Friedrich-Alexander-Universität Erlangen-Nürnberg

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Master Thesis

Deep-Learning Approaches for Fundamental Frequency Estimation of Music Recordings

submitted by

Judith Bauer

submitted

November 18, 2019

Supervisor / Advisor

Prof. Dr. Meinard Müller
M.Sc. Sebastian Rosenzweig

Reviewers

Prof. Dr. Meinard Müller
Prof. Dr. Andreas Maier

AUDIO LABS

International Audio Laboratories Erlangen
A joint institution of the
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) and
the Fraunhofer-Institut für Integrierte Schaltungen IIS.

Fraunhofer IIS

Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form in einem Verfahren zur Erlangung eines akademischen Grades vorgelegt.

Erlangen, November 18, 2019

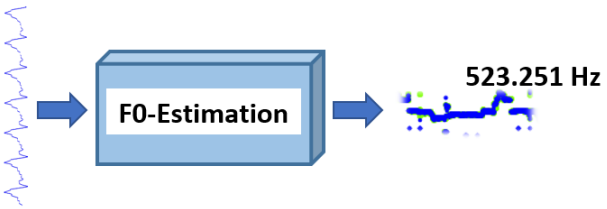
Judith Bauer

Acknowledgements

First, I would like to thank my supervisors Sebastian Rosenzweig and Prof. Dr. Meinard Müller from the International Audio Laboratories Erlangen. Sebastian Rosenzweig was always helpful and supporting in case of questions and provided me with constructive critics and suggestions. Thanks also to Prof. Dr. Meinard Müller for the opportunity to write my Master thesis in this interesting research area and his guidance and continuous support. Furthermore, I would like to extend my thanks to the complete AudioLabs group for the friendly working atmosphere.

Many thanks of course also to my family and Manuel for the support and encouragement.

Abstract



A main characteristic of music is its melody. A melody is a sequence of notes. Notes are characterized by their duration and pitch. So the investigation of pitches plays an important role in music processing. The task of extracting pitches closely relates to the examination of the fundamental frequencies (F0) of an audio recording. The pitch is a perceived property of a note whereas the fundamental frequency is a physical property of the waveform of an audio recording. For most cases, the pitch corresponds to the fundamental frequency. F0-estimation is a challenging task, because each note played on an instrument consists of the fundamental frequency and multiple (differently weighted) overtones. For polyphonic audio recordings with simultaneously occurring notes, this task becomes even more difficult.

For fundamental frequency estimation, several approaches already exist. There are multiple model-based approaches that use techniques from signal processing. Additionally, with the rise of artificial intelligence and deep learning, approaches using neural networks are developed for F0-estimation.

In this thesis, we first give an overview of model-based approaches and evaluate them for some audio examples. As main contribution, we analyze and evaluate CREPE, a convolutional neural network for F0-estimation developed by Kim et al. [1]. Then, as a next step, modifications for the network and the training process are presented and evaluated. This includes data augmentation, layer freezing, and modifications to the network structure. Finally, CREPE is evaluated on a dataset consisting of Georgian vocal music.

The result of our experiments is that CREPE achieves high accuracies on a dataset of resynthesized audio recordings and performs similar or better compared to previous model-based approaches. We also successfully retrained the network. In order to improve training in the presence of few training samples we found out that data augmentation is a useful technique. The training process was accelerated using layer freezing or a smaller network with only a slight decline in accuracies.

Zusammenfassung

Eine wichtige Eigenschaft von Musik ist die Melodie. Eine Melodie ist eine Sequenz von Tönen. Töne werden durch ihre Dauer und die Tonhöhe beschrieben. Deshalb spielt die Untersuchung von Tonhöhen eine wichtige Rolle in der Musikverarbeitung. Die Extraktion von Tonhöhen hängt eng mit der Untersuchung der Fundamentalfrequenzen (F0) einer Audioaufnahme zusammen. Die Tonhöhe ist eine wahrgenommene Eigenschaft eines Tons, während es sich bei der Fundamentalfrequenz um eine physikalische Eigenschaft der Wellenform einer Audioaufnahme handelt. In den meisten Fällen entspricht die wahrgenommene Tonhöhe der Fundamentalfrequenz. F0-Schätzung ist eine anspruchsvolle Aufgabe, weil jeder Ton, der auf einem Instrument gespielt wird, sich aus der Fundamentalfrequenz und mehreren (unterschiedlich gewichteten) Obertönen zusammensetzt. Für mehrstimmige Audioaufnahmen mit gleichzeitig erklingenden Tönen wird diese Aufgabe noch schwieriger.

Zur Fundamentalfrequenzschätzung existieren bereits mehrere Ansätze. Es gibt modellbasierte Ansätze, die Techniken aus der Signalverarbeitung nutzen. Durch den immer größeren Einfluss von künstlicher Intelligenz und Deep Learning werden auch zunehmend Ansätze zur F0-Schätzung mit neuronalen Netzen entwickelt.

In dieser Arbeit geben wir zuerst einen Überblick über modellbasierte Ansätze und wenden diese auf ausgewählte Audiobeispiele an. Der Hauptbeitrag dieser Arbeit ist die Analyse und Evaluation von CREPE, einem Convolutional Neural Network zur F0-Schätzung von Kim et al. [1]. Des Weiteren werden Anpassungen an dem Netz und dem Trainingsprozess vorgestellt und evaluiert. Dazu gehören die Anreicherung der Daten, die Fixierung von Schichten und Änderungen der Netzstruktur. Abschließend wird CREPE auf einem Datensatz mit georgischer Vokalmusik evaluiert.

Das Ergebnis unserer Experimente ist, dass CREPE hohe Genauigkeiten auf einem Datensatz mit resynthetisierten Daten erzielt und vergleichbar oder besser abschneidet als bisherige Ansätze. Des Weiteren war ein erneutes Training des Netzes erfolgreich. Zur Verbesserung des Trainings mit einem eingeschränkten Datensatz hat sich die Anreicherung der Daten als nützlich erwiesen. Außerdem konnten wir das Training des Netzes mit Hilfe von fixierten Schichten oder einer reduzierten Netzgröße beschleunigen, was nur zu geringfügig niedrigeren Genauigkeiten führte.

Contents

Chapter 1

Introduction

Music and speech recordings are complex data with a high informational content which can be analyzed automatically. In both music and speech, the pitch is a characteristic attribute. In speech, the pitch depends on the speaker and the context of the speech recording (surrounding, situation of the speaker). While speaking, the pitch changes and indicates for example that a sentence is a question or a request. Also in music, pitch is an important aspect to be analyzed. The pitches correspond to the musical notes and therefore describe the melody of a music recording. The results of pitch estimation can be used for example to convert a recording into a symbolic representation using a sequence of musical notes. Another application is to synchronize a music recording with an existing music transcription by analyzing the pitches of the recording using pitch estimation.

Pitch is a property of a note perceived by the listener [2]. In contrast, the fundamental frequency (F0) is a physical property of the waveform of an audio recording. Although, the pitch corresponds to the fundamental frequency for most cases, so we use these terms interchangeably. The research concerning fundamental frequency estimation can be separated into different tasks. First, it can be divided into F0-estimation for monophonic audio and F0-estimation for polyphonic audio. When analyzing the pitches of polyphonic audio, the object can either be to analyze the fundamental frequencies of the main melody (predominant melody estimation) or to analyze multiple simultaneously occurring pitches, e.g. multiple music instruments (multipitch estimation). In this thesis, we focus on monophonic pitch tracking as basic task of pitch estimation.

For monophonic pitch estimation, the input is a monophonic audio recording. At each time instance, there is only one musical note played. A note played on a melodic instrument usually consist of a fundamental frequency. Additionally, there are harmonic partials which can be differently weighted depending on the timbre of the instrument. The task is to examine the music recordings over time and determine the fundamental frequencies at these time instances. An example using an excerpt of “In the Hall of the Mountain King” (Edvard Grieg) is shown

in Figure 1.1. In Figure 1.1a, the sheet music representation is displayed. Figure 1.1b shows the corresponding waveform. The expected sequence of fundamental frequencies is shown in Figure 1.1c. Several approaches for fundamental frequency estimation exist. These can be divided into model-based methods (e.g. YIN [3] and pYIN [4]) and approaches based on deep neural networks (DNN). Model-based approaches use techniques from signal processing to analyze a music signal. DNN-based approaches are data-driven. They are trained using annotated data consisting of audio excerpts and the correct fundamental frequencies for the excerpts. A recently published approach is CREPE (A Convolutional Representation for Pitch Estimation) developed by Kim et al. [1], which is based on a deep convolutional neural network. The input for CREPE is an audio recording (as in Figure 1.1b) and the output is a sequence of estimated fundamental frequencies (as in Figure 1.1c). In this thesis, we use the network of CREPE as an example for a data-driven approach for fundamental frequency estimation and compare it to model-based approaches. As a main contribution, we present modifications to the network and evaluate the modifications by retraining the network.

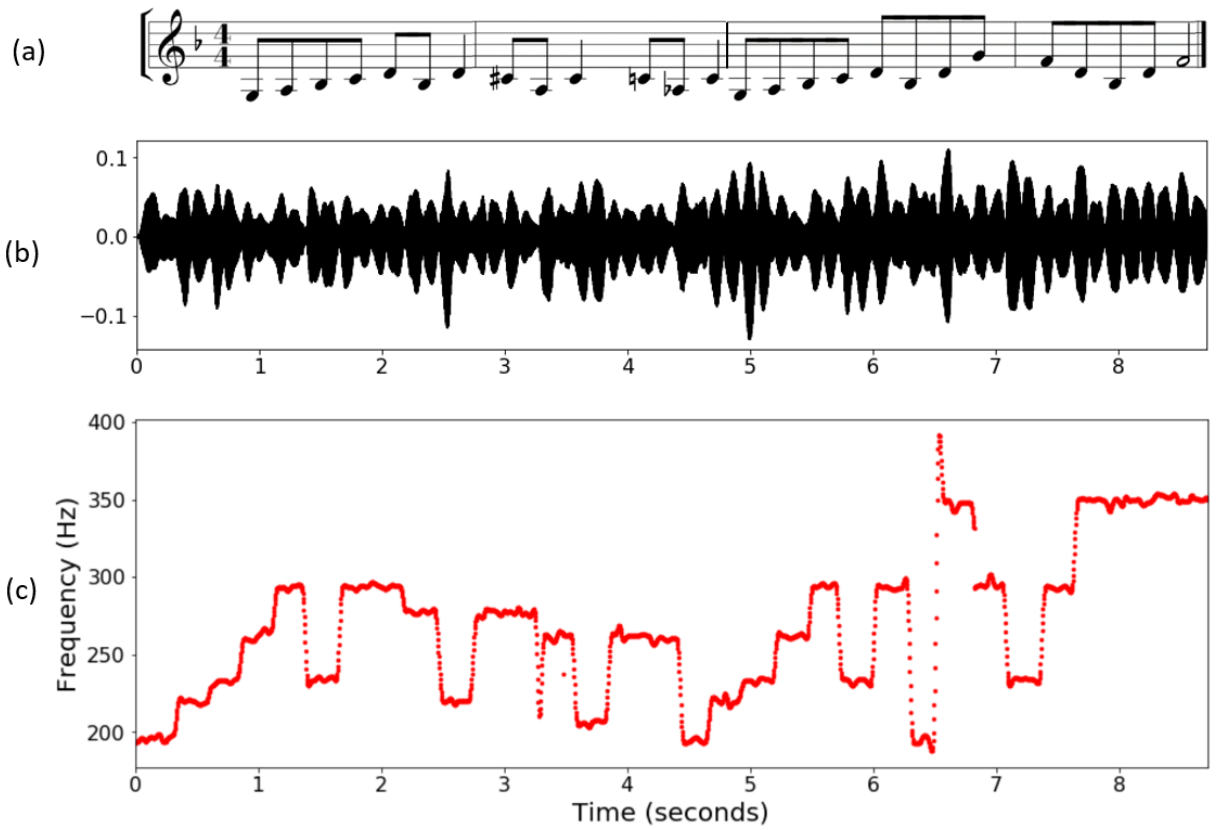


Figure 1.1: Short excerpt of "In the Hall of the Mountain King" (Edvard Grieg) played by a viola. **(a)** Sheet music representation. **(b)** Waveform of a resynthesized recording. **(c)** Sequence of fundamental frequency values from the annotation.

1.1 Structure of Thesis

In Chapter 2, we discuss the fundamentals that are necessary for this thesis. This includes basic aspects of signal processing. Then the task of fundamental frequency estimation is explained. For the development and the evaluation of F0-estimation approaches, some datasets are necessary, so the utilized datasets are presented. Furthermore, we discuss evaluation metrics for pitch estimation and voicing estimation.
In Chapter 3, we give an overview of model-based approaches for fundamental frequency estimation. This includes YIN [3] and its probabilistic version pYIN [4]. Also a pitch estimator using a sawtooth function (SWIPE [5, 6]) and an algorithm for polyphonic music (MELODIA [7]) are presented.
In Chapter 4, we discuss models based on deep neural networks. First, the convolutional neural network CREPE [1] is presented. It is examined using some artificial signals. Then, we develop modifications using data augmentation, layer freezing and changes to the network structure.
We evaluate CREPE and the new approaches in Chapter 5. For evaluation, a dataset with monophonic audio stems and pitch annotations (MDB-stem-synth [8]) is used.
Finally, we use a more realistic dataset of traditional Georgian vocal music for evaluation of pitch estimation and voicing estimation in Chapter 6.

1.2 Main Contributions

The first contribution of this thesis is the analysis of CREPE [1]. Here, the functioning is examined using artificially generated data of sinusoidals, a chirp signal and a signal with harmonics. CREPE is also applied to some examples of the MDB-stem-synth dataset [8] and the results are compared to the results obtained with other pitch estimators. Then, the original version of CREPE is applied to the full MDB-stem-synth dataset.
As our main contribution, we test several strategies to improve the network performance and the training process. First, we use data augmentation in order to increase the available dataset for training and compensate for biases in the dataset. For this, we apply pitch shifting to the audio files. Second, we examine the role of the first convolutional layer and exclude this layer from training by precomputing appropriate weights. Next, we analyze the output layer and increase its dimension. Then, we modify the network structure and decrease the filter sizes and the number of filters in order to create a smaller network.
We evaluate these modifications using the MDB-stem-synth dataset [8]. In summary, we show that data augmentation (pitch shifting) is useful for improving the network performance. Furthermore, we show that the training process can be accelerated using fixed weights for the first layer and a smaller network.

Chapter 2

Fundamentals

In this chapter, we present the fundamentals that are necessary for this thesis. We start with the basics of signal processing (Fourier analysis and the short-time Fourier transform) in Section 2.1. Next, in Section 2.2, we present the task of fundamental frequency estimation. Then, the datasets that are used in the following chapters are introduced in Section 2.3. Here, also the running examples are presented. Finally, we introduce evaluation measures for pitch estimation and voicing estimation in Section 2.4.

2.1 Signal Processing

Music is physically represented by audio signals. When, e.g. an instrument is played, this creates air pressure changes. At one location, these air pressure changes can be measured over time. This leads to a pressure-time signal which captures properties of the music like pitch, loudness and timbre. Mathematically, the signal is a function $f : \mathbb{R} \to \mathbb{R}$. So each point in time t is mapped to a value $f(t)$. If a pattern of air pressure occurs repeatedly after a period of time, it is a periodic waveform. The number of periods per second (reciprocal of the period) is the frequency ω. It is measured in Hertz (Hz). In order to examine the frequencies in a signal, Fourier analysis is an important tool.

2.1.1 Fourier Analysis

The object of the Fourier analysis is to examine which frequencies are present in a signal. For each signal, it is possible to decompose it into weighted sinusoids of different frequencies (and phases). Figure 2.1 shows an example signal (blue curve) that can be decomposed into two components (gray lines) using the Fourier transform. The representation of the original signal

as two sinusoids with weights 1 and 0.7 is the Fourier representation of the signal. When an

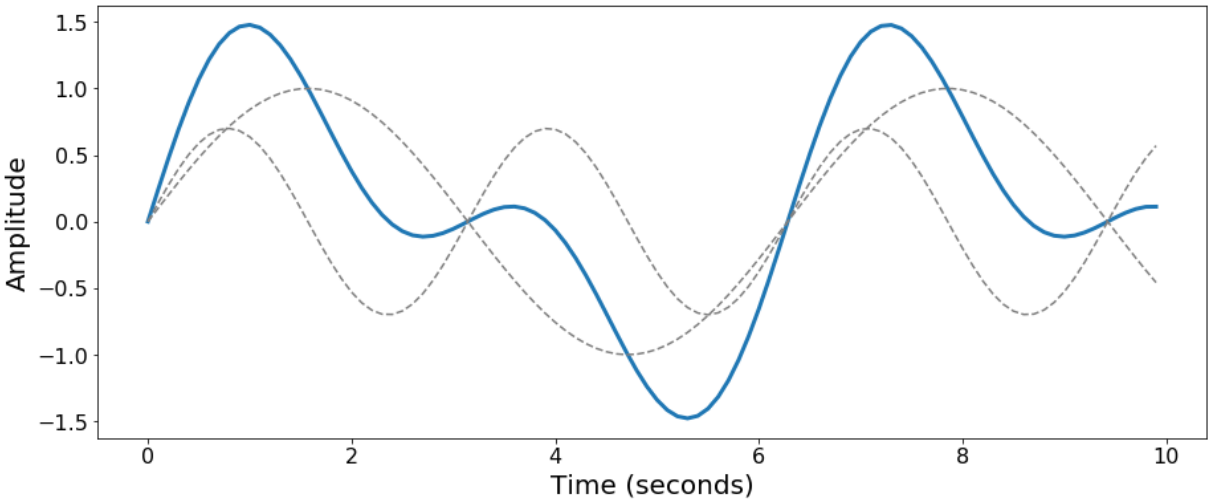


Figure 2.1: The blue signal can be decomposed into the two grey signals: sin(t) + 0.7 * sin(2t).

ideal artificial single tone is produced, it causes a waveform with a frequency specific to the pitch of the tone. For example if an artificial A4 is produced it leads to changes in air pressure with a frequency of 440 Hz. So analyzing the sinusoids and their amplitudes in a waveform corresponds to analyzing the pure tones of which the signal is composed. To approach the present frequencies and their amplitudes, the Fourier transform is computed. As continuous functions cannot be processed by computers, a discrete version of the Fourier transform is often used. For this, instead of the original signal f, we use a discrete-time signal x that is sampled equidistantly with a sampling period T

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

The sampling rate F_s in Hertz (Hz) can be derived from the sampling period by taking the inverse

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

The chosen sampling rate determines whether the original signal can be reconstructed from the discretized signal. According to the sampling theorem, reconstruction is possible, if there are no frequencies higher than $F_\mathrm{s}/2$. So if a signal has a sampling rate of e.g. 44100 Hz, the original signal can be reconstructed if it contains only frequencies below 22050 Hz (humans can hear frequencies until approximately 20 kHz). The formula for the discrete Fourier transform (DFT) is (cf. [9, Section 2.1.3]):

$$X(k) = \sum_{n=0}^{N-1} x(n)\exp(-2\pi ikn/N) \tag{2.3}$$

for samples $x(0), x(1), ..., x(N-1)$ and frequency indices $k \in [0 : N-1]$.
$X(k)$ then corresponds to the physical frequency

$$F_{\text{coef}}(k) := \frac{k \cdot F_{\text{s}}}{N} \tag{2.4}$$

measured in Hertz.

2.1.2 Short-Time Fourier Transform

The discrete Fourier transform analyzes the frequencies of the complete signal. However, most audio signals change over time and are only consistent in a short time interval. So for audio processing, the short-time Fourier transform (STFT) can be used instead. It only examines the frequencies in a short interval. The size of the interval is determined by the window size $N \in \mathbb{N}$. A window with this size is slided over the signal in intervals determined by the hop size $H \in \mathbb{N}$. The Fourier transform is then computed for each frame. This leads to the following definition of the short-time Fourier transform for a discrete time signal $x : \mathbb{Z} \to \mathbb{R}$ (according to [9, Section 2.1.4]):

$$\mathcal{X}(m,k) = \sum_{n=0}^{N-1} x(n+mH)w(n)\exp(-2\pi ikn/N). \tag{2.5}$$

with $m \in \mathbb{Z}$ (time index), $k \in [0 : K]$ (frequency index) and $w : [0 : N-1] \to \mathbb{R}$ (window function). The time index m corresponds to the physical time

$$T_{\text{coef}}(m) := \frac{m \cdot H}{F_{\text{s}}} \tag{2.6}$$

measured in seconds.

From the short-time Fourier transform, the spectrogram can be computed by taking the magnitude:

$$\mathcal{Y}(m,k) = \left|\mathcal{X}(m,k)\right|. \tag{2.7}$$

After computing a spectrogram, a compressed spectrogram can be derived. This procedure balances out the sound components with low and high energy. The motivation for this is that the intensity of sounds is perceived logarithmically. So components with low energy are enhanced using logarithmic compression (cf. [9, Section 3.1.2.1]):

$$(\Gamma_\gamma \circ \mathcal{Y})(m,k) := \log(1 + \gamma \cdot \mathcal{Y}(m,k)) \tag{2.8}$$

with the logarithmic compression function $\Gamma_\gamma(v) := \log(1 + \gamma \cdot v)$ and $\gamma \in \mathbb{R}_{>0}$. The constant γ determines the amount of compression. A higher value of γ indicates a higher compression.

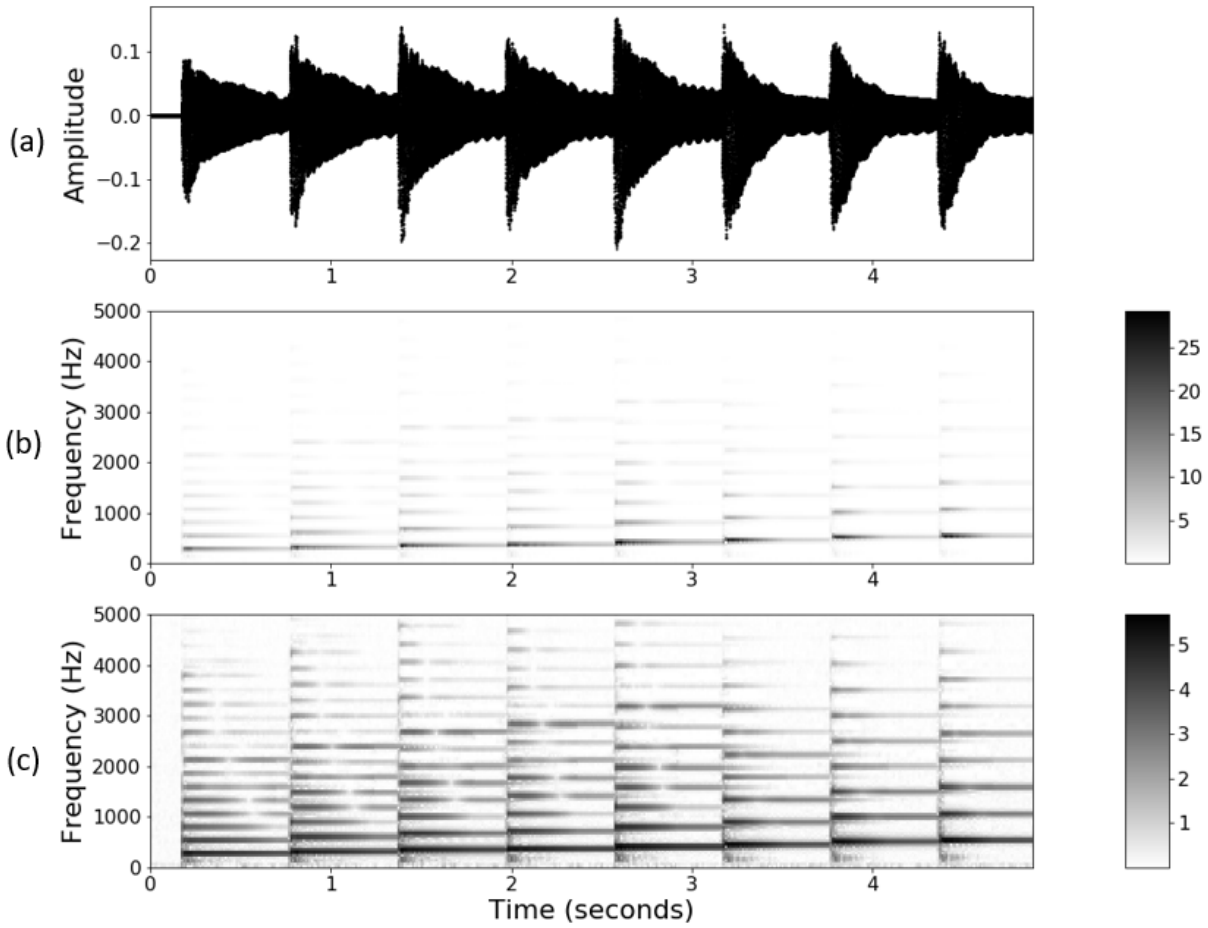


Figure 2.2: Signal of a C-major scale played by a piano (audio from [10]). **(a)** Waveform. **(b)** Magnitude spectrogram. **(c)** Compressed spectrogram ($\gamma = 10$).

For an example using a C-major scale played by a piano, see Figure 2.2 (audio from [10]). It shows the difference between a magnitude spectrogram and a logarithmically compressed spectrogram ($\gamma = 10$). In the compressed spectrogram, the components with a lower energy are more visible.

2.2 Fundamental Frequency Estimation

The analysis of the melody of an audio recording is closely related to the task of estimating the fundamental frequencies. This task, its variants and challenges are presented in this section. Additionally, we explain one approach to solve this task using salience-based fundamental frequency estimation.

2.2.1 Task Description

The task of fundamental frequency estimation can be separated into different tasks depending on the input audio. If the input is a monophonic audio recording with only one pitch present at each time frame, this is referred to as monophonic pitch estimation. The input can also be a polyphonic audio recording (multiple pitches present at some time frames). This can be the case if an instrument is recorded that produces multiple tones like a piano or if multiple instruments are recorded where each instrument produces one pitch like multiple flutes. Also a mixture of multiphonic and monophonic instruments is possible. When analyzing the pitches of a polyphonic audio recording, this task can be divided into predominant melody estimation and multipitch estimation. In predominant melody estimation, the objective is to identify the pitches belonging to the main melody (for some recordings difficult to determine). If all pitches should be investigated, this is referred to as multipitch estimation [11]. Here, we focus on monophonic pitch estimation as a basic task of pitch estimation.

When referring to pitch estimation, this is often used interchangeably with fundamental frequency estimation. As explained in Section 2.1, the signal of a recorded tone can be decomposed into multiple sinusoids. Each sinusoid is called a partial. The fundamental frequency (F0) is the frequency of the partial with the lowest frequency. This frequency is also referred to as the first partial or the first harmonic [9]. Further harmonics are the partials with a frequency that is an integer multiple of the fundamental frequency. So the fundamental frequency is a physical property of the waveform. Pitch, in contrast, is a property perceived by the listener [2]. For most cases, the pitch directly corresponds to the fundamental frequency. Still, there are examples where another partial has a higher amplitude then the first partial.

Analyzing the melody of an audio recording is an important part of music processing. A melody consists of multiple pitches over time. Therefore, when analyzing pitches of an audio recording, a sequence of frequency values should be computed over time. A frequency path over time is a frequency trajectory [9, Section 8.2]. Formally, a frequency trajectory can be defined as a function η

$$\eta : \mathbb{Z} \to \mathbb{R} \cup \{*\}. \tag{2.9}$$

The function η assigns to each time index a frequency value in Hertz (in $\mathbb{R}$) or a symbol ($*$) indicating that there is no melody present at this time instance (unvoiced). In pitch estimation, we are interested in the fundamental frequencies. Hence, the resulting trajectory is referred to as F0-trajectory.

To calculate a F0-trajectory from a monophonic recording, there are different approaches. They can be categorized as model-based or based on a deep neural network (DNN-based) approaches. Model-based approaches often use techniques from signal processing. They can either directly analyze the signal in the time-domain or transfer the signal to the frequency domain using the

Fourier transform and then analyze the spectrogram. Examples for model-based approaches are YIN [3], pYIN [4], SWIPE [5, 6] and Melodia [7]. These are presented in Chapter 3. Besides model-based approaches, there are DNN-based approaches. They use deep neural networks that are trained for the estimation of fundamental frequencies using annotated audio data. One example for an approach using a deep convolutional neural network is CREPE [1] which will be presented in Section 4.1.

2.2.2 Salience-Based Fundamental Frequency Estimation

For frequency-domain based approaches for fundamental frequency estimation, a salience representation based on a refined log-frequency spectrogram can be useful. We start with the computation of a short-time Fourier transform as explained in Section 2.1.2. The frequency bins of the STFT are equally spaced over the frequency space. But the perceived pitch is logarithmic in the frequency. So instead of directly using the frequency values, each frequency ω is assigned to a bin index $\mathrm{Bin}(\omega)$(cf. [9, Section 8.2.2.1]):

$$\mathrm{Bin}(\omega) := \left\lfloor \frac{1200}{R} \cdot \log_2 \left(\frac{\omega}{\omega_{\mathrm{ref}}} \right) + 1.5 \right\rfloor \tag{2.10}$$

with $\omega_{\mathrm{ref}} \in \mathbb{R}$ (reference frequency, assigned to bin index 1) and $R \in \mathbb{R}$ (resolution of frequency axis).

Using Equation 2.10, sets of frequency indices that are assigned to the same bin are defined (cf. [9, Section 8.2.2.1]):

$$P(b) := \{k : \mathrm{Bin}(F_{\mathrm{coef}}(k)) = b\} \tag{2.11}$$

for bin indices $b \in [1 : B]$. Now, we can define the refined log-frequency spectrogram by summing over all frequency indices belonging to the same set:

$$\mathcal{Y}_{\mathrm{LF}}(n, b) := \sum_{k \in P(b)} |\mathcal{X}(n, k)| \tag{2.12}$$

This refined log-frequency spectrogram can be further processed. First, in Section 2.2.2.1, a technique called “Instantaneous Frequency” will be explained which leads to a higher frequency resolution. Then, in Section 2.2.2.2, a salience representation is derived using harmonic summation. These sections follow the explanations in [9].

2.2.2.1 Instantaneous Frequency

Using the center frequencies $F_{\text{coef}}(k)$ as defined in Equation 2.4 often results in a blurry spectrogram. The reason for this is that the frequency resolution is too low. A higher frequency resolution is possible by increasing the window length N of the STFT. However, this also results in a decreased time resolution which might not be desired. This can be improved using the instantaneous frequency. It takes the phase information of the spectrogram into account to increase the frequency resolution without changing the time resolution. This section follows [9, Section 8.2.1 and Section 8.2.2.2].

The components of the Fourier representation of a signal are described by a frequency parameter $\omega \in \mathbb{R}$ and a phase parameter $\varphi \in [0, 1)$. When measuring the phase φ_1 at time t_1 and the phase φ_2 at time t_2, the phase φ^{Pred} can be predicted for time t_2

$$\varphi^{\text{Pred}} := \varphi_1 + \omega \cdot (t_2 - t_1), \tag{2.13}$$

if there is a frequency ω present in the signal. The prediction error is then defined by the difference between the correct phase and the predicted phase:

$$\varphi^{\text{Err}} := \Psi(\varphi_2 - \varphi^{\text{Pred}}) \tag{2.14}$$

with a function Ψ that maps the phase difference into the range [-0.5, 0.5]. Based on this, an improved frequency estimate can be computed:

$$\text{IF}(\omega) := \omega + \frac{\varphi^{\text{Err}}}{t_2 - t_1}. \tag{2.15}$$

$\text{IF}(\omega)$ is the instantaneous frequency at ω. This concept can be transferred to the STFT, leading to an instantaneous frequency $F^{\text{IF}}_{\text{coef}}(k, n)$. From this, bins of frequency indices can be computed and a refined log-frequency spectrogram $\mathcal{Y}^{\text{IF}}_{\text{LF}}$ is obtained.

