigates today's applications, their requirements, and the usefulness of currently employed metrics. To aid future evaluations, we present a large public repository containing evaluation code as well as estimates by many different systems and different ground truths for popular datasets.

Next, in Chapter 7, we broaden the scope with respect to MIR tasks. Concretely, we explore how the different semantics of spectrograms' time and frequency axes can be exploited for musical tempo and key estimation using CNNs. By addressing both tasks with the same network architectures ranging from shallow, domain-specific approaches to deep variants with directional filters, we show that axis-aligned architectures perform similarly well as common VGG-style networks developed for computer vision, while being less vulnerable to confounding factors and requiring fewer model parameters.

In Chapters 8 and 9 we put the network architectures from Chapter 7 to practical use on two tasks we have not paid much attention to so far: *local* key and tempo estimation. We also switch our object of investigation from popular to Western classical music. In Chapter 8, we begin with local tempo estimation. Remarkably, even though local tempo estimation promises musicological insights into expressive musical performances, it has never received as much attention in the MIR research community as either beat tracking or global tempo estimation. One reason for this may be the lack of a generally accepted definition. In this chapter, we discuss how to model and measure local tempo in a musically meaningful way using a dataset of multiple performances (versions) of five Frédéric Chopin's Mazurkas as a use case. In particular, we explore how tempo stability can be measured and taken into account during evaluation. Comparing existing and newly trained systems, we find that CNN-based approaches can accurately measure local tempo even for expressive classical music, if trained on the target genre. Furthermore, we show that different training–test splits have a considerable impact on accuracy for difficult segments.

Finally, in Chapter 9, we address local key estimation. While global key and chord estimation for both popular and Western classical music recordings have received a lot of attention, little research has been devoted to estimating the local key for classical music, which may be due to its inherent ambiguity and subjectivity. We approach local key estimation on a cross-version dataset comprising nine performances (versions) of Schubert's song cycle *Winterreise*. We compare a Hidden Markov Model with a CNN-based approach. For both models, we employ a similar training procedure including the optimization of hyperparameters on a validation split. We systematically evaluate the model predictions and provide musical explanations for key confusions. As our main contribution, we explore how different training–test splits affect the models' efficacy. Splitting along the song axis, we find that both methods perform similarly well. Splitting along the version axis leads to clearly higher results for both approaches, but especially for the CNN,

which seems to effectively learn the harmonic progressions of the songs ("cover song effect") and successfully generalizes to unseen performances.

Supplemental material for this thesis is provided in the appendices. Appendix A describes how to obtain and use implementations of tempo estimation systems proposed in this work. Similarly, Appendix B describes key estimation systems. Appendix C adds a measure-wise evaluation of local key estimation errors that went beyond the scope of Chapter 9. Finally, in Appendix D, we give a brief overview of the main features of the consumer MIR system beaTunes.

Throughout the thesis, you will find notices like the following to highlight aspects of reproducibility.

Reproducibility

See Appendix X.1 for information about an open-source implementation.

The dataset for this experiment is available at DOI YYYYY.XXXXX.

1.2. Contributions

The main contributions of this thesis can be summarized as follows.

- Two novel techniques for correcting octave errors of global tempo estimators (Chapter 3).
- A CNN-based system capable of estimating local and global tempo directly from conventional mel-spectrograms (Chapter 4).
- Global tempo annotations for the *MTG Key* dataset [45] (Section 4.1.2).
- Free and open global tempo annotations featuring multiple annotations per track, modeling perceptual tempo ambiguity for the *GiantSteps Tempo* dataset (Chapter 5).
- A thorough evaluation of global tempo estimation evaluation, critically discussing use cases and the suitability of metrics and datasets, finding potential for improvement in every aspect (Sections 6.1 to 6.4).
- A survey among domain experts regarding applications and metrics for global tempo estimation (Section 6.5).
- A large open repository containing both reference annotations for popular global tempo datasets as well as estimates by many tempo estimation systems along with evaluation code (Section 6.6).
- Global key and tempo annotations for previews of the Lakh Midi Dataset (*LMD*) [136] (Section 4.1.1 and Chapter 7).

- Exploration of using directional filters in different deep and shallow CNN-architectures for tempo and key estimation (Chapter 7).
- A deepened understanding of how to model local tempo and tempo stability, establishing the coefficient of variation for inter-beat intervals as suitable measure (Chapter 8).
- An in-depth exploration of the effects of different training–test splits on local key and tempo estimation for cross-version datasets (Chapters 8 and 9).
- CNN-based local key estimation for classical Western music (Chapter 9).

1.3. Main Publications

The main contributions of this thesis are based on the following publications, which either appeared in peer-reviewed conference proceedings in the fields of audio signal processing and MIR or have been submitted to conferences or journals and are currently under review. ISMIR is widely considered the main conference for MIR, while ICASSP is the main conference for signal processing.

[158] Hendrik Schreiber and Meinard Müller. Exploiting global features for tempo octave correction. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 639–643, Florence, Italy, 2014.

[160] Hendrik Schreiber and Meinard Müller. A post-processing procedure for improving music tempo estimates using supervised learning. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 235–242, Suzhou, China, 2017.

[161] Hendrik Schreiber and Meinard Müller. A single-step approach to musical tempo estimation using a convolutional neural network. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 98–105, Paris, France, 2018.

[162] Hendrik Schreiber and Meinard Müller. A crowdsourced experiment for tempo estimation of electronic dance music. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 409–415, Paris, France, 2018.

[163] Hendrik Schreiber and Meinard Müller. Musical tempo and key estimation using convolutional neural networks with directional filters. In *Proceedings of the Sound and Music Computing Conference (SMC)*, pages 47–54, Málaga, Spain, 2019.

[166] Hendrik Schreiber, Christof Weiss, and Meinard Müller. Local key estimation in classical music recordings: A cross-version study on Schubert's Winterreise. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Barcelona, Spain, 2020.

[167] Hendrik Schreiber, Frank Zalkow, and Meinard Müller. Modeling and estimating local tempo: A case study on Chopin's Mazurkas. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Montreal, Quebec, Canada, October 2020.

[165] Hendrik Schreiber, Julián Urbano, and Meinard Müller. Music tempo estimation: Are we done yet? *Transactions of the International Society for Music Information Retrieval (TISMIR)*, 2020.

1.4. Additional Publications

The following peer-reviewed publications by the thesis author are also related to MIR, but are not considered in this thesis.

[164] Hendrik Schreiber, Peter Grosche, and Meinard Müller. A re-ordering strategy for accelerating index-based audio fingerprinting. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, pages 127–132, Miami, Florida, USA, 2011.

[159] Hendrik Schreiber and Meinard Müller. Accelerating index-based audio identification. *IEEE Transactions on Multimedia*, 16(6):1654–1664, 2014.

[154] Hendrik Schreiber. Improving genre annotations for the Million Song Dataset. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 241–247, Málaga, Spain, 2015.

[155] Hendrik Schreiber. Genre ontology learning: Comparing curated with crowd-sourced ontologies. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 400–406, New York City, NY, USA, 2016.

[37] Jonathan Driedger, Hendrik Schreiber, Bas de Haas, and Meinard Müller. Towards automatically correcting tapped beat annotations for music recordings. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 200–207, Delft, The Netherlands, 2019.

[12] Dmitry Bogdanov, Alastair Porter, Hendrik Schreiber, Julián Urbano, and Sergio Oramas. The AcousticBrainz genre dataset: Multi-source, multi-level, multi-label, and large-scale. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 360–367, Delft, The Netherlands, 2019.

1.5. Acknowledgments

Unknowingly, but certainly intentionally, Dr. Frans Wiering did me a great favor by insisting that ISMIR 2010 in Utrecht, The Netherlands, should be accompanied by a summer school. In the previous year, I had just moved back from Raleigh, NC, USA, to Cologne, Germany, and was eager to learn more about music content analysis to further improve my software beaTunes. What a great opportunity, right at my doorstep! And I was not disappointed. The summer school was a blast—academically and socially. During the day, Olivier Lartillot introduced us to MIRtoolbox [97], and at night we students hung out, cooked and ate together, played the guitar and sang. During these days, I met a lot of wonderful and smart people. Among them Nicola Montecchio, John Ashley Burgoyne, Sebastian Stober, Sebastian Ewert, Peter Grosche, Johan Pauwels, Anja Volk, Tom Collins, Bas de Haas, and Thomas Prätzlich who was about to

start working with Prof. Dr. Meinard Müller at the Max-Planck-Institute (MPI) in Saarbrücken. Thomas introduced me to Meinard, which—as it turned out—changed my life. Meinard pointed me to Prof. Dr. Michael Clausen's signal-processing lecture at the University of Bonn, which Prof. Clausen kindly let me attend, although I was not even enrolled there. In that same winter semester of 2010/11, I also participated in Meinard's MIR seminar in Saarbrücken, which he conveniently scheduled at a time slot that allowed me to travel there from Cologne and back by train in one day. In this seminar, I met Jonathan Driedger, who I still very much enjoy discussing MIR problems with over cappuccinos and second breakfasts at Café Franck. In the following years, Meinard generously kept supporting me in my part-time academic pursuits. He advised me on a Master thesis, even though he had nothing to do with the university I attended, gave me timely, constructive, and critical feedback on whatever idea I cooked up, and invited me to stay at his and Vlora's place in Erlangen whenever I visited. After many an ISMIR, I eventually officially joined his group as an external PhD student. Now, at the end of my PhD, it is time to say thanks to all the amazing people who made this journey possible and fun.

Obviously, without Meinard's generous and unwavering support, this would never have happened.

Thank you, Meinard, for all your help, the intense scientific discussions, constructive feedback, and insistence on concise, didactic scientific writing.

My thanks also go to current and former members of Meinard's group for their insightful support, advice, and cooperation in all things MIR. In no particular order these are Thomas Prätzlich, Jonathan Driedger, Peter Grosche, Stefan Balke, Christian Dittmar, Patricio López-Serrano, Vlora Arifi-Müller, Frank Zalkow, Michael Krause, Sebastian Barista Rosenzweig, and Christof Weiß. Working mostly from Cologne, I have not spent a lot of time at the International Audio Laboratories in Erlangen.[9] But I can say with certainty, that it is not only one of the most renowned institutions in the field of audio signal processing, but also a warm and welcoming place. I have greatly benefitted from all the support I have received there. In particular, I would like to thank the administrative staff: Stefan Turowski, Elke Weiland, Tracy Harris, and Day-See Riechmann, for their prompt and kind help in all administrative matters, as well as the AudioLabs professors Bernd Edler, Emanuël Habets, Frank Wefers, and Jürgen Herre for encouraging such an open and inspiring atmosphere.

During my PhD studies, I had the opportunity to do an internship at Amazon Music ML in San Francisco. I would like to thank Emile Richard, Emanuele Coviello, Ted Sandler, Ben London, Jerome Serrano, Doug Mason, Jonathan Pollack, Amina Shabbeer, Brandyn Kusenda, Warren Freitag, Blake Mason, Shereen Oraby, Garrett Bernstein, Fabian Mörchen, and Gert Lanckriet for this fun experience and the great collaborative work.

[9] The International Audio Laboratories Erlangen joint institution of the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) and Fraunhofer IIS.

For the final six months of my studies I was financed by the DFG project SeReCo (DFG MU 2686/10-1) for which I would like to thank the German Research Foundation.

Thanks also to Julián Urbano for contributing to and giving valuable tips for Chapter 6. Thanks to Dmitry Bogdanov, Alastair Porter, Sergio Oramas, and again Julián Urbano for including me in our MediaEval efforts. Special thanks also to Sebastian Stober for agreeing to review this thesis.

As one might suspect when reading the title, this thesis relies a lot on data. Therefore I would like to give special thanks to everybody, who generously provided theirs. Among them Christof Weiß, Junseong Park, and Vlora Arifi-Müller, who annotated the *Winterreise* dataset in Chapter 9, Graham Percival and George Tzanetakis [128], who provided third party results for Chapters 3 and 6, Masataka Goto for permitting redistribution of the AIST Annotations for the *RWC* Music Database [61, 62], Anssi Klapuri and Tuomas Virtanen for releasing beat annotations for the *Klapuri* dataset [86], Colin Raffel for providing *LMD*-related data [136], all participants of the experiment in Chapter 5, who tapped to the beats of EDM, and everybody, who answered questions about tempo estimation evaluation for the survey presented in Section 6.5.

Unfortunately, I cannot list everybody in the MIR research community. It is such a wonderful, inclusive, and inspiring group of people that I am certain other communities aspire to be like it. Thank you for letting me be a part of it. You have been my scientific family for the last ten years.

Which brings me to my actual family. Thank you for believing in me. Especially Franca, who challenged me to start this 🇳🇴, always had my back while I was working on it, and will be there to celebrate its conclusion with me.

Thank you for always being there.

To ●

2. Fundamentals

In this chapter, I introduce some digital signal processing fundamentals, closely following work by Meinard Müller [117]. Furthermore, I introduce a standard tempo estimation pipeline as basis for the following chapters, which is based on my work in [158] and [160].

Music can be represented in many different ways. Symbolically, for example as Music Instrument Digital Interface (MIDI) file. Visually, as printed sheet music (Figure 2.1a) or even as image of an engraving (Figure 2.1b). Or as audio, i.e., as representation of acoustic sound waves (Figure 2.2). In this thesis we will work exclusively with audio representations. For a detailed overview of other representations, we refer to [117, Chapter 1].

2.1. Audio Signal

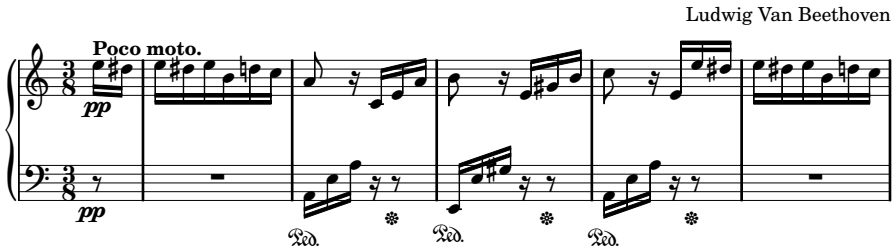
An audio signal can be described as a signal that transports acoustic information. It encodes the sonic waves that we perceive as sound and thus allows transmission, broadcast, storage, and analysis. Closely following [117, Section 2.2.1], we define a *continuous-time* (CT) signal to be a function $f : \mathbb{R} \to \mathbb{R}$, that assigns an amplitude value $f(t) \in \mathbb{R}$ to each point in time $t \in \mathbb{R}$. Such a CT-signal can represent an arbitrary sound, regardless of how loud, quiet, short, or long it is. Because of the inherent constraints of digital systems, we cannot process CT-signals using such systems directly. The signal first has to be discretized, i.e., converted into a finite representation. This is also called *digitization* and typically involves two steps: *sampling* and *quantization*.

The most common sampling method is *equidistant sampling*. Given a CT-signal $f : \mathbb{R} \to \mathbb{R}$ and a positive real number $T > 0$, one defines a function $x : \mathbb{Z} \to \mathbb{R}$ by setting

$$x(n) := f(n \cdot T) \tag{2.1}$$

for $n \in \mathbb{Z}$. Since x is only defined on a discrete set of points in time, it is also referred to as *discrete-time* (DT) signal. The value $x(n)$ for some sample index $n \in \mathbb{Z}$ is called a *sample* taken at time $t = n \cdot T$ of the analog signal f. This procedure is also known as T-*sampling*, where the

(a) Ludwig van Beethoven's *Für Elise*



(b) Second Delphic Hymn to Apollo

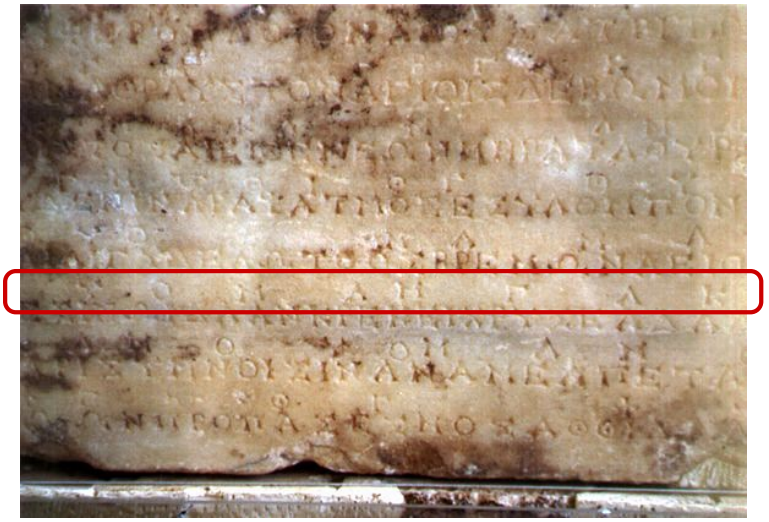


Figure 2.1.: Sheet music now and then. **(a)** The first few bars of Ludwig van Beethoven's *Für Elise* for piano. **(b)** Photograph of a stone at Delphi containing the second of the two Delphic Hymns to Apollo. The music notation is the line of symbols above the main, uninterrupted line of Greek lettering. © Public Domain.

number T is the *sampling period*. Its inverse

$$F_\mathrm{s} = 1/T \tag{2.2}$$

is called the DT-signal's *sampling rate*, measured in Hertz (Hz). Figure 2.2b shows individual samples of a short section of a recording of Ludwig van Beethoven's *Für Elise*, a classical piece for piano. In this particular example the sampling rate is 22,050 Hz. For high quality audio signals one typically uses the higher sampling rate $F_\mathrm{s} = 44{,}100\,\mathrm{Hz}$. The Nyquist-Shannon-Kotelnikov theorem states that this rate is high enough to perfectly represent sounds with frequencies up to

$$\Omega = \frac{F_\mathrm{s}}{2} = 22{,}050\,\mathrm{Hz}. \tag{2.3}$$

Ω is also referred to as *Nyquist frequency*. Because human hearing is limited to $20 - 20{,}000\,\mathrm{Hz}$, sampling rates much higher than 44,100 Hz cannot improve the sound quality noticeably. Despite this, studio technicians often use higher sampling rates of 96 kHz or even 192 kHz in order to

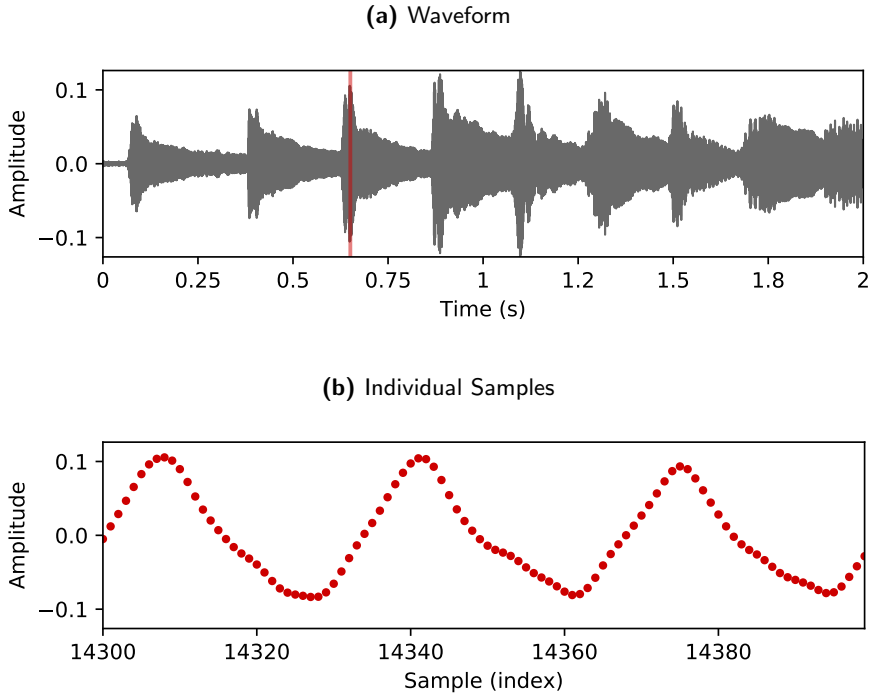


Figure 2.2.: Visualization of a discrete time audio signal. **(a)** Waveform of a performance of the first two seconds of Ludwig van Beethoven's *Für Elise*. Individual samples are not visible. An enlarged version of the section marked in red ■ is shown in **(b)**. At this magnification, individual samples are clearly visible.

create so-called high-definition audio recordings. The commercial Blu-ray format, for example, supports sampling rates of 48 kHz, 96 kHz, and 192 kHz for uncompressed audio.

Because a DT-signal is discrete in time, but not necessarily in amplitude, we need to quantize each real-valued sample, before we can work with it in a digital environment. Oftentimes, a *uniform quantizer* is used for this. It is characterized by the fact that continuous values are rounded to some precision level specified by a step size Δ. Given such a quantization step size, a uniform quantizer $Q : \mathbb{R} \to \Gamma$ for an amplitude $a \in \mathbb{R}$ and $\Gamma \subset \mathbb{R}$ can be defined by

$$Q(a) := \operatorname{sgn}(a) \cdot \Delta \cdot \left\lfloor \frac{|a|}{\Delta} + \frac{1}{2} \right\rfloor. \tag{2.4}$$

An example for a uniformly quantized audio signal is the audio CD format. Each audio CD sample has a storage size of 16 bits and can therefore encode $2^{16} = 65{,}536$ different values (steps). Assuming that we want to represent amplitudes between -1 and $+1$, this means that for CDs the quantization step size is $\Delta = 2^{-15}$. Higher bit-depths, like 24 bits/sample or even 32 bits/sample, are possible and used in recording studios, for high-definition audio formats, and even by music streaming services (e.g., Amazon Music Ultra HD). While 24 bits are a sensible choice during production to keep quantization noise at a minimum when sequentially applying

many audio effects, it is questionable whether humans are capable of distinguishing a downmixed 24-bit recording that has been converted to 16 bits from one that is left in the original 24-bit format [116].

Note that there are alternatives to uniform quantization. An example for non-uniform quantization is the μ-law companding[1] algorithm, used by low bandwidth telecommunication systems in North America and Japan.

Sampling and quantization allow us to discretize a CT-signal, and convert it to a DT-signal. To easily process such a DT-signal with the finite resources of a digital computing environment, we still need to address one last issue: CT-signals have infinite length by definition, which means that the corresponding DT-signals also have infinite length. Fortunately, digital audio recordings are finite and only have $L \in \mathbb{N}_{\geq 0}$ number of samples, which can be processed digitally. When modeling a DT-signal mathematically, we typically set $x(n) = 0$ for $n \in \mathbb{Z} \setminus [0 : L-1]$ where $[0 : L-1] := \{0, 1, \ldots, L-1\}$, thus we obtain a signal $x : \mathbb{Z} \to \mathbb{R}$. This lets us avoid boundary cases in the mathematical equations.

2.2. Discrete Fourier Transform

The Fourier transform, named after the French mathematician Jean-Baptiste Joseph Fourier ($\star$21 March 1768, †16 May 1830), is probably the most important mathematical tool in digital signal processing (DSP). In a nutshell, it allows us to decompose a signal into its constituent frequencies. Since in this thesis, we work exclusively with DT-signals, we will only introduce the discrete Fourier transform (DFT). Other definitions for Fourier transforms and comprehensive discussions of their properties can be found in [117, Chapter 2.3].

Given a DT-signal x of length L, the DFT X of x is defined as

$$X(k) = \sum_{n=0}^{L-1} x(n) \cdot e^{-\frac{2\pi i}{L} kn}, \tag{2.5}$$

for $k \in [0 : L-1]$. The complex Fourier coefficient $X(k)$ encodes the magnitude and the phase of the signal's sinusoidal component with frequency

$$F_{\text{coef}}(k) = \frac{k \cdot F_{\text{s}}}{L}. \tag{2.6}$$

The distance between frequency bins, i.e., the frequency resolution, is $\Delta_f = F_{\text{coef}}(1)$.

To illustrate the DFT's application, Figure 2.3 shows a magnitude spectrum, i.e., $|X|$, for the *Für Elise* recording introduced above. Clearly visible are the spikes at distinct frequencies

[1] Portmanteau of the words **com**pressing and ex**panding**.

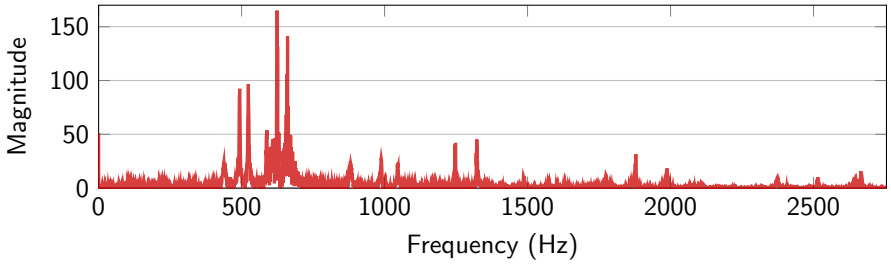


Figure 2.3.: Magnitude Fourier spectrum for the first two seconds of Beethoven's *Für Elise*

corresponding to the fundamental frequencies of the played notes and their harmonics. What we cannot see in this depiction is *when* the notes are played. This makes the unmodified DFT unsuitable as a basis for MIR tasks involving temporally local structures like chords.

A solution to this problem is the short-time Fourier transform (STFT) [52]. Its main idea is to apply the transform only to short parts of the signal, which are also known as frames. This effectively allows determining in which of these frames which frequencies occur. The STFT is defined as

$$X(t,k) = \sum_{n=-\frac{N}{2}}^{\frac{N}{2}-1} w(n) \cdot x(n+tH) \cdot e^{-\frac{2\pi i}{N}kn}, \tag{2.7}$$

where $t \in \mathbb{Z}$ is the frame index, $k \in [0 : N-1]$ is the frequency index, w is a window function centered around time position zero, N is the frame length, and H is the hopsize, i.e., the distance between two consecutive frames in samples. Similar to the DFT (Eq. 2.6), the frequency index k corresponds to the frequency band with center frequency

$$F_{\text{coef}}(k) = \frac{k \cdot F_{\text{s}}}{N}. \tag{2.8}$$

Since the STFT is time-dependent, its magnitude is referred to as spectrogram rather than simply as spectrum. It is denoted by

$$Y = |X|. \tag{2.9}$$

To illustrate, Figure 2.4a shows a magnitude spectrogram for our recording of *Für Elise* using $N = 1024$, $H = 512$, and a Hann window function.[2] Clearly visible are dark horizontal lines, which correspond to the fundamental frequency of the played notes, as well as very soft parallels referred to as overtones or harmonics. A common procedure to enhance these hardly visible overtones is *logarithmic compression* [117, Section 3.1.2.1]. It serves as an alternative to using the decibel scale and is achieved by applying a compression function $\Gamma_\gamma : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ to each of the spectrogram's values. Concretely, Γ_γ is defined as

$$\Gamma_\gamma(v) := \ln(1 + \gamma \cdot v) \tag{2.10}$$

[2]Named after the Austrian meteorologist Julius Ferdinand von Hann (⋆23 March 1839, †1 October 1921).

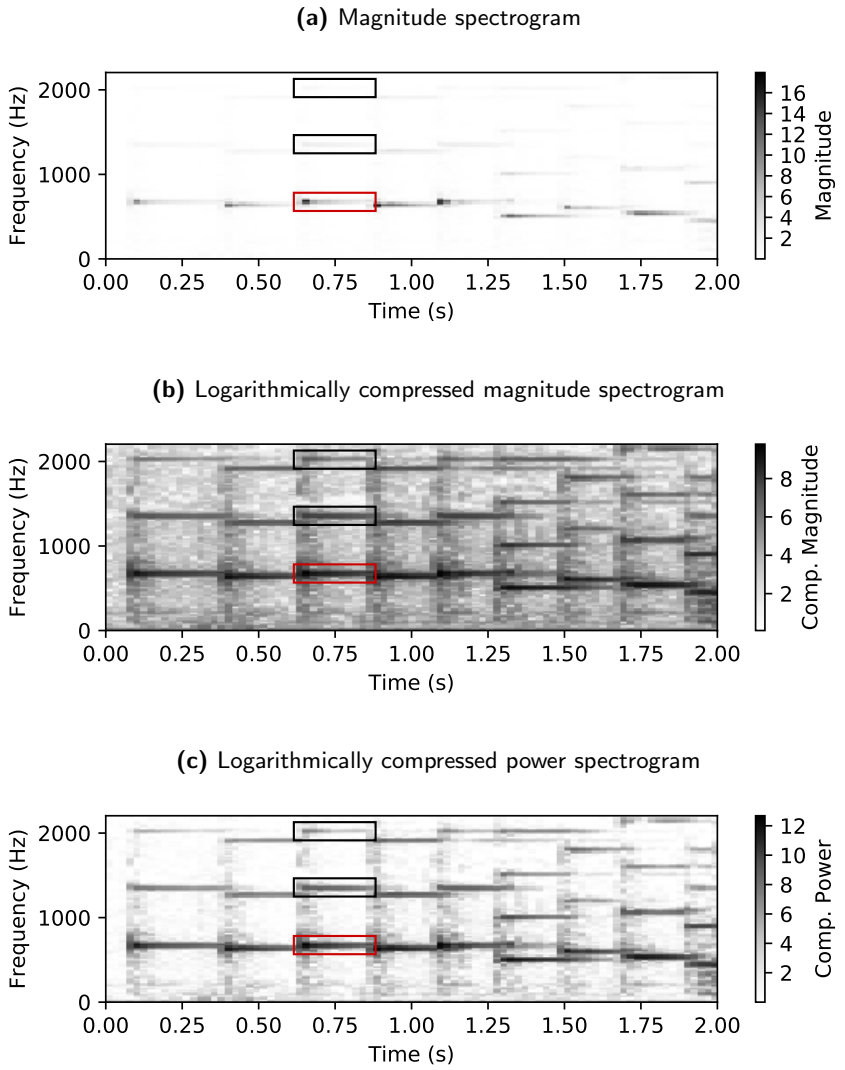


Figure 2.4.: Spectrograms for the first two seconds of Beethoven's *Für Elise*. The fundamental frequency of the third note is framed in red ■, its second and third harmonics are framed in black ■. **(a)** STFT-based magnitude spectrogram. **(b)** STFT-based logarithmically compressed magnitude spectrogram with $\gamma = 1000$. **(c)** STFT-based logarithmically compressed power spectrogram with $\gamma = 1000$.

with a positive constant $\gamma \in \mathbb{R}_{>0}$. Figure 2.4b shows a magnitude spectrogram that has been enhanced using logarithmic compression with $\gamma = 1000$. We can now clearly see the overtones. However, the spectrogram does not have a lot of contrast, i.e., instead of white we see a lot of gray in areas between the harmonics that should be low energy. Another method to obtain a contrast-rich spectrogram is to use the signal's power instead of its magnitude. We define the power spectrogram Y_{p} as the squared magnitude spectrogram:

$$Y_{\mathrm{p}} = Y^2 = |X|^2. \tag{2.11}$$

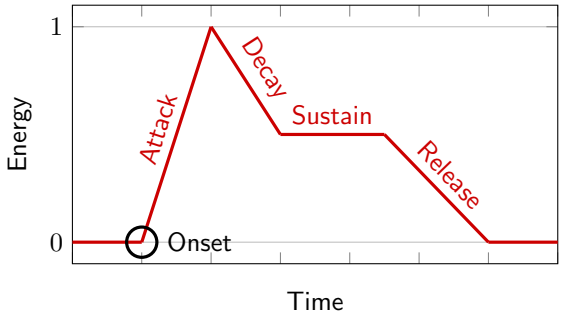


Figure 2.5.: Schematic of the ADSR envelope

As an example, Figure 2.4c shows a power spectrogram enhanced with logarithmic compression ($\gamma = 1000$). Compared to the enhanced magnitude spectrogram (Figure 2.4b), the main frequencies of the played notes are more easily recognizable. Note that this does not mean that using a power spectrogram is always the better choice. Choosing a different γ for the magnitude spectrogram may also have led to better results. In fact, it has been shown that a well-chosen γ-value has a considerable impact on MIR tasks like chord recognition [81].

2.3. Basic Tempo Estimation

Describing its speed, *tempo* is one of the most relevant descriptors for a piece of music. It can be defined as the number of times a listener "taps" a beat with his or her foot per time interval [150, 33]. As unit of measurement for tempo serves *beats per minute* (BPM). Automatic *tempo estimation* or *induction* is commonly used to estimate the general tempo of a musical piece. Unlike *beat tracking* [2, 60], tempo estimation does not attempt to determine the exact location of individual beats. Knowing the tempo can be useful for a wide variety of applications including music retrieval, score alignment, playlist generation, and DJ techniques like beatmixing. Because of its usefulness, the automatic extraction of tempo is a traditional task in MIR and has received a lot of attention over the years [60, 150, 201, 64].

In order to determine beats per minute of a musical piece from audio, it is intuitively useful to locate beats in the audio signal. But what is a beat? It can be described as a *perceived pulse position*. Being a construct of perception, it is related but not identical to an *onset event*, i.e., the start time of a note, drum beat or other musical event. Contrary to beats, onset events are physical phenomena that can be measured and detected in the audio signal. To define onsets, it is helpful to model what happens when a note is played, e.g., a single piano key. Figure 2.5 shows a schematic depiction of the ADSR model [129, p. 59], which describes the amplitude or energy envelope of the audio signal we can observe when playing a single note (see also [117, p. 27]). It consists of the attack (A), decay (D), sustain (S), and release (R) phases. The attack is usually characterized by a wide range of frequencies also described as noise or transients. During the

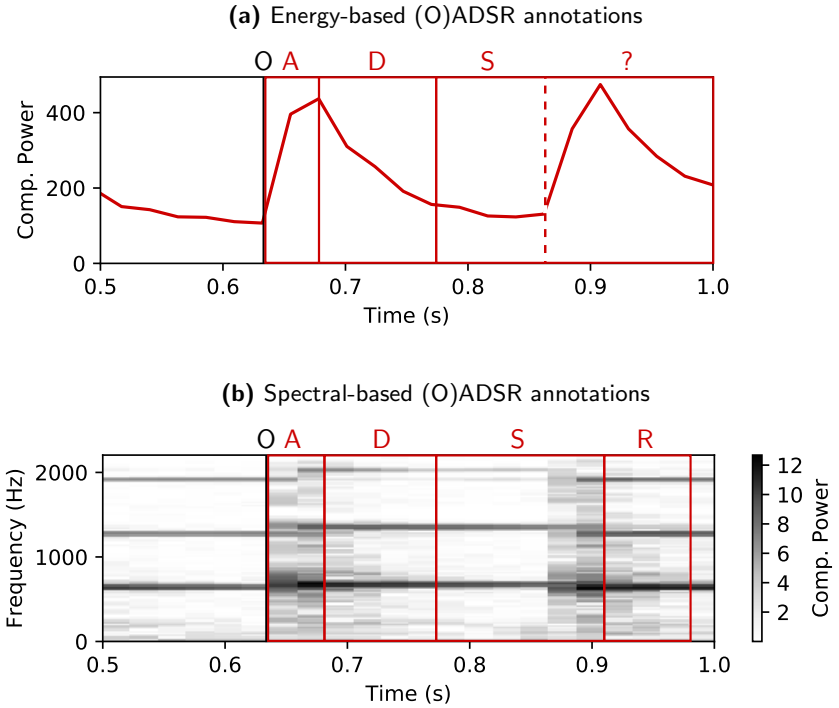


Figure 2.6.: Onset, attack, decay, sustain, and release for the third note of our running example. The sustain/release phases overlap with the beginning of the fourth note. **(a)** Energy-based (O)ADSR annotations. **(b)** Spectral-based (O)ADSR annotations.

decay phase the sound of the musical tone stabilizes. In the sustain phase we can clearly hear the tone, i.e., periodic patterns rather than noise, and the energy stays relatively constant. Finally, during the release phase, the energy gradually decreases to zero. The onset event for such an idealized note occurs right at the beginning of the attack. It is instantaneous and marks the earliest point in time at which the transient can be detected [117, p. 305].

Moving from the ideal world to the phyiscal world, Figure 2.6a shows the framewise energy, in this case the framewise sum of a logarithmically compressed Y_{p}, for an excerpt of our example. Based on this energy-curve, we manually added (O)ADSR-annotations, labeling the third note of our running example. While onset, attack, and the beginning of the decay are easily identifiable, we cannot say with certainty when the decay ends and the release begins. In fact, we cannot say anything about the release phase, because it is masked by the attack of the fourth note. Even in a *monophonic* music signal, like the given example, we have difficulties labeling all parts of the ADSR model correctly, when relying on just one energy-curve. The task becomes even more difficult for *polyphonic* music with simultaneously occurring sound events. Therefore, rather than being based on a single energy feature, onset detection for polyphonic music is often based on spectral features. They allow the detection of energy spikes in sub-bands of the signal as well as frequency dependent pre- and post-processing. To illustrate, Figure 2.6b shows manual (O)ADSR-annotations based on the logarithmically compressed Y_{p}. We can determine both

the end of the sustain phase and the release phase—not perfectly, but with more confidence. Fortunately, for onset detection and ultimately tempo estimation, we are mostly interested in finding onsets and not releases. To achieve this computationally, we use a spectral-based novelty function, also referred to as *spectral flux*. It is typically defined as the distance between consecutive spectral vectors (frames). When used for onset detection, it is also more generally referred to as *onset strength signal* (OSS). To derive a tempo value, the OSS function is analyzed for periodicities and the frequency of the dominant pulse is extracted in the hope that it is identical to the frequency of the perceived periodic pulses, i.e., the beats.

Thus, traditional methods for tempo estimation usually follow a three step procedure:

1. The audio signal is transformed to a suitable representation.
2. An OSS function is derived.
3. The frequency of the OSS's dominant pulse is determined and converted to BPM.

We now describe in detail how such a method may be implemented.

2.3.1. Representation

To construct our spectral novelty-based OSS function, we transform the signal into a suitable representation. We do so by first converting the signal to mono with a sample rate of $F_\mathrm{s} = 11{,}025\,\mathrm{Hz}$. The relatively low sample rate F_s has the advantage of reducing the amount of data and thus speeding up computation. We can do this without fearing decreased overall accuracy, because higher frequencies are known to be mostly irrelevant for tempo detection. After resampling, we compute the power spectrogram (Eq. 2.11) using a window length $N = 1024$ ($93\,\mathrm{ms}$), a hopsize $H = \frac{N}{2} = 512$, and a Hamming window[3] w. In order to enhance weak spectral components, we apply logarithmic compression with $\gamma = 1000$ to derive the positive logarithmic power spectrum

$$Y_\mathrm{ln}(t,k) = \Gamma_{1000}(Y_\mathrm{p}(t,k)) \tag{2.12}$$

Y_ln is the signal representation we base our OSS on. Its frame rate is $F_\mathrm{f} = \frac{F_\mathrm{s}}{H} \approx 21.53\,\mathrm{Hz}$.

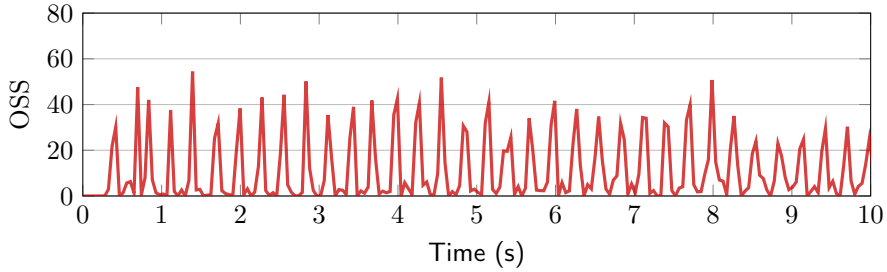


Figure 2.7.: OSS for a 10 s segment of Adele's *Rolling in the Deep*.

2.3.2. Onset Signal Strength

Similar to [85], we define the onset signal strength function (or novelty curve) OSS as the sum of the bandwise differences between the logarithmic powers $Y_{\ln}(t,k)$ and $Y_{\ln}(t-1,k)$ for those k where $30\,\text{Hz} \leq F_{\text{coef}}(k) \leq 720\,\text{Hz}$ and $Y(t,k) > \alpha Y(t-1,k)$:

$$\begin{aligned} I(t,k) &= \begin{cases} 1 & \text{if } Y(t,k) > \alpha Y(t-1,k) \\ & \text{and } 30 \leq F_{\text{coef}}(k) \leq 720, \\ 0 & \text{otherwise} \end{cases} \\ \text{OSS}(t) &= \sum_k \Big(Y_{\ln}(t,k) - Y_{\ln}(t-1,k) \Big) \cdot I(t,k) \end{aligned} \tag{2.13}$$

The indicator function I is designed to select only frequency bands in the desired frequency range, if the energy from one frame to the next increases at least by a factor α. For $\alpha = 1$ this is equivalent to band-wise *half-wave rectification*. For the method proposed here, we set $\alpha = 1.76$ to disregard small increases in loudness and thus reduce noise in the onset strength signal. Just like the used frequency range, α was found experimentally. A depiction of the resulting signal can be seen in Figure 2.7.

2.3.3. Dominant Pulse

There are different ways to determine the dominant pulse of an OSS, most importantly the autocorrelation function (ACF) [28, 40, 125, 170, 128], comb filter banks [86, 150, 6], and the discrete Fourier transform (DFT) [66, 198]. Furthermore, tempograms [197] and inter-onset interval (IOI) histograms [169] have been used.

[3] Named after the American mathematician Richard Wesley Hamming (⋆11 February 1915, †7 January 1998).

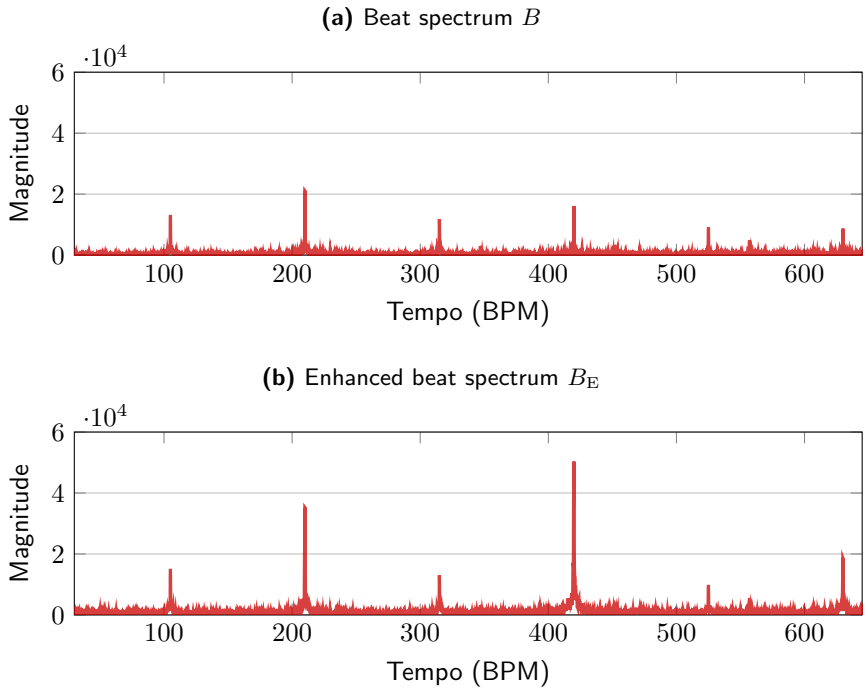


Figure 2.8.: **(a)** Regular and **(b)** enhanced beat spectrum for *Rolling in the Deep* by Adele.

For the method proposed here, we use the DFT to analyze the OSS with length $T = 8192$:[4]

$$B(k) = \left| \sum_{t=0}^{T-1} \mathrm{OSS}(t) \cdot e^{-\frac{2\pi i}{T} kt} \right|. \tag{2.14}$$

Given the frame rate F_{f} and the OSS length T, we can determine the frequency resolution $\Delta_f = \frac{F_{\mathrm{f}}}{T} \approx 0.00263\,\mathrm{Hz}$, which is equivalent to $\Delta_f \cdot 60 \approx 0.156\,\mathrm{BPM}$. The peaks of the resulting beat spectrum B represent the strength or salience of BPM values in the signal [48]. Naïvely, we could now determine the highest peak and derive a tempo value in BPM using

$$\mathrm{TMP} = F(\arg\max_k \ B(k)) \cdot 60. \tag{2.15}$$

As an example, Figure 2.8a shows B for the recording *Rolling in the Deep* by Adele, which has a tempo of 104 BPM. Clearly visible are peaks at 104 BPM and its multiples, with the highest peak at 208 BPM. These peaks at multiples of the fundamental frequency are also called *tempo harmonics*. A 104 BPM peak usually implies a 208 BPM peak, one at 312 BPM, and so

[4] To ensure this length, our implementation either zero-pads or truncates the OSS.

on [126, 68]. But which peak is the "right" one? To boost frequencies that are supported by harmonics, we derive an enhanced beat spectrum B_{E}:

$$B_{\mathrm{E}}(k) = \sum_{i=0}^{2} B\left(\lfloor k/2^i + 0.5 \rfloor\right). \tag{2.16}$$

Similar to computing a spectral sum [3] or an enhanced beat histogram [186], B_{E} incorporates harmonics by simply adding to each bin the magnitudes of the bins denoted by half and by a quarter of its own frequency—or, if not available—the closest available bin. We choose to use fractions instead of multiples for modeling harmonics and thus essentially model the fourth harmonic, not the first. This allows us to easily take advantage of the full DFT resolution without oversampling, as each first harmonic bin is mapped to four different fourth harmonic bins. Also, because most popular western music is in 4/4-time, and most octave errors are by factor 2 [201], we purposefully leave out the third harmonic. To calculate the estimated tempo $\mathrm{TMP}_{\mathrm{base}}$, we determine the highest value of B_{E}, divide its frequency by 4 to find the first harmonic, and finally convert its associated frequency to BPM:

$$\mathrm{TMP}_{\mathrm{base}} = F(\arg\max_k \ B_{\mathrm{E}}(k)) \cdot \frac{60}{4}. \tag{2.17}$$

Thus we arrive at a final tempo estimate $\mathrm{TMP}_{\mathrm{base}}$. In the following chapters we will refer to this system as `base`. For our example *Rolling in the Deep*, the enhanced beat spectrum is depicted in Figure 2.8b. The highest B_{E}-peak is at 415.26 BPM, which corresponds to $\mathrm{TMP}_{\mathrm{base}} = 103.82\,\mathrm{BPM}$, the correct result (with some rounding).

Note that using arg max on B_{E} is by far not the only possible approach. The decision for a tempo can been made in many different ways. Other strategies take advantage of simple heuristics, genre classification [168, 75], the discrete cosine transform (DCT) of inter-onset interval (IOI) histograms [43], or feature-based learning approaches like Gaussian mixture models (GMM) [127], support vector machines (SVM) [56, 128], k-nearest neighbor classification (k-NNC) [197], and neural networks [42]. Furthermore, we will explore two novel approaches to picking the correct tempo in Chapter 3.

2.4. Metrics

Throughout this thesis, we will evaluate the performance of different estimation systems. We will usually do so, using two established metrics: *accuracy 1* and *accuracy 2*, denoted as ACC_1 and ACC_2 [64]. ACC_1 is defined as the percentage of estimates that are within 4% of the ground truth tempo, and ACC_2, as the percentage of estimates that are within 4% of 1/3, 1/2, 2, or 3

times the ground truth tempo. Note that we refer to estimates that are integer multiples or fractions of the true value as *octave errors*. Attempting to avoid them is a recurring theme in this work (Chapters 3 and 4). While ACC_1 and ACC_2 are widely accepted metrics, they are not without alternative. An in-depth discussion about tempo estimation evaluation can be found in Chapter 6.

2.5. Datasets

There can be no tempo estimation system evaluation without datasets. Throughout this thesis, we will test our systems against popular and/or publicly available datasets like *GTzan* [185], *ACM Mirum* [127], or *Ballroom* [64]. But that is not the only function we need datasets for. The move from pure DSP towards machine learning and data-driven approaches is an arc encompassing this work. For the used methods we also require sizable training and validation datasets. Selecting and creating these datasets is therefore a recurring theme in this thesis.

All datasets are introduced and properly cited, when they are first used. To give an informal overview, we present an alphabetically ordered list here.

ACM Mirum	Tempo dataset. Used for testing in Chapters 3, 4, and 6.
Ballroom	Tempo dataset. Used for testing in Chapters 3.
Extended Ballroom	Tempo dataset used for training and validation in Chapters 3, 4, and 7.
GiantSteps Key	Key dataset. Used for testing Chapter 7.
GiantSteps Tempo †	Tempo dataset. Used for testing in Chapters 4, 5, 6, and 7.
GTzan Tempo	Tempo dataset. Used for testing in Chapters 3, 4, 6, and 7.
GTzan Key	Key dataset. Used for testing in Chapter 7.
Hainsworth	Tempo dataset. Used for testing in Chapters 3, 4, and 6.
ISMIR04 Songs	Tempo dataset. Used for testing in Chapters 3, 4, and 6
LMD Tempo *	Tempo dataset. Based on *LMD*. Used for training, validation, and testing in Chapters 4 and 7. Mentioned in Chapter 6.
LMD Key *	Key dataset. Based on *LMD*. Used for training, validation, and testing in Chapter 7.
Mazurka-5	Local tempo dataset. Based on beat annotations used in Chapter 8.

MIREX	Tempo dataset suitable for P-Score evaluation. Mentioned in Chapter 6.
MTG Key	Key dataset. Used for training and validation in Chapter 7.
MTG Tempo *	Tempo dataset. Based on *MTG Key*. Used for training and validation in Chapters 4 and 7.
RWC	Classic tempo datasets. Mentioned in Chapter 6.
SMC	Tempo dataset. Originally meant for beat-tracking, used for testing in Chapters 3, 4, and 6.
Winterreise	Local key dataset. Used in Chapter 9.

* Newly derived or created for this work.

† Revised annotations have been created in this work.

Page numbers to usages of specific datasets can also be found in the Index.

3. Tempo Octave Correction

In this chapter, we discuss how to address the so called *octave error* in tempo estimation using global features and linear regression as well as random forests. It is based on my work in [158] and [160].

Probably the biggest problem with global tempo estimation, apart from fluctuating tempi, is the so-called *octave error*, which occurs when estimates are integer multiples or fractions of the reference tempo. To understand how this problem has been approached so far, we recapitulate how a traditional tempo estimation system works:

1. Via an OSS or novelty curve, beat candidates are found.
2. One or more periodicities are extracted from the OSS.
3. A single periodicity is chosen as tempo.

As mentioned in Section 2.3.3, the final decision for a tempo—i.e., a BPM value—is often based on simple heuristics, genre classification, etc. Making the right choice, arguably fixes the octave error.

In the following two sections, we propose two novel approaches to improve this decision. The first is based on a global feature and linear regression, the second on algorithm-specific error estimation and correction using random forests [14].

3.1. Octave Correction with Global Features and Linear Regression

In this section we combine the basic method for tempo estimation from Section 2.3 with a simple method for tempo *octave estimation* based on a single global feature. We strongly emphasize simplicity, but are nevertheless able to show that this approach leads to convincing results. In Section 3.1.1 we start with deriving a global feature for rough tempo estimation, then, in Section 3.1.2, we describe the algorithm and how it estimates a dominant pulse, a tempo octave, and ultimately a BPM value. In Section 3.1.3, we evaluate the algorithm comparing it with other methods using a large dataset. Finally, in Section 3.1.4, we present our conclusions.

Reproducibility

See Appendix A.1 for an implementation of the method presented in this section.

3.1.1. Feature Selection

Inspired by [72] we collected a number of global song features via the consumer application beaTunes (Appendix D). For each song we retrieved Last.fm's most popular tags and selected those songs associated with the tags *slow* or *fast*, but not both. For the genres Rock, Pop, Jazz, Alternative, Industrial, Heavy Metal, Soul, and Dance the dataset contained 8517 songs, 1296 (15.2%) of which labeled as *fast*. Because of the obvious imbalance between slow and fast songs also observed in [103], we grouped the data by genre, each group with an equal number of songs labeled as *slow* or *fast*. We did so by randomly removing songs of the overrepresented tempo class. We then classified the songs into *slow* and *fast* using one feature at a time. This turned out to be most successful for the feature *mean spectral novelty* (SNM).

SNM_L is calculated by building a self-similarity matrix S, using the cosine of the angle between two spectral vectors $Y(t)$ (as defined in Eq. 2.11 with parameters from Section 2.3.1) as similarity score. The novelty score SN_L is calculated with a square Gaussian checkerboard kernel C_L with length $L = 64$ [47]. Considering the given sample rate and window overlap, this is equivalent to a 2.97 s kernel. We choose to normalize the score SN_L by dividing by the sum of the absolute values of all kernel elements (Eq. 3.1). To obtain the mean SNM_L, we average $\mathrm{SN}_L(t)$ for $t \in \left[\frac{L}{2} : T - \frac{L}{2}\right]$.

$$\mathrm{SN}_L(t) = \frac{\sum_{m=-\frac{L}{2}}^{\frac{L}{2}-1} \sum_{n=-\frac{L}{2}}^{\frac{L}{2}-1} C_L(m,n) \cdot S(t+m, t+n)}{\sum_{m=-\frac{L}{2}}^{\frac{L}{2}-1} \sum_{n=-\frac{L}{2}}^{\frac{L}{2}-1} |C_L(m,n)|} \tag{3.1}$$

To illustrate SNM_L, Figure 3.1 shows spectral novelty values computed with a 3.81 s long kernel for a slow (Figure 3.1a) and a fast song (Figure 3.1b). It appears that the fast song is noisier over the length of the kernel and therefore its novelty scores are lower than the slow song's.

With an overall correct slow/fast classification rate of 85% (Table 3.1), SNM_{64} can obviously help estimating the perceived tempo of music. But since our dataset did only contain few values for genres other than Rock, Pop, and Jazz, the validity of this statement is clearly limited to those genres. Also, a perceived slow tempo does not guarantee a certain BPM range. For example, a Viennese Waltz may be perceived as slow, but its tempo is typically 174 to 180 BPM.

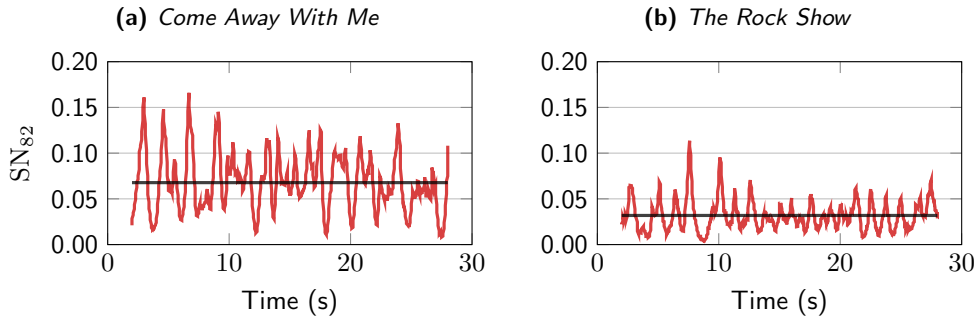


Figure 3.1.: Spectral novelty SN_{82} of excerpts of Norah Jones' slow *Come Away With Me*, and of Blink-182's fast *The Rock Show*. The mean is shown as horizontal line.

Genre	Songs	Correct	SNM_{64} Threshold
Rock	1706	87.86%	0.034
Pop	262	88.17%	0.036
Jazz	330	86.06%	0.040
Alternative	72	81.94%	0.034
Industrial	68	80.88%	0.023
Heavy Metal	54	83.33%	0.032
Soul	36	77.78%	0.032
Dance	36	83.33%	0.027
All	2564	85.02%	0.036

Table 3.1.: Classification results for different genres using SNM_{64} and a threshold that was calculated using a decision tree. Last.fm labels *slow* and *fast* served as ground truth.

