DEEP LEARNING FOR AUDIO EFFECTS MODELING

MARCO A. MARTÍNEZ RAMÍREZ

DEEP LEARNING FOR AUDIO EFFECTS MODELING

MARCO A. MARTÍNEZ RAMÍREZ

Submitted in partial fulfilment of the requirements of the
University of London Degree of Doctor of Philosophy
School of Electronic Engineering and Computer Science
Queen Mary University of London

SUPERVISORS:
Prof Joshua D. Reiss
Dr Emmanouil Benetos

November 2020

The intelligence is in the sound.
—-Verfaille, Zölzer, and Arfib (2006)

ABSTRACT

Audio effects modeling is the process of emulating an audio effect unit and seeks to recreate the sound, behaviour and main perceptual features of an analog reference device. Audio effect units are analog or digital signal processing systems that transform certain characteristics of the sound source. These transformations can be linear or nonlinear, time-invariant or time-varying and with short-term and long-term memory. Most typical audio effect transformations are based on dynamics, such as compression; tone such as distortion; frequency such as equalization; and time such as artificial reverberation or modulation based audio effects.

The digital simulation of these audio processors is normally done by designing mathematical models of these systems. This is often difficult because it seeks to accurately model all components within the effect unit, which usually contains mechanical elements together with nonlinear and time-varying analog electronics. Most existing methods for audio effects modeling are either simplified or optimized to a very specific circuit or type of audio effect and cannot be efficiently translated to other types of audio effects.

This thesis aims to explore deep learning architectures for music signal processing in the context of audio effects modeling. We investigate deep neural networks as black-box modeling strategies to solve this task, i.e. by using only input-output measurements. We propose different DSP-informed deep learning models to emulate each type of audio effect transformations.

Through objective perceptual-based metrics and subjective listening tests we explore the performance of these models when modeling various analog audio effects. Also, we analyze how the given tasks are accomplished and what the models are actually learning. We show virtual analog models of nonlinear effects, such as a tube preamplifier; nonlinear effects with memory, such as a transistor-based limiter; and electromechanical nonlinear time-varying effects, such as a Leslie speaker cabinet and plate and spring reverberators.

We report that the proposed deep learning architectures represent an improvement of the state-of-the-art in black-box modeling of audio effects and the respective directions of future work are given.

ACKNOWLEDGEMENTS

This thesis represents the fruit of a passion for music and engineering and, more specifically, the result of a drive towards *sound* and its timeless characteristic of immediacy. Throughout my life, this feeling always aroused a certain curiosity in me, which led me to aspire to contribute to the technologies that allow us to create, transform and deliver sounds.

Therefore, the completion of this document can be seen as an important milestone towards the fulfilment of this journey, where each moment has been of equal importance; from the day I thought I had hearing damage after listening to distortion samples all day; to the day that something finally coherent came from the output of a trained model. However, this thesis would not have been possible without the support of many people.

First of all, I am very grateful to my supervisor, Joshua Reiss, who with great enthusiasm and a deep knowledge of audio engineering introduced me to the field of research and always supported me in any path I chose. To my second supervisor, Emmanouil Benetos, for guiding me through the difficult world of music computing and machine learning, always pushing for excellence.

Also, I would like to thank Bob Sturm and Sebastian Ewert for such a demanding stage-1 evaluation, which helped me put this research into perspective and consequently course-correct. To my MSc and BSc supervisors; Patrick Gaydecki and Alfredo Restrepo Palacios, whose vast knowledge of digital signal processing pointed the way forward. To Adam Keen, whose teachings and friendship have been truly life-changing.

All these years would not have been the same without all the brilliant people at C4DM. Special gratitude to; Giulio Moro, for always being very much himself and for always knowing and caring about what is important; Delia Fano Yela, for her contagious joy and editor skills; William Wilkinson, for being the best and always willing to explain stationary covariance functions; Changhong Wang, for her inspiring dedication which is what unity is all about. Hazar and Lucia for the 108.