2.2.2.2 Harmonic Summation

Depending on the instrument, each tone produces not only a signal of the fundamental frequency but also harmonic partials. The first partial is the fundamental frequency. The other partials are integer multiples of the first partial. This can be used to improve a spectrogram. So instead of directly using the frequencies estimated by the STFT, for each entry in the spectrogram a harmonic summation is performed.

For a spectrogram as defined in Equation 2.7, the harmonic-sum spectrogram is computed as follows (cf. [9, Section 8.2.2.3]):

$$\tilde{\mathcal{Y}}(n,k) := \sum_{h=1}^{H} \mathcal{Y}(n, k \cdot h) \tag{2.16}$$

for $n, k \in \mathbb{Z}$. The parameter $H \in \mathbb{N}$ determines the amount of harmonics that are used for the harmonic summation. When applying harmonic summation to a log-frequency spectrogram, the product turns into a sum. This is because in the original spectrogram, the partials are integer multiples. For log-frequency spectrograms, the partials have always the same distance, independent of the fundamental frequency. So a slightly modified formula has to be used.

When applying instantaneous frequency (see Section 2.2.2.1) to improve the frequency resolution and harmonic summation to further improve the spectrogram, the result is a spectrogram $\tilde{\mathcal{Y}}^{IF}_{LF}$. This spectrogram is the salience representation of an audio recording. From the salience representation, the fundamental frequencies can be estimated.

2.3 Datasets

For the evaluation of pitch estimation approaches and the training of DNN-based approaches, some audio files are necessary. The data used for this work can be divided into three groups. First, some artificial data like chirps and tones with harmonics were created. Then the MDB-stem-synth dataset [8] is used for training and evaluation. Additionally, a dataset of Georgian Vocal Music [12, 13] is used for evaluation on a more complex dataset.

2.3.1 Artificial Data

For the examination of a pitch estimation approach, we created some artificial signals. The three presented signals also serve as running artificial examples for the following chapters.

The first signal consists of sinusoids. It starts with a C2 (65.406 Hz), continues with an A5 (880.0 Hz), then there is a break and finally an E5 (659.255 Hz). The voiced parts have a duration of three seconds each, the break has a duration of two seconds. The magnitude spectrogram of this signal is shown in Figure 2.3a.

The second example is a signal with an additional harmonic partial. With this signal, the influence of partials for the estimation of fundamental frequencies can be examined. The duration is ten seconds. The signal has a frequency of 130.813 Hz (C3), which is the first harmonic. The amplitude of this frequency decreases from 1.0 to 0.0. Additionally, a frequency of 261.626 Hz (C4, second harmonic) is present, for which the amplitude increases from 0.0 to 1.0. The amplitudes

of both tones change linearly over the length of the signal. For the magnitude spectrogram see Figure 2.3b.

The third example is a chirp signal. It is useful in order to examine the pitch estimations for a larger range of frequencies. Here, we use a chirp which has a duration of 60 seconds. It starts with 30 Hz (between B0 flat and B0) and reaches to 2000 Hz (between B6 and C7). The fundamental frequency estimator CREPE is supposed to be applicable to frequencies between 31.70 Hz and 2005.50 Hz, so its range is similar to the range covered by the chirp. The magnitude spectrogram of the chirp signal is shown in Figure 2.3c.

All three signals (pure sinusoids, signal with harmonics, chirp signal) are created with a sampling rate of 16000 Hz.

2.3.2 MDB-stem-synth

Another dataset that will be used in this thesis is the MDB-stem-synth dataset [8]. It contains several annotated monophonic audio files and is therefore useful for larger evaluations and training of neural networks. In this section, we start with a description of the dataset and then introduce the running examples from this dataset.

2.3.2.1 Dataset Description

The MDB-stem-synth dataset [8] is useful for evaluating pitch estimation approaches. It consists of multiple audio files which are more realistic than the artificial signals presented in Section 2.3.1. So this dataset can also be used for training DNN-based pitch estimation approaches.

The MDB-stem-synth dataset is based on the MedleyDB dataset developed by Bittner et al. [14]. MedleyDB consists of royalty-free recordings from 122 songs including 108 melody-annotated songs. For each song, the stems (monophonic tracks) are also contained in the database. MedleyDB offers melody annotations for the stems. These annotations were created using pYIN [4] (a model-based approach for fundamental frequency estimation which will be discussed in Section 3.2) and were corrected manually.

This database was used as a basis for the MDB-stem-synth dataset. Salamon et al. [8] first applied a monophonic pitch tracker (Spectrum AutoCorrelation, SAC [15]) to some stems of the MedleyDB dataset. For the next step “Sinusoidal Modeling”, an algorithm developed by Bonada [16] is used. It segments the signal according to the periods estimated by SAC in the previous step and then computes harmonic parameters like frequency, amplitude and phase for each segment. Next, in the synthesis step, the harmonic parameters are used to create synthesized stems. For this, an oscillator is created (with the appropriate frequency and amplitude) for each

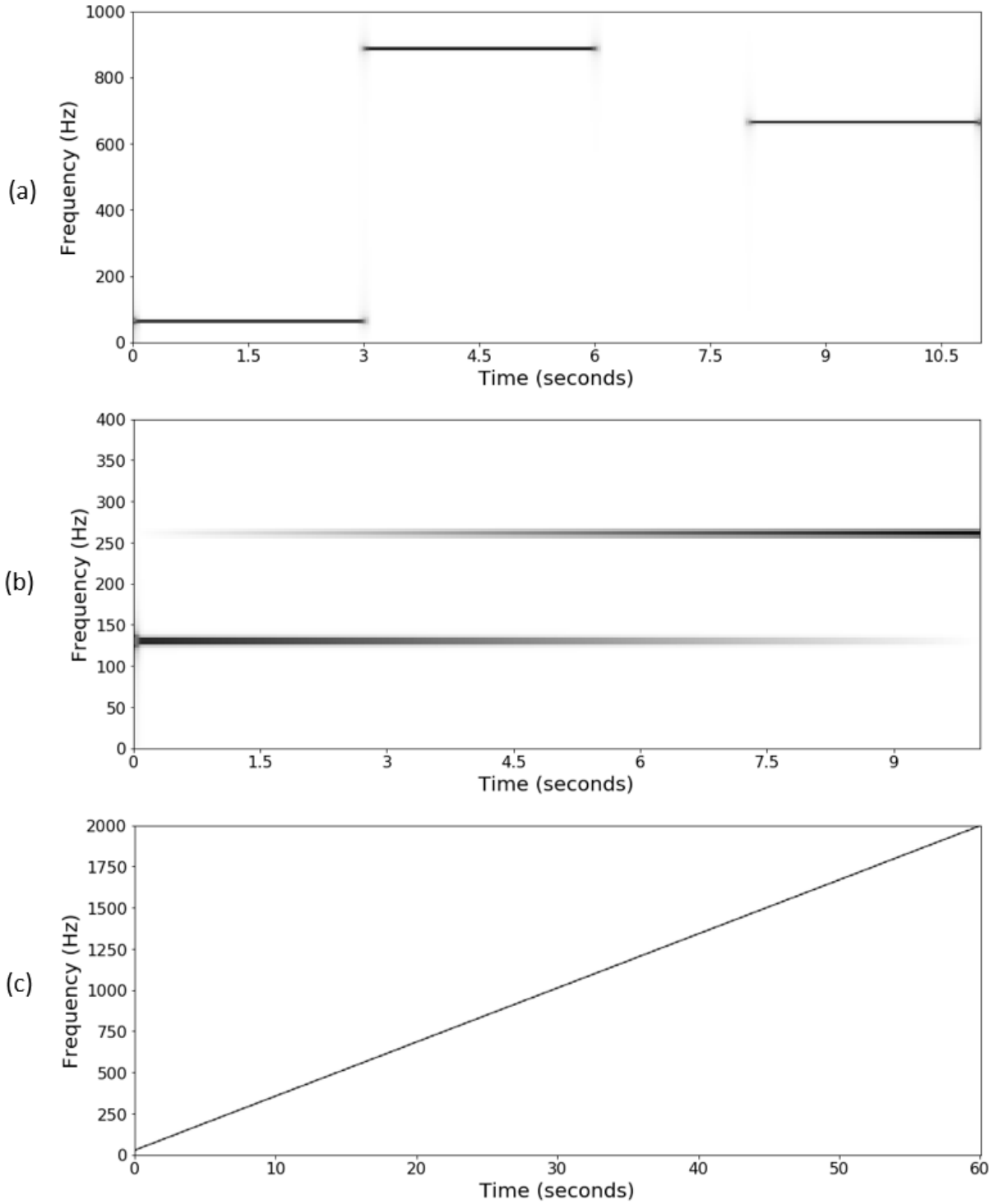


Figure 2.3: Magnitude spectrograms of running examples (artificial data). **(a)** Sinusoids. **(b)** Signal with harmonics. **(c)** Chirp signal.

partial. From the result of the synthesis step, a final mix of the created stems can be produced (Remixing). In this step, it is important to weight the stems according to the volume of the stems in the original mix. The resulting stems and the final mix maintain the original timbre and dynamics because of the consideration of partials.

Using the described method, Salamon et al. [8] created five datasets: MDB-melody-synth (for melody tracks), MDB-bass-synth (for bass tracks), MDB-mf0-synth (for monophonic pitched instruments), MDB-stem-synth (for solo stems) and Bach10-mf0-synth (for 10 Bach chorales) available at [17]. Here, we focus on monophonic pitch tracking. So the MDB-stem-synth dataset is appropriate for this task. MDB-stem-synth is published under the terms of the Creative Commons Attribution-NonCommercial 4.0 International License. It can be downloaded without charges after filling a download form.

MDB-stem-synth consists of 230 solo stems (tracks) from different instruments and voices. These stems belong to 85 musical pieces for which between 1 and 17 stems are included. The database consists of audio and annotation files:

- **Audio stems:** The audio stems are created using the methodology of Salamon et al. [8]. They are published with a sampling rate of 44100 Hz. The shortest stems (from MusicDelta_Rock) have a duration of 13 seconds, the longest stems (from MatthewEntwistle_AnEveningWithOliver) have a duration of 1061 seconds (approx. 17.7 minutes). The timbre of the stems is similar to the original stems.

- **Annotation stems:** There is an annotation file for each stem with perfect F0-annotations. The first row of an annotation file contains the time stamps in seconds and the second row contains the F0-value in Hertz. For the annotation files, frames with a hop size of 128 samples (sampling rate of audio: 44100 Hz) were used. This hop size corresponds to approximately 2.902 milliseconds. Unvoiced frames have a frequency value of 0 Hz. From the complete dataset, approximately 44.8 % of the frames are voiced.

The total duration of the audio files in MDB-stem-synth is approximately 15.56 hours. The dataset contains frequencies approximately between 30 Hz and 1220 Hz with more frequencies in the lower range. For the distribution of the frequencies see Figure 2.4.

2.3.2.2 Running Examples

As running examples for the following chapters, two stems are selected: MusicDelta_Rock_STEM_02 and MusicDelta_InTheHalloftheMountainKing_STEM_03. They are both from MusicDelta. MusicDelta was a platform for interactive music experience, but is not online anymore (was available at http://musicdelta.com). It offered information about the fundamentals of music

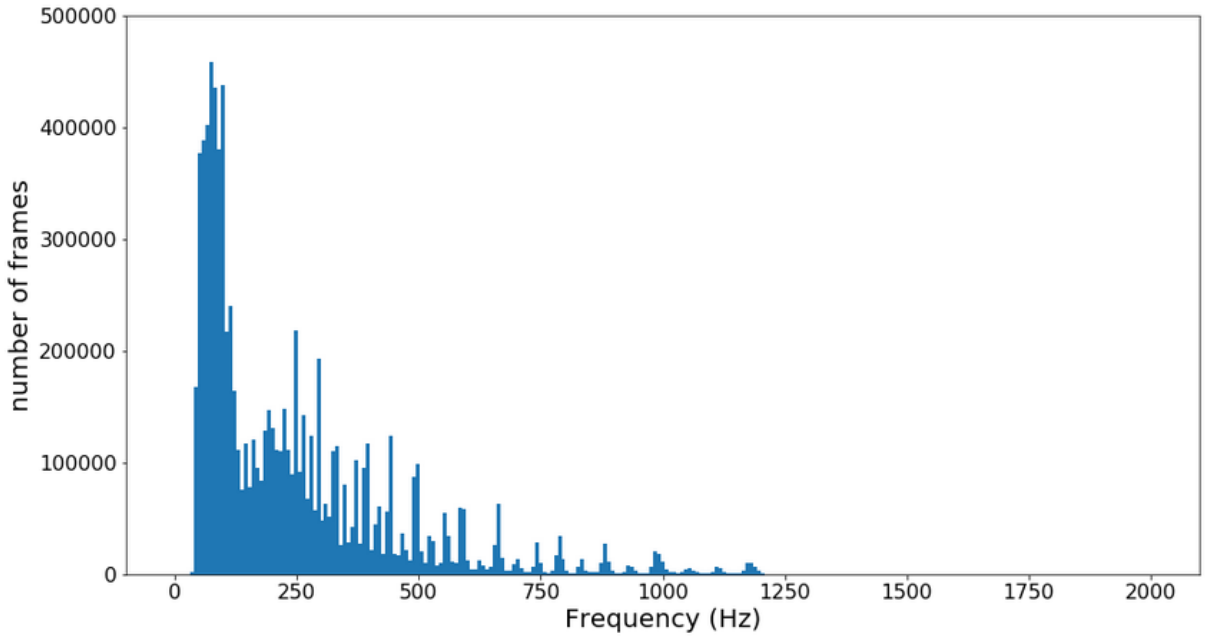


Figure 2.4: Distribution of frequencies present in the MDB-stem-synth dataset.

theory and about music history. Additionally it had an area ("the stage") with 40 musical pieces from classic to pop where some instruments could be selected and listened to separately [18].

The running examples are (metadata from [19]):

- **MusicDelta_Rock_STEM_02**: This is the second stem from a rock song played by an electric bass. Its duration is 13.10 seconds. The annotated frequencies are between 40 Hz and 160 Hz.
- **MusicDelta_InTheHalloftheMountainKing_STEM_03**: This is the third stem of an excerpt of "In The Hall Of The Mountain King" of Peer Gynt composed by Edvard Grieg. The duration of this stem is 17.44 seconds. It is played by a viola. The annotated frequencies are between 180 Hz and 480 Hz.

For plots of the annotated frequencies of both stems, see Figure 2.5. These two examples were chosen, because they are rather short, they are played by different instruments and have different frequency ranges.

2.3.3 Georgian Vocal Music (GVM Scherbaum)

Georgia is a country at the border of Europe and Asia to the east of the Black Sea and has about four million inhabitants [20]. The traditional chants in Georgia differ from Western music in their tonal structure [21]. Scherbaum et al. recorded five Georgian chants which were performed

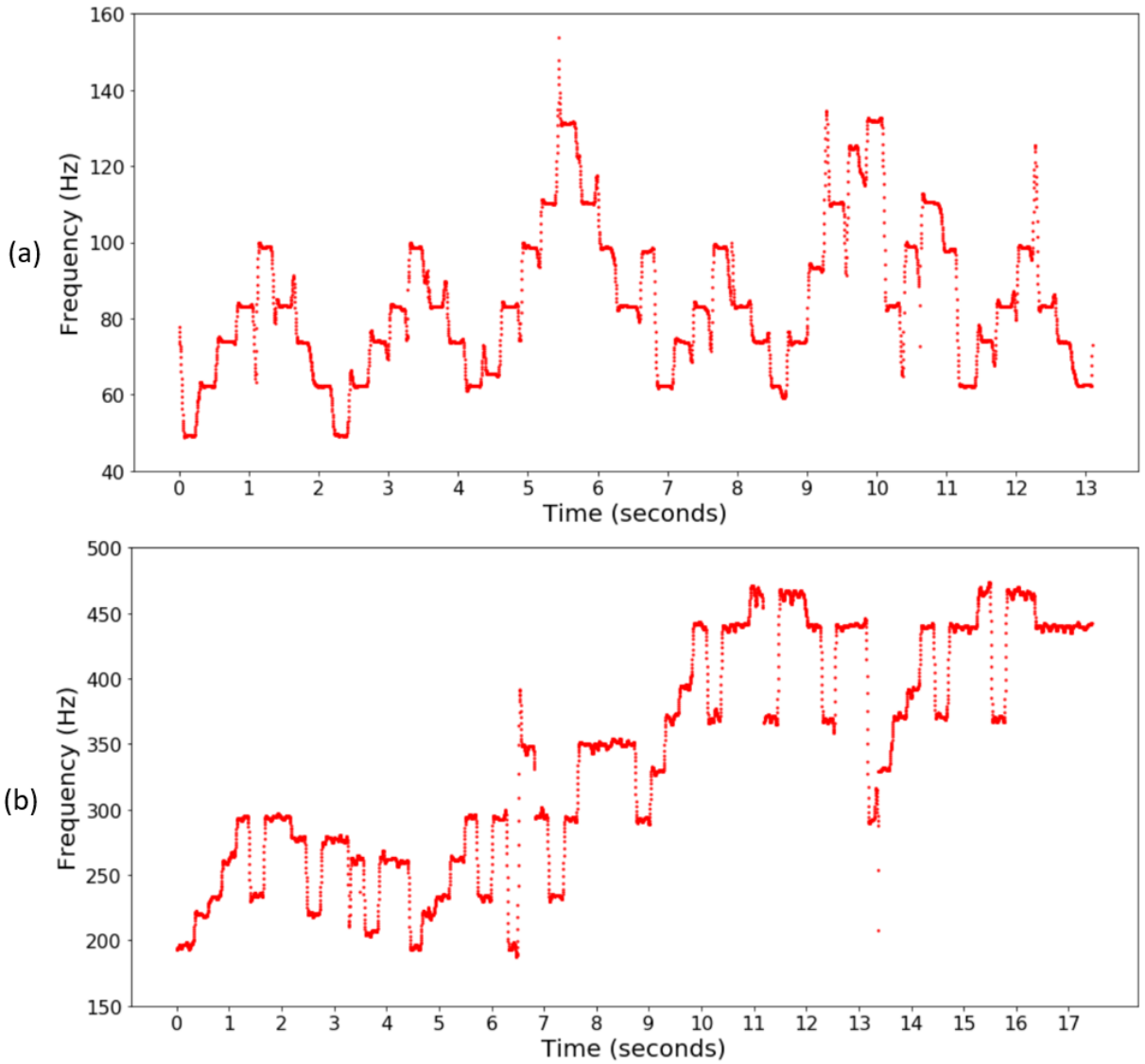


Figure 2.5: Annotated frequencies for running examples from MDB-stem-synth.
(a) MusicDelta_Rock_STEM_02. **(b)** MusicDelta_InTheHalloftheMountainKing_STEM_03.

by three singers [12, 13]. For the recordings, a camera microphone and a room microphone (audio recorder) were used. Additionally, each singer was recorded with a throat microphone and a headset microphone. Throat microphones (also referred to as larynx microphones) are microphones that record vibrations of the larynx. For this purpose, they are directly attached to the singers throat skin. This increases the quality of the resulting recordings, because the signal contains less noise from the environment, e.g. from other singers. Therefore, the recordings from the larynx microphones can be used as ground truth for pitch tracking or voicing estimation.

The example that will be used here is GVM009_BatonebisNanina_Tbilisi_Mzetamze which was recorded on September 18, 2016. The audio files were published under the Creative Commons

Attribution-NonCommercial-NoDerivatives 4.0 International License. The singers are Nana Valishvili, Nino Shvelidze and Nino Makharadze. Each recording has a duration of 120.5 seconds. The single singer recordings (headset and larynx microphone for each of the three singers) have a sampling rate of 48000 Hz and the recordings of the room and the camera microphone have a sampling rate of 44100 Hz. There are no F0-annotations available, but F0-estimations from the larynx microphones can be considered as very accurate.

2.4 Evaluation Measures

For the evaluation of pitch estimation approaches, evaluation measures are necessary. In this chapter, we present measures for evaluating pitch estimations and measures for evaluating voicing estimations.

2.4.1 Pitch Estimation Measures

When performing fundamental frequency estimation, the result is a F0-trajectory. If frequency annotations exist, the estimated frequencies can be compared to the annotated frequencies. For comparing frequencies, we often use the difference in cents. Cent is a unit that is logarithmic in the frequencies. One semitone corresponds to 100 cents and one octave consists of 1200 cents (12 semitones). The difference between two frequencies ω_1 and ω_2 is computed as follows (cf. [9, Section 1.3.2]):

$$\log_2\left(\frac{\omega_1}{\omega_2}\right) \cdot 1200. \tag{2.17}$$

Using the difference between frequencies, we can define evaluation metrics for pitch estimation tasks. Often, the raw pitch accuracy (RPA) and the raw chroma accuracy (RCA) are computed. These metrics were first introduced in [22] and were used in the melody competition of the Music Information Retrieval Evaluation Exchange (MIREX) [23] in 2005. For the computation of the raw pitch/chroma accuracy, a threshold can be passed as parameter. This threshold is the tolerance in cents. If the difference between the frequency of the reference and the frequency of the estimate is lower than this threshold, the estimated frequency is considered as correct. The number of correct frames is then divided by the number of frames that are voiced according to the reference.

2.4.1.1 Raw Pitch Accuracy (RPA)

The raw pitch accuracy (RPA) computes the proportion of frames for which the pitch estimate is correct (depending on a threshold ϵ) compared to the total number of voiced frames in the annotation.

We assume that an estimated frequency trajectory $\eta : [1 : N] \to \mathbb{R} \cup \{*\}$ and an annotated (ground truth) frequency trajectory $\eta^* : [1 : N] \to \mathbb{R} \cup \{*\}$ are given for a discrete signal x. It should be noted that the sampling rate of the signal x is not necessary equal to the sampling rate for the frequency trajectory. E.g. if a signal x with a sampling rate of 16000 Hz is given, the time instances of x have a distance of 0.0625 milliseconds. Although, the frequency trajectory can map time frames with a distance of 10 milliseconds (hop size) to frequency values. In the following, we use the time index n for the frames of the frequency trajectory. So for time instance n, we can compute the difference between the estimated and the annotated frequencies:

$$\Delta_{\text{cents}}(n) := \left| \log_2 \left(\frac{\eta(n)}{\eta^*(n)} \right) \cdot 1200 \right|. \tag{2.18}$$

For the computation of the raw pitch accuracy, a threshold ϵ has to be fixed. This threshold determines how large the difference in cents can be between the estimation and the annotation so that the estimation is still considered as correct. For example if the annotated frequency is 440 Hz and the estimated frequency is 445 Hz, the difference in cents between the frequencies is approximately 19.56 cents. If a threshold of 10 cents is fixed, the pitch for this frame is considered to be wrong. With a threshold of 25 cents, the frame would be counted as correct. Depending on the threshold $\epsilon \in \mathbb{N}$ in cents, a threshold function $\mathcal{T}_{\text{pitch}}$ can be defined:

$$\mathcal{T}_{\text{pitch}}(n) := \begin{cases} 1 & \text{if } \Delta_{\text{cents}}(n) < \epsilon \text{ and } \eta(n) \neq * \text{ and } \eta^*(n) \neq * \\ 0 & \text{else} \end{cases}. \tag{2.19}$$

The threshold function $\mathcal{T}_{\text{pitch}}$ returns 1 if the pitch estimation is correct for time instance n depending on the threshold ϵ. Since only the frames that are voiced in the annotation are considered for the raw pitch accuracy, we define a voicing function:

$$v_\eta(n) := \begin{cases} 1 & \text{if } \eta(n) \neq * \\ 0 & \text{else} \end{cases}. \tag{2.20}$$

Using the definition of the threshold function and the voicing function, we define the raw pitch accuracy $\text{Acc}_{\text{pitch}}$ for the trajectories η (estimation) and η^* (annotation):

$$\text{Acc}_{\text{pitch}} := \frac{\sum_n v_{\eta^*}(n) \cdot \mathcal{T}_{\text{pitch}}(n)}{\sum_n v_{\eta^*}(n)}. \tag{2.21}$$

So we sum over all time instances n and count the frames that are voiced and for which the estimation is correct. This number is divided by the number of voiced frames in the annotation.

A small example for the computation of the raw pitch accuracy is shown in Table 2.1.

time index	n_1	n_2	n_3	n_4
$\eta^*(n)$	440 Hz	300 Hz	220 Hz	$*$ (unvoiced)
$\eta(n)$	446 Hz	296 Hz	442 Hz	300 Hz
$\Delta_{\text{cents}}(n)$	23.4 cents	23.2 cents	1207.9 cents	-
$\mathcal{T}_{\text{pitch}}(n)$ with $\epsilon = 25$ cents	1	1	0	0

Table 2.1: Example for the computation of the raw pitch accuracy. If the difference between the annotated and the estimated frequency is below 25 cents, the estimation is considered as correct. Here, 2 of 3 voiced frames are correct, hence the raw pitch accuracy is 2/3.

2.4.1.2 Raw Chroma Accuracy (RCA)

For the raw chroma accuracy, only the chroma of a pitch value is taken into account. A chroma is an attribute of a pitch value without consideration of the octave. There are twelve chroma values corresponding to the twelve note names “C”, “C sharp”, ..., “B” [9]. So e.g. a tone with a frequency of 440 Hz (A4) and a tone with a frequency of 880 Hz (A5) have the same chroma value. The raw chroma accuracy only focuses on the chroma and therefore does not recognize octave errors (right chroma predicted, but wrong octave). So the difference between the RCA and the RPA is an indication for the amount of octave errors.

As for the raw pitch accuracy, we assume that there is an estimated frequency trajectory $\eta : [1 : N] \to \mathbb{R} \cup \{*\}$ and an annotated (ground truth) frequency trajectory $\eta^* : [1 : N] \to \mathbb{R} \cup \{*\}$ for the signal x.

We use the function $\Delta_{\text{cents}}(n)$ for computing the difference between the estimated frequency and the annotated frequency at time instance n from Equation 2.18. With this formula, we can define a modified threshold function $\mathcal{T}_{\text{chroma}}(n)$:

$$\mathcal{T}_{\text{chroma}}(n) = \begin{cases} 1 & \text{if } (\Delta_{\text{cents}}(n) \bmod 1200) < \epsilon \text{ and } \eta(n) \neq * \text{ and } \eta^*(n) \neq * \\ 1 & \text{if } 1200 - (\Delta_{\text{cents}}(n) \bmod 1200) < \epsilon \text{ and } \eta(n) \neq * \text{ and } \eta^*(n) \neq * \\ 0 & \text{else} \end{cases} \tag{2.22}$$

with mod being the modulo operation. With the modulo operation, the difference in cents is mapped to values between 0 and 1199 (one octave). Using for example a threshold of 25 cents, the first case of the equation applies if the frequencies have a difference of e.g. 1220 cents (1220 mod 1200 = 20). The second case is necessary if the difference is e.g. 1180 cents (1180 mod 1200 = 1180 $> \epsilon$, but 1200 - (1180 mod 1200) = 20 $< \epsilon$). With this, we can define the raw chroma accuracy similar to the raw pitch accuracy

$$\mathrm{Acc}_{\mathrm{chroma}} := \frac{\sum_n v_{\eta^*}(n) \cdot \mathcal{T}_{\mathrm{chroma}}(n)}{\sum_n v_{\eta^*}(n)}. \tag{2.23}$$

for frequency trajectories η (estimation) and η^* (annotation).

For the same example as before, the computation of the raw chroma accuracy is shown in Table 2.2. The raw chroma accuracy for this example is higher than the raw pitch accuracy, because the estimation of 442 Hz instead of 220 Hz is considered as correct for the raw chroma accuracy.

time index	n_1	n_2	n_3	n_4
$\eta^*(n)$	440 Hz	300 Hz	220 Hz	$*$ (unvoiced)
$\eta(n)$	446 Hz	296 Hz	442 Hz	300 Hz
$\Delta_{\mathrm{cents}}(n)$	23.4 cents	23.2 cents	1207.9 cents	-
$\mathcal{T}_{\mathrm{chroma}}(n)$ with $\epsilon = 25$ cents	1	1	1	0

Table 2.2: Example for the computation of the raw chroma accuracy. If the difference between the chroma of the annotated and the estimated frequency is below 25 cents, the estimation is considered as correct. Here, 3 of 3 voiced frames are correct, hence the raw chroma accuracy is 1.

2.4.2 Voicing Estimation Measures

Many pitch estimation algorithms also have the possibility to predict that a frame is unvoiced. This is the case, when there is no melody present in this frame. There are different common methods for dealing with unvoiced frames:

- **0 Hertz:** If a frame is recognized to be unvoiced by the algorithm, it returns a frequency value of 0 Hz.
- **Negative Frequency**: If it is recognized that a frame is likely to be unvoiced, a frequency value is computed which would be the output if the frame is nevertheless voiced. The negative of the value is the output. So if the frame is predicted to be unvoiced, but if it was still voiced, the estimation would be 440 Hz, the output would be -440 Hz.

- **Confidence**: For each frame, an estimated frequency value is returned. Additionally, for each estimation, a confidence value is computed. The confidence value indicates how likely it is that the frame is voiced.

When predicting whether a frame is voiced or unvoiced, the estimated result can be compared to a voicing annotation. For this, voicing estimation measures have to be defined. Therefore, the true positives (TP, estimated as voiced and voiced in the annotation), true negatives (TN, estimated as unvoiced and unvoiced in the annotation), false positives (FP, estimated as voiced but unvoiced in the annotation) and the false negatives (FN, estimated as unvoiced but voiced in the annotation) can be counted. From those values, two commonly used metrics can be derived: recall and precision. The recall describes how many frames are estimated correctly from those that were voiced in the annotation:

$$\text{Recall} := \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{2.24}$$

The precision describes how many frames are estimated correctly from those that were estimated as voiced:

$$\text{Precision} := \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{2.25}$$

Recall and precision can be summarized as F-measure:

$$\text{F-measure} := 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{2.26}$$

The F-measure can be used as evaluation metric for voicing estimation. Its values range between 0 (worst estimation) and 1 (perfect estimation).

Chapter 3

Model-Based Fundamental Frequency Estimation

In this chapter, we present four model-based approaches for fundamental frequency estimation. In Section 3.1, we start with YIN, an approach for F0-estimation which uses the autocorrelation method. Based on YIN, pYIN was developed as a probabilistic version of YIN. It is explained in Section 3.2. This approach computes multiple pitch candidates which are input to a hidden Markov model to compute the final pitch estimates. Another approach is SWIPE, presented in Section 3.3. It uses the distance between peaks and valleys of a signal. SWIPE as well as YIN and pYIN are developed for F0-estimation for monophonic audio. For polyphonic audio, Melodia was developed. It is explained in Section 3.4. Melodia is based on the computation of a salience function.

3.1 YIN

One approach for fundamental frequency estimation is YIN [3], which is based on the autocorrelation method. We first give some general information about YIN, then present the voicing estimation of YIN and finally apply YIN to the running examples.