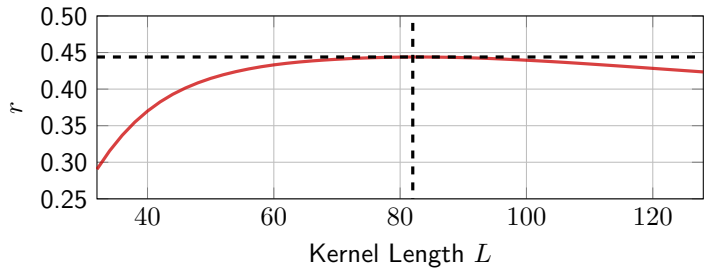


Figure 3.2.: Strength of linear relationship between SNM_L and ground truth BPM of songs in *GTzan* measured with correlation coefficient r depending on kernel length L. A maximum of $r = 0.44$ is reached at $L = 82$.

Furthermore, the kernel length $L = 64$ was chosen before the relationship to the perceived tempo class was discovered.

Therefore we investigated the relationship between SNM and the ground truth of the tempo annotated *GTzan* genres dataset [185]. *GTzan* consists of 1000 songs, 100 from each genre. As a simple measure of relationship we computed Pearson's correlation coefficient r between ground truth BPM and SNM_L for the kernel lengths 32 to 128 and found the maximum of $r = 0.44$ at

Genre	r	MAE	RMSE
Blues	0.22	22.56	27.50
Classical	*0.16*	22.13	27.92
Country	0.42	17.18	21.46
Disco	0.30	12.33	17.35
Hiphop	0.34	7.51	11.00
Jazz	0.60	15.35	19.44
Metal	0.29	20.95	24.44
Pop	**0.61**	12.37	15.32
Reggae	0.22	13.41	17.48
Rock	0.23	20.54	24.25
All	0.44	17.50	21.86

Table 3.2.: Genre-specific correlation r between *GTzan* ground truth BPM and SNM_{82} along with mean absolute errors (MAE) and root mean squared errors (RMSE) in BPM for genre-specific linear regressions.

$L = 82$, covering 3.81 s (Figure 3.2). The low correlation coefficient indicates that this is not a strong linear relationship—at least not for the whole collection. In fact, the results in Table 3.2 suggest, that the relationship between SNM and BPM is genre-dependent. With $r = 0.61$ and 0.60 it is very promising for Pop and Jazz, and less so for Blues, Classical, or Reggae with $r = 0.22$, 0.16, and 0.22. But considering that we just want to estimate the tempo octave rather than the precise BPM, mean absolute errors (MAE) of less than 23 BPM and root mean squared errors (RMSE) of less than 28 BPM for genre-specific linear regressions (Table 3.2), make a linear model suitable enough for our purposes.

3.1.2. Algorithm

We are dividing the problem of tempo estimation into three separate tasks:

1. Compute a dominant pulse while largely ignoring the tempo octave.
2. Determine a rough estimate of the perceived tempo and thus the tempo octave.
3. Combine the two results in a meaningful way.

To solve the 1st task we re-use the basic tempo estimation method from Chapter 2, in particular Equation (2.17). The 2nd and 3rd task are solved as follows.

3.1.2.1. Estimating the Tempo Octave

Since we have found a somewhat linear relationship between SNM and BPM, all we have to do to estimate the rough tempo $\mathrm{TMP}_{\mathrm{oct}}$, is to find the kernel length L that leads to SNM_L values that correlate well with a training ground truth and then perform a linear regression.

(a) ACC_1

Dataset	Size	gflr	base	marsyas	gkiokas	zplane	echonest	ibt	qm_vamp
ACM Mirum	1410	**76.1**	70.3-	71.6-	72.6-	70.2-	73.8	63.0-	63.9-
ISMIR04 Songs	465	**73.1**	61.9-	58.5-	56.8-	56.1-	57.0-	46.7-	42.8-
Ballroom	698	66.3	65.5	63.3	62.9-	**66.5**	56.6-	63.8	65.3
Hainsworth	222	70.7	68.9	66.7	64.4	69.8	66.7	**72.5**	**72.5**
GTzan	1000	**77.0**	69.2-	74.6	71.1-	68.5-	67.8-	60.4-	57.9-
Dataset Avg	759	**72.7**	67.2	66.9	65.6	66.2	64.4	61.3	60.5
Combined	3795	**73.9**	68.0-	69.0-	68.0-	67.3-	66.6-	61.0-	60.5-

(b) ACC_2

Dataset	Size	gflr	base	marsyas	gkiokas	zplane	echonest	ibt	qm_vamp
ACM Mirum	1410	96.0	96.5	96.0	**97.8+**	93.8-	92.8-	92.8-	92.3-
ISMIR04 Songs	465	91.8	**92.0**	83.2-	90.8	82.4-	78.5-	76.8-	77.9-
Ballroom	698	96.3	97.6	91.6-	**97.7**	94.4	86.1-	89.8-	87.8-
Hainsworth	222	**86.9**	84.7	82.0	84.7	82.4	85.6	82.0	83.8
GTzan	1000	92.6	92.6	90.8	**92.9**	88.6-	86.7-	86.2-	85.8-
Dataset Avg	759	92.7	92.7	88.7	**92.8**	88.3	85.9	85.5	85.5
Combined	3795	94.1	94.4	91.4-	**94.9**	90.5-	87.8-	87.9-	87.5-

Table 3.3.: Tempo results for **(a)** ACC_1 and **(b)** ACC_2 in percent. The + and − signs indicate a statistically significant difference between an algorithm and `gflr`. Bold numbers mark the best-performing algorithm(s) for a dataset. *Dataset Avg* is the mean of the algorithms' results for each dataset. *Combined* is the accuracy over all datasets combined.

Because the time complexity of computing SNM_L is quadratic, we prefer smaller L. We found that the value determined for *GTzan*, $L = 82$, represents a good tradeoff between correlation and runtime behavior. To compute the linear regression with WEKA [71], we use the combined five datasets [103, 127, 64, 70, 185] also used in [186], but a ground truth improved by Percival. Thus, we arrive at Equation (3.2) for the rough perceived tempo estimate TMP_{oct}:

$$TMP_{oct} = -851.144 \cdot SNM_{82} + 137.623 \tag{3.2}$$

3.1.2.2. Combining Tempo and Tempo Octave

As mentioned above, most octave errors are by factor 2 [201], therefore, to compute the final tempo TMP_{gflr}[1], we divide/multiply TMP_{base} (defined in Eq. 2.17) with/by 2 until it is closest to TMP_{oct}. Formally:

$$TMP_{gflr} = 2^i \cdot TMP_{base} \tag{3.3}$$

[1]Named after the estimation method used in this section, **G**lobal **F**eature-based **L**inear **R**egression.

Algorithm	×1/2	×2	×1/3	×3	other
gflr	11.7	8.5	0.0	0.1	5.9
base	10.8	14.6	0.4	0.7	5.6
marsyas	11.7	9.6	0.7	0.5	8.6
gkiokas	10.4	14.6	1.3	0.5	5.1
zplane	8.9	13.9	0.0	0.4	9.5
echonest	8.2	12.2	0.6	0.3	12.2
ibt	6.8	19.6	0.0	0.6	12.1
qm_vamp	4.7	21.4	0.0	0.8	12.5

Table 3.4.: Percentages of the reported results for the combined datasets that are equal to X times the ground truth (4% tolerance).

with $i \in \mathbb{Z}$ such that

$$0.75 \cdot \mathrm{TMP}_{\mathrm{oct}} < 2^i \cdot \mathrm{TMP}_{\mathrm{base}} < 1.5 \cdot \mathrm{TMP}_{\mathrm{oct}}. \tag{3.4}$$

3.1.3. Evaluation

We evaluate the proposed method gflr with the best performing algorithms [57, 122, 121, 28] [2,3] discussed in [186] and the baseline method base from Chapter 2 using the same five datasets, with the aforementioned improved ground truth.[4] As measures of accuracy we employ ACC_1 and ACC_2 (Section 2.4). We test for statistical significance with McNemar's test and a significance value of $p < 0.01$ [201]. Table 3.3 shows the results computed with data kindly made available by Tzanetakis and Percival. For ACC_1, gflr performs either as well or better, often significantly, than all other algorithms. In particular, ACC_1 for the combined datasets is with 73.9% significantly higher. For ACC_2, gflr reaches values similar to the best performing algorithm gkiokas. With an ACC_2 of 94.1% for the combined datasets, gflr performs significantly better than all other algorithms except the much more complex gkiokas and base.

When analyzing why gflr scored better in ACC_1 than its closest contenders, it becomes clear, that marsyas [186] scores lower, because of slightly more factor 2 errors (1.1 percentage points, pp) and 2.7 pp more other errors (Table 3.4). This hints at room for improvements in pulse estimation. gkiokas [57] on the other hand, has 6.1 pp more factor 2 errors, hinting at possible improvements in octave correction—something they addressed for ballroom genres in [56]. Compared to the baseline method base, gflr scores significantly higher in ACC_1, mostly because of much fewer factor 2 errors (6.1 pp) and hardly any factor 3 or 1/3 errors. This more than makes up for slight increases in factor 1/2 and other errors.

[2] zplane [aufTAKT] V3, http://www.beat-tracking.com/
[3] Dev build v3.2, http://developer.echonest.com/
[4] Therefore the results are not identical to [186].

3.1.4. Conclusions

We have presented a very simple and effective tempo estimation algorithm that combines standard pulse detection with a continuous tempo octave estimation using the single global feature SNM_{82}. Broad experimental evaluation shows that our method performs as well or significantly better than other state-of-the-art algorithms for a large, mixed-genre dataset. This indicates that perceptual global features can play an important role in tempo octave estimation.

3.2. Error Classification and Correction with Random Forests

As a second novel method for tempo octave error correction, we propose a supervised learning approach, which re-frames error correction as a classification problem. We are able to demonstrate that the proposed method performs better or as well as state-of-the-art algorithms when combined with the simple tempo estimation method from Section 2.3. Furthermore, because our error correction approach can be trained for any tempo detection method, we are able to show improvements in ACC_1 for previously published algorithms via post-processing.

The remainder of this section is structured as follows. In Section 3.2.1, we describe the tempo estimation algorithm, test datasets, measures, and investigate common estimation error classes. Then, in Section 3.2.2, we explain our post-processing procedure, which corrects tempo estimates using supervised learning based on a small number of audio features. In Section 3.2.3 we evaluate the proposed features, and then compare our results with those from other methods. Finally, in Section 3.2.4, we present our conclusions.

Reproducibility

See Appendix A.2 for an implementation of the method presented in this section.

3.2.1. Tempo Estimation

To lay the groundwork for our error correction method, we first describe the used tempo estimation algorithm, then introduce several test datasets and discuss common pitfalls. In Section 3.2.1.5, we introduce performance metrics and describe observed errors.

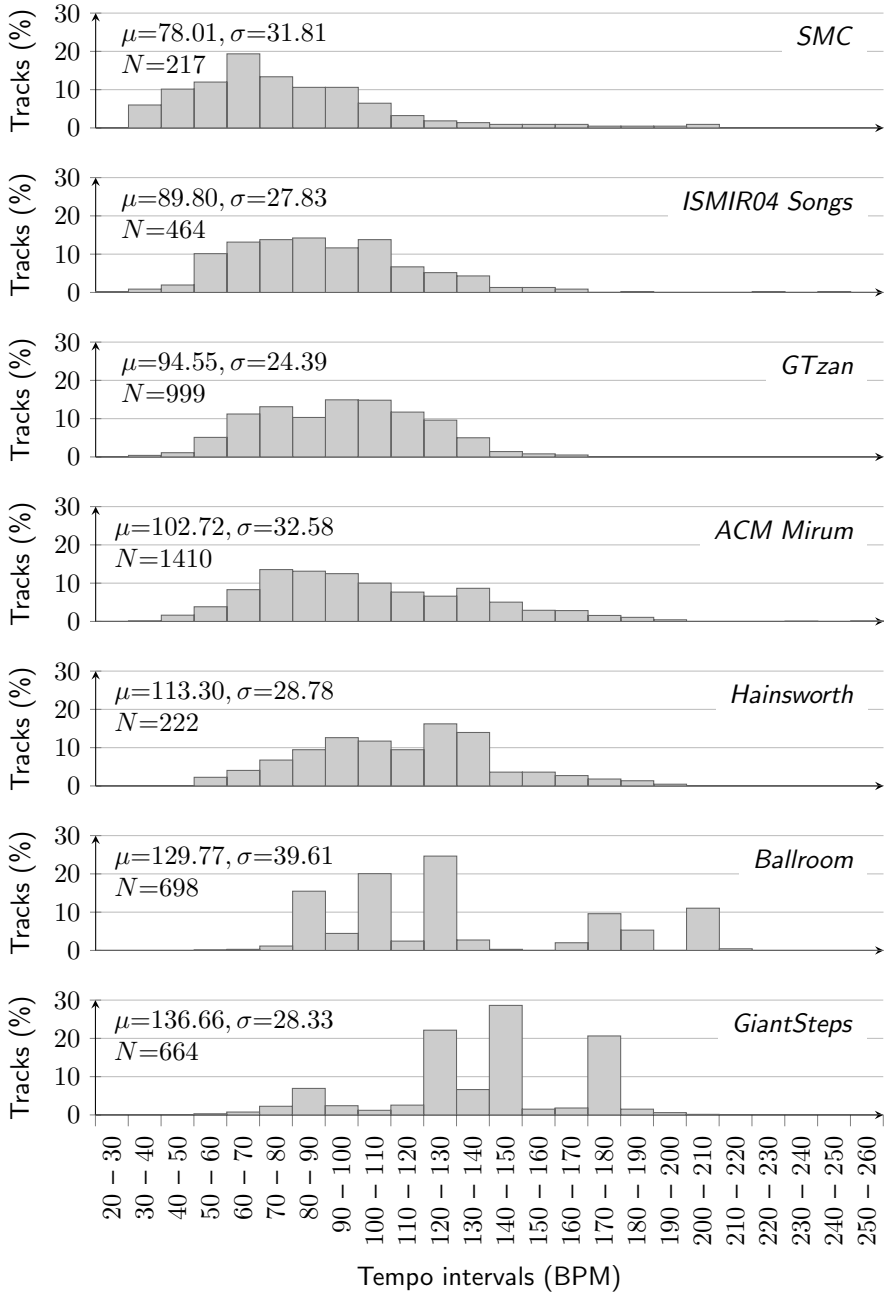


Figure 3.3.: Tempo distributions for the test datasets.

3.2.1.1. Algorithm

To estimate the dominant pulse we follow the approach taken in Section 2.3, which is similar to [186, 128]. To ensure meaningful results for most kinds of Western music, we constrain $\mathrm{TMP}_{\mathrm{base}}$, defined in Eq. 2.17, to $[40, 250]$ by halving or doubling its value, if necessary.[5]

[5]Which is why `base`-results are not identical to Table 3.3.

Dataset	Sweet Octave (BPM)	Coverage (%)	90% (BPM)	95% (BPM)
ACM Mirum	69 – 138	72.8	50 – 152	50 – 170
Ballroom	71 – 142	71.1	84 – 204	82 – 204
Hainsworth	79 – 158	82.4	58 – 150	57 – 167
GTzan	66 – 132	80.9	55 – 130	52 – 138
ISMIR04 Songs	59 – 118	74.1	48 – 131	36 – 136
SMC	51 – 102	68.7	32 – 115	32 – 143
Combined	69 – 138	72.9	40 – 150	50 – 180
GiantSteps	91 – 182	88.1	85 – 175	80 – 180

Table 3.5.: Sweet octaves and their respective coverage in percent for the test datasets (left). Shortest BPM intervals required to achieve a test set coverage of 90% or 95% (right).

3.2.1.2. Test Datasets

It has become customary to benchmark tempo estimation methods with results reported for a small set of well known datasets. These are *ACM Mirum* [103, 127], *Ballroom* [64], *GTzan* [185], *Hainsworth* [70], *ISMIR04 Songs* [64], and *SMC* [74]. The latter was specifically designed to be difficult for beat trackers. Where applicable, we used the corrected annotations from [128]. We refer to the union of these six datasets as the *Combined* dataset. Additionally, we test against the recently published *GiantSteps* dataset for electronic dance music (EDM) [88]. It is not included in *Combined* to allow direct comparisons with older literature.

Not surprisingly, all mentioned datasets differ in their composition (Figure 3.3). The mean tempo ranges from 78 BPM (*SMC*) to 137 BPM (*GiantSteps*) and the standard deviation spans from 24 (*GTzan*) to 40 (*Ballroom*). Furthermore, the tempo distributions of *Ballroom* and *GiantSteps* contain some distinct spikes, while the other datasets more closely resemble normal distributions. None of the datasets have uniformly distributed tempi.

3.2.1.3. Octave Bias

If a dataset's tempo distribution is not uniform and most values fall into a relatively small interval, constraining results to this interval may lead to fewer octave errors. We call deliberately choosing such an interval *octave bias*.

To illustrate this, assume an algorithm for the *GiantSteps* dataset with 50% ACC_1, but 100% ACC_2. Further assume that all errors are by a factor of 2 or 1/2. 88.1% of all tempi in *GiantSteps* happen to be in $[91, 182)$. If we constrained results to this interval by halving and doubling, ACC_1 would increase from 50% to 88.1%.

Each described dataset has such a *sweet octave*, i.e., a tempo interval $[j, 2j)$ that contains more of the dataset's songs than any other octave (Table 3.5). In the absence of a uniform test set, it is

Dataset	E_0	E_1	E_2	$E_{1/2}$	E_3	$E_{1/3}$	E_4	$E_{1/4}$	$E_{3/2}$	$E_{2/3}$	$E_{5/4}$	$E_{4/5}$	$E_{4/3}$	$E_{3/4}$
ACM Mirum	0.7	73.5	16.0	7.0	0.8	0.0	0.1	0.0	1.8	0.1	0.0	0.0	0.0	0.1
Ballroom	0.4	64.3	1.0	29.7	0.0	1.1	0.0	0.7	0.1	2.3	0.0	0.0	0.0	0.3
Hainsworth	8.6	64.4	1.4	19.4	0.0	0.5	0.0	0.0	1.8	1.8	0.9	0.5	0.5	0.5
GTzan	4.0	72.2	15.7	5.1	0.4	0.1	0.0	0.0	1.0	0.4	0.1	0.4	0.5	0.1
ISMIR04 Songs	4.3	64.7	19.4	6.3	1.1	0.2	0.0	0.0	2.4	0.4	0.4	0.2	0.2	0.4
SMC	29.0	37.8	6.0	10.6	0.0	2.3	0.0	0.0	5.1	2.3	0.5	2.3	0.9	3.2
Combined	3.9	68.1	12.3	11.2	0.5	0.4	0.0	0.1	1.5	0.8	0.1	0.3	0.2	0.4
GiantSteps	7.1	63.1	4.1	21.5	0.0	0.0	0.0	0.0	0.6	0.3	0.8	0.9	0.5	1.2

Table 3.6.: Error class distribution for tempo estimates $\mathrm{TMP}_{\mathrm{base}}$ (given in BPM) for different datasets in percent.

therefore important to test the same algorithm against datasets with different sweet octaves, thus revealing the effects of octave bias. On the positive side, a specialized or genre-aware algorithm may benefit from exploiting knowledge about the test dataset (e.g., EDM-specific tempi [75]). Additionally to sweet octaves, Table 3.5 lists the shortest BPM intervals required to achieve a certain test set coverage. For example, to cover 90% of the tempi in the *ACM Mirum* test set, one only needs to look at the interval $[50, 152]$ and not at the considerably larger interval $[37, 257]$ required for full coverage.

3.2.1.4. Genre Bias

While octave bias describes how algorithms can exploit constraining results to certain tempo intervals, *genre bias* describes a technique for algorithms to constrain their output to a relatively small set of distinct tempi that are characteristic for the genres in the dataset.

A good example for this is the *Ballroom* dataset. Even though the dataset contains 698 songs, only 63 different tempi occur. Assuming an unbiased algorithm with integer precision is constrained to the $[40, 250]$ BPM interval, it solves a task equivalent to choosing one out of 210 classes. An algorithm trained on the *Ballroom* dataset using k-fold cross validation "knows" that there are only 63 classes and therefore has a considerably easier task to solve.

3.2.1.5. Measures

As mentioned above, tempo estimation algorithms are usually evaluated with the metrics ACC_1 and ACC_2 (Section 2.4). Because we aim to correct estimation errors, we need to test our tempo estimation method against the test datasets and record not just accuracies, but also the kinds of errors. To do so, we define the error classes E_2, E_3, E_4, $E_{3/2}$, $E_{5/4}$, $E_{4/3}$ and their reciprocals with the index indicating the error factor. Just like ACC_1 and ACC_2 we allow a 4% tolerance. Since not all estimates are wrong and some errors are not covered by the mentioned classes, we

define E_1 for correct estimates (equivalent to ACC_1) and E_0 for errors not described otherwise. This leads to a total of 14 classes forming the label set $\mathbb{E} := \{E_0, E_1, \ldots\}$. Table 3.6 shows the distribution of estimated tempi over $\mathbb{E}$ for the test datasets using the tempo estimation method from Section 3.2.1.1. For *Combined*, 12.3% of all tempi are in E_2, while 11.2% are in $E_{1/2}$. Only 3.9% of all estimated values cannot be explained by one of the defined factors and thus are collected under the label E_0. This implies an upper bound of 96.1% ACC_1 for any error correction scheme based on $\mathbb{E}$ w.r.t. the *Combined* datasets.

3.2.2. Tempo Error Correction

As we have seen, most wrong tempo estimates are off by a limited number of factors. Therefore the correction of $\mathrm{TMP}_{\mathrm{base}}$ can be re-framed as a classification problem, which is solvable using supervised machine learning. Knowing the error class for an estimate then allows us to calculate the true tempo. In the following subsections we describe the features used for classification, the training dataset, and the tempo correction procedure.

3.2.2.1. Features

In order to keep the algorithm simple, we use as features only $\mathrm{TMP}_{\mathrm{base}}$ and a very small set of audio features. While not attempting to specifically model genres, the features we use aim at characterizing rhythm, tonality, and beat intensity. Combined, we expect them to capture essential information about a musical piece.

Log Beat Spectrum The tempi corresponding to the most common estimation classes $E_{1/2}$, E_1, and E_2 fall onto a logarithmic scale. To mirror this, we use a *logarithmic beat spectrum* (LBS) to describe the different periodicities in the signal. LBS is computed by resampling/interpolating the beat spectrum B (Section 2.3.3, Eq. 2.14) into 10 logarithmically spaced bands representing tempi ranging from 40 to 500 BPM. Subsequently it is normalized so that its highest value is 1.

While LBS provides a more complete picture of the periodicities than just the dominant tempo $\mathrm{TMP}_{\mathrm{base}}$, it does not add any information about the frequency bands these periodicities stem from. As a second modification to B, we create different versions of LBS based on the five slightly overlapping bands $[30, 110]$, $[100, 220]$, $[200, 440]$, $[400, 880]$ and $[800, 1600]$ Hz. Combined, these spectra form the *multiband logarithmic beat spectrum* ($\mathrm{LBS}_{\mathrm{M}}$). For a given song, $\mathrm{LBS}_{\mathrm{M}}$ consists of $5 \times 10 = 50$ features.

Spectral Flatness To represent tonality we use *spectral flatness* (SF), also known as Wiener entropy. It is defined as the ratio between the geometric and the arithmetic mean of the power spectrum (Eq. 2.11):

$$\mathrm{SF}(t) = \frac{\left(\prod_{k=0}^{K-1} Y_{\mathrm{p}}(t,k)\right)^{\frac{1}{K}}}{\frac{1}{K}\sum_{k=0}^{K-1} Y_{\mathrm{p}}(t,k)} \tag{3.5}$$

To determine $\mathrm{SF}(t)$, we re-use the power spectra $Y_{\mathrm{p}}(t,k)$ defined in Section 2.3.1, Eq. 2.11. For increased robustness against low sample rates, we limit k to $F_{\mathrm{coef}}(k) \in [30, 3000]$. As the two features for a given song we use both the mean and the variance of all its $\mathrm{SF}(t)$.

Temporal Flatness To represent onset intensity we use a feature called *temporal flatness* (TF). Instead of calculating the Wiener entropy along the frequency axis of $Y_{\mathrm{p}}(t,k)$, as we did for SF, we calculate it over a window of length ℓ along the time axis:

$$\mathrm{TF}(t,\ell,k) = \frac{\left(\prod_{i=0}^{\ell-1} Y_{\mathrm{p}}(t+i,k)\right)^{\frac{1}{\ell}}}{\frac{1}{\ell}\sum_{i=0}^{\ell-1} Y_{\mathrm{p}}(t+i,k)} \tag{3.6}$$

To compute TF values, we again re-use Y_{p} and limit k to $F_{\mathrm{coef}}(k) \in [30, 3000]$. Y_{p} is split into non-overlapping windows with length $\ell = 100$. For each bin k in a given window we compute TF. We then calculate the average $\mathrm{TF}_{\mathrm{W}}(t,\ell)$ over all k. As the two features for a song we use the mean and the variance of all its $\mathrm{TF}_{\mathrm{W}}(t,\ell)$ values.

3.2.2.2. Training Dataset

To avoid learning the test datasets, we use a dataset for training the classifier that has been created separately. *Train* is the union of an annotated, private music collection and the Extended *Ballroom* dataset [105] minus the 354 songs also occurring in the regular *Ballroom* set. Genre labels are available for 71% of the recordings. The genre as well as the tempo distribution are shown in Figure 3.4.

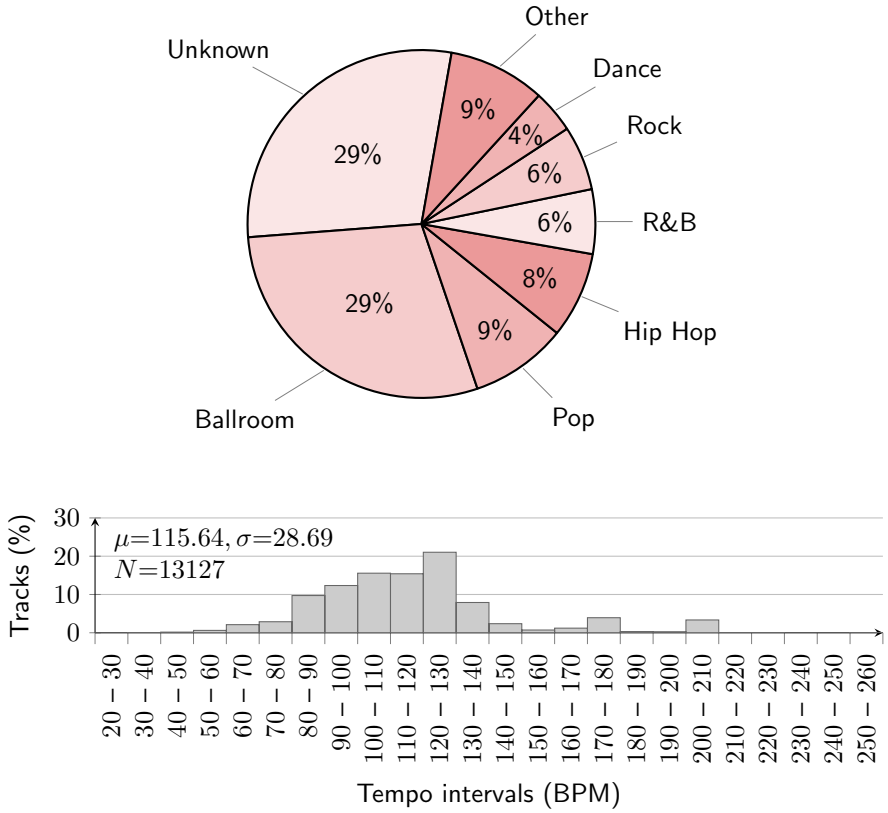


Figure 3.4.: Distribution of genres and tempi for *Train*.

3.2.2.3. Correcting the Tempo Estimate

Differences between the training dataset's true tempo values and the estimated tempo values let us derive error class labels. With those and the proposed features, we can train a classifier. Using the classifier, we are then able to predict an estimated error class $E \in \mathbb{E}$ for any song for which we also have features and a tempo estimate. Note that this estimate does not have to stem from our own algorithm introduced in Chapter 3.2.1.1. One main idea of this method, indeed, is that the classification model is algorithm-specific. In other words, the classifier must be trained for each tempo estimation algorithm. Once trained, it can be used to correct octave errors inherent to the given tempo estimation algorithm. For the prediction process itself, we use a random forest [14] with 300 trees and a maximum depth of 25.

Given the estimated tempo $\mathrm{TMP}_{\mathrm{base}}$ and the predicted error E the calculation of the corrected tempo $\mathrm{TMP}_{\mathrm{oerf}}$ is straight forward:

Features	ACC_1	ACC_2
`base`	68.10-	92.54
LBS	74.89-	92.54
LBS + SF	75.66	92.37
LBS + TF	**76.43**	92.47
LBS + SF + TF	76.28	92.32
LBS_M	75.61	**93.87**
LBS_M + SF	74.81-	92.59
LBS_M + TF	76.33	92.37
LBS_M + SF + TF	75.11	92.52

Table 3.7.: ACC_1 and ACC_2 for different feature combinations trained on *Train* and tested against *Combined*. The '−' signs indicate a statistically significant difference between the marked results and LBS + TF.

$$\begin{aligned} J(\mathrm{TMP}_{\mathrm{base}}, E) &= \begin{cases} 1 & \text{if } E = E_0 \\ i & \text{for } E_i \end{cases} \\ \mathrm{TMP}_{\mathrm{oerf}} &= \mathrm{TMP}_{\mathrm{base}} \cdot J(\mathrm{TMP}_{\mathrm{base}}, E) \end{aligned} \tag{3.7}$$

3.2.3. Evaluation

In a first evaluation step, we compute ACC_1 for different feature combinations. We then compare the best combination with publicly available algorithms as well as other simple correction schemes.

3.2.3.1. Feature Evaluation

We trained the classifier using the dataset *Train* with different combinations of the proposed audio features and measured the performance against the dataset *Combined*. All tested feature combinations clearly outperformed the baseline algorithm `base` (TMP_{base} without correction) by at least 6 pp (percentage points) for ACC_1 (Table 3.7). As was to be expected, ACC_2 didn't change significantly. The best performing feature combination was LBS + TF with an ACC_1 of 76.43%. When testing for significance with McNemar's test and a significance level of $p < 0.01$ [201], we found that LBS + TF performed significantly better w.r.t. ACC_1 than LBS, LBS_M + SF, and `base`. In the following we refer to the error classifier trained with LBS + TF as `post`. If no tempo estimation algorithm is explicitly mentioned, the method from Section 3.2.1.1 is otherwise implied.

Dataset	base	base+post (oerf)	stem	stem+post	böck	böck+post	gflr	gflr+post
ACM Mirum	73.5	**81.6+**	73.6	79.0+	74.0	75.6+	76.0	81.0+
ISMIR04 Songs	64.7	68.1	58.8	61.0	55.0	58.0+	**73.0**	69.6-
Ballroom	64.3	85.0+	63.2	80.8+	84.0	**89.5+**	66.8	84.7+
Hainsworth	64.4	73.0+	67.1	72.0+	80.6	**81.5**	70.7	73.9
GTzan	72.2	**77.2+**	76.1	75.1	69.7	70.3	77.1	75.4
SMC	37.8	33.2	21.2	21.7	**44.7**	43.8	35.0	31.3-
Dataset Avg	62.8	69.7	60.0	65.0	68.0	**69.8**	66.5	69.3
Combined	68.1	**76.4+**	67.5	72.8+	71.2	73.3+	71.8	75.8+
GiantSteps	61.1	62.7	44.4	62.2+	58.9	**68.7+**	56.6	58.6
GiantSteps + Combined	67.4	**74.5+**	64.2	71.3+	69.5	72.6+	69.7	73.4+

Table 3.8.: Tempo results for ACC_1 (with 4% tolerance) in percent. The '+' and '−' signs indicate a statistically significant difference between an algorithm and the same algorithm enhanced with `post`. Bold numbers mark the best-performing algorithm(s) for a dataset. *Dataset Avg* is the mean of the algorithms' results for each dataset except *GiantSteps*.

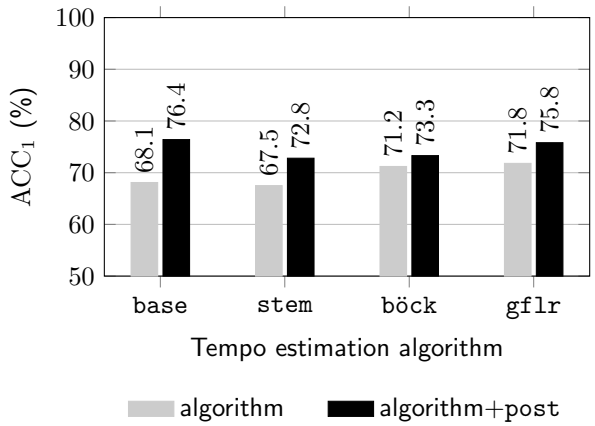


Figure 3.5.: ACC_1 for *Combined* for different algorithms with and without `post` error correction. All algorithms reach significantly higher scores when combined with `post`.

3.2.3.2. Comparative Evaluation

We compared our method `base+post`[6] to its baseline `base` and the three publicly available algorithms `böck`, `stem`, and `gflr` (Section 3.1) using the test datasets described in Section 3.2.1.2.

`böck`[7] is the algorithm published by Böck et al. in [6], but trained with different datasets—among them our test data, i.e., the algorithm is "familiar" with the test sets. According to the authors, this configuration participated in MIREX 2016. `stem` is an algorithm aiming for low computational complexity published by Percival et al. [128]. We used the implementation

[6] In later chapters `base+post` is referred to as `oerf`, **O**ctave **E**rror correction with **R**andom **F**orests.
[7] `https://github.com/CPJKU/madmom/`

Figure 3.6.: ACC_1 for *Combined* using no error correction, constraint-based correction, and `post` correction for various tempo estimation algorithms.

contained in Marsyas 0.5.0.[8] Lastly, `gflr` is the system described in Section 3.1. Since our error estimation and correction method can be used as a post-processor for any tempo estimator, we also trained the classifier for each of these three tempo estimation algorithms to investigate potential improvements.

Figure 3.5 shows the ACC_1 results for the four algorithms when tested against *Combined*, both with and without post-processing. All of them score significantly higher values when combined with `post` than in their plain form (McNemar, $p < 0.01$). The algorithms `base` (76.4%, increase of +8.3 pp) and `stem` (72.8%, +5.3 pp) clearly benefit the most, but also `böck` (73.3%, +2.1 pp) and `gflr` (75.8%, +4.0 pp) gain several percentage points.

As discussed in Section 3.2.1.3, a simple error correction scheme can be based on octave bias exploiting statistical properties of the test dataset. We therefore compared our method with such a constraint-based scheme, where tempi below a lower interval bound are doubled and above an upper bound are halved. For intervals we used the sweet octave and those listed in Table 3.5 with 90% and 95% coverage. Results are shown in Figure 3.6. Except for `böck`, none of the algorithms benefitted much from the simple correction—perhaps a certain bias is already built-in. When comparing `böck+post` and `böck +90%` we were not able to observe a significant difference. It appears, as if `böck`'s octave errors are harder to predict and correct than those of the other algorithms, perhaps because they are less systematic in nature.

Table 3.8 provides a detailed overview of ACC_1 results for each of the test algorithms for all test datasets.[9] As mentioned, `base+post` reaches the highest score for the *Combined* dataset (76.4%). To the best of our knowledge, this is the highest ACC_1 score reported for *Combined* to date.[10] For four of the six *Combined* datasets, `base+post` reaches significantly higher values than `base`

[8] `http://marsyas.info/`

[9] Results differ from [160], because in [160] erroneously an 8% tolerance was used instead of 4%.

[10] At the time, when this work was originally published.

(indicated by '+' signs in Table 3.8). The largest improvement was achieved for the *Ballroom* test set. The score for `base+post` is more than 20 pp higher than `base`'s. The fact that 29% of *Train* consists of ballroom tracks certainly plays a role here. While the `base+post` score for *ISMIR04 Songs* is 3.4 pp higher than `base`'s, the improvement is not significant. Similarly, the change for the *SMC* dataset (−4.6 pp) is not significant, but noteworthy. We believe that both octave and genre bias may play a role here. Tracks in *SMC* are very different in style from those in *Train*. And compared to *SMC*, *Train* contains relatively few examples for slow tracks with 60 BPM or less. Informal tests confirm that choosing a different training dataset leads to better results.

Dataset-specific scores for `böck+post` are all higher than those for `böck`—more than half of them significantly. The largest increase can be observed for the *GiantSteps* dataset. Plain `böck` scores 58.9%—combined with `post` it reaches 68.7% (+9.8 pp). To the best of our knowledge, this is the highest reported value for an unbiased, non-commercial algorithm to date.[11,12]

Dataset Avg is the mean of the results for each of the six datasets in *Combined*. Because it is an unweighted average, it is not dominated by the larger datasets. But just like for *Combined*, we can observe higher scores for all algorithms when combined with `post`. With 69.8% (+1.8 pp) `böck+post` reaches the highest score, closely followed by `base+post` with 69.7% (+6.9 pp). `stem` (65.0%, +5.0 pp) and `gflr` (69.3%, +2.8 pp) benefitted as well.

Though not the topic of this section, we also measured ACC_2. As expected, the results did not surprise and stayed stable.

3.2.4. Conclusions

We have shown that the proposed error correction method based on supervised learning of tempo estimation errors is capable of significantly improving ACC_1 results for existing tempo estimation algorithms. It does so in an algorithm-specific post-processing step. Combined with a simple tempo estimation algorithm, it outperforms other state-of-the-art algorithms for most of the tested datasets. We believe the error correction method can be enhanced even further by carefully selecting and incorporating other genre-related features.

We also discussed different kinds of biases that can have a large influence on the accuracy of tempo estimation algorithms. Ideally, evaluations of general purpose tempo estimators should be based on datasets with a mostly uniform tempo and genre distribution. Because better training data potentially leads to better results, training datasets should be an integral part of the comparison to make fair benchmarking possible. Defined train/test splits for existing datasets could be a first step in this direction.

[11] `http://www.cp.jku.at/datasets/giantsteps/`

[12] At the time, when this work was originally published in [160].

4. CNN-Based Tempo Estimation

In this chapter, we propose a method for local and global tempo estimation that works without an explicit OSS or other mid-level representation using solely a convolutional neural network. It is based on my work in [161].

Many different approaches to tempo estimation have been taken in the past. Early tempo estimation methods often combined signal processing with heuristics. Scheirer [150] for example used bandpass filters, followed by parallel comb filters, followed by peak picking. Klapuri et al. [86] replaced the conventional bandpass approach with STFTs, producing 36 band spectra. By differentiating and then half-wave rectifying the power in each band, they created band-specific onset strength signals (OSS), which were then combined into four accent signals and fed into comb filters in order to detect periodicities. Instead of processing an OSS with comb filters, several other methods have been proposed. Among them autocorrelation [4, 117], clustering of inter-onset intervals (IOI) [33, 169], and the discrete Fourier transform (DFT) [126, 117].

Recent approaches put emphasis on finding not just *a* periodicity, but on finding one corresponding to the perceived tempo, trying to avoid so-called *octave errors* (Chapter 3) [64]. The methods used range from genre classification (e.g., obtained by a genre classification component) [168, 75], secondary tempo estimation (Section 3.1), and the discrete cosine transform of IOI histograms [43], to machine learning approaches like Gaussian mixture models (GMM) [127], support vector machines (SVM) [56, 128], k-nearest neighbor classification (k-NNC) [197, 196], neural networks [42], and our own method using random forests (Section 3.2).

Another area of active research aims at creating a better OSS through the use of neural networks. Elowsson [42] uses harmonic/percussive source separation and two different feedforward neural networks to classify a frame as beat or non-beat. Böck et al. [6] use a bidirectional long short-term memory (BLSTM) recurrent neural network (RNN) to map spectral magnitude frames and their first order differences to beat activation values. These are then processed further with comb filters. For their dancing robot application, Gkiokas et al. [55] use a convolutional neural network (CNN) to derive a beat activation function, which is then used for beat tracking and tempo estimation.

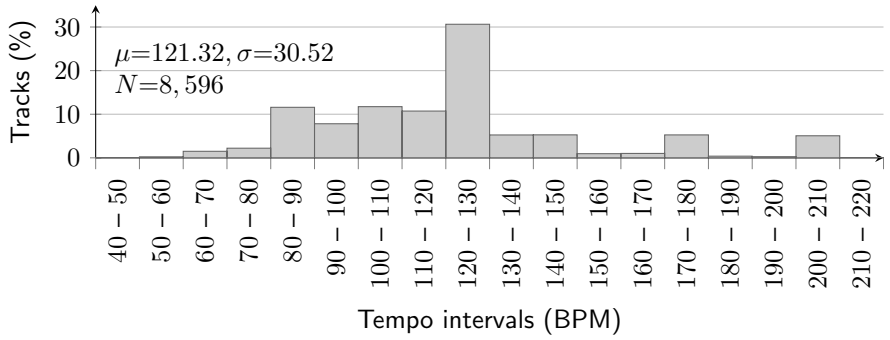


Figure 4.1.: Tempo distribution for the *Train* dataset consisting of *LMD Tempo*, *MTG Tempo*, and *EBall*.

What all these methods have in common is the *multi-step* approach of decomposing the signal into sub-bands, deriving some kind of OSS, detecting periodicities, and then trying to pick the best one. As Humphrey et al. [76] point out, this can be described as a deep architecture consisting of multiple components ("layers") that has evolved naturally. But to the best of our knowledge, nobody has replaced the traditional multi-component architecture with a single deep neural network (DNN) yet. In this chapter we describe a CNN-based approach that estimates the local tempo of a short musical piece or excerpt based on mel-scaled spectrograms in a single step, i.e., without explicitly creating mid-level features like an OSS or a beat activation function that need to be processed further by another, separate system component. Using averaging, we can combine multiple local tempi into a global tempo. Please note that in this thesis we are not going to introduce basics of deep learning, but refer to [58].

The remainder of this chapter is structured as follows: Section 4.1 introduces our training datasets. Then, Section 4.2 describes the signal representation, network architecture, network training, and how we combine multiple local estimates into a global estimate. In Section 4.3 we evaluate our global tempo estimation approach quantitatively by benchmarking against known datasets and state-of-the-art algorithms. Then we discuss local tempo estimation qualitatively using samples from different genres and eras. Finally, in Section 4.4 we present our conclusions.

Reproducibility

See Appendix A.3 for an implementation of the method presented in this section.

Training dataset annotations are available at DOI 10.5281/zenodo.3553592.

4.1. Training Datasets

Our goal is to create a general purpose system that does not suffer from strong genre bias. Therefore we avoid cross-validation on small datasets and instead created a large, multi-genre training dataset, consisting of three smaller datasets: One derived from a subset of the Lakh

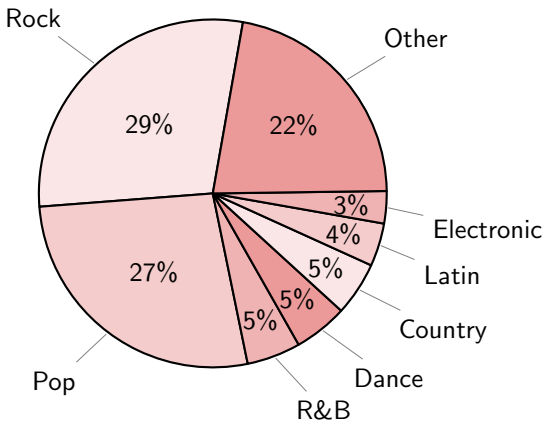


Figure 4.2.: Genre distribution in *LMD Tempo*.

MIDI dataset (*LMD*) [136], a subset of the GiantSteps MTG key dataset (*MTG Key*) [45][1], and a subset of the *Extended Ballroom* [105] dataset. Two of the derived ground-truths have been newly created for this work.

4.1.1. LMD Tempo

LMD is a dataset containing MIDI files that have been matched to 30 s audio excerpts. While some of the MIDI files contain tempo information, none of the audio files are annotated, and there is no guarantee that associated MIDI and audio files have the same tempo. Our idea is to create a sub-dataset, called *LMD Tempo*, that can be used for training supervised tempo induction algorithms. To this end, we estimated the tempo of the matched audio previews using the algorithm from Section 3.2. Then the associated MIDI files were parsed for tempo change messages. If the value of more than half the tempo messages for a given preview were within 2% of the estimated tempo, we assumed the estimated tempo of the audio excerpts to be correct and added it to *LMD Tempo*. This resulted in 3,611 audio tracks. We were able to match more than 76% of the tracks to the Million Song Dataset (*MSD*) genre annotations from [154]. Of the matched tracks 29% were labeled Rock, 27% Pop, 5% R&B, 5% Dance, 5% Country, 4% Latin, and 3% Electronic. Less than 22% of the tracks were labeled Jazz, Soundtrack, World and others (Figure 4.2). Thus it is fair to characterize *LMD Tempo* as a good cross-section of popular music.

4.1.2. MTG Tempo

The *MTG Key* dataset was created by Faraldo [45] as a ground-truth for key estimation of electronic dance music (EDM), a genre that is very much underrepresented in *LMD Tempo*. Each two-minute track in *MTG Key* is annotated with one or more keys and a confidence value

[1] `https://github.com/GiantSteps/GiantSteps-mtg-key-dataset`

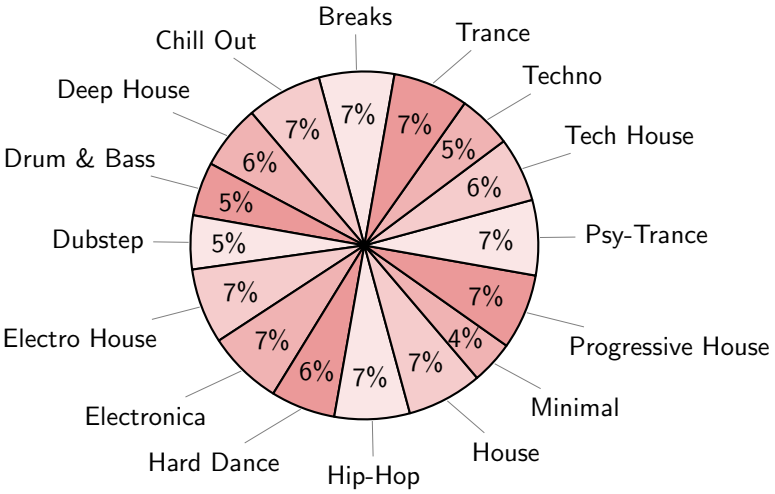


Figure 4.3.: Genre distribution in *MTG Tempo*.

$c \in \{0, 1, 2\}$ for the key annotation. We annotated those tracks that have an unambiguous key and a confidence of $c = 2$ with a manually tapped tempo, which makes it one of the very few datasets that is suitable for key *and* tempo estimation. The resulting dataset size is 1,159 tracks. In the following we will refer to this new ground-truth as *MTG Tempo*. The detailed genre distribution is depicted in Figure 4.3.

4.1.3. Extended Ballroom

The original *Ballroom* dataset [64] is still used as test dataset today, which is why we exclude it from training. Better suited is the recently released and much larger *Extended Ballroom* dataset. Because it contains some songs also occurring in *Ballroom*, we use the complement *Extended Ballroom* \ *Ballroom*. We refer to the resulting dataset as *EBall*. It contains 3,826 tracks with 30 s length each. *EBall* contributes tracks from genres that are underrepresented or simply absent from both *MTG Tempo* and *LMD Tempo*, like Waltz and Foxtrot. The exact genre distribution is depicted in Figure 4.4.

4.1.4. Combined Training Dataset

Combined, *LMD Tempo*, *MTG Tempo*, and *EBall* have a size of 8,596 tracks with tempi ranging from 44 to 216 BPM (Figure 4.1). In the following we will call it *Train*. The *sweet octave* (i.e., the tempo interval $[\tau, 2\tau)$ that contains the most tracks; see Section 3.2.1.3) for *Train* is $77-154$ BPM, covering 84.4% of the items. The shortest interval that covers 99% of the items is $65-204$ BPM. Even though many different tempi are represented, *Train* is not tempo-balanced. More than 30% of its tracks have tempi in the $[120, 130)$ interval. Its mean is $\mu = 121.32$ and the standard

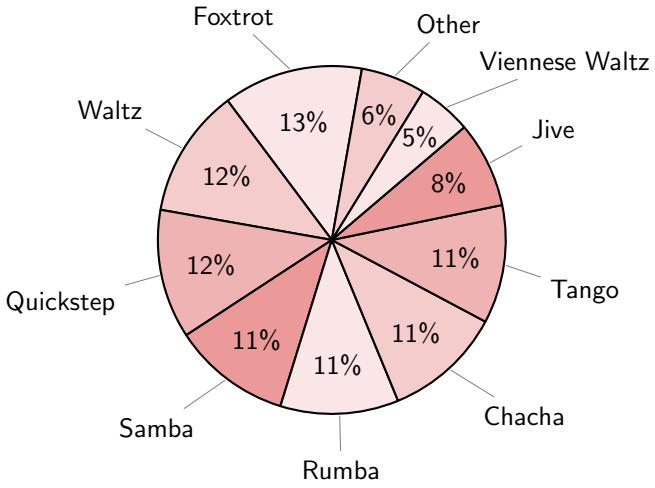


Figure 4.4.: Genre distribution in *EBall*.

deviation $\sigma = 30.52$. And while covering many different genres, *Train* is not genre-balanced, either. Genres like Jazz and World only have relatively few representatives. But despite these shortcomings, *Train* is a very rich, multi-faceted dataset and completely independent from the test datasets we are going to use for evaluation in Section 4.3.1.

4.2. Method

Our proposed method for estimating a local tempo consists of a single step. Using a suitable representation we classify the signal with a CNN, which produces a BPM value. We extend the system for global tempo estimation by averaging the softmax activation function over different parts of a full track.

4.2.1. Signal Representation

Although we believe that it is possible to build a system like ours with raw audio as input [30, 99], we choose to represent the signal as mel-scaled magnitude spectrogram to reduce the amount of data that needs to be processed by the CNN. The mel-scale as opposed to a linear scale was chosen for its relation to human perception and instrument frequency ranges.

To create the spectrogram, we convert the signal to mono, downsample to 11,025 Hz and use half-overlapping windows of 1,024 samples. This is equivalent to a frame rate of 21.5 Hz, which (according to the Nyquist-Shannon-Kotelnikov sampling theorem) suffices to represent tempi up to 646 BPM—well above the tempi we usually find in music. Each window is transformed into a 40 band mel-scaled magnitude spectrum covering $20 - 5{,}000$ Hz by applying a Hamming window,

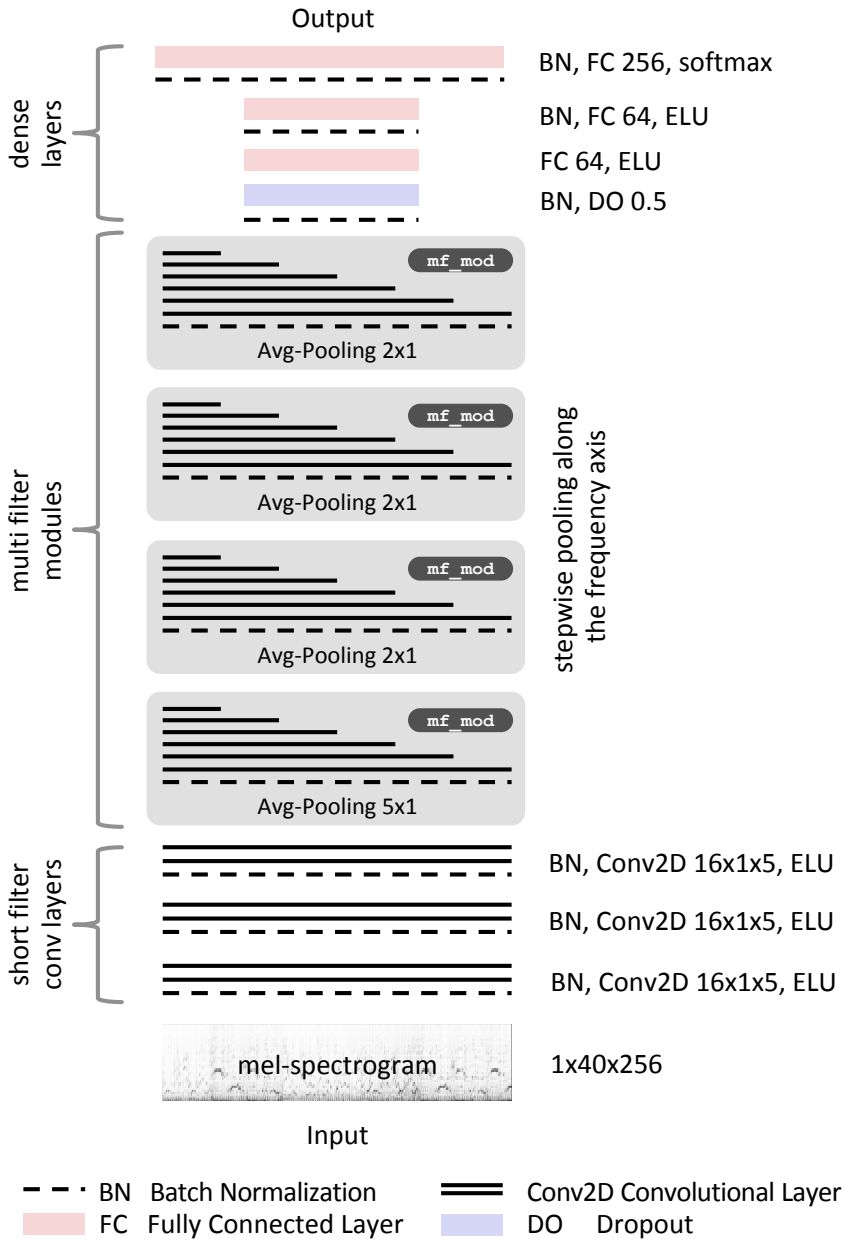


Figure 4.5.: Schematic overview of the network architecture. Three convolutional layers are followed by four *mf_mod* modules, which in turn are followed by four dense layers.

the DFT, and a suitable filterbank. Since musical tempo is not an instantaneous quantity, we require a spectrogram of a musically sufficient length. As such we choose 256 frames, equivalent to $\approx 11.9\,\mathrm{s}$.

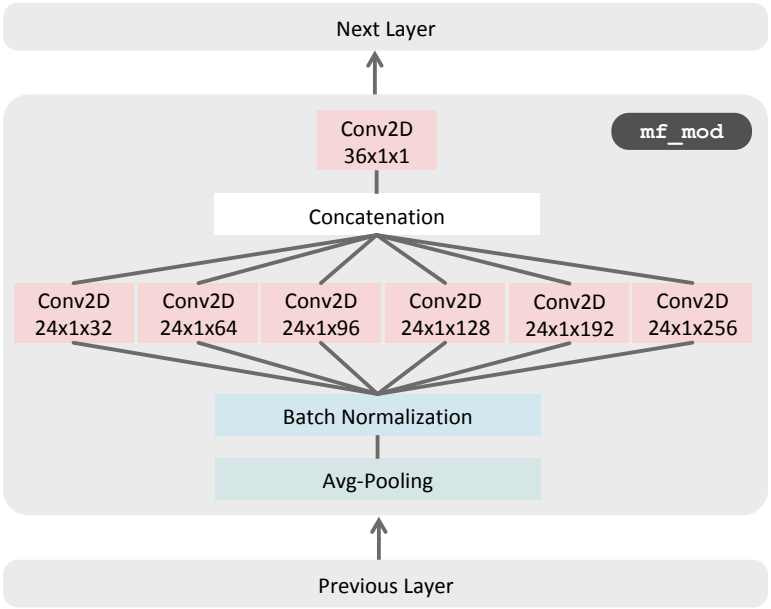


Figure 4.6.: Each multi-filter module *mf_mod* consists of a pooling layer, batch normalization, six different convolutional layers, a concatenation layer, and a bottleneck layer. The activation function for all convolutional layers is ELU.