Lida, for all the love and moments filled with her kindness that made the end of the PhD not so difficult. Juliana, for all these years of invaluable friendship and many more to come.

Lastly, and most importantly, huge thanks to my family. *Padre y Madre*, for always being the best example to follow. I owe all my merits to both of you. Gustavo, Lilo, Sophia and Salomón, I love you all.

All in all, a very interesting experience !

This doctorate was funded by COLFUTURO. The Titan Xp GPU used for this research was donated by the NVIDIA Corporation.

DECLARATION

I, Marco A. Martínez Ramírez, confirm that the research included within this thesis is my own work or that where it has been carried out in collaboration with, or supported by others, that this is duly acknowledged below and my contribution indicated. Previously published material is also acknowledged below.

I attest that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge break any UK law, infringe any third party's copyright or other Intellectual Property Right, or contain any confidential material.

I accept that the College has the right to use plagiarism detection software to check the electronic version of the thesis.

I confirm that this thesis has not been previously submitted for the award of a degree by this or any other university.

The copyright of this thesis rests with the author and no quotation from it or information derived from it may be published without the prior written consent of the author.

Date: November 30, 2020

[illegible]

Marco A. Martínez Ramírez

CONTENTS

ACRONYMS

AI	Artificial Intelligence
BBD	Bucket Brigade Delay
Bi-LSTM	Bidirectional Long Short-Term Memory
CNN	Convolutional Neural Network
CAFx	Convolutional audio effects modeling network
CEQ	Convolutional EQ modeling network
Conv1D	One-dimensional Convolutional layer
Conv1D-Local	Locally Connected One-dimensional Convolutional layer
CRAFx	Convolutional Recurrent audio effects modeling network
CSAFx	Convolutional recurrent Sparse filtering audio effects modeling network
CWAFx	Convolutional and WaveNet audio effects modeling network
CPU	Central Processing Unit
dBFS	Decibels Relative to Full Scale
DCT	Discrete Cosine Transform
deConv1D	One-dimensional Deconvolution layer
Dense-Local	Locally Connected Dense layer
DNN	Deep Neural Network
DSP	Digital Signal Processing
EQ	Equalization
ERB	Equivalent Rectangular Bandwidth
FIR	Finite Impulse Response
FC	Fully Connected

FFT	Fast Fourier Transform
Fx	Effects
GPU	Graphics Processing Unit
IIR	Infinite Impulse Response
JFET	Junction Field Effect Transistor
KL	Kullback–Leibler divergence
LC	Locally Connected
LTI	Linear Time Invariant
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MFCC	Mel-Frequency Cepstral Coefficients
MSE	Mean Squared Error
OTA	Operational Transconductance Amplifier
ReLU	Rectifier Linear Unit
RNN	Recurrent Neural Network
SAAF	Smooth Adaptive Activation Function
SE	Squeeze-and-Excitation
SFIR	Sparse FIR
SGD	Stochastic Gradient Descent
STFT	Short-Time Fourier Transform
VST	Virtual Studio Technology
WaveNet	Feedforward Wavenet audio effects modeling network
WDF	Wave Digital Filter

1 INTRODUCTION

Audio effects are widely used in various media such as music, live performances, television, films or video games. In the context of music production, audio effects are mainly used for aesthetic reasons and are usually applied to manipulate the dynamics, spatialisation, timbre or pitch of vocal or instrument recordings. This manipulation is achieved through effect units, or audio processors, that can be linear or nonlinear, time-invariant or time-varying and with short-term or long-term memory.

Most of these effects can be implemented directly in the digital domain through the use of digital filters and delay lines. Nevertheless, modeling specific effect units or analog circuits and their salient perceptual qualities has been heavily researched and remains an active field. This is because their analog circuitry, often together with mechanical elements, yields a nonlinear and time-varying system which is difficult to fully emulate digitally.