3.1.1 General Information

YIN [3] is a fundamental frequency estimator which can be used for speech and music. It was developed by de Cheveigné and Kawahara in 2001/2002. The name YIN comes from the interconnected forces “yin” and “yang” in Chinese philosophy. It is an approach for pitch estimation in the time-domain and is based on the autocorrelation method. The autocorrelation

function computes the similarity of a signal $x : \mathbb{Z} \to \mathbb{R}$ to itself at a time instance n with a lag τ and a window size N:

$$r_{\mathrm{n}}(\tau) = \sum_{j=n}^{n+N} x(j)x(j+\tau) \tag{3.1}$$

If the similarity is high, the signal is likely to have a period τ. This function has high error rates, because the search range for lags has to be fixed and the method is prone to picking a lag of zero or a multiple of the correct lag. If the amplitude increases in the signal, this equation is prone to picking integer multiples of the correct lag, because they lead to a higher r_{n} value. This can be avoided by using a difference function d_{n} instead:

$$d_{\mathrm{n}}(\tau) = r_{\mathrm{n}}(0) + r_{\mathrm{n}+\tau}(0) - 2r_{\mathrm{n}}(\tau) \tag{3.2}$$

The result of this function can be improved by applying the following error-reduction steps (cf. [3]):

- Cumulative mean normalized difference function d'_{n}: With this function, a lower limit for the search range is no longer necessary. It improves the difference function 3.2 by returning 1 for a lag of zero and the average over shorter-lag values for all other lags.
- Absolute threshold: By setting an absolute threshold s for the difference function and picking the smallest lag value below the threshold, subharmonic errors can be reduced.
- Parabolic interpolation: A parabola is fit to a local minimum of the cumulative mean normalized difference function. This can prevent errors caused by an unfavorable combination of the sampling period and the signal period (if the signal period is not a multiple of the sampling period).
- Best local estimate: When searching for a good lag estimate at time index t, a rather broad search range is selected first. After finding a good estimate for the lag, lags in the direct neighborhood are explored to possibly identify an even more accurate lag.

These steps result in an estimation for a lag. From the lag τ the estimated fundamental frequency can be computed: $\omega = \frac{1}{\tau}$.

3.1.2 Voicing in YIN

Furthermore, voicing estimates can be computed using YIN. A voicing estimate indicates how likely it is according to YIN that a frame is voiced (contains a pitch). YIN computes multiple values related to voicing: the fundamental power (energy estimate of the estimated fundamental), the aperiodic power and the total power. Additionally, there is the aperiodic measure, which is

proportional to the ratio of the aperiodic power to the total power. For voicing estimation, the fundamental power or the aperiodic measure can be used. The aperiodic power and the total power alone are not usable for voicing estimation. Here, we decide to use the aperiodic measure. If the aperiodic measure is low, this indicates that there is not much aperiodicity present and therefore the frame is likely to be voiced. In contrast, if the aperiodic measure is high, the frame is likely to be unvoiced. So the aperiodic measure can be used as indication for a voicing estimate. However, the aperiodic measure is not normalized to values between 0 and 1. This makes a clear decision whether a frame is voiced or not more complicated, since it is unclear which value of the aperiodic measure to use as boundary for the decision for voiced/unvoiced. Using an audio example from the Georgian vocal music dataset [12, 13] (see Section 2.3.3), we examined different methods for fixing a threshold experimentally. Three possible methods for fixing a threshold are: (1) The mean of the aperiodic measure, (2) the median of the aperiodic measure and (3) the value in the middle between the maximum and the minimum of the aperiodic measure $\frac{max-min}{2}$. We examined the three methods using the recording of Batonebis Nanina (larynx microphone, singer 1) of the Georgian vocal music dataset (see Section 2.3.3). They lead to thresholds of 0.1467 (method 1), 0.0034 (method 2) and 0.8559 (method 3). By looking at the plots and listening to the recording, we selected method 1 using the mean of the aperiodic measure. According to this result, we assume frames with an aperiodic measure below 0.1467 to be voiced and frames with an aperiodic measure above or equal to 0.1467 as unvoiced. For a plot of the recording with the fundamental power, the aperiodic measure and the selected threshold, see Figure 3.1.

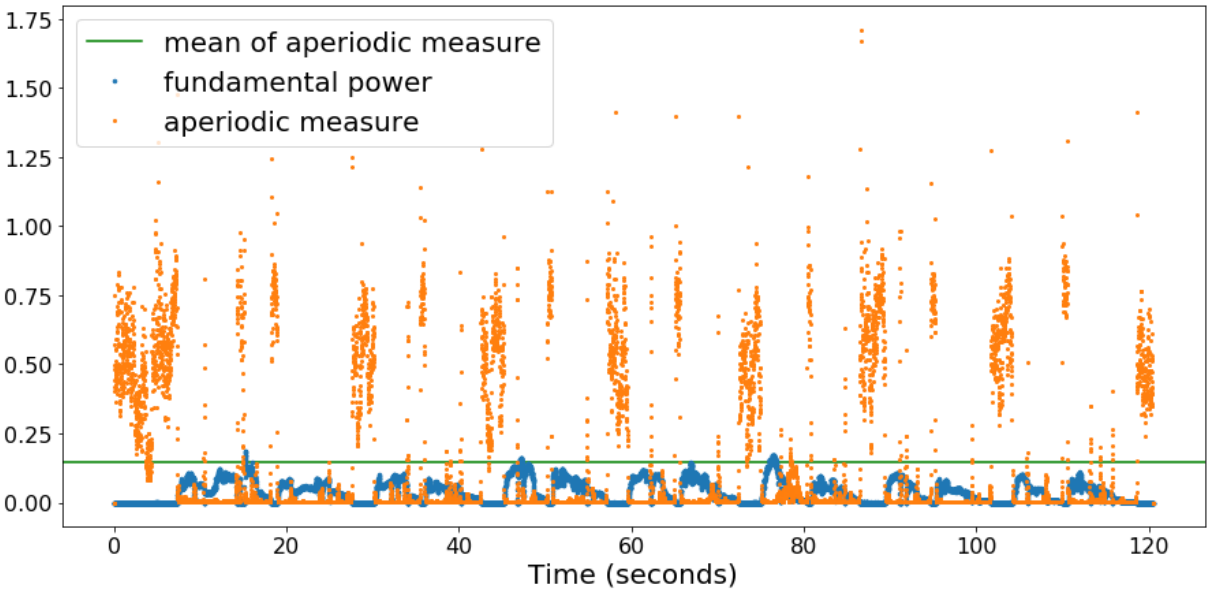


Figure 3.1: Fundamental power and aperiodic measure of YIN for a recording from the Georgian vocal music dataset [12, 13] (larynx microphone, singer 1). As threshold for the voicing estimation, we use the mean of the aperiodic measure (0.1467).

3.1.3 Application of YIN to Running Examples

A MATLAB implementation of YIN was provided by Sebastian Rosenzweig (July 2018) and Peter Grosche (February 2009). We applied YIN to the artificial running examples from Section 2.3.1 and to the examples from the MDB-stem-synth dataset [8] introduced in Section 2.3.2.2. For the different datasets, we used different configurations for YIN. For the examples from the MDB-stem-synth dataset (sampling rate 44100 Hz), we used a hop size of 512 samples and a window size of 2048 samples. The sampling rate of the artificial examples is only 16000 Hz, therefore, we use a hop size of 256 samples and a window size of 1024 samples. The minimum F0-frequency that can be detected is set to 30 Hz and the maximum F0-frequency is 2000 Hz for all running examples. The results of applying YIN to the running examples are shown in Figure 3.2. There the estimated frequencies of YIN and the annotated frequencies are plotted.

For the first artificial signal consisting of sinusoids, it is remarkable that YIN predicts a frequency of 1777.78 Hz during the pause. This can be explained by the algorithm of YIN. At YIN, different lags are examined and the optimal lag is chosen. These lags are determined by the minimum and the maximum frequency (parameter set by the user, here 30 Hz and 2000 Hz). The smallest possible lag with these settings is 9. If the smallest lag was 8, the signal would be examined for oscillations which repeat after 8 samples. Such a signal would oscillate with approximately 2000 Hz (16000/8) leading to a frequency that is not below the maximum frequency. In contrast, if YIN starts with a lag of 9, this results in possible output frequencies below or equal 1777.78 Hz (16000/9). So the lowest possible lag is 9 and the highest frequency that can be detected is 1777.78 Hz. Since the lowest lag is chosen, in this case the output is a frequency of 1777.78 Hz.

For the second artificial example consisting of a signal with a second partial, YIN outputs the lower frequency for more than 8 seconds and then switches to the higher frequency.

Next, YIN is applied to the chirp signal. The signal stops at a frequency of 2000 Hz and YIN is configured to examine frequencies until an upper bound of 2000 Hz. However, YIN correctly predicts the frequencies until 1880 Hz, but some errors occur for higher frequencies.

For the examples from the MDB-stem-synth dataset, the estimations are almost similar to the annotations. At the second example, MusicDelta_InTheHalloftheMountainKing_STEM_03, there is an octave error at seconds 6.4 to 6.5 (one octave too high, see Figure 3.3 for details). At the beginning and the end of all recordings, there are some estimates of 0 Hz. This is also the case at the beginning and the end of the pause in the first artificial example.

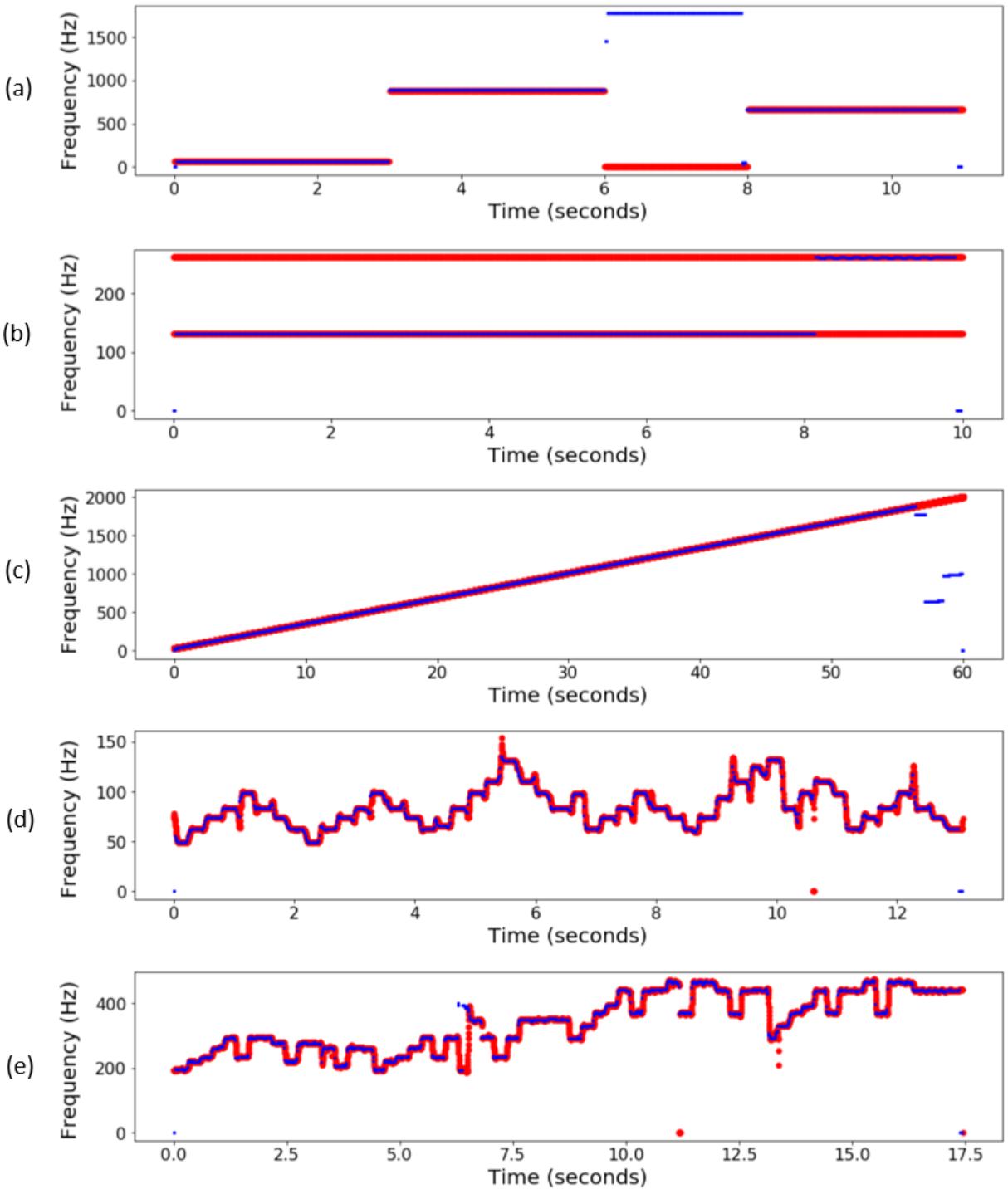


Figure 3.2: Frequencies estimated by YIN (blue) and frequency annotations (red) for running examples. **(a)** Sinusoids. **(b)** Signal with Harmonics. **(c)** Chirp signal.
(d) MusicDelta_Rock_STEM_02. **(e)** MusicDelta_InTheHalloftheMountainKing_STEM_03.

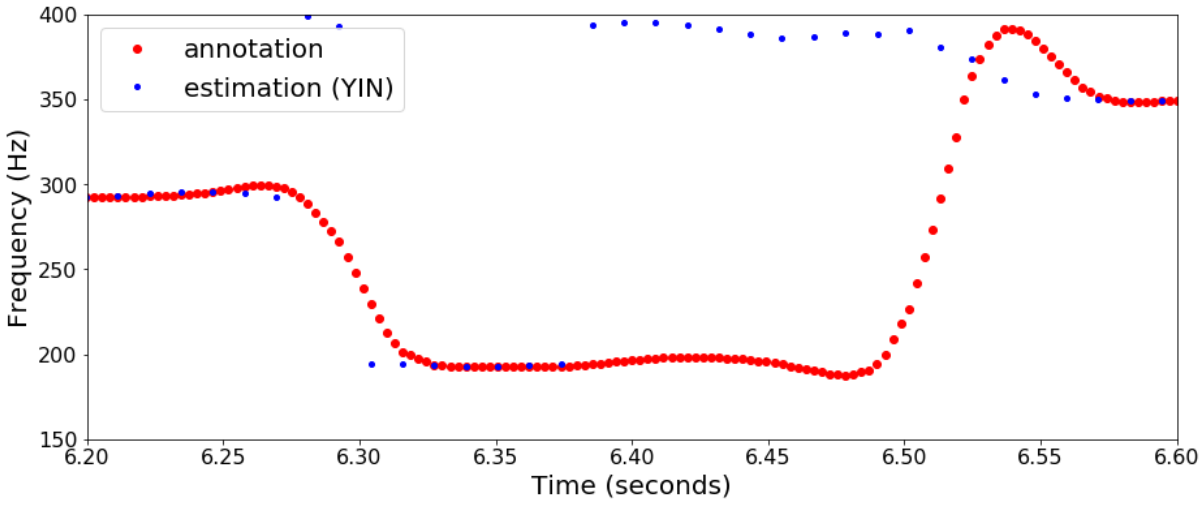


Figure 3.3: Annotated frequencies and frequencies estimated using YIN for MusicDelta_InTheHalloftheMountainKing_STEM_03 with an octave error.

3.2 pYIN

In this section, we present pYIN [4], a pitch estimation approach based on YIN [3]. We start with some general information about pYIN, explain how pYIN generates F0-Candidates and how a pitch trajectory is derived using a hidden Markov model. Then, we apply pYIN to the running examples.

3.2.1 General Information

In this section, we present pYIN [4] (Probabilistic YIN), a time-based approach for fundamental frequency estimation developed by Mauch and Dixon in 2014. It is based on YIN [3] (Section 3.1) and extends the previous algorithm by providing multiple pitch estimates per frame. From these pitch candidates, the estimated fundamental frequencies can be derived using a hidden Markov model.

The first steps of pYIN are the same as of YIN. The algorithm starts with the difference function defined in Equation 3.2. This function is then replaced by the cumulative mean normalized difference function d'_n. The next step in YIN would be to set an absolute threshold s and picking the smallest lag value τ below the threshold s: $d'_n(\tau) < s$. The estimated period of the original YIN after this step is referred to as $Y(x(n), s)$ where $x : \mathbb{Z} \to \mathbb{R}$ is the signal. For the next steps, pYIN differs from YIN. In pYIN, first some F0-candidates are generated and from those, a pitch trajectory is computed using HMM-based pitch tracking.

3.2.2 F0-Candidates

The threshold s affects the result of YIN, though its value is variable and difficult to choose correctly. Instead of using a fixed threshold, pYIN uses a beta distribution of parameters S given by $P(s(i))$ with possible thresholds $s(1), s(2)..., s(N)$. If $d'_{\mathrm{n}}(\tau)$ is above the threshold for all lags τ, YIN would use the smallest $d'_{\mathrm{n}}(\tau)$. The probability of YIN choosing the absolute minimum is estimated using a prior probability p_{a} (can be set to 0.01). With this, a formula can be constructed to compute for each possible period τ how likely it is that τ is the estimated fundamental period τ_0 when using YIN. This is computed by looping over all possible thresholds $s(i)$. For each possible threshold, the probability is increased, if YIN would pick this lag given the threshold $s(i)$. In this case, the probability increases by the probability of the threshold $P(s(i))$, if the difference function is below the threshold and otherwise the probability increases by $p_{\mathrm{a}} \cdot P(s(i))$:

$$P(\tau = \tau_0 | S, x(n)) = \sum_{i=1}^{N} a(s(i), \tau) P(s(i)) [Y(x(n), s(i)) = \tau] \tag{3.3}$$

where $[\cdot]$ is one if the expression is true and 0 otherwise and

$$a(s(i), \tau) = \begin{cases} 1 & \text{if } d'_{\mathrm{n}}(\tau) < s(i) \\ p_{\mathrm{a}} & \text{otherwise} \end{cases}. \tag{3.4}$$

The pitch candidates are the frequencies belonging to the periods τ which have a positive probability. Among these candidates, there is also the output of the original YIN, so no information is lost using pYIN compared to YIN. All pitch candidates are used for a HMM-based pitch tracking.

3.2.3 HMM-based Pitch Tracking

In order to derive a pitch trajectory from the F0-Candidates, a hidden Markov model (HMM) is used. For this, vectors are derived from the frequency candidates and their probabilities from the previous step. Each vector describes the probabilities of the pitch candidates in one frame. In the vectors, each entry describes the probability of a pitch value corresponding to the index of the entry (55 Hz to 880 Hz, 10 cents intervals). The dimension of each vector is then doubled to also reflect a voicing/no voicing indication. The probability that a frame is unvoiced can be computed using Equation 3.3. From this, the observation probabilities can be computed using the results of the previous step. The transition probabilities should then lead to pitch trajectories with small pitch changes and few voicing changes. Compared to YIN, pYIN is found to have a higher precision and recall due to the generation of pitch candidates.

3.2.4 Application of pYIN to Running Examples

For the evaluation of pYIN, we used the pYIN Vamp plugin for Sonic Annotator developed by Mauch et al. [4]. As distribution $S(i)$, the Beta distribution with a mean of 0.15 is used. See Figure 3.4 for an application of pYIN to the running examples. The result of the application of pYIN to the example consisting of sinusoidals shows that the estimations are similar to the annotated frequencies. It is remarkable that pYIN has the possibility to predict that a frame is unvoiced. So during the pause, there are no pitch estimates. The application of pYIN to the example consisting of a signal with the second partial shows different results compared to YIN. YIN predicted the lower frequency for more than 8 seconds whereas pYIN predicts the lower frequency for less than 6 seconds. During the second half of the audio (seconds 5 to 10), the amplitude of the second harmonic is higher than the amplitude of the fundamental frequency. The estimations for the chirp signal are very accurate.

For the first example from the MDB-stem-synth dataset [8], MusicDelta_Rock_STEM_02, the estimates are similar to the annotated frequencies, except for some frames where too high frequencies are predicted. The estimations for MusicDelta_InTheHalloftheMountainKing_STEM_03 are not very accurate. For some frames, the estimations are similar to the annotations, but there are many octave errors and also errors with different distances. Most of the wrong estimates are due to too high estimates. One possible explanation is that this recording was performed by a viola. So the special timbre of a stringed instrument could lead to the wrong (too high) estimates.

3.3 SWIPE

Another approach for fundamental frequency estimation is SWIPE [5, 6] which is based on a sawtooth waveform. In this section, we give some general information about SWIPE and then apply an improved version of SWIPE, SWIPE′ to the running examples.

3.3.1 General Information

SWIPE [5, 6] is short for "sawtooth waveform inspired pitch estimator" and was developed by Arturo Camacho and John G. Harris in 2008. The idea is that the fundamental frequency can be defined as a frequency whose harmonics are strongly represented in the signal compared to the frequencies between the harmonics. This can be examined by using the Fourier transform X of a signal x. So e.g. $|X(k)|, |X(2k)|, |X(3k)|$ are high, but the valleys $|X(0.5k)|, |X(1.5k)|, |X(2.5k)|$ are low if the frequency $F_{\text{coef}}(k)$ is a good estimate for the fundamental frequency. Using this idea, the average peak-to-valley distance (APVD) for a frequency can be computed by examining

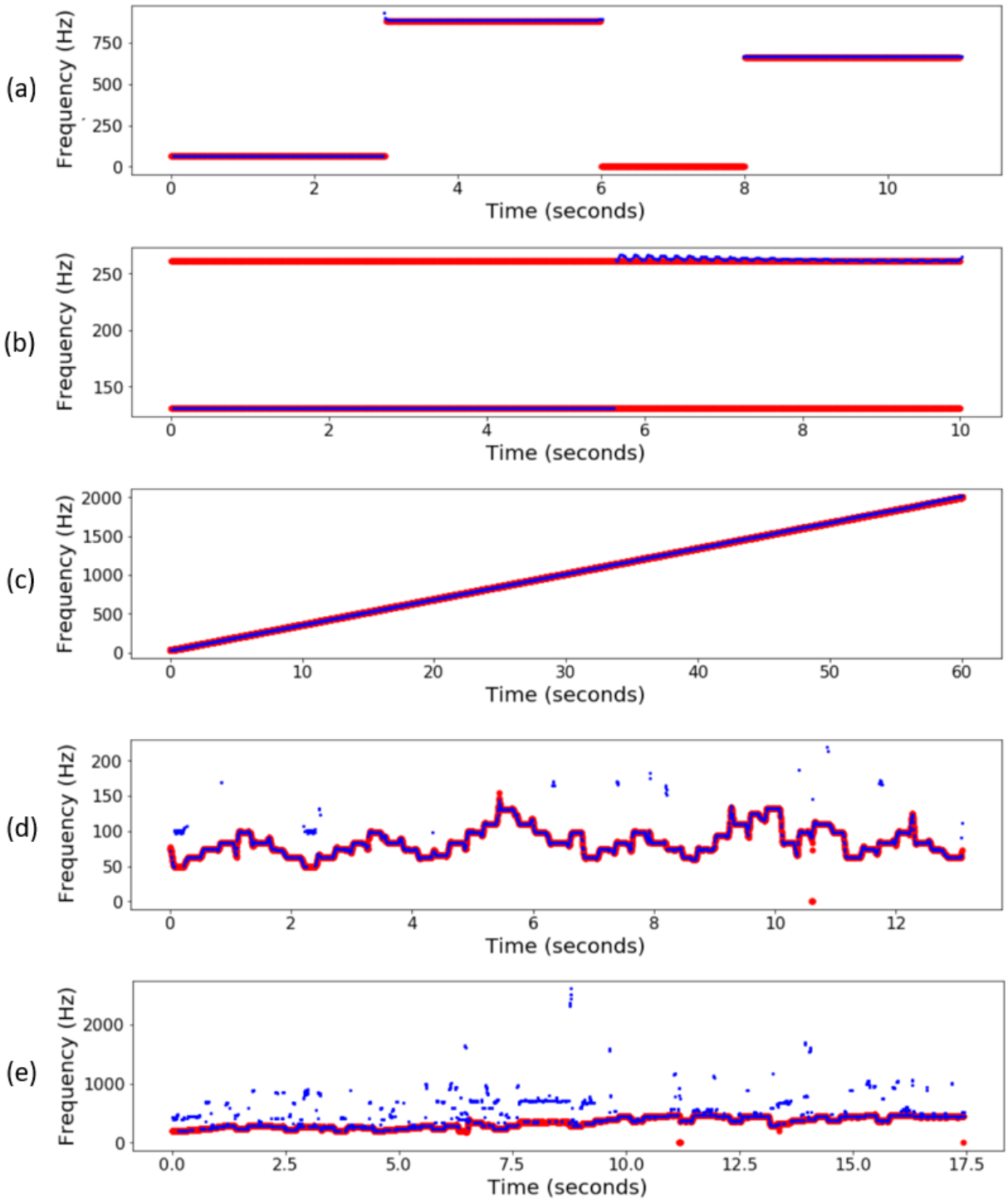


Figure 3.4: Frequencies estimated by pYIN (blue) and frequency annotations (red) for running examples. **(a)** Sinusoids. **(b)** Signal with Harmonics. **(c)** Chirp signal.
(d) MusicDelta_Rock_STEM_02. **(e)** MusicDelta_InTheHalloftheMountainKing_STEM_03.

the difference between the Fourier transform of a possible peak and the Fourier transform of the neighboring valleys. To extend this for the first n harmonics, the global APVD can be computed by averaging over the APVD for the n harmonics. The pitch estimation is then the frequency with the highest average peak-to-valley distance.

This approach is then further improved by the following steps (names from the original paper [5]):

- **Warping of the spectrum**: Instead of the raw spectrum of the signal, the square root of the spectrum is used.
- **Weighting the amplitude of the harmonics:** In the original formula, all harmonics have the same influence to the APVD. The idea is to use decaying weights instead. The best results are achieved for a decay of $\frac{1}{\sqrt{k}}$ for a harmonic k.
- **Blurring of the harmonics:** Instead of considering only exact multiples of the frequency (exact harmonics), also the neighborhood of the harmonics was examined.

Next to SWIPE, the authors also developed SWIPE′, a further improved version of SWIPE. It prevents the estimation of subharmonics instead of the expected fundamental frequency.

3.3.2 Application of SWIPE′ to Running Examples

An implementation of SWIPE′ is available on github [24]. It was implemented by Kyle Gorman and written in C. Here, we used version 1.5. As frequency range, we selected a minimum frequency of 20 Hz and a maximum frequency of 2500 Hz. The hop size is set to 0.01 seconds. We applied SWIPE′ to the artificial running examples and the examples from MDB-stem-synth [8]. For the resulting pitch estimations, see Figure 3.5. The estimations for all examples are very accurate and similar to the annotations. For the first example consisting of pure sinusoids, SWIPE′ predicts an undefined number (nan) during the pause. So no values are included in the plot for the pause. For the second artificial example (signal with second harmonic), it is remarkable that SWIPE′ predicts the lower frequency for more than 9 seconds, although the amplitude of the second harmonic is higher. For the remaining examples, the estimations of SWIPE′ are similar to the annotations.

3.4 Melodia

In this section, we present Melodia [7]. This is an approach for F0-estimation which can be used for polyphonic music. We start with some general information about Melodia and then apply Melodia to the running examples.

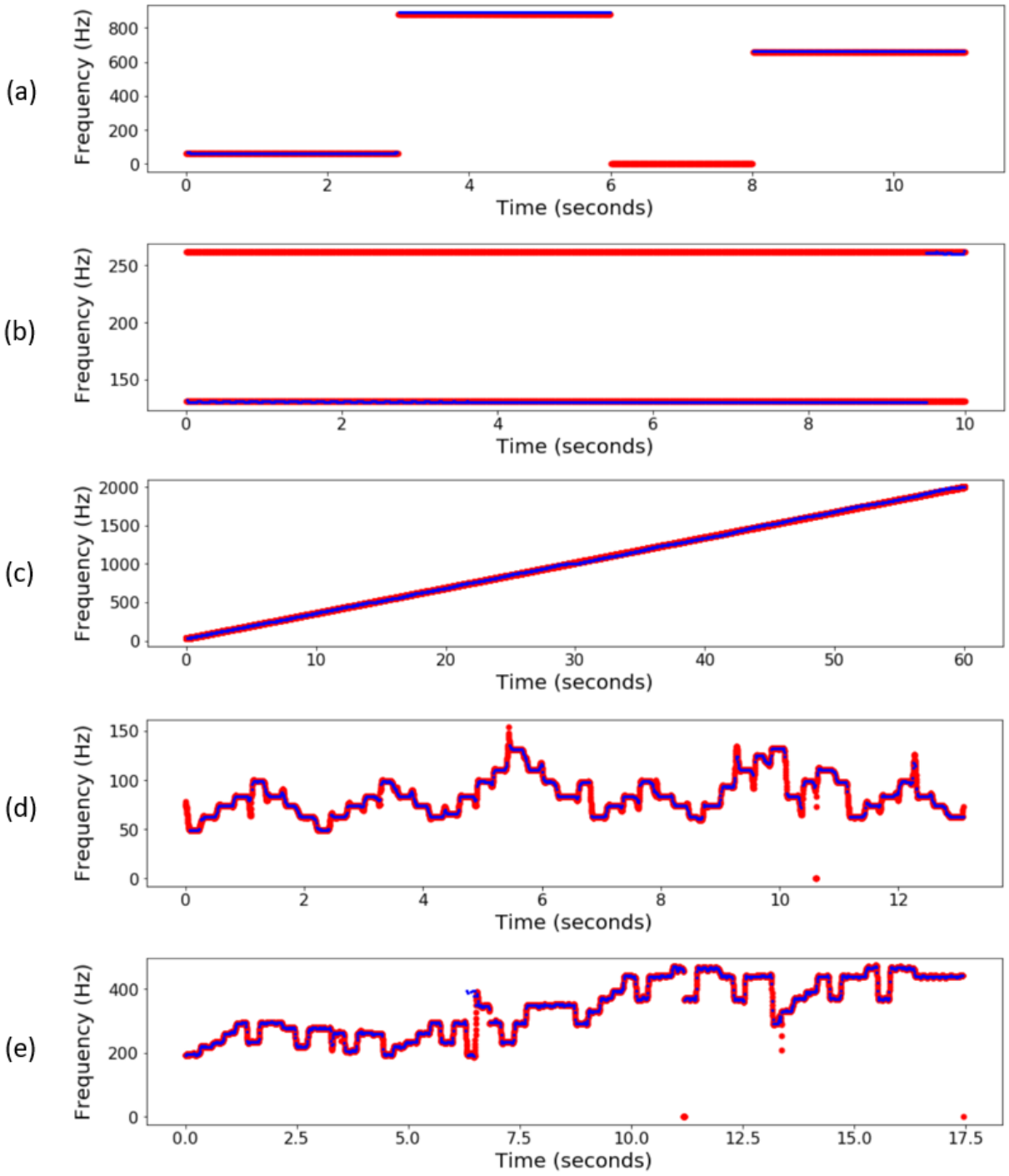


Figure 3.5: Frequencies estimated by SWIPE′ (blue) and frequency annotations (red) for running examples. **(a)** Sinusoids. **(b)** Signal with Harmonics. **(c)** Chirp signal. **(d)** MusicDelta_Rock_STEM_02. **(e)** MusicDelta_InTheHalloftheMountainKing_STEM_03.