4.2.2. Network Architecture

Even though tempo estimation appears to be a regression problem, we are approaching it as a classification problem for two reasons. First, a probability distribution over multiple classes allows us to judge how reliable a given estimate is. Additionally, such a distribution is naturally capable of representing tempo ambiguities [110], allowing for the estimation of a second best tempo. Second, in informal experiments we found that a classification-based approach led to more stable results compared to a regression-based approach. So instead of attempting to estimate a BPM value as decimal number, we are choosing one of 256 tempo classes, covering the integer tempo values from 30 to 285 BPM.

The proposed network architecture (Figure 4.5) is inspired by the traditional approach of first creating an OSS, which is then analyzed for periodicities. In our approach, we first process the input with three convolutional layers with 16 (1×5) filters each. All filters are oriented along the time axis using padding and a stride of 1. Using these fairly short filters, we hope to match onsets in the signal.

These three layers are followed by four almost identical multi-filter modules (*mf_mod*, Figure 4.6) each consisting of an average pooling layer $(m \times 1)$, parallel convolutional layers with different filter lengths ranging from (1×32) to (1×256), a concatenation layer and a (1×1) bottleneck layer for dimensionality reduction. With each of these modules we are trying to achieve two goals: 1) Pooling along the frequency axis to summarize mel-bands, and 2) matching the signal

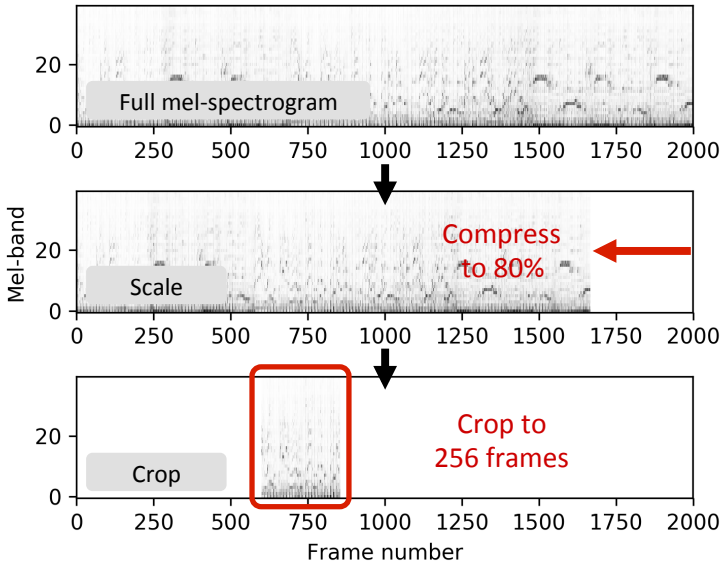


Figure 4.7.: Scale-&-crop data augmentation. During training, the mel-spectrogram is first stretched or compressed along the time axis, which requires an adjustment of the ground-truth label, and then cropped to 256 frames at a randomly chosen offset.

with a variety of filters that are capable of detecting long temporal dependencies. Using parallel convolutional layers with different filter lengths has been inspired by [130, 184]. In a traditional system, this could be regarded as some sort of comb filterbank.

To classify the features delivered by the convolutional layers, we add two fully connected layers (64 units each) followed by an output layer with 256 units. The output layer uses softmax as activation function, while all other layers use ELU [24]. Each convolutional or fully connected layer is preceded by batch normalization [78]. The first fully connected layer is additionally preceded by a dropout layer with $p = 0.5$ to counter overfitting. As loss function we use categorical cross-entropy. Overall, the network has 2,921,042 trainable parameters.

4.2.3. Network Training

We use 90% of *Train* for training and 10% for validation. To counter the tempo class imbalance and, at the same time, augment the dataset during training, for each epoch, we use a scale-&-crop-approach borrowed from image recognition systems (see e.g., [174]). Contrary to regular images, the two dimensions of spectrograms have very different meaning, which is why we cannot simply scale-&-crop indiscriminately. Instead, we have to be careful to either not change the labeled meaning of a sample or change its label suitably (Figure 4.7). In our case this means that we have to preserve the properties of the frequency axis, but may manipulate the time axis. Concretely, we scale the time axis of the samples' mel-spectrograms with a randomly chosen factor $\in \{0.8, 0.84, 0.88, \ldots, 1.16, 1.2\}$ using spline interpolation and adjust the ground-truth

	(a) ACC_0			(b) ACC_1			(c) ACC_2		
Dataset	`oerf`	`böck`	`cnn`	`oerf`	`böck`	`cnn`	`oerf`	`böck`	`cnn`
ACM Mirum	**45.2+**	29.4-	40.6	**81.8**	74.0-	79.5	97.3	**97.7**	97.4
ISMIR04 Songs	**39.0**	27.2-	34.1	**66.8+**	55.0	60.6	91.6	**95.0**	92.2
Ballroom	60.9-	33.8-	**67.9**	85.5-	84.0-	**92.0**	98.0	**98.7**	98.4
Hainsworth	**48.6**	33.8	43.2	73.0	**80.6**	77.0	85.1	**89.2+**	84.2
GTzan	**43.7+**	32.2-	36.9	**77.4+**	69.7	69.4	93.2	**95.0+**	92.6
SMC	14.3	**17.1**	12.4	35.0	**44.7+**	33.6	53.9	**67.3+**	50.2
GiantSteps	52.1-	37.2-	**59.8**	62.2-	58.9-	**73.0**	88.7	86.4-	**89.3**
Combined	**46.3**	31.2-	44.8	**74.6**	69.5-	74.2	92.1	**93.6+**	92.1
Dataset Avg	**43.4**	30.1	42.1	68.8	66.7	**69.3**	86.8	**89.9**	86.4

Table 4.1.: Accuracies in percent. The '+' and '−' signs indicate a statistically significant difference between either `oerf` or `böck`, and `cnn`. Bold numbers mark the best-performing algorithm(s) for a dataset. *Dataset Avg* is the mean of the algorithms' results for each dataset.

tempo labels accordingly. This substantially increases the number of different samples we can present to the network. Since the full mel-spectrogram for a sample is longer than the network input layer (e.g., covering 60 s vs. 11.9 s), we crop each scaled sample at a randomly chosen time axis offset to fit the input layer. This again drastically increases the number of different samples we can offer to the network. After scaling and cropping, the values of the resulting sub-spectrogram are rescaled to $[0, 1]$. In order to ensure comparability, time-axis augmentations are skipped during validation.

We define ACC_0 as the fraction of estimates that are correct when rounding decimal ground-truth labels to the nearest integer. To avoid overfitting, we train until ACC_0 for the validation set has not improved for 20 epochs using Adam [84] (with a learning rate of 0.001, $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1\text{e}{-8}$) as optimizer, and then keep the model that achieved the highest validation ACC_0 (early stopping).

4.2.4. Global Tempo Estimation

Since the input layer is usually shorter than the mel-spectrogram of a whole track, it estimates merely a local tempo. To estimate the global tempo for a track, we calculate multiple output activations using a sliding window with half-overlap, i.e., a hop size of 128 frames ≈ 5.96 s. The activations are averaged class-wise and then—just like in the local approach—the tempo class with the greatest activation is picked as the result.

4.3. Evaluation

For evaluation, we trained three models and chose the one with the highest ACC_0 measured against the validation set as our final model. As metrics we used ACC_0 as well as ACC_1 and ACC_2.

4.3.1. Global Tempo Benchmarking

It has become customary to benchmark tempo estimation methods with results reported for a small set of datasets: *ACM Mirum* [127], *Ballroom* [64], *GTzan* [185], *Hainsworth* [70], *ISMIR04 Songs* [64], *GiantSteps Tempo* [88], and *SMC* [74]. The latter was specifically designed to be difficult for beat trackers. Where applicable, we used the corrected annotations from [128]. A detailed description of the datasets is given in Section 3.2.1.2. We refer to the union of these seven datasets as *Combined*. Unweighted averages of results for all seven datasets will be referred to as *Dataset Avg*. We benchmarked our approach `cnn` with the algorithms by Böck et al. (`böck`) [6][2] and our own approach from Section 3.2 (`oerf`). Table 4.1 shows the results.[3]

Overall, `cnn` achieves results similar to `oerf`'s when tested against *Combined* with the strict metrics ACC_0 (44.8 %) and ACC_1 (74.2 %).[4] Both accuracy values are slightly lower when summarized as *Dataset Avg*. Here, `cnn` reaches the highest ACC_1 value (69.3 %) of the compared systems. When testing with octave-error tolerance, i.e., ACC_2, `böck` reaches 93.6 % for *Combined*, versus 92.1 % reached by `oerf`, and 92.1 % reached by `cnn`. In essence, `cnn` and `oerf` are better than `böck` at estimating the tempo octave correctly, while `böck` achieves a slightly higher accuracy when ignoring the metrical level. This may be due to the fact that `böck` first and foremost attempts to identify individual beats instead of looking at the larger picture.

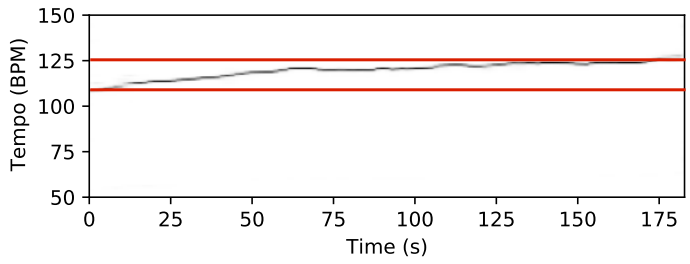
When inspecting the dataset-specific results, we find that `cnn`'s ACC_1 is particularly high for *Ballroom* (92.0 %) and *GiantSteps* (73.0 %). In fact, they are significantly higher than `böck`'s (+8.0 pp/+14.1 pp) or `oerf`'s (+6.9 pp/+10.8 pp) results. Both the *Ballroom* and *GiantSteps* values can be explained through our training dataset. They clearly correspond to *EBall* and *MTG Tempo*, therefore high values are not surprising. To us these results indicate that a genre-complete training set may lead to better results for the other datasets as well. This hypothesis is supported by the fact that *GTzan* contains genres like Reggae, Classical, Blues, and Jazz, and *Hainsworth* contains the genres Choral, Classical, Folk, and Jazz—none of which are well represented in

[2] madmom-0.15.1, default options, available at `https://github.com/CPJKU/madmom`

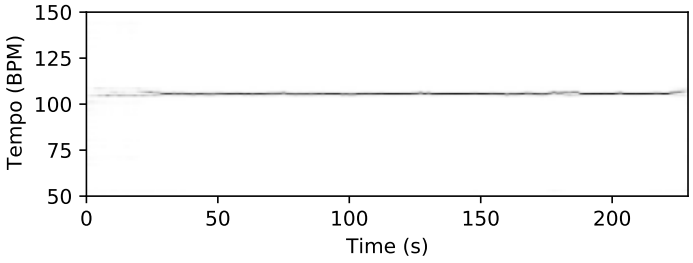
[3] Due to a (now fixed) programming error in the public `oerf` package (Appendix A.2), we have erroneously reported lower values for `oerf` in [161].

[4] Similar ACC_1 results are also reported in the recent work by Foroughmand and Peeters [49], but a significant difference has not been shown.

(a) *Honky Tonk Women* by The Rolling Stones



(b) *Rolling in the Deep* by Adele



(c) *Typhoon* by Foreign Beggars/Chasing Shadows

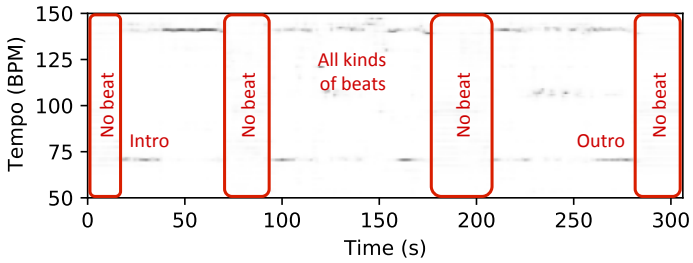


Figure 4.8.: Tempo class probabilities for tracks from different genres and eras. **(a)** The tempo drift of the performance is clearly visible: the track starts with 108 BPM and ends with 125 BPM. **(b)** Very stable tempo of a modern pop music production. **(c)** Dubstep track with several no beat passages, a very active middle section, and halve tempo intro and outro.

Train. A similar connection may exist for `böck` and *GiantSteps*—as far as we know, `böck` has not been trained on EDM.

4.3.2. Local Tempo Visualization

To illustrate the system's performance for continuous local tempo estimation, we analyzed several tracks from different genres using overlapping windows with a relatively small hop size of 32 frames, i.e., $e \approx 1.5$ seconds. For clarity, we cropped the images at 50 and 150 BPM. Figure 4.8a beautifully reveals the tempo drift in The Rolling Stone's 1969 performance of *Honky Tonk Women*, starting out at 108 BPM and ending in 125 BPM. In contrast, Adele's relatively recent

studio production *Rolling in the Deep* (Figure 4.8b) stays very stable at 105 BPM. A more complicated picture is presented by the dubstep track *Typhoon* by Foreign Beggars/Chasing Shadows (Figure 4.8c). After several seconds of weather noises, the intro starts with 70 BPM. The main part's tempo is clearly 140 BPM interrupted by two sections with no beat. The outro again feels like 70 BPM followed by a fade out.

4.4. Conclusions

We have presented a single-step tempo estimation system consisting of a convolutional neural network (CNN). With a conventional mel-spectrogram as input, the system is capable of estimating the musical tempo using multi-class classification. The network's architecture consolidates traditional multi-step approaches into a single CNN, avoiding explicit mid-level features such as onset strength signals (OSS) or beat activation functions. Consequently and contrary to many other systems, our approach does not rely on handcrafted features or ad-hoc heuristics, but is completely data-driven. The system was trained with samples from the union of several large datasets, two of which were newly created. To aid training, we applied problem-specific data augmentation techniques. For global tempo estimation, we have shown that our single network, data-driven approach performs as well as other more complicated state-of-the-art systems, especially w.r.t. ACC_1. Furthermore, by visualizing examples for local tempo estimations, we have demonstrated qualitatively how the system can aid music analysis, e.g., to identify tempo drift.

We believe that the system can be improved even further by training with a more balanced dataset that contains tracks for all tested genres. Notably missing from the current training set are Jazz, Classical, or Reggae tracks. Another area of potential improvement is the network architecture. Shorter filters, dilated convolutions, residual connections, and a suitable replacement for the fully connected layers might be used to reduce the number of parameters and thus the number of operations needed for training and estimation.

5. Electronic Dance Music: A Crowdsourced Experiment

In this chapter we describe an online experiment that aimed at investigating the reason for low accuracy scores for the *GiantSteps Tempo* dataset and report its results . It is based on my work in [162].

A necessary precondition for successful global tempo estimation is the existence of a stable tempo as it often occurs in Rock, Pop, or Dance music. To evaluate a tempo estimation system one needs the system itself, a dataset with suitable tempo annotations, and one or more metrics. One such dataset, named *GiantSteps Tempo*, has been released by Knees et al. in 2015 [88]. It was created by scraping a forum that lets listeners discuss Beatport[1] songs with wrong tempo labels. Scraping was done via a script and 15% of the labels were manually verified. All 664 tracks in the dataset belong to the umbrella genre electronic dance music (EDM) with its subgenres Trance, Drum-and-Bass, Techno, etc. Since its release, several academic and commercial tempo estimation systems have been tested against the dataset (e.g., Sections 3.2.3.2 and 4.3). As is common for datasets annotated with only a single tempo per track, the two metrics ACC_1 and ACC_2 were used. The highest results reported for the *GiantSteps* dataset are 77.0% ACC_1 by the applications NI Traktor Pro 2[2] (with octave bias $88-175$) and 90.2% ACC_2 by CrossDJ[3] (with octave bias $75-150$).[4] These results are surprisingly low—the highest reported ACC_2 values for other commonly used datasets like *ACM Mirum* [127], *Ballroom* [64], and *GTzan* [185] are greater than 95% [6]. Since EDM is often associated with repeating bass drum patterns and steady tempi [106, 15], it should be comparatively easy to estimate the tempo for this genre. We hypothesize that relatively low accuracy values were achieved for multiple possible reasons. Since the annotations were scraped off a forum for disputed tempo labels, the dataset may contain many tracks that are especially hard to annotate for humans. And if not difficult for humans to annotate, it is conceivable that the tracks are particularly hard for algorithms to analyze. Lastly, if neither humans nor algorithms fail, perhaps some of the scraped annotations are simply wrong.

[1] `https://www.beatport.com/`, an online music store

[2] `https://www.native-instruments.com/en/products/traktor/dj-software/traktor-pro-2/`

[3] `https://www.mixvibes.com/cross-free-dj-software/`

[4] More benchmark results are available at `http://www.cp.jku.at/datasets/giantsteps/`

In this chapter, we investigate why tempo estimation systems perform so poorly for *GiantSteps Tempo*. To this end, we conducted a large, crowdsourced experiment to collect new tempo data for *GiantSteps Tempo* from human participants. The experiment is described in detail in Section 5.1. The data is analyzed in Section 5.2 and used to create a new ground-truth. This ground-truth is then compared to the original ground-truth and used to evaluate three recent algorithms. The results are discussed in Section 5.3. Finally, in Section 5.4, we summarize our findings and draw conclusions.

Reproducibility

All collected data and derived annotations are available at DOI 10.5281/zenodo.3553625.

5.1. Experiment

In order to generate a new ground-truth for the *GiantSteps Tempo* dataset, we set up a web-based experiment in which we asked participants to tap along to audio excerpts using their keyboard or touchscreen. The user interface for this experiment is depicted in Figure 5.1. Since most tracks from the dataset are 2 min long and tapping for the full duration is difficult, we split each track into half-overlapping 30 s segments. Out of the 664 tracks we created 4,640 such segments (in most cases 7 per track). To measure tempo, it is not important for tap and beat to occur at the same time. In contrast to experiments for beat tracking, phase shifts, input method latencies, or anticipatory early tapping—known as *negative mean asynchrony* (NMA)—are irrelevant, as long as they stay constant (see [140] for an overview of tapping and [25, 74] for beat tracking). Therefore participants were asked to tap along to randomly chosen segments *as steadily as possible*, over the entire duration of 30 s without skipping beats. To encourage steady tapping, the user interface gave immediate feedback in the form of the mean tempo μ in BPM, the median tempo med in BPM, the standard deviation of the inter-tap intervals (ITI) σ in milliseconds, as well as textual messages and emojis (Figure 5.1). When calculating the standard deviation, the first three taps were ignored, as those are typically of low quality (users have to find their way into the groove). When the standard deviation σ stayed very low, smilies, thumbs up and textual praise were shown. When σ climbed above a certain threshold, the user was shown sad faces and messages like "Did you miss a beat? Try to tap more steadily." To prevent low quality submissions, users were only allowed to proceed to the next track, once four conditions were met:

1. 20 or more taps
2. Taps cover at least 15 s
3. ITI standard deviation: $\sigma < 50$ ms
4. Median tempo: $50 \leq \text{med} \leq 210$ BPM

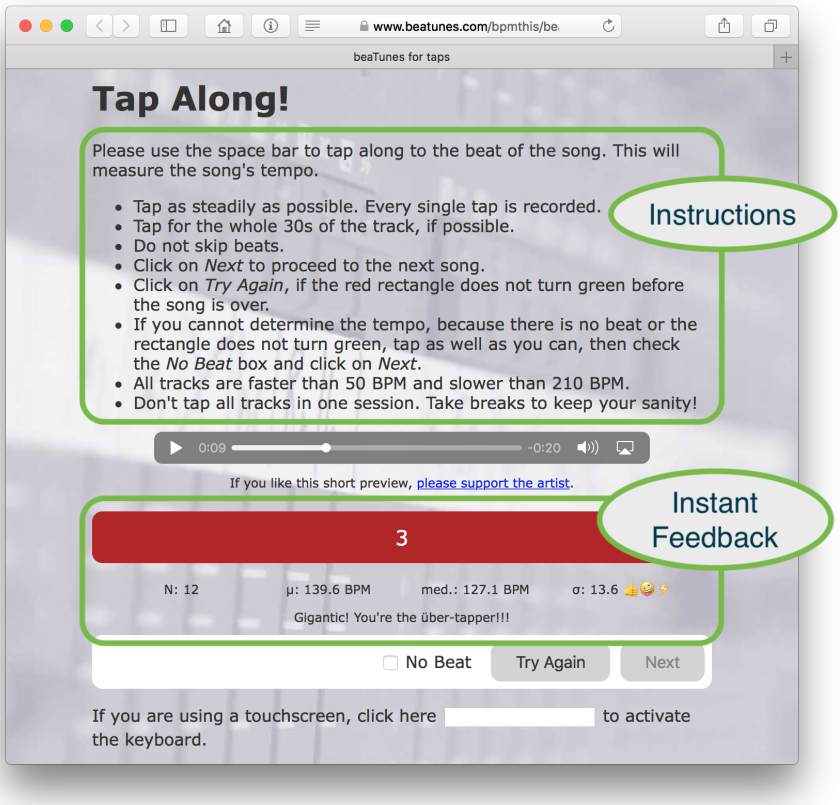


Figure 5.1.: Illustration of the web-based interface used in our experimental user study.

While the first three conditions were not explicitly communicated, the instructions made participants aware that the target tempo lies between 50 and 210 BPM. Once all four conditions were met, a large red bar turned green and the *Next* button became enabled. For situations in which the user was not able to fulfill all conditions, the user interface offered a *No Beat* checkbox. Once checked, it allowed users to bypass the quality check and proceed to the next song. It must be noted that there is a tradeoff between encouraging participants to tap well (i.e., steadily) and a bias towards stable tempi. We opted for this design for two reasons. 1) tempo in EDM is usually is very steady [106, 15]. 2) the bias is limited to individual tapping sessions at the segment level, i.e., we can still detect tempo stability problems on the track level by aggregating segment level annotations.

Participants were recruited from two distinct groups: Academics and people interested in the consumer-level music library management system beaTunes (Appendix D). We refer to the former group as *academics* and the latter as *beaTunes*. While members of the *academics* group were asked to help in this experiment via relevant mailing lists without offering any benefits, members of the *beaTunes* group were incentivized by promising a reward license for the beaTunes software, if they submitted 110 valid annotations. While it was not explicitly specified what a "valid

annotation" is, we attempted to steer people in the right direction using instructions and the instant feedback mechanisms described above (Figure 5.1).

5.2. Data Analysis

Over a period of 2½ months 266 persons participated in the experiment, 217 (81.6%) belonging to *beaTunes* and 49 (18.4%) to *academics*. Together they submitted 18,684 segment annotations (avg = 4.03/segment). We made sure that all segments were annotated at least twice. Since some segments are harder to annotate than others, we monitored submissions and ensured that segments annotated by participants as very different from the original ground-truth—exceeding a tolerance of 4%—were presented to participants more often than others. The vast majority of annotations was submitted by the *beaTunes* group (95.1%). Overall 7.5% of all submissions were marked with *No Beat*. With 7.6% the *No Beat*-rate was slightly higher among members of the *beaTunes* group. Members of *academics* checked *No Beat* only for 5.2% of their submissions. Since the experiment was run in the participant's web-browser, the browser's user-agent for each submission was logged by the web-server. Among other information the user-agent contains the name of the participant's operating system. 17,012 (91.1%) of the submissions were sent from desktop operating systems that are typically connected to a physical keyboard. 1,672 (8.9%) were from mobile operating systems that are usually associated with touchscreens. Participants interested in a reward license, also had to enter name and email. Both datapoints have been removed from the collected data to ensure anonymity.

We analyzed the submitted data to find out whether we can find quality differences between submissions from different participant groups (Section 5.2.1). Section 5.2.2 introduces metrics for ambiguity and stability. In Section 5.2.3, we measure to which extent participants agree on one or multiple tempi for the same segment. Then, in Section 5.2.4, we take a look at segment annotations aggregated on the track-level. Finally, in Section 5.2.5, we investigate whether tempo ambiguity is a genre-dependent phenomenon.

5.2.1. Submission Quality

We wondered how steadily participants tapped and whether some groups of participants tapped more steadily than others. Specifically, are the *beaTunes* submissions as good as the *academics* submissions? We can use the coefficient of variation[5]

$$c_{\text{var}} = \frac{\sigma}{\mu} \tag{5.1}$$

[5]Also known as CV or relative standard deviation (RSD).

Dataset Split	+	−	p-value
±*academics*	0.0074	0.0090	3.11×10^{-29}
±*keyboard*	0.0088	0.0095	9.74×10^{-7}
beaTunes±*keyboard*	0.0089	0.0099	6.71×10^{-10}
academics±*keyboard*	0.0074	0.0073	8.68×10^{-1}

Table 5.1.: Average coefficients of variation $\overline{c_{\text{var}}}$ for dataset splits, *academics* or not, *keyboard* or not, and *keyboard* or not for either *beaTunes* or *academics*. The low p-values indicate a significant difference between the dataset splits.

of each submission's ITIs as a normalized indicator for how steadily a participant tapped. To remove tapping outliers within a segment, we sort each submission's ITIs and only keep the central 10 before calculating the c_{var}. This has the effect of reducing c_{var} for all submissions. The average c_{var} for all submissions is $\overline{c_{\text{var}}} = 0.0089$. Assuming a normal distribution, this means that on average 99.7% of all central 10 ITIs lie within ±2.67% ($\equiv 3\sigma$) of their submission's mean value. Using $\overline{c_{\text{var}}}$ as a measure for the submission quality of different dataset splits, we found that members of *academics* tapped significantly more steadily ($\overline{c_{\text{var}}} = 0.0074$) than members of *beaTunes* ($\overline{c_{\text{var}}} = 0.0090$) (Table 5.1). To test for significance we used Welch's t-test. Also, submissions from desktop operating systems that are typically installed on devices connected to a physical keyboard (i.e., no touchscreen) are of significantly higher quality ($\overline{c_{\text{var}}} = 0.0088$) than submissions from devices using iOS or Android as operating system ($\overline{c_{\text{var}}} = 0.0095$). Despite the differences, we found that even the ITIs from the group with the highest $\overline{c_{\text{var}}}$, i.e., *beaTunes* without keyboard, still lie within only ±2.97% ($\equiv 3\sigma$) of their mean value 99.7% of the time—again assuming a normal distribution. This is well below the tolerance of 4% allowed by ACC_1.

We conclude that the data submitted by *academics* with keyboard is of the highest quality with regard to tempo stability, but find that the data submitted by members of *beaTunes* without keyboard is still acceptable, because the difference in $\overline{c_{\text{var}}}$ is not very large. This may be a direct result of the experiment's design which did not permit participants to submit highly irregular taps.

5.2.2. Tempo Distribution Metrics

How steadily participants tapped does not say anything about whether they tapped along to the true tempo. But since the purpose of the experiment is to create a new ground-truth, we cannot easily verify submissions for correctness. What we can do though, is to measure annotator (dis)agreement both for a segment and for all segments belonging to the same track. To this end, we define some metrics based on tapped tempo distributions. To create such a tapped tempo distribution for a segment, we combine the 10 central ITIs from each of its submissions in a histogram T with a bin width of 1 BPM and then normalize so that $\sum_{i=1}^{n} T(x_i) = 1$, with n as

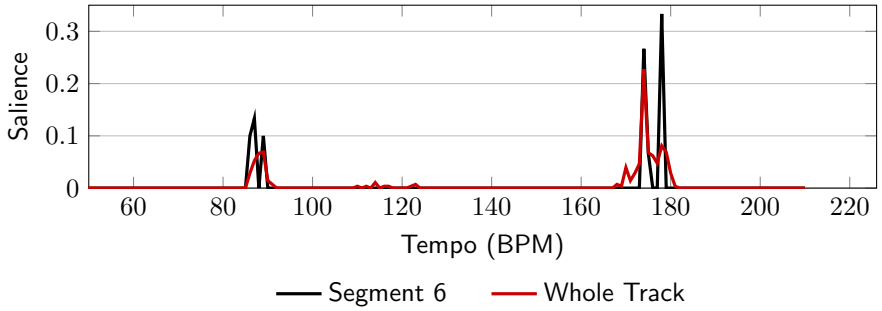


Figure 5.2.: Tempo salience distribution for segment 6 of track *Neoteric D&B Mix* by Polex (Beatport id 4397469). Measured values are: $P(T_{\text{track}}) = 4$, $P(T_{\text{seg6}}) = 2$, $A(T_{\text{track}}) = 0.30$, $A(T_{\text{seg6}}) = 0.40$, and JSD = 0.24.

the number of bins and x_i as the corresponding BPM values. For T we define local peaks as the highest non-zero $T(x_i)$ for all intervals $[x_i - 5, x_i + 5]$. This may include very small peaks. We interpret the BPM values x_i of the histogram's local peaks as the perceptually strongest tempi and their heights equivalent to their saliences. Per-track tempo distributions are created simply by averaging the 7 segment histograms belonging to a given track. For an example, please see Figure 5.2.

As a first, very simple indicator for annotator disagreement, we define $P(T)$ as the number of histogram peaks we find in a given tempo distribution T. A high peak count for a single segment $P(T_{\text{seg}})$ indicates annotator disagreement for that segment. This is not necessarily true for the peak count for a track $P(T_{\text{track}})$, since it may also be a sign of tempo instability, i.e., tempo changes or no-beat-sections. Because the peak count P does not say anything about the peaks' height or salience, it is a relatively crude measure. Therefore we define as second metric the salience ratio between the most salient and the second most salient peak as a measure for ambiguity. More formally, if s_1 is the salience of the highest peak and s_2 the salience of the second highest peak, then the ambiguity $A(T)$ is defined as:

$$A(T) := \begin{cases} 1, & \text{for } P(T) = 0 \\ 0, & \text{for } P(T) = 1 \\ s_2/s_1, & \text{for } P(T) > 1 \end{cases} \tag{5.2}$$

A value close to 0 indicates low and a value close to 1 high ambiguity. This definition is inspired by McKinney et al. [110] approach to ambiguity, but not identical. Just like P, we can use A for both segment and track tempo distributions. Again, for tracks we cannot be sure of the ambiguity's source.

Finally, we introduce a third metric that focuses more on tempo instability within tracks. Obvious indicators for instabilities are large differences between the tempo distributions of segments belonging to one track. Since we create tapped tempo distributions for each segment in a

way that lets us interpret them as probability distributions, we can use the Jensen-Shannon Divergence (JSD) for this purpose, which is based on the Shannon entropy H. With the JSD we measure the difference between the tempo distribution's entropy for the whole track and the average of the the individual segment tempo distributions' entropies.

$$\mathrm{H}(T) := -\sum_{i=1}^{n} T(x_i)\log_b T(x_i) \tag{5.3}$$

$$\mathrm{JSD}(T_1, ..., T_m) := \mathrm{H}\left(\sum_{j=1}^{m} \frac{1}{m} T_j\right) - \sum_{j=1}^{m} \frac{1}{m}\mathrm{H}(T_j) \tag{5.4}$$

To allow an easy interpretation of JSD-values, we choose an unusual base for the entropy's logarithm. By setting $b = n$ in Equation (5.3), we ensure that $0 \leq \mathrm{JSD} \leq 1$. This means, that a JSD-value near 0 indicates a small difference between the tempo distributions for a track's segments. Correspondingly, a JSD-value closer to 1 means that the tempo distributions of a track's segments are very different. To avoid detecting small tempo changes due to annotator disagreements, we convert the segment tempo distributions T to a bin width of 10 BPM before calculating JSD.

5.2.3. Segment Annotator Agreement

How much do participants agree on a tempo for a given segment? Recall that we have 4,640 segments (and 18,684 annotations for these segments) coming from 664 tracks. As depicted in Figure 5.3 top, the submissions for more than half the segments (2,500 or 53.9%) have just one peak, i.e., $P(T_{\mathrm{seg}}) = 1$. For 1,514 or 32.6% of all segments we were able to find two peaks, indicating some ambiguity. For 432 segments (9.3%) we found 3 peaks and for 184 segments (4.0%) 4 peaks or more. 10 segments have no peak at all, because they have been marked as *No Beat* in all their submissions. When interpreting these numbers one has to keep in mind that some segments have been annotated by very few participants (Figure 5.3 bottom). To give an example, while the segments annotated with one peak are based on 3.64 submissions on average, the segments annotated with 6 peaks are annotated with 9.42 submissions per segment. This reflects the fact that we presented difficult segments to participants more often, but could also be caused by increased variability introduced by a higher number of submissions. Because submissions marked as *No Beat* do not show up in this overview unless all submissions for a segment were *No Beat*, we counted the segments for which a majority of submissions were marked with *No Beat*. That was the case for 118 segments (2.5%).

As mentioned in Section 5.2.2, the peak count does not say anything about the peaks' height or salience and is therefore a relatively crude measure. We found that the average ambiguity for

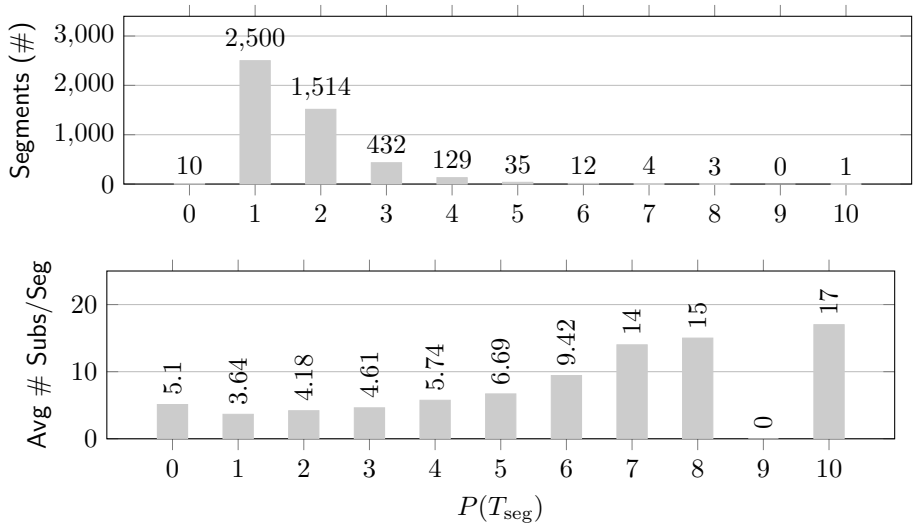


Figure 5.3.: (top) Segments per peak count. **(bottom)** Average number of submissions per segment by peak count.

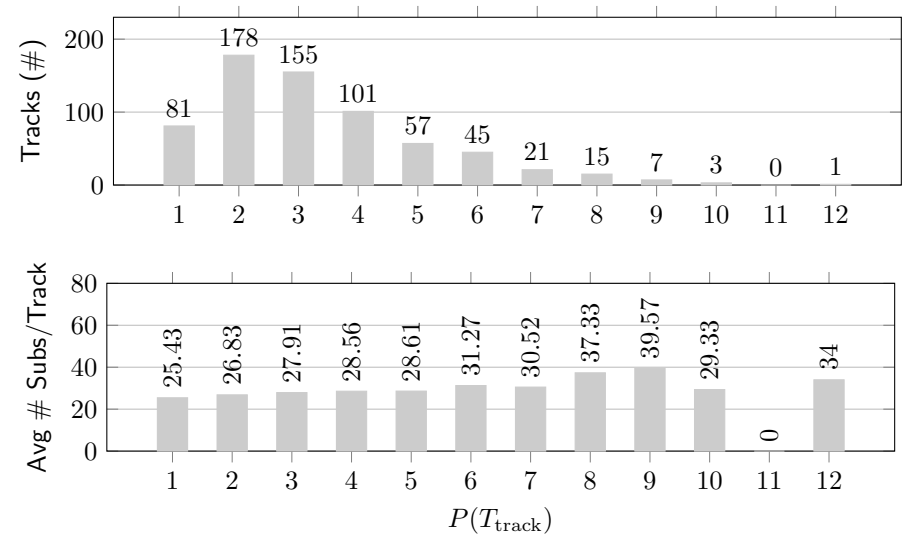


Figure 5.4.: (top) Tracks per peak count. **(bottom)** Average number of submissions per track by peak count.

all segments is $\overline{A(T_{\text{seg}})} = 0.25$ (with standard deviation $\sigma = 0.32$), meaning that on average the highest peak is four times more salient than the second highest peak. In other words, we can often observe a peak that is much more salient than others. At the same time, there may also be a second peak with considerable salience.

5.2.4. Track Annotator Agreement

Just like for the segments, we looked at the number of tracks per peak count. We found only 81 tracks (12.2%) with one peak and 582 tracks (87.8%) with two or more peaks (Figure 5.4 top).

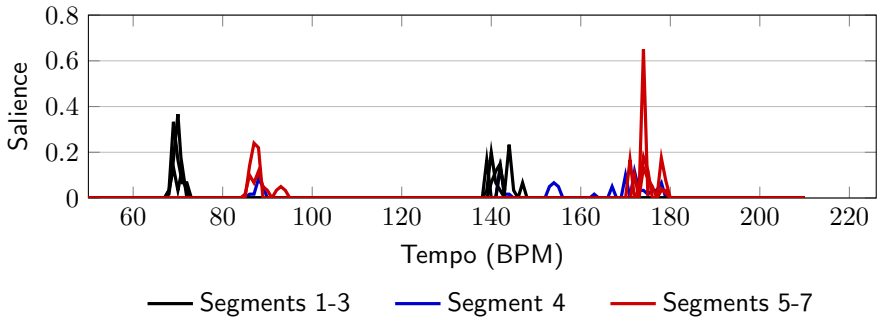


Figure 5.5.: Tempo salience distributions for segments of the track *Rude Boy feat. Omar LinX Union Vocal Mix* by Zeds Dead (Beatport id 1728723). The track's tempo changes in segment 4, leading to four distinct peaks. With JSD = 0.44 its Jensen-Shannon divergence is high.

The largest group among the multi-peak tracks are tracks with two peaks (178 or 26.8%). These numbers are much more reliable than the segment peak counts as they are based on at least 25 submissions per track (Figure 5.4 bottom). Compared to the segments' peak counts we see a larger proportion of tracks with more than one peak. But this does not necessarily mean that the ambiguity A is much higher than for the segments, because peak counts do not account for salience and even small local peaks are counted. In fact, we measured an average ambiguity of $\overline{A(T_{\text{track}})} = 0.26$ (with standard deviation $\sigma = 0.27$)—almost the same average as for the segments. Therefore we attribute the shift towards more peaks to the much higher number of submissions per item and possible tempo instabilities in the tracks themselves. By tempo instability we mean for example a tempo change in the middle of the track, a quiet section, or no beat at all. Any of these cases inherently lead to more peaks. A typical example for a track with a tempo change is shown in Figure 5.5.

In an attempt to quantify tempo instabilities in the submissions we calculated the JSD introduced in Section 5.2.2. The histogram in Figure 5.6 shows the distribution of tracks per JSD interval with a bin width of 0.05. The average divergence for the whole dataset is $\mu_{\text{JSD}} = 0.15$, the standard deviation is $\sigma_{\text{JSD}} = 0.11$. To test whether a high JSD correlates with tempo instabilities, we considered all tracks with JSD $> \mu_{\text{JSD}} + 2\sigma_{\text{JSD}} = 0.375$, resulting in 39 tracks. Performing an informal listening test on these tracks revealed that 3 had no beat, 10 contained a tempo change (e.g. Figure 5.5), 7 had sections that felt half as fast as other sections (metrical ambiguity), 8 contained larger sections with no discernible beat, 9 were difficult to tap, and 2 had a stable tempo through the whole track. From this result one may conclude that a high JSD is connected to tempo instabilities, but it may also just indicate that a track is difficult to tap. Nevertheless, using JSD helped us find tracks in the *GiantSteps Tempo* dataset that exhibit tempo stability issues. Since 2.5% of the segments were annotated most often with *No Beat*, we wondered whether any tracks have a majority of segments that have predominantly been annotated with *No Beat*, hinting at the absence of not just a local beat (e.g., a sound effect or a silent section), but the lack of a global beat. This is true for 6 tracks, i.e., 0.9% of the dataset. All 6 of them

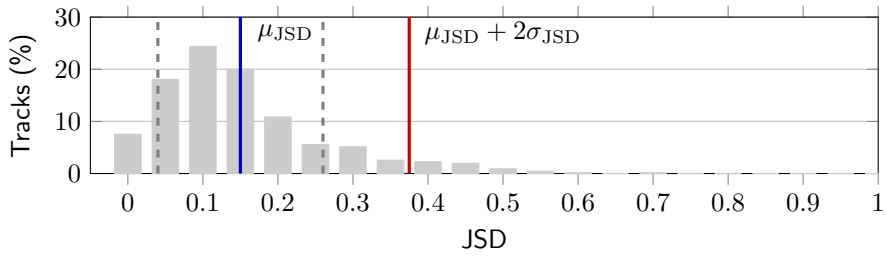


Figure 5.6.: Distribution of tracks in the dataset per JSD interval with a bin width of 0.05. The blue line shows μ_{JSD} and the red line shows $\mu_{\text{JSD}} + 2\sigma_{\text{JSD}}$.

Genre	$\overline{A(T_{\text{seg}})}$	$\overline{A(T_{\text{track}})}$
All	0.25	0.26
Techno	0.12	0.10
Trance	0.17	0.12
Drum & Bass	0.37	0.39
Electronica	0.36	0.38
Dubstep	0.35	0.43

Table 5.2.: Average ambiguity for the top five genres.

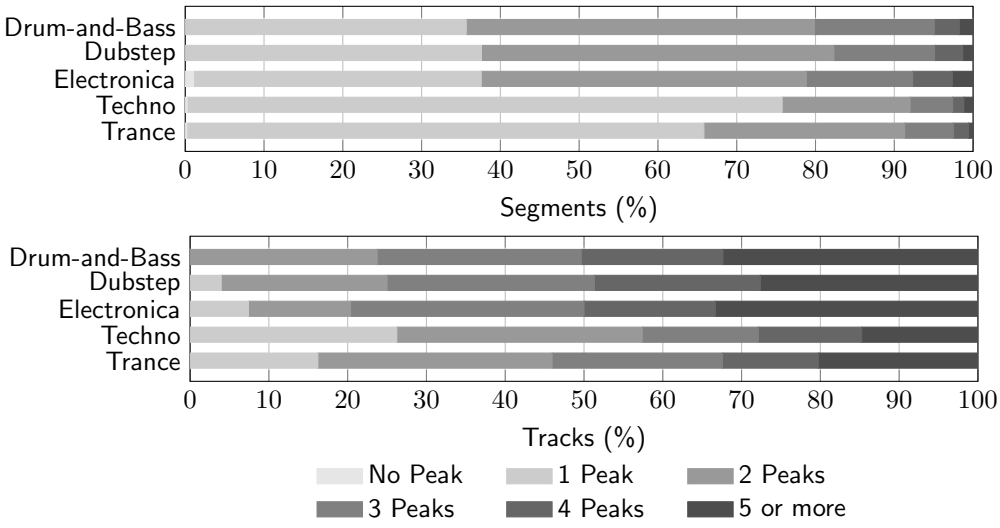


Figure 5.7.: Percentage of segments **(top)** and tracks **(bottom)** with a given number of peaks by genre. Drum-and-Bass, Dubstep, and Electronica suffer much more from tapped tempo ambiguity than Techno and Trance.

are among the 39 tracks with very high JSD and either have no beat, are very difficult to tap or contain large sections without a beat.

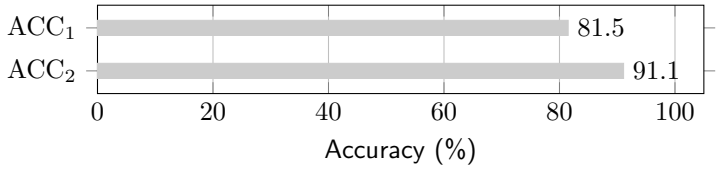


Figure 5.8.: Accuracies measured when comparing *GSNew* with *GSOrig*.

5.2.5. Ambiguity by Genre

We wondered whether we can confirm findings by McKinney and Moelants [111] that the amount of tempo ambiguity depends on the genre or musical style. To ensure meaningful results, we considered only the 5 most often occurring genres in the dataset with 54 or more tracks each. We found that the genres Techno and Trance do not seem to be very affected by ambiguity. More than 65% of their segments are annotated with just one peak. In contrast to that, fewer than 38% of all segments in the genres Drum-and-Bass, Dubstep, and Electronica are annotated with just one peak (Figure 5.7 top). A similar picture presents itself when looking at the average segment ambiguity $\overline{A(T_{\text{seg}})}$. As shown in Table 5.2, it is 0.12 for Techno segments and thus much lower than the overall average of 0.25. The same is true for Trance (0.17). Contrary to that, the ambiguity values for Drum-and-Bass (0.37), Electronica (0.36) and Dubstep (0.35) are all well above the average. We found similar relations for peak counts on the track level (Figure 5.7 bottom) and the average track ambiguity $\overline{A(T_{\text{track}})}$ (Table 5.2). This strongly supports McKinney and Moelants' finding that tapped tempo ambiguity is genre-dependent. Perhaps it is even an inherent property.

5.3. Evaluation

The tempo histograms for tracks can easily be turned into single tempo per track or two tempi+salience labels. This provides us the opportunity to evaluate the original ground-truth for the *GiantSteps Tempo* dataset by treating it like an algorithm. Since the original annotations are single tempo per track only, we are using ACC_1 and ACC_2 as metrics. To obtain one tempo value per track from a distribution, we are using just the tempo value with the highest salience. The three tracks without a beat have been removed. We refer to these new annotations as *GSNew* and to the original ones as *GSOrig*. Figure 5.8 shows the accuracy results for the comparison of *GSOrig* with *GSNew* and reveals a large discrepancy between the two. Only 81.5 of the labels match when using ACC_1, and only 91.1% match when using ACC_2.

Coming back to the original motivation for this chapter—the poor performance of tempo estimation systems for *GiantSteps Tempo* —we evaluated the three state-of-the-art algorithms `oerf` (Section 3.2), `cnn` (Chapter 4),[6] and `böck` [6] with both the old and the new annotations.

[6]The evaluation with `cnn` was added after the original publication in [162] to paint a more complete picture.

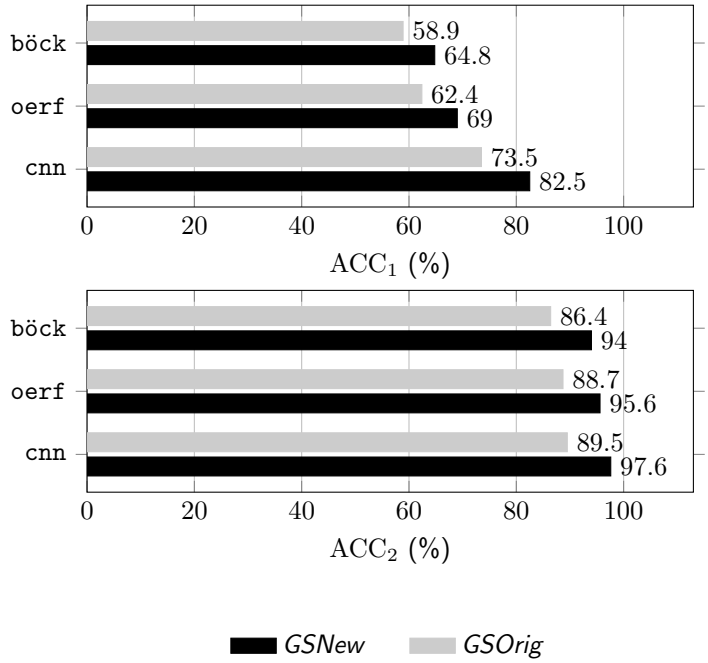


Figure 5.9.: Accuracies for the algorithms `böck`, `oerf`, and `cnn` measured against both *GSOrig* and *GSNew*.

The algorithms were chosen for their proven performance and conceptual dissimilarity. While `oerf` implements a conventional onset detection approach followed by an error correction procedure, `cnn` implements a single convolutional neural network (CNN), and `böck`'s core consists of a bidirectional long short-term memory (BLSTM) recurrent neural network (RNN). Despite their conceptual differences, all three algorithms reach considerably higher accuracy values when tested against *GSNew* (Figure 5.9). ACC_1 increases for `böck` by 5.9 pp (58.9% to 64.8%), for `oerf` by 6.6 pp (62.4% to 69.0%), and for `cnn` by 9.0 pp (73.5% to 82.5%). ACC_2 shows similar increases, 7.6 pp (86.4% to 94.0%) for `böck`, 6.5 pp (88.7% to 95.6%) for `oerf`, and 8.1 pp (89.5% to 97.6%) for `cnn`. Remarkably, all three systems reach higher ACC_2 values for *GSNew* than the original annotations reached, when compared with *GSNew*. The increased results for *GSNew* are much more in line with values reported for other tempo datasets. We therefore believe that this increase and the discrepancy between *GSOrig* and *GSNew* are hardly coincidences, but strong indicators for incorrect annotations in *GSOrig*.

5.4. Discussion and Conclusions

In this chapter we described a crowdsourced experiment for tempo estimation. We collected 18,684 tapped annotations from 266 participants for electronic dance music (EDM) tracks contained in the *GiantSteps Tempo* dataset. To analyze the data, we used multiple metrics and found that half of the annotated segments and more than half of the tracks exhibit some degree

of tempo ambiguity, which may either stem from annotator disagreement or from intra-track tempo instability. This refutes the assumption that it is always easy to determine a single global tempo for EDM. We were able to identify tracks with no tempo at all, no-beat-sections or tempo changes, which raises questions about the suitability of parts of the dataset for the global tempo estimation task. Furthermore, we provided additional evidence for genre-dependent tempo ambiguity. Based on the user-submitted data we derived the new annotations *GSNew*. The relatively low agreement with the original annotations *GSOrig* indicates that one of the two ground-truths contains incorrect annotations for up to 8.9% of the tracks (ignoring octave errors). We re-evaluated three recent tempo estimation algorithms against both ground-truths and measured considerably higher accuracies when testing against *GSNew*. This leads us to the following conclusions: *GSOrig* contains incorrectly annotated tracks as well as tracks that are not suitable for the global tempo estimation task. The accuracy of state-of-the-art tempo estimations systems is considerably higher than previously thought. And last but not least, as a community, we have to get better at evaluating tempo algorithms in the sense that we need verified, high quality datasets that represent reality with tempo distributions instead of single value annotations. If we cannot accurately measure progress, we have no way of knowing when the task is done.

6. Reflections on Global Tempo Estimation and Evaluation

In this chapter, we reflect on the current state of global tempo estimation evaluation. It is based on [165], which is currently under review. The Dataset Size section (Section 6.3.1) is based on computations, insights, and editing by Julián Urbano, who also made valuable suggestions for the rest of this chapter. My contributions are the main idea, the remaining text, all visualizations, and programming as well as the `tempo_eval` repository.

Estimation of a music recording's global tempo is a classic MIR task. It is often defined as estimating the frequency with which humans tap along to the beat [150, 33]. In contrast to beat-tracking [2, 60] or local tempo estimation [125], successful global tempo estimation requires the existence of a stable tempo as it often occurs in Rock, Pop, or Dance music. To conduct a basic evaluation of a global tempo estimation system one needs the system itself, test recordings with globally stable tempo, suitable annotations, and at least one metric. Starting with the work of Goto and Muraoka [60] and Scheirer [150], the MIR research community has been conducting such evaluations for 25 years. Acknowledging the importance of making results comparable, the first systematic evaluation with a defined set of metrics and datasets was conducted in 2004 [64]. One year later, the 2005 Music Information Retrieval Evaluation eXchange (MIREX) [36] established an automatic tempo extraction task, which has been conducted almost every year ever since. Through both the datasets and metrics established in 2004 and for MIREX, we have seen global tempo estimation systems mature and have been able to track their performance. In the meantime, new datasets have been published and another large-scale evaluation has been conducted [201], but neither applications nor metrics have been fundamentally questioned or updated. This is why recent near-perfect MIREX results by Böck et al. [6] and our CNN-based method from Chapter 4 beg the question: Are we done yet?

In this chapter, we critically discuss the evaluation of global tempo estimation systems. We do so based on the idea that applications lead to *use cases* that define who the users are, how they use the system, in what context and for what purpose [148]. The combination of these

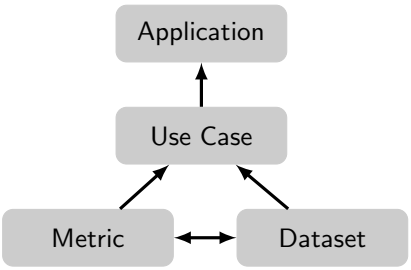


Figure 6.1.: Dependencies between application, use case, metric, and dataset (depends on: →).

elements determines the *success criteria* to evaluate systems and judge whether the task is indeed *solved* [183, 182].

We start our investigation in Section 6.1 with discussing presumed and actual applications. To evaluate how well an application's success criteria are met, datasets and metrics are used which must fulfill a central condition: both have to match the use case (Figure 6.1). With this in mind, we review metrics in Section 6.2 and datasets in Section 6.3. In Section 6.4, we describe how applications, metrics, and datasets fit into the MIR research cycle.

In order to learn what tempo estimation is used for and how the community measures success, we conducted a small survey among domain experts, which is presented in Section 6.5. In Section 6.6, we propose a public repository for reference annotations, estimates, and metrics to help with future evaluations. Finally, in Section 6.7, we draw conclusions.

Throughout this chapter we will illustrate some observations with tempo estimates produced by three systems: `perc` ■ [128], `böck` ■ [6][1], and `cnn` ■ (Chapter 4). They were chosen for illustrative purposes, their conceptual differences, and availability, not because they necessarily represent the state of the art.

6.1. Applications

Even though tempo estimation is a well established MIR task, the existing research rarely discusses in depth why tempo estimation is *relevant* and what the *application requirements* are.

[1]Estimates produced using *madmom TempoDetector 2016 version 0.17.dev0*. Results differ from the original publication, which used cross-validation for *Ballroom*, which may have introduced a strong genre bias (see Section 3.2.1.4).

6.1.1. Research Justifications

Dixon [33] identifies four main application types: performance analysis, perceptual modeling, audio content analysis for retrieval, and performance synchronization. Most applications described in later work fall into these four broad categories. Alonso et al. [3] mention automatic rhythmic alignment of audio, indexing for retrieval, and synchronized computer graphics. Peeters [126] explicitly adds automatic playlist generation, DJ applications like beat-mixing and looping, and further beat-synchronous analysis (e.g., cover song identification, Ellis and Poliner [41]). Tzanetakis and Percival [186] list applications such as music similarity and recommendation, semi-automatic audio editing, automatic accompaniment, polyphonic transcription, beat-synchronous audio effects, and computer assisted DJ systems. Böck et al. [6] add to this the contribution tempo estimation can make to beat-tracking, such that beats are aligned to a previously estimated tempo. Elowsson and Friberg [43] consider tempo annotations useful for automated mixing, e.g., for beat-synchronous delay and compressor release settings. Similarly, Font and Serra [46] mention remixing and browsing as potential applications.