3.4.1 General Information

Melodia [7] was developed by Salamon and Gómez in 2010. It is a pitch estimation approach for polyphonic music. The objective is to identify the fundamental frequencies belonging to the main melody. For music with multiple simultaneously occurring pitches, the main melody is "the 'essence' of that music when heard in comparison" [22]. Salamon and Gómez propose a system that first identifies melody pitch candidates and then groups them into pitch contours. The algorithm consists of the four steps "Sinusoid extraction", "Salience function", "Pitch contour creation" and "Melody selection":

- **Sinusoid Extraction:** First, a filter is applied in order to enhance frequencies which are common for melody pitches (mid-band frequencies). Then, a short-time Fourier transform is applied. Next, the peaks of the magnitude spectrum are identified for all frames. To improve the frequency resolution for low frequencies, the instantaneous frequency (see Section 2.2.2.1) is computed.
- **Salience Function Computation:** In order to compute a salience function, the frequencies are assigned to bins. The bins cover frequencies between 55 Hertz and 1760 Hertz with a 10 cent frequency resolution. For each frame, harmonic summation is applied to the spectral peaks.
- **Creating Pitch Contours:** First, the peaks are further examined and peak filtering is applied. For this, peaks are compared to the highest peak in the same frame. If the salience of a peak is too low compared to the salience of the highest peak, the peak is discarded. Also peaks with a salience that is too low compared to all other peaks are filtered out in order to remove peaks at possibly unvoiced frames.
  For peak streaming, the highest peak is selected. The pitch contour is then created by repeatedly searching for a neighboring peak at the next time frame with a distance of at most 80 cents. For short time intervals, also the choice of previously removed peaks is possible (masked salience). The same strategy is used to extend the salience to frames before the initial maximum peak. The complete peak streaming stops, if all peaks are processed.
  For the created pitch contours, some characteristics are computed (e.g. pitch mean, contour mean salience, length, vibrato presence,...).

- **Melody Selection:** The first step of melody selection is the voicing detection. This is done using the average contour mean salience. Then octave errors and pitch outliers should be avoided by comparing the distance of contours. If two contours are equal (with octave distance), characteristics from the previous step are used to choose one contour. Contours that have a distance of more than one octave to the melody pitch mean are also filtered out (pitch outliers). To retrieve a final melody output, only one pitch can be selected per time frame. If contours overlap, the contour with the highest total salience is selected.

3.4.2 Application of Melodia to Running Examples

We evaluated the running examples using Melodia [7]. As implementation we used the MELODIA - Melody Extraction Vamp plugin for Sonic Annotator developed by Salamon and Gómez. As output type, we chose "Melody". This returns the fundamental frequencies of the main melody in Hertz. Melodia also performs voicing detection. If a frame is estimated to be unvoiced, a negative frequency is predicted (negative of the frequency that would be predicted if the frame is nevertheless voiced). The estimated fundamental frequencies are shown in Figure 3.7. For the first artificial example (sinusoids), Melodia correctly predicts the second and the third tone, which have frequencies of 880 Hz and 659.255 Hz. For the first part (65.406 Hz), Melodia predicts a frequency of -880 Hz. This could be due to the enhancement of mid-band frequencies, so this frequency could be too low.

For the second artificial example consisting of a tone with an additional second harmonic, Melodia predicts the first 1.472 seconds to be unvoiced. If there is a melody, a frequency of 110 Hz is predicted. The annotated frequency is 130.81 Hz. Then, until second 8.49, the correct frequency (lower frequency) of 130.81 Hz is predicted. At the end of the signal, Melodia predicts the frames to be unvoiced. However, the estimated frequency of -130.81 is correct. There is no frame for which the frequency of the second harmonic (261.62 Hz) is predicted.

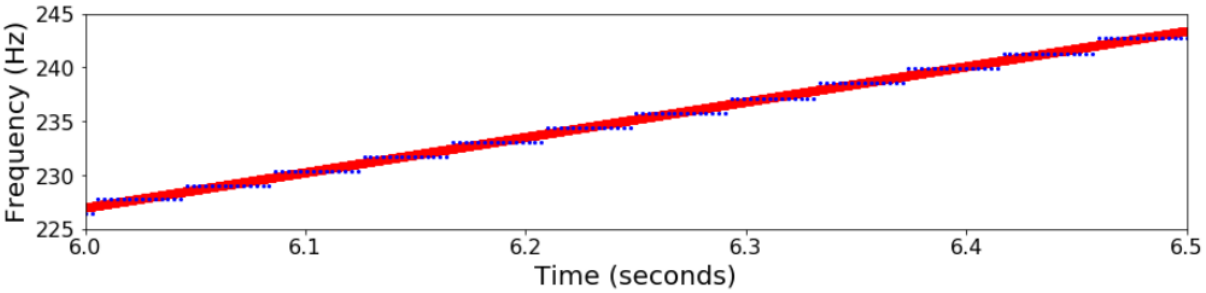


Figure 3.6: Frequencies estimated by Melodia (blue) and frequency annotations (red) for an excerpt of the chirp signal. Melodia assigns the frequencies to bins with a 10 cent frequency resolution.

The third artificial example is the chirp signal (30 Hz to 2000 Hz). At the beginning, Melodia predicts 0 Hz and then -220 Hz. For the annotated frequencies above approximately 171.6 Hz, the frequency estimates are correct. Melodia assigns the frequencies to bins for the computation of a salience function. These bins cover frequencies between 55 Hz and 1760 Hz. So Melodia can only predict frequencies in this range. For the chirp signal, we observed that Melodia predicts subharmonics for frequencies above approximately 1729.5 Hz. The bins for the salience function have a frequency resolution of 10 cents. So the frequencies that can be predicted by Melodia have a resolution of 10 cents. This can be observed when examining the estimated frequencies of the chirp signal in more detail. See Figure 3.6 for an excerpt from the chirp signal.

For MusicDelta_Rock_STEM_02, the estimated frequencies are not very accurate. There are some octave errors (estimation is one octave too high). This is probably due to the frequency range of Melodia. There are some frequencies below 55 Hz in the annotation of this audio example, however Melodia can only predict frequencies above 55 Hz. Furthermore, Melodia enhances mid-band frequencies. This leads to errors for audio examples which contain low frequencies.

The F0-estimations of Melodia for MusicDelta_InTheHalloftheMountainKing_STEM_03 are similar to the annotations with only few wrong predictions.

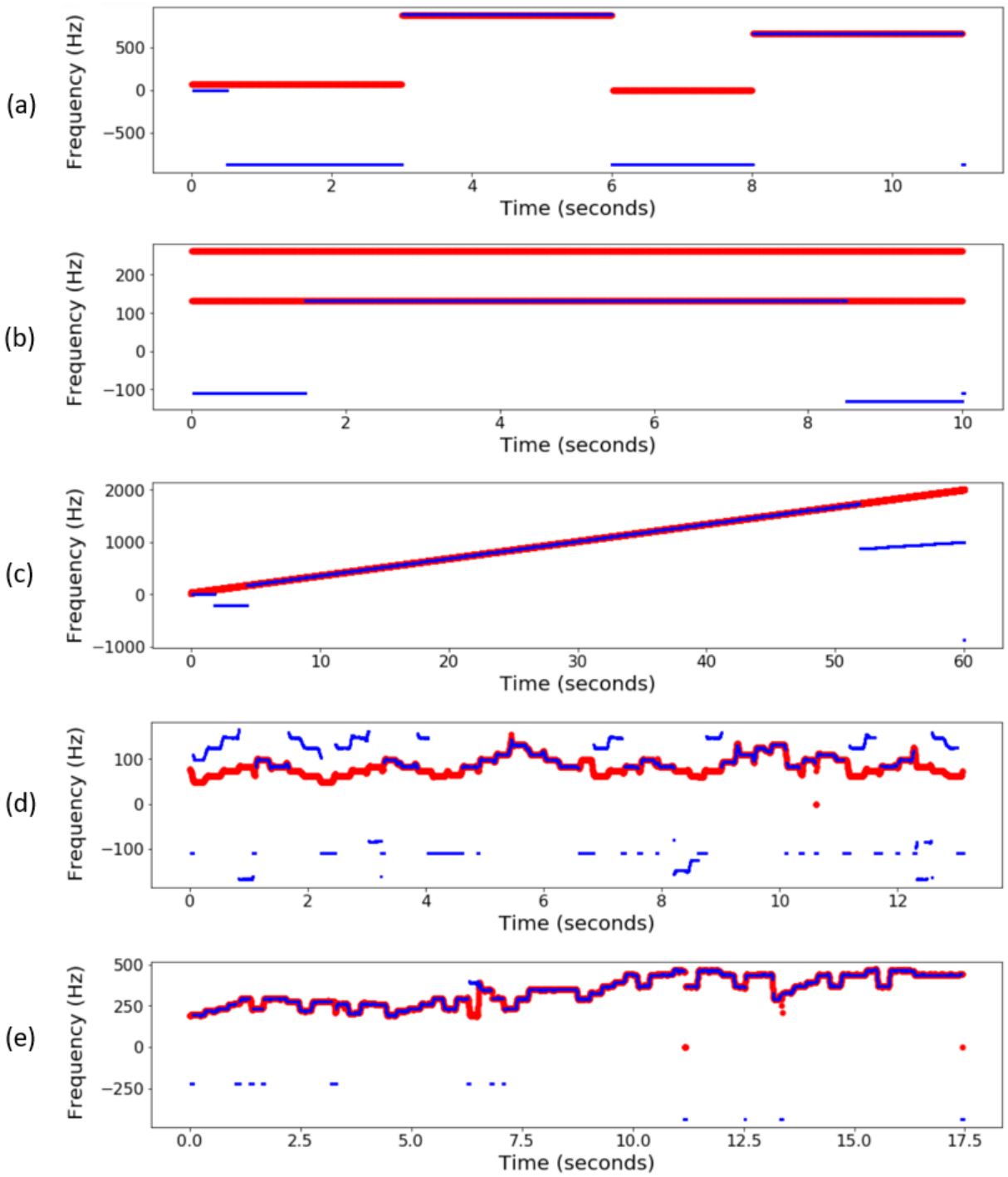


Figure 3.7: Frequencies estimated by Melodia (blue) and frequency annotations (red) for running examples. **(a)** Sinusoids. **(b)** Signal with Harmonics. **(c)** Chirp signal. **(d)** MusicDelta_Rock_STEM_02. **(e)** MusicDelta_InTheHalloftheMountainKing_STEM_03.

Chapter 4

DNN-Based Approach

For fundamental frequency estimation, there exist model-based approaches and DNN-based approaches. In this chapter, we discuss approaches based on deep neural networks (DNNs). First, CREPE is introduced in Section 4.1. CREPE is a convolutional neural network developed by Kim et al. [1]. We give some general information about CREPE and its network architecture and evaluate it on the running examples. Then, in Section 4.2, own approaches based on CREPE are introduced. This includes data augmentation, a modified resolution for the output vector, layer freezing and a modified network structure.

4.1 CREPE

CREPE is a data-driven approach for monophonic fundamental frequency estimation using a deep convolutional neural network (Deep CNN). The acronym CREPE is short for "A Convolutional Representation for Pitch Estimation". This network was developed by Jong Wook Kim, Justin Salamon, Peter Li and Joan Pablo Bello in 2018 [1]. In contrast to previous approaches such as e.g. SWIPE [5, 6] (Section 3.3), it computes the F0-trajectory directly from the audio data using a neural network without the application of signal processing steps.

An implementation of the neural network with trained weights is available online on github [25]. This version of CREPE is trained on six datasets of monophonic audio (vocal and instrumental). The following datasets were used: MIR-1K [26], Bach10 [27], RWC-Synth [4, 28], MedleyDB [14], MDB-STEM-Synth [8] and NSynth [29].

4.1.1 General Information

When pitch estimation is performed using CREPE [1], audio recordings are necessary as input. For this, the audio recordings can have an arbitrary sampling rate, but should be given in the *.wav* file format. CREPE can process one or multiple audio files simultaneously. When evaluating an audio file, it is first resampled by CREPE using the python module "resampy" [30]. The network is trained for audio files with a sampling rate of 16000 Hz, so each file is resampled to this sampling rate. The resampled audio files are then splitted into short excerpts also referred to as frames. This is done using a window length of 1024 samples (corresponding to 64 milliseconds). The excerpts have a distance of 10 milliseconds (corresponding to 160 samples). This distance is referred to as the hop size. The hop size can also be adjusted via a parameter of CREPE. Each excerpt is evaluated by the neural network of CREPE separately. The output of the neural network is a 360-dimensional vector for each audio excerpt. From this vector, the estimated fundamental frequency and a confidence value can be derived. For the relation between the input and the output of CREPE, see also Figure 4.1.

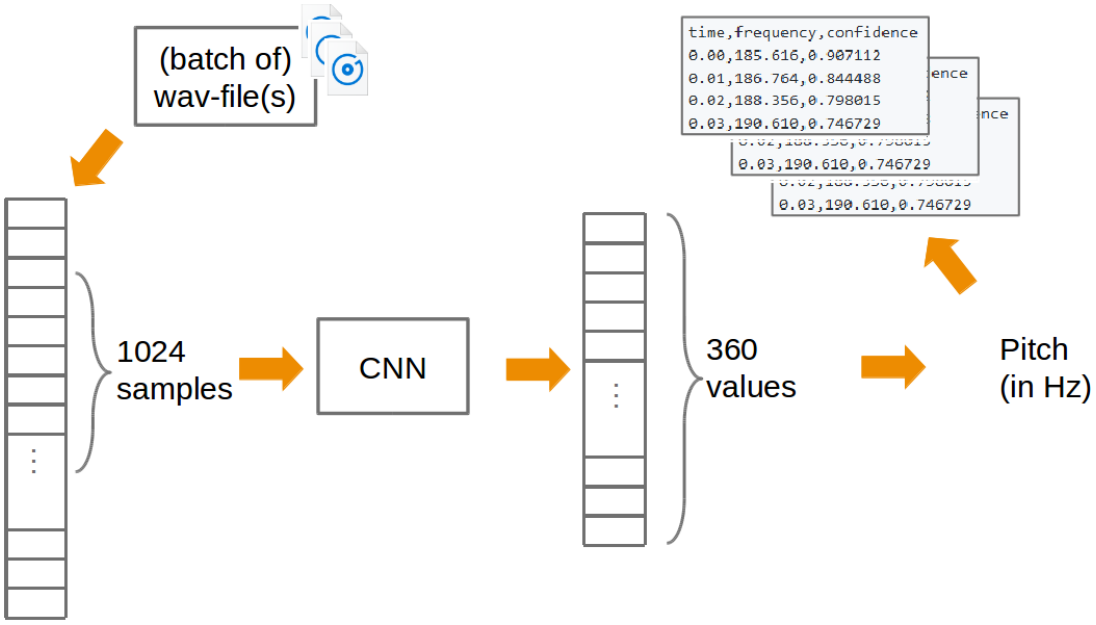


Figure 4.1: Input and output of the convolutional neural network of CREPE: Input is a batch of *.wav* files which are split into excerpts of 1024 samples. Using a neural network, a 360-dimensional vector is computed from which a pitch value can be derived.

The output of the network for an audio excerpt is a vector

$$\hat{y} := (\hat{y}(1), \hat{y}(2), ..., \hat{y}(360)) \in [0, 1]^{360} \tag{4.1}$$

with values between 0 and 1. A sequence of output vectors for all excerpts of an audio recording

is also referred to as the activation of CREPE. The values in a vector $\hat{y}$ are used as weights for a fixed 360-dimensional vector of cent values

$$\omega_{ç} = (\omega_{ç}(1), \omega_{ç}(2), ..., \omega_{ç}(360)) \in \mathbb{R}^{360}. \tag{4.2}$$

All cent values in the vector are given as distance in cents to a reference frequency of 10 Hz. The values in $\omega_{ç}$ have a difference of 20 cents. They range over six octaves with twelve semitones per octave. In combination with a 20 cents resolution (five entries per semitone) this results in $6 \cdot 12 \cdot 5 = 360$ values. The vector $\omega_{ç}$ starts with a cent value $\omega_{ç}(1) = 1997.3794084376191$. This corresponds to 31.70 Hz (between B0 and C1). The last value $\omega_{ç}(360)$ in the vector is 9177.3794084376191. This cent value corresponds to approximately 2005.50 Hz (between B6 and C7). For the values of the vector $\omega_{ç}$ and the corresponding frequency values, see also Figure 4.2.

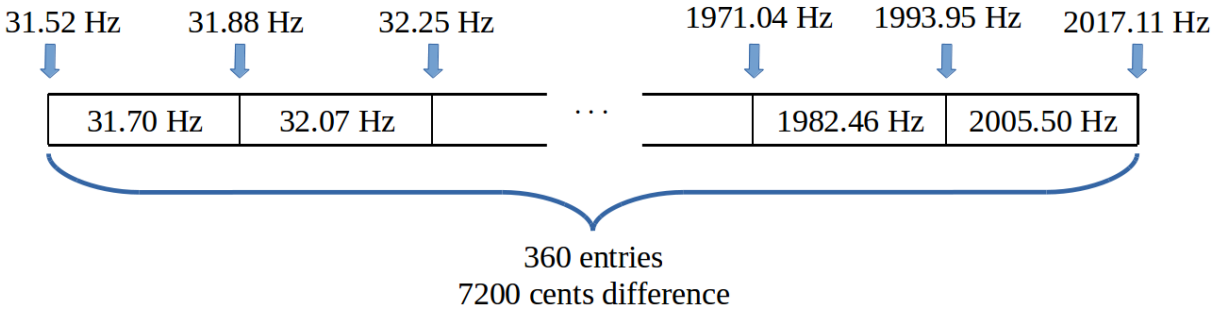


Figure 4.2: Values of the cent vector $\omega_{ç}$. CREPE returns a vector $\hat{y}$ which consists of weights for the vector $\omega_{ç}$. The 360-dimensional vector $\omega_{ç}$ contains fixed cent values (distances to a reference frequency of 10 Hz) with a difference of 20 cents.

To derive the estimated result in cents for a given audio excerpt, the vector $\omega_{ç}$ is multiplied with the weight vector $\hat{y}$ (output vector of the network):

$$\omega_{ç,\text{result}} := \frac{\langle \hat{y} | \omega_{ç} \rangle}{\sum_{i=1}^{360} \hat{y}(i)} \in \mathbb{R}. \tag{4.3}$$

The value $\omega_{ç,\text{result}}$ is the estimated result for an audio excerpt given in cents. This value can be converted into a frequency value given in Hertz using the following formula:

$$\omega_{\text{result}} := 10 \text{ Hz} \cdot 2^{\omega_{ç,\text{result}}/1200}. \tag{4.4}$$

In the formula for $\omega_{ç,\text{result}}$, all values from the output vector $\hat{y}$ have an influence on the result. Though, it is questionable, whether values with a larger difference to the peak of the estimated vector should have an influence on the result. For example, if there are high weights for cent values of approximately 7000 cent, there might still be small weights for lower cent values. However, all weights are considered in the formula. This can be especially problematic for a

weight vector $\hat{y}$ with a peak at a very low or very high index and small weights (> 0) in other entries. In those cases, the formula can lead to a cent value that is pulled towards a more centered result. Another problem can occur, if the weight vector contains two peaks at different positions. In this case, both peaks influence the result. This leads to a resulting frequency value between the frequencies of the peaks. So the result is a frequency that is not present in the audio excerpt. These problems are addressed by modifying the formula for $\omega_{\dot{\text{C}},\text{result}}$. Instead of all weights from the output vector of the network, only selected weights influence the pitch result. For this purpose, the index of the maximum value of the output vector is determined. For the pitch result, we only consider values at indices around this maximum index. This formula for the estimated result given in cents is also used in the code published on github:

$$\omega_{\dot{\text{C}},\text{result}} := \frac{\sum_{i=m-4}^{m+4} \hat{y}(i) \cdot \omega_{\dot{\text{C}}}(i)}{\sum_{i=m-4}^{m+4} \hat{y}(i)} \qquad m = \underset{i}{\text{argmax}} \quad \hat{y}(i). \tag{4.5}$$

The estimated cent value can then be converted into a fundamental frequency estimate in Hertz using Equation 4.4. An illustration of both methods for the computation of the resulting estimate $\omega_{\dot{\text{C}},\text{result}}$ in cents is shown in Figure 4.3. Figure 4.3a depicts the method using Equation 4.3. It

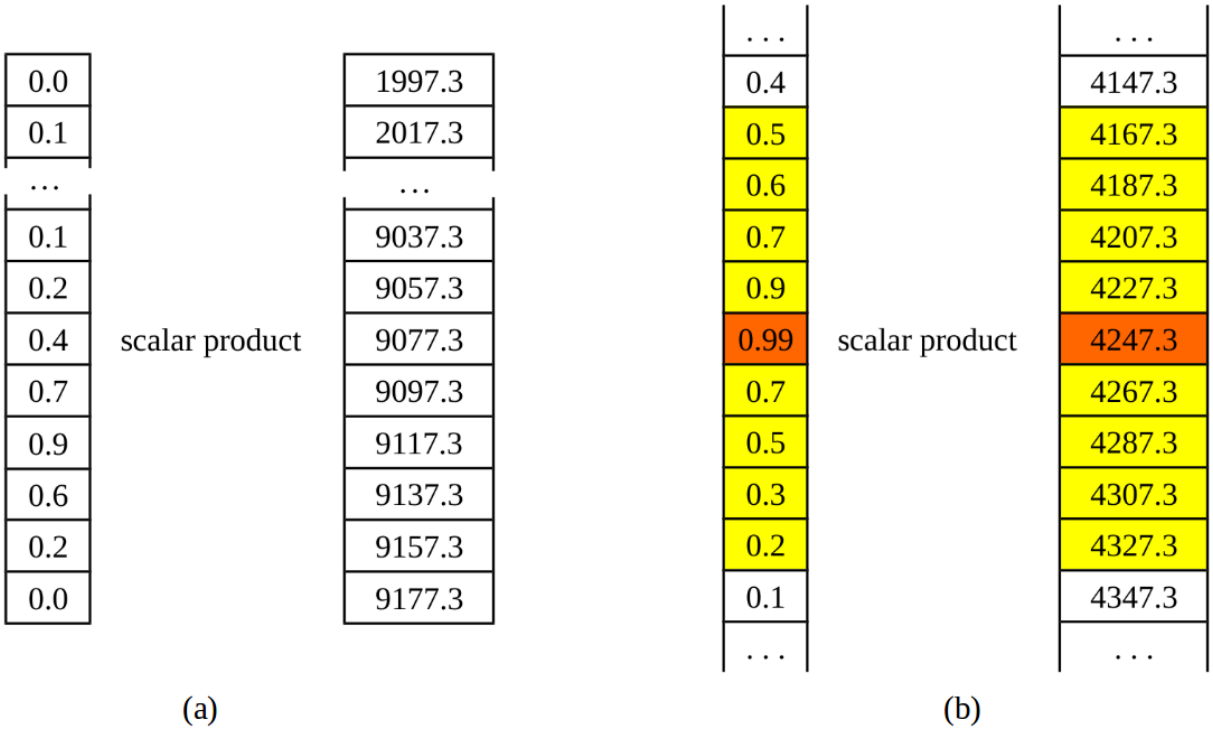


Figure 4.3: Different methods for the computation of a cent value from the output vector of CREPE. **(a)** Method using the complete vector $\hat{y}$ (left) and the complete vector $\omega_{\dot{\text{C}}}$ (right). **(b)** Method using only selected values from the vector $\hat{y}$ (left) and the vector $\omega_{\dot{\text{C}}}$ (right).

uses the complete vector $\hat{y}$ (left) and the complete vector $\omega_{\dot{\text{C}}}$ (right). At each position, the values

from the both vectors are multiplied: $0.0 \cdot 1997.3 + 0.1 \cdot 2017.3 + ... + 0.2 \cdot 9157.3 + 0.0 \cdot 91773$. This value is then divided by the sum over all values in vector $\hat{y}$: $0.0 + 0.1 + ... + 0.2 + 0.1$. The result of the division is the resulting cent value for an audio excerpt.

Figure 4.3b shows the method from Equation 4.5. It uses only selected values from the vector $\hat{y}$ (left) and the vector $\omega_{\text{Ç}}$ (right). The maximum of the vector $\hat{y}$ is determined (here: 0.99, orange entry). For the computation, only the values at indices with a distance of at most 4 are considered (yellow entries). The scalar product of these values is computed: $0.5 \cdot 4167.3 + 0.6 \cdot 4187.3 + ... + 0.3 \cdot 4307.3 + 0.2 \cdot 4327.3$. This value is then divided by the sum over the 9 marked values in vector $\hat{y}$: $0.5 + 0.6 + ... + 0.3 + 0.2$, resulting in the estimated cent value of CREPE. So Equation 4.5 only focuses on an area around a high weight. This method can be considered as more reasonable, since weights with a higher distance to the peak do not influence the result. In the presence of e.g. two peaks, the previous Equation 4.3, which uses the complete vectors, results in a pitch estimate between the pitches belonging to the peaks. With the method of Equation 4.5, one peak is selected and the resulting frequency value is only influenced by this peak (and the neighboring values). So if there are two pitches in an audio excerpt, one is chosen for the pitch estimate. However, exactly four values before and after the peak are considered. Also different values for the size of the neighborhood are possible and the selection of this number can not be justified. Another open problem is the choice of an appropriate cent resolution. In CREPE, the values in $\omega_{\text{Ç}}$ have a difference of 20 cents. Also different values would be possible. However, this resolution also allows the estimation of pitches with a finer resolution than 20 cents. E.g. if a vector has high weights at $\hat{y}(200)$ and $\hat{y}(201)$, both weights are multiplied with the corresponding cent values at $\omega_{\text{Ç}}(200)$ and $\omega_{\text{Ç}}(201)$. This leads to a frequency estimate between $\omega_{\text{Ç}}(200)$ and $\omega_{\text{Ç}}(201)$.

In summary, there exist two methods for the interpretation of the output vector $\hat{y}$. One possibility is to consider the complete vector for the computation of a pitch estimate, another possibility is to focus on an area around the peak of the vector. The second method as described in Equation 4.5 can be considered as more accurate. However, the number of neighboring values around the peak and the resolution of the output vector could be selected differently.

In addition to an estimate for the fundamental frequency, a confidence value can be derived. This is done using the output vector of the network $\hat{y}$. The confidence value is the maximum value in the vector $\hat{y}$. Since all values in this vector are between 0 and 1, the confidence is also in this range. A higher confidence value corresponds to a higher predicted probability that the audio frame is voiced. In the implementation of CREPE, a threshold of 0.5 was fixed. If the confidence is below 0.5, the corresponding frame is predicted to be unvoiced. This threshold could also be evaluated and set to another value. The voicing estimate in combination with the pitch estimate for a sequence of time instances is the result of CREPE evaluated on an audio file.

These results are summarized in one *.csv* file for each audio input file with the following columns:

- timestamp: Time at which a frequency value was estimated in seconds.
- frequency: Estimated frequency value in Hertz.
- confidence: Confidence value describing how likely it is that the given frame is voiced.

For the evaluation of audio files with CREPE, there are some further parameters that can be set by the user:

- Step size: When splitting the audio files into processed sections, a hop size is necessary. The default hop size is 10 milliseconds.
- Model capacity: There are five different model sizes for the neural network: tiny, small, medium, large and full. The models differ in their number of filters. The default model capacity is "full".
- Viterbi: Viterbi allows to temporal smooth the resulting pitch curve. There are no further parameters for Viterbi, it can be set to true or false (default: set to false).
- Parameters for plotting/storage:
  - Save Activation: Saves the output activation matrix (result of the last layer of CREPE).
  - Save Plot: Saves the plot of the activation matrix (a colored plot of the activation values can be stored).
  - Plot Voicing: This option is only available, if the activation matrix plot is saved. Then this parameter can be used to include a plot for the voicing activity detection.

Each of the five different models consists of its network architecture (defined by the layers, filter sizes and number of filters) and the trained weights for the layers. In the following we refer to the models as M_{tiny}, M_{small}, M_{medium}, M_{large} and M_{full}. Here, we define a model to be a tuple of the architecture and the weights, e.g.

$$M_{\text{tiny}} := (M_{\text{tiny,architecture}}, M_{\text{tiny,weights}}). \tag{4.6}$$

The differences between the models is explained in more detail in Section 4.1.2.

4.1.2 Network Architecture

CREPE consists of five different models with different capacities: $M_{\text{tiny}}, M_{\text{small}}, M_{\text{medium}}, M_{\text{large}}$ and M_{full}. However, the general structure of the architecture is similar for all models.

CREPE receives audio files as input, which are resampled to a sampling frequency of 16000 Hz. Excerpts of 1024 samples from the audio recordings are used as input for the neural network. Therefore, the neural network of CREPE starts with an input layer with a shape of 1024. Then (after a reshaping layer), a convolutional block is repeated six times. Each convolutional block consists of a Convolution (Conv2D) layer, a Batch Normalization layer [31], a Max Pooling (MaxPooling2D) layer and a Dropout layer [32]. The result of the last convolutional block (after Permution to switch some dimensions) is flattened. The last layer is then a Fully Connected layer (Dense) which leads to a 360-dimensional output. All models, independent of the model capacity, use the same filter sizes for the convolutional layers. The filter sizes are shown in Table 4.1. The number of filters of the convolutional layers depend on the position of the layer in the network and on the chosen model capacity. Though, the ratio of the number of the filters is the same for each model capacity. For the number of filters, see Table 4.2. The structure of the full model M_{full} and its layers is shown in Figure 4.4 (figure similar to [1]). To get an overview of the dimensions and the number of parameters for each layer in M_{full}, see Figure 4.5. The number of parameters for a convolutional layer depends on the number of filters, the filter sizes (including number of channels of the filter) and the number of filter biases (each filter has a bias value). So it can be computed as

$$\#\text{filters} \cdot \text{filtersize} \cdot \#\text{channels} + \#\text{biases}. \tag{4.7}$$

For example the number of parameters of the third convolutional layer (conv3) of M_{full} is $128 \cdot 64 \cdot 128 + 128 = 1048704$. The total number of trainable parameters of the full model is 22 239 976 (from Figure 4.5).

index of convolutional layer	filter size
1	512
2	64
3	64
4	64
5	64
6	64

Table 4.1: Filter sizes of the convolutional layers depending on the index of the convolutional block.

As activation functions, Kim et al. used the Rectified Linear Unit (ReLU) for the convolution layers and a sigmoid for the fully connected layer. The authors implemented the model architecture in Keras [33]. The neural networks are trained using annotated audio excerpts. So the input for the

index of convolutional layer	M_{tiny}	M_{small}	M_{medium}	M_{large}	M_{full}
1	128	256	512	768	1024
2	16	32	64	96	128
3	16	32	64	96	128
4	16	32	64	96	128
5	32	64	128	192	256
6	64	128	256	384	512

Table 4.2: Number of filters depending on the index of the convolutional block and the chosen model capacity.