In publications focused on new methods, most application descriptions serve a motivational purpose justifying the conducted research. Often they are presented in a casual, anecdotal fashion. To the best of our knowledge, no formal application survey for tempo estimation has ever been conducted. Neither are we aware of a published user-study with tempo as topic (like, e.g., Lee and Waterman [98]). Therefore we simply do not know *how relevant* tempo estimation is for any of the mentioned applications and what *requirements* these applications have. Rephrased in terms of commercial engineering: for the past 25 years we have largely ignored the customer. Even though this work focuses on tempo estimation, this failure is hardly specific to this particular task. As Salamon [143] recently observed, "There is a disconnect between MIR research and potential users of MIR technologies." This is not to say that the MIR community has conducted the wrong kind of research. After all, it is the privilege of basic research to not require an immediate application, and prefacing each science project with a market study is not expedient. But as tempo estimation and MIR as a whole mature, one might want sound justifications as to why and what for research is conducted.

6.1.2. Presumed Applications

We would like to illustrate the issue with two presumed applications of tempo estimation: similarity and recommendation [186, 128, 6]. By definition, two recordings with the same tempo are similar, but since similarity has many facets, tempo cannot be the only feature used to predict it. It may not even be very important. In fact, in their introduction to music similarity Knees and Schedl [87] briefly mention tempo, but do not deem it important enough to thoroughly discuss it. To quantify how important tempo estimation is for music similarity, we counted the

number of MIREX submissions for the similarity task that used tempo as a feature.[2] Many submissions used low-level temporal or rhythmic features, but only 8 of 62 (13%) explicitly used tempo [10, 11, 100, 101, 50]. One team even removed tempo as feature in a subsequent submission [102].[3]

Music recommendation is another application mentioned when justifying tempo estimation research [186, 128, 6]. But is tempo estimation really useful for recommendation? Content-based systems certainly *can* take advantage of tempo annotations [191], but to the best of our knowledge this is not a common approach. Slaney [176] points out that recommendation based on collaborative filtering usually outperforms content-based systems, if enough usage data is available. Merely in *cold-start* scenarios (e.g., lack of usage data) does content-based recommendation play a noteworthy role. Schedl et al. [149] report that if content-based recommendation is attempted, "almost all existing approaches rely on a number of predefined audio features that have been used over and over again, including spectral features, MFCCs, and a great number of derivatives." This does certainly not exclude tempo, but in their report on current challenges for music recommender research tempo is never mentioned. Therefore, we conjecture that global tempo estimation is only of marginal importance for similarity and recommendation.

6.1.3. Actual Applications

On the positive side, there are plenty of existing applications that are very similar to those stated in the literature. Tempo estimation has been used in computational ethnomusicology [25, 135]. Life science researchers who study connections between exercise and music tempo [192, 82, 39] and athletes who want to control the tempo of their workout naturally benefit from tempo estimation systems. Consumer applications like beaTunes (Appendix D) provide this information via offline analysis, and streaming services like Spotify[4] or Deezer[5] offer playlists with narrow BPM ranges made for runners. The music store Beatport[6] labels all its tracks with global BPM and key values to help DJs when shopping. And when performing, DJs can take advantage of tempo analysis and beat-tracking/matching features of their DJ software (e.g., Traktor[7]). Thus useful applications exist, even though they are typically not the result of user studies or other requirements gathering processes by the MIR community.

[2] MIREX 2006 to 2014.

[3] Because some links on the MIREX website are broken, we were unable to check all 74 distinct submissions. Some teams submitted multiple algorithms in a given year. Re-submissions were ignored.

[4] `https://www.spotify.com/`

[5] `https://www.deezer.com/`

[6] `https://www.beatport.com/`

[7] `https://www.native-instruments.com/`

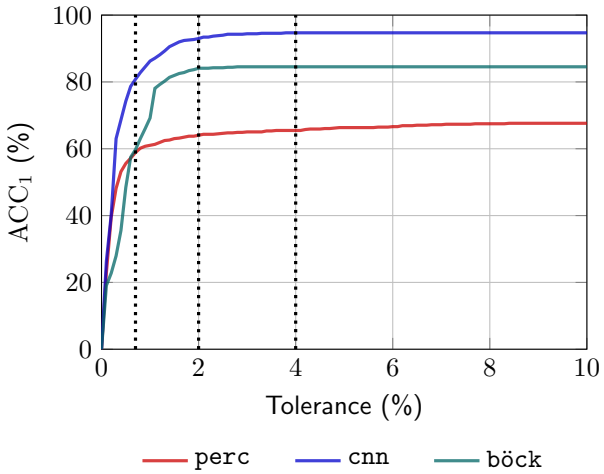


Figure 6.2.: ACC_1 of tempo estimation systems depending on tolerance measured on *Ballroom* with a median IMI-derived ground truth based on Krebs et al. [93].

6.2. Metrics

The exemplary evaluation during the 2004 ISMIR conference effectively established the accuracy metrics ACC_1 and ACC_2 as standards. Few subsequent publications discuss the musical concept of global tempo, but many assume that measuring ACC_1 and ACC_2 is identical to measuring global tempo. *De facto*, the metrics have *become* the task definition [143]. The only popular alternative is the P-Score metric.

6.2.1. Accuracy 1 and 2

ACC_1 computes a 0 or 1 score per track, which indicates the correctness of an estimate, allowing a 4% tolerance. This tolerance is described as "somewhat arbitrary" [64]. It was not chosen, because someone defined an application that required a certain precision, but because it was assumed that the test tracks have "approximately constant tempi." This may have been a good choice for traditionally produced music, but seems lenient for electronic music or music produced with modern production techniques like click tracks [96], because such music rarely contains tempo changes. Attempting to justify the tolerance with limitations of human perception, Gouyon et al. [64] argue that according to Friberg and Sundberg [51] the Just-Noticeable Difference (JND) for music tempi is approximately 4% and therefore "4% is probably the highest precision level that should be considered."

We unfortunately see problems in this argument. First, Friberg and Sundberg's experiment measured, whether participants were able to perceive the non-isochronous placement of the fourth tone in a sequence of six tones. But instead of 4%, they actually found an average JND of 2.5% for tracks with tempi between 60 and 250 BPM. Secondly, and more importantly, it is not

conclusively explained how this experiment relates to determining the tempo of a 30 s sample, as it was the task during the ISMIR 2004 contest. We therefore do not believe that the results of the experiment are suitable to derive the ACC_1 tolerance parameter for tempo estimation metrics. In fact, when plotting ACC_1 for the tempo estimation systems `böck`, `cnn`, and `perc` with different tolerances (Figure 6.2), we see that `cnn` still reaches 93.0% accuracy for the *Ballroom* [64] dataset when reducing the tolerance from 4% to 2% (−1.7 percentage points, pp), and 80% accuracy when reducing the tolerance even further to 0.7%. At 2% tolerance `böck` reaches 84.1%—only 0.4 pp less than for 4%, and `perc` only loses 1.4 pp accuracy. All three systems are capable of estimating tempo for *Ballroom* [64] tracks with almost the same accuracy at 2% tolerance as they are at 4% tolerance.

This points to issues inherent to binary metrics. The threshold is usually arbitrary, because it cannot be derived in an indisputable, objective way. Furthermore, it hides information. ACC_1 does not tell us *how* wrong an estimate is, nor in which *direction*. This means that we cannot easily characterize the quality of estimates with an error distribution or other descriptive statistics. ACC_1 is also blind to small systematic errors, i.e., it is unable to detect a systematic error of +2%, because it lies below the threshold. At the same time, it may overemphasize differences between systems. Systematic errors of +4.01% and +3.99% may not differ much, but their ACC_1 scores could not be further apart. Specifying the tolerance for ACC_1 in percent may also be questioned. Assuming a fictional tolerance of 50%, a recording may be estimated half as fast, but not twice as fast. Contrary to that, estimating a triple meter recording at half its tempo is arguably less appropriate than at twice its tempo [43].

ACC_2 additionally allows estimates to be wrong by the factors 2, 3, 1/2 or 1/3 (so-called *octave errors*). This metrical tolerance was not motivated by application requirements either, but by the realization that the used annotations may not match the perception of human listeners. Unfortunately, because the meter is not taken into account, ACC_2 counts some perceptually erroneous estimates as correct [64]. Consequently, Elowsson and Friberg [43] regard it as "inappropriate." Peeters [126] attempted to alleviate the issue by only allowing justifiable octave errors depending on the track's meter. Another limitation of ACC_2 is that it says nothing about a system's ability to distinguish between *slow* and *fast*. This reduces this metric's usefulness for applications like playlist generation based on tempo continuity or when searching for slow music [127]. Gärtner [53] states: "From the perspective of the user of a DJ software, it is absolutely mandatory that the tempo is annotated correctly. The so-called octave errors are unacceptable." This mismatch between metric and usefulness means that the *construct validity* [188] of ACC_2, i.e., the correlation between use case, success criteria, and the employed metric, is far from perfect *for the mentioned use cases*.

6.2.2. P-Score

A metric that takes metrical ambiguity into account and treats it as inherent property of music [115] is the P-Score proposed by Moelants and McKinney for the MIREX audio tempo extraction task in 2005.[8] The original metric incorporated two metrical levels as well as a phase estimate, and considered an estimation system's salience estimation. In 2006 it was simplified to:

$$P = \mathrm{ST1} * \mathrm{TT1} + (1 - \mathrm{ST1}) * \mathrm{TT2} \tag{6.1}$$

where each track is annotated with two reference tempi, T1 and T2, and T1's relative perceptual strength ST1 $\in [0, 1]$. T1, T2, and ST1 are the result of an expensive process involving many annotators per track. To calculate a P-Score, TT1 $\in \{0, 1\}$ is defined as the ability of an estimation system to identify T1 with a tolerance of 8%. TT2 $\in \{0, 1\}$ is defined correspondingly.[9] In addition to the P-Score, 'One Correct' and 'Both Correct' percentages are published for systems participating in MIREX. For the 'Both Correct' metric it is undefined what happens when a track exhibits no tempo ambiguity. Because P-Score accounts for ambiguity in human perception and does not reward perceptually erroneous estimates, it is an improvement compared to ACC_2, but still has shortcomings. We were unable to find any formal justification for the used 8% tolerance. According to McKinney, "the tolerance was derived empirically through the evaluation of a number of excerpts, algorithms and studies. It is somewhat arbitrary [...]."[10] Furthermore, since 2006 the metric does not require an estimation system to assign a salience value to its two estimates per track.[11] This means that an application using a system with a perfect P-Score still has to guess which of the two estimates is the more salient one. Just like ACC_2, P-Score, as it is used since MIREX 2006, does not test the ability of a system to distinguish between *slow* and *fast*. It also is not efficient in the sense that it is relatively expensive to create the necessary ground truth. This might explain why only one other suitable dataset has been created since the original MIREX dataset in 2005 (Chapter 5).

6.2.3. Tempo Estimation as Classification

Instead of attempting to measure BPM, some approaches to tempo estimation have proposed categorizing music into coarse Perceptual Tempo Classes (PTC), e.g., *very slow*, *somewhat slow*, *somewhat fast*, and *very fast* [20, 19, 127]. Hockman and Fujinaga [72] showed that automatic classification based on global features into just two classes *slow* and *fast* is possible, and Levy [103] proved that humans can also distinguish between *slow* and *fast* tracks very well. While many

[8] `http://www.music-ir.org/mirex/wiki/2005:Audio_Tempo_Extraction`

[9] `http://www.music-ir.org/mirex/wiki/2006:Audio_Tempo_Extraction`

[10] Private correspondence.

[11] Confusingly, the MIREX tempo estimation task requires estimation systems to estimate the salience, but has not used it in any evaluation since 2005.

Dataset	Recordings	Tempo Ann.	Beat Ann.
ISMIR04 Songs [64][1]	464	BPM	No
Ballroom [64, 93][1]	698	BPM	Yes
RWC-C [61][2]	50	BPM	Yes
RWC-G [62][2]	100	BPM	Yes
RWC-J [61][2]	50	BPM	Yes
RWC-P [61][2]	100	BPM	Yes
RWC-R [61][2]	15	BPM	Yes
GTzan [185, 104][3]	999	BPM	Yes
Hainsworth [70][1]	222	BPM	Yes
ACM Mirum [127][1]	1,410	BPM	No
SMC [74][1]	217	BPM	Yes
GiantSteps Tempo [88, 162][4]	664	BPM/T1,T2,ST1	No
Extended Ballroom [105][1]	4,180	BPM	No
LMD Tempo [136, 161][5]	3,611	BPM	No

[1] Excerpts available. [2] Requires application and purchase. [3] Excerpts have been available. [4] Beatport previews have been available. [5] 7Digital previews have been available.

Table 6.1.: Popular public tempo datasets.

approaches using PTCs ultimately aim at improving ACC_1, the motivation and experimental setup for Chen et al. [19] was directly derived from the real-world applications of music navigation and playlist generation. For navigation to the next song, correct PTC classification is desirable, therefore precision and recall for a given collection of songs and queries was used in the evaluation. For playlist generation, lists of songs with perceptually similar tempi were created using test systems and presented to users who were asked to reject songs that do not fit (track-rejection rate).

6.3. Datasets

Evaluation of tempo estimation systems relies on datasets consisting of suitable recordings and annotations that model what we want to measure. Without claim to completeness, Table 6.1 lists popular tempo datasets. Unfortunately, some of these datasets are relatively small, focus on a particular genre, are not freely available (anymore), or have other flaws (e.g., [180]). Salamon [143] states generally: "Existing datasets for MIR are mostly too small, artificial or homogeneous, [...] but we use them anyway!"

6.3.1. Dataset Size

To reliably measure differences between systems, a dataset must be sufficiently large to minimize the effect of random variation due to the sampling of tracks it contains. Generalizability Theory (GT) offers a statistical tool to estimate the required size for performance assessments

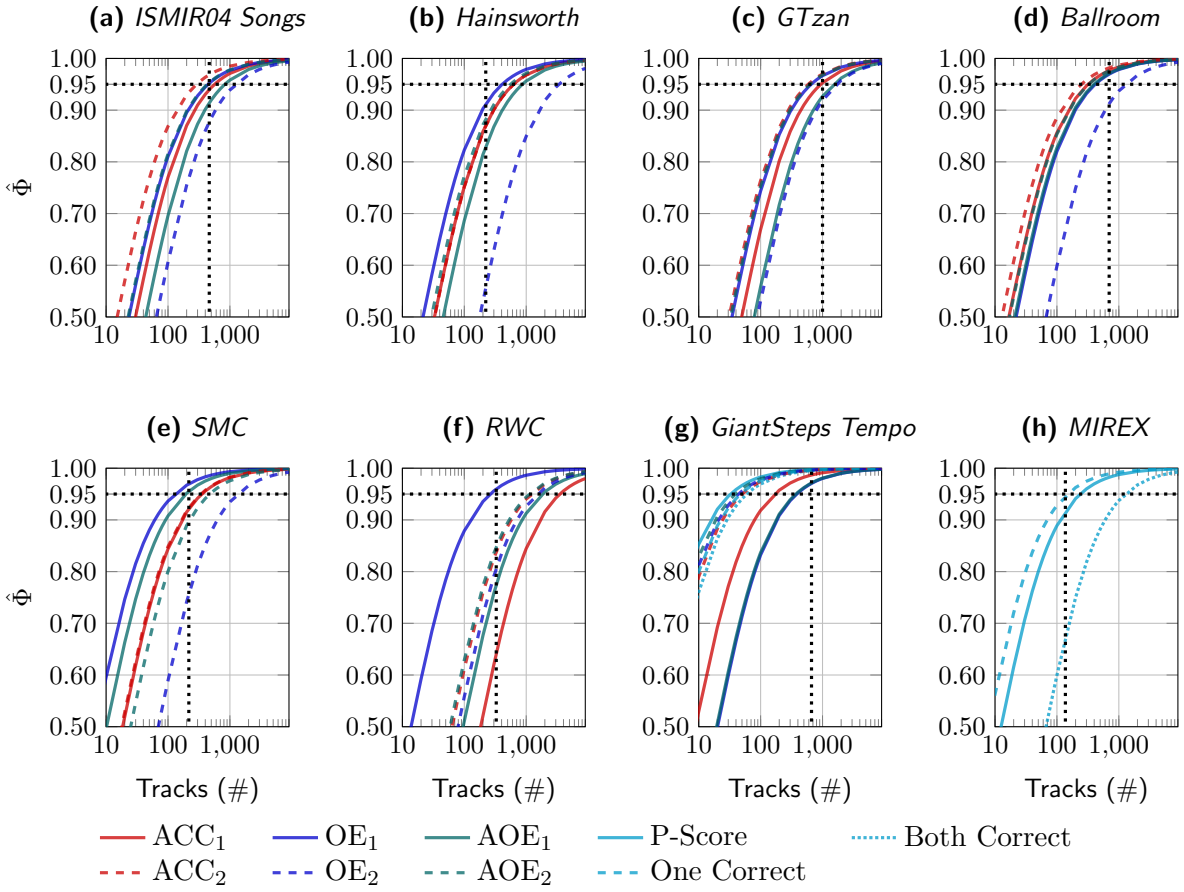


Figure 6.3.: Dependability index $\hat{\Phi}$ as function of metric and track count. Vertical dotted line: actual number of tracks in dataset. Horizontal dotted line: $\hat{\Phi} = 0.95$.
1st row: **(a)** *ISMIR04 Songs*, **(b)** *Hainsworth* (median of corresponding IBIs), **(c)** *GTzan* (median of IBIs), **(d)** *Ballroom* (median of corresponding IBIs), $\hat{\Phi}$ based on estimates by Scheirer [150], Klapuri et al. [86], Davies et al. [29], Oliveira et al. [121], Gkiokas et al. [57], Percival and Tzanetakis [128], Schreiber and Müller [158], Böck et al. [6], Schreiber and Müller [160, 161]; echonest v3.2.1; and zplane auftakt v3.
2nd row: **(e)** *SMC* (median of IBIs), **(f)** combined *RWC* (median of corresponding IBIs), **(g)** *GiantSteps Tempo* [162], $\hat{\Phi}$ based on estimates by Davies et al. [29], Percival and Tzanetakis [128], Böck et al. [6], Schreiber and Müller [160, 161]. **(h)** *MIREX* dataset with $\hat{\Phi}$ based on MIREX 2018 results.

in general [9, 17, 145, 13, 187]. Essentially, the GT framework decomposes the variability in the observed scores into variability due to actual differences between systems (σ_s^2), variability due to differences in track difficulty (σ_t^2), and residual variability (σ_e^2), which often refers to system-track interactions. The total variance of the observed scores is therefore modeled as:

$$\sigma^2 = \sigma_s^2 + \sigma_t^2 + \sigma_e^2. \tag{6.2}$$

An evaluation with high σ_s^2 does not require large datasets, because the evaluated systems are very different to begin with, but evaluations with high σ_t^2 or high σ_e^2 do require large datasets, because systems tend to perform similarly for the given tracks.

There are several coefficients in GT, but here we will report only the dependability index $\Phi \in [0, 1]$, which measures the ratio of system variance to itself plus error variance:

$$\Phi = \frac{\sigma_s^2}{\sigma_s^2 + \frac{\sigma_t^2 + \sigma_e^2}{N}}, \tag{6.3}$$

where N is the size of the dataset. A high Φ-value means that the dataset can reliably separate actual differences among systems from random variation due to sampling of tracks. Φ-values greater than 0.95 are generally considered high enough, but because this is rather arbitrary we focus more on qualitative comparisons among datasets and metrics.

We estimated Φ through an Analysis of Variance (ANOVA) for the datasets *ISMIR04 Songs*, *Hainsworth*, *GTzan* , and *Ballroom* using scores from 12 different systems.[12] To determine how many songs would be necessary for a reliable evaluation, we show $\hat{\Phi}$ as a function of the number of songs N. The actual number of songs in the respective dataset is indicated by a vertical dotted line. Figure 6.3(a-d) shows $\hat{\Phi}$ for ACC_1 and ACC_2 as well as the metrics OE_1, OE_2, AOE_1, and AOE_2, which we will describe in Section 6.6.3. The graphs show that when using ACC_1 for evaluation, *ISMIR04 Songs* and *GTzan* are barely large enough to reach a reliability level ≥ 0.95. For *Hainsworth* $\hat{\Phi}$ is 0.87. Only the *Ballroom* dataset crosses the 0.95 line by a good margin. For ACC_2, $\hat{\Phi}$-values are generally a little higher, but *Hainsworth* again reaches only 0.87. Except for *Ballroom*, OE_1 lets us better recognize performance differences between systems than ACC_1. OE_2 on the other hand leads to less reliable results, indicating that the OE_2 distributions for the tested systems are very similar.

Figure 6.3(e-g) shows $\hat{\Phi}$-curves for the datasets *SMC*, *RWC* (here, the union of *RWC-C*, *RWC-G*, *RWC-J*, *RWC-P*, and *RWC-R*), and *GiantSteps Tempo* and estimates by seven different systems. Just like *Hainsworth*, *SMC* and the *RWC* datasets are not large enough to reach the 0.95 level for ACC_1. With $\hat{\Phi} = 0.97$ for *SMC* and $\hat{\Phi} = 0.96$ for *RWC*, OE_1 delivers again more reliable results. For *GiantSteps Tempo* all considered metrics lead to reliable results, with P-Score requiring the fewest tracks. Lastly, Figure 6.3(h) shows an evaluation of the *MIREX* dataset [112] based on the published MIREX 2018 results.[13] 'One Correct' reaches $\hat{\Phi} = 0.95$, P-Score reaches $\hat{\Phi} = 0.92$, but 'Both Correct' only $\hat{\Phi} = 0.67$. Note that these $\hat{\Phi}$-values depend on the tested systems. Removing older, worse performing systems from the evaluation may actually lower the $\hat{\Phi}$-value.

[12] We used the R package `https://github.com/julian-urbano/gt4ireval/`.

[13] Data from `https://nema.lis.illinois.edu/nema_out/mirex2018/results/ate/mck/files.html`. Based on 137 tracks, since some estimates for three tracks are missing.

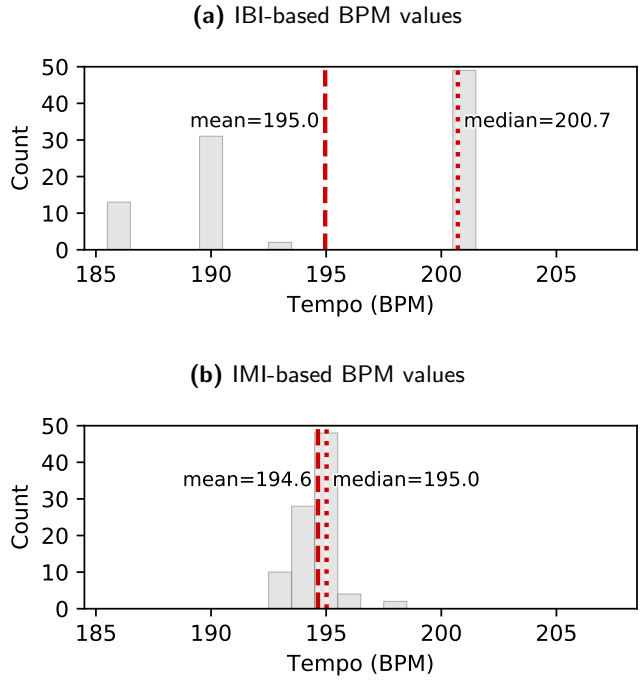


Figure 6.4.: Histograms of BPM values for *GTzan* `jazz.00053` based on **(a)** IBIs and **(b)** IMIs.

We conclude that the majority of the evaluated datasets is large enough to differentiate tempo estimation system performance using ACC_1 and ACC_2. However, *Hainsworth*, *SMC*, and *RWC* do not allow differentiating systems as reliably as other datasets, and the *MIREX* dataset is barely large enough for P-Score, but not for 'Both Correct.'

6.3.2. Dataset Quality

Ten years after *Ballroom* had been used for the first time, Percival and Tzanetakis [128] investigated the accuracy of the annotations and corrected 32 (4.6%) of them. Corrections were also made to *ACM Mirum* (135, 9.6%) and *GTzan* (24, 2.4%). Interestingly, Percival and Tzanetakis emphasize the importance of using correct annotation, because testing systems on faulty data may lead researchers to optimize for these errors, i.e., to introduce a bias based on the test set. This mindset is indicative for the state of MIR at the time. Machine learning was not ubiquitous yet and tuning hyperparameters using test sets was not perceived as the methodological faux-pas it is seen as now. But there are other good reasons to strive for quantifiable quality in test datasets: *interpretability* and *comparability*. If the quality of a test dataset is unknown, a metric like accuracy can at best be used for ranking or to approximate the lower bound of a system's true performance. At worst it is simply useless. It is impossible to say whether any changes to the system can still increase performance, i.e., one cannot know when a task is solved. Metrics produced on flawed datasets can lead to ill-informed actions—e.g., an accuracy of 58% on a

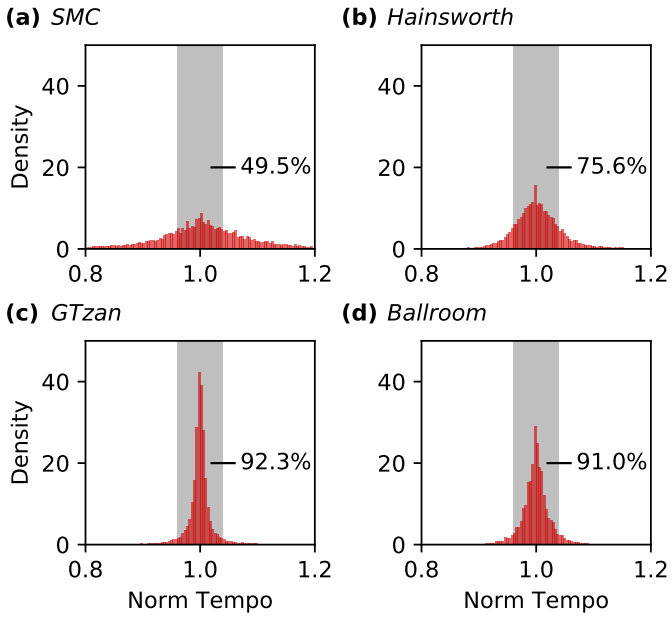


Figure 6.5.: Distributions of normalized tempi. The gray area marks the interval $[0.96, 1.04]$. The shown percentage is the fraction of normalized tempi within the interval.

dataset with only 60% correct data may cause a researcher to abandon a method, if the dataset accuracy is naïvely assumed to be perfect. Additionally to this lack of interpretability, it is impossible to compare results for different datasets in a meaningful way, if the dataset quality is unknown. As we have seen, when comparing the performance of `böck` on the original *GiantSteps Tempo* annotations with the performance in the new annotations derived in Chapter 5, ACC_1 jumps from 58.9% to 64.8% and ACC_2 from 86.4% to 94.0%.

Because many subjects participated in the crowdsourced experiment, we were able to model perceived tempo ambiguity and derive a P-Score-style ground truth. However, this may have come at the price of precision. Beat annotations created by manual tapping always contain small inaccuracies. Input method latencies and anticipatory early tapping—known as negative mean asynchrony (NMA) [140]—lead to precision problems (see also Cornelis et al. [25]). Some researchers (e.g., Holzapfel et al. [74]) therefore choose to annotate based on cues visible in spectrograms using tools like SonicVisualizer by Cannam et al. [16] or other representations derived from the waveform of the music recording. Another way to avoid annotation errors is synthetic dataset creation.

6.3.3. Modeling Global Tempo

It is well known that some of the tracks in popular datasets have varying tempi [70, 126, 128]. To address this issue, Hainsworth defined the tempo for the tracks in his dataset as the mean of

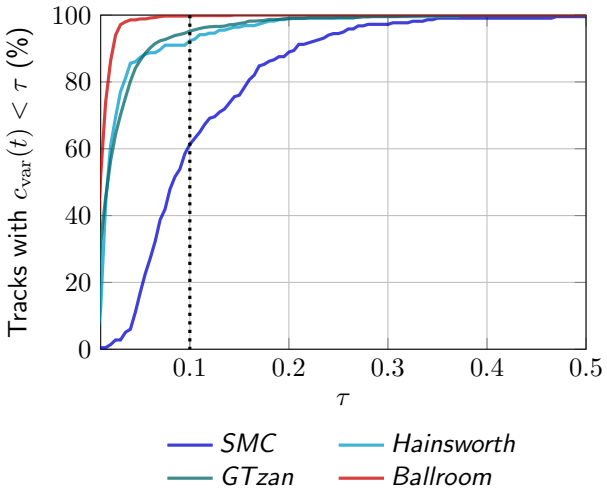


Figure 6.6.: Percentage of tracks with $c_{\text{var}}(t) < \tau$. Values for *GTzan*, *Ballroom*, and *Hainsworth* are based on IMIs, *SMC* values are based on IBIs (no IMIs available).

the inter-beat intervals (IBI). Percival and Tzanetakis [128] suggested using the median instead, to counter the influence of outliers—an idea already used by Peeters [126] and Oliveira et al. [121] to deal with non-constant tempo estimates. Böck et al. [6] followed this suggestion, and derived annotations based on the median IBI, for all beat-annotated test sets they used, but to the best of our knowledge did not publish their annotations. Subsequent publications still used the original mean-based annotations (Section 3.2) or tempo values obtained in some other way. For example, Elowsson [42] derived tempi from the peaks of smoothed IBI histograms.

In addition to changing tempi, some datasets [104, 70] contain recordings with swing. One may argue that for such recordings neither the mean nor the median IBI is an ideal solution, because by definition swing beats are non-isochronous, i.e., the time interval from a downbeat to the next backbeat is not the same as from a backbeat to the next downbeat. As a result, one may see multiple peaks in an IBI histogram. For example, the IBI-based BPM histogram for the *GTzan* recording `jazz.00053` (Figure 6.4a) shows distinct peaks at 186, 190 and 201 BPM even though the tempo of the track does not change over time (beat annotations by Marchand and Peeters [104]). Choosing the median of the IBIs (200.7 BPM) ignores the lower peaks at 186 and 190 BPM even though they are no outliers. If we know a track's meter, we therefore may rather use the median of the intervals between *corresponding* beats, i.e., an inter-measure interval (IMI), and use it to derive the tempo, thus neutralizing swing as well as outliers (Figure 6.4b).

6.3.4. Dataset Suitability

While improving and versioning annotations is commendable, it does not ensure that the dataset fits the use case. Obviously, if the use case focuses on Ballroom music, using a Jazz dataset for

testing is the wrong approach. Similarly, if a metric is chosen that was designed for a certain kind of music, one must use this kind of music. As pointed out above, a precondition for using ACC_1 and ACC_2 with 4% tolerance is a stable tempo in each test track. This precondition is not met for all tracks in *SMC*, *Hainsworth*, *GTzan*, and *ISMIR04 Songs* [128]. Additionally, Peeters [126] remarks that both Classical and Flamenco tracks in *ISMIR04 Songs* have variable tempo. We can visualize this by converting IBIs to normalized tempi and plotting their distribution. Concretely, given a track's IBIs $b = \{b_0, b_1, ..., b_{N-1}\}$ in seconds with $b_n \in \mathbb{R}_{>0}$, we define its (local) tempo values $t = \{t_0, t_1, ..., t_{N-1}\}$ in BPM as

$$t_n = \frac{60}{b_n}. \tag{6.4}$$

The normalized tempi $t^{\mathrm{norm}} = \{t_0^{\mathrm{norm}}, t_1^{\mathrm{norm}}, ..., t_{N-1}^{\mathrm{norm}}\}$ for *each track* are then defined as:

$$t_n^{\mathrm{norm}} = \frac{t_n}{\frac{1}{N}\sum_{i=0}^{N-1} t_i} \tag{6.5}$$

Figure 6.5 depicts distributions of t^{norm} for all tracks in *SMC*, *Hainsworth*, *GTzan*, and *Ballroom*. For *SMC*, only half the normalized local tempi fall into the ±4% interval $[0.96, 1.04]$ (shown in gray). For *Hainsworth*, it is 75.6%. *GTzan* and *Ballroom* have values of 91% or more and are thus much better suited for $\mathrm{ACC}_1/\mathrm{ACC}_2$.

To get an impression of how many tracks in a dataset have large tempo variability, we can use the standard deviation of the normalized tempi $\sigma(t^{\mathrm{norm}})$—also known as the coefficient of variation c_{var} defined in Equation (5.1):

$$c_{\mathrm{var}}(t) = \frac{\sigma(t)}{\mu(t)} = \sigma(t^{\mathrm{norm}}). \tag{6.6}$$

Figure 6.6 shows the percentage of tracks for which $c_{\mathrm{var}}(t) < \tau$ with $\tau \in [0, 0.5]$. Among the shown datasets, *SMC* contains the highest percentage of tracks with large tempo variability.[14] For only 61.3% of the tracks is $c_{\mathrm{var}}(t) < 0.1$. In contrast, $c_{\mathrm{var}}(t)$ is less than 0.1 for 99.7% of all *Ballroom* tracks. This affects accuracy. To demonstrate, we measure ACC_2 using `böck`, `cnn`, and `perc` for subsets of the datasets containing only tracks with $c_{\mathrm{var}}(t) < \tau$, $\tau \in [0, 0.5]$.[15] The used tempo annotations are based on median IMI-values for *GTzan*, *Hainsworth*, and *Ballroom*, and based on median IBI-values for *SMC*. For *SMC* (Figure 6.7a), all three systems reach higher scores at $\tau = 0.1$ than for greater τ. Comparing ACC_2 for $\tau = \infty$ to $\tau = 0.1$, accuracy increases for `böck` from 67.3% to 85.7% by 18.4 pp, for `cnn` from 47.9% to 59.4% by 11.5 pp, and for `perc` from 45.6% to 55.6% by 10.0 pp. For *Hainsworth* (Figure 6.7b) the systems also achieve higher

[14] This is also true when calculating $c_{\mathrm{var}}(t)$ for the other datasets with IBIs.

[15] ACC_2 is appropriate, because the question of suitability does not hinge on octave errors.

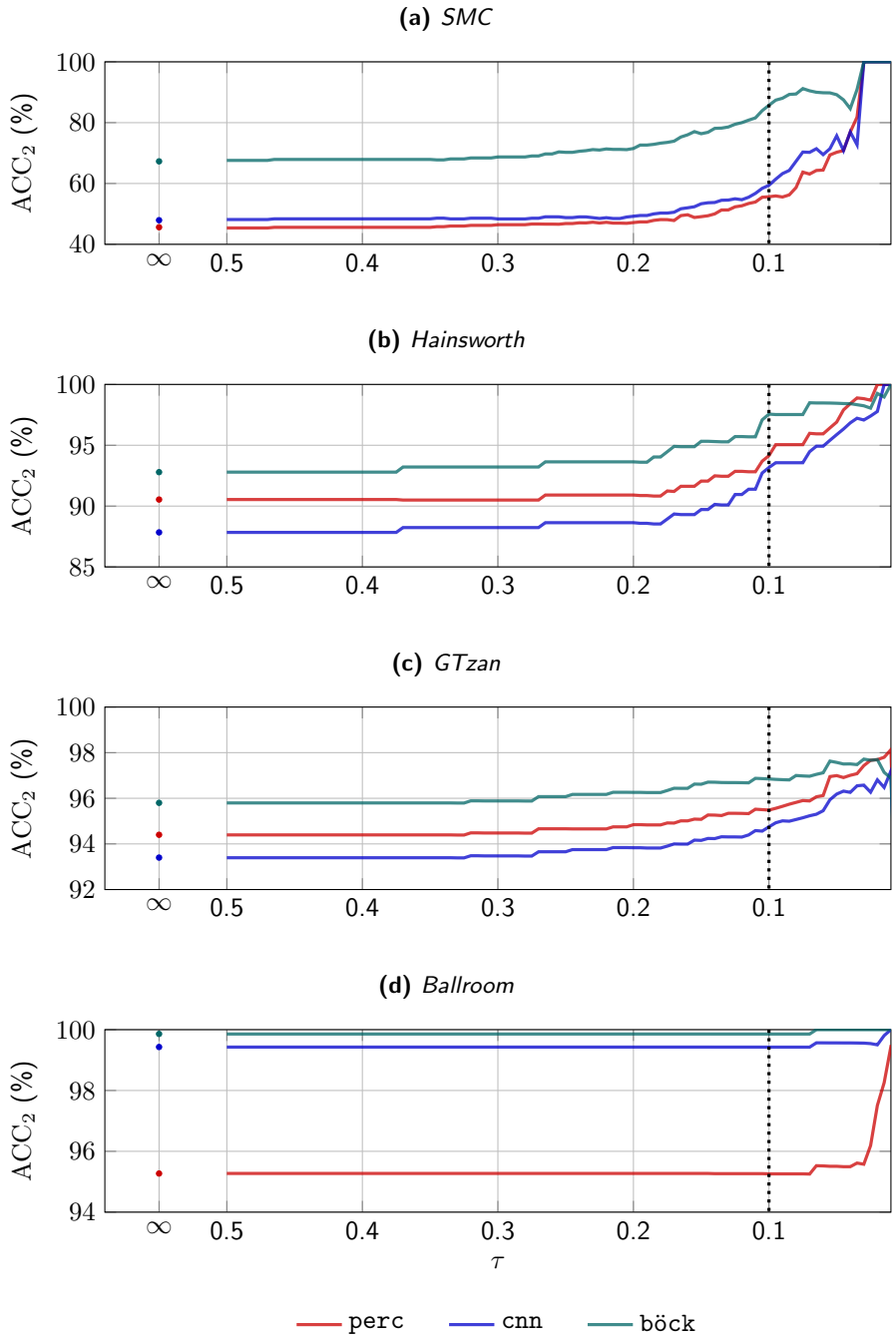


Figure 6.7.: ACC_2 for tracks with $c_{var}(t) < \tau$. Lower τ coincides with higher accuracy. Datasets: **(a)** *SMC* **(b)** *Hainsworth* **(c)** *GTzan* **(d)** *Ballroom*. Different y-scales used for clarity.

scores at $\tau = 0.1$, but not as much in absolute numbers. ACC_2 for `böck` increases by +4.8 pp to 97.6%, `cnn`'s ACC_2 increases by +5.4 pp to 93.2%, and `perc`'s increases by +3.6 pp to 94.1%. For *GTzan* (Figure 6.7c) the increase is still a little smaller, and for *Ballroom* (Figure 6.7d) there is none, because almost all tracks have small $c_{var}(t)$. Using $c_{var}(t) = 0.1$ as a lenient threshold

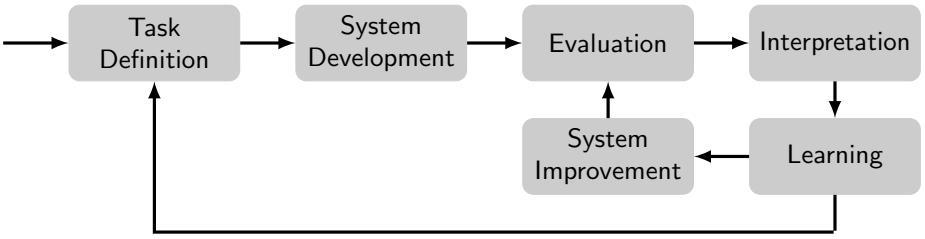


Figure 6.8.: IR research cycle [188]

for "stable tempo" reveals that of the four datasets only *Ballroom* is suitable for ACC_2 (and thus ACC_1) without restriction.

6.4. Research Cycles

Evaluation allows us to acquire new knowledge and advance the field [171, p. 31], if experiments are followed by interpretation of results, learning, system improvement, and eventually re-evaluation or even re-definition of the task or the evaluation methodology (Figure 6.8). This is referred to as *research cycle* [188, 182]. For it to succeed, we need to be able to conduct analyses of all parts of the evaluation process: task definition, data, metrics, systems, and analysis. As has been pointed out before [188, 179, 137, 144], this disqualifies evaluation campaigns with private or secret data and closed source evaluation code.

Evaluation itself must follow the same cycle of learning as general MIR research. How we evaluate must be analyzed, questioned, and improved [188, 171, p. 33]. Do datasets and metrics match current use cases? Are there recordings for which no system estimates the correct tempo, or recordings most systems estimate different tempi for? Does that mean the annotation is wrong, the tempo is hard to estimate, or the recording is not suitable for the task? To become aware of and address these issues, we need versioned annotations and publicly archived estimates to develop better evaluation methods and conduct error analysis (see Section 6.6). This would allow us to easily identify problematic samples as well as re-evaluate old estimation systems with improved annotations and changed metrics.

Another learning obstacle is oversimplified evaluation metrics. Single value figures effectively hide the complex details of success and failure behind an opaque number. For tempo estimation it is interesting whether a system performs well for certain genres, but not for others, or always fails for tempi below a certain threshold. And what about a cappella or violin recordings with very soft onsets? To foster learning, we must paint a detailed picture that supports researchers in discovering what it is that makes systems work properly for one recording but not for another (e.g., Grosche et al. [69]).

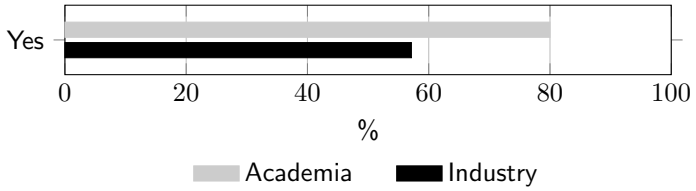


Figure 6.9.: Question: Given its limited usefulness for tracks with varying tempo and the good performance of current beat trackers, is global tempo estimation as a music information retrieval task still relevant?

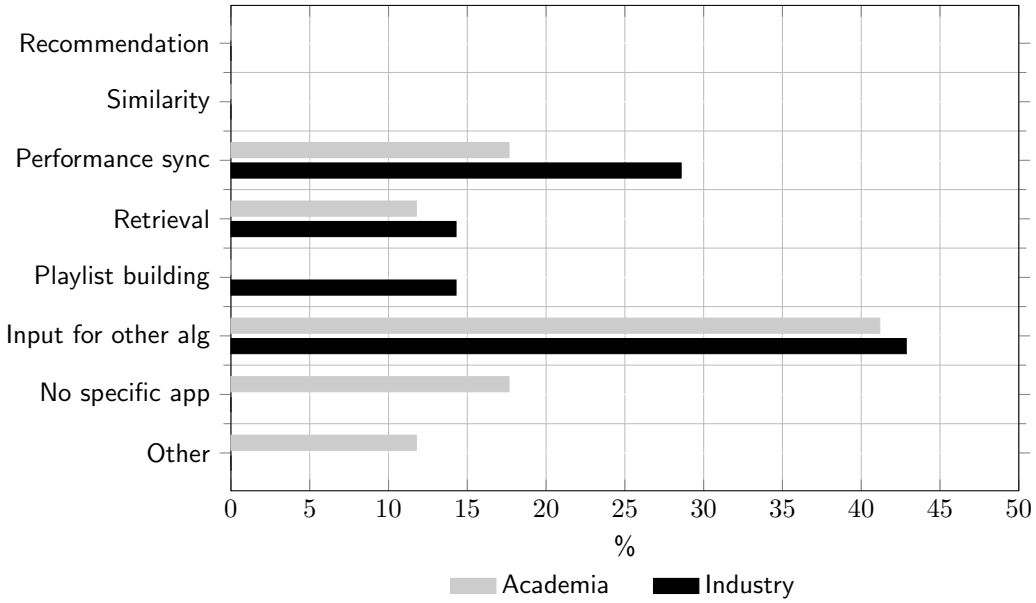


Figure 6.10.: Question: Which main application is your tempo estimation system intended for (single-choice)?

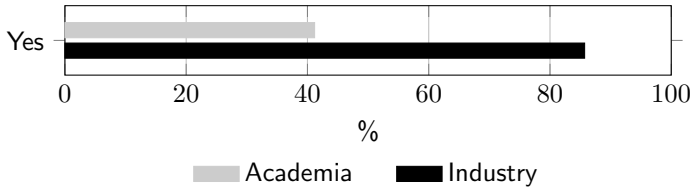


Figure 6.11.: Question: Does your tempo estimation system target specific genres?

6.5. Survey

To answer some of the questions raised in Sections 6.1 and 6.2, we have conducted a survey among scientists and practitioners who have worked on tempo estimation. Invitations to fill out the survey were sent to the ISMIR community mailing list, known industry players, and MIREX tempo estimation task participants. Given that the field is very small, the presented results must be interpreted qualitatively. All respondents have agreed to anonymized publication of their answers. In the following we summarize the most important results. Note that not all

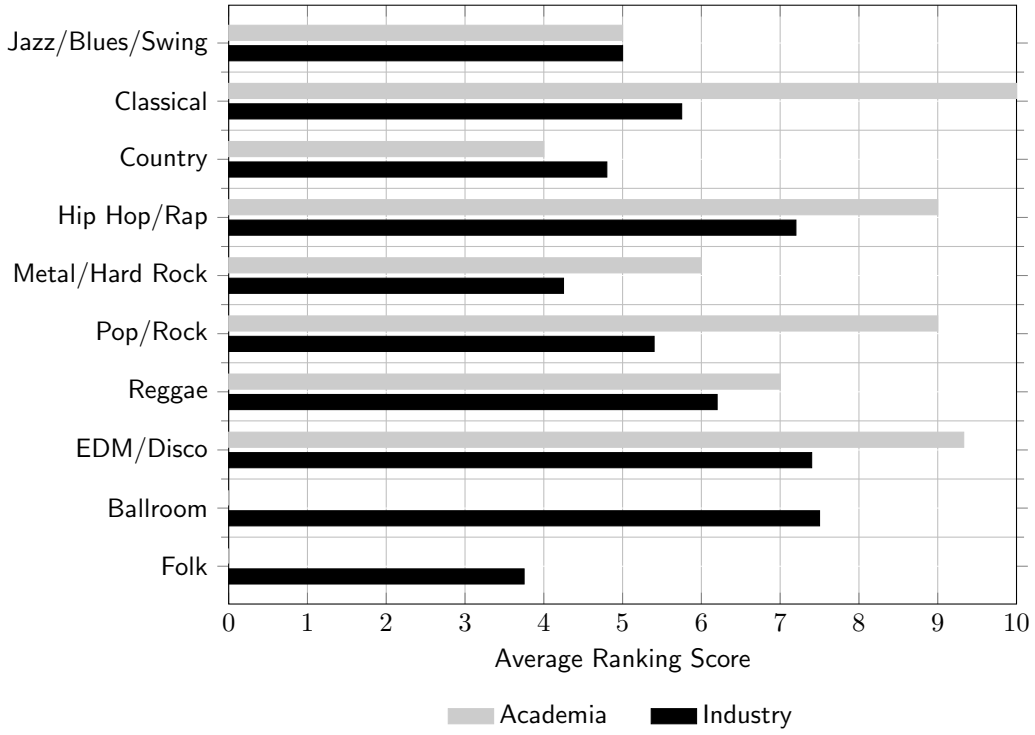


Figure 6.12.: Task: Please rank the following genres by importance for your main application.

participants answered every question. Stated percentages are over the number of responses to a question, not overall participants.

6.5.1. Participants

Of the 24 individuals who filled out the questionnaire, 17 (71%) belonged to academia ■ and 7 (29%) to the industry ■. Most participants identified themselves as researchers (92%), and a majority claimed to be involved in hands-on algorithm implementation (71%).

6.5.2. Relevance as MIR Task

Given its limited usefulness for tracks with varying tempo and the good performance of current beat trackers, we asked whether global tempo estimation as an MIR task is still relevant (Figure 6.9). 16 of the 22 (73%) participants who answered the question answered with *yes*. Positive arguments included: "useful input for other applications (recommenders, beat-trackers)", "tempo tracking at sub-decimal place", "first step of hierarchical analysis", and "customers require to see a single value." Negative reasons (doubting the relevance) included: "tempo estimation is good enough for most industrial use cases", "local tempo estimation is a much more useful task", and "beat tracking, as a more general task than tempo estimation, solves all problems." Three

of the six participants with doubts are from the industry. One respondent perceived the question as leading towards a *no* answer and another did not agree with the premise that beat trackers have good performance.

6.5.3. Application

When asked about the main application for their tempo estimation system, 10 participants (42%) chose "input for other algorithms" and 5 (21%) chose "performance synchronization." Figure 6.10 depicts answers split by academia/industry. Lending support to our argument from Section 6.1.2, no one chose "recommendation" or "similarity" as the main application.

6.5.4. Genres

Six of seven (86%) industry members indicated that their tempo estimation system targets specific genres in the sense that some genres are more important than others for their application (Figure 6.11). In academia only 7 of 17 (41%) respondents target specific genres. We believe this reflects a tendency in the industry to have a more specific understanding of customer needs, while academia often concentrates on basic research without a marketable product in mind.

We asked those who target specific genres to rank a list of 10 prechosen genres by importance, allowing the value N/A for genres they are not interested in. Figure 6.12 depicts the results (in this depiction N/A values were ignored and did not affect the average ranking score). Members of academia have a preference for Classical, EDM/Disco, Hip Hop/Rap and Pop/Rock, while industry members regard Ballroom and EDM/Disco as most important, followed by Hip Hop/Rap, Reggae, Classical and others.

6.5.5. Slow, Fast, and Ambiguous

We asked participants how important it is for their main application to correctly distinguish between *slow* and *fast* using a 5-point Likert response format ranging from "not at all" to "essential" (Figure 6.13). 12 out of the 19 (63%) participants chose "essential." Three members of academia chose "neutral", while all industry respondents chose "essential" or between "neutral" and "essential". No one chose less than "neutral." Clearly, distinguishing between *slow* and *fast* is important. When asked whether their application requires a single BPM value or a tempo distribution, 5 of 7 (71%) industry respondents answered "single value", while 6 of 14 (43%) academic scientists answered "tempo distribution" (Figure 6.14). This is also reflected in the usefulness participants assigned to the metrics ACC_1, ACC_2, and P-Score (Figure 6.15). While ACC_1 and ACC_2 are somewhat more appreciated by the industry than by academia, P-Score,

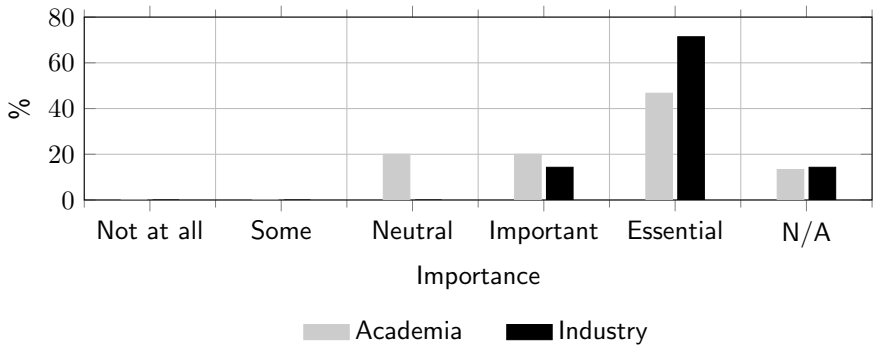


Figure 6.13.: Question: How important is it for your main application to correctly distinguish between slow and fast music?

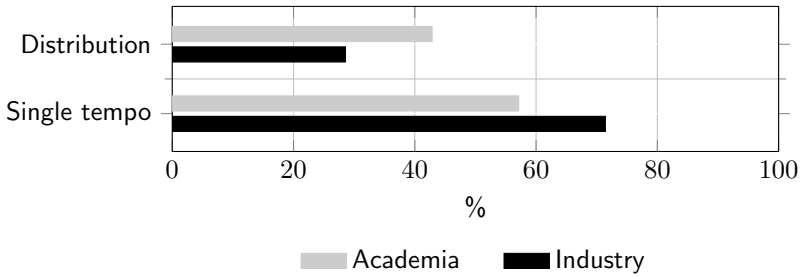


Figure 6.14.: Question: Does your main application require a single BPM value or a tempo distribution?

which embraces the idea of tempo ambiguity, is much less favored by the industry than by academia. While 47% of the academics believe P-Score is "essential", none of the industry members believe so. Instead, 57% state it is "not useful" at all or between "not useful" and "neutral."

6.5.6. Accuracy Tolerances

Since the 4% tolerance of ACC_1 was never justified by an application, we asked what accuracy the participants' applications require. Given several relative and absolute choices, and "Other", "Other" was most popular among academics. Answers ranged from "BPM with as small as possible tolerance" over "no specific application yet" to "depends on the dataset and the accuracy of the annotations." The second most popular choices among academics were "nearest integer" and "2% tolerance." Industry members often chose "2 decimal places" (Figure 6.16). Interestingly, neither "nearest integer", "2% tolerance", nor "2 decimal places" is commonly used in the literature.[16] Of the five participants who chose performance synchronization as their main application regardless of affiliation, one chose "1% tolerance", three chose "2 decimal places" and one chose "3 decimal places", documenting requirements far stricter than the commonly used

[16]With the exception of our own work presented in Chapter 4, which also uses "nearest integer."

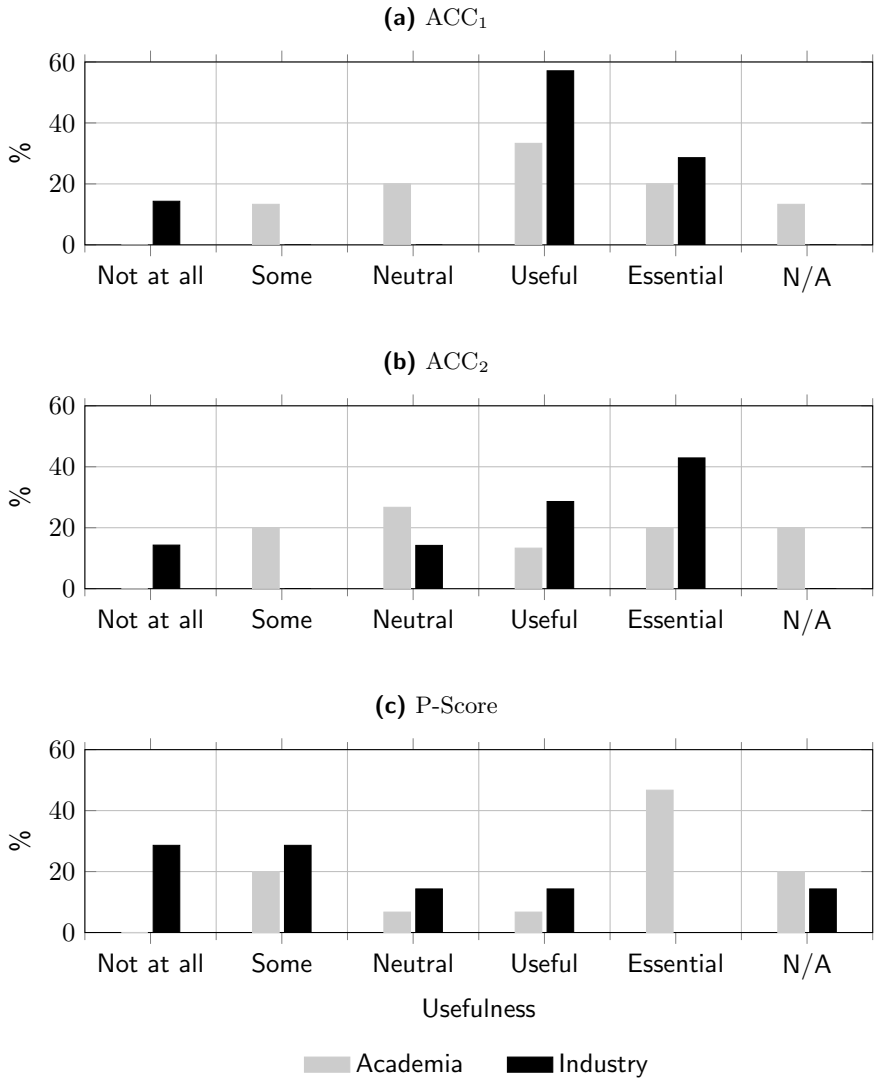


Figure 6.15.: Question: How useful are the following metrics as success indicators for meeting your main application's requirements or reaching your research goals?

4%. Regardless of application or affiliation, no one chose 8%—the tolerance traditionally used at MIREX.[17]

Additionally to our survey among scientists we conducted an informal Twitter poll among users of the consumer application beaTunes. The poll was advertised to people interested in working out, indoor cycling, ballroom dancing, and DJing. Participants were asked "What tempo/BPM detection accuracy do you really need?" and given four choices ranging from 0 to 3 decimal places. 72% of the 97 respondents answered that a 0 or 1 decimal place precision is sufficient for their purposes (Figure 6.17). Some people commented that even a 10% precision is good enough for their application.

[17] In 2018, an additional evaluation with 4% was conducted.

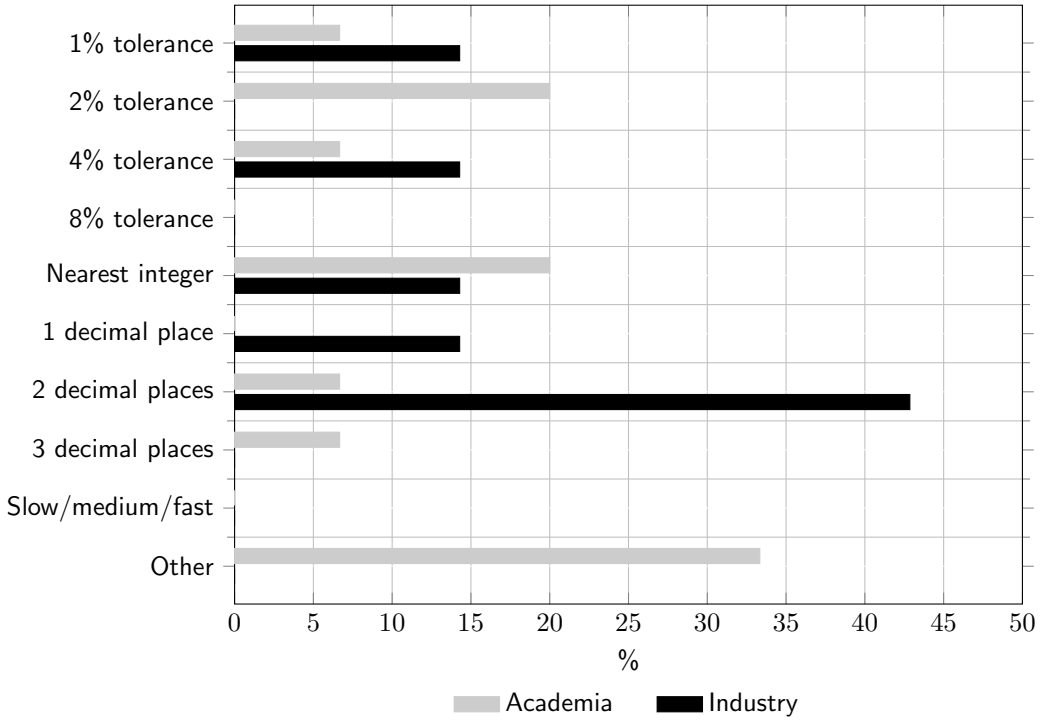


Figure 6.16.: Question: What tempo estimation accuracy do you require for your main application (single-choice)?

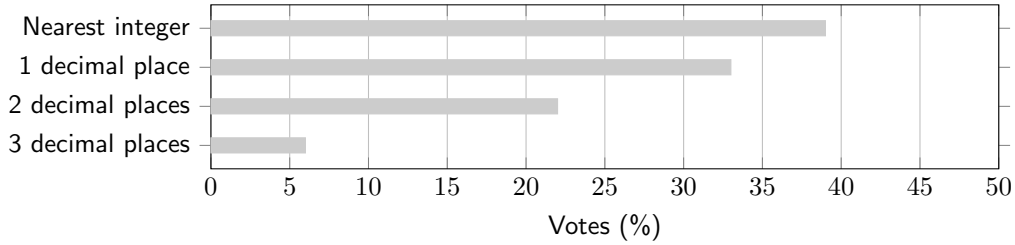


Figure 6.17.: Twitter poll: What tempo/BPM detection accuracy do you really need (single-choice)? Given four choices, most of the 97 participants were content with nearest integer or 1 decimal place precision.

6.6. Proposal

To address some of the issues raised in Sections 6.2, 6.3, and 6.4, we have created a public GitHub repository called *tempo_eval* (`https://tempoeval.github.io/tempo_eval/`) that hosts different versions of corpus annotations (Section 6.6.1), estimates for these corpora (Section 6.6.2), and evaluation code that goes beyond single figure binary metrics (Section 6.6.3). Section 6.6.4 demonstrates how the repository can be used for evaluation.

6.6.1. Reference Annotations

The *tempo_eval* repository allows the continuous improvement of reference annotations without shadowing past versions. This makes it possible to evaluate against all reference versions, improving comparability to older published results and thus transparency as well as interpretability. To provide easy access to reference data we converted published annotations to JAMS [77] for which tools already exists [137].

6.6.2. Estimated Annotations

Rather than just serving as a static source of reference data, the *tempo_eval* repository offers a place for researchers to publish and archive their algorithms' estimates instead of just mentioning single value metrics in their publications. This allows re-evaluation with new reference annotations and proper development of new metrics, which may ultimately lead to a better understanding of tempo estimation systems and the tempo estimation task. For example, Figure 6.18 shows values for a proposed metric (Section 6.6.3) for historic estimates measured against a ground truth, which has been newly derived from median IMI-values. Because the repository is open and public, contributing is easy, e.g., via pull requests. As a starting point, we have added estimates by several recent and classic systems for commonly used datasets.

6.6.3. Formal Octave Error

We have argued in Sections 6.2.1 and 6.2.2 that the tolerances of ACC_1, ACC_2, and P-Score are difficult to justify and that the binary nature of these metrics hides information. Furthermore, using a percentage as threshold is sub-optimal. We therefore encourage the use of an alternative metric that measures how close and in which direction an estimate is to a reference value. Inherently, this kind of metric supports meaningful visual depiction of error distributions. Since tempo perception and metrical levels follow a hierarchical structure, we suggest using a $\log_2$-scale, so that the error for double- and half-tempo has the same magnitude. A metric that combines these ideas is a formalized version of the familiar octave error and has previously been used for illustrative purposes in [64, 126]. We formally define the octave error OE_1 as

$$\mathrm{OE}_1(y, \hat{y}) = \log_2 \frac{\hat{y}}{y}, \tag{6.7}$$

with $y, \hat{y} \in \mathbb{R}_{>0}$ as ground truth and estimate. OE_1 is designed to highlight the most important error class, octave errors, in an intuitive way. Errors by factors k and $1/k$ have the same magnitude, which means that in an OE_1 visualization the octave errors 2, $1/2$, 3, and $1/3$, are easily identifiable as clusters around 1, -1, 1.58, and -1.58. Figure 6.18a shows examples for OE_1

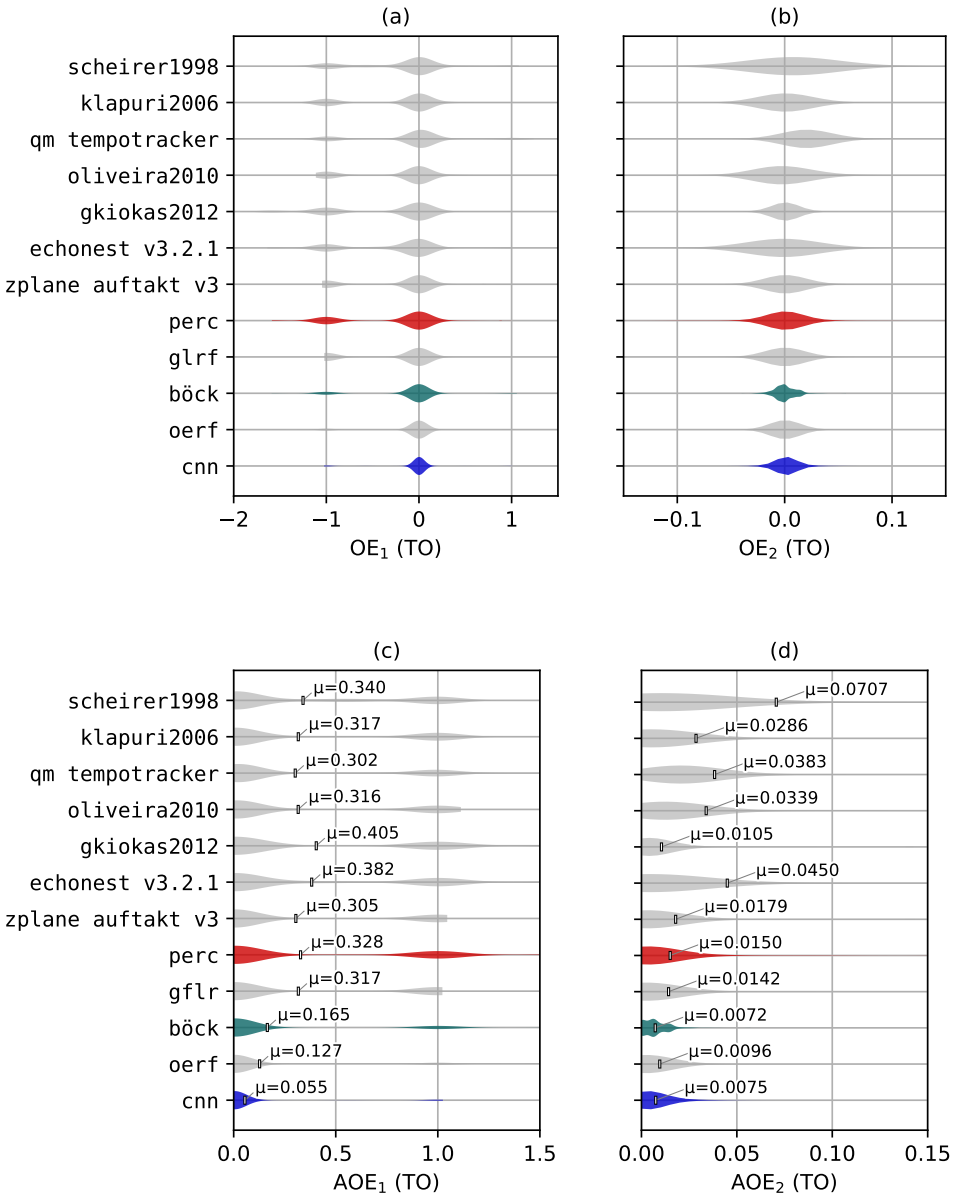


Figure 6.18.: Empirical distributions of **(a)** OE_1, **(b)** OE_2, **(c)** AOE_1, and **(d)** AOE_2 using kernel density estimation (KDE). Based on values measured for *Ballroom* using a median IMI-derived ground truth created from beat annotations by Krebs et al. [93]. Ordered by year of publication [150, 86, 29, 121, 57, 128, 158, 6, 160, 161].

distributions for *Ballroom* rendered as violin plots.[18] Clearly visible is the concentration around −1 tempo octaves (TO) for all systems but `böck`, `cnn`, and `oerf`. The extend of the horizontal spread of the concentrations around 0 TO visualizes non-octave errors. OE_1 distributions can

[18] We were unable to obtain either an implementation of the approach taken by Elowsson [42] or estimates of his system for the *Ballroom* dataset.

serve as indicator for the overall performance of a global tempo estimation system including the capability to distinguish between *slow* and *fast*. Most importantly, one can see at a glance what kind of errors the tested systems are prone to.

We have seen in Section 6.2.2 that taking metrical ambiguity into account is desirable, but that suitable datasets are rare and new datasets are expensive to create. For these pragmatic reasons, we do not attempt to solve the metrical level problem, but define OE_2 similar to ACC_2 as

$$\mathrm{OE}_2(y, \hat{y}) = \arg\min_{x \in \Omega}(|x|), \tag{6.8}$$

with

$$\begin{aligned} \Omega := \{ &\mathrm{OE}_1(y, \hat{y}), \mathrm{OE}_1(y, 2\hat{y}), \mathrm{OE}_1(y, 1/2\hat{y}), \\ &\mathrm{OE}_1(y, 3\hat{y}), \mathrm{OE}_1(y, 1/3\hat{y})\}. \end{aligned} \tag{6.9}$$

OE_2 (Figure 6.18b) measures accuracy on a micro level, where the most common errors on the metrical level are ignored, i.e., it measures how close the estimate is to the nearest plausible tempo.[19] This is useful for genres with high metrical ambiguity, e.g., Dubstep (Section 5.2.5), and for applications that require errors to be as small as possible. The latter is a use case not currently supported by either ACC_1 or ACC_2, but apparently implemented by the industry (Section 6.5.6).

While the mean of OE_1 or OE_2 indicates whether an algorithm is expected to over- or underestimate the tempo, the *absolute* octave error ($\mathrm{AOE} = |\mathrm{OE}|$) can be used for a system comparison with respect to precision. To illustrate, Figure 6.18c shows annotated AOE_1-distributions. Most older systems have an average AOE_1 between 0.3 and 0.4 TO, `böck` managed to halve this figure and `cnn` further reduced it to 0.055 TO. When ignoring octave errors by using AOE_2 (Figure 6.18d), we can see that `böck` and `cnn` perform on a similar level.

6.6.4. Evaluation

The *tempo_eval* repository also contains evaluation code. Implemented are ACC_1, ACC_2, and P-Score, along with McNemar's test for significant differences for ACC_1 and ACC_2, OE_1, OE_2, their corresponding absolute incarnations, and t-tests for estimates from algorithm pairs. Results can be rendered in a publishable report (Markdown/HTML), and figures and data are exportable in several formats. As argued in Section 6.4, reporting single value metrics is not sufficient for an in-depth evaluation. We have therefore implemented visualizations for system performance depending on tolerances (Figure 6.2), tempo stability (Figure 6.7), tempo range (Figure 6.19), and—if available—genre- or free-form-tags (Figure 6.20). As an example, we will discuss tempo-

[19] The criticism voiced about ACC_2 in Section 6.2.1 obviously applies to OE_2 as well.

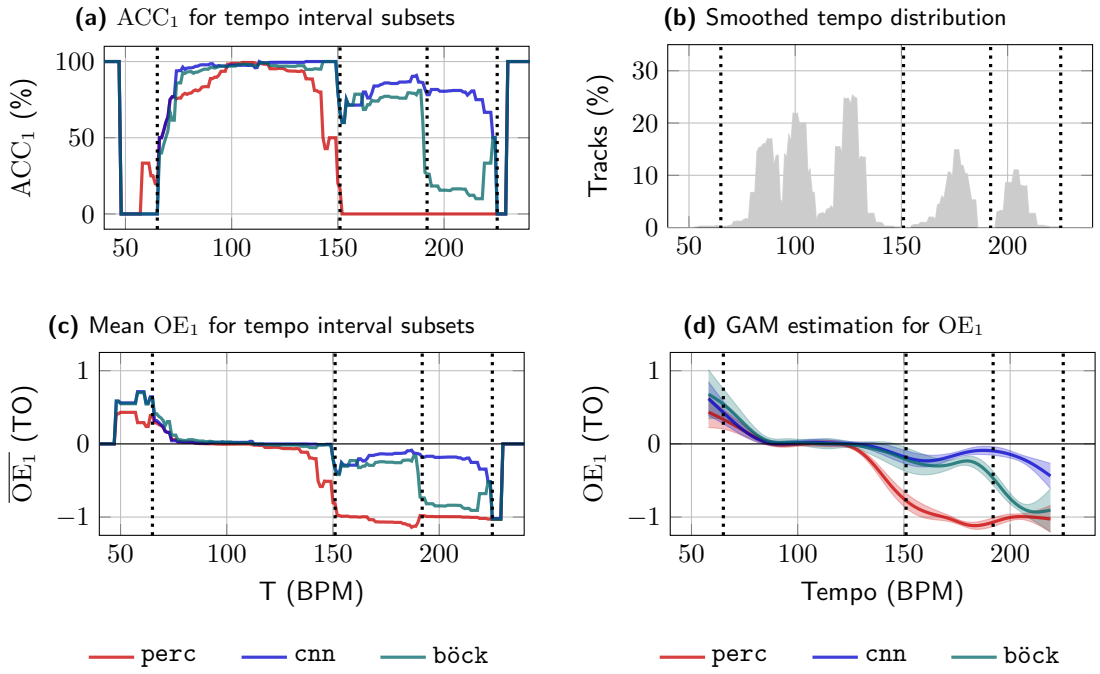


Figure 6.19.: **(a)**, **(c)** ACC_1 and mean OE_1 for $T \pm 10$ BPM intervals. **(b)** Smoothed tempo distribution of tracks in *Ballroom* according to the ground truth from Percival and Tzanetakis [128]. **(d)** OE_1 predictions of generalized additive models (GAM). Shaded areas correspond to 95% confidence intervals.

and genre-dependent evaluation using the *Ballroom* dataset with annotations from Percival and Tzanetakis [128].

Figure 6.19a shows ACC_1 values for subsets defined by tempo ranges $[T - 10, T + 10]$ BPM. Clearly visible, `perc`'s ACC_1 drops to zero for $T > 150$ BPM, and `böck`'s ACC_1 sharply decreases to 27.3% or less for $T > 190$ BPM. Both systems seem to exhibit some form of octave bias (Section 3.2.1.3), i.e., the ability to estimate the tempo appears tied to certain tempo ranges. Figure 6.19c depicts mean OE_1 values for the same scenario and shows what kind of errors lead to the observed low accuracy. Apparently, `perc` suffers from octave errors of -1 TO for $T > 150$ BPM. The same is true for `böck` and $T > 190$ BPM. None of the systems seem do well for tracks with $T < 66$ BPM or $T > 225$ BPM, but as we can see in Figure 6.19b, the dataset contains only very few songs in these tempo ranges. Figure 6.19d combines error magnitude, error direction, and significance in a single graph. It shows the predictions and their 95% confidence interval of generalized additive models (GAM)[20] fitted on the respective systems' OE_1 results. A large confidence interval indicates tempo regions with few samples or large variability in performance. In Figure 6.19d this can be seen for less than 75 BPM (few tracks),

[20] GAMs were generated using pyGAM by Servén and Brummitt [172].

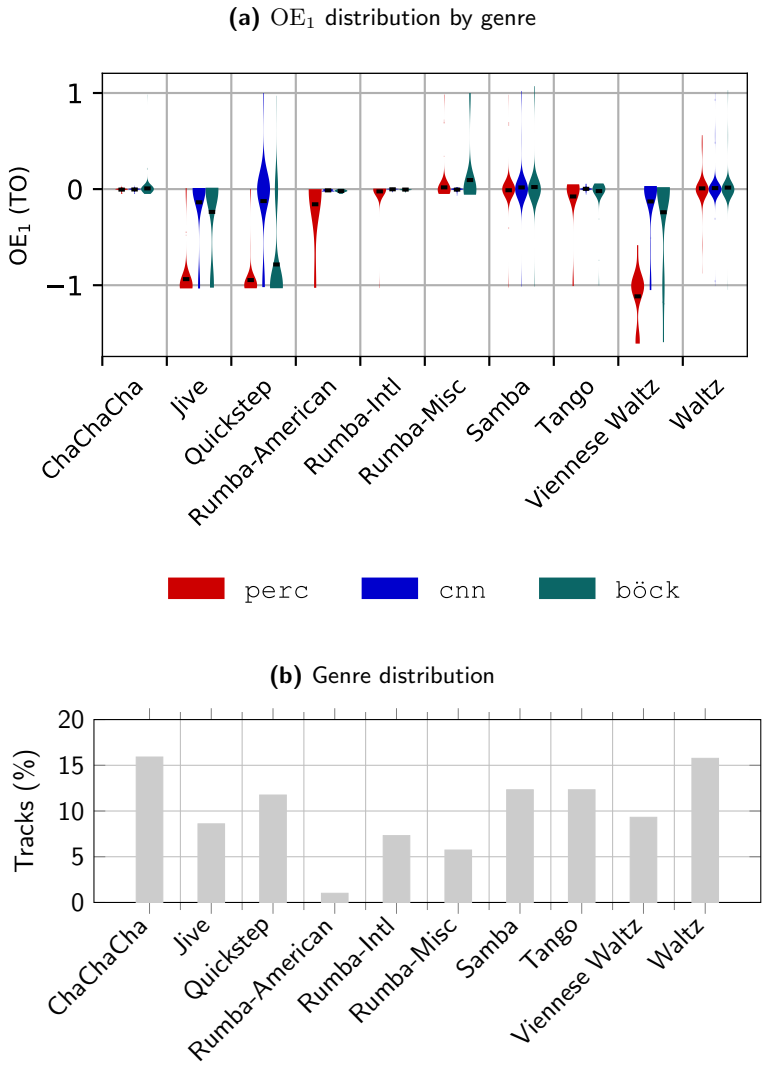


Figure 6.20.: **(a)** Per genre OE_1 distributions based on kernel density estimation (KDE) for tracks from *Ballroom* using the ground truth from Percival and Tzanetakis [128]. Mean OE_1 values are marked in black. **(b)** Genre distribution in *Ballroom*.

around 150 BPM (performance starts to shift), and for more than 210 BPM (few tracks, low performance).

Because JAMS supports additional annotations like genre, tags, and beat positions, these can be incorporated into the evaluation. As example, Figure 6.20a shows OE_1 distributions by genre. Mostly due to −1 TO octave errors, `perc` does poorly on Jive, Quickstep, and Viennese Waltz—the three genres with the highest average tempo. `böck` faces the same issue with Quickstep. This is noteworthy, because Jive, Quickstep, and Viennese Waltz combined make up almost 30% of the *Ballroom* dataset, as shown in Figure 6.20b.

Note that evaluation by ballroom genre is just an example. The code picks up on any `tag_open` annotation.

6.7. Conclusions

Our answer to the opening question, whether the task of global tempo estimation is solved yet, is *no*. Not because estimation systems are not good enough—we do not really know whether that's the case or not, but because it is impossible to solve a task for which neither use cases with success criteria have been well motivated and properly defined, nor the suitability of metrics or datasets has been shown. Instead, currently used metric tolerances are somewhat arbitrary, some popular datasets are too small, contain unsuitable tracks or are single-genre, and there is no definite agreement on how to define a global tempo. To actually solve the task, we suggest to first tackle these evaluation issues, i.e., clearly define use cases and success criteria (e.g., ballroom dance tournament, 99.9% accuracy, integer precision), choose the appropriate metrics for the use case, and curate suitable datasets. Only then an estimation system may succeed at solving whatever the resulting task may be.

Acknowledgement

Julián Urbanos's contribution to this chapter was supported by European Union's H2020 programme (770376-2–TROMPA).

7. Tailored CNN-Architectures for Tempo and Key Estimation

In this chapter we discuss how directional filters can be used to steer what a CNN might learn. To this end, we implement systems for two different tasks using similar architectures. This chapter is based on my work in [163].

In recent years, CNNs have been employed for various MIR tasks, such as key detection [90, 91], tempo estimation (Chapter 4), beat and rhythm analysis [73, 38, 55], genre recognition [31, 151], and general-purpose tagging [34, 22]. Typically, a spectrogram is fed to the CNN and then classified in a way appropriate for the task. In contrast to recent computer vision approaches like Oxford's Visual Geometry Group's (VGG) deep image recognition network [175], some of the employed CNN architectures for MIR tasks use rectangular instead of square filters. The underlying idea is that, while for images the axes *width* and *height* have the same meaning, the spectrogram axes *frequency* and *time* have fundamentally different meaning. For MIR tasks mainly concerned with temporal aspects of music (e.g., tempo estimation, rhythmic patterns), rectangular filters aligned with the time axis appear suitable (Chapter 4). Correspondingly, tasks primarily concerned with frequency content (e.g., chord or key detection), may be approached with rectangular filters aligned with the frequency axis [108]. In fact, tempo and key estimation can be seen as tasks from two different ends of a spectrum of common MIR tasks, which are addressed by systems relying more or less on temporal or spectral signal properties (Figure 7.1). Systems for other tasks like general-purpose tagging or genre recognition are found more towards the center of this spectrum as they usually require both spectral and temporal information.

In [131] Pons et al. explored the role of CNN filter shapes for MIR tasks. In particular, they examined using rectangular filters in shallow CNNs for automatic genre recognition of ballroom tracks. Defining *temporal* filter shapes as $1 \times n$ and *spectral* filter shapes as $m \times 1$, they showed that using temporal filters alone, 81.8% accuracy can be reached, which is in line with a Nearest Neighbor classifier (k-NN) using tempo as feature scoring 82.3% [63]. Using just spectral filters, the test network reached 59.6% accuracy, and a fusion architecture with both temporal and spectral filters performed as well as an architecture using square filters, scoring 87%. The

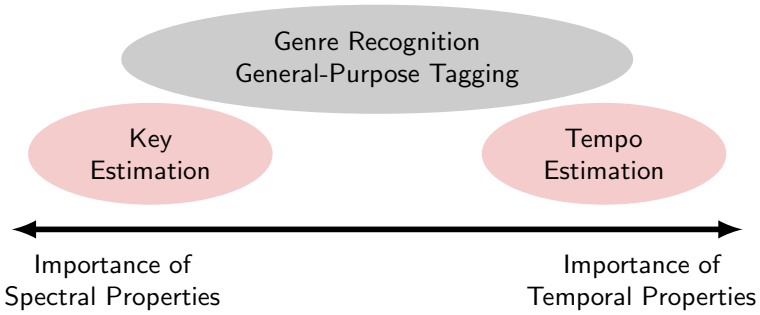


Figure 7.1.: Several MIR tasks and their reliance on spectral or temporal signal properties.

experiments confirmed that such *directional* filters can be used to match either temporal or spectral signal properties and that both may be useful for genre recognition.

Even though directional filters did not outperform square filters, there are good arguments for using them: First, CNNs using specialized, directional filters may use fewer parameters or match musical concepts using fewer layers [130]. Second, by limiting what a filter can match, one can influence what a CNN might learn, thus better avoid "horses" [181] and improve explainability. The latter is especially interesting for genre recognition systems, given their somewhat troubled history with respect to explicit matching of musical concepts [179, 130]. To further explore how and why directional or square filters contribute to results achieved by CNN-based classification systems for MIR tasks, we believe it is beneficial to build on Pons et al.'s work and experiment with tasks that explicitly aim to recognize either high-level temporal or spectral properties, avoiding hard to define concepts like genre. Such tasks are global key and tempo estimation.

The remainder of this chapter is structured as follows: In Section 7.1 we describe our experiments by defining both tasks, the network variants used, the training procedure, and evaluation. The results are then presented in Section 7.2 and discussed in Section 7.3. Finally, in Section 7.4 we present our conclusions.

Reproducibility

Code to recreate models and reproduce the reported results can be found at `https://github.com/hendriks73/directional_cnns`. For ready-to-run tempo and key models presented in this chapter, please see Appendix A.3 and B.1.

7.1. Experiments

For the purpose of comparing the effects of using different filter shapes we train and evaluate different CNN architectures for the key and tempo estimation tasks using several datasets. In

this section, we first describe the two tasks, then discuss the network architectures and datasets used, and finally outline the evaluation procedure.

7.1.1. Key Estimation

Key estimation attempts to predict the correct key for a given piece of music. Oftentimes, the problem is restricted to major and minor modes, ignoring other possible modes like Dorian or Lydian, and to pieces without modulation. Framed this way, key estimation is a classification problem with $N_{\mathrm{K}} = 24$ different classes (12 tonics, major/minor). The current state-of-the-art system is CNN-based using a VGG-style architecture with square filters [91] and a fully convolutional classification stage, as opposed to a fully connected one. This allows training on short and prediction on variable length spectrograms.

In our experiments we follow a similar approach. As input to the network (Section 7.1.3) we use constant-Q magnitude spectrograms with the dimensions $F_{\mathrm{K}} \times T_{\mathrm{K}} = 168 \times 60$; F_{K} being the number of frequency bins and T_{K} the number of time frames. F_{K} covers the frequency range of 7 octaves with a frequency resolution of two bins per semitone. Time resolution is 0.19 s per time frame, i.e., 60 frames correspond to 11.1 s. Since all training samples are longer than 11.1 s, we choose a random offset for each sample during each training epoch and crop the spectrogram to 60 frames. To account for class imbalances within the two modes, each spectrogram is randomly shifted along the frequency axis by $\{-4, -3, \ldots, +6, +7\}$ semitones and the ground truth labels are adjusted accordingly. We define *no shift* to correspond to a spectrogram covering the 7 octaves starting at pitch E1. In practice, we simply crop an 8 octaves spanning spectrogram that starts at C1 to 7 octaves. After cropping the spectrogram is normalized so that it has zero mean and unit variance.

7.1.2. Tempo Estimation

Even though tempo estimation naturally appears to be a regression task, we have shown in Chapter 4 that it can also be treated as a classification task by mapping BPM values to distinct tempo classes. Concretely, our system maps the integer tempo values $\{30, \ldots, 285\}$ to $N_{\mathrm{T}} = 256$ classes. As input to a CNN with temporal filters and elements from [184] and [130] we use mel-magnitude-spectrograms. Even though we work with other network architectures than we did in Chapter 4 (Section 7.1.3), we use the same general setup. We also treat tempo estimation as classification into 256 classes and use mel-magnitude-spectrograms with the dimensions $F_{\mathrm{T}} \times T_{\mathrm{T}} = 40 \times 256$ as input; F_{T} being the number of frequency bins and T_{T} the number of time frames. F_{T} covers the frequency range $20 - 5,000$ Hz. The time resolution is 46 ms per time frame, i.e., 256 frames correspond to 11.9 s.

(a) `Shallow`

Module
`ShallowMod`
`ClassMod`

(b) `Deep`

Module	Size
`DeepMod`	$\ell = 0$
`DeepMod`	$\ell = 1$
`DeepMod`	$\ell = 2$
`DeepMod`	$\ell = 2$
`DeepMod`	$\ell = 3$
`DeepMod`	$\ell = 3$
`ClassMod`	

Table 7.1.: Network architectures used. **(a)** `Shallow` architecture consisting of a variant of the `ShallowMod` module and a `ClassMod` module. **(b)** `Deep` architecture consisting of multiple, `DeepMod` modules parameterized with ℓ to influence the filter count and a `ClassMod` module.

Just like the training excerpts for key estimation, the mel-spectrograms are cropped to the right size using a different randomly chosen offset during each epoch. To augment the training dataset, spectrograms are scaled along the time axis before cropping using the factors $\{0.8, 0.84, \ldots, 1.16, 1.2\}$. Ground truth labels are adjusted accordingly [161]. After cropping and scaling spectrograms are normalized ensuring zero mean and unit variance per sample.

7.1.3. Network Architectures

To gain insights into how filter shapes affect the performance of CNN-based key and tempo estimation systems we run experiments with two very different architectures: a relatively shallow but specialized one, and a commonly used much deeper one from the field of computer vision. Both architectures are used for both tasks.

7.1.3.1. Shallow Architectures

Our `Shallow` architecture, depicted in Figure 7.2 and textually outlined in Table 7.1a, consists of two parts: the feature extraction module `ShallowMod` and the classification module `ClassMod`. `ShallowMod` (Table 7.2a), is inspired by a classic signal processing approach that first attempts to find local spectrogram peaks along one axis, averages these peaks over the other axis, and then attempts to find a global pattern, i.e., a periodicity for tempo estimation [85] and a pitch profile for key detection [94]. In terms of CNNs this means that our first convolutional layer consists of short directional filters (local peaks), followed by a one-dimensional average pooling layer that is orthogonal to the short filters, followed by a layer with long directional filters (global pattern) that stretch in the same direction as the short filters. We use ReLU as activation function for the convolutional layers and to avoid overfitting we add a dropout layer [177] after each ReLU. The parameters k and p_{D} let us scale the number of convolutional filters and set

(a) `ShallowMod`

Layer	Temp	Spec	Square
Input			
Conv, ReLU	k, 1×3	k, 3×1	n.a.
Dropout	p_D	p_D	n.a.
AvgPool	$F_\mathrm{T} \times 1$	$1 \times T_\mathrm{K}$	n.a.
Conv, ReLU	$64k$, $1 \times T_\mathrm{T}$	$64k$, $F_\mathrm{K} \times 1$	n.a.
Dropout	p_D	p_D	n.a.

(b) `DeepMod`

Layer	Temp	Spec	Square
Input			
Conv, ReLU	$2^\ell k$, 1×5	$2^\ell k$, 5×1	$2^\ell k$, 5×5
BatchNorm			
Conv, ReLU	$2^\ell k$, 1×3	$2^\ell k$, 3×1	$2^\ell k$, 3×3
BatchNorm			
MaxPool	2×2	2×2	2×2
Dropout	p_D	p_D	p_D

(c) `ClassMod`

Layer	Temp	Spec	Square
Input			
Conv, ReLU	N_T, 1×1	N_K, 1×1	n.a.
GlobalAvgPool			
Softmax			

Table 7.2.: Layer definitions for the three modules `ShallowMod`, `ClassMod`, and `DeepMod`, describing number of filters (e.g., k or $64k$) and their respective shapes (e.g., 1×3 or 5×5).

dropout probabilities. `ShallowMod` is followed by a fully convolutional classification module named `ClassMod` (Table 7.2c), which consists of a 1×1 bottleneck layer (pointwise convolution) with as many filters as desired classes (N_K or N_T), a global average pooling layer, and the softmax activation function. Note, that because all directional filters are identically aligned, the model has an asymmetric, *directional capacity*, i.e., it has a much larger ability to describe complex relationships in one direction than in the other.

We use the same general architecture for both key and tempo estimation. The only differences are the filter and pooling directions and dimensions. For tempo estimation we use temporal filters with pooling along the frequency axis, and for key estimation spectral filters with pooling along the time axis. Both architectures are named after their filter directions, `ShallowTemp` and `ShallowSpec`, respectively. We also adjust the pooling and the long filters shape to the size of the input spectrogram, which is different for the two tasks.

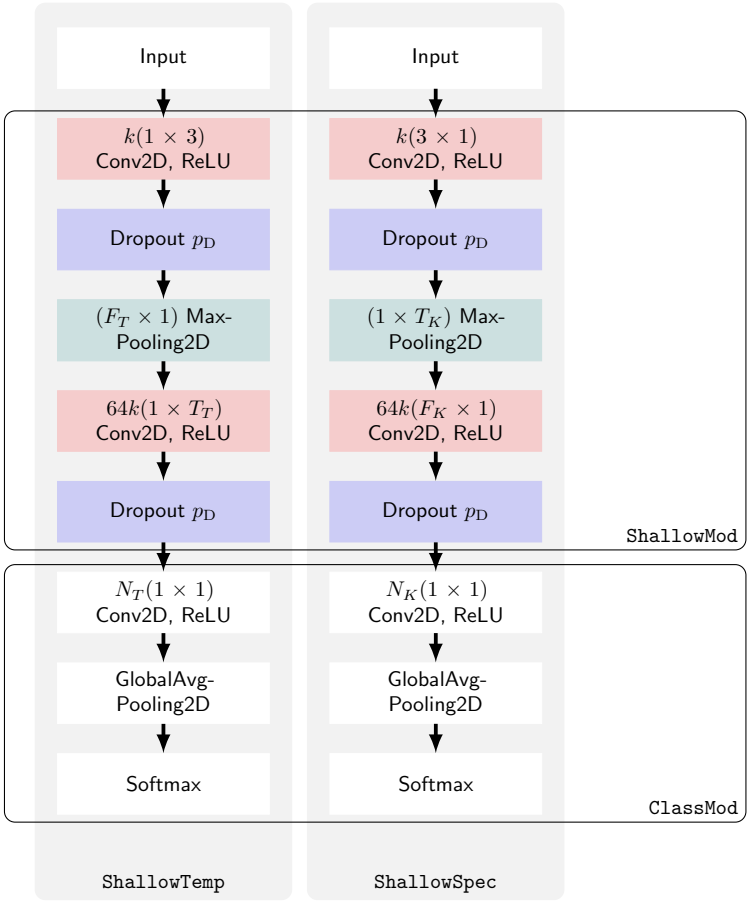


Figure 7.2.: Conceptual overview of `Shallow` architectures.

7.1.3.2. Deep Architectures

The second architecture, `Deep` (Figure 7.3, Table 7.1b), is a common VGG-style architecture consisting of six parameterized feature extraction modules `DeepMod` (Table 7.2b, Figure 7.3 right) followed by the same classification module that we have already used in `Shallow`. Each of the feature extraction modules contains a convolutional layer with 5×5 filters followed by a convolutional layer with 3×3 filters. The convolutional layers consist of $2^\ell k$ filters each, with network parameter k and module parameter ℓ. While ℓ influences the number of filters in an instance of `DeepMod`, k lets us scale the total number of parameters in the network. As shown in Table 7.1b, deeper instances have more filters. The convolutional layers are followed by a 2×2 max pooling layer. Should pooling not be possible along an axis, because the output from the previous layer is only 1 wide or high, pooling is skipped along that axis. This happens for example, when a tempo spectrogram with its 40 bands passes through more than 5 max pools. Each pooling layer is followed by a dropout layer with probability p_D. To counter covariate shift, we add batch normalization [78] layers after each convolutional layer.

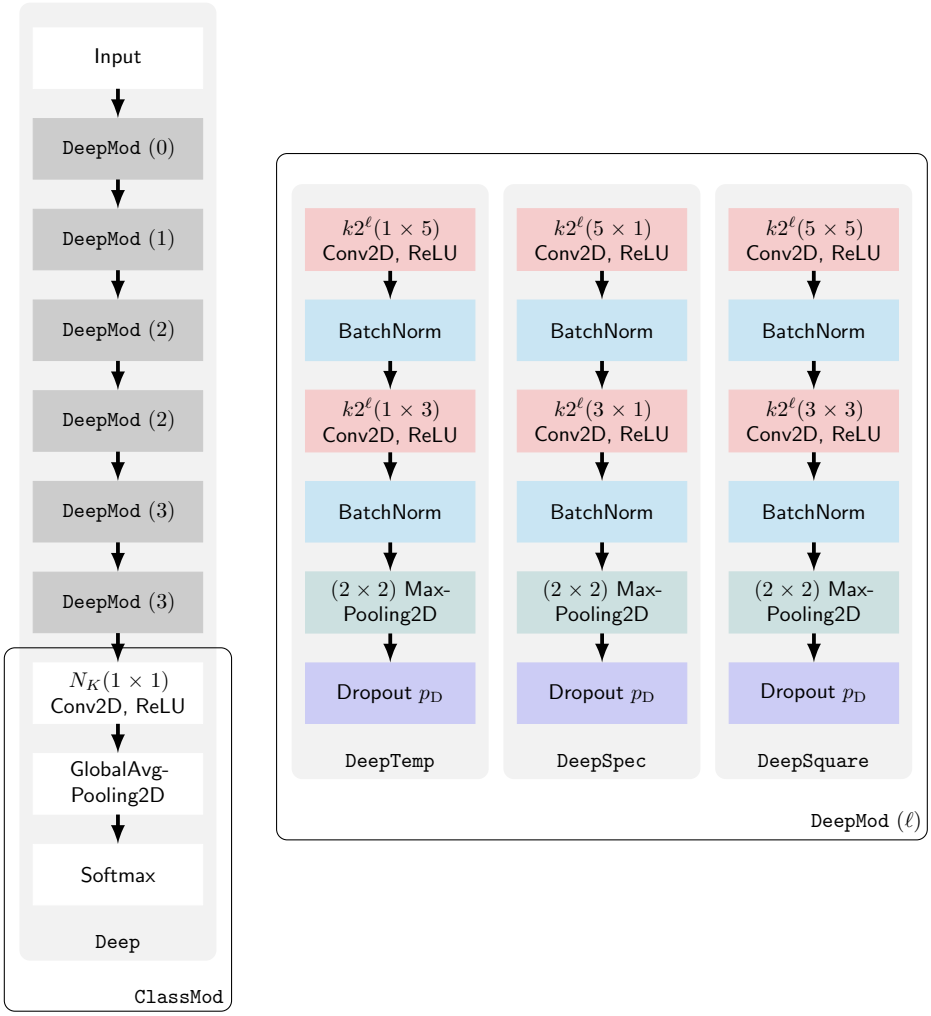


Figure 7.3.: (left) Conceptual overview of `Deep` architectures with **(right)** different `DeepMod` modules.

The general structure of the `Deep` architecture is customized neither for the key nor for the tempo task. However, in order to investigate how different filter shapes affect the network's performance, we modify the described architecture by replacing the square convolutional kernels with directional ones, i.e., 3×3 with 1×3 or 3×1, and 5×5 with 1×5 or 5×1. Analogous to the naming scheme used for shallow networks, we denote the directional variants `DeepTemp` and `DeepSpec`. The original variant is named `DeepSquare`.

7.1.4. Datasets

We use the following publicly available datasets from different genres for both training and evaluation (listed in alphabetical order). The used splits are randomly chosen and listed in Table 7.3.

Split	Key Datasets
Training	80% of *LMD Key* ∪ 80% of *MTG Key*
Validation	10% of *LMD Key* ∪ 20% of *MTG Key*
Testing	*GiantSteps Key*, *GTzan Key*, 10% of *LMD Key*

Split	Tempo Datasets
Training	80% of *EBall* ∪ 80% of *MTG Tempo* ∪ 80% of *LMD Tempo*
Validation	20% of *EBall* ∪ 20% of *MTG Tempo* ∪ 10% of *LMD Tempo*
Testing	*GiantSteps Tempo*, *GTzan Tempo*, 10% of *LMD Tempo*, *Ballroom*

Table 7.3.: Dataset splits used in key **(top)** and tempo **(bottom)** estimation experiments.

Ballroom (698 samples) 30 s excerpts with tempo annotations [64].

EBall (3,826 samples) 30 s excerpts with tempo annotations, excluding tracks also occurring in the regular *Ballroom* dataset [105, 64, 161].

GiantSteps Key (604 samples) 2 min excerpts of electronic dance music (EDM) [88].

GiantSteps Tempo (661 samples) 2 min excerpts of EDM [88]. Revised tempo annotations from [162].

GTzan Key (836 samples) 30 s excerpts from 10 different genres [185]. Key annotations from [92].[1] Most tracks with missing key annotations belong to the genres Classical, Jazz, and Hip-hop.

GTzan Tempo (999 samples) 30 s excerpts from 10 different genres [185]. Tempo annotations from [128].

LMD Key (6,981 samples) 30 s excerpts, predominantly Rock and Pop [136, 156]. Due to a MIDI peculiarity, this dataset does not contain any tracks in C major. Some form of data augmentation as described above is therefore necessary.

LMD Tempo (3,611 samples) 30 s excerpts, predominantly Rock and Pop [136, 161].

MTG Tempo / MTG Key (1,158 samples) 2 min EDM excerpts annotated with both key and tempo [45, 161]. We used only tracks that are still publicly available, have an unambiguous key, and a high key annotation confidence.[2]

[1] `https://github.com/alexanderlerch/gtzan_key`
[2] `https://github.com/GiantSteps/giantsteps-mtg-key-dataset`

7.1.5. Evaluation

Since the proposed network architectures are fully convolutional, we can choose at prediction time to pass a track either in one long spectrogram or as multiple shorter windows through the network. In the latter case, predictions for all windows would have to be aggregated. Informal experiments did not show a remarkable difference. For this work we choose to predict values for whole spectrograms.

When evaluating key estimation systems either a simple accuracy or a score is used that assigns additional value to musically justifiable mistakes, like being off by a perfect fifth.[3] For this work, we choose to only report the percentage of correctly classified keys. Tempo estimation systems are typically evaluated using the metrics ACC_1 and ACC_2 [64]. We choose to report only ACC_1.

For training we use Adam [84] as optimizer with a learning rate of 0.001, a batch size of 32, and early stopping once the validation loss has not decreased any more during the last 100 epochs. In this work, one epoch is defined as having shown all training samples to the network once, regardless of augmentation. We choose k so that we can compare architectures with similar parameter counts. `Shallow` is trained with $k \in \{1, 2, 4, 6, 8, 12\}$ and `Deep` with $k \in \{2, 4, 8, 16, 24\}$. Additionally, `DeepSquare` is trained with $k = 1$. For both architectures we apply various dropout probabilities $p_\mathrm{D} \in \{0.1, 0.3, 0.5\}$. Each variant is trained 5 times and mean validation accuracy along with its standard deviation is recorded for each variant. In total we train 420 models with 84 different configurations.

For testing, we pick the dropout variant of each network class that performed best on the validation set and evaluate it against the test datasets. Again, we report the mean accuracies for 5 runs along with their standard deviations.

7.2. Results

Figure 7.4 shows mean validation accuracies of 5 runs for each configuration, using their best performing dropout probability p_D. The dashed black line is the accuracy a random classifier achieves, and the dotted black line shows the accuracy of the algorithm that always outputs the class that most often occurs in the validation set. With accuracy values slightly above random, `ShallowSpec` and `ShallowTemp` perform worst of all architectures, when used for the task they were *not* meant for. But when used for the task they were designed for, both perform well. A higher number of parameters leads to slightly better results. When training `ShallowTemp` with $k = 1$ for the tempo task, the network performed very poorly in one of the five runs, which is the cause for the very large standard deviation of 32.2 shown in Figure 7.4. The mean accuracy

[3] `https://www.music-ir.org/mirex/wiki/2018:Audio_Key_Detection`

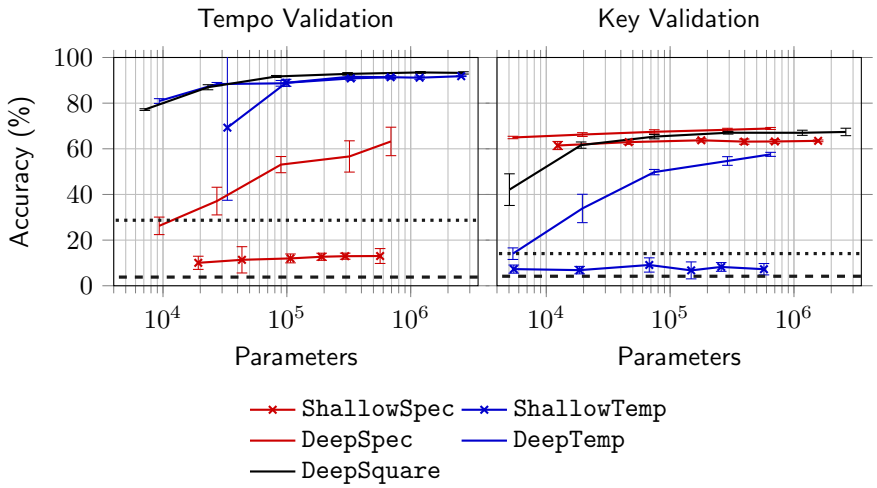


Figure 7.4.: Mean validation accuracies for the various network configurations depending on their number of network parameters. Only the best dropout configuration is shown. Whiskers represent the standard deviation based on five runs.

for the 4 successful runs was 85.2%. When comparing with the `Deep` architectures, we see that `DeepTemp` performs just as well as `ShallowTemp` with $k > 1$ on the tempo task, and `DeepSpec` clearly outperforms `ShallowSpec` on the key task. Surprisingly, the `DeepSpec` architecture reaches fairly high accuracy values (up to 63%) on the tempo task when increasing the model capacity via k, even though it only has convolutional filters aligned with the frequency axis. We can make a similar observation for the `DeepTemp` architecture. It too reaches relatively high accuracy values on the key task (up to 57%) when increasing k. The unspecialized `DeepSquare` is by a small margin the best performing architecture for the tempo task, and comes in as a close second for key detection with $k > 1$. But for $k = 1$, `DeepSquare` performs considerably worse than `DeepSpec` with $k = 2$ (42% compared to 64%), even though both have similar parameter counts of ca. 5 000.

We selected the dropout variant for each architecture and parameter setting with the best validation accuracy and ran predictions on the test sets. Detailed results are shown in Figure 7.5. The general picture is very similar to validation: `Deep` architectures tend to perform slightly better than `Shallow` architectures on the tasks they are meant for and `Shallow` architectures perform poorly on the task they were not meant for. In fact, `ShallowTemp` performs no better on *GTzan Key* and *GiantSteps Key* than the random baseline. For both key and tempo `DeepSquare` performs as well or better than any other architecture, except when drastically reducing the model capacity for the key task ($k = 1$). Then accuracy decreases well below `DeepSpec`'s performance with similar parameter count: 33% compared to 50% for *GTzan Key*, and 21% compared to 51% for *GiantSteps Key*.

To provide an absolute comparison, we chose the best performing representative from each architecture (based on validation accuracy, regardless of dropout configuration or capacity), and

(a) Tempo

Architecture	*GiantSteps*	*GTzan*	*LMD*	*Ballroom*
`ShallowTemp`	86.5 1.5	60.3 2.7	94.0 1.0	87.9 2.3
`DeepTemp`	**88.7** 0.6	63.1 0.6	94.5 0.7	88.2 2.4
`ShallowSpec`	4.5 1.9	11.5 1.3	9.4 2.1	16.7 5.7
`DeepSpec`	49.6 2.5	40.2 1.4	73.0 2.4	59.6 9.1
`DeepSquare`	88.1 1.3	**64.7** 2.1	**96.2** 0.4	**92.4** 1.7
Literature	82.5 [161]	78.3 [128]	—	92.0 [161]

(b) Key

Architecture	*GiantSteps*	*GTzan*	*LMD*
`ShallowTemp`	1.7 0.4	4.9 0.7	11.0 3.7
`DeepTemp`	46.8 4.3	38.4 2.4	60.7 0.4
`ShallowSpec`	50.8 3.8	43.8 1.4	67.1 0.9
`DeepSpec`	55.4 2.7	44.8 2.0	**71.3** 0.2
`DeepSquare`	**58.5** 3.9	**49.9** 2.0	68.9 2.5
Literature	67.9 [91]	≈45 [92]	—

Table 7.4.: Mean estimation accuracies of 5 runs with standard deviation (small font). Best results per test are set in **bold**. Model variants chosen based on validation performance (ignoring parameter count).

calculated accuracies for each test set (Table 7.4, incl. reference values from the literature). In 5 out of 7 test cases `DeepSquare` reaches the highest accuracy score among our architectures. The other two are reached by `DeepTemp` for *GiantSteps Tempo* and by `DeepSpec` for *LMD Key*. For both tasks we observe that the margin by which the best performing network is better than the second best for a given dataset differs considerably. `DeepSquare` reaches an accuracy of 92.4% for the *Ballroom* tempo dataset, which is 4.2 pp (percentage points) better than the second best network (`DeepTemp`, 88.2%). The differences between best and second best accuracy are considerably lower for the other datasets: 1.7 pp (*LMD Tempo*), 1.6 pp (*GTzan Tempo*), and 0.6 pp (*GiantSteps Tempo*). For the key task, `DeepSquare` reaches an accuracy of 49.9% on *GTzan Key*, which is 5.1 pp better than the second best network (`DeepSpec`, 44.8%), while the differences between best and second best for the other datasets are 3.1 pp (*GiantSteps Key*), and 2.4 pp (*LMD Key*).

7.3. Discussion

The results show that simple shallow networks with axis-aligned, directional filters can perform well on both the key and tempo task. Conceptually, both tasks are similar enough that virtually the same architecture can be used for either one, as long as the input representation and the filter direction are appropriate. Using the wrong filter direction, e.g., `ShallowSpec` for the tempo task, leads to very poor results close to the random baseline. Together, this strongly supports

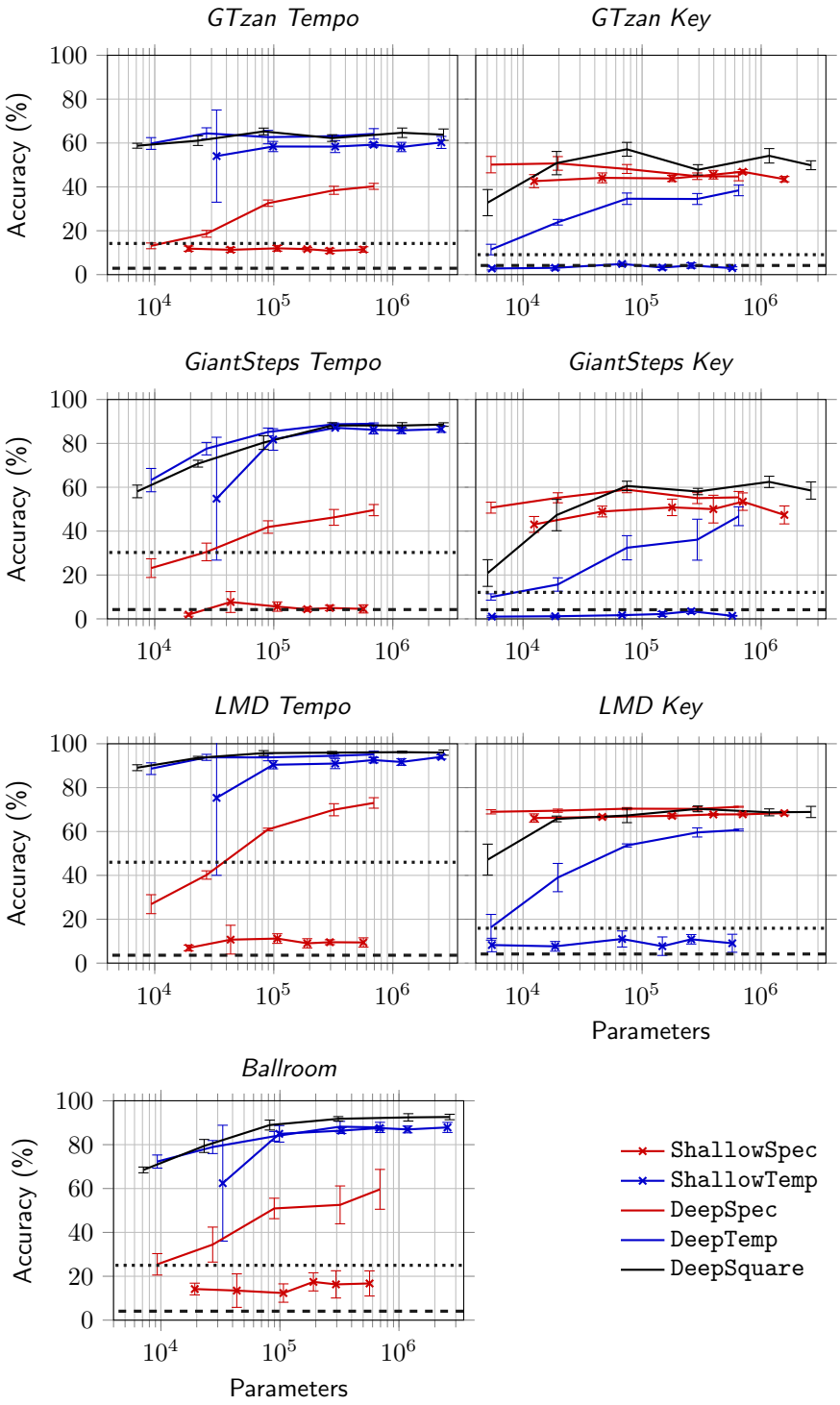


Figure 7.5.: Mean test accuracies for various network configurations and datasets depending on their number of network parameters. Whiskers represent one standard deviation based on 5 runs. Dropout was chosen based on performance during validation.

the hypothesis that the `Shallow` architecture indeed learns what we want it to learn, i.e., pitch patterns for key detection or rhythmic patterns for tempo detection, but not both.