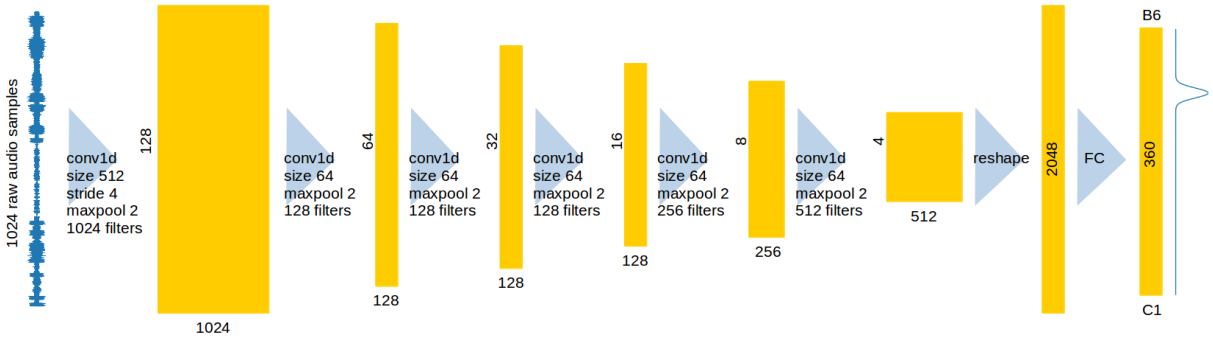


Figure 4.4: Architecture of the original CREPE model M_{full} (model capacity "full").

training are audio excerpts with 1024 samples. Additionally, for each audio excerpt the expected frequency result in cents $\omega_{\text{Ç,correct}} \in \mathbb{R}$ is known. From this, the expected 360-dimensional output vector

$$y_{\text{correct}} = (y_{\text{correct}}(1), y_{\text{correct}}(2), ..., y_{\text{correct}}(360)) \in [0,1]^{360} \tag{4.8}$$

can be derived. This vector is necessary for the training process. It consists of high values at positions that are near to weights for the correct cent values (gaussian-blurred with a standard deviation of 25 cents):

$$y_{\text{correct}}(i) = \exp\left(-\frac{(\omega_{\text{Ç}}(i) - \omega_{\text{Ç,correct}})^2}{2 \cdot 25^2}\right). \tag{4.9}$$

So if the correct frequency has only a small distance to the frequency corresponding to index i of the vector y_{correct}, the value is near to $\exp(0) = 1$. For indices corresponding to a frequency with a large distance to the correct frequency, this value is near to 0.

Layer (type)	Output Shape	Param #
input (InputLayer)	(None, 1024)	0
input-reshape (Reshape)	(None, 1024, 1, 1)	0
conv1 (Conv2D)	(None, 256, 1, 1024)	525312
conv1-BN (BatchNormalization	(None, 256, 1, 1024)	4096
conv1-maxpool (MaxPooling2D)	(None, 128, 1, 1024)	0
conv1-dropout (Dropout)	(None, 128, 1, 1024)	0
conv2 (Conv2D)	(None, 128, 1, 128)	8388736
conv2-BN (BatchNormalization	(None, 128, 1, 128)	512
conv2-maxpool (MaxPooling2D)	(None, 64, 1, 128)	0
conv2-dropout (Dropout)	(None, 64, 1, 128)	0
conv3 (Conv2D)	(None, 64, 1, 128)	1048704
conv3-BN (BatchNormalization	(None, 64, 1, 128)	512
conv3-maxpool (MaxPooling2D)	(None, 32, 1, 128)	0
conv3-dropout (Dropout)	(None, 32, 1, 128)	0
conv4 (Conv2D)	(None, 32, 1, 128)	1048704
conv4-BN (BatchNormalization	(None, 32, 1, 128)	512
conv4-maxpool (MaxPooling2D)	(None, 16, 1, 128)	0
conv4-dropout (Dropout)	(None, 16, 1, 128)	0
conv5 (Conv2D)	(None, 16, 1, 256)	2097408
conv5-BN (BatchNormalization	(None, 16, 1, 256)	1024
conv5-maxpool (MaxPooling2D)	(None, 8, 1, 256)	0
conv5-dropout (Dropout)	(None, 8, 1, 256)	0
conv6 (Conv2D)	(None, 8, 1, 512)	8389120
conv6-BN (BatchNormalization	(None, 8, 1, 512)	2048
conv6-maxpool (MaxPooling2D)	(None, 4, 1, 512)	0
conv6-dropout (Dropout)	(None, 4, 1, 512)	0
transpose (Permute)	(None, 1, 4, 512)	0
flatten (Flatten)	(None, 2048)	0
classifier (Dense)	(None, 360)	737640

Total params: 22,244,328
Trainable params: 22,239,976
Non-trainable params: 4,352

Figure 4.5: Model summary of Keras for the full model M_{full}.

In order to train the network, the output of the network $\hat{y}$ (Equation 4.1) is compared to the correct output vector y_{correct} (Equation 4.8). For this, the authors used the binary cross entropy loss:

$$\mathcal{L}(y_{\text{correct}}, \hat{y}) = \sum_{i=1}^{360}(-y_{\text{correct}}(i) \log \hat{y}(i) - (1 - y_{\text{correct}}(i)) \log(1 - \hat{y}(i))). \tag{4.10}$$

This loss function is optimized using the ADAM optimizer [34] with a learning rate of 0.0002.

4.1.3 Evaluation on Running Examples

For the evaluation of pitch estimation approaches, some artificial signals were introduced in Section 2.3.1. In Section 2.3.2.2, we introduced some further audio examples from the MDB-stem-synth dataset [8]. In this section, we evaluate CREPE [1] on these audio examples. For all evaluations in this section, we use the tiny model of CREPE M_{tiny}.

4.1.3.1 Sinusoids

The first artificial signal consists of pure sinusoids. It starts with a C2 (65.4064 Hz), then continues with an A5 (880.0 Hz), then a pause and finally an E5 (659.255 Hz). All tones last for 3 seconds each and the pause takes 2 seconds. We created the signal with a sampling frequency of 16000 Hz so that no resampling for CREPE is necessary. On this signal, we applied CREPE (with model M_{tiny}) with the standard hop size of 10 milliseconds. The estimated fundamental frequency values are shown in Figure 4.6a. The estimations of CREPE are similar to the frequency annotations. Only at second 8, at the end of the pause, some too high frequencies are predicted. The reason can be that at this time instance CREPE receives an audio excerpt which consists of the pause and the new tone. During the pause, CREPE predicts a frequency of 0 Hz. If the convolutional neural network of CREPE gets a zero vector as input, the output is a vector of undefined numbers (not a number, *nan*). In cases where such values are part of the frequency predictions, instead a frequency of 0 Hz is returned. So CREPE returns 0 Hz during the pause.

4.1.3.2 Signal with Harmonics

Music recordings that are not artificially created consist of fundamental frequencies and harmonics. In order to examine the estimations of CREPE in the presence of harmonics, we applied CREPE to a signal with one overtone. The amplitude of the fundamental frequency (C3) increases linearly from 0 to 1 whereas the amplitude of the second partial (C4) decreases from 1 to 0. As CREPE was trained with data including overtones, it is expected to correctly predict the fundamental

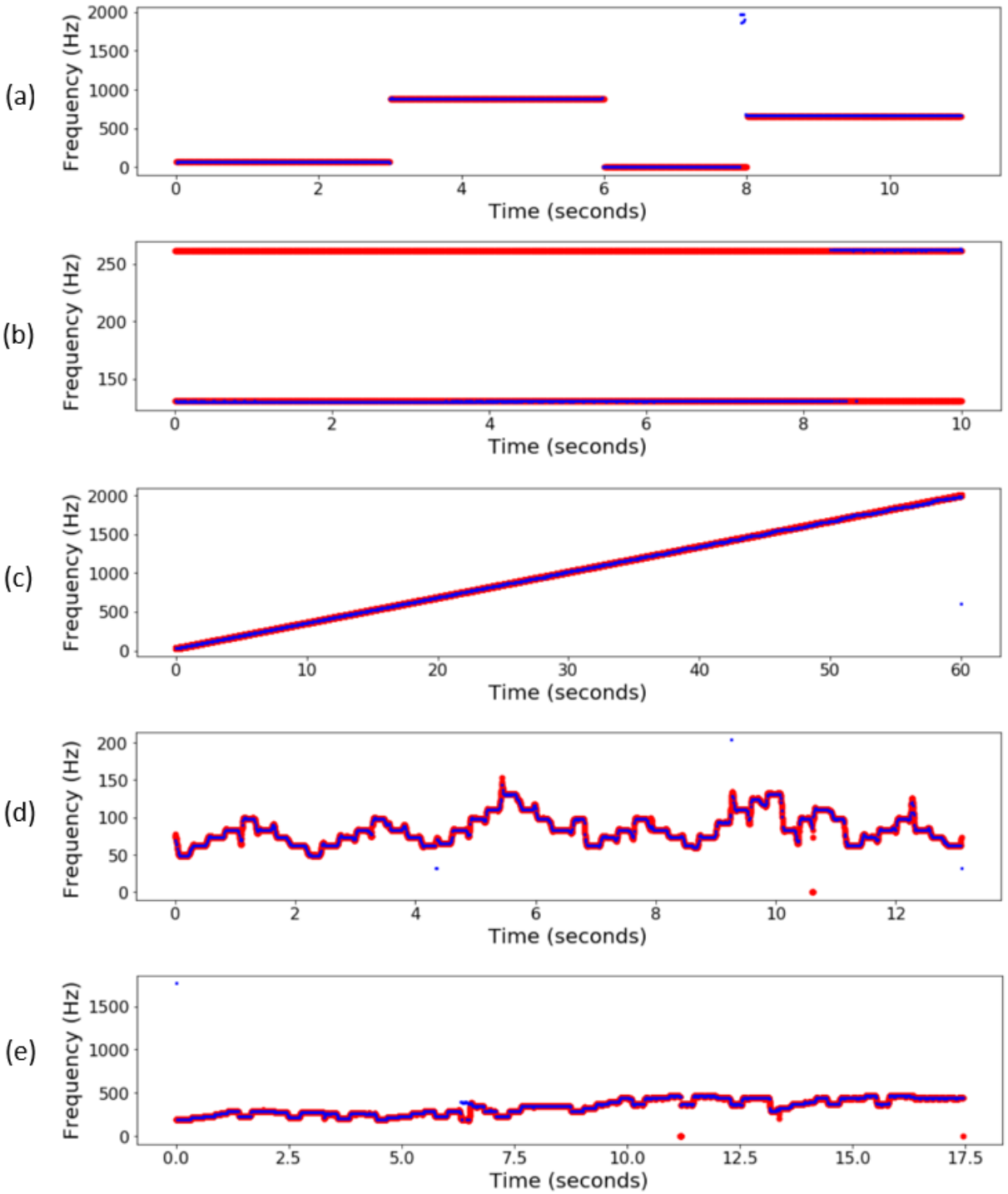


Figure 4.6: Frequencies estimated by the tiny model M_{tiny} of CREPE (blue) and frequency annotations (red) for running examples. **(a)** Sinusoids. **(b)** Signal with Harmonics. **(c)** Chirp signal. **(d)** MusicDelta_Rock_STEM_02.
(e) MusicDelta_InTheHalloftheMountainKing_STEM_03.

frequency in spite of the presence of overtones. So the objective was to examine, how high the amplitude of the second partial can be compared to the amplitude of the first partial so that the correct fundamental frequency is predicted by CREPE. First, we applied CREPE (M_{tiny}) with a hop size of 10 milliseconds to the signal (sampling rate of 16000 Hz). Figure 4.6b shows the estimated frequencies of CREPE and the annotated frequencies. It can be recognized that CREPE predicted the lower frequency (C3) for more than 8 seconds and then predicted the higher frequency. Starting with the fifth second, the amplitude of the C4 is higher than the amplitude fo the C3. However, the lower frequency was predicted. To further examine this, we analyzed the output vectors of CREPE (activations) and the confidence values. Figure 4.7a shows the vectors returned by the neural network. Each column in the figure corresponds to one 360-dimensional output vector $\hat{y}$. The colors in a column indicate the values of the entries in the vector. Since the entries of the output vector correspond to frequencies, they are shown in Hertz. It is remarkable that in the first half, there were not only high values for a frequency of 130.81 Hz corresponding to a C3, but also for the subharmonic of 65.406 Hz corresponding to a C2. This indicates that CREPE learned to recognize a certain frequency by also searching for the second partials. So the presence of the C3 increased the probability for the presence of a C2. The derived frequency estimates are shown in Figure 4.7b. The prediction of the lower frequency at seconds 5 to 8 (fundamental frequency has lower amplitude than overtone) also indicates that CREPE learned to predict the fundamental frequency in the presence of harmonics. For a higher amplitude of the second partial, the estimations switched to the higher frequency. At these time instances, CREPE predicted lower confidence values. For a plot of the confidence values of CREPE see Figure 4.7c. As the amplitude of the second partial converges to 1, the confidence values increased.

So CREPE also uses the second partial of a signal for the prediction of the fundamental frequency and the estimated frequency does not always correspond to the frequency with the highest amplitude if this is the second partial.

4.1.3.3 Chirp Signal

The last artificial example is a chirp signal. CREPE is designed to correctly predict frequencies between 31.70 Hz and 2005.50 Hz. The chirp signal covers frequencies between 30 Hz and 2000 Hz. The estimated fundamental frequencies of CREPE are shown in Figure 4.6b. The predictions of CREPE are similar to the expected frequencies. At the beginning of the signal, there are small deviations, because the frequency range of CREPE starts at a frequency of 31.70 Hz instead of 30 Hz.

Additionally, we examined the pitch estimations for CREPE for frequencies above the upper limit of CREPE. So another chirp signal is created that starts with 30 Hz and ends with 2500 Hz

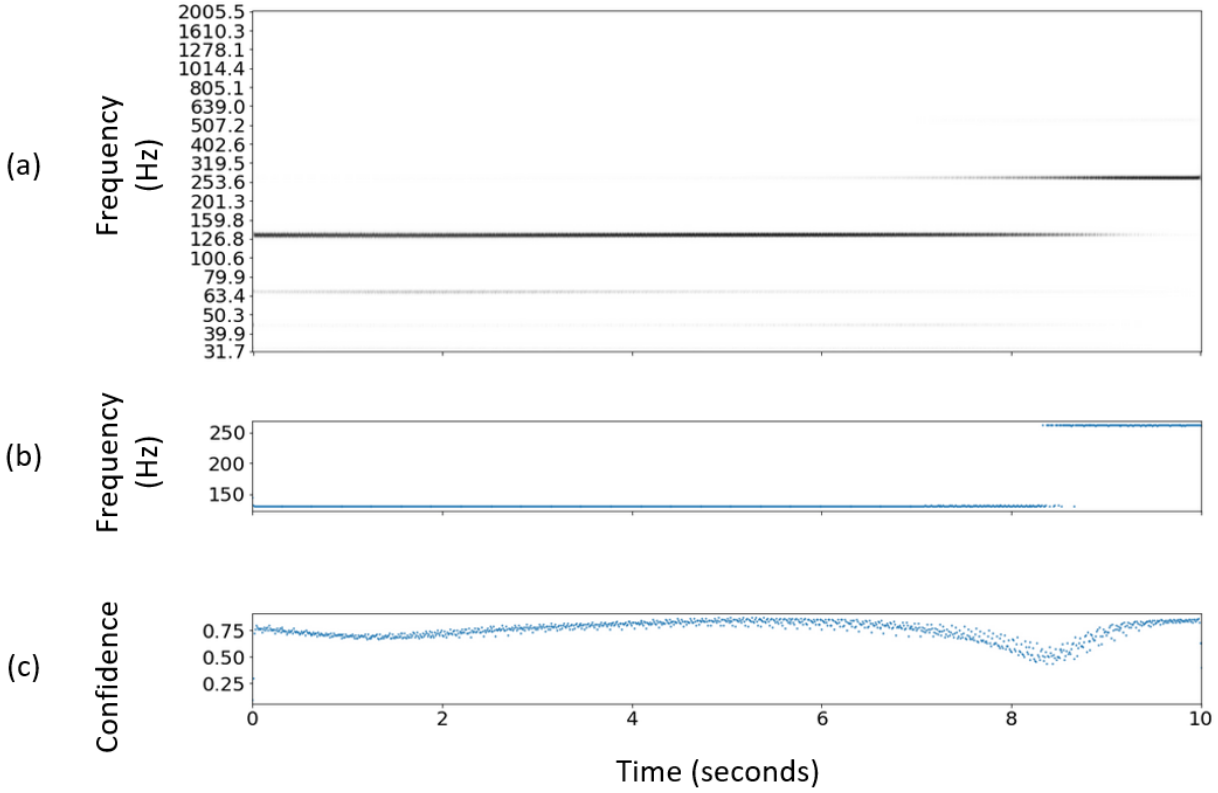


Figure 4.7: Output of CREPE (M_{tiny}) applied to a signal with 130.81 Hz (C3, amplitude linearly decreasing) and 261.63 Hz (C4, amplitude linearly increasing). **(a)** Activation (output of network of CREPE). **(b)** Predicted fundamental frequencies of CREPE. **(c)** Predicted confidence of CREPE.

($>$ 2005.50 Hz) with frequencies increasing linearly. The duration of the signal is 60 seconds. We created the signal with a sampling frequency of 16000 Hz. For the resulting pitch estimates of CREPE see Figure 4.8. We applied CREPE using the tiny model M_{tiny} and a hop size of 10 milliseconds. The figure shows that the estimated and the expected frequencies are similar until a frequency around 2000 Hz. For higher frequencies, CREPE predicted the subharmonics of the correct frequency instead (subharmonics plotted in gray). This is also reasonable, because the vector $\omega_{\text{Ç}}$ does not contain values for higher frequencies, so the upper limit for possible frequencies is fixed due to this vector.

Furthermore, we created a chirp with a smaller frequency range. The purpose was to examine the similarity of the predicted and the estimated frequencies in detail. Here, we use a chirp signal with frequencies between 220 Hz (A3) and 440 Hz (A4). The signal has a duration of 10 seconds and a sampling frequency of 16000 Hz. We applied CREPE to this signal using M_{tiny} and a hop size of 10 milliseconds to obtain frequency estimations. These estimations were then compared to the expected frequencies. The differences are plotted in Figure 4.9. The vertical lines in the plot show the position of the natural tones between A3 and A4. The difference function indicates

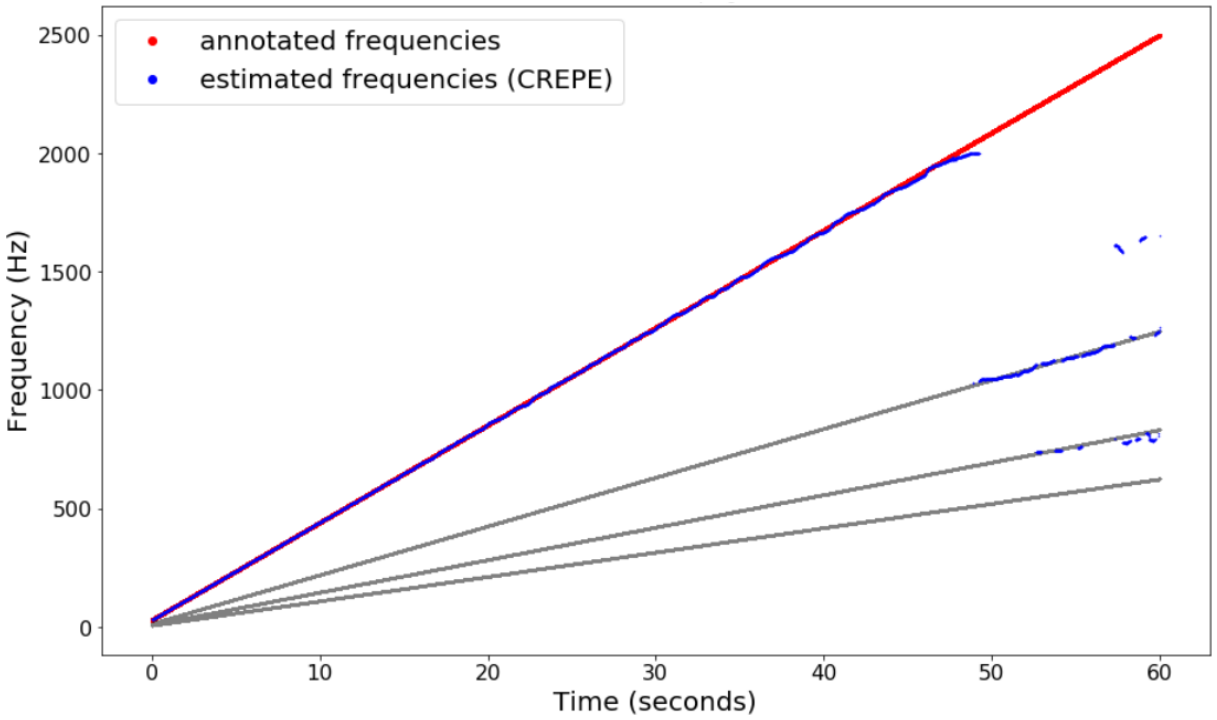


Figure 4.8: Linear chirp signal (30 Hz to 2500 Hz, duration: 60 seconds) estimated with CREPE (M_{tiny}). The annotated frequencies are shown in red, the frequencies estimated by CREPE in blue and the subharmonics of the annotated frequencies in gray.

that the estimations are more exact for musical notes compared to frequencies that can not be assigned to a musical note. Since CREPE was not trained on frequency values but on music recordings, it can be assumed that frequencies belonging to musical notes occurred more often than frequencies between musical notes. So the distribution of frequencies in the training dataset can be the reason for a higher similarity of the estimations and the annotations at frequencies belonging to musical notes.

4.1.3.4 Examples from MDB-stem-synth Dataset

For the two running examples from the MDB-stem-synth dataset [8], MusicDelta_Rock_STEM_02 and MusicDelta_InTheHalloftheMountainKing_STEM_03, we applied CREPE using the tiny model M_{tiny} and a hop size of 10 milliseconds. For the estimated fundamental frequency values see Figure 4.6d and Figure 4.6e. In summary, it can be said that the estimations are similar to the annotated frequencies. For MusicDelta_Rock_STEM_02, there are two outliers at seconds 4.34 and 4.35 (too low frequencies), one outlier at second 9.26 (too high frequency) and one wrong estimate at the end of the recording (too low frequency). For MusicDelta_InTheHalloftheMountainKing_STEM_03, the estimations are also close to the annotated frequencies. At the beginning of the audio example, there is one wrong estimation (too

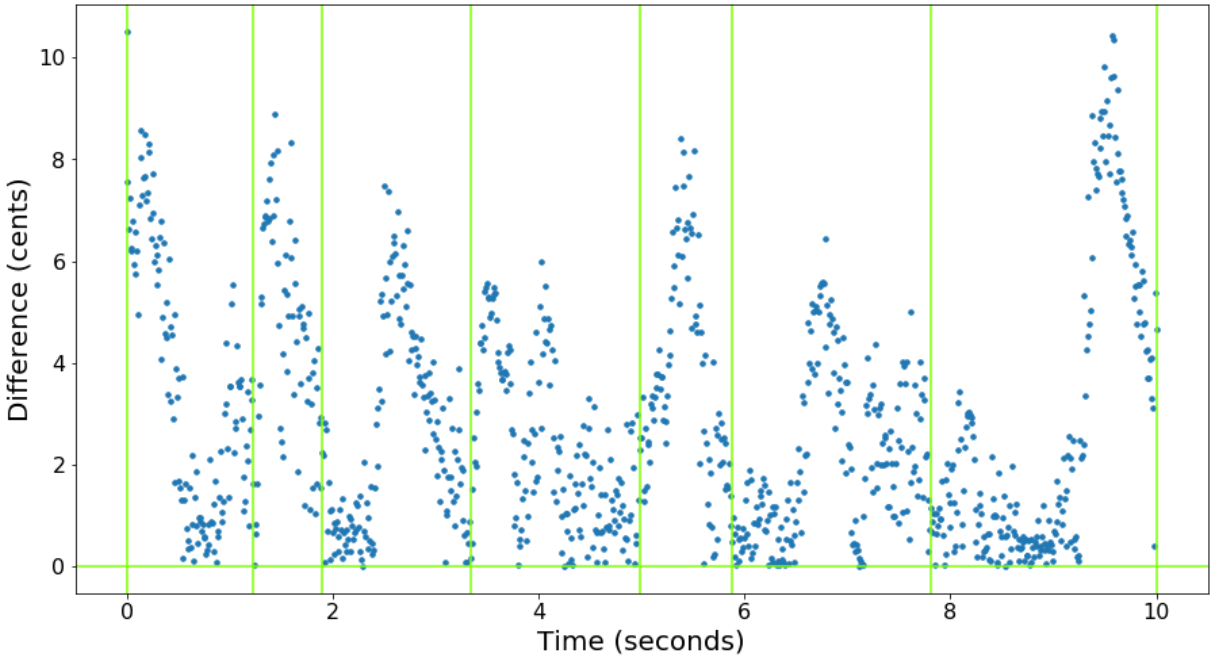


Figure 4.9: CREPE was applied to a chirp signal (220 Hz to 440 Hz, duration: 10 seconds). The resulting estimation was compared to the correct frequencies and the distance in cents is plotted over the time. The vertical green lines indicate tones: A3, B3, C4, D4, E4, F4, G4, A4.

high frequency) and at second 6.31 to second 6.49, there is an octave error (estimated frequencies one octave above annotated frequencies).

Summing up, the estimations of CREPE using the tiny model structure have a high similarity to the annotated frequencies.

4.2 Own Approaches

In the following, we present some own approaches for modifications of the neural network. They can help to improve the training of the network. Also the network can be further examined. First, we used data augmentation in order to increase the amount of available training data. Then, the influence of the dimension of the last layer of the network was examined. Next, we discuss the influence of the first layer of the network and fixed suitable weights. Finally, we propose a modified network structure with different filter sizes and a lower number of filters.

4.2.1 Data Augmentation

When training a network, a representative dataset is necessary. This is in some cases difficult to obtain. For pitch estimation tasks, the music recordings should be representative concerning

instruments, music genre and pitch distribution. When using the MDB-stem-synth dataset [8], some variety regarding instruments and genre is ensured. Although, there are no frequencies above 1200 Hz present in this dataset and even frequencies above 800 Hz are rather rare. The neural network of CREPE should be trained for frequencies until 2005.50 Hz. So using the MDB-stem-synth dataset for a training starting with random weights is not a promising approach. In order to include also higher frequencies in the dataset, data augmentation can be used.

Here, instead of using the original dataset for training the network, the pitch of some audio files was shifted. For this, a random shift between 0 cents (no shift) and 800 cents was chosen for each audio file. The lower limit of 0 cents was chosen to preserve the low frequencies in the dataset. We set the upper limit to 800 cents, because it should be prevented that frequencies above the maximal frequency of CREPE occur in the dataset. If a frequency of 1200 Hz (approximately maximal frequency in original dataset) is shifted by the maximal shift of 800 cents, this leads to a frequency of approximately 1905 Hz which is still in the frequency range covered by CREPE. The pitch shifting was performed using the python package LibROSA [35, 36, 37] (librosa.effects.pitch_shift). The distribution of the pitches of the annotation frames is shown in Figure 4.10. Compared to the original distribution, the proportion of low frequencies decreased and the proportion of high frequencies increased due to pitch shifting. Though, there are still more frames for low pitches than for high pitches. A similar pitch shifting was also applied to the validation set. We then retrained the network (architecture of M_{tiny}) starting with random weights using the pitch shifted training and validation set. This is expected to lead to a better performance of the network for higher pitches compared to a network trained on the original dataset. The model trained using pitch shifting is referred to as M_{shift}.

4.2.2 Output Vector Resolution

The last layer of CREPE is a Fully Connected layer. The output of this layer is a 360-dimensional vector $\hat{y}$ with values between 0 and 1 which are used as weights for a fixed 360-dimensional vector $\omega_{\text{Ç}}$. The vector $\omega_{\text{Ç}}$ contains cent values with a difference of 20 cents, see Section 4.1.1 for more details. When training a network and evaluating it, we found out that especially the raw pitch accuracies for a threshold of 10 cents are lower than the results reported by Kim et al. [1]. The idea was to start with the architecture of the tiny model M_{tiny} of CREPE , but change the size of the output layer. So the last fully connected layer with an output dimension of 360 was replaced by a fully connected layer with an output dimension of 720. This change also entails a modification of the vector $\omega_{\text{Ç}}$ defined in Section 4.1.1. This is now a vector

$$\omega_{\text{Ç},720} = (\omega_{\text{Ç},720}(1), \omega_{\text{Ç},720}(2), ..., \omega_{\text{Ç},720}(720)) \in \mathbb{R}^{720}. \tag{4.11}$$

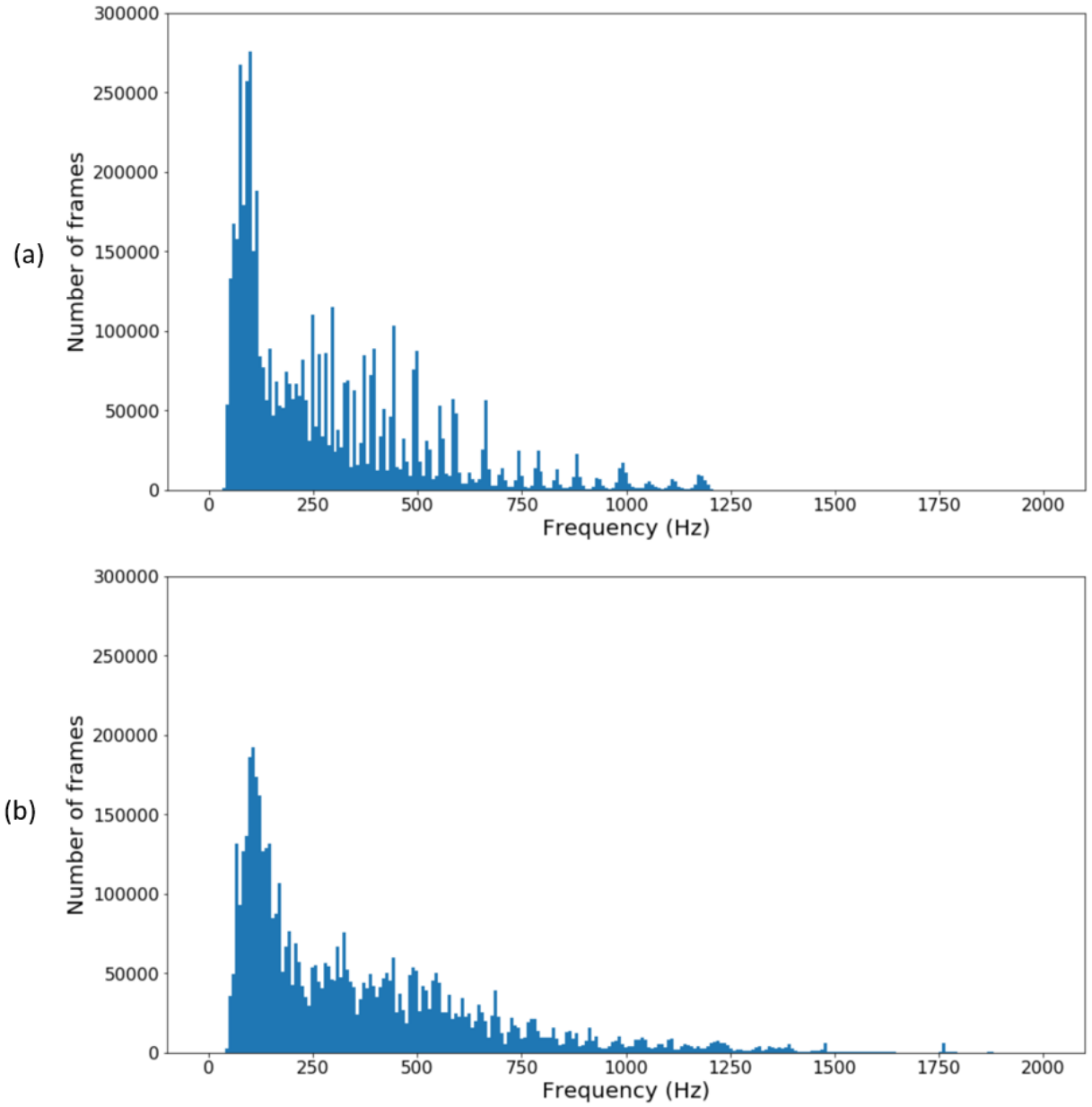


Figure 4.10: Distribution of frequency annotations in the training set. **(a)** Before pitch shift. **(b)** After pitch shift.