This stands in contrast to the standard VGG-style network (`DeepSquare`). Because of its square filters, we cannot be certain what it learns, just by analyzing its static architecture. It is designed to pick up on anything that could provide a hint towards correct classification, be it rhythm and pitch patterns, or timbral properties like instrumentation. And indeed our experiment shows that without being specialized for either key or tempo estimation in any way, `DeepSquare` works very well for both tasks. In Section 7.2 we noted that `DeepSquare` achieved the greatest tempo accuracy for *Ballroom* and the greatest key accuracy for *GTzan Key* by a considerable margin of 4.2 pp and 5.1 pp, respectively. This margin may be a result of the fact that key and tempo are related to genre [75, 168, 199, 44]. Specifically, in *Ballroom* there is a strong correlation between genre and tempo, and *GTzan Key* is the key test set with the greatest genre diversity and therefore stands to benefit the most from an architecture that can distinguish genres based on *both* temporal and timbral properties. Consequently, square filters *should* improve accuracy results for these datasets. But this does not conclusively show that only the network's ability to measure specifically key or tempo is reflected by these results, as the system is by design vulnerable to confounds [181]. By using directional filters in `DeepSpec` and `DeepTemp` we intentionally limit the standard VGG-style architecture in a way that seeks to lessen this vulnerability as well as reduce the number of required parameters.

The results for `DeepSpec` and `DeepTemp` show that a VGG-style network with directional filters can perform very well on either task. For networks with a large number of parameters test results are similar to `DeepSquare`, with a tendency towards a slightly worse performance. Interestingly, the situation is different for low-capacity networks with $k = 2$ for `DeepSpec`, and $k = 1$ for `DeepSquare`. Here, `DeepSpec` clearly outperforms `DeepSquare`, even though the parameter count is similar. Perhaps with ca. 5 000 parameters `DeepSquare` simply does not have enough capacity aligned in the right direction to still perform well on the task.

The fact that `DeepSpec` and `DeepTemp` with $k = 2$ perform very poorly on the tasks they are not meant for, supports the hypothesis that they only learn the intended features for the tasks they are meant for. For $k > 2$ we cannot be quite as certain, as both architectures reach higher accuracy scores on the tasks they were not meant for for greater values of k. We believe this effect may be a result of the 2×2 max pooling in the `DeepMod` modules.

7.4. Conclusions

We have shown that shallow, signal processing-inspired CNN architectures using directional filters can be used successfully for both tempo and key detection. By using shallow networks designed

for key detection on the tempo task and vice versa, we were able to experimentally support the hypothesis that these networks are incapable of matching information from the domain they were not meant for, which would make them less susceptible to confounds.

We further demonstrated that a standard VGG-style architecture can be used for tempo estimation, as it has been shown before for key detection [91]. By replacing square filters with directional filters, we derived a musically motivated, directional VGG-variant that performs similarly well as the original one, but is less vulnerable to confounds, especially when used for key detection with low capacity models. In such scenarios we were also able to observe efficiency gains, i.e., better performance than the standard VGG-style network with similar parameter counts.

8. Chopin Mazurkas: A Cross-Version Study on Local Tempo Estimation

In this chapter, we discuss local tempo and its stability. It is based on my work in [?], which is currently under review. Jonathan Driedger and Frank Zalkow contributed in preliminary discussions about tempo modeling, Frank Zalkow also helped with editing.

While *global* tempo is well defined for music with little or no tempo variability [64], this is less so the case for *local* tempo, especially for expressive classical music. Composer markings like *rubato* (expressive, local tempo change) or *ritardando* (slow down) indicate continuous or even abrupt tempo changes, leading to one or more segments with stable tempi and segments of tempo instability in between. Figure 8.1, for example, shows tempo markings for Frédéric Chopin's Mazurka Op. 68, No. 3 (details are discussed in Section 8.1). Naïvely, one may model local tempo for such a piece as one of two extremes: at the *micro level*, as an instantaneous value, e.g., as the inter-beat interval (IBI) between two consecutive beats, or at the *macro level*, by averaging the number of beats over a longer period of time. For expressive music, both approaches have disadvantages. IBIs exhibit a large variance, and averaging beat counts may underestimate the tempo, because expression leads more often to longer than shorter IBIs [138]. Repp therefore attempts to find a definition for the *basic tempo* [139], i.e., the implied tempo the instantaneous tempo varies around. In [138] he suggests to derive the basic tempo from the first quartile of eighth-note inter-onset intervals (IOIs). Similarly, Dixon [33] proposes IOI clustering, using centroids as tempo hypotheses. Grosche et al. [67] propose yet another approach by defining local tempo as the mean of three consecutive IBIs, which is identical to using inter-measure intervals (IMIs) for pieces in $^3/_4$ time. In summary, local tempo is usually modeled by aggregating local pulse information, but there appears to be no clear consensus on how. Even though local tempo estimates are popular intermediate features for beat trackers (e.g., [40, 57]), few works explicitly estimate and evaluate local tempo estimates. Peeters [125] simply measures whether 75% of the estimated local tempi match the annotated global tempo. In subsequent work [126], he compares the median of local tempi with a global ground truth. A similar approach is taken in [121]—after beat tracking, the median IBI is used as global tempo and then evaluated. Similar

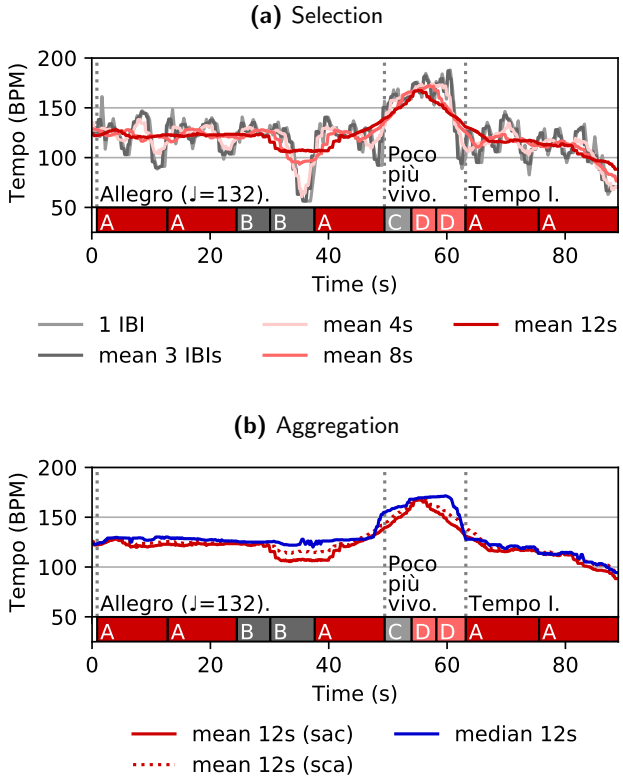


Figure 8.1.: Local reference tempo depending on **(a)** selection and **(b)** aggregation functions for Op. 68, No. 3 (Cohen, 1997) with part boundaries and score tempo markings.

to global tempo evaluation, Grosche et al. [67] compute the accuracy of their IMIs allowing a 4% tolerance and certain integer factors. Our own method, proposed in Chapter 4, only provides visualizations for local tempo estimates. Apparently, there is no commonly accepted evaluation procedure. Even less researched than local tempo is tempo *stability*. Grosche et al. [69] mention that beat trackers tend to have problems with the first and last few beats of Mazurkas due to boundary problems, and observe increased error-levels caused by sudden tempo changes, but as far as we know no measure for local tempo stability has been proposed.

In this chapter we investigate how to model local tempo (Section 8.1) and tempo stability (Section 8.2) for expressive music using Mazurkas by Chopin. As our main contribution, we estimate local tempi using state-of-the-art neural network-based approaches, adapt these approaches to our use case, and explore their behavior and potential (Section 8.3). In our evaluation, we focus on identifying error classes and sources, and in particular the effect of stability. In Section 8.4 we discuss our findings and draw conclusions.

Reproducibility

Please see Appendix A.3 for trained and ready-to-run versions of models presented in this chapter.

Work	Measures	Beats	Recordings
Op. 17, No. 4	132	396	62
Op. 24, No. 2	120	360	64
Op. 30, No. 2	65	193	34
Op. 63, No. 3	77	229	88
Op. 68, No. 3	61	181	50

Table 8.1.: *Mazurka-5* dataset overview: Number of measures, beats, recordings for five Chopin Mazurkas.

8.1. Local Tempo

When modeling local tempo, we are interested in a musically meaningful, single-value description of a segment of limited length. We can describe this length musically, e.g., as three consecutive IBIs [67], or physically, e.g., as 6 s or 8 s segments [125, 126]. In either case, we first *select* beat events, because they fall into a time span, and then *aggregate* them. For example, we may use the mean or the median of all IBIs falling into a 4 s interval. Note that we are not attempting to find the *most* suitable selection and aggregation functions (see [139]), but merely discuss options and aim to establish a framework that can be used for such an endeavor. To illustrate different choices, we use Chopin's Op. 68, No. 3 (piano: Cohen, 1997) as example. It is one of over 2,700 recordings of 49 Mazurkas by Chopin collected by the Mazurka Project.[1] Of all collected recordings, 298 recordings of five Mazurkas have been manually beat-annotated [147]. We refer to this subset as the *Mazurka-5* dataset. It contains between 34 and 88 different versions of each of the five Mazurkas (Table 8.1).

Our example, Op. 68, No. 3, consists of four different musical parts A to D (Figure 8.1). Its score explicitly specifies two tempo changes: at the start of C from *Allegro, ma non troppo (♩=132)* (fast, but not too fast), to *Poco più vivo* (a little more lively), and back to *Tempo I* after the second D-part. Figure 8.1a depicts the effects of different selection functions using the mean for aggregation. We see that defining local tempo as individual IBIs leads to very high variance. Using three consecutive IBIs smoothes the tempo curve slightly. The shown tempo curves based on 4 s, 8 s, and 12 s segments progressively lead to less variance. While the 4 s tempo curve still follows the phrasing closely (distinct minima at the end of each musical part), this is less so the case for the curves based on longer segments. This is especially obvious at the end of the 2$^{\text{nd}}$ B part at 38 s.

Figure 8.1b shows the differences between using mean and median as aggregation function. The tempo curves for mean show local over-smoothing in transitional sections, leading to a triangular shape in the more lively CDD-section from $50 - 60$ s. Because of the edge-preserving property of median-filtering, the median curve captures sudden tempo changes better. The CDD-section much more resembles a rectangle, i.e., a high tempo plateau. At the same time, the local

[1] `http://www.mazurka.org.uk/`

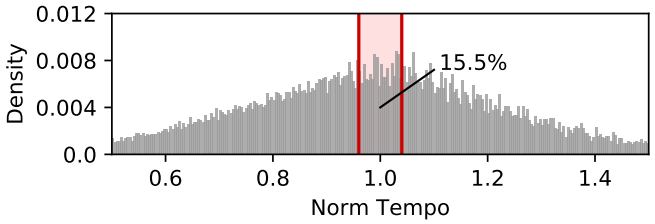


Figure 8.2.: Per recording normalized tempo distribution with percentage of values between 0.96 and 1.04 (light-red area).

minimum at the end of the 2nd B disappears. Thus, the median curve corresponds nicely to the composer's markings.

So far we first selected IBIs, aggregated them, and then converted the result to BPM (selection → aggregation → conversion: sac). As an alternative, we could have first converted IBIs to BPM and then aggregated them (selection → conversion → aggregation: sca). When using mean, the result is not the same. For sections with changing tempo (Figure 8.1b, $30 - 70$ s), local tempo values are lower when we first average and then convert (sac, solid red line) as opposed to first convert and then average (sca, dotted red line). Note that the median is unaffected by this issue.

8.2. Tempo Stability

As a first approach to describe tempo stability quantitatively on the intra-track level, we convert all *Mazurka-5* IBIs to tempo values and normalize them by dividing with their respective track's average. Figure 8.2 depicts the resulting normalized histogram.[2] Only 15.5% of the *Mazurka-5*'s normalized tempi are in the interval between 0.96 and 1.04—the often used ±4% tolerance interval for stable tempi [64]. For comparison, 90.9% of the *Ballroom* [64, 93] dataset's normalized tempi are in the same interval. Obviously, the two datasets are very different w.r.t. intra-track tempo stability.

While the ±4% interval is illustrative when categorizing stable vs. unstable, it is a rather arbitrary threshold (see also Section 6.2.1). Arguably, the standard deviation of a track's normalized tempi is better suited to describe intra-track tempo variability. It is identical to the coefficient of variation introduced in Equation (5.1), which is defined as the ratio between the standard deviation σ and the mean μ.

We show this IBI-based c_{var}-value for our example Op. 68, No. 3 (Cohen, 1997) as a horizontal gray line in Figure 8.3a. As discussed in Section 8.1, instantaneous tempo values like IBIs tend to overestimate the variance of a musically meaningful local tempo for expressive music. From a musical point of view, it is therefore more appropriate to analyze tempo stability of Mazurkas

[2]The comb pattern is a consequence of the 10 ms resolution of the original annotations.

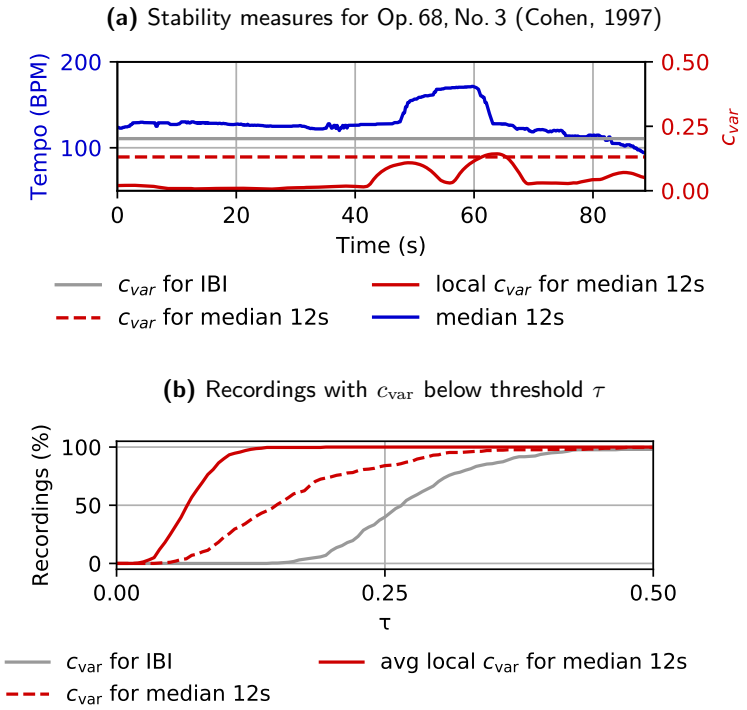


Figure 8.3.: **(a)** Local tempo (blue line) and stability (c_{var}) for Op. 68, No. 3 (Cohen, 1997). c_{var} based either on IBIs (gray line), the (sampled) median tempo over 12 s intervals (dashed red line), or the averaged local c_{var} over 12 s segments of median tempi (solid red line). **(b)** Percentage of recordings with $c_{\text{var}} \leq \tau$.

not based on individual IBIs, but on the basic tempo, which we approximate (for the purpose of this discussion) with the median tempo over 12 s segments (Figure 8.3a, blue line). Sampling the local median tempo allows us to calculate an arguably more appropriate c_{var} (Figure 8.3a, dashed red line), which lies well below the gray line, indicating higher stability. This however, still ignores the fact that Mazurkas may contain multiple sections with stable but different tempi. We can take this into account by calculating local coefficients of variation for short segments of the median-based tempo curve. The solid red curve in Figure 8.3a shows the results for overlapping 12 s-segments. For most of the recording it is very low. Only in the transitional regions, at the beginning and end of the CDD-section, we see higher values. Note that by averaging the local c_{var} we can obtain a measure for intra-*segment* stability, while the two track-level c_{var} measures represent intra-*track* stability. Figure 8.3b depicts how many recordings have a c_{var} below a threshold τ for all three ways of calculating it. The comparison shows that for *Mazurka-5* intra-segment variability is far smaller than intra-track variability.

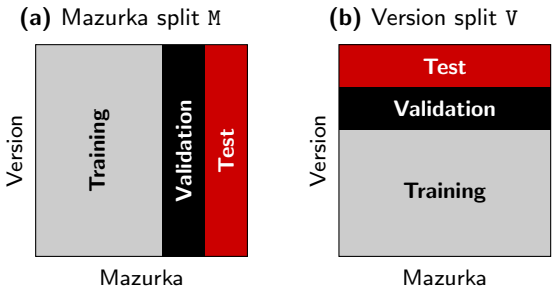


Figure 8.4.: Dataset splitting into training, validation, and test sets.

8.3. Experiment

We now investigate how different local tempo estimation systems perform when tested with *Mazurka-5*. We consider the following systems: The RNN-based beat tracking system `böck`[3] [7] (estimated beats are aggregated identically to the ground truth), the CNN-based tempo estimation system `DeepTemp` from Section 7.1.3.2—scaled with model sizing parameter k=16—and the system `DT-Maz`, which is set up identically to `DeepTemp`, but has been trained on *Mazurka-5* recordings instead of Pop/Rock, EDM, and Ballroom music. Based on our observations in Section 8.1 and informal experiments with several segment lengths we model the local tempo with median-aggregated IBIs from 11.9 s segments.

8.3.1. Setup

We trained `DT-Maz` from scratch[4] on *Mazurka-5* recordings using 5-fold cross-validation with two different kinds of splits, `M` for Mazurka and `V` for version (or performance). For `M`, each split contains all versions of one Mazurka (Figure 8.4a). For `V`, each split consists of a disjoint 5^{th} of all versions of each of the five Mazurkas (Figure 8.4b). During training, three splits were used as training data and one for validation. The remaining 5^{th} split was used for testing. Each split was used exactly once for validation or testing. We refer to the models trained on `M`-splits as $\texttt{DT-Maz}_{\texttt{M}}$ and to the `V`-split models as $\texttt{DT-Maz}_{\texttt{V}}$. The employed training procedure was very similar to the one described in Section 4.2.3. Audio is first converted to mel-magnitude-spectrograms. Then samples with the dimensions $F \times T$ are used as network input. $F = 40$ being the number of frequency bins covering the frequency range $20 - 5{,}000$ Hz, and $T = 256$ being the number of time frames with a length of 46 ms per frame, corresponding to 11.9 s. We further use scale-&-crop data augmentation with time scale factors drawn from $\mathcal{N}(1,\, 0.1)$, but limited to $[0.7, 1.3]$ to avoid extreme distortions. After augmentation, samples are standardized to zero mean and unit variance. Like in Section 4.2.3, we cast tempo estimation as classification problem and use

[3] `https://github.com/CPJKU/madmom` with default parameters.
[4] Transfer learning on the `DeepTemp` model led to very similar results.

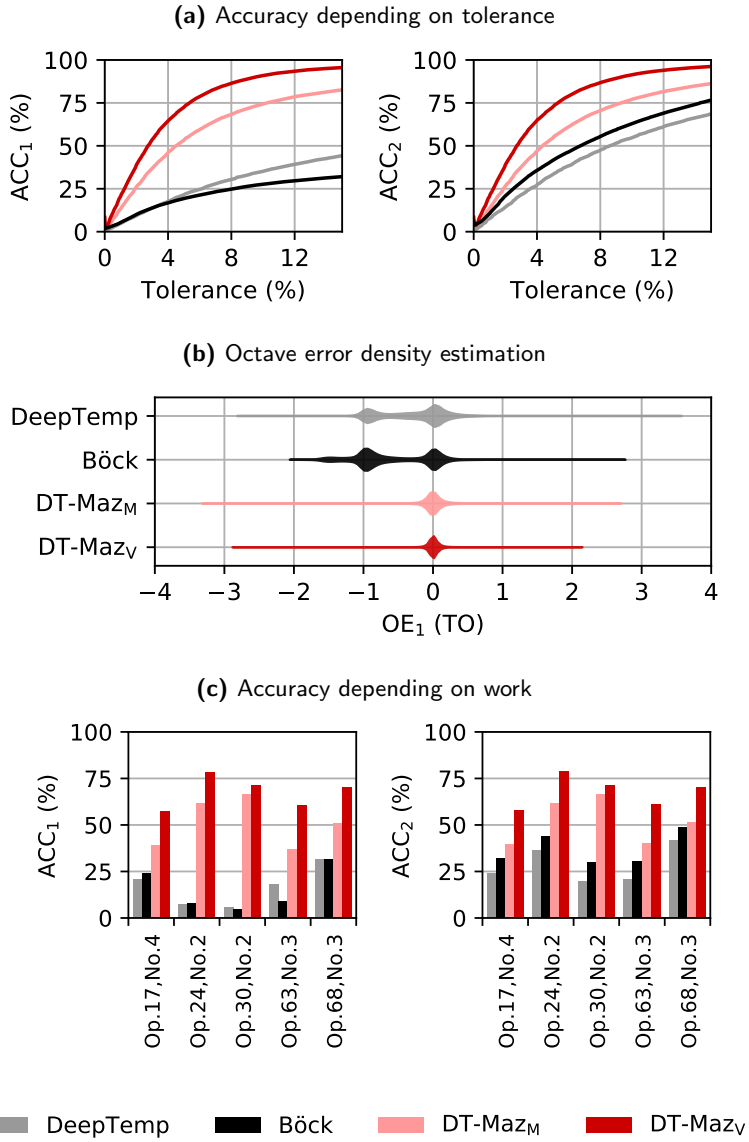


Figure 8.5.: **(a)** Local ACC_1 and ACC_2 depending on accuracy tolerance **(b)** Density estimation for OE_1 **(c)** Local ACC_1 and ACC_2 for the five Mazurkas

categorical cross-entropy as loss function. Adam [84] is used as optimizer with a batch size of 32 and an initial learning rate of 0.001. The rate is halved once the validation loss stops improving and training is continued with the best performing model up to that point (stepwise annealing). We repeat this at most 10 times. If reduction does not lead to a lower validation loss three times in a row, training is stopped. To avoid overfitting to longer recordings, we ensure that samples from all training recordings are presented with the same frequency.

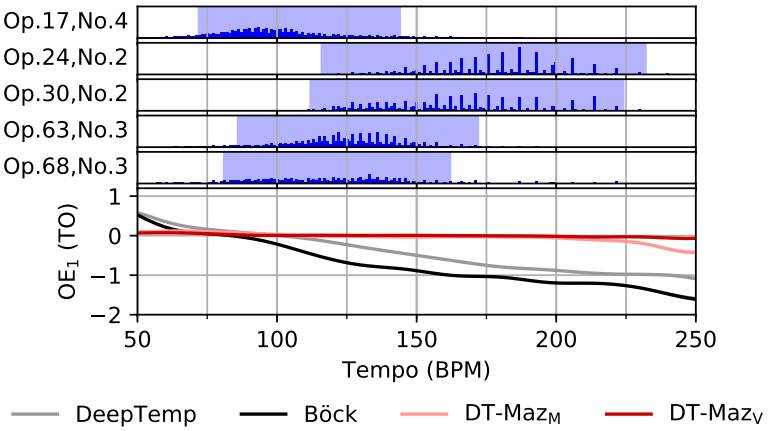


Figure 8.6.: (top) Sweet octaves **(bottom)** Estimates of generalized additive models fit to OE_1/tempo-pairs

8.3.2. Evaluation

To evaluate, we estimate the tempo for a sliding segment with length 11.9 s (256 frames) and a hop size of 186 ms (4 frames) over all recordings. As metric we use ACC_1 (tempo accuracy) and ACC_2 (accuracy allowing *octave errors*) from the global tempo estimation task [64], which are meant for music with low intra-track tempo variability. This is reasonable, because we apply the metric locally for each segment, so that the tolerance does not have to correspond to intra-track, but to intra-segment variability, and as we have shown in Section 8.2, intra-segment variability is relatively low. Nevertheless, we consider the typical 4% tolerance an arbitrary threshold and therefore plot accuracy values for the tolerance interval $0 - 15\%$ in Figure 8.5a. For both variants of `DT-Maz`, ACC_1 values are higher than for the other systems, regardless of tolerance. Not surprisingly, ACC_1 values are also generally higher for higher tolerances.[5] The best performing system for the tolerances 4%, 8%, and 12% is $\texttt{DT-Maz}_\texttt{V}$ with remarkable 64.6%, 86.4%, and 93.5%. The worst performing system is `böck`, with 16.8%, 24.8%, and 29.7%. For ACC_2 the best performing system is also $\texttt{DT-Maz}_\texttt{V}$ with 64.8%, 86.8%, and 94.0%, and the worst performing system is `DeepTemp` with 27.3%, 47.5%, and 61.2%. In the following paragraphs we discuss the most prominent errors, namely octave errors, tempo stability related errors, and problems with specific musical properties.

8.3.2.1. Tempo Octave

Using violin plots, Figure 8.5b depicts kernel density estimates (KDE) of the octave error OE_1 introduced in Section 6.6.3. Identifiable by the very dense section around −1 Tempo Octaves (TO), `DeepTemp` and `böck` suffer most from underestimating the actual tempo. As Figure 8.5c shows,

[5] To keep the evaluation concise, the reported local accuracy in all following accuracy figures use 4% tolerance.

octave errors are not evenly distributed among the five Mazurkas. Op. 24, No. 2 and Op. 30, No. 2 are much more affected than the other three. This can be partially explained by the fact that on average versions for Op. 24, No. 2 and Op. 30, No. 2 are performed much faster than the other three Mazurkas. Their *sweet octaves* (Section 3.2.1.3) are [116, 232) and [112, 224) BPM, while the sweet octaves for the other three are [72, 144), [86, 172), and [81, 162) BPM (Figure 8.6, top). A closer investigation shows that for the tested Mazurkas, both `DeepTemp` and `böck` lean towards negative octave errors for higher tempi, revealing an *octave bias* (Section 3.2.1.3). This is visualized in Figure 8.6, bottom. It shows the estimates of generalized additive models (GAM) fit to measured OE_1 per reference tempo. In other words, it illustrates what kind of estimation error we can expect depending on a given true tempo. For tempi greater than 100 BPM, `böck` and `DeepTemp` tend to suffer from negative octave errors.

8.3.2.2. Stability

Figure 8.7a shows that accuracy is higher when considering only segments with low c_{var}-values—our proxy for tempo variability. When only considering relatively stable segments with $c_{\text{var}} < 0.025$ (Figure 8.7b), the accuracy scores for all five Mazurkas increase substantially. But at least for $\texttt{DT-Maz}_{\texttt{M}}$, scores for Op. 17, No. 4, Op. 63, No. 3, and Op. 68, No. 3 remain well below those for Op. 24, No. 2 and Op. 30, No. 2. Apparently, differences in stability cannot fully explain differences in accuracy for the five works.

8.3.2.3. Musical Properties

We have seen in Figure 8.7b that even for stable segments, $\texttt{DT-Maz}_{\texttt{V}}$ performs better than $\texttt{DT-Maz}_{\texttt{M}}$. To find out why, we exploit beat annotations for each recording of the five Mazurkas. They allow us to compute stability and the absolute octave error $|OE_1|$ for 11.9 s segments with a beat at their center, i.e., stability and error on a musical time axis. Using musical time, we can summarize errors and stability measures in a *cross-version* fashion by averaging per beat over all recordings of a given Mazurka (Figure 8.8). Figure 8.8a shows the results for Op. 17, No. 4 and as expected, the $\overline{c_{\text{var}}}$-curve roughly correlates with errors by both $\texttt{DT-Maz}_{\texttt{M}}$ and $\texttt{DT-Maz}_{\texttt{V}}$. For $\texttt{DT-Maz}_{\texttt{M}}$ we see four large additional peaks around beats 42, 89, 162, and 305 (highlighted in light-blue). These peaks loosely correlate with the occurrence of dense mixtures of ornamented beats (red ■, trills, grace notes, and arpeggios) and weak bass beats (cyan ■, only the left hand is played), i.e., piece-dependent musical properties (classification from [69]), which are apparently the main reason for the difference in accuracy. The same kind of graph for Op. 63, No. 3 is shown in Figure 8.8d. It also has large additional peaks for $\texttt{DT-Maz}_{\texttt{M}}$, which do not coincide with tempo instabilities. In this case we cannot provide beat classes that explain the errors. The fact that they occur in particular sections and that $\texttt{DT-Maz}_{\texttt{V}}$ does not suffer from them supports the

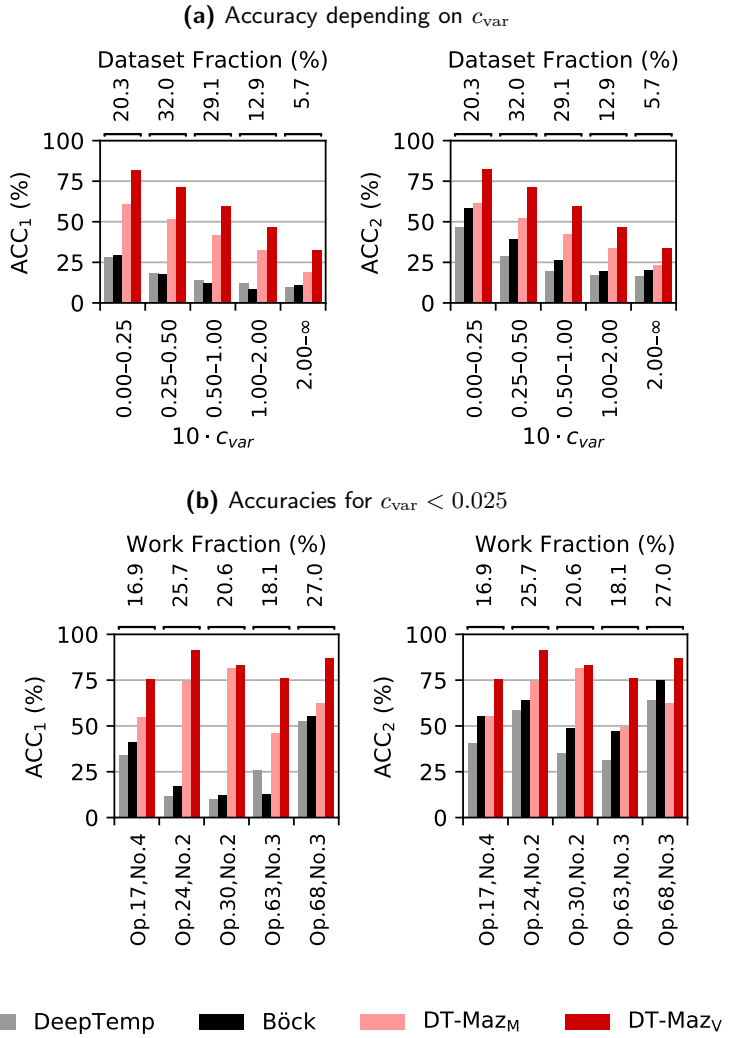


Figure 8.7.: **(a)** Local ACC_1 and ACC_2 considering c_{var} ranges. **(b)** Accuracies for segments with $c_{\text{var}} < 0.025$.

argument that these errors are caused by musical properties of the work. Trained on the `V`-split, $\texttt{DT-Maz}_{\texttt{V}}$ was able to learn piece-specific musical properties and generalize them across versions. This implies that expecting $\texttt{DT-Maz}_{\texttt{M}}$'s accuracy levels is more realistic when using either of the two models on completely unseen Mazurkas.

8.4. Discussion and Conclusions

With five Chopin Mazurkas as use case, we have shown that local tempo for expressive music can be modeled using median aggregated IBIs, and tempo stability can be measured using the coefficient of variation (c_{var}) of tempo values. Using these tools, we have found that the five Chopin Mazurkas exhibit high intra-track tempo variability, but low intra-segment variability,

i.e., the local tempo is relatively stable and thus musically meaningful. This has allowed us to conduct a local tempo estimation experiment. The results show that a standard beat-tracker like `böck` and a tempo estimation CNN like `DeepTemp`—trained on Pop, EDM, and Ballroom music—perform relatively poorly for Mazurkas. Even when ignoring tempo octave errors, the results are by far inferior to those achieved by the same kind of CNN as `DeepTemp`, but trained on recordings from the target genre. It is reasonable to assume that training the `böck` system on Mazurkas would also improve performance substantially—at the price of a strong genre bias (see Section 3.2.1.4). Through our experiments, we have been able to confirm a relationship between estimation accuracy and tempo stability measured in c_{var}. Arguably, segments with a very high c_{var} may not have a meaningful local tempo. Via comparison of local accuracy results for `DT-Maz`-models trained on either the piece-wise split across Marzurkas (`M`-split) or the performance-wise split across versions (`V`-split), we have been able to identify piece-specific, musically difficult passages. When training and testing on the `V`-split, the network apparently has a chance to learn these piece-specific features not covered by data augmentation. One might also argue, $\texttt{DT-Maz}_{\texttt{V}}$ overfits to the pieces ("cover song effect"). As with all deep learning systems, performance depends largely on the training data. For a production system, one is therefore well advised to use a larger and more diverse training set than we did in this case study.

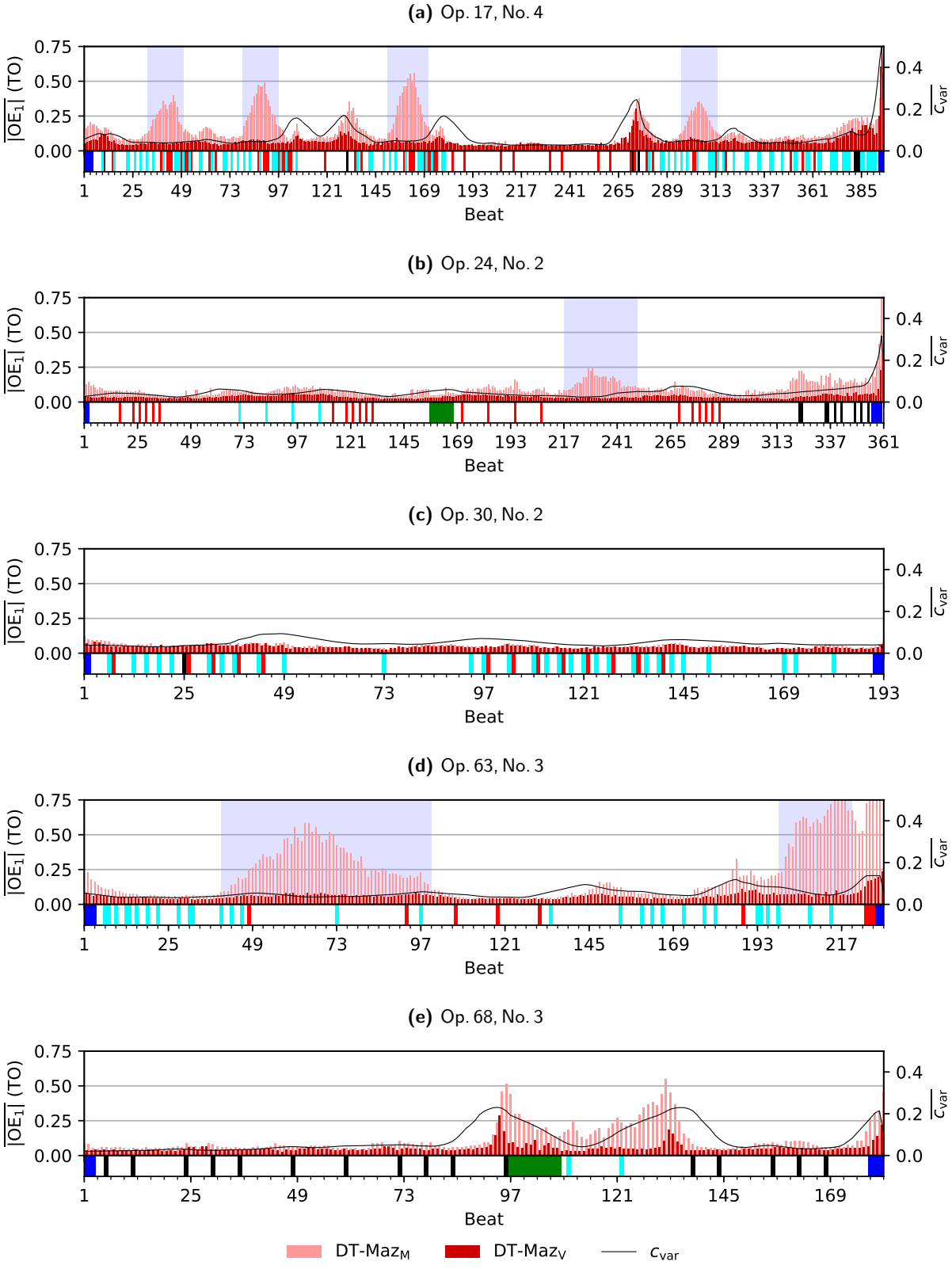


Figure 8.8.: Averaged c_{var} and $|\text{OE}_1|$ around beats with classifications from [69]: non-event beats (black ■) , boundary beats (blue ■), ornamented beats (red ■), constant harmony beats (green ■), and weak bass beats (cyan ■). Sections with high OE_1-values that cannot be explained by tempo instability, i.e., high c_{var}-values, are highlighted with a light-blue background.

9. Schubert Winterreise: A Cross-Version Study on Local Key Estimation

In this chapter, we discuss local key estimation comparing an HMM and a CNN approach using a cross-version dataset. It is based on joint work with Christof Weiß, which is currently under review [166]. Christof Weiß contributed local key annotations, HMM implementation and training, as well as the musicological evaluation and parts of the discussion. My main contributions are the CNN implementation and training, as well as parts of the discussion.

The tonal analysis of music audio recordings is of high relevance for both musicologists and music listeners and therefore constitutes a central task in MIR research. Notions of tonal structures relate to different temporal scales. Many researchers have focused on local (i.e., temporally concentrated) structures such as *chords* [21, 81, 89, 173, 108]—loosely defined as sets of pitches that are perceived as an entity. In contrast, the *global key* describes the tonality of a whole song, piece, or movement. It can be defined as a set of pitch relationships that establishes a particular major or minor chord as a tonal center [142], attaining a subjective sense of arrival and rest [27]. In this chapter, we consider the intermediate notion of *local key*, which relates to mid- and large-scale segments of a piece.

For the global key in Western classical music, the beginning and ending sections [189] and the final chord [193] play an important role, and the key label is often provided by the composer as part of the title. Contrasting this global view, the musical key may also change over the course of a piece, thus calling for a *local key* analysis. When the harmonic structure prepares the arrival of the new key, we speak of a modulation [142]. Modulations often proceed gradually over a certain time span leading to ill-defined segment boundaries. Furthermore, some keys are closely related to each other such as *relative keys* (e.g., C major $\leftrightarrow$ A minor), which share the same underlying diatonic scale. Some researchers therefore focus on the 12-class problem of diatonic scale detection [202, 194]. There is also a high similarity between parallel keys (C major $\leftrightarrow$ C minor) or fifth-related keys (C major $\leftrightarrow$ G major), whose associated scales largely overlap by sharing many pitch classes. Due to these issues, local key estimation (LKE) is a challenging

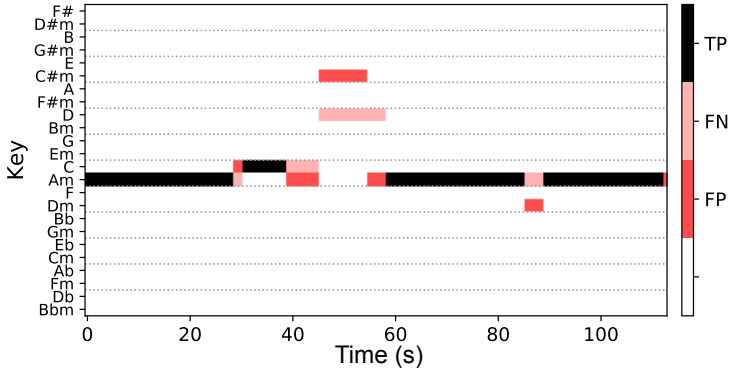


Figure 9.1.: Local key predictions of the CNN model for song 15 "Die Krähe" from Schubert's song cycle *Winterreise*, performed by T. Quasthoff and C. Spencer (1998). Dark red bars indicate false positives, brighter red bars false negatives, black bars true positives.

task where annotations are often ambiguous and highly subjective by nature. Several approaches therefore avoid the "hard" detection of keys and boundaries and propose multi-scale [146, 65], self-referential [80], or probabilistic [134, 194, 195] visualization techniques instead. Figure 9.1 shows a visualization of LKE results with an arrangement of keys according to the circle of fifths—thus showing closely related keys next to each other. At second 40, we observe a confusion with the relative key and around second 90, a confusion with a fifth-related key.

To address automatic LKE from audio recordings, different methods have been proposed. Traditional approaches combine chroma features with template-based recognition [79, 124]. For segmentation and post-filtering, many researchers used Hidden Markov Models (HMMs) [18, 202] with non-negative matrix factorization (NMF) as an alternative [79]. As we know from chord estimation research [21, 81], HMMs are useful mainly due to the context-sensitive smoothing effect and less due to their quality as a language model for key transitions. Several methods address chord estimation, LKE, and (down-)beat tracking simultaneously [141, 107, 124]. Recently, deep-learning techniques have become popular for chord estimation [173, 108] and global key estimation [90, 91, 163] in music recordings. Korzeniowski et al. [91] successfully used convolutional neural networks (CNN) to estimate the global key for music recordings across different genres. Though we are not aware of any research using deep neural networks for LKE, this is an obvious endeavor due to the task's similarity to chord and global key estimation—both of which have been tackled successfully using CNNs.

While most audio-based LKE systems were developed and tested on popular music, Western classical music has rarely been approached. Mearns et al. [113] analyze modulations in synthesized recordings of twelve chorales by J. S. Bach. Papadopoulos and Peeters [124] consider recordings of Mozart's piano sonatas. Weiss et al. [195] provide visualizations of local key regions in Wagner's operas. Compared to popular music, the use of closely related keys and gradual modulations are particularly prominent in classical music. Moreover, many classical music styles involve altered

ID	Singer	Pianist	Year	Duration
AL98	Thomas Allen	Roger Vignoles	1998	1:13:33
FI55	Dietrich Fischer-Dieskau	Gerald Moore	1955	1:14:35
FI66	Dietrich Fischer-Dieskau	Jörg Demus	1966	1:11:23
FI80	Dietrich Fischer-Dieskau	Daniel Barenboim	1980	1:13:07
HU33	Gerhard Hüsch	Hanns-Udo Müller	1933	1:07:31
OL06	Thomas Oliemans	Bert van den Brink	2006	1:14:42
QU98	Thomas Quasthoff	Charles Spencer	1998	1:12:24
SC06	Randall Scarlata	Jeremy Denk	2006	1:06:45
TR99	Roman Trekel	Ulrich Eisenlohr	1999	1:15:21

Table 9.1.: Cross-version dataset of Franz Schubert's *Winterreise*.

chords featuring non-scale tones that make LKE even harder. As a peculiarity of classical music, there are usually many recorded performances (interpretations) available. Together with other representations, such as symbolic scores, we consider these as individual *versions* of an abstract musical work. Exploiting several such versions in a *cross-version scenario* allows for studying and improving the robustness and generalization for various tasks such as chord [89] and scale analysis [195] or singing voice detection [32, 114].

In this chapter, we study LKE within a cross-version scenario. We make use of a dataset comprising nine recorded performances (referred to as *versions*) of Franz Schubert's 24-song cycle *Winterreise* [65]. Using measure annotations as anchor points, we semi-automatically generate local key annotations [1, 65]. We propose a straightforward LKE approach based on a CNN and compare it to a traditional method using chroma features and HMMs. In our experiments, we evaluate the efficacy of both methods and systematically assess their robustness. As our main contribution, we investigate the effect of using different training–test splits that require generalization across versions, songs, or both. Furthermore, we conduct an in-depth analysis and investigate musical reasons for key confusions.

The chapter is organized as follows. In Section 9.1, we start with the description of the dataset and our training–test scenarios. We proceed in Section 9.2 with introducing the technical approaches (HMM and CNN). In Section 9.3, we then discuss our results in detail. We draw our conclusions in Section 9.4.

9.1. Cross-Version Dataset

In this section, we describe our dataset and annotation procedure followed by the different splits used for training, validation, and testing.

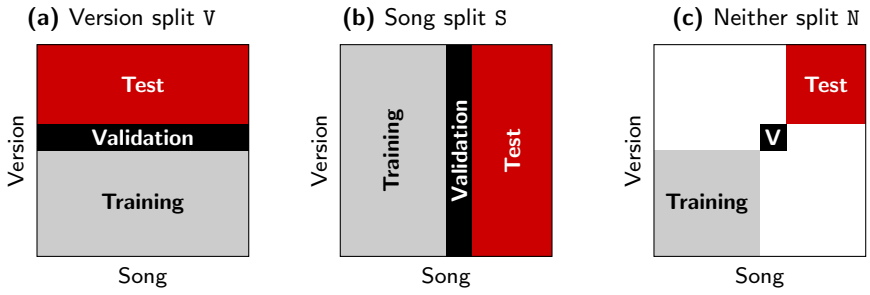


Figure 9.2.: Dataset splitting into training, validation, and test sets.

9.1.1. Dataset

Franz Schubert's song cycle *Winterreise* (Winter Journey, published 1828) consists of 24 songs for voice (originally tenor) and piano. The individual songs differ in length and complexity. Some songs are harmonically unambiguous showing distinct key regions of diatonic pitch content (No. 2) or being based on a single tonic chord (No. 24). Other songs involve many altered chords (No. 10) and ambiguous key regions (No. 16). Inspired by previous analyses [1, 65], each song has been annotated on score level (musical time axis) with continuous local key segments by the professionally trained musician Christof Weiß. Since the local key is sometimes ambiguous, our annotations differ from [1, 65] in several respects: We did not label unclear or transitional passages with "no key" but decided on the most likely key. Furthermore, we ensured a certain continuity of the key segments.

Our dataset [65] comprises nine complete performances by different duos, recorded in a studio setting (Table 9.1). On average, each song lasts 3 min (σ=1:10 min), ranging from 0:44 min (No. 18, SC06) to 6:18 min (No. 1, OL06). We manually annotated measure positions for two recordings (HU33, SC06) and automatically transferred these to the other recordings using synchronization techniques as proposed in [200]. Using the measure positions as anchor points, we semi-automatically transferred the local key regions from the score level to the nine recordings (physical time axis).

9.1.2. Splits

To train our models and optimize their hyperparameters, we split our dataset into training, validation, and test subsets so that each song in each version is analyzed exactly once in a cross-validation procedure. Since our dataset has a specific structure, we can split along two axes—the "version axis" and the "song axis" (see Figure 9.2). In order to systematically investigate the models' efficacy when trained in different ways, we create three different splits:

- **Version split** `V` (Figure 9.2a). The training subset contains all songs in five versions, the validation subset all songs of one version, and the test subset all songs of three versions. In this case, the models can exploit their knowledge of the abstract musical structure (harmonic progressions), but have to generalize to unseen *acoustic conditions* and different interpretations, which is not trivial.
- **Song split** `S` (Figure 9.2b). The training subset contains recordings of 13 songs in all nine versions, the validation subset three songs in all versions, and the test subset eight songs in all versions. The models have to generalize to unseen *musical pieces* with different harmonic properties but can adapt to the acoustic conditions of each version during training.
- **Neither split** `N` (Figure 9.2c). In this strict split, the training subset contains 19 songs in four versions, the validation subset two other songs in two other versions, and the test subset three other songs in three other versions. Thus, the model knows neither song nor version and has to generalize across both axes. This is the only split where not all data is used in one fold, and it is the most realistic one.

To ensure comparability, we fix the exact versions and songs in each of the splits (no randomization) for both models.

9.2. Methods

We present two approaches for LKE. The first one is a baseline system relying on HMMs, the second one uses CNNs.

9.2.1. HMM-Based Method

Our first system, denoted as `HMM`, relies on the extraction of chroma features using the filter-bank method proposed in [118].[1] We post-process the filter-bank output (pitch features) by applying logarithmic compression with a parameter $\gamma \in \{100, 1000, 10\,000\}$ and apply pitch weighting to emphasize the pitch range around C4 [21]. We smooth the resulting 10 Hz chroma features with a median filter of length $\lambda \in \{81, 85, \ldots, 157\}$. On the training subset, we learn Gaussian models for the 24 keys (assuming enharmonic equivalence) in the chroma space $\mathbb{R}^{12}$. We cyclically average the major and the minor key model over the chroma dimension in order to achieve transposition-blind models, which we then use for generating the HMM emission probabilities. Inspired by [21], we apply a uniform, diagonal-enhanced transition matrix with a self-transition probability of $1 - \sigma$ where $\sigma \in \{10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}\}$. Using these HMM parameters, we run Viterbi decoding to predict a key label for every 10 Hz frame. We optimize the parameters γ, λ,

[1] We use the *librosa* implementation [109].

and σ on the validation subset. That way, we exploit the available data in a way similar to the CNN-based system described next.

9.2.2. CNN-Based Method

The second system, denoted as `CNN`, is set up identically to the global key estimation network `DeepSquare` from Section 7.1.3.2 (Figure 7.3), scaled with the model sizing parameter k=8 and dropout probability p_{D}=0.3. Overall, the network has 293 296 trainable parameters. The employed training procedure is similar to the one described in Chapter 7. We first convert the audio to constant-Q magnitude spectrograms. Then, we use samples of dimension $F \times T$ as input to the network. F=168 is the number of frequency bins covering a frequency range of seven octaves with a frequency resolution of two bins per semitone. T=60 is the number of time frames with a resolution of 0.19 s per frame, i.e., 60 frames correspond to 11.1 s. To account for class imbalances within the major or minor keys, we randomly shift each spectrogram along the frequency axis by $\{-4, -3, \ldots, 6, 7\}$ semitones and adjust the ground truth labels accordingly. Since key estimation is a single-label, multi-class problem, we use categorical cross-entropy as loss function. Adam [84] is used as optimizer with a batch size of 32 and an initial learning rate of 0.001. Once the validation loss plateaus, we halve the learning rate and continue training with the best performing model up to that point (stepwise annealing). We repeat this at most ten times. If reduction does not lead to a lower validation loss three times in a row, we stop training.

9.3. Results

In order to evaluate LKE on classical music, we trained both systems on recorded songs from Schubert's *Winterreise* using different data splits. As general evaluation measure, we compute the accuracy while ignoring "no key" regions (which only occur at the beginning and ending of a piece). Moreover, we analyze musically explainable key confusions such as relative, parallel, and fifth-related keys. In the following, we discuss the results with a focus on the data splits and musical key confusions.

9.3.1. Detailed Results

We first consider the realistic split `N`, where neither test songs nor test versions are seen during training or validation. Figure 9.3a depicts the `HMM`'s results. With most songs, we observe a similar accuracy for the different versions. However, the accuracy varies greatly between songs. For example, song no. 1 reaches high accuracies around 93% for all versions, which is expected due to its clear harmonic structure. In contrast, song no. 10, which is highly chromatic, shows

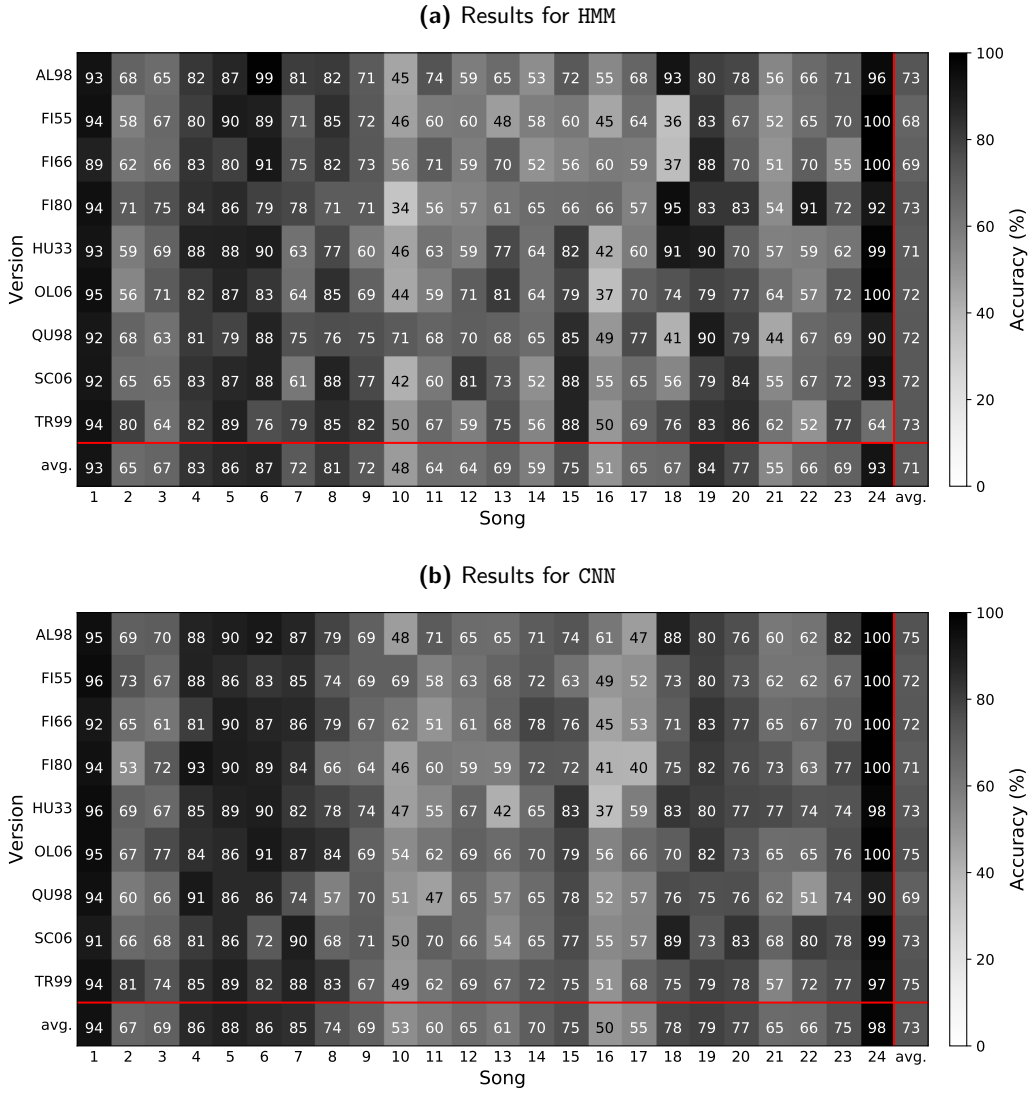


Figure 9.3.: Individual accuracy values (in percent) per song and version for the strict neither split `N`.

low accuracies around 50%. For few songs, we observe higher variance along the version axis. An example for such an outlier is song no. 18, whose accuracy is strongly version-dependent. Investigating these results in detail, we find that this is a very short song of approx. 45 seconds, whose beginning and ending sections are monophonic (unisono) without any chords in the piano, thus posing a particular challenge. The `HMM`'s tendency to stay in a key reinforces the impact of such errors on the overall accuracy. Comparing the `HMM`'s results with the `CNN`'s (Figure 9.3b), we observe similar tendencies in both plots. With an average of 73%, the `CNN` performs only marginally better than the `HMM` trained on split `N` (71%). For the `CNN`, accuracies are also similar across different versions of a song. Moreover, the variation across songs is very similar to the `HMM`'s results, which indicates that *musical* properties of the individual songs may pose the main challenge for both systems.

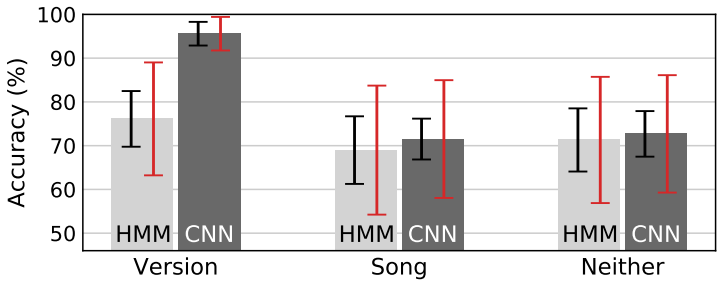


Figure 9.4.: Average accuracies for songs based on different splits (`V`, `S`, `N`) and models (`HMM`, `CNN`). Black errorbars denote standard deviations *across versions* (averaged over all songs), red errorbars denote standard deviations *across songs* (averaged over all versions).