The lowest and the highest value are the same as in the previous version:

$$\omega_{\dot{\text{C}},720}(1) = \omega_{\dot{\text{C}}}(1) \text{ and } \omega_{\dot{\text{C}},720}(720) = \omega_{\dot{\text{C}}}(360). \tag{4.12}$$

But with a range of 6 octaves (same as before), twelve semitones per octave and 10 values per semitone ($6 \cdot 12 \cdot 10 = 720$), we obtain a frequency resolution of 10 cents instead of 20 cents. For the computation of the final frequency estimate, we use the same formula as defined in

Equation 4.5. So we identify the index of the maximum value in the output vector $\omega_{\hat{c},720}$ and only consider values in a neighborhood of 4 before and after this index for the computation. However, these values correspond to frequencies $\pm(4 \cdot 10)$ cents due to the increased resolution. Using the original resolution, they corresponded to frequencies $\pm(4 \cdot 20)$ cents. The trained network with an increased dimension of the last layer is referred to as M_{dim720}.

4.2.3 Layer Freezing

A pretrained version of CREPE is published with the filter weights for all model capacities. To investigate how the input data is processed through the network, we examined the weights of the filters. Here, the focus is on the weights of the first convolutional layer, because this gives insights in how the input is transformed before it is further evaluated in the network. The first layer of the tiny model structure has 128 layers each with a dimension of 512. Some typical filters are shown in Figure 4.11. It can be recognized that some of the layers have a periodic

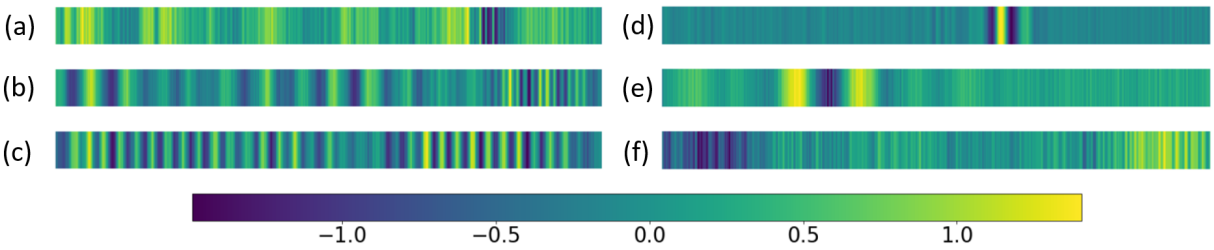


Figure 4.11: Example filters from the first convolutional layer of the pretrained version of CREPE (M_{tiny}).

structure, e.g. Figure 4.11a, Figure 4.11b and Figure 4.11c (with increasing frequency). This means that the filters produce high values if they are applied to an array which has high values with the same distance as the peaks of the layer. So a periodic filter measures the similarity between the periodic structure of the signal and its own periodic structure. This indicates that the functionality of the first layer is similar to a Fourier analysis which also compares a signal with sinusoids of different frequencies. In addition to the periodic filters, there are also filters with one peak (e.g. filter shown in Figure 4.11d) or multiple not periodic peaks (e.g. filter shown in Figure 4.11e). Furthermore, there are filters that have another structure, e.g. the filter shown in Figure 4.11f. The role of these filters for the network is still open.

Since the filters show some periodicity, we further examined them by applying fast Fourier transformation (fast algorithm for the computation of the DFT) to each filter. Next, the absolute value of each transformed filter was computed. The resulting vectors were then sorted ascendingly according to the index of their maximal value. The result is shown in Figure 4.12. In this figure, there is an exponential curve visible starting at 0 Hertz and ending at approximately 2000 Hertz. The maximal value for each filter (each column) indicates on which frequencies the

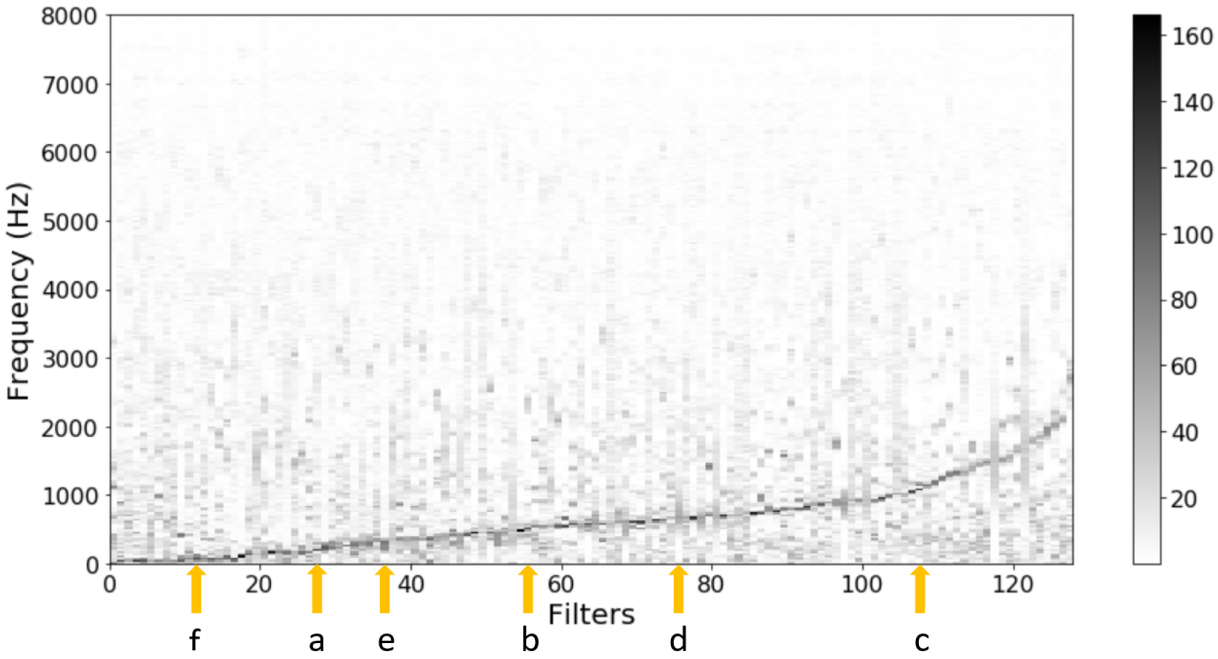


Figure 4.12: To the filters of the first layer of the original network M_{tiny}, Fourier transform was applied. Each column corresponds to one filter. The columns are sorted ascendingly according to the indices of the maximum values. The orange arrows indicate the positions of the filters shown in Figure 4.11.

filter focuses. There are more filters focusing on lower frequencies. This result is reasonable, because the perception of pitch is logarithmic in frequencies. The difference in Hertz is smaller for lower pitches. So in a lower frequency range, more pitches are present and more filters are necessary to distinguish the pitch values.

The idea of layer freezing was now to precompute weights for a layer and use these for training. Here, we examined the functionality of the filters of the first convolutional layer and used the results to fix weights for these filters. As already discussed, the Fourier transformations of the filters form an exponential curve between 0 Hertz and 2000 Hertz. So the filter with the highest frequency contains 64 oscillations in one filter length (512 samples, 32 milliseconds). This filter can be created by a sinus from 0 to 128 π. Since $e^{\frac{128}{26.3807}} = 128$ the filters can be created as follows: For each filter create an array with linear values between 0 and $\pi \cdot e^{\frac{\text{filter index}}{26.3807}}$, then compute the sinus of each value. The Fourier transform of the computed filters is shown in Figure 4.13. These filters are then fixed before training the neural network. The weights of the first convolutional layer are set to be not trainable. The bias of the layer (128 dimensional) is set to 0. We refer to the network trained with precomputed weights for the first convolutional layer as M_{freezed}.

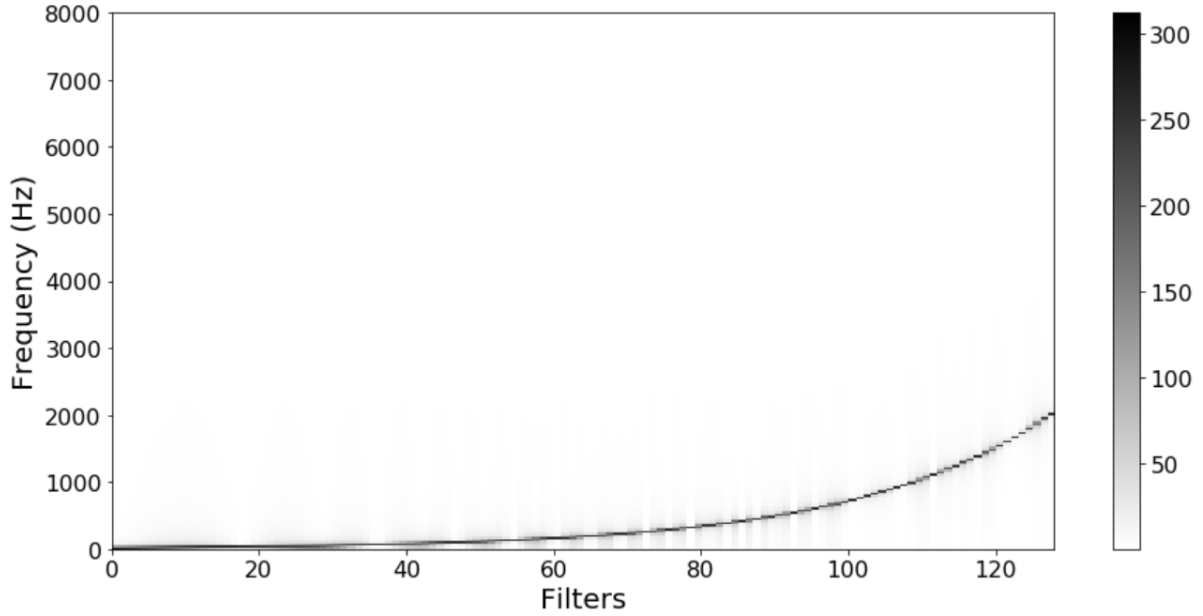


Figure 4.13: Spectrogram of the filters fixed for the first convolutional layer. These filters are used for M_{freezed}.

	M_{reduced}	M_{tiny}	M_{small}	M_{medium}	M_{large}	M_{full}
trainable parameters	119,672	486,552	1,628,104	5,877,288	12,747,912	22,239,976

Table 4.3: Number of trainable parameters for the models provided by CREPE and the new model.

4.2.4 Modify Network Structure

The tiny model of CREPE M_{tiny} still has over 486000 trainable parameters (486552). In order to decrease the number of trainable parameters, we developed a new model which has only 24.6 % of trainable parameters compared to the tiny model. The objective was to examine the performance of a smaller model. Training this model was expected to be faster, since less parameters need to be trained. For a comparison of the number of trainable parameters depending on the model sizes, see Table 4.3.

The number of filters in the original tiny network M_{tiny} is high for the first convolutional layer. So instead of 128 filters, we use only 64 filters. The last convolutional layer of the tiny model uses 64 filters. For the new model, this was decreased to 32 layers. We also reduced the filter sizes. Before, the sizes were 512 (convolution layer 1) or 64 (all other convolutional layers). For the new model, we use filter sizes of 32 (convolution layer 1) or 8 (all other convolutional layers).

Especially using filters of size 32 instead of 512 in the first convolutional layer is a remarkable

modification. Each filter of the new model uses only 32 neighboring samples (corresponds to 2 milliseconds) of the input audio for one application of the filter.

In the last convolutional block, no MaxPooling is applied, because the output dimension of the convolutional layer is (8,32), so it contains 256 values. With MaxPooling with stride 2, the number of output values would be reduced to 128. The output vector of the neural network has a dimension of 360, so the number of dimensions before the fully connected layer can be increased by omitting the MaxPooling layer. For the architecture of the new model, see Figure 4.14.

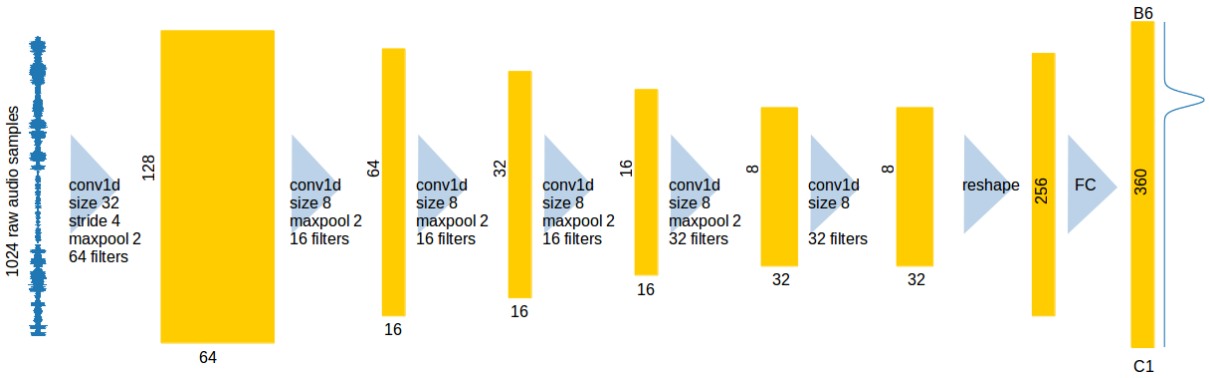


Figure 4.14: Architecture of new model structure M_{reduced} with less trainable parameters.

We refer to the trained version of the model with reduced trainable parameters as M_{reduced}.

Chapter 5

Evaluation on MDB-stem-synth

In this chapter, the MDB-stem-synth dataset [8] (see Section 2.3.2 for details) is used for evaluation of DNN-based pitch estimation approaches. Here, in comparison to previous chapters, the results over the complete dataset are analyzed instead of single audio recordings. This leads to a less detailed view, but more generalizable results. First, we evaluated the original network of CREPE [1] in Section 5.1, then we retrained the network from scratch and developed some own approaches in Section 5.2.

The developers of CREPE published the network structure and the model weights for the five different model capacities. Also the code which is necessary to apply CREPE to music recordings is published. The code for the training procedures is not available and was implemented by us. For the evaluation, we computed raw pitch accuracies (RPA) and raw chroma accuracies (RCA) using the python library mir_eval [38].

5.1 CREPE

Salamon et al. trained the network on multiple datasets (see Section 4.1) among them the MDB-stem-synth dataset [8]. The weights of the trained filters are published online (e.g. `https://github.com/marl/crepe/raw/models/model-tiny.h5.bz2` for the tiny model). For evaluating the raw pitch accuracy and the raw chroma accuracy of a batch of audio recordings, there are three possibilities:

- **Eval1** (same weight for all files):
  Compute a RPA value for each file, get the mean RPA value by averaging over the values.

- **Eval2** (weight depending on file length):
  Compute a RPA value for each file, get the mean RPA value by weighting the single values by the file length and taking the average.

- **Eval3** (weight depending on number of voiced frames in reference):
  Compute a RPA value for each file, get the mean RPA value by weighting the single values by the number of voiced frames in the reference and taking the average.

We evaluated the complete MDB-stem-synth dataset using the version of CREPE published by the authors. For this, we used the full model (M_{full}, highest number of filters in each layer) with a stepsize of 10 milliseconds (160 samples). For the results of the raw pitch accuracies using the three different evaluation methods and thresholds of 10/25/50 cents, see Table 5.1. The comparison of the evaluation methods shows that the resulting values for the raw pitch

tolerance	Eval1	Eval2	Eval3	Kim et al. [1]
10 cents	0.839	0.878	0.868	0.909
25 cents	0.926	0.949	0.946	0.953
50 cents	0.952	0.969	0.970	0.967

Table 5.1: We computed the RPA values for the complete MDB-stem-synth dataset using the original network (full model M_{full}, step size 10 milliseconds) with different aggregation methods. The last column shows the results reported by Kim et al. [1] (evaluation method unknown).

accuracies differ by up to 0.039 (0.839 for a tolerance of 10 cents and Eval1 and 0.878 for a tolerance of 10 cents and Eval2). In this case, evaluation method Eval2 leads to higher accuracies then Eval1. This indicates that there are for example long files with high accuracies (higher weight in evaluation method Eval2) or there are short files with low accuracies (lower weight in evaluation method Eval2). Also evaluation method Eval3 returns higher pitch accuracies than Eval1. So there are either files with a low number of voiced frames and low accuracies or files with a high number of voiced frames and high accuracies. The accuracies reported by Kim et al. [1] are higher for thresholds of 10 and 25 cents than our results. When compared to the evaluation method that returns the highest accuracies, the differences are 0.031 (10 cents) and 0.004 (25 cents). For a threshold of 50 cents, we obtained a higher accuracy than reported by Kim et al. using evaluation method Eval2.

So the results of evaluating a system on a dataset with multiple files vary depending on the type of weighting which is used for aggregating the results of the single files.

In the following we use the evaluation method Eval1 (same weight for all files).

5.2 Own Approaches

First, we retrained the network structure from scratch to examine the training process of the network. Then, to avoid a bias induced by the choice of the test set, 5-fold cross-validation was used. Afterwards, in order to further investigate the network, we evaluated the approaches presented in Section 4.2 using the MDB-stem-synth dataset [8]. We started with data augmentation in order to improve training results when training with a biased or too small dataset. Then, we examined the influence of the first layer of the network and replaced it by a filter with precomputed weights. Furthermore, we investigated a modified network with an adjusted output dimension. Finally, we evaluated a network with modified layers.

When presenting the training of a neural network in the following sections, we used for most cases a fixed training/validation/test split of the MDB-stem-synth dataset. We created this split by hand and it is designed so that all solo stems (tracks) of one musical piece are in the same set. The training set consists of 60% of the available stems, the other two sets consist of 20% of the stems each. Training was performed on the voiced frames only, because the network needs audio excerpts with annotation vectors. If a frame is unvoiced, no expected annotation vector can be computed. Therefore, training is only possible with voiced frames. In MDB-stem-synth, approximately 44.8% of the annotated frames are voiced.

5.2.1 Retrained Network

To get a first understanding of the network, we retrained the network from scratch. For this, we used the network structure as proposed by Kim et al. [1], but instead of the pretrained weights, we started with random weights. For the training, we used the architecture of the tiny model M_{tiny}. The training/validation/test split was created by hand as described above. We trained the network for 800 epochs. Each epoch consists of 30 training batches and 20 validation batches. One batch includes 32 samples. As suggested by the authors, we used the Adam optimizer [34] with a learning rate of 0.0002 and the binary cross entropy loss defined in Equation 4.10. Retraining the network took approximately 161 minutes (approximately 2.69 hours).

During training, the raw pitch accuracies for the training and validation were computed after each epoch. This was done by computing the raw pitch accuracy for each training and each validation batch and averaging the results of all batches in one epoch. For the resulting curves see Figure 5.1. As expected, the raw pitch accuracy generally increased over the epochs. For a lower threshold of 10 cents, the accuracy increased slowlier than for a high threshold of 50 cents. A high increase in accuracy can be observed in the first 200 epochs, afterwards, the accuracies increased slowlier. For later epochs (epoch 200 to 800), the increase of accuracy is higher for a threshold of 10 cents than for a threshold of 50 cents. So it can be assumed that the network

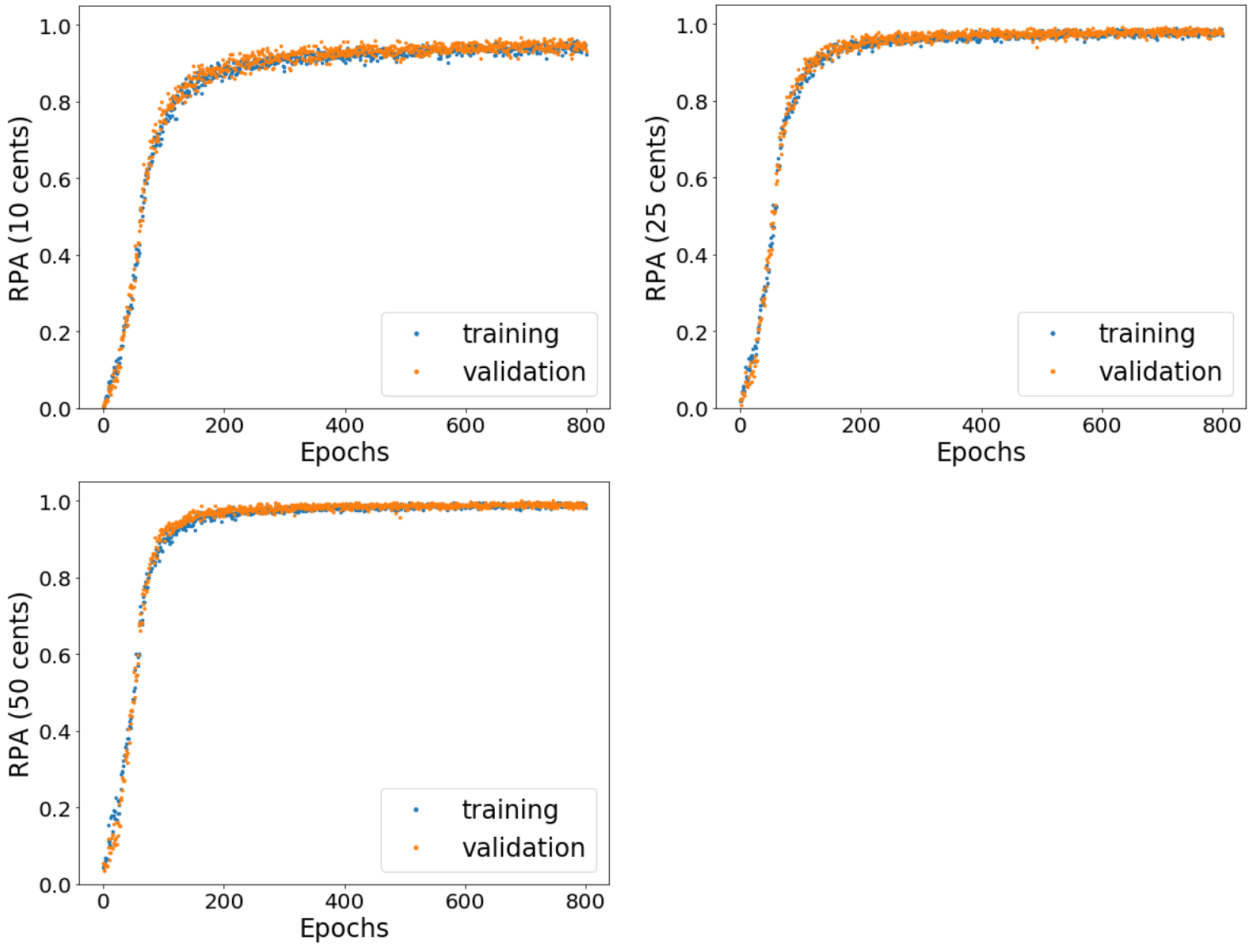


Figure 5.1: During the retraining of the original version of CREPE (architecture of M_{tiny}, starting with random weights), the raw pitch accuracies for different thresholds were measured after each training/validation batch. These results were averaged over all training/valiation batches in one epoch.

identified a solution which approximately estimates the pitches correctly at the beginning of the training (epoch 0 to approximately 150). However, later epochs were necessary to refine the estimations and thereby increase the raw pitch accuracy for low thresholds.

It can also be observed that generally the validation accuracy is higher than the training accuracy. For this, there are multiple possible explanations. The training/validation set could be biased (samples in validation loss were easier to estimate). Also when using regularization, a lower training accuracy is reasonable, because dropout layers [32] are only activated during training and can increase the error.

After the network was trained, it was applied to the test set. The test set only contains audio files that are not used for training or validation. So the evaluation on the test set is not biased due to the training process. As for the original version of CREPE, all files were resampled to a

sampling rate of 16000 Hz before the network was applied. The network was configured to use a window size of 1024 samples and a step size of 10 milliseconds. We computed the raw pitch accuracy for each file in the test set with thresholds of 10, 25 and 50 cents. These values were summarized using method Eval1 (see Section 5.1). For the resulting raw pitch accuracies and raw chroma accuracies see Table 5.2. The table shows that high accuracies can be achieved when

tolerance \ measure	RPA	RCA
10 cents	0.826 ± 0.136	0.826 ± 0.136
25 cents	0.955 ± 0.042	0.956 ± 0.042
50 cents	0.982 ± 0.019	0.982 ± 0.019

Table 5.2: We trained a model with the same architecture as M_{tiny} starting with random weights. The trained network was then applied to the files in the test set. Using evaluation method Eval1 these results were aggregated in order to obtain RPA and RCA values.

using a threshold of 50 cents (or 25 cents). For a threshold of 10 cents, the accuracies are lower. It is also remarkable that there are no significant differences between the raw pitch accuracies and the raw chroma accuracies. This indicates that the retrained model does not predict wrong pitches because of octave errors.

For comparison, we evaluated the original version of CREPE as published by Kim et al. on the test set (resampled by CREPE to 16000 Hz) using the tiny model M_{tiny} and a step size of 10 milliseconds (same as for the retrained network). Furthermore, YIN was evaluated on the test set. YIN used the original files with a sampling rate of 44100 Hertz. The evaluation was performed with a hop size of 512 samples (approximately 11.6 milliseconds, CREPE: 10 milliseconds) and a window length of 4096 samples (approximately 92.9 milliseconds, CREPE: 64 milliseconds). The minimum frequency is set to 30 Hertz and the maximum frequency is set to 1760 Hertz.

Table 5.3 shows the raw pitch accuracies of the retrained network, the original network and YIN. It is notable that compared to the original version of CREPE, the retrained network

tolerance	RPA (M_{tiny})	RPA (retrained CREPE)	RPA (YIN)
10 cents	0.837 ± 0.091	0.826 ± 0.136	0.630 ± 0.154
25 cents	0.942 ± 0.043	0.955 ± 0.042	0.797 ± 0.104
50 cents	0.969 ± 0.027	0.982 ± 0.019	0.881 ± 0.069

Table 5.3: RPA values for the test set evaluated using method Eval1. The first column shows the results of the original version of CREPE (M_{tiny}). The second column shows the results of the retrained model (same architecture as M_{tiny}). The third column shows the result of YIN.

achieves a higher RPA for a tolerance of 25 and 50 cents, but a lower RPA for a tolerance of 10 cents. This shows that the retrained network performs worse for exact estimations, but better for approximate estimations. The results for tolerances of 25 and 50 cents can be due to the restriction to only one dataset. So the training and testing is performed on sets of the same

dataset and the model can overfit on characteristics of this dataset. The low RPA for a tolerance of 10 cents can be explained by involving the RPA curves. The RPA values during training were still increasing for epochs 700 to 800, so further improvements for this tolerance are possible when training for more than 800 epochs.

In summary, we showed that a retraining of the network is possible in comparatively short time and achieves raw pitch accuracies similar to the original network.

5.2.2 Cross-Validation

The results in Section 5.2.1 highly depend on the train/validation/test split. If for example, the test set contains examples that are comparatively easy to evaluate, this leads to higher raw pitch accuracies.

So in order to obtain a less biased result, we applied cross-validation. Here, 5-fold cross-validation is used. So the complete dataset was split into five groups. Each of the groups consists of the same number of stems and each stem can only be part of one group. Then we performed training, validation and testing in five independent iterations. Each time, one of the five groups (20% of the stems) served as test set. The stems that are not in the test set were split into a validation set (20% of all stems) and a training set (60% of all stems). We created the training/validation/test sets so that all stems of one musical piece are assigned to the same set. This is necessary in order to avoid training and testing on the same musical piece, because this would lead to too high RPA results for the test set.

For creating the five groups and for splitting the rest of the stems into a validation/training set, we used backtracking. So the splitting algorithm starts with one file and adds more files as long as all conditions are fulfilled. If a suitable set is created, the algorithm returns this set as suggested split. Otherwise, the algorithm recursively removes files from the set and replaces them by other files.

We used the model architecture from the tiny model of CREPE (M_{tiny}) with random weights as starting point. Training was performed with each group for 800 epochs ($5 \cdot 800$ epochs in total). Each epoch consists of 30 batches for training and 20 batches for evaluation. The batch size is 32. The training process for all five groups took approximately 933 minutes (15.55 hours). Divided by 5 (5 folds), this results in approximately 3.11 hours per fold. This duration is longer than the duration of the retraining in Section 5.2.1 (2.69 hours). The reason is that here we included the evaluation on the test set in the process. The test set for 5-fold cross-validation is determined during the process, so the additional time is necessary to evaluate the files in the test set. As suggested by the authors, we used the Adam optimizer [34], a learning rate of 0.0002 and the binary cross-entropy loss (Equation 4.10). In each iteration, we trained a network and then evaluated it on the test set with a step size of 10 milliseconds and the evaluation method

Eval1. The results of the RPA values of the five folds are shown in Table 5.4. The accuracies

tolerance	fold1	fold2	fold3	fold4	fold5	mean of folds	M_{tiny}
10 cents	0.770	0.829	0.768	0.791	0.815	0.795	0.837
25 cents	0.899	0.947	0.912	0.928	0.953	0.928	0.942
50 cents	0.932	0.971	0.947	0.961	0.978	0.958	0.969

Table 5.4: Using 5-fold cross-validation, we trained a network with the same architecture as M_{tiny} five times. Each model was then applied to the test set with a step size of 10 milliseconds. The raw pitch accuracy for each file in the test set was computed and the results of the files were aggregated using method Eval1. The table shows the raw pitch accuracies for the five training iterations. For comparison, we added the results of the original version of CREPE (M_{tiny}, Eval1) applied to the test set.

for fold1 and fold3 are lower than the mean accuracy. This can indicate that the test sets used for these folds are more challenging than the test sets of the other folds. With fold2 and fold5, accuracies above the mean accuracy are obtained and fold 4 leads to results similar to the mean accuracy. The mean accuracies are lower than the accuracies achieved by applying the original version of CREPE to the test set. This is possibly due to the short training time.

The mean accuracy achieved using 5-fold cross-validation is a more general result since it does not depend on the train/validation/test split.

5.2.3 Data Augmentation

As explained in Section 4.2.1, we used data augmentation in order to improve the training results. The MDB-stem-synth dataset [8] contains more examples for low frequencies than for high frequencies. Therefore, we applied pitch shifting to the audio files in the training and the validation set, so that these sets cover a larger frequency range. This is expected to improve the performance of the network for high frequencies.

For the training, we used the architecture of the tiny network M_{tiny}. The network was initialized with random weights. For the training process, we used the Adam optimizer [34], a learning rate of 0.0002 and the binary cross-entropy loss defined in Equation 4.10. The training was performed for 800 epochs where each epoch consisted of 30 batches for training and 20 batches for validation. The batch size was 32, so per batch, 32 samples were used. We refer to the trained network as M_{shift}. The complete training took 156 minutes (approximately 2.61 hours). This duration is similar to the duration for retraining the network without data augmentation (see Section 5.2.1, training duration was 2.69 h). During training, we computed the raw pitch accuracies. This was done after each epoch for the training and the validation batches. We measured the raw pitch accuracy for thresholds of 10, 25 and 50 cents. As expected, the raw pitch accuracies increased over the epochs. For higher thresholds (e.g. 50 cents), the raw pitch accuracy increased faster

then for lower thresholds, since the network gets more accurate over training time. However, it is remarkable that especially for a threshold of 10 cents, the raw pitch accuracies are rather small. For the raw pitch accuracies over the epochs, see Figure 5.2. The new trained network M_{shift}

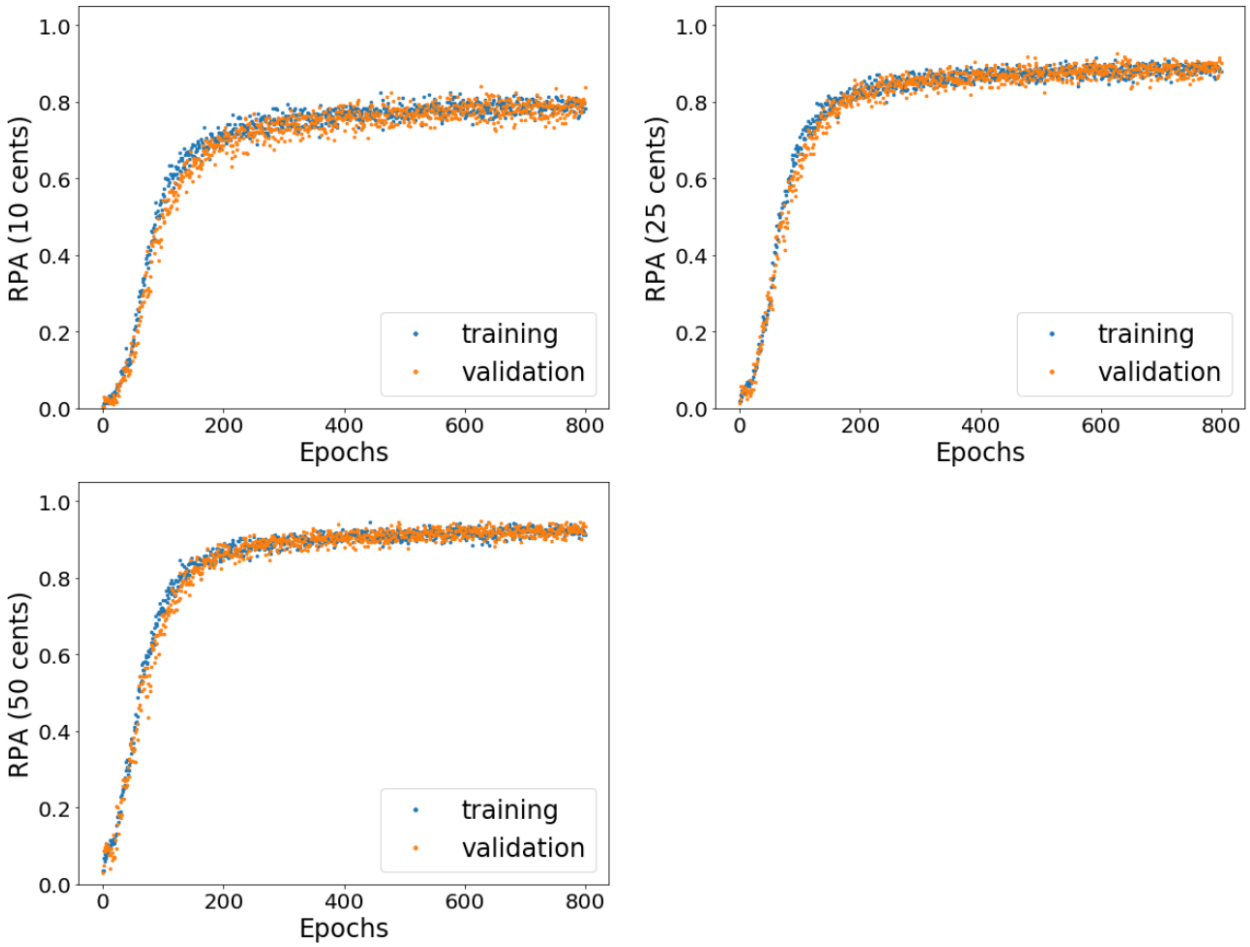


Figure 5.2: We trained a model with the same architecture as the architecture of the tiny model M_{tiny} using pitch shifting. The resulting model is referred to as M_{shift}. The figure shows the raw pitch accuracy (RPA) for the training and validation batches after each epoch for different thresholds.

was then applied to the test set consisting of 20% of the audio files of MDB-stem-synth. These audio files were not used during training. No pitch shifting is applied to the test set to ensure comparability to other approaches (same test set). The evaluation of the new trained network on the test set lead to the RPA/RCA values shown in Table 5.5 (evaluated with method Eval1).

The accuracies are lower then the accuracies achieved with the original version of CREPE (M_{tiny}). However, we applied pitch shifting in order to improve the performance for high frequencies. To examine the frequency range, the network trained with pitch shifting was applied to the chirp signal with frequencies between 30 and 2000 Hertz. For comparison, also the retrained network

measure / tolerance	RPA (M_{shift})	RCA (M_{shift})	RPA (M_{tiny})
10 cents	0.649 ± 0.118	0.667 ± 0.106	0.837 ± 0.091
25 cents	0.871 ± 0.105	0.894 ± 0.082	0.942 ± 0.043
50 cents	0.927 ± 0.081	0.953 ± 0.043	0.969 ± 0.027

Table 5.5: To the training and the validation set, pitch shifting was applied. After training a model with the architecture of M_{tiny} using pitch shifting, we evaluated the test set using the new network M_{shift} and computed RPA and RCA values (aggregated using Eval1). The raw pitch accuracies and raw chroma accuracies are shown in the first and second column. For comparison, we added the results of the original version of CREPE (M_{tiny}, Eval1) evaluated on the test set.

from Section 5.2.1 was applied to the same signal. The estimations of the retrained network and the network trained with data augmentation are shown in Figure 5.3. This figure shows that the network without data augmentation estimates frequencies below 1200 Hertz correctly. For higher frequencies, the network predicts the pitches of the subharmonics. Using data augmentation, the network recognizes frequencies until approximately 1600 Hz with a high accuracy. Also for higher frequencies, subharmonics are predicted. So the learned frequency range increased compared to a network trained on the original MDB-stem-synth dataset by approximately 400 cents. However, not the complete frequency range until 2000 Hz is learned. This is probably due to the still not completely balanced frequency distribution.

5.2.4 Output Vector Resolution

Another approach was to modify the output vector. The original version of CREPE outputs a 360-dimensional vector from which the final pitch estimate is computed. The values in the vector have a resolution of 20 cents. Here, we replaced this vector by a 720 dimensional vector which covers the same frequency range, but has a resolution of 10 cents. For training, we used the same train/validation/test set that was created by hand as before. The network was trained using the Adam optimizer [34] and a learning rate of 0.0002. For the loss function, we used the binary cross-entropy loss defined in Equation 4.10. We trained for 800 epochs (30 training batches/20 validation batches per epoch, batch size 32). The complete training process took approximately 220 minutes (3.66 hours). We then applied the trained network (referred to as M_{dim720}) to the audio files of the test set and computed the raw pitch accuracy for each file. From these values, the resulting raw pitch accuracies were derived using evaluation method Eval1, see Table 5.6 for the results. The raw pitch accuracy obtained with the new network M_{dim720} is lower than the RPA obtained with the original for a threshold of 10 cents. For thresholds of 25 cents and 50 cents, the raw pitch accuracy of M_{dim720} is higher. However, all raw pitch accuracies are lower compared to the retrained version of CREPE from Section 5.2.1. This indicates that the increased output dimension of the network does not improve the accuracy for low thresholds.

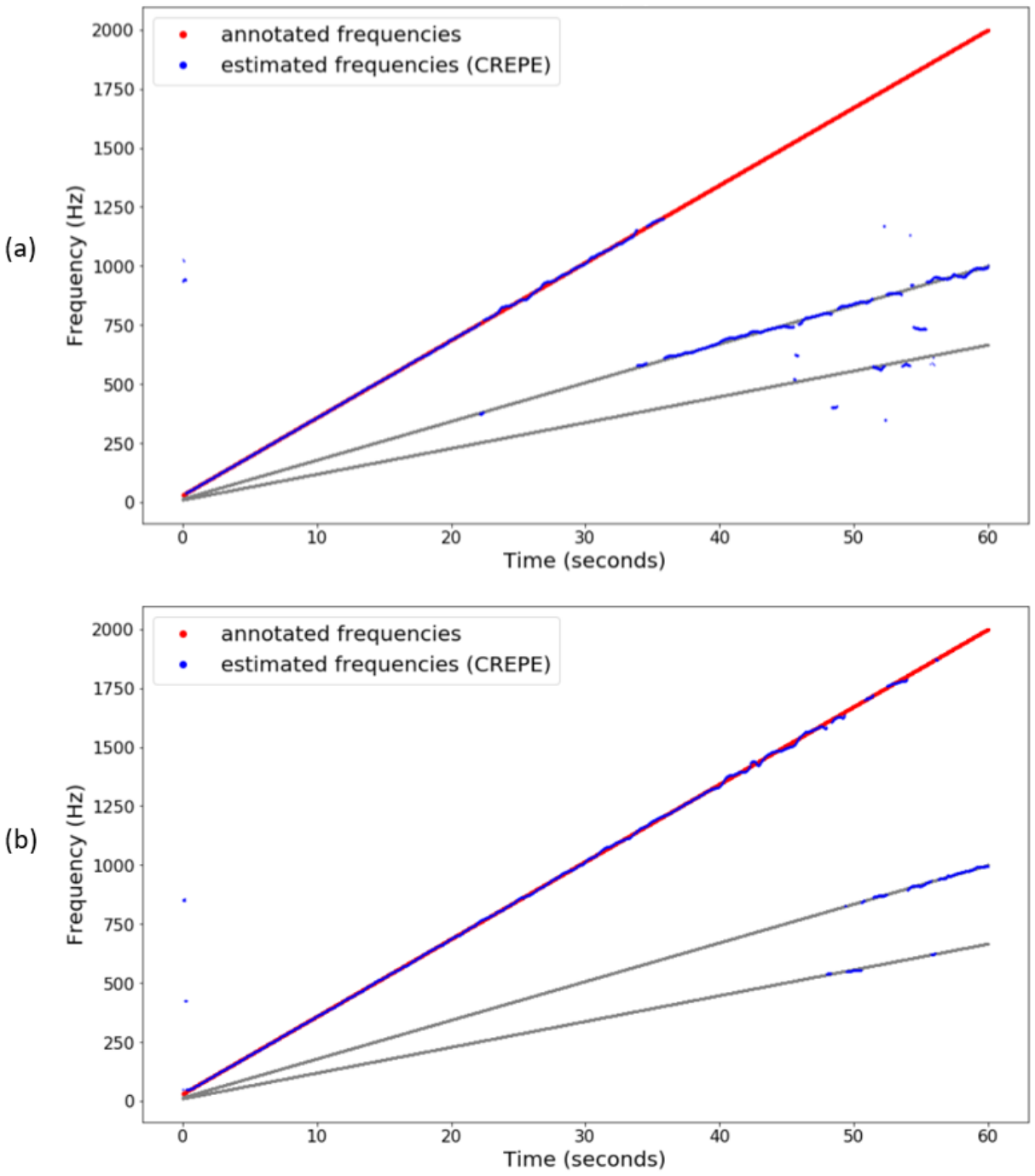


Figure 5.3: F0-estimations of a chirp signal from 30 Hertz to 2000 Hertz. **(a)** Estimations using the retrained network from Section 5.2.1. Frequencies until approximately 1200 Hertz are predicted correctly. **(b)** Estimations using a new network M_{shift} trained with a pitch shifted dataset. Frequencies until approximately 1600 Hertz are predicted correctly. For higher frequencies, the subharmonics are predicted.

tolerance \ measure	RPA (M_{dim720})	RCA (M_{dim720})	RPA (M_{tiny})
10 cents	0.778 ± 0.152	0.778 ± 0.152	0.837 ± 0.091
25 cents	0.945 ± 0.052	0.945 ± 0.052	0.942 ± 0.043
50 cents	0.978 ± 0.023	0.979 ± 0.023	0.969 ± 0.027

Table 5.6: We trained a new network based on the architecture of the tiny model of CREPE (M_{tiny}). The new network M_{dim720} has an increased output resolution. After training, the test set was evaluated using the new network and evaluation method Eval1. The resulting raw pitch accuracies and raw chroma accuracies are shown in the first and second column. For comparison, we added the results of the original version of CREPE (M_{tiny}, Eval1) evaluated on the test set.

One possible explanation is that the training of the network is slower due to the higher number of parameters. It is also possible that the number of potential outputs represented by the values in the last layer is too high compared to the available training data.

5.2.5 Layer Freezing

In Chapter 4.2.3, we examined the filters of the first convolutional neural network of CREPE (M_{tiny}). The weights of some filters have a periodic structure. So we developed filters for the first convolutinal layer so that the Fourier transformation of the filters form an exponential curve when sorted for the maximum index. The tiny model M_{tiny} of CREPE has 128 filters with a filter size of 512 in the first convolutional layer. We used the original network structure of CREPE (M_{tiny}) but replaced the weights with random weights. For the first convolutional layer, we fixed the precomputed weights. This network was trained using the fixed train/validation/test set. As optimizer, we used the Adam optimizer [34] and the learning rate was set to 0.0002. The loss function was the binary cross-entropy loss (defined in Equation 4.10). For the training, we used a batch size of 32 samples. In each epoch, we trained with 30 batches and evaluated using 20 batches. Training was performed for 800 epochs and the complete training took 152 minutes (2.54 hours). In comparison to a retraining of the network without a fixed first layer (Section 5.2.1), this duration is 9 minutes shorter. It is expected that the training is faster, since no training of the first layer is necessary.

After training the network (referred to as M_{freezed}), we applied it to the files in the test set. For each file, we computed a raw pitch accuracy and a raw chroma accuracy. These values were summarized using method Eval1. For the results, see Table 5.7. The table indicates that the raw pitch accuracies are smaller for a threshold of 10 cents and 25 cents and similar for a threshold of 50 cents compared to the original version of CREPE (M_{tiny}). There are mainly two possible reasons for this result. One explanation is that the Fourier transform of the filters from the original network do not form a perfect exponential curve. Rather the curve ascends faster at the beginning. A large number of filters focus on medium frequencies of approximately 300 Hz to

measure / tolerance	RPA (M_{freezed})	RCA (M_{freezed})	RPA (M_{tiny})
10 cents	0.746 ± 0.129	0.749 ± 0.130	0.837 ± 0.091
25 cents	0.928 ± 0.058	0.934 ± 0.056	0.942 ± 0.043
50 cents	0.966 ± 0.035	0.972 ± 0.029	0.969 ± 0.027

Table 5.7: We trained a new network based on the architecture of the tiny model of CREPE (M_{tiny}). The weights of the first convolutional layer were precomputed and not trained. After training, the test set was evaluated using the new network M_{freezed} and evaluation method Eval1. The resulting raw pitch accuracies and raw chroma accuracies are shown in the first and second column. For comparison, we added the results of the original version of CREPE (M_{tiny}, Eval1) evaluated on the test set.

1000 Hz. The exponential curve in contrast ascends slowlier at the beginning. So the number of precomputed filters for a low frequency range could be too high which corresponds to a too small number of filters focusing on medium frequencies that occur frequently in most datasets. Another possible explanation for the low raw pitch accuracies is the concentration on the maximum of the Fourier transform of each filter. It might be that each filter focuses on more than one frequency and additionally measures the presence of higher frequencies in an audio excerpt. This can be necessary in order to use overtones for the estimation of the fundamental frequency. By generating filters which consist of only one frequency, the noise in the Fourier transform is avoided. This possibly leads to lower accuracies.

For this experiment, we generated the filters using sine waves with the same phase for all filters. Additionally, we experimented with the usage of sine waves and cosine waves. For this purpose, we generated filters with ascending frequencies similar to the first experiment. But the filters were generated alternating as sine wave or cosine wave. The training was performed similar to the training for M_{freezed} on the training set and the validation set and the duration was 162 minutes (2.35 hours). We refer to the trained model as M_{freezed2}. The resulting raw pitch accuracies for the test set (computed with Eval1) are shown in Table 5.8. The second model results in only slightly better raw pitch accuracies for thresholds of 10 and 25 cents.

measure / tolerance	RPA (M_{freezed})	RPA (M_{freezed2})
10 cents	0.746 ± 0.129	0.790 ± 0.120
25 cents	0.928 ± 0.058	0.935 ± 0.061
50 cents	0.966 ± 0.035	0.965 ± 0.044

Table 5.8: We trained two networks with a fixed first layer. For the first network M_{freezed}, the filters were generated using sine waves. For the filters of the second network M_{freezed2} we used sine waves and cosine waves. Both networks were evaluated on the test set using evaluation method Eval1.

5.2.6 Modify Network Structure

As presented in Section 4.2.4, we developed a new network with a lower number of filters and smaller filters. This leads to a reduced number of trainable parameters. Compared to the tiny model of CREPE M_{tiny}, this network has only 24.6% of the trainable parameters. The new network M_{reduced} was trained using the fixed train/validation/test split that was created by hand. For training, we used the Adam optimizer [34], a learning rate of 0.0002 and the binary cross-entropy loss (defined in Equation 4.10). We trained for 800 epochs with 30 training batches and 20 evaluation batches per epoch and a batch size of 32. The training process took 149 minutes (2.48 hours). So the training is faster than the training of the tiny model structure of CREPE (duration: 161 minutes). Additionally, the training of this model is faster than the training of a model with precomputed filters for the first layer (duration: 152 minutes). After the training, the network was evaluated on the test set. For the results of the raw pitch accuracy and the raw chroma accuracy see Table 5.9. It is remarkable that the raw pitch accuracy is significantly lower for a threshold of 10 cents compared to the raw pitch accuracy of the original version of CREPE. However, the raw pitch accuracy for a threshold of 50 cents is only slightly lower for M_{reduced} compared to the original version of CREPE (M_{tiny}).

measure / tolerance	RPA (M_{reduced})	RCA (M_{reduced})	RPA (M_{tiny})
10 cents	0.590 ± 0.119	0.591 ± 0.119	0.837 ± 0.091
25 cents	0.893 ± 0.086	0.894 ± 0.085	0.942 ± 0.043
50 cents	0.958 ± 0.051	0.959 ± 0.050	0.969 ± 0.027

Table 5.9: Raw pitch accuracies and raw chroma accuracies for the test set evaluated with a network with less parameters M_{reduced} and evaluation method Eval1. For comparison, we added the results of the original version of CREPE (M_{tiny}, Eval1) evaluated on the test set.

We applied the new trained network M_{reduced} to a chirp signal (30 Hz to 2000 Hz). For the estimated frequencies see Figure 5.4. Similar to the retrained network based on the tiny model structure of CREPE, subharmonics are predicted for frequencies above approximately 1200 Hz. For frequencies below 1200 Hz, the estimations of M_{reduced} are rather inaccurate. This is probably due to the small network that is not capable of learning a detailed F0-estimation.

The new network M_{reduced} has only 64 filters instead of 128 filters (M_{tiny}) for the first convolutional layer. Each filter has a dimension of 32 (M_{reduced}) instead of 512 (M_{tiny}). To the learned filters of M_{reduced}, we applied the Fourier transform and sorted the result ascendingly according to the index of the maximum value. For the result, see Figure 5.5. The figure still slightly shows a curve similar to the Fourier transform of the filters from the original network M_{tiny}. So the network M_{reduced} has a lower number of trainable parameters and training can be performed faster. However, this also leads to a lower raw pitch accuracy for a threshold of 10 cents.

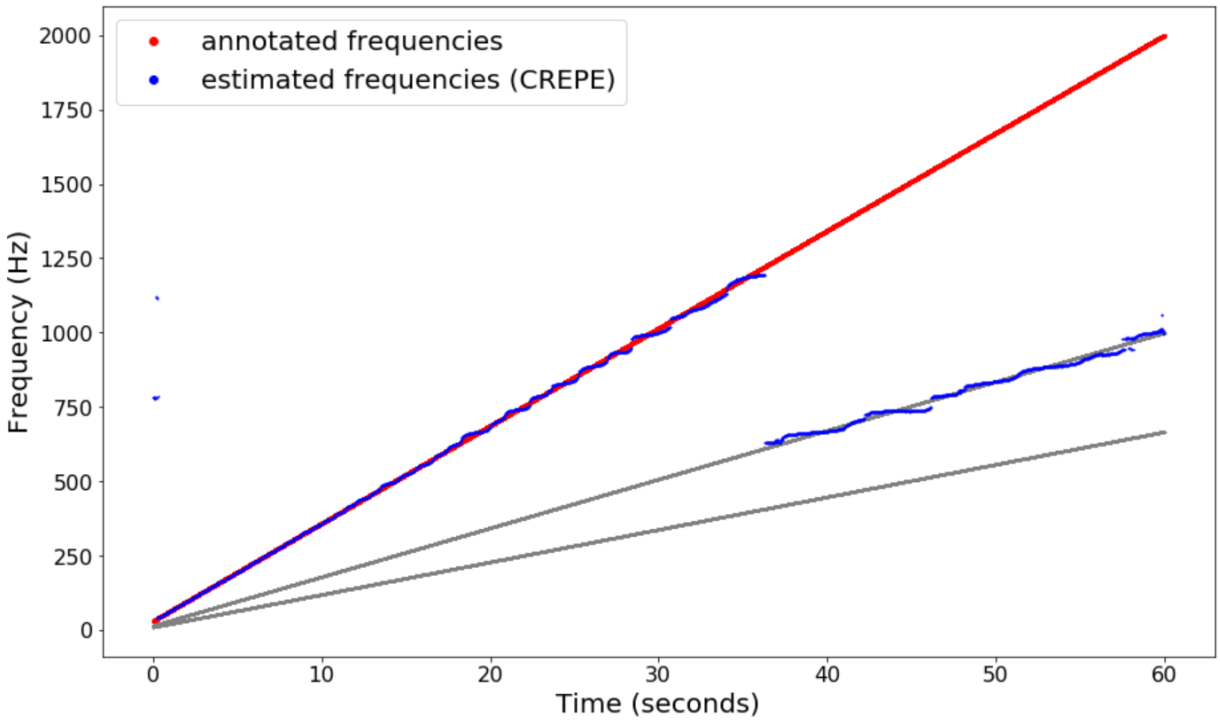


Figure 5.4: We trained a model with a reduced number of parameters. The figure shows the F0-estimations of this model M_{reduced} for a chirp signal from 30 Hertz to 2000 Hertz.

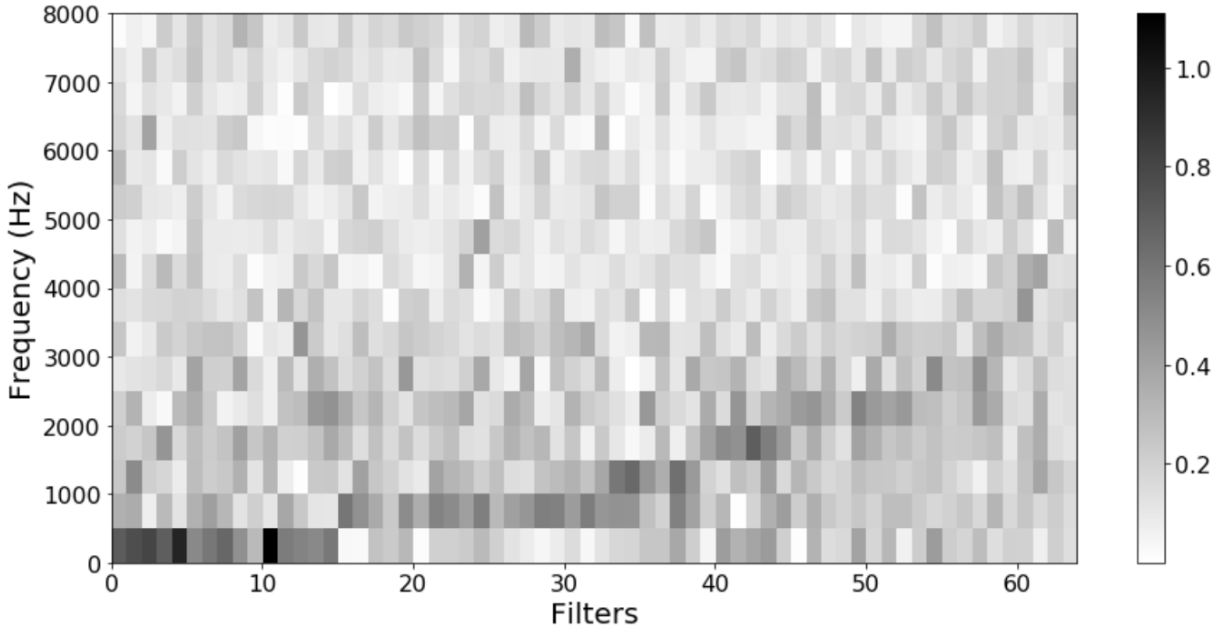


Figure 5.5: We trained a model with a lower number of filters and smaller filters M_{reduced}. To the filters of the first convolutional layer, Fourier transform was applied. Each column corresponds to one filter. We sorted the columns for ascending indices of the maximum value.

5.2.7 Summary

In the previous sections, we described the training of four different approaches. We started with a model (referred to as M_{shift}) trained using a pitch shifted dataset. Next, we presented a model with an increased output vector with a resolution of 10 cents instead of 20 cents. We refer to the trained model as M_{dim720}. Then, the weights of the first convolutional layer were fixed and this layer was excluded from training. For the computation of the weights, we used two different approaches. For M_{freezed}, we used only sine waves and for M_{freezed2}, we used sine waves and cosine waves. Furthermore, we trained a network with a lower number of filters and smaller filters. The trained network is referred to as M_{reduced}.

We performed all trainings using a fixed train/validation/test split of the MDB-stem-synth dataset [8]. So all approaches were tested on the same test set. We evaluated the audio files in the test set with the models and summarized the results of the files using evaluation method Eval1. An overview of the resulting raw pitch accuracies is given in Table 5.10. The best/worst RPA in each row is indicated with a green/red font. The table shows that for a threshold of 10 cents,

	M_{shift}	M_{dim720}	M_{freezed2}	M_{reduced}	M_{tiny}
RPA 10 cents	0.649 ± 0.118	0.778 ± 0.152	0.790 ± 0.120	0.590 ± 0.119	0.837 ± 0.091
RPA 25 cents	0.871 ± 0.105	0.945 ± 0.052	0.935 ± 0.061	0.893 ± 0.086	0.942 ± 0.043
RPA 50 cents	0.927 ± 0.081	0.978 ± 0.023	0.965 ± 0.044	0.958 ± 0.051	0.969 ± 0.027
training	156 min	220 min	162 min	149 min	-

Table 5.10: Summary of raw pitch accuracies and training durations of different approaches.

none of the approaches leads to a higher raw pitch accuracy then the original model of CREPE M_{tiny}. For a threshold of 25 cents, the RPA obtained with M_{dim720} is comparable (slightly higher) to the RPA obtained with M_{tiny}. This model (M_{dim720}) and M_{freezed2} also achieve raw pitch accuracies similar to the raw pitch accuracy of M_{tiny} for a threshold of 50 cents. The model M_{dim720} leads to the highest accuracy compared to the other approaches. However, this model also had a training duration of 220 minutes which is higher than the durations for the remaining models. In order to save training time, we developed the architecture of M_{reduced}. Although all models were trained for 800 epochs, the training for this model had the shortest duration.

So using M_{shift} we were able to extend the range of frequencies that can be estimated. Through the experiments with M_{freezed2}, we received new insights in the role of the first convolutional layer. And by training M_{reduced}, we developed a faster and smaller model for F0-estimation.

Chapter 6

Evaluation on Georgian Recordings

In this chapter, we evaluate CREPE on a more complex and realistic dataset. For this, we use the dataset consisting of Georgian vocal music that was introduced in Section 2.3.3. It consists of five Georgian songs performed by three singers. Here, only the chant Batonebis Nanina is used as running example. For this chant, each singer was recorded with a headset microphone (HDS) and a larynx microphone (LRX). The difference between these microphone types is that the larynx microphone is directly attached to the throat skin of the singer. Thereby, it only records the pitches, but not the words of the singer. The larynx microphone is also less influenced by the other singers. So if the singer has a pause, the voices of the other singers are more audible in the recording of the headset microphone than in the recording of the larynx microphone. Additionally, a recording from the camera microphone and a recording from the room microphone are available. These contain the voices of all singers. Using this example, we will examine the pitch estimations of CREPE in Section 6.1 and the voicing estimations of CREPE in Section 6.2.

6.1 Pitch Estimation

To examine the pitch estimations of CREPE for a more complex example, we first applied CREPE to the larynx and the headset recordings of the singers and compared the results. Then, the estimations of CREPE are compared to the estimations of YIN for the larynx microphones. Finally, we apply CREPE and Melodia to the recording of the room microphone (polyphonic recording).

6.1.1 Evaluation with CREPE

For each singer, CREPE is applied to both recordings (larynx and headset). For this, we use the full model structure M_{full} and a hop size of 10 milliseconds (from audio resampled to 16000 Hz). In Figure 6.1, the results for the first singer are shown. In Figure 6.1a, the pitch trajectories for the larynx microphone and the headset microphone evaluated with CREPE are plotted. For some time ranges, the estimates are similar, and for some time ranges, they differ much more. The differences are computed in cents and a threshold of 25 cents is applied. If both estimates for the larynx recording and the headset recording are almost similar (difference $\leq$ 25 cents), we assume that both predictions are correct. The frames for which the pitch estimates differ by more than 25 cents (estimate assumed to be wrong) are marked in gray in Figure 6.1b. CREPE also computes a confidence value (voicing estimate). A confidence between 0.0 and 0.5 indicates that a frame is predicted to be unvoiced (no melody) and a confidence between 0.5 and 1.0 indicates that a frame is predicted as voiced. The voicing estimate of CREPE for the larynx recording is shown in Figure 6.1c. In the figure, it is visible that frames with a high difference in the estimated pitches for the two microphones highly coincide with frames that are predicted as unvoiced. So for the unvoiced frames (pauses), the pitch estimation is often incorrect. It is also notable that the pitch estimations for the larynx recording vary more than the pitch estimations for the headset microphone. This is probably due to the stronger presence of other voices in the headset recording. So during the pauses, pitches of other singers are predicted.

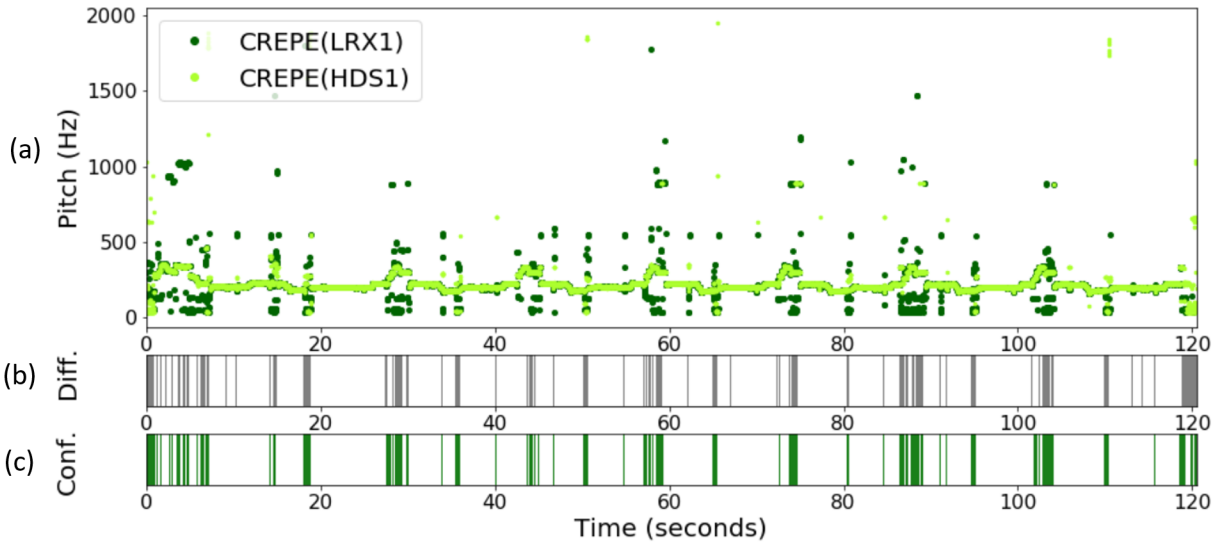


Figure 6.1: Evaluation using CREPE (M_{full}) for the first singer. **(a)** Pitch estimations of CREPE for larynx recording and for headset recording. **(b)** Frames with a pitch difference of more than 25 cents (marked in gray). **(c)** Confidence evaluated with CREPE on larynx recording (frames that are predicted as unvoiced marked in green).