9.3.2. Data Splits

To statistically summarize these results, we report the average per-song accuracy values in Figure 9.4. For the "neither" split `N`, the two right-most bars correspond to the overall averages (lower-right values) in Figures 9.3a and 9.3b. Black errorbars indicate the average standard deviation over all versions of a song ("vertical direction" in Figure 9.3). Red errorbars denote the average standard deviation over all songs of a version ("horizontal direction" in Figure 9.3). The standard deviation across songs (14.4% for `CNN`) is substantially greater than across versions (5.2% for `CNN`), which confirms our observation that the accuracy variance can be traced back more to differences in songs than in versions. This also holds for the other splits `V` and `S` depicted in Figure 9.4. Comparing the average accuracy between splits, we find that the "song split" `S` leads to similar results (69% for `HMM`, 72% for `CNN`). Interestingly, accuracies are a bit lower than for `N`, despite having more training data available in each step. In `N`, the split between training and validation is stricter, which might lead to more robust systems. Contrary to findings for genre classification [123], we found no advantage for either system to being exposed to other versions from the same CD recording, i.e., no observable "album effect." Looking at the "version split" `V`, we find a remarkable result. Accuracies are considerably higher with 76% for `HMM` and 96% for `CNN`. Both systems apparently have a capacity to learn the specific *musical* characteristics of the individual songs (resp. their specific annotations), with the `CNN`'s capacity being greater than the `HMM`'s. The `CNN` system is effectively (over)fitting to harmonic progressions in the songs. Therefore, we cannot expect it to perform similarly well for other classical songs—we might call this a "cover song effect." Interestingly, generalization to unseen acoustic conditions works well, especially for the `CNN`. Having several versions available for training (and validation) seems to build up the model's robustness against version differences and thus, is sufficient for avoiding the "album effect."

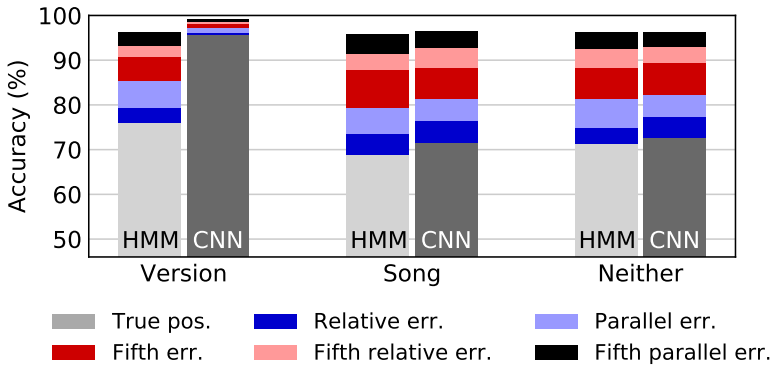


Figure 9.5.: Accumulated accuracy including different types of musically explainable key confusions.

9.3.3. Musical Key Confusions

Finally, we want to discuss the specific types of confusions in the models' predictions. Figure 9.5 shows the percentage of frames with certain prediction errors on top of the accuracy (true positives). We report confusions with the relative key (e.g., C major ↔ A minor), the parallel key (C major ↔ C minor), fifth-related keys (C major ↔ G major, or C major ↔ F major), as well as the relative of fifth-related keys (C major ↔ E minor, or C major ↔ D minor) and the parallel of fifth-related keys (C major ↔ G minor, or C major ↔ F minor). For both the `S`- and the `N`-split, we see that the most common errors are fifth errors, parallel key errors, relative key errors, relative fifth errors, and parallel fifth errors (roughly in this order). Together, these errors explain most of the performance gap between the `CNN` trained on the `V`-split and either system trained on one of the other splits (Figure 9.5). Gray and dark blue bars together constitute the accuracy for estimating the correct diatonic scale [194, 195]. Correct predictions and all musical errors together comprise $\approx 95\%$ of all frames. From this, we conclude that it is most challenging for the models to learn how *musically ambiguous* regions have to be labeled in order to predict the local key label as given by a specific annotator. For a detailed visualization of which *kind* of error each model is prone to for each measure of the 24 songs, please see Appendix C.

9.4. Conclusions

We approached the task of local key estimation in classical music recordings and systematically explored the efficacy of an HMM-based and a CNN-based approach. Using a cross-version dataset of Schubert's song cycle *Winterreise*, we trained, validated, and tested both systems on splits along songs or versions. Moreover, we explored a strict split where *neither* test songs *nor* test versions are shown during training. For the song and the "neither" split, we found that `CNN` and `HMM` models perform similarly well, reaching an accuracy of approx. 70%. Most of the observed errors can be explained through musical ambiguities where annotations are often subjective.

Comparing results for different splits, we showed that generalization across versions ("album effect") does not pose major problems for the models. Knowing the specific songs (version split) leads to clearly higher results, especially for the `CNN` (96%). We call this the "cover song effect." While this is beneficial in our scenario, it means that the `CNN` (over)fits to the harmonic progressions of specific songs and learns how a single annotator labeled these songs. Song and "neither" split therefore show more realistic results as can be expected for unseen pieces. Yet, providing `CNN` models with more training data covering a wide variety of harmonic progressions should have high potential for improving local key estimation systems in general.

10. Summary and Future Work

In this thesis, we explored data-driven solutions to the classic MIR tasks tempo and key estimation. Starting with traditional digital signal-processing methods, we applied increasingly complex machine learning techniques, essentially moving from handcrafted to data-driven solutions.

Beginning with a basic handcrafted system, we used linear regression and random forests on engineered features with the goal of correcting already reasonable results (Chapter 3). Machine learning was used as an "add-on", not as the core of the solution. This differentiates the first part of this thesis from the second. Realizing that the presented basic tempo estimation pipeline with its two-level Fourier transforms is just another computational graph with weights that can be learned, we searched for a suitable problem formulation that can be expressed in the shape of a DNN. This led to our first CNN-based tempo estimation system proposed in Chapter 4. Its architecture reflects a key insight from our signal-processing method, namely first matching short-time patterns ("onsets") as building blocks for detecting long-time periodicities ("tempo"). Following the same design principle, we showed in Chapter 7 that relatively shallow, but highly specialized DNNs can reach competitive performance for both the tempo and the key estimation task. Ultimately, they may be inferior to deep architectures that have a greater capacity to learn a large range of rhythmical patterns. Nevertheless, we believe that building ML systems by exploiting domain knowledge is key to a better understanding and thus better solutions. We have further demonstrated how knowing the subject matter and available data is essential for approaching MIR tasks in Chapters 8 and 9. In both chapters the CNNs used are primarily tools that help us explore cross-version datasets and their musical properties. The interesting insights in both chapters are not found in how well the CNN performs, but what we can conclude from its successes and failures depending on how we split the data.

Data is what working ML models are made from. That is why it is so important that we create, analyze, and correct datasets as we have done throughout this work. As shown in Chapter 5, there is much to be questioned and improved. Just because we have always used this dataset or that metric does not make achieved evaluation results automatically meaningful or useful. We therefore critically discussed tempo estimation evaluation in Chapter 6. It is simply not enough to search for better methods. The way we evaluate these methods has to be constantly challenged as well. We were able to show that use cases, metrics, and dataset are in need of improvement and proposed suitable remedies, namely new metrics and an open evaluation repository.

We firmly believe that methods for MIR tasks like tempo and key estimation will continue to improve thanks to larger and better datasets and advances in ML. This may also make it easier to learn directly from audio signals instead of spectrograms [30, 132]. Since annotating music datasets tends to be expensive, interesting solutions may arise from semi-supervised or multi-task learning [5, 8]. Very likely, we will also see progress w.r.t. explainability [23], to better understand not only how to learn, but also what is learned in order to gain true insights into music and MIR—and not just ML.

A. Implementations: Tempo Estimation Systems

This appendix gives an overview of implementations for the tempo estimation systems we proposed in this thesis.

A.1. Global Features and Linear Regression

An implementation of the algorithm described in Section 3.1 is available at `https://www.audiolabs-erlangen.de/fau/assistant/schreiber/data/schreiber_icassp2014.zip` and `http://www.tagtraum.com/download/schreiber_icassp2014.zip`.

The software requires Java 6 or later.

To start, execute one of these three programs:

```
$ java -cp jipes-0.9.7.jar:schreiber_icassp2014-0.0.1.jar BPM <input> [output]
$ java -cp jipes-0.9.7.jar:schreiber_icassp2014-0.0.1.jar Mirex <input> [output]
$ java -cp jipes-0.9.7.jar:schreiber_icassp2014-0.0.1.jar SNM82 <input> [output]
```

- BPM: Produces just the BPM
- Mirex: Output as required in the MIREX competition
- SNM82: The mean spectral novelty feature with kernel length 82

Note, that on Windows the path separator between the two jars is not `':'`, but `';'`. Both jars have to reside in the current directory (unless you edit the classpath accordingly). The output is either written to the given file or, if omitted, to standard out. Supported audio input formats are `.au` and `.wav` with samples rate that are multiples of 11,025 Hz.

The code was implemented using the open source audio feature extraction framework *jipes* [153].

A.2. Octave Error Correction with Random Forests

An implementation of the algorithm described in Section 3.2 is available at `https://www.audiolabs-erlangen.de/fau/assistant/schreiber/data/schreiber_ismir2017-0.0.4.zip` and `http://www.tagtraum.com/download/schreiber_ismir2017-0.0.4.zip`.

The software requires Java 8 or later.

To start, execute one of these two programs:

```
$ java -cp schreiber_tempo_ismir2017-0.0.4-jar-with-dependencies.jar \
    -Djava.awt.headless=true \
    BPM [--model=MODEL_FILE|ismir2017|mirex2017] <input> [output]
$ java -cp schreiber_tempo_ismir2017-0.0.4-jar-with-dependencies.jar \
    -Djava.awt.headless=true \
    Mirex [--model=MODEL_FILE|ismir2017|mirex2017] <input> [output]
```

- BPM: Produces just the BPM
- Mirex: Output as required in the MIREX competition

You may optionally specify a model. By default the model for the original ISMIR paper is used. The MIREX model has been trained on a wider range of datasets than the ISMIR model and may produce better results.

The output is either written to the given file or, if omitted, to standard out. Supported audio input formats are `.au` and `.wav` with samples rate that are multiples of 11,025 Hz.

The code was implemented using the open source audio feature extraction framework *jipes* [153].

A.3. CNN-Based Systems

The GitHub repository `https://github.com/hendriks73/tempo-cnn` hosts several ready-to-use tempo estimation models from Chapters 4, 7, and 8 of this thesis. The Python library *librosa* [109] is used for feature extraction.

A.3.1. Installation

To install, please clone the repository using `git`, create a fresh `conda` environment, and then run the installation script:

```
$ git clone git@github.com:hendriks73/tempo-cnn.git
$ cd tempo-cnn
```

```
$ conda create -n tempocnn "python=3.6.*"
$ conda activate tempocnn
$ python setup.py install
```

After the installation succeeded, you may use the scripts `tempo` and `tempogram`. Both support the command line flag `--help` and provide detailed usage information.

A.3.2. Global Tempo Estimation

To estimate the tempo of a music audio file named `my_audio.wav`, please run:

```
$ tempo -i my_audio.wav -o my_audio.bpm
```

The result will be written to `my_audio.bpm`.

The most important `tempo` parameters are explained below. You may find additional options using `--help`.

A.3.2.1. Model Selection

There are multiple trained models to choose from, which can be specified with the parameter `-m`:

`ismir2018`	Model, set-up and trained as described in Chapter 4.
`cnn`	CNN model, trained for MIREX 2018 [157] on a large dataset, including some of the typical test sets.
`fcn`	Fully convolutional model (FCN), trained for MIREX 2018 [157] on a large dataset, including some of the typical test sets.
`shallowtemp_k{1,2,4,6,8,12}`	`ShallowTemp`-style model (Section 7.1.3.1) for selected values of the model scaling parameter k.
`deeptemp_k{2,4,8,16,24}`	`DeepTemp`-style model (Section 7.1.3.2) for selected values of the model scaling parameter k.
`deepsquare_k{1,2,4,8,16,24}`	`DeepSquare`-style model (Section 7.1.3.2) for selected values of the model scaling parameter k.
`dt_maz_m_fold[0-4]`	`DeepTemp`-style model (Section 7.1.3.2), trained on *one* of five folds of *Mazurka-5* using `M`-split (Figure 8.4a).
`dt_maz_v_fold[0-4]`	`DeepTemp`-style model (Section 7.1.3.2), trained on *one* of five folds of *Mazurka-5* using `V`-split (Figure 8.4b).

Note that we trained `shallowtemp_kx`, `deeptemp_kx`, `deepsquare_kx` multiple times (five runs each, i.e., five different models each), but the `tempo-cnn` GitHub repository only contains the models from the first run for each setup. The code used for training these models can be found at `https://github.com/hendriks73/directional_cnns`.

A.3.2.2. Output Format

Besides the default (just a BPM value), results may be presented in different formats using one of the following flags:

`--mirex` Format for MIREX

`--jams` JSON-based data format JAMS [77]

A.3.2.3. Interpolation

Since the networks supported by `tempo` perform classification, the output has a limited resolution. Final tempi are usually picked by selecting the class with the highest value of a softmax-distribution. Since the discrete distributions can be (quadratically) interpolated, we can also estimate sub-class tempo-values. You can request this behavior by setting the flag `--interpolate`.

A.3.3. Local Tempo Estimation and Visualization

To visualize the local tempo of a music audio file named `my_audio.wav`, please run:

```
$ tempogram my_audio.wav
```

The most important `tempogram` parameters are explained below. You may find additional options using `--help`.

A.3.3.1. Model Selection

You may choose the same models as for global tempo estimation (Section A.3.2.1) using the `-m` parameter.

A.3.3.2. CSV Export

In order to export local estimates as machine-readable data, you may set the `--csv` flag. Results are written to a file named like the input audio file with the extension `.csv` appended.

A.3.3.3. Time Resolution

By default, local tempo is determined every 1.486 s (0.67 Hz, hop-length 32). You may change this by specifying a different hop-length as multiples of 0.0464399093 s.

For example, `--hop-length 1` is equivalent to a hop length of 0.0464399093 s, i.e., a time resolution of 21.5 Hz.

A.3.3.4. Post Processing

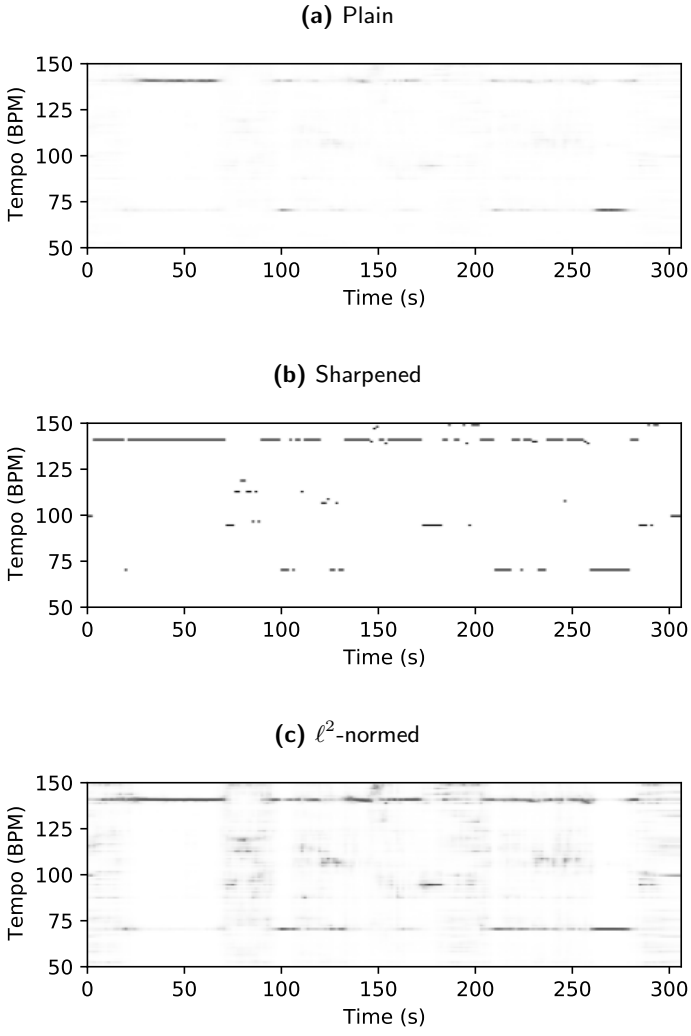


Figure A.1.: *Typhoon* by Foreign Beggars/Chasing Shadows without and with different kinds of post-processing using the `fcn` model and standard hop-length. **(a)** Plain visualization **(b)** Sharpened **(c)** ℓ^2-normed.

You may choose to have `tempogram` post-process the computed softmax distributions before visualization. For examples, please see Figure A.1.

`--sharpen`	Sharpen the image by only showing argmax values (one-hot encoding, Figure A.1b).
`--norm-frame NORM_FRAME`	Enable framewise normalization using max, ℓ^1, or ℓ^2 norm (Figure A.1c)).

B. Implementations: Key Estimation Systems

This appendix gives an overview of implementations for the key estimation systems we proposed in this thesis.

B.1. CNN-Based Systems

The GitHub repository `https://github.com/hendriks73/key-cnn` hosts several ready-to-use key estimation models from Chapter 7 of this thesis. The Python library *librosa* [109] is used for feature extraction.

B.1.1. Installation

To install, please clone the repository using `git`, create a fresh `conda` environment, and then run the installation script:

```
$ git clone git@github.com:hendriks73/key-cnn.git
$ cd key-cnn
$ conda create -n keycnn "python=3.6.*"
$ conda activate keycnn
$ python setup.py install
```

After the installation succeeded, you may use the scripts `key` and `keygram`. Both support the command line flag `--help` and provide detailed usage information.

B.1.2. Global Key Estimation

To estimate the key of a music audio file named `my_audio.wav`, please run:

```
$ key -i my_audio.wav -o my_audio.key
```

The result will be written to `my_audio.key`.

The most important `key` parameters are explained below. You may find additional options using `--help`.

B.1.2.1. Model Selection

There are multiple trained models to choose from, which can be specified with the parameter `-m`:

`shallowspec_k{1,2,4,6,8,12}` `ShallowSpec`-style model (Section 7.1.3.1) for selected values of the model scaling parameter k.

`deepspec_k{2,4,8,16,24}` `DeepSpec`-style model (Section 7.1.3.2) for selected values of the model scaling parameter k.

`deepsquare_k{1,2,4,8,16,24}` `DeepSquare`-style model (Section 7.1.3.2) for selected values of the model scaling parameter k.

Note that we trained `shallowspec_kx`, `deepspec_kx`, `deepsquare_kx` multiple times (five runs each, i.e., five different models each), but the `key-cnn` GitHub repository only contains the models from the first run for each setup. The code used for training these models can be found at `https://github.com/hendriks73/directional_cnns`.

B.1.2.2. Output Format

Besides the default (just a key value), results may be presented in different formats using one of the following flags:

`--mirex` Format for MIREX

`--jams` JSON-based data format JAMS [77]

B.1.3. Local Key Estimation and Visualization

To visualize the local tempo of a music audio file named `my_audio.wav`, please run:

```
$ keygram my_audio.wav
```

The most important `keygram` parameters are explained below. You may find additional options using `--help`.

B.1.3.1. Model Selection

You may choose the same models as for global tempo estimation (Section B.1.2.1) using the `-m` parameter.

B.1.3.2. CSV Export

In order to export local estimates as machine-readable data, you may set the `--csv` flag. Results are written to a file named like the input audio file with the extension `.csv` appended.

B.1.3.3. Time Resolution

By default, local tempo is determined every 1.4861 s (0.6729 Hz, hop-length 8). You may change this by specifying a different hop-length as multiples of 0.1857 s.

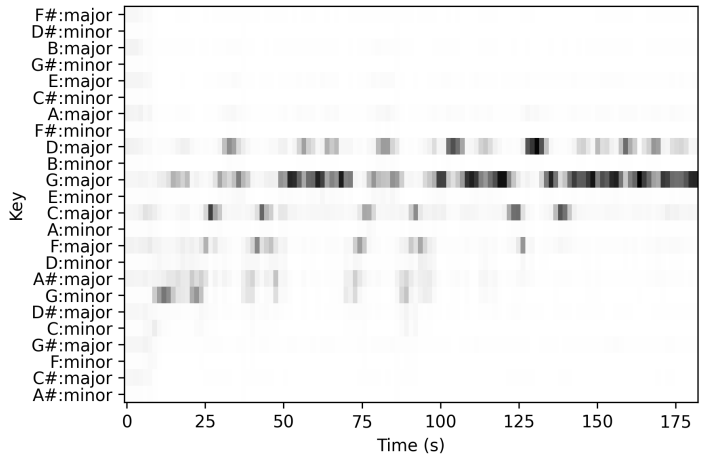
For example, `--hop-length 1` is equivalent to a hop length of 0.1857 s, i.e., a time resolution of 5.3833 Hz.

B.1.3.4. Post Processing

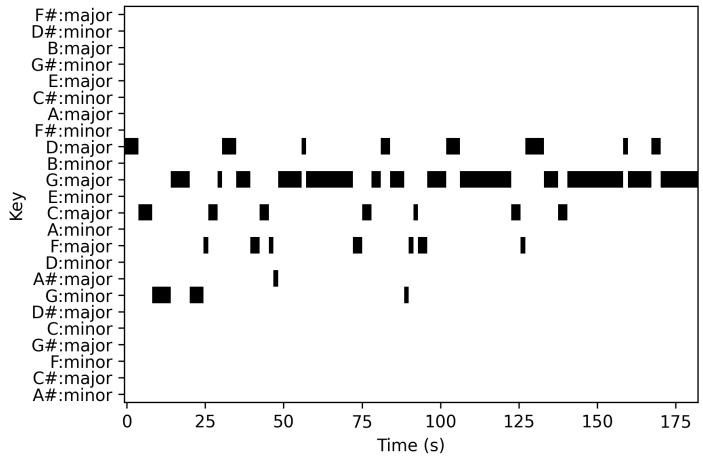
You may choose to have `tempogram` post-process the computed softmax distributions before visualization. For examples, please see Figure B.1.

`--sharpen`	Sharpen the image by only showing argmax values (one-hot encoding, Figure B.1b).
`--norm-frame NORM_FRAME`	Enable framewise normalization using max, ℓ^1, or ℓ^2 norm (Figure B.1c)).

(a) Plain



(b) Sharpened



(c) ℓ^2-normed

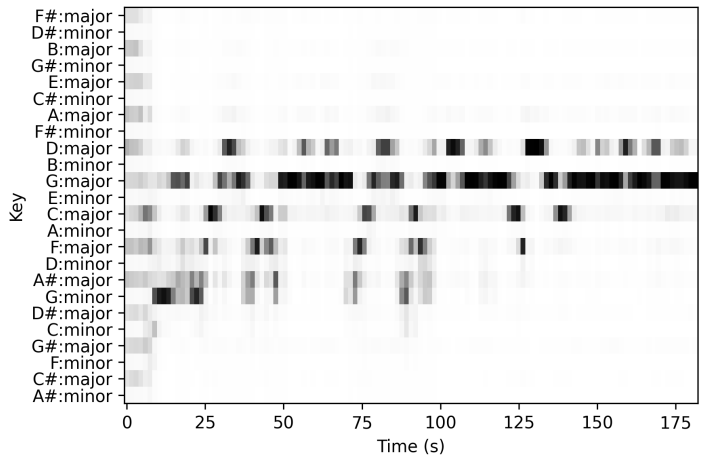


Figure B.1.: *Honky Tonk Women* by The Rolling Stones without and with different kinds of post-processing using the `deepspec` model and standard hop-length. **(a)** Plain visualization **(b)** Sharpened **(c)** ℓ^2-normed.

C. Schubert Winterreise: Measure Estimation Errors

In order to better understand what kind of errors the two local key estimation (LKE) systems from Chapter 9 make, we conducted an additional measurewise evaluation. Particularly, we were interested in musically motivated errors that do not depend on a particular performance or version. To this end, we determined whether estimates by one of our systems for a specific measure of a specific version were mostly correct or mostly wrong. In case the estimates were mostly wrong, we counted which *kind* of error occurred most often in the measure. We then added these *dominant* errors for each measure and version and visualized them in a stacked bar diagram. Both systems were trained on the "neither" split `N` described in Section 9.1.2

Just as an example, we can clearly see that, given our ground truth, both models tend to make relative key errors in song 23 in measures 10 to 12. Or that in song 19, both models make fifth errors from measure 25 to 30.

Further discussion of measure-specific errors is beyond the scope of this work, but certainly interesting in the context of annotator subjectivity.

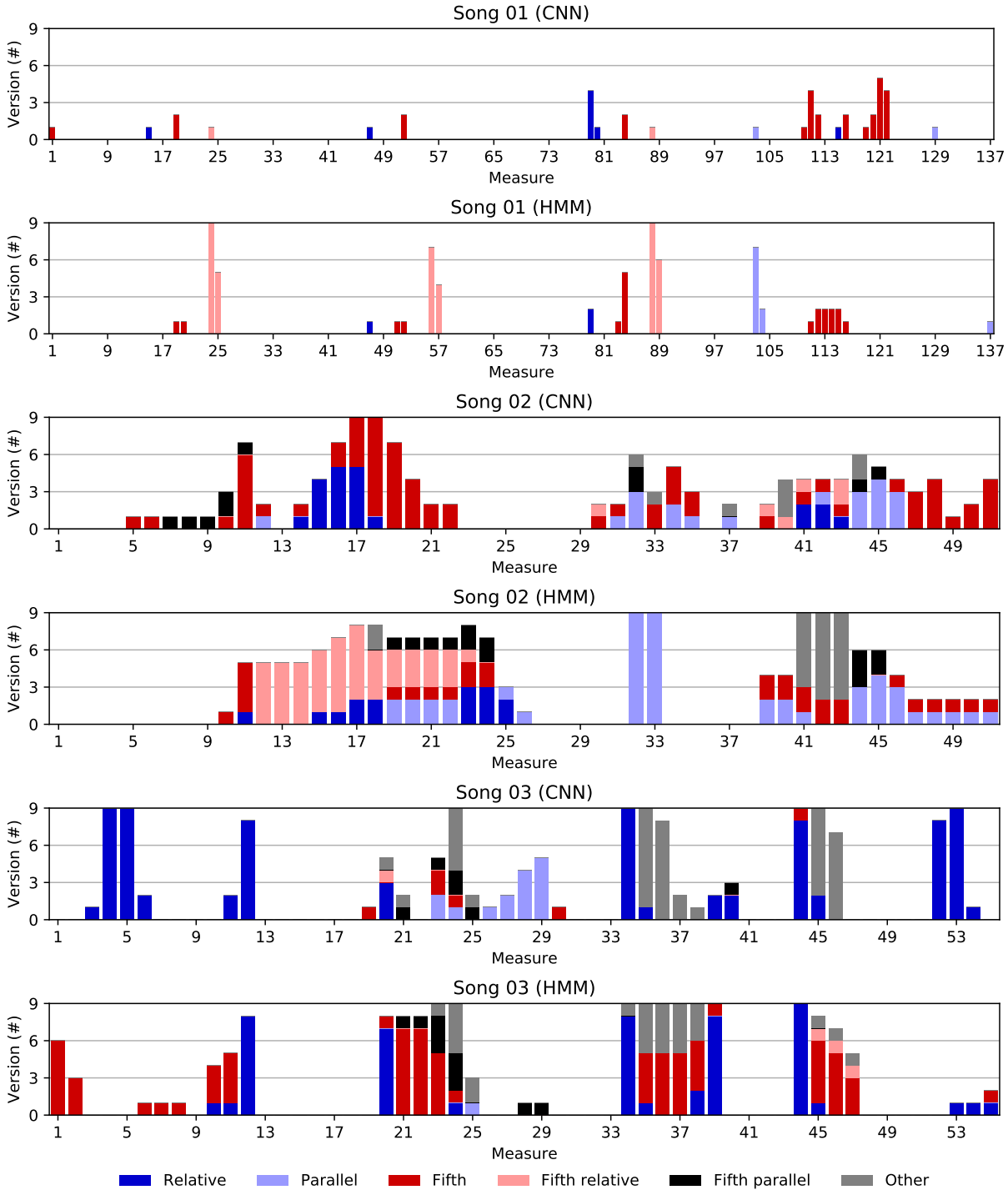


Figure C.1.: Dominant measure errors for songs 1 to 3.

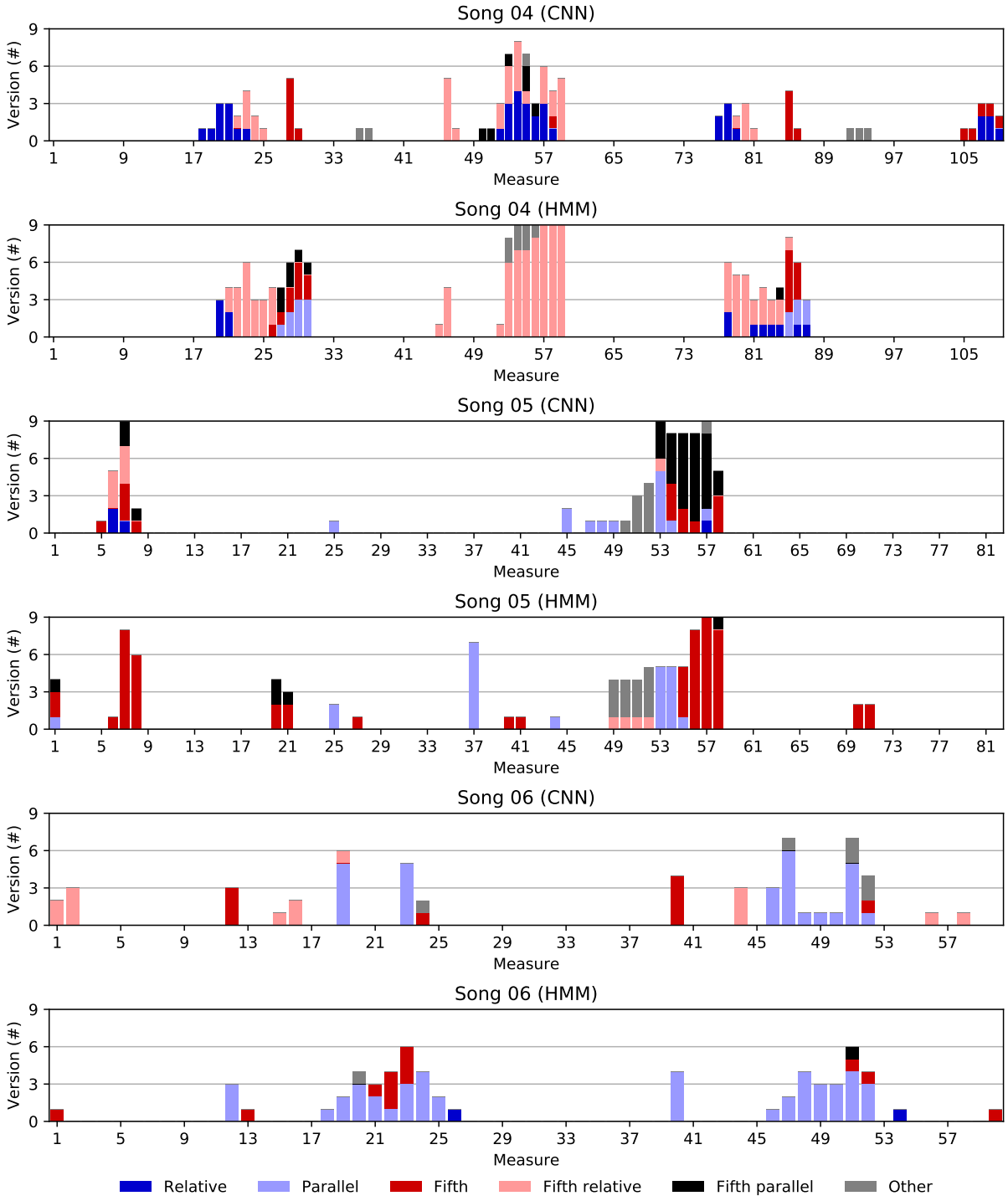


Figure C.2.: Dominant measure errors for songs 4 to 6.

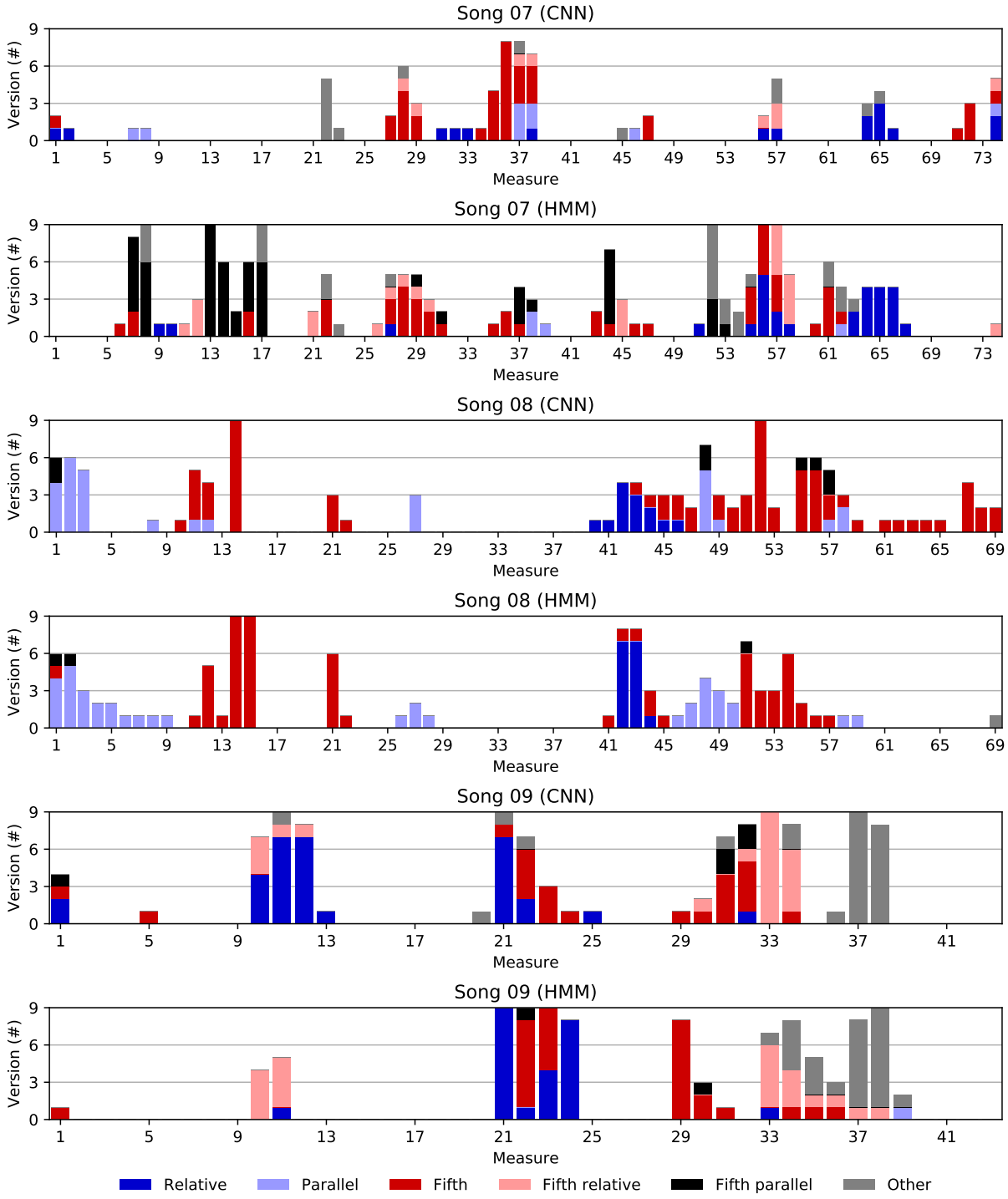


Figure C.3.: Dominant measure errors for songs 7 to 9.

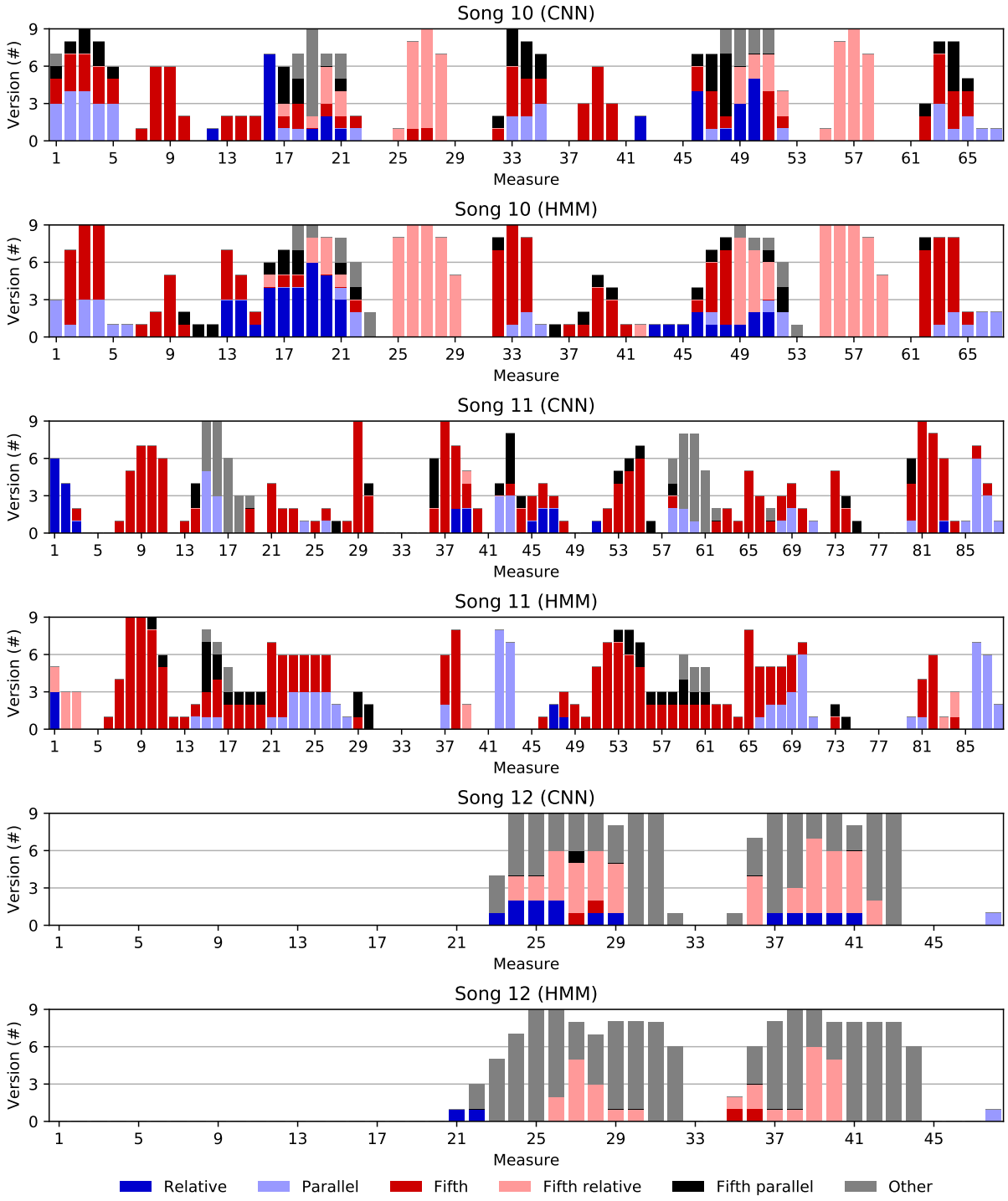


Figure C.4.: Dominant measure errors for songs 10 to 12.

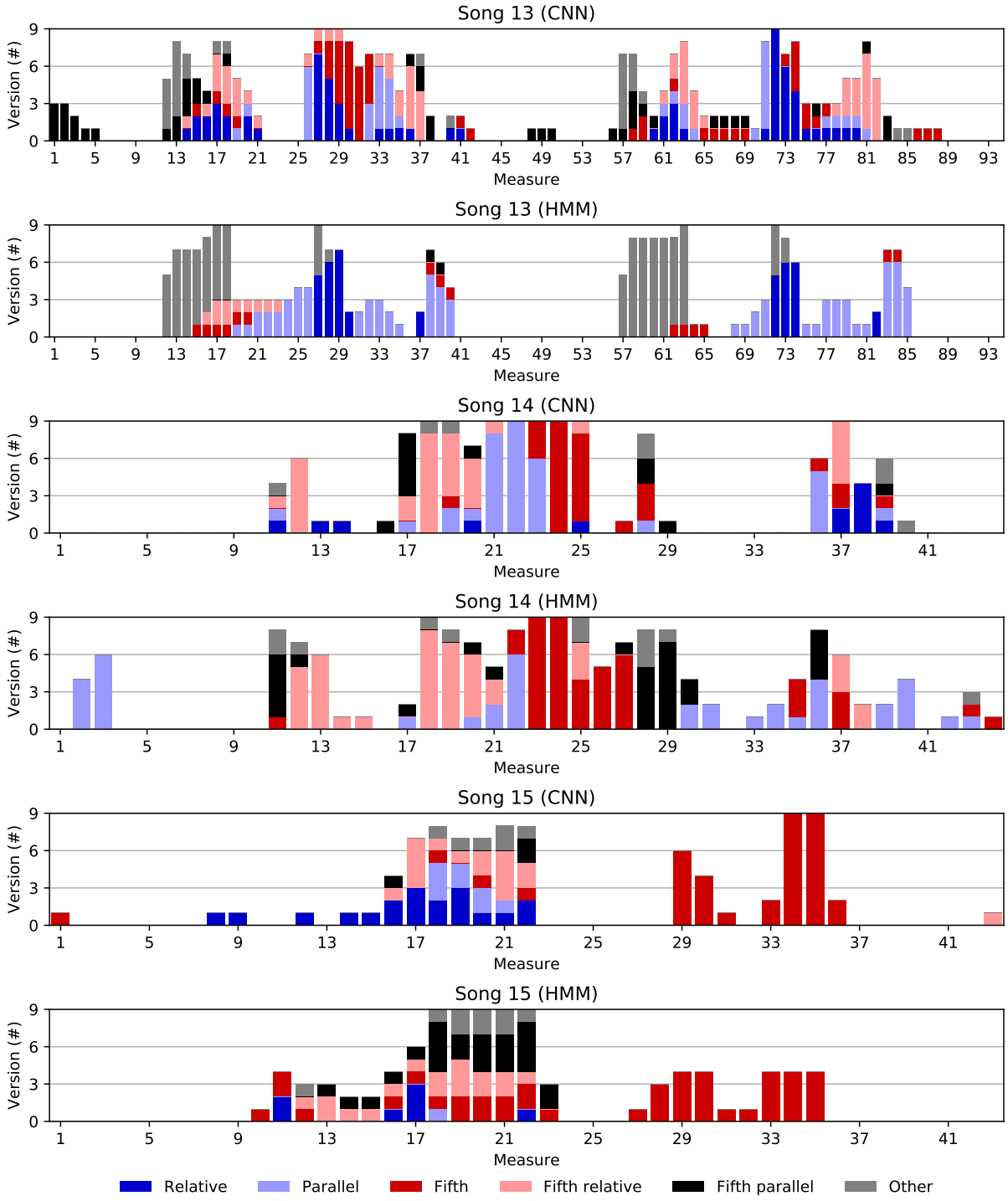


Figure C.5.: Dominant measure errors for songs 13 to 15.

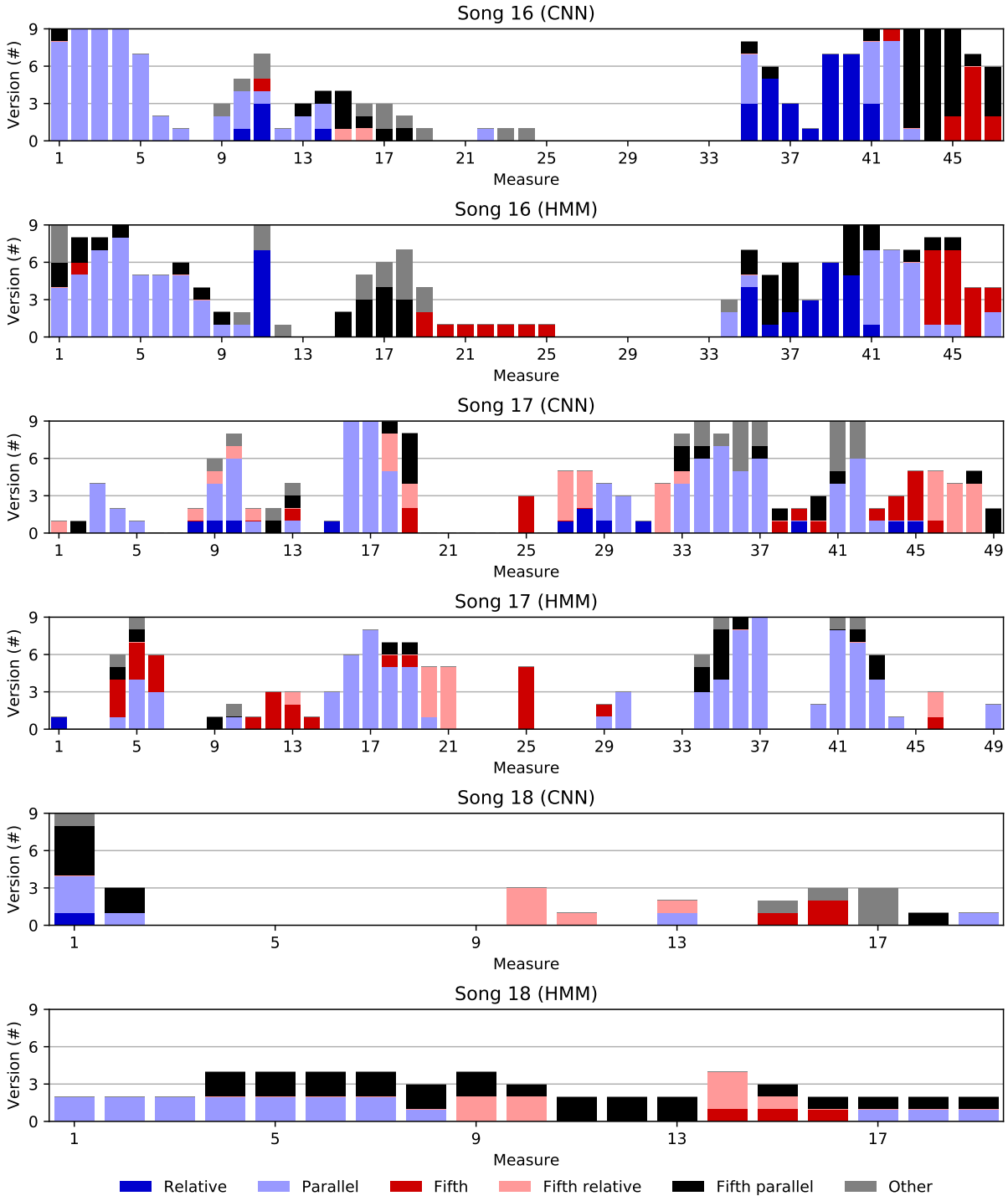


Figure C.6.: Dominant measure errors for songs 16 to 18.

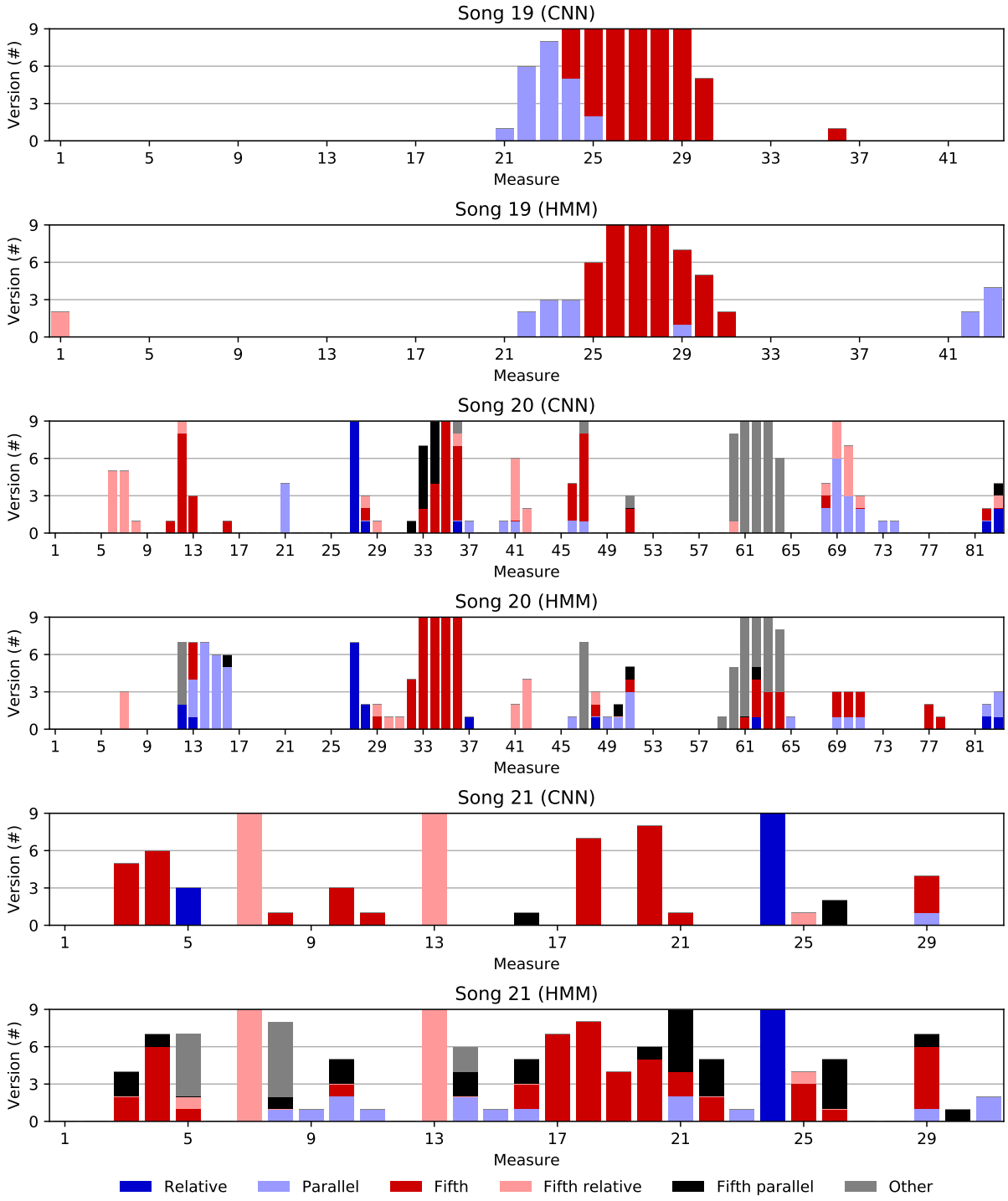


Figure C.7.: Dominant measure errors for estimates for songs 19 to 21.

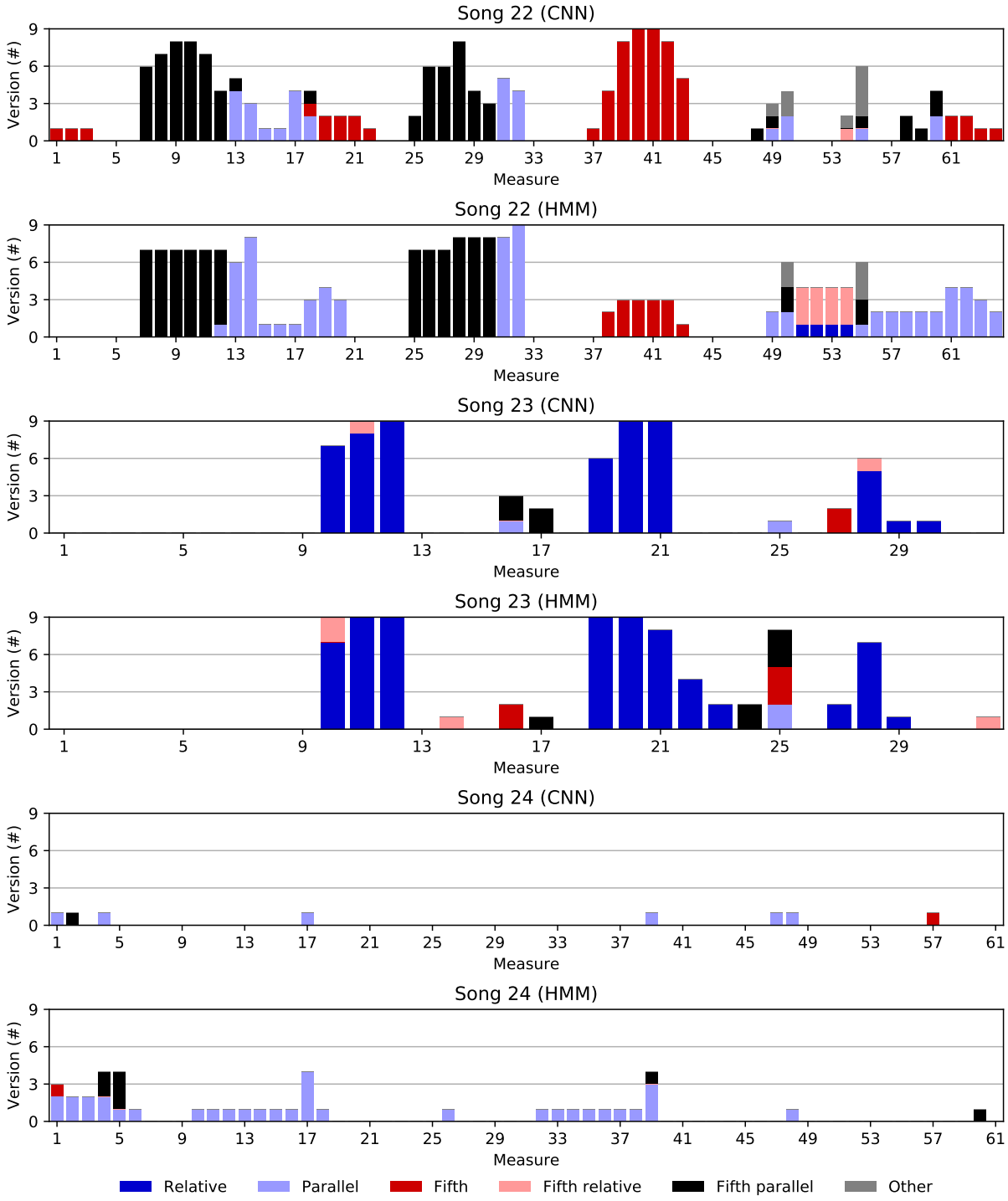
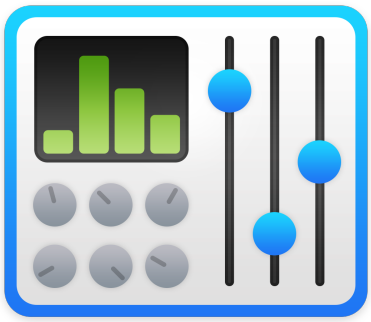


Figure C.8.: Dominant measure errors for estimates for songs 22 to 24.