The pitch estimations for the larynx recordings can be considered as more reliable than the estimations for the headset recordings, because less noise from other singers is present. Therefore the estimations from the larynx recordings can be used as ground truth fundamental frequency values. Thereby, the raw pitch accuracy can be computed without other annotations. The raw pitch accuracies for the three singers and different thresholds are shown in Table 6.1. Especially for the first and the second singer, we achieved high accuracies.

threshold	Singer 1	Singer 2	Singer 3
10 cents	0.901	0.900	0.863
25 cents	0.967	0.956	0.932
50 cents	0.978	0.978	0.955

Table 6.1: Raw pitch accuracies for the headset recordings of the three singers evaluated with CREPE (M_{full}, 10 milliseconds hop size for audio resampled to 16000 Hz). As ground truth, the fundamental frequency estimations of CREPE applied to the larynx microphones are used.

6.1.2 Comparison to YIN

In the following, we compare the fundamental frequency estimates of CREPE to the frequencies obtained with YIN [3]. For more details on YIN see Section 3.1.

For the comparison to YIN, the audio files were resampled to 16000 Hz before applying YIN. CREPE also resamples audio files to 16000 Hz, so this ensures that both pitch estimation approaches use exactly the same input and increases the comparability of the results. The hop size for the evaluation with YIN is set to 160 samples and the window length is 1024 samples. These values are equal to the values used for the evaluation with CREPE: The window length of CREPE is fixed to 1024 samples and the hop size is set to 10 milliseconds (corresponds to 160 samples). For YIN, a minimum frequency and a maximum frequency have to be fixed. We use 30 Hz (minimum) and 1760 Hz (maximum). For the comparison of CREPE (M_{full}) and YIN, we only use the recordings from the larynx microphones, because they contain less noise from the other singers. Therefore, pitch estimations on the larynx microphones are more accurate. The pitch estimations from CREPE and YIN for the first singer are shown in Figure 6.2a. The difference between the estimates is computed in cents. Here, we consider frequencies that differ by more than 25 cents as wrong. If the difference is small ($\leq$ 25 cents), we assume that CREPE and YIN both predicted the correct fundamental frequency. Figure 6.2b shows areas with wrong predictions (difference $>$ 25 cents) in gray. A voicing confidence is obtained by applying CREPE to the larynx recording. It is added in Figure 6.2c. For the confidence values, a threshold of 0.5 is used and frames with a confidence below this threshold (predicted as unvoiced) are marked in green. It is notable that frames which are predicted as unvoiced by CREPE are often also frames with a high difference between the frequency estimates. So probably, these are the time ranges where singer 1 is not singing.

In order to further investigate the pitch results, the raw pitch accuracy is computed. As ground truth, we use the fundamental frequencies obtained by applying CREPE to the larynx recordings. For CREPE, we choose the full model M_{full} and a hop size of 10 milliseconds of the audio resampled to 16000 Hz. The evaluated estimations are the fundamental frequency values obtained using YIN with the same settings as described above. We compute the raw pitch accuracy for the larynx recording of each singer. For the results, see Table 6.2. The highest accuracies are achieved for singer 1. However, the raw pitch accuracies for YIN evaluated on the larynx recordings are significantly lower than the raw pitch accuracies for CREPE evaluated on the headset recordings (cf. Table 6.1). This indicates that the estimations of CREPE for the two microphone types are more similar than the estimations of CREPE and YIN on the same microphone type (larynx microphone). One possible explanation for this result is that in the larynx recordings, the voices of the other singers are still slightly audible during pauses of the recorded singer. So it could be the case that CREPE evaluated on a larynx recording predicts the pitches of the other singers during pauses for some time instances.

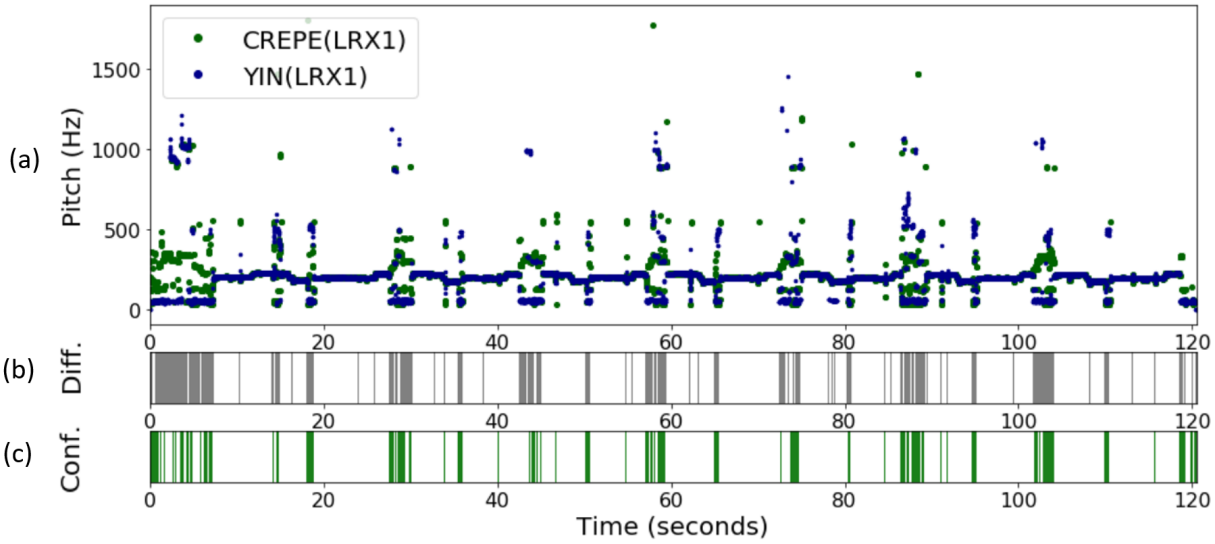


Figure 6.2: Evaluation using CREPE (M_{full}) and YIN for the first singer (larynx recording). **(a)** Pitch estimations for larynx recording with CREPE and YIN. **(b)** Frames with a pitch difference of more than 25 cents (marked in gray). **(c)** Confidence evaluated with CREPE on larynx recording (frames that are predicted as unvoiced marked in green).

6.1.3 Polyphonic Music

CREPE is developed to estimate the fundamental frequencies of monophonic audio recordings. So in contrast to other approaches, CREPE is not trained to recognize the main melody of a recording (predominant melody estimation). Furthermore, CREPE only predicts one frequency value for

threshold	Singer 1	Singer 2	Singer 3
10 cents	0.767	0.655	0.669
25 cents	0.882	0.788	0.803
50 cents	0.908	0.837	0.860

Table 6.2: Raw pitch accuracies for the larynx recordings of the three singers evaluated with YIN. As ground truth, the fundamental frequency estimations of CREPE (M_{full}) applied to the larynx microphones are used.

each time instance and does not have the possibility to investigate multiple simultaneously occurring pitches (multipitch estimation).

However, we apply CREPE to a polyphonic audio recording in order to investigate its results. For the evaluation with CREPE, we use the full model structure M_{full} and a hop size of 10 milliseconds (audios are resampled to 16000 Hz by CREPE). As example, we use the recording from the room microphone for Batonebis Nanina. It contains the voices of all three singers. The fundamental frequency estimates of CREPE are shown in Figure 6.3a. In order to provide some context, we also applied CREPE (same settings as for the room microphone) to the larynx recordings and added the resulting values to the figure. It is notable that the trajectory of the estimations from the room microphone is not very stable. Instead, the estimations jump between the estimations for the other recordings. This can be explained by the architecture of CREPE. The neural network of CREPE returns a 360-dimensional output vector $\hat{y}$. The pitch estimation is derived from this vector by searching for the peak in the vector and considering only entries at a certain neighborhood around the peak. These values serve as weights for cent values. For polyphonic music, the output vectors of CREPE probably contain multiple peaks. Still, only the maximum is considered. So if there are e.g. two peaks, the maximum of both peaks is selected. This is a rather random choice and leads to the jumps in the pitch trajectory.

In Section 3.4, we presented Melodia. Melodia is an approach for pitch estimation developed by Salamon et al. [7]. Since it is an approach for polyphonic music, we use it here for evaluation of the room microphone recording. For the resulting fundamental frequencies see Figure 6.3b. The objective of Melodia is to identify the fundamental frequency values of the main melody. However, the recording of Batonebis Nanina has no clearly determined main melody. Although, the estimations of Melodia are more stable and are less frequently jumping between the estimates for the singers. Also Melodia focuses more on the pitches belonging to singer 3. This is the singer with the highest pitches in general. Melodia only rarely predicts the frequencies belonging to singer 1 (lowest voice).

So in summary it can be said that CREPE predicts frequencies belonging to one of the voices, but the pitch trajectory is jumping frequently. Melodia instead rather focuses on higher voices and produces a more stable pitch trajectory.

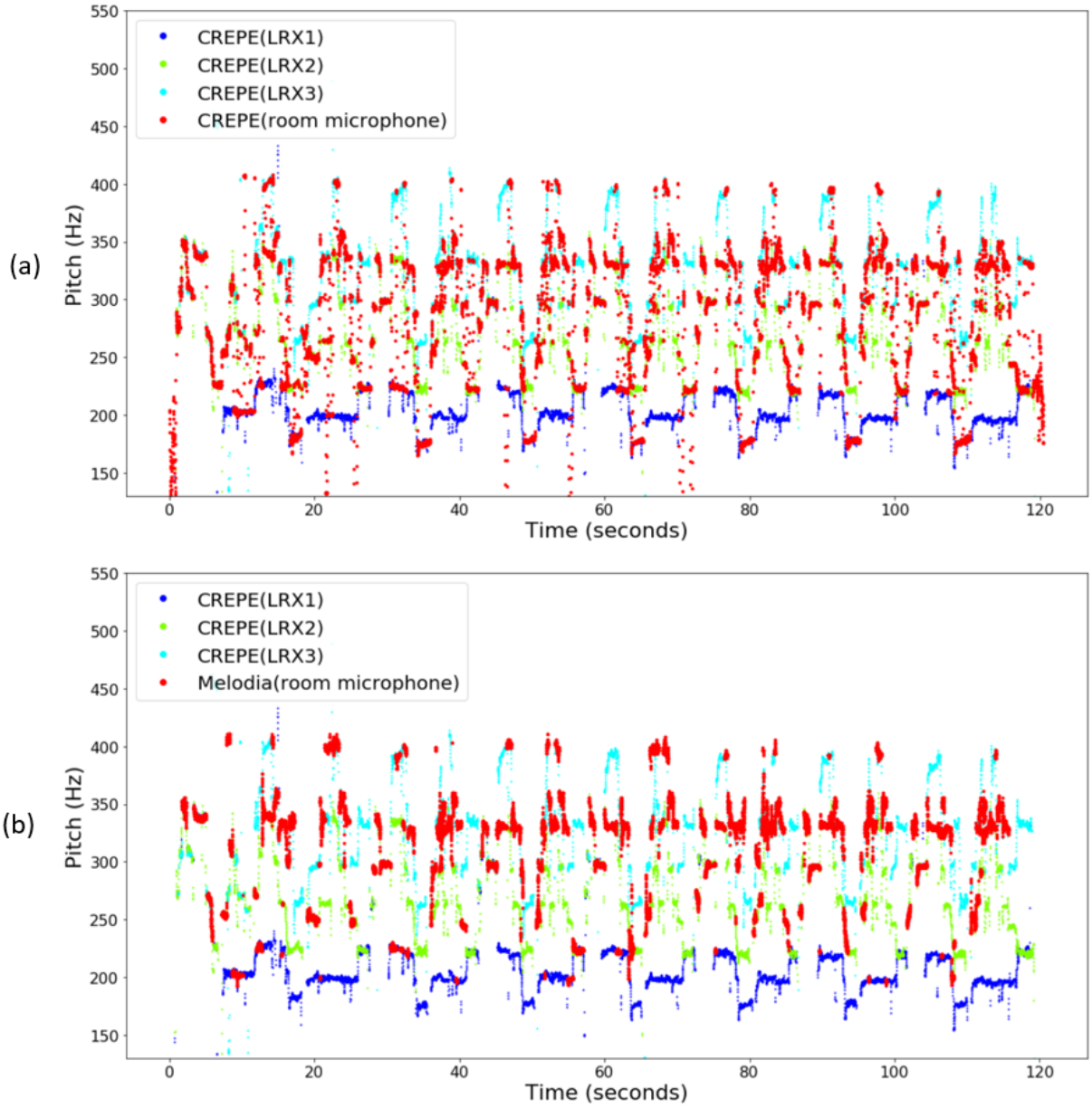


Figure 6.3: Pitch estimation for the larynx recordings of all singers and the recording from the room microphone. **(a)** Estimated with CREPE. **(b)** Estimated with YIN.

6.2 Voicing Estimation

In the previous section, we discussed the estimation of fundamental frequencies for Georgian vocal music. Besides F0-estimation, there is also the task of voicing estimation. The objective of voicing estimation is to predict for a given time instance, whether the corresponding frame is voiced or not. If a frame is unvoiced, this implies that there is no melody present at this time instance.

CREPE outputs a confidence value for each time instance which can be interpreted as voicing estimate. So the voicing function for CREPE maps each time instance n to a confidence value between 0 and 1:

$$V_{\text{Crepe}}(n) \in [0, 1]. \tag{6.1}$$

For CREPE, confidence values above 0.5 indicate that a frame is voiced. So from the voicing function $V_{\text{Crepe}}(n)$, an activity function $A_{\text{Crepe}} : \mathbb{Z} \to \{0, 1\}$ for CREPE can be derived which is 1 if a frame is predicted as voiced and 0 otherwise:

$$A_{\text{Crepe}}(n) = \begin{cases} 1 & \text{if } V_{\text{Crepe}}(n) > 0.5 \\ 0 & \text{else} \end{cases}. \tag{6.2}$$

YIN computes multiple values related to voicing. We use the aperiodic measure which is the ratio of the aperiodic power to the total power. For more details on voicing in YIN, see Section 3.1.2. So a voicing function for YIN can be defined as a function that maps each time instance n to a voicing value:

$$V_{\text{Yin}}(n) \in \mathbb{R}_{\geq 0}. \tag{6.3}$$

In Section 3.1.2, we deduced a threshold of 0.1467 for the aperiodic measure. If the aperiodic measure is above this threshold, this indicates that a high aperiodicity is present and therefore the frame is considered as unvoiced. Otherwise, a frame is considered as voiced. This leads to the following activation function $A_{\text{Yin}} : \mathbb{Z} \to \{0, 1\}$ for YIN:

$$A_{\text{Yin}}(n) = \begin{cases} 1 & \text{if } V_{\text{Yin}}(n) < 0.1467 \\ 0 & \text{else} \end{cases}. \tag{6.4}$$

As example, we examined the voicing functions and the activation functions of CREPE and YIN on the recording of Batonebis Nanina (singer 1). For CREPE, we use the full model structure M_{full} and a hop size of 10 milliseconds. For YIN, we first resampled the audio files to 16000 Hz, in order to use the same input as for CREPE. YIN was then evaluated with a hop size of 160 samples and a window length of 1024 samples (same as CREPE). The minimum frequency was set to 30 Hz and the maximum frequency was set to 1760 Hz. See Figure 6.4 for the results. For the plot, the result of the voicing function of YIN is multiplied by (-1). The reason is that YIN has low voicing values at frames that are estimated as unvoiced whereas CREPE has high voicing values at frames that are estimated as unvoiced. In order to receive a standardized plot, the multiplication by (-1) is necessary. The plot shows that the voicing values are higher for the headset recordings (Figure 6.4a) than for the larynx recordings (Figure 6.4b). This can be explained by the pauses in the larynx recordings. The activity function of CREPE and YIN for

the larynx recordings is shown in Figure 6.4c. Here, we also added the union and the intersection of the activity functions:

$$(A_{\text{Crepe}} \cup A_{\text{Yin}})(n) = \begin{cases} 1 & \text{if } A_{\text{Crepe}}(n) == 1 \text{ or } A_{\text{Yin}}(n) == 1 \\ 0 & \text{else} \end{cases} \tag{6.5}$$

and

$$(A_{\text{Crepe}} \cap A_{\text{Yin}})(n) = \begin{cases} 1 & \text{if } A_{\text{Crepe}}(n) == 1 \text{ and } A_{\text{Yin}}(n) == 1 \\ 0 & \text{else} \end{cases}. \tag{6.6}$$

Through listening to the recording, especially the intersection of the activity functions of CREPE and YIN $A_{\text{Crepe}} \cap A_{\text{Yin}}$ was found to be a good estimate for voicing.

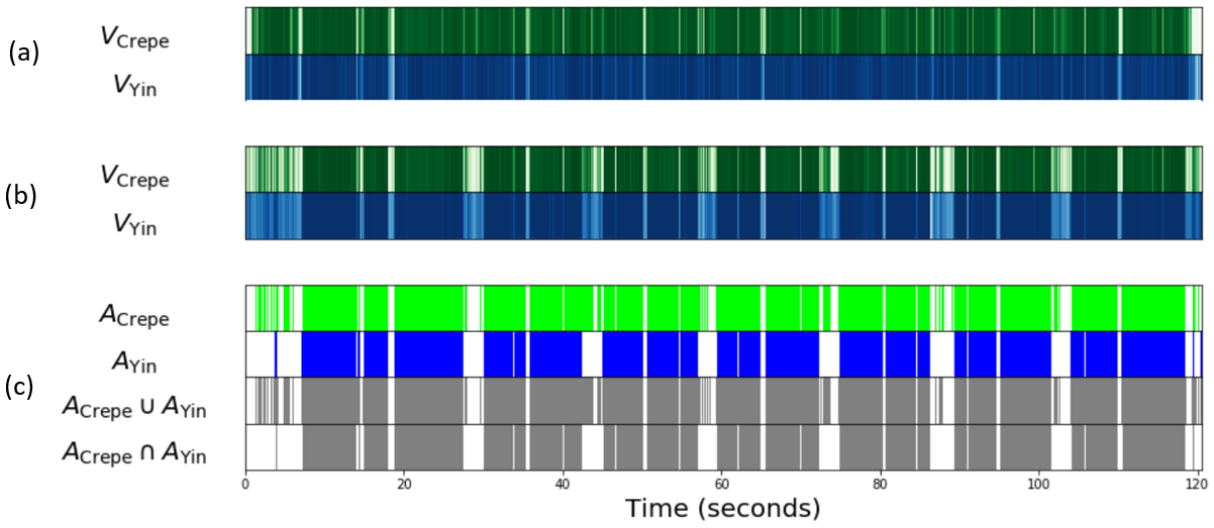


Figure 6.4: Voicing of CREPE and YIN applied to Batonebis Nanina. **(a)** Voicing estimates from CREPE and YIN for the headset recording of singer 1. **(b)** Voicing estimates from CREPE and YIN for the larynx recording of singer 1. **(c)** Activities for the larynx recording of singer 1.

Using the intersection of the activity functions of CREPE and YIN, we compare the voicing activities of the three singers. So CREPE and YIN are applied to the larynx recordings of the singers. For CREPE and YIN, we use the same settings as described above. The resulting activities are shown in Figure 6.5. When listening to the recording, it can be recognized that there are certain seconds where only one singer is singing. By listening, these solo parts were found to occur at seconds 0, 28, 43, 57, 71, 87 and 101. These time instances are marked in orange in the plot. The voicing activity in the figure also shows that there is only the second person singing.

So a very accurate voicing estimation can be obtained by using the voicing estimation of CREPE and YIN and combine those using intersection.

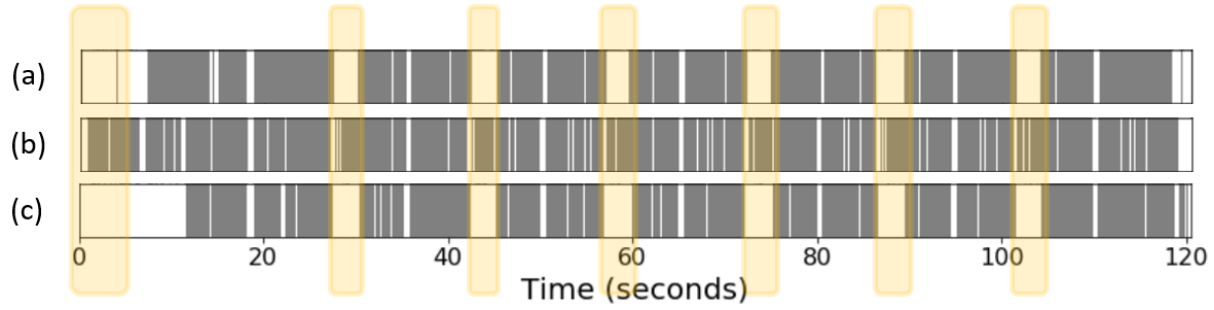


Figure 6.5: Voicing activation using the intersection of the activities of CREPE and YIN. Parts where only singer 2 is singing are marked in orange. **(a)** $A_{\text{Crepe}} \cap A_{\text{Yin}}$ for larynx recording of singer 1. **(b)** $A_{\text{Crepe}} \cap A_{\text{Yin}}$ for larynx recording of singer 2. **(c)** $A_{\text{Crepe}} \cap A_{\text{Yin}}$ for larynx recording of singer 3.

Chapter 7

Conclusions

In this thesis, we analyzed an approach for fundamental frequency estimation based on a deep neural network and developed further approaches to improve the network performance and the training process.

We started with an evaluation of model-based approaches for fundamental frequency estimation in Chapter 3. For the evaluation, we used YIN [3], pYIN [4], SWIPE′ [5, 6] and Melodia [7]. These approaches for F0-estimation were evaluated on five examples consisting of three artificial examples and two examples from the MDB-stem-synth dataset [8]. With YIN, we obtained good frequency estimations, although the frequency range has to be restricted. The second approach, pYIN resulted in inaccurate estimations for one of the examples of the MDB-stem-synth, possibly due to the instrument (viola). Using SWIPE′, we obtained estimations with a high similarity to the annotated frequencies. Also Melodia was found to produce accurate estimations, though there are some errors due to the assignment of frequencies to bins with a fixed frequency range.

Next, in Chapter 4, we presented CREPE, a convolutional neural network developed by Kim et al. [1]. This is an approach for fundamental frequency estimation based on a deep convolutional neural network. We evaluated this approach on some audio examples and obtained accurate frequency estimations. However, the frequency range of CREPE is limited, so only frequencies in this range can be estimated correctly. For higher frequencies, CREPE predicts the subharmonic instead according to our experiments.

Additionally, we developed further approaches for modifications of CREPE in Chapter 4. Since only a limited dataset was available for training, we used data augmentation. For this, pitch shifting was applied to the dataset. Another approach was to increase the dimension of the output layer of the network. This leads to a higher frequency resolution of the output vector. Furthermore, we evaluated the weights of the first convolutional layer of the network. The Fourier transform of the filters was found to form a curve similar to an exponential curve when sorted

for the maximum index. We used this result to precompute weights for the filters in order to exclude the first layer from the training process. Finally, we developed a new network with a reduced number of filters and smaller filters.

We then evaluated these modifications in Chapter 5. For this purpose we retrained the network including the new approaches. Especially data augmentation (pitch shifting) was found to be a useful technique. A trained network only learns to predict frequencies correctly in ranges for which it was trained. So using pitch shifting, we were able to increase this frequency range.

Finally, we applied CREPE to a more complex dataset consisting of traditional Georgian vocal music in Chapter 6. For one recording, we compared the F0-estimations obtained from different microphones and compared the estimations of CREPE and YIN. We found out that the difference of the estimations of CREPE and YIN is a good indication for unvoiced segments. If there is no voice present, the difference between the estimations is higher. Furthermore, we applied CREPE to a polyphonic recording (three singers). The F0-estimations of CREPE are equal to the frequency of one of the singers at most time instances. This can be explained by the architecture of CREPE. Additionally, we examined the voicing estimation of CREPE and YIN on different microphones. The result is that the best binary voicing detection is obtained using the intersection of the estimations of CREPE and YIN applied to the recording of a larynx microphone. Using this voicing estimation, we were also able to detect segments where only one singer is active in a recording.

Future research concerning the training process of CREPE could evaluate the influence of more training epochs on the results. The approach using data augmentation could also be extended by adding reverberation and a different pitch shifting method. Furthermore, CREPE could be retrained for Georgian Vocal Music [12, 13] in order to improve the F0-estimations on this dataset.

Also concerning the architecture of CREPE, further research is possible. This could address the examination of the parameters of CREPE. This includes the choice of the sampling rate for the audio input for CREPE, the window size for the evaluation and the formula for the computation of an output in Hertz (size of neighborhood around peak). Also the network structure could be further modified. The derivation of the confidence determined by CREPE uses a threshold that is not theoretically justified. Instead, it could be trained along with the F0-estimation. Theoretically, also multipitch estimation or predominant melody estimation could be possible with CREPE. This would require a modified interpretation of the output vector and further postprocessing steps.

Bibliography

[1] Jong Wook Kim, Justin Salamon, Peter Li, and Juan Pablo Bello. CREPE: A convolutional representation for pitch estimation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 161–165, 2018.

[2] Brian C. J. Moore. *An introduction to the psychology of hearing, 5th ed.* Academic Press, 2003.

[3] Alain de Cheveigné and Hideki Kawahara. YIN, a fundamental frequency estimator for speech and music. *Journal of the Acoustical Society of America (JASA)*, 111(4):1917–1930, 2002.

[4] Matthias Mauch and Simon Dixon. pYIN: A fundamental frequency estimator using probabilistic threshold distributions. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 659–663, Florence, Italy, 2014.

[5] Arturo Camacho and John G. Harris. A sawtooth waveform inspired pitch estimator for speech and music. *The Journal of the Acoustical Society of America*, 124(3):1638–1652, 2008.

[6] Arturoa Camacho. *Swipe: A Sawtooth Waveform Inspired Pitch Estimator for Speech and Music. 2007.* PhD thesis, University of Florida.

[7] 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.

[8] Justin Salamon, Rachel M Bittner, Jordi Bonada, Juan J Bosch, Emilia Gómez, and Juan Pablo Bello. An analysis/synthesis framework for automatic f0 annotation of multitrack datasets. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, 2017.

[9] Meinard Müller. *Fundamentals of Music Processing.* Springer Verlag, 2015.

[10] C-major scale piano. `https://upload.wikimedia.org/wikipedia/commons/7/72/C_major.ogg`. Accessed: 2019-10-22.

[11] Justin Salamon, Emilia Gómez, Daniel P. W. Ellis, and Gaël Richard. Melody extraction from polyphonic music signals: Approaches, applications, and challenges. *IEEE Signal Processing Magazine*, 31(2):118–134, 2014.

[12] Frank Scherbaum, Nana Mzhavanadze, and Elguja Dadunashvili. A web-based, long-term archive of audio, video, and larynx-microphone field recordings of traditional Georgian singing, praying and lamenting with special emphasis on Svaneti. *International Symposium on Traditional Polyphony*, 2018.

[13] Frank Scherbaum, Sebastian Rosenzweig, Meinard Müller, Daniel Vollmer, and Nana Mzhavanadze. Throat microphones for vocal music analysis. In *Demos and Late Breaking News of the International Society for Music Information Retrieval Conference (ISMIR)*, Paris, France, 2018.

[14] Rachel M. Bittner, Justin Salamon, Mike Tierney, Matthias Mauch, Chris Cannam, and Juan Pablo Bello. MedleyDB: A multitrack dataset for annotation-intensive MIR research. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 155–160, Taipei, Taiwan, 2014.

[15] Emilia Gómez and Jordi Bonada. Towards computer-assisted flamenco transcription: An experimental comparison of automatic transcription algorithms as applied to a cappella singing. *Computer Music Journal*, 37(2):73–90, 2013.

[16] Jordi Bonada. Wide-band harmonic sinusoidal modeling. In *11th International Conference on Digital Audio Effects DAFx*, volume 8, pages 265–272, Espoo, Finland, 2008.

[17] Synth datasets. `http://synthdatasets.weebly.com/`. Accessed: 2019-10-22.

[18] Musik aktiv lernen mit music delta. `https://www.lerneo.de/lernwelten/music-delta/`. Accessed: 2019-10-22.

[19] Medleydb annotations. `https://github.com/marl/medleydb/tree/master/medleydb/data/Metadata`. Accessed: 2019-10-22.

[20] Georgia population worldometers. `https://www.worldometers.info/world-population/georgia-population/`. Accessed: 2019-10-22.

[21] Meinard Müller, Sebastian Rosenzweig, Jonathan Driedger, and Frank Scherbaum. Interactive fundamental frequency estimation with applications to ethnomusicological research. In *Proceedings of the AES International Conference on Semantic Audio*, pages 186–193, Erlangen, Germany, 2017.

[22] Graham E. Poliner, Daniel P.W. Ellis, Andreas F. Ehmann, Emilia Gómez, Sebastian Streich, and Beesuan Ong. Melody transcription from music audio: Approaches and evaluation. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(4):1247–1256, 2007.

[23] J Stephen Downie, Kris West, Andreas Ehmann, and Emmanuel Vincent. The 2005 music information retrieval evaluation exchange (mirex 2005): Preliminary overview. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, pages 320–323, London, U.K., 2005.

[24] Swipe' implementation. `https://github.com/kylebgorman/swipe`. Accessed: 2019-11-15.

[25] Crepe pitch tracker. `https://github.com/marl/crepe`. Accessed: 2019-11-15.

[26] Chao-Ling Hsu and Jyh-Shing Roger Jang. On the improvement of singing voice separation for monaural recordings using the MIR-1K dataset. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(2):310–319, February 2010.

[27] Zhiyao Duan, Bryan Pardo, and Changshui Zhang. Multiple fundamental frequency estimation by modeling spectral peaks and non-peak regions. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(8):2121–2133, 2010.

[28] Masataka Goto, Hiroki Hashiguchi, Takuichi Nishimura, and Ryuichi Oka. RWC music database: Popular, classical and jazz music databases. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, volume 2, pages 287–288, 2002.

[29] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Mohammad Norouzi, Douglas Eck, and Karen Simonyan. Neural audio synthesis of musical notes with wavenet autoencoders. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 1068–1077. JMLR.org, 2017.

[30] Resampy documentation. `https://resampy.readthedocs.io`. Accessed: 2019-10-22.

[31] 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, 2015.

[32] 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.

[33] Keras: The python deep learning library. `https://keras.io/`. Accessed: 2019-10-22.

[34] 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.

[35] Librosa. `https://librosa.github.io/librosa/`. Accessed: 2019-10-22.

[36] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, volume 8, 2015.

[37] Brian McFee, Matt McVicar, Oriol Nieto, Stefan Balke, Carl Thome, Dawen iang, Eric Battenberg, Josh Moore, Rachel Bittner, Ryuichi Yamamoto, Dan Ellis, Fabian-Robert Stöter, Douglas Repetto, Simon Waloschek, CJ Carr, Seth Kranzler, Keunwoo Choi, Petr Viktorin, Joao Felipe Santos, Adrian Holovaty, Waldir Pimenta, and Hojin Lee. librosa 0.5.0, February 2017.

[38] 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.