D. beaTunes



beaTunes[1] ['biːtuːnz] is a commercial desktop software (macOS/Windows) for personal music library management developed by the author of this thesis. It is one of the main motivations for the work presented. The core idea of beaTunes is to support users in richly annotating their music in order to take advantage of this metadata when creating playlists, searching for or playing music.

beaTunes attempts to do this in multiple ways:

Inspection	Inconsistency search and repair
Analysis	Batch-processing of content-analysis tasks
Matchlists	User-configurable song-similarity functions
Semantic Navigation	Segmentation-aware music playback

Tempo and key estimation, as described in this work, are important pillars of beaTunes' automatic content analysis, as they are necessary inputs for building playlists with a certain tempo and harmonic progression.

In the following sections we will highlight some of the application's features.

[1] `https://www.beatunes.com/`

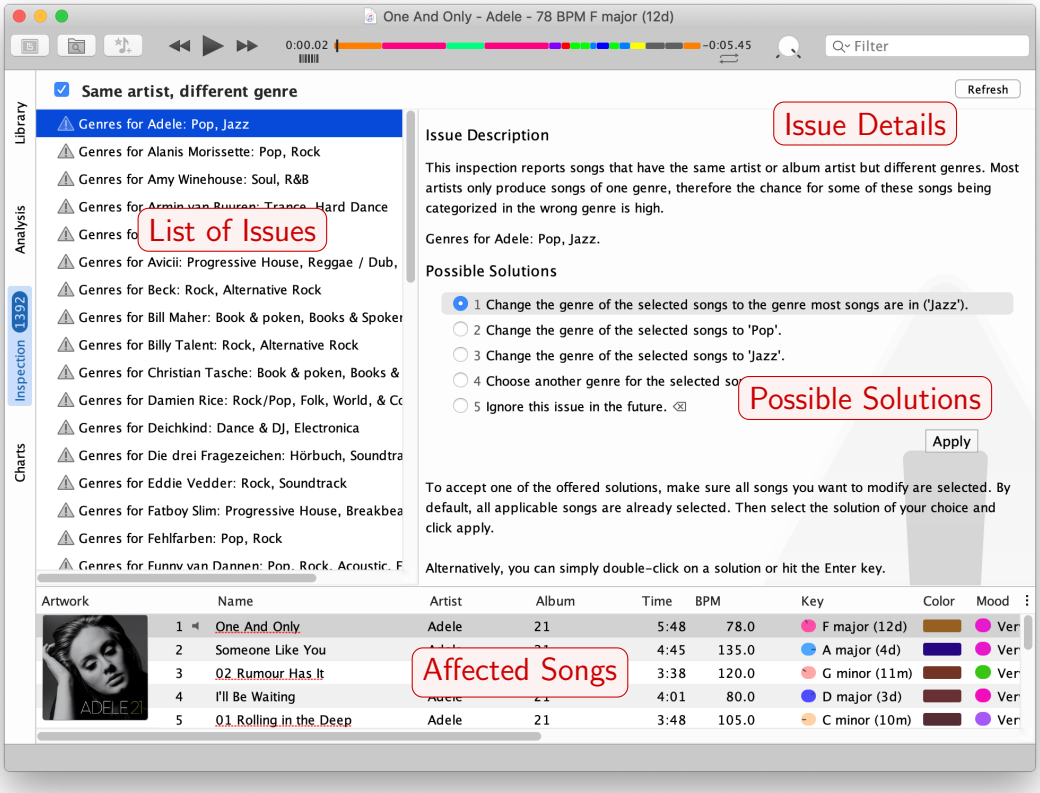


Figure D.1.: beaTunes inspection dialog. The left panel shows a list of potential issues found by one of beaTunes' inspectors. The right panel offers a detailed description of the selected issue along with potential fixes. The affected songs are shown in the bottom panel.

D.1. Inspection

Inspection is a process in which beaTunes attempts to find textual inconsistencies or errors in your music's metadata. For example, different spellings of the band name R.E.M.: REM, rem, Rem, etc. Once inconsistencies have been identified, beaTunes offers reasonable solutions like "change all artist names to the one used most often." Inspection is interactive, which means that the user is completely in control. Figure D.1 shows a screenshot of an inspection dialog. This feature is used most by collectors and audiophiles who aspire to maintain a well-curated collection.

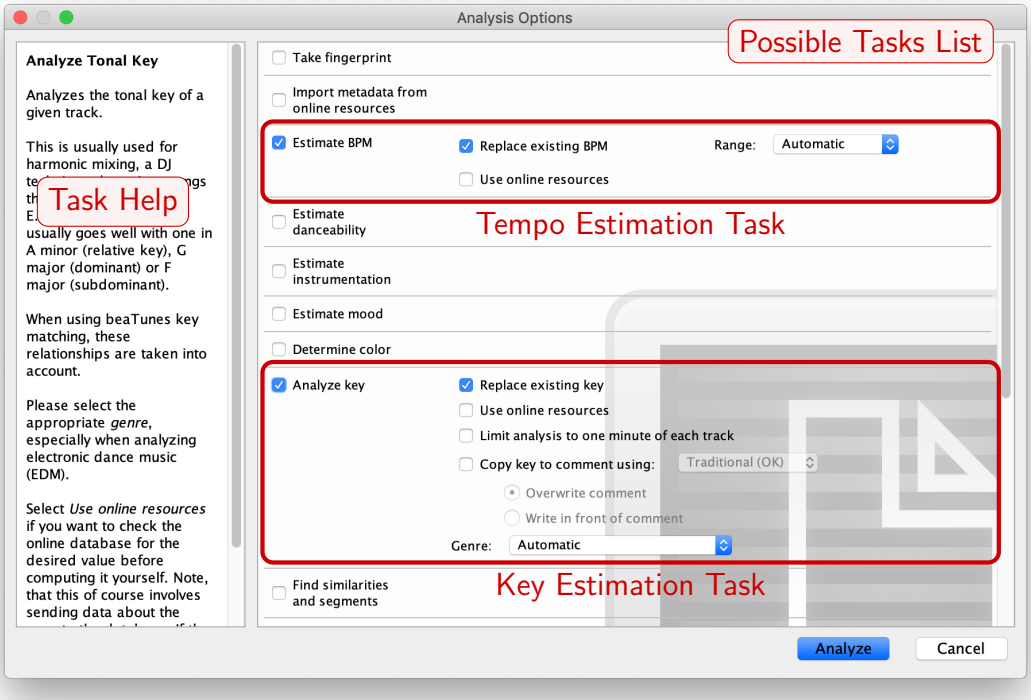


Figure D.2.: beaTunes analysis options dialog. After selecting the songs to be analyzed, this dialog lets the user choose the analysis tasks, e.g., tempo, key, or mood estimation.

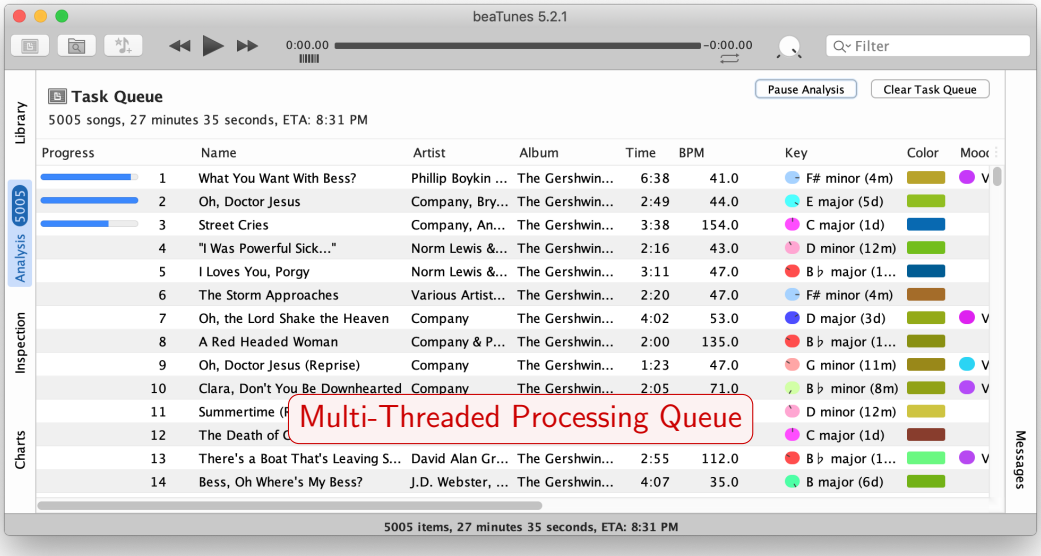


Figure D.3.: beaTunes's batch processing is based on a persistent queue that is polled by multiple worker threads.

D.2. Analysis

The *Analysis* process supports multi-threaded batch-processing (Figure D.3) for certain analysis tasks. Among them are automatic tempo and key estimation, automatic segmentation, loudness normalization, mood estimation, metadata lookup via audio fingerprinting, etc. (Figure D.2). While some of these tasks are based on content analysis, others are simple web-lookups, e.g., using AcousticBrainz[2] [133], or query the central beaTunes database. This database contains anonymized metadata of beaTunes user's collections [154, 155, 12] (if they chose to interact with the database). In contrast to Inspection (Section D.1), Analysis only lets the user select which songs to process and what tasks to execute. It is not interactive and therefore does not allow the user to make any decision once processing started.

[2] `https://acousticbrainz.org/`

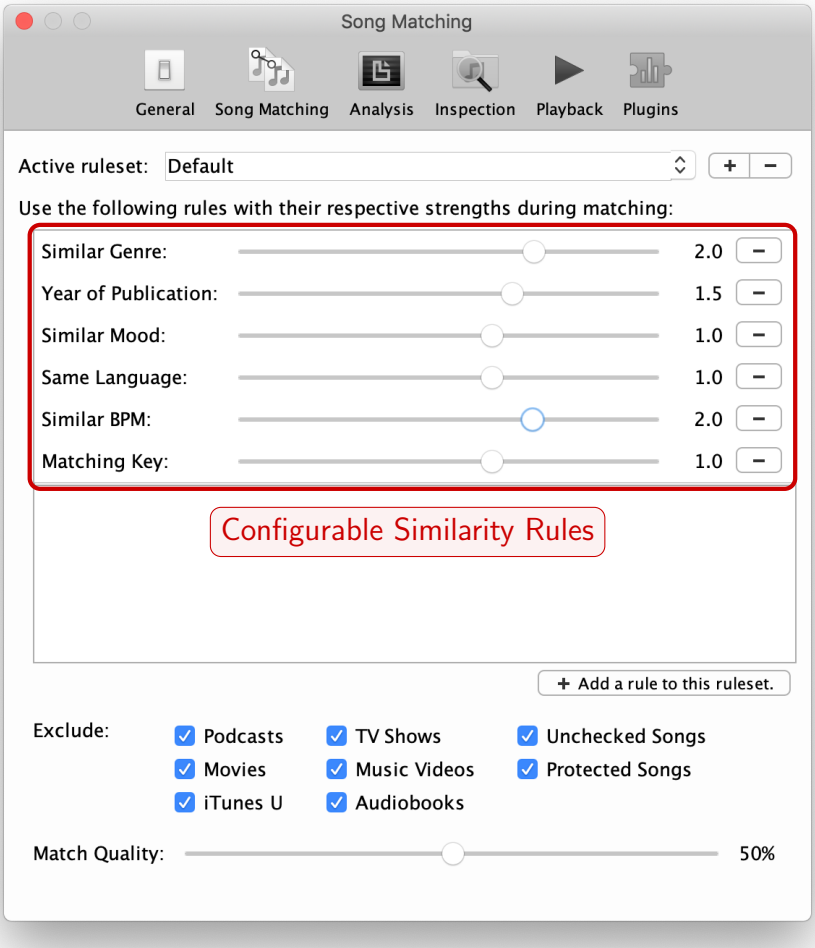


Figure D.4.: beaTunes similarity rulesets allow the user to create configurable similarity functions with emphasis on particular aspects. This allows users to create playlists around different concepts, like tempo, release year, popularity, or language.

D.3. Matchlists

The *Matchlist* feature allows users of beaTunes to create custom similarity rules (Figure D.4), emphasizing certain aspects of similarity. These rules can then be used for automatic playlist building based on one of multiple seed songs (Figure D.5). Additionally, beaTunes supports iterative playlist building by displaying matching songs in a separate panel (Figure D.6).

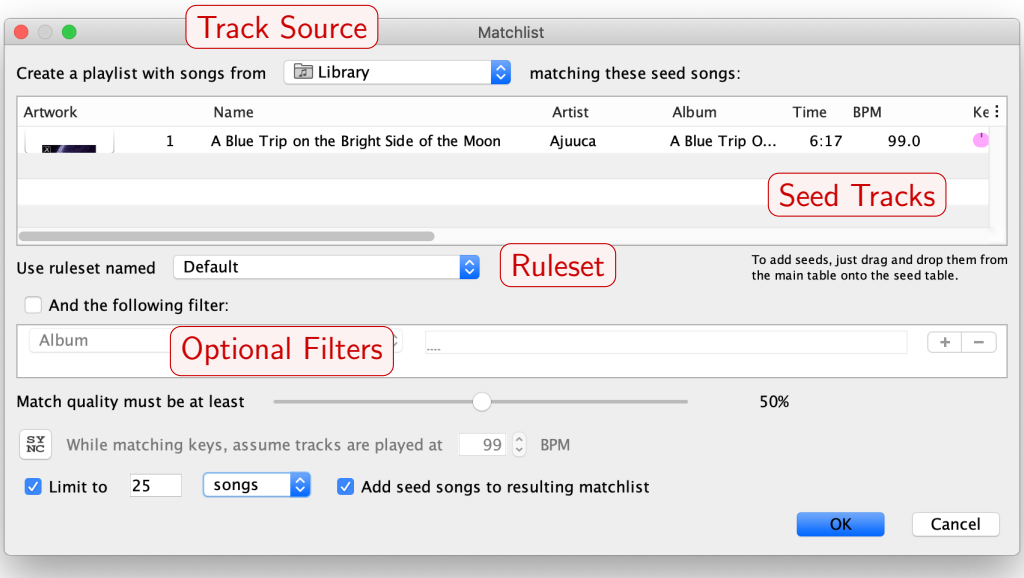


Figure D.5.: beaTunes Matchlists enables users to create new playlists based on seed songs, a similarity ruleset (Figure D.4), and optional filters.

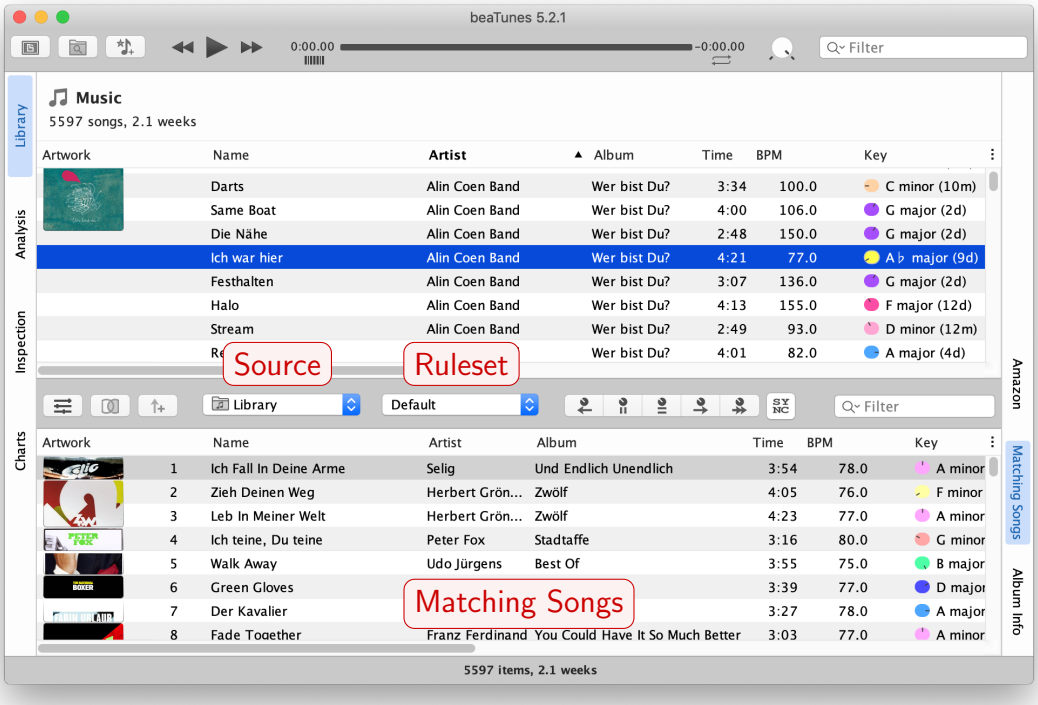


Figure D.6.: The beaTunes Matching Songs panel (bottom) shows songs that are similar to the selected song in the main panel (top). How similarity is determined depends on the active similarity ruleset (Figure D.4).

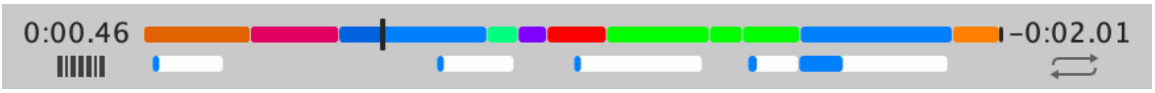


Figure D.7.: beaTunes semantic playback navigation. (top line) bars with the same color are of one kind, e.g., the blue chorus. (bottom line) blue/white bars visualize segments elsewhere in the song that are very similar to the current playhead position and indicate the corresponding playhead position in the similar segment. Clicking on such a bar takes the user to such a similar sounding position.

D.4. Semantic Navigation

Semantic Navigation is an innovative playback feature [59, 95] that allows users of beaTunes to navigate within a song from verse to verse or segment to segment (Figure D.7). It exploits the similarities and segments analysis task (Section D.2), which performs an automatic segmentation. This effectively allows users to jump from one instance of the chorus to the next, visually identify the end of the intro, etc. beaTunes exploits these annotation also in its *scan* feature, which allows listening through a playlist by only listening to prominent parts (e.g, choruses). How well this works, obviously depends highly on the quality of the automatic segmentation.

Bibliography

[1] Frans G.J. Absil. *Musical Analysis – Visiting the Great Composers*. Frans Absil Music, 6th edition, 2017.

[2] Paul E. Allen and Roger B. Dannenberg. Tracking musical beats in real time. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 140–143, Glasgow, Scotland, 1990.

[3] Miguel Alonso, Bertrand David, and Gaël Richard. A study of tempo tracking algorithms from polyphonic music signals. In *Proceedings of the COST 276 Workshop*, Bordeaux, France, 2003.

[4] Miguel Alonso, Bertrand David, and Gaël Richard. Tempo and beat estimation of musical signals. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Barcelona, Spain, 2004.

[5] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin Raffel. MixMatch: A holistic approach to semi-supervised learning. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, pages 5050–5060, Vancouver, Canada, 2019.

[6] Sebastian Böck, Florian Krebs, and Gerhard Widmer. Accurate tempo estimation based on recurrent neural networks and resonating comb filters. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 625–631, Málaga, Spain, 2015.

[7] Sebastian Böck, Florian Krebs, and Gerhard Widmer. Joint beat and downbeat tracking with recurrent neural networks. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 255–261, New York City, USA, 2016.

[8] Sebastian Böck, Matthew E. P. Davies, and Peter Knees. Multi-task learning of tempo and beat: Learning one to improve the other. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 486–493, Delft, The Netherlands, 2019.

[9] David Bodoff. Test theory for evaluating reliability of IR test collections. *Information Processing & Management*, 44(3):1117–1145, 2008.

[10] Dmitry Bogdanov, Joan Serra, Nicolas Wack, and Perfecto Herrera. Hybrid similarity measures for music recommendation. In *Music Information Retrieval Evaluation eXchange (MIREX)*, Kobe, Japan, 2009.

[11] Dmitry Bogdanov, Joan Serra, Nicolas Wack, and Perfecto Herrera. Hybrid music similarity measure. In *Music Information Retrieval Evaluation eXchange (MIREX)*, Utrecht, The Netherlands, 2010.

[12] Dmitry Bogdanov, Alastair Porter, Hendrik Schreiber, Julián Urbano, and Sergio Oramas. The AcousticBrainz genre dataset: Multi-source, multi-level, multi-label, and large-scale. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 360–367, Delft, The Netherlands, 2019.

[13] Juan J. Bosch, Ricard Marxer, and Emilia Gómez. Evaluation and combination of pitch estimation methods for melody extraction in symphonic classical music. *Journal of New Music Research*, 45(2): 101–117, 2016.

[14] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[15] Mark J. Butler. *Unlocking the Groove: Rhythm, Meter, and Musical Design in Electronic Dance Music*. Profiles in popular music. Indiana University Press, 2006.

[16] Chris Cannam, Christian Landone, Mark B. Sandler, and Juan Pablo Bello. The Sonic Visualiser: A visualisation platform for semantic descriptors from musical signals. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 324–327, Victoria, Canada, 2006.

[17] Ben Carterette, Virgil Pavlu, Evangelos Kanoulas, Javed A. Aslam, and JamesAllan. If I had a million queries. In *Proceedings of the European Conference on Information Retrieval (ECIR)*, pages 288–300, Toulouse, France, 2009.

[18] Wei Chai and Barry Vercoe. Detection of key change in classical piano music. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 468–474, London, UK, 2005.

[19] Ching-Wei Chen, Markus Cremer, Kyogu Lee, Peter DiMaria, and Ho-Hsiang Wu. Improving perceived tempo estimation by statistical modeling of higher-level musical descriptors. In *Proceedings of the Audio Engineering Society Convention (AES)*, Munich, Germany, 2009.

[20] Ching-Wei Chen, Kyogu Lee, and Ho-Hsiang Wu. Towards a class-based representation of perceptual tempo for music retrieval. In *Proceedings of the IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 602–607, Miami, Florida, USA, 2009.

[21] Taemin Cho and Juan Pablo Bello. On the relative importance of individual components of chord recognition systems. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(2): 477–492, 2014.

[22] Keunwoo Choi, György Fazekas, and Mark B. Sandler. Automatic tagging using deep convolutional neural networks. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 805–811, New York City, USA, 2016.

[23] Shreyan Chowdhury, Andreu Vall, Verena Haunschmid, and Gerhard Widmer. Towards explainable music emotion recognition: The route via mid-level features. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 237–243, Delft, The Netherlands, 2019.

[24] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUS). In *International Conference on Learning Representations (ICLR)*, San Juan, Puerto Rico, 2015.

[25] Olmo Cornelis, Joren Six, Andre Holzapfel, and Marc Leman. Evaluation and recommendation of pulse and tempo annotation in ethnic music. *Journal of New Music Research*, 42(2):131–149, 2013.

[26] d-lusion interactive media. d-lusion MJ Studio. `http://www.d-lusion.com/ProductsMJStudio.html`, last accessed 11/13/2019, 1998.

[27] Carl Dahlhaus, Julian Anderson, Charles Wilson, Richard L. Cohn, and Brian Hyer. Harmony. In *Grove Music Online: Oxford Music Online*. Oxford University Press, 2001.

[28] Matthew E. P. Davies and Mark D. Plumbley. Context-dependent beat tracking of musical audio. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(3):1009–1020, 2007.

[29] Matthew E.P. Davies, Mark D. Plumbley, and Douglas Eck. Towards a musical beat emphasis function. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 61–64, New Paltz, NY, USA, 2009.

[30] Sander Dieleman and Benjamin Schrauwen. End-to-end learning for music audio. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6964–6968, Florence, Italy, 2014.

[31] Sander Dieleman, Philemon Brakel, and Benjamin Schrauwen. Audio-based music classification with a pretrained convolutional network. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 669–674, Miami, Florida, 2011.

[32] Christian Dittmar, Bernhard Lehner, Thomas Prätzlich, Meinard Müller, and Gerhard Widmer. Cross-version singing voice detection in classical opera recordings. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 618–624, Málaga, Spain, October 2015.

[33] Simon Dixon. Automatic extraction of tempo and beat from expressive performances. *Journal of New Music Research*, 30:39–58, 2001.

[34] Matthias Dorfer and Gerhard Widmer. Training general-purpose audio tagging networks with noisy labels and iterative self-verification. Technical report, DCASE2018 Challenge, 2018.

[35] J. Stephen Downie. Music information retrieval. *Annual Review of Information Science and Technology (Chapter 7)*, 37:295–340, 2003.

[36] J. Stephen Downie. The music information retrieval evaluation exchange (2005–2007): A window into music information retrieval research. *Acoustical Science and Technology*, 29(4):247–255, 2008.

[37] Jonathan Driedger, Hendrik Schreiber, Bas de Haas, and Meinard Müller. Towards automatically correcting tapped beat annotations for music recordings. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 200–207, Delft, The Netherlands, 2019.

[38] Simon Durand, Juan P. Bello, Bertrand David, and Gaël Richard. Robust downbeat tracking using an ensemble of convolutional networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(1):76–89, 2017.

[39] Edith Van Dyck, Bart Moens, Jeska Buhmann, Michiel Demey, Esther Coorevits, Simone Dalla Bella, and Marc Leman. Spontaneous entrainment of running cadence to music tempo. *Sports Medicine-Open*, 1(1):15, 2015.

[40] Daniel P.W. Ellis. Beat tracking by dynamic programming. *Journal of New Music Research*, 36(1): 51–60, 2007.

[41] Daniel P.W. Ellis and Graham E. Poliner. Identifying 'cover songs' with chroma features and dynamic programming beat tracking. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 4, pages 1429–1432, Honolulu, Hawaii, USA, 2007.

[42] Anders Elowsson. Beat tracking with a cepstroid invariant neural network. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 351–357, New York City, USA, 2016.

[43] Anders Elowsson and Anders Friberg. Modeling the perception of tempo. *The Journal of the Acoustical Society of America*, 137(6):3163–3177, 2015.

[44] Ángel Faraldo, Emilia Gómez, Sergi Jordà, and Perfecto Herrera. Key estimation in electronic dance music. In *Proceedings of the European Conference on Information Retrieval (ECIR)*, pages 335–347, Padua, Italy, 2016.

[45] Ángel Faraldo, Sergi Jordà, and Perfecto Herrera. A multi-profile method for key estimation in EDM. In *Proceedings of the AES International Conference on Semantic Audio*, pages 172–178, Erlangen, Germany, 2017.

[46] Frederic Font and Xavier Serra. Tempo estimation for music loops and a simple confidence measure. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 269–275, New York, NY, USA, 2016.

[47] Jonathan Foote. Automatic audio segmentation using a measure of audio novelty. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, pages 452–455, New York, NY, USA, 2000.

[48] Jonathan Foote and Shingo Uchihashi. The beat spectrum: A new approach to rhythm analysis. In *Proceedings of the International Conference on Multimedia and Expo (ICME)*, pages 881–884, Tokyo, Japan, August 2001.

[49] Hadrien Foroughmand and Geoffroy Peeters. Deep-rhythm for tempo estimation and rhythm pattern recognition. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 636–643, Delft, The Netherlands, 2019.

[50] Peter Foster, György Fazekas, Matthias Mauch, and Simon Dixon. MIREX submission: Sequential complexity as a descriptor for musical similarity. In *Music Information Retrieval Evaluation eXchange (MIREX)*, Taipei, Taiwan, 2014.

[51] Anders Friberg and Johan Sundberg. Time discrimination in a monotonic, isochronous sequence. *The Journal of the Acoustical Society of America*, 98(5):2524–2531, 1995.

[52] Dennis Gabor. Theory of communication. *Journal of the Institution of Electrical Engineers (IEE)*, 93(26):429–457, 1946.

[53] Daniel Gärtner. Tempo detection of urban music using tatum grid non negative matrix factorization. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 311–316, Curitiba, Brazil, 2013.

[54] William Henry Gates III. Information at your fingertips 2005. In *COMDEX keynote*, Las Vegas, NV, USA, November 1994.

[55] Aggelos Gkiokas and Vassilis Katsouros. Convolutional neural networks for real-time beat tracking: A dancing robot application. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 286–293, Suzhou, China, 2017.

[56] Aggelos Gkiokas, Vassilios Katsouros, and George Carayannis. Reducing tempo octave errors by periodicity vector coding and SVM learning. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 301–306, Porto, Portugal, 2012.

[57] Aggelos Gkiokas, Vassilios Katsouros, George Carayannis, and Themos Stafylakis. Music tempo estimation and beat tracking by applying source separation and metrical relations. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 421–424, Kyoto, Japan, 2012.

[58] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge and London, 2016. `http://www.deeplearningbook.org`.

[59] Masataka Goto. A chorus-section detecting method for musical audio signals. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 437–440, Hong Kong, China, 2003.

[60] Masataka Goto and Yoichi Muraoka. A beat tracking system for acoustic signals of music. In *Proceedings of the ACM International Conference on Multimedia*, pages 365–372, San Francisco, CA, USA, 1994.

[61] Masataka Goto, Hiroki Hashiguchi, Takuichi Nishimura, and Ryuichi Oka. RWC music database: Popular, classical and jazz music databases. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 287–288, Paris, France, 2002.

[62] Masataka Goto, Hiroki Hashiguchi, Takuichi Nishimura, and Ryuichi Oka. RWC music database: Music genre database and musical instrument sound database. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 229–230, Baltimore, MD, USA, 2003.

[63] Fabien Gouyon, Simon Dixon, Elias Pampalk, and Gerhard Widmer. Evaluating rhythmic descriptors for musical genre classification. In *Proceedings of the Audio Engineering Society (AES) International Conference*, London, UK, 2004.

[64] Fabien Gouyon, Anssi P. Klapuri, Simon Dixon, Miguel Alonso, George Tzanetakis, Christian Uhle, and Pedro Cano. An experimental comparison of audio tempo induction algorithms. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(5):1832–1844, 2006.

[65] Harald Grohganz. *Algorithmen zur strukturellen Analyse von Musikaufnahmen.* PhD thesis, University of Bonn, Germany, 2015.

[66] Peter Grosche and Meinard Müller. A mid-level representation for capturing dominant tempo and pulse information in music recordings. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 189–194, Kobe, Japan, October 2009.

[67] Peter Grosche and Meinard Müller. Extracting predominant local pulse information from music recordings. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(6):1688–1701, 2011.

[68] Peter Grosche, Meinard Müller, and Frank Kurth. Cyclic tempogram – a mid-level tempo representation for music signals. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 5522–5525, Dallas, Texas, USA, March 2010.

[69] Peter Grosche, Meinard Müller, and Craig Stuart Sapp. What makes beat tracking difficult? A case study on Chopin Mazurkas. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, pages 649–654, Utrecht, The Netherlands, 2010.

[70] Stephen Webley Hainsworth. *Techniques for the Automated Analysis of Musical Audio.* PhD thesis, University of Cambridge, UK, 2004.

[71] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter*, 11 (1):10–18, 2009.

[72] Jason Hockman and Ichiro Fujinaga. Fast vs slow: Learning tempo octaves from user data. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 231–236, Utrecht, The Netherlands, 2010.

[73] Andre Holzapfel and Thomas Grill. Bayesian meter tracking on learned signal representations. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 262–268, New York City, USA, 2016.

[74] André Holzapfel, Matthew E. P. Davies, José R. Zapata, João Lobato Oliveira, and Fabien Gouyon. Selective sampling for beat tracking evaluation. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(9):2539–2548, 2012.

[75] Florian Hörschläger, Richard Vogl, Sebastian Böck, and Peter Knees. Addressing tempo estimation octave errors in electronic music by incorporating style information extracted from Wikipedia. In *Proceedings of the Sound and Music Computing Conference (SMC)*, Maynooth, Ireland, 2015.

[76] Eric J. Humphrey, Juan Pablo Bello, and Yann LeCun. Moving beyond feature design: Deep architectures and automatic feature learning in music informatics. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 403–408, Porto, Portugal, 2012.

[77] Eric J. Humphrey, Justin Salamon, Oriol Nieto, Jon Forsyth, Rachel M. Bittner, and Juan Pablo Bello. JAMS: A JSON annotated music specification for reproducible MIR research. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 591–596, Taipei, Taiwan, October 2014.

[78] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 448–456, Lille, France, 2015.

[79] Özgür İzmirli. Localized key finding from audio using nonnegative matrix factorization for segmentation. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 195–200, Vienna, Austria, 2007.

[80] Nanzhu Jiang and Meinard Müller. Automated methods for analyzing music recordings in sonata form. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, pages 595–600, Curitiba, Brazil, 2013.

[81] Nanzhu Jiang, Peter Grosche, Verena Konz, and Meinard Müller. Analyzing chroma feature types for automated chord recognition. In *Proceedings of the AES Conference on Semantic Audio*, Ilmenau, Germany, 2011.

[82] Costas I. Karageorghis, Leighton Jones, David-Lee Priest, Rose I. Akers, Adam Clarke, Jennifer M. Perry, Benjamin T. Reddick, Daniel T. Bishop, and Harry B.T. Lim. Revisiting the relationship between exercise heart rate and music tempo preference. *Research Quarterly for Exercise and Sport*, 82(2):274–284, 2011.

[83] Eric Kemp. Low tech history. `http://id3.org/History`, last accessed 11/13/2019, 1996.

[84] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference for Learning Representations (ICLR)*, San Diego, California, USA, 2015.

[85] Anssi P. Klapuri. Sound onset detection by applying psychoacoustic knowledge. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 3089–3092, Washington, DC, USA, 1999.

[86] Anssi P. Klapuri, Antti J. Eronen, and Jaakko Astola. Analysis of the meter of acoustic musical signals. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(1):342–355, 2006.

[87] Peter Knees and Markus Schedl. *Music Similarity and Retrieval*. Springer Verlag, 2016. ISBN 978-3-662-49720-3.

[88] Peter Knees, Ángel Faraldo, Perfecto Herrera, Richard Vogl, Sebastian Böck, Florian Hörschläger, and Mickael Le Goff. Two data sets for tempo estimation and key detection in electronic dance music

annotated from user corrections. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 364–370, Málaga, Spain, 2015.

[89] Verena Konz, Meinard Müller, and Rainer Kleinertz. A cross-version chord labelling approach for exploring harmonic structures—a case study on Beethoven's Appassionata. *Journal of New Music Research*, 42(1):61–77, 2013.

[90] Filip Korzeniowski and Gerhard Widmer. End-to-end musical key estimation using a convolutional neural network. In *Proceedings of the European Signal Processing Conference (EUSIPCO)*, Kos Island, Greece, 2017.

[91] Filip Korzeniowski and Gerhard Widmer. Genre-agnostic key classification with convolutional neural networks. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 264–270, Paris, France, 2018.

[92] Sebastian Kraft, Alexander Lerch, and Udo Zölzer. The tonalness spectrum: feature-based estimation of tonal components. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, pages 2–5, Maynooth, Ireland, 2013.

[93] Florian Krebs, Sebastian Böck, and Gerhard Widmer. Rhythmic pattern modeling for beat and downbeat tracking in musical audio. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 227–232, Curitiba, Brazil, 2013.

[94] Carol L. Krumhansl. *Cognitive Foundations of Musical Pitch*. Oxford University Press, Oxford, UK, 1990.

[95] Frank Kurth, Meinard Müller, David Damm, Christian Fremerey, Andreas Ribbrock, and Michael Clausen. SyncPlayer - an advanced system for multimodal music access. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, pages 381–388, London, UK, November 2005.

[96] Paul Lamere. In search of the click track. Blog post `https://musicmachinery.com/2009/03/02/in-search-of-the-click-track/`, last accessed 12/9/2018, 2009.

[97] Olivier Lartillot and Petri Toiviainen. MIR in MATLAB (II): A toolbox for musical feature extraction from audio. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 127–130, Vienna, Austria, 2007.

[98] Jin Ha Lee and Nichole Maiman Waterman. Understanding user requirements for music information services. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 253–258, Porto, Portugal, 2012.

[99] Jongpil Lee, Jiyoung Park, Keunhyoung Luke Kim, and Juhan Nam. Sample-level deep convolutional neural networks for music auto-tagging using raw waveforms. In *Proceedings of the Sound and Music Computing Conference (SMC)*, pages 220–226, Espoo, Finland, 2017.

[100] Franz De Leon and Kirk Martinez. Submission to MIREX 2011 genre classification and audio similarity tasks. In *Music Information Retrieval Evaluation eXchange (MIREX)*, Miami, FL, USA, 2011.

[101] Franz De Leon and Kirk Martinez. Using timbre, rhythm and tempo models for audio music similarity estimation. In *Music Information Retrieval Evaluation eXchange (MIREX)*, Porto, Portugal, 2012.

[102] Franz De Leon and Kirk Martinez. Using timbre models for audio music similarity estimation. In *Music Information Retrieval Evaluation eXchange (MIREX)*, Curitiba, Brazil, 2013.

[103] Mark Levy. Improving perceptual tempo estimation with crowd-sourced annotations. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 317–322, Porto, Portugal, 2011.

[104] Ugo Marchand and Geoffroy Peeters. Swing ratio estimation. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, pages 423–428, Trondheim, Norway, 2015.

[105] Ugo Marchand and Geoffroy Peeters. The extended ballroom dataset. In *Late Breaking Demo of the International Conference on Music Information Retrieval (ISMIR)*, New York, NY, USA, 2016.

[106] Michaelangelo Matos. Electronic dance music. Encyclopædia Britannica `https://www.britannica.com/art/electronic-dance-music`, last checked 10/25/2019, 2015.

[107] Matthias Mauch and Simon Dixon. Simultaneous estimation of chords and musical context from audio. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(6):1280–1289, 2010.

[108] Brian McFee and Juan Pablo Bello. Structured training for large-vocabulary chord recognition. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 188–194, Suzhou, China, 2017.

[109] Brian McFee, Colin Raffel, Dawen Liang, Daniel P.W. Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. Librosa: Audio and music signal analysis in python. In *Proceedings the Python Science Conference*, pages 18–25, Austin, Texas, USA, 2015.

[110] Martin F. McKinney and Dirk Moelants. Deviations from the resonance theory of tempo induction. In *Proceedings of Conference on Interdisciplinary Musicology (CIM)*, Graz, Austria, 2004.

[111] Martin F. McKinney and Dirk Moelants. Ambiguity in tempo perception: What draws listeners to different metrical levels? *Music Perception: An Interdisciplinary Journal*, 24(2):155–166, 2006.

[112] Martin F. McKinney, Dirk Moelants, Matthew E. P. Davies, and Anssi P. Klapuri. Evaluation of audio beat tracking and music tempo extraction algorithms. *Journal of New Music Research*, 36(1): 1–16, 2007.

[113] Lesley Mearns, Emmanouil Benetos, and Simon Dixon. Automatically detecting key modulations in J.S. Bach chorale recordings. In *Proceedings of the Sound and Music Computing Conference (SMC)*, pages 25–32, Padova, Italy, 2011.

[114] Stylianos I. Mimilakis, Christof Weiß, Vlora Arifi-Müller, Jakob Abeßer, and Meinard Müller. Cross-version singing voice detection in opera recordings: Challenges for supervised learning. In *Proceedings of the International Workshop on Machine Learning and Music (MML)*, Würzburg, Germany, 2019.

[115] Dirk Moelants and Martin F. McKinney. Tempo perception and musical content: What makes a piece fast, slow or temporally ambiguous. In *Proceedings of the International Conference on Music Perception and Cognition (ICMPC)*, pages 558–562, Evanston, IL, USA, 2004.

[116] Christopher Montgomery. 24/192 music downloads ... and why they make no sense. `https://people.xiph.org/~xiphmont/demo/neil-young.html`, last accessed 11/20/2019, March 2012.

[117] Meinard Müller. *Fundamentals of Music Processing*. Springer Verlag, 2015. ISBN 978-3-319-21944-8.

[118] Meinard Müller, Frank Kurth, and Michael Clausen. Chroma-based statistical audio features for audio matching. In *Proceedings of the IEEE Workshop on Applications of Signal Processing (WASPAA)*, pages 275–278, New Paltz, NY, USA, October 2005.

[119] Meinard Müller, Christof Weiss, and Stefan Balke. Deep neural networks in MIR. In *Proceedings of the Jahrestagung der Gesellschaft für Informatik (GI)*, Chemnitz, Germany, 2017.

[120] Hans Georg Musmann. Genesis of the MP3 audio coding standard. *IEEE Transactions on Consumer Electronics*, 52(3):1043–1049, 2006.

[121] João Lobato Oliveira, Fabien Gouyon, Luis Gustavo Martins, and Luís Paulo Reis. IBT: A real-time tempo and beat tracking system. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 291–296, Utrecht, The Netherlands, 2010.

[122] João Lobato Oliveira, Matthew E. P. Davies, Fabien Gouyon, and Luís Paulo Reis. Beat tracking for multiple applications: A multi-agent system architecture with state recovery. *IEEE Transactions on Audio, Speech & Language Processing*, 20(10):2696–2706, 2012.

[123] Elias Pampalk, Arthur Flexer, and Gerhard Widmer. Improvements of audio-based music similarity and genre classification. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 628–633, London, UK, 2005.

[124] Hélène Papadopoulos and Geoffroy Peeters. Local key estimation from an audio signal relying on harmonic and metrical structures. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(4):1297–1312, 2012.

[125] Geoffroy Peeters. Time variable tempo detection and beat marking. In *Proceedings of the International Computer Music Conference (ICMC)*, Barcelona, Spain, 2005.

[126] Geoffroy Peeters. Template-based estimation of time-varying tempo. *EURASIP Journal on Advances in Signal Processing*, 2007.

[127] Geoffroy Peeters and Joachim Flocon-Cholet. Perceptual tempo estimation using GMM-regression. In *Proceedings of the international ACM workshop on Music information retrieval with user-centered and multimodal strategies (MIRUM)*, pages 45–50, Nara, Japan, 2012.

[128] Graham Percival and George Tzanetakis. Streamlined tempo estimation based on autocorrelation and cross-correlation with pulses. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 22(12):1765–1776, 2014.

[129] Trevor Pinch and Frank Trocco. *Analog days: The invention and impact of the Moog synthesizer*. Harvard University Press, 2009.

[130] Jordi Pons and Xavier Serra. Designing efficient architectures for modeling temporal features with convolutional neural networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 2472–2476, New Orleans, Louisiana, USA, March 2017.

[131] Jordi Pons, Thomas Lidy, and Xavier Serra. Experimenting with musically motivated convolutional neural networks. In *Proceedings of International Workshop on Content-Based Multimedia Indexing (CBMI)*, pages 1–6, Bucharest, Romania, 2016.

[132] Jordi Pons, Oriol Nieto, Matthew Prockup, Erik Schmidt, Andreas Ehmann, and Xavier Serra. End-to-end learning for music audio tagging at scale. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 637–644, Paris, France, 2018.

[133] Alastair Porter, Dmitry Bogdanov, Robert Kaye, Roman Tsukanov, and Xavier Serra. AcousticBrainz: a community platform for gathering music information obtained from audio. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 786–792, Málaga, Spain, 2015.

[134] Hendrik Purwins, Benjamin Blankertz, and Klaus Obermayer. Constant Q profiles for tracking modulations in audio data. In *Proceedings of the International Computer Music Conference (ICMC)*, Havana, Cuba, 2001.

[135] Elio Quinton, Florabelle Spielmann, and Bob L. Sturm. Computational ethnomusicology for exploring trends in trinidad steelband music through history. In *Proceedings of the International Symposium on Computer Music Multidisciplinary Research (CMMR)*, Matosinhos, Portugal, 2017.

[136] Colin Raffel. *Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching*. PhD thesis, Columbia University, USA, 2016.

[137] Colin Raffel, Brian McFee, Eric J. Humphrey, Justin Salamon, Oriol Nieto, Dawen Liang, and Daniel P. W. Ellis. MIR_EVAL: A transparent implementation of common MIR metrics. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, pages 367–372, Taipei, Taiwan, 2014.

[138] Bruno H. Repp. Diversity and commonality in music performance: An analysis of timing microstructure in Schumann's "Träumerei". *The Journal of the Acoustical Society of America*, 92(5):2546–2568, 1992.

[139] Bruno H. Repp. On determining the basic tempo of an expressive music performance. *Psychology of Music*, 22(2):157–167, 1994.

[140] Bruno H. Repp. Sensorimotor synchronization: A review of the tapping literature. *Psychonomic Bulletin & Review*, 12(6):969–992, 2005.

[141] Thomas Rocher, Matthias Robine, Pierre Hanna, and Laurent Oudre. Concurrent estimation of chords and keys from audio. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 141–146, Utrecht, The Netherlands, 2010.

[142] Miguel A. Roig-Francolí. *Harmony in Context*. McGraw-Hill Humanities/Social Sciences/Languages, New York, USA, 2011.

[143] Justin Salamon. What's broken in music informatics research? Three uncomfortable statements. In *International Conference on Machine Learning (ICML), Workshop on Machine Learning for Music Discovery*, Long Beach, CA, USA, 2019.

[144] Justin Salamon and Emilia Gómez. Melody extraction from polyphonic music signals using pitch contour characteristics. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(6): 1759–1770, 2012.

[145] Justin Salamon and Julián Urbano. Current challenges in the evaluation of predominant melody extraction algorithms. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 289–294, Porto, Portugal, October 2012.

[146] Craig Stuart Sapp. Visual hierarchical key analysis. *ACM Computers in Entertainment*, 3(4):1–19, 2005.

[147] Craig Stuart Sapp. Hybrid numeric/rank similarity metrics. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 501–506, Philadelphia, USA, 2008.

[148] Markus Schedl, Arthur Flexer, and Julián Urbano. The neglected user in music information retrieval research. *Journal of Intelligent Information Systems*, 41(3):523–539, 2013.

[149] Markus Schedl, Hamed Zamani, Ching-Wei Chen, Yashar Deldjoo, and Mehdi Elahi. Current challenges and visions in music recommender systems research. *International Journal of Multimedia Information Retrieval*, 7(2):95–116, 2018.

[150] Eric D. Scheirer. Tempo and beat analysis of acoustical musical signals. *Journal of the Acoustical Society of America*, 103(1):588–601, 1998.

[151] Alexander Schindler, Thomas Lidy, and Andreas Rauber. Comparing shallow versus deep neural network architectures for automatic music genre classification. In *Proceedings of the Forum Media Technology (FMT)*, volume 1734, pages 17–21, St. Pölten, Austria, 2016.

[152] Jan Schlüter and Sebastian Böck. Improved musical onset detection with convolutional neural networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 6979–6983, Florence, Italy, May 2014. doi: 10.1109/ICASSP.2014.6854953.

[153] Hendrik Schreiber. jipes. `http://www.tagtraum.com/jipes/`, last accessed 11/18/2019, 2011.

[154] Hendrik Schreiber. Improving genre annotations for the Million Song Dataset. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 241–247, Málaga, Spain, 2015.

[155] Hendrik Schreiber. Genre ontology learning: Comparing curated with crowd-sourced ontologies. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 400–406, New York City, NY, USA, 2016.

[156] Hendrik Schreiber. CNN-based automatic musical key detection. In *Music Information Retrieval Evaluation eXchange (MIREX)*, Suzhou, China, 2017.

[157] Hendrik Schreiber. CNN-based tempo estimation. In *Music Information Retrieval Evaluation eXchange (MIREX)*, Paris, France, 2018.

[158] Hendrik Schreiber and Meinard Müller. Exploiting global features for tempo octave correction. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 639–643, Florence, Italy, 2014.

[159] Hendrik Schreiber and Meinard Müller. Accelerating index-based audio identification. *IEEE Transactions on Multimedia*, 16(6):1654–1664, 2014.

[160] Hendrik Schreiber and Meinard Müller. A post-processing procedure for improving music tempo estimates using supervised learning. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 235–242, Suzhou, China, 2017.

[161] Hendrik Schreiber and Meinard Müller. A single-step approach to musical tempo estimation using a convolutional neural network. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 98–105, Paris, France, 2018.

[162] Hendrik Schreiber and Meinard Müller. A crowdsourced experiment for tempo estimation of electronic dance music. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 409–415, Paris, France, 2018.

[163] Hendrik Schreiber and Meinard Müller. Musical tempo and key estimation using convolutional neural networks with directional filters. In *Proceedings of the Sound and Music Computing Conference (SMC)*, pages 47–54, Málaga, Spain, 2019.

[164] Hendrik Schreiber, Peter Grosche, and Meinard Müller. A re-ordering strategy for accelerating index-based audio fingerprinting. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, pages 127–132, Miami, Florida, USA, 2011.

[165] Hendrik Schreiber, Julián Urbano, and Meinard Müller. Music tempo estimation: Are we done yet? *Transactions of the International Society for Music Information Retrieval (TISMIR)*, 2020.

[166] Hendrik Schreiber, Christof Weiss, and Meinard Müller. Local key estimation in classical music recordings: A cross-version study on Schubert's Winterreise. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Barcelona, Spain, 2020.

[167] Hendrik Schreiber, Frank Zalkow, and Meinard Müller. Modeling and estimating local tempo: A case study on Chopin's Mazurkas. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Montreal, Quebec, Canada, October 2020.

[168] Björn Schuller, Florian Eyben, and Gerhard Rigoll. Tango or waltz?: Putting ballroom dance style into tempo detection. *EURASIP Journal on Audio, Speech, and Music Processing*, 2008.

[169] Jarno Seppänen. Tatum grid analysis of musical signals. In *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 131–134, New Paltz, NY, USA, 2001.

[170] Jarno Seppänen, Antti J. Eronen, and Jarmo Hiipakka. Joint beat & tatum tracking from music signals. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 23–28, Victoria, Canada, 2006.

[171] Xavier Serra, Michela Magas, Emmanouil Benetos, Magdalena Chudy, Simon Dixon, Arthur Flexer, Emilia Gómez, Fabien Gouyon, Perfecto Herrera, Sergi Jordà, Oscar Paytuvi, Geoffroy Peeters, Jan Schlüter, Hugues Vinet, and Gerhard Widmer. *Roadmap for Music Information Research*. 2013. ISBN 978-2-9540351-1-6.

[172] Daniel Servén and Charlie Brummitt. pyGAM: generalized additive models in python. Zenodo, 2018. `DOI:10.5281/zenodo.1208723`.

[173] Siddharth Sigtia, Nicolas Boulanger-Lewandowski, and Simon Dixon. Audio chord recognition with a hybrid recurrent neural network. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 127–133, Málaga, Spain, 2015.

[174] Patrice Y. Simard, David Steinkraus, John C. Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR)*, volume 3, pages 958–962, Edinburgh, Scotland, UK, 2003.

[175] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL `http://arxiv.org/abs/1409.1556`.

[176] Malcolm Slaney. Web-scale multimedia analysis: Does content matter? *Multimedia, IEEE*, 18(2): 12–15, 2011.

[177] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, June 2014.

[178] Gerhard Stoll and Karlheinz Brandenburg. The ISO/MPEG-audio codec: A generic standard for coding of high quality digital audio. In *Proceedings of the Audio Engineering Society Convention (AES)*, 1992.

[179] Bob L. Sturm. Classification accuracy is not enough. *Journal of Intelligent Information Systems*, 41 (3):371–406, 2013.

[180] Bob L. Sturm. The GTZAN dataset: Its contents, its faults, their effects on evaluation, and its future use. *Computing Research Repository (CoRR)*, 2013. URL `http://arxiv.org/abs/1306.1461`.

[181] Bob L. Sturm. A simple method to determine if a music information retrieval system is a "horse". *IEEE Transactions on Multimedia*, 16(6):1636–1644, 2014.

[182] Bob L. Sturm. Revisiting priorities: improving MIR evaluation practices. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 488–494, New York, NY, USA, 2016.

[183] Bob L. Sturm, Rolf Bardeli, Thibault Langlois, and Valentin Emiya. Formalizing the problem of music description. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 89–94, Taipei, Taiwan, 2014.

[184] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, Boston, MA, USA, 2015.

[185] George Tzanetakis and Perry Cook. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293–302, 2002.

[186] George Tzanetakis and Graham Percival. An effective, simple tempo estimation method based on self-similarity and regularity. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 241–245, Vancouver, Canada, 2013.

[187] Julián Urbano, Mónica Marrero, and Diego Martín. On the measurement of test collection reliability. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 393–402, Dublin, Ireland, 2013.

[188] Julián Urbano, Markus Schedl, and Xavier Serra. Evaluation in music information retrieval. *Journal of Intelligent Information Systems*, 41(3):345–369, 2013.

[189] Steven van de Par, Martin F. McKinney, and André Redert. Musical key extraction from audio using profile training. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 328–329, Victoria, Canada, 2006.

[190] Mijk van Dijk. Native Instruments Traktor: Evolution der DJ-Software. *www.bonedo.de*, 2018, last accessed 11/13/2019. URL `https://www.bonedo.de/artikel/einzelansicht/native-instruments-traktor-evolution-der-dj-software.html`.

[191] Fabio Vignoli and Steffen Pauws. A music retrieval system based on user driven similarity and its evaluation. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 272–279, London, UK, 2005.

[192] Jim Waterhouse, Pollyana Hudson, and Ben Edwards. Effects of music tempo upon submaximal cycling performance. *Scandinavian Journal of Medicine & Science in Sports*, 20(4):662–669, 2010.

[193] Christof Weiß. Global key extraction from classical music audio recordings based on the final chord. In *Proceedings of the Sound and Music Computing Conference (SMC)*, pages 742–747, Stockholm, Sweden, 2013.

[194] Christof Weiß and Julian Habryka. Chroma-based scale matching for audio tonality analysis. In *Proceedings of the Conference on Interdisciplinary Musicology (CIM)*, pages 168–173, Berlin, Germany, 2014.

[195] Christof Weiß, Frank Zalkow, Meinard Müller, Stephanie Klauk, and Rainer Kleinertz. Computergestützte Visualisierung harmonischer Verläufe: Eine Fallstudie zu Wagners Ring. In *Proceedings of the Jahrestagung der Gesellschaft für Informatik (GI)*, pages 205–217, Chemnitz, Germany, 2017.

[196] Fu-Hai Frank Wu. Musical tempo octave error reducing based on the statistics of tempogram. In *Proceedings of the Mediterranean Conference on Control and Automation (MED)*, pages 993–998, Torremolinos, Spain, 2015.

[197] Fu-Hai Frank Wu and Jyh-Shing Roger Jang. A supervised learning method for tempo estimation of musical audio. In *Proceedings of the Mediterranean Conference of Control and Automation (MED)*, pages 599–604, Palermo, Italy, 2014.

[198] Fu-Hai Frank Wu, Tsung-Chi Lee, Jyh-Shing Roger Jang, Kaichun K. Chang, Chun-Hung Lu, and Wen-Nan Wang. A two-fold dynamic programming approach to beat tracking for audio music with time-varying tempo. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 191–196, Miami, Florida, USA, 2011.

[199] Linxing Xiao, Aibo Tian, Wen Li, and Jie Zhou. Using statistic model to capture the association between timbre and perceived tempo. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 659–662, Philadelphia, USA, 2008.

[200] Frank Zalkow, Christof Weiß, Thomas Prätzlich, Vlora Arifi-Müller, and Meinard Müller. A multi-version approach for transferring measure annotations between music recordings. In *Proceedings of the AES International Conference on Semantic Audio*, pages 148–155, Erlangen, Germany, 2017.

[201] Jose R. Zapata and Emilia Gómez. Comparative evaluation and combination of audio tempo estimation approaches. In *Proceedings of the Audio Engineering Society (AES) Conference on Semantic Audio*, Ilmenau, Germany, 2011.

[202] Yongwei Zhu and Mohan S. Kankanhalli. Music scale modeling for melody matching. In *Proceedings of the ACM International Conference on Multimedia*, pages 359–362, New York, USA, 2003.

Index