Methods for modeling audio effects mainly involve circuit modeling and optimization for specific analog components such as vacuum-tubes, operational amplifiers or transistors. Such audio processors are not easily modeled, requiring complex, customized digital signal processing (DSP) algorithms. This often requires models that are too specific for a certain circuit or making certain assumptions when modeling specific nonlinearities or components. Therefore such models are not easily transferable to different effects units since expert knowledge of the type of circuit being modeled is always required. Also, musicians tend to prefer analog counterparts because their digital implementations may lack the broad behaviour of the analog reference devices.

In recent years, deep neural networks (DNN) for music have experienced a significant growth. Most music applications are in the fields of music information retrieval, music recommendation, and music generation. End-to-end deep learning architectures, where raw audio is both the input and the output of the system, follow black-box modeling approaches where an entire problem can be taken as a single indivisible task which must be learned from input to output. Thus, the desired output is obtained by learning and processing directly the incoming raw audio, which reduces the amount of required prior knowledge and minimizes the engineering effort.

Prior to this PhD project, deep learning architectures using this principle, i.e. processing directly raw audio, had not been explored for audio processing tasks such as audio effects modeling, although they have been investigated for various music processing tasks (see Section 3.8).

Nevertheless, DNNs for audio effects modeling have recently become an emerging field and have been investigated as end-to-end methods or as parameter estimators of audio processors (see Section 3.9). Most of the end-to-end research has focused on modeling nonlinear audio processors with short-term memory, such as distortion effects. Moreover, the methods based on parameter estimation are based on fixed audio processing architectures. As a result, generalization among different types of audio effect units is usually difficult. This lack of generalization is accentuated when we take into account the broad characteristics of the different types of audio effects, some of which are based on highly complex nonlinear and time-varying systems whose modeling methods remain an active field.

In this thesis, *we aim to investigate a general-purpose deep learning architecture for audio processing in the context of audio effects modeling*. Thus, our motivation is to demonstrate the feasibility of DNNs as audio processing blocks for generic black-box modeling of all types of audio effects. In this way, given an arbitrary audio processor, we research whether a neural network learns and applies the intrinsic characteristics of this transformation.

In this respect, *we explore whether a deep neural network is capable of recreating the sound, behaviour and main perceptual features of various types of audio effects.* Based on the modeling capabilities of DNNs together with domain knowledge from digital audio effects, we propose different deep learning architectures and investigate if these models can process and output audio that matches the sonic and perceptual qualities of a reference audio effect. Throughout this thesis, we measure the performance of the models via objective perceptual-based metrics and subjective listening tests.

1.1 OUTLINE

Chapters 2 and 3 present the relevant literature related to digital audio effects, audio effects modeling and deep learning. In Chapter 4, as a proof-of-concept, we implement a novel deep learning architecture to model linear effects such as equalization (EQ). Based on end-to-end convolutional neural networks (CNN), we introduce a general-purpose architecture for equalization matching. Thus, by using an end-to-end learning approach, the model approximates the equalization target as a content-based transformation without directly finding the transfer function. We show the model performing matched equalization for *shelving*, *peaking*, *lowpass* and *highpass* digital equalizers.

In Chapter 5 we build on the previous model in order to emulate much more complex transformations such as nonlinearities. Also, we analyse a model solely based on temporal dilated convolutions, thus we investigate end-to-end DNNs for modeling linear and nonlinear audio effects with short-term memory. We explore nonlinear emulation as a content-based transformation without explicitly obtaining the solution of the nonlinear system. We show the model performing nonlinear modeling for *distortion*, *overdrive*, *amplifier emulation* and *combinations of linear and nonlinear* digital audio effects.

Since the previous architectures do not generalize to transformations with long temporal dependencies such as modulation based audio effects, in Chapter 6 we explore how a DNN can learn the long-term memory which characterizes these effect units as well as the possibilities to match nonlinearities within the audio effects. We build on the preceding models and we explore whether a latent-space based on recurrent neural networks (RNN) or temporal dilated convolutions is able to learn time-varying transformations. We show the model matching modulation based digital audio effects such as *chorus*, *flanger*, *phaser*, *tremolo*, *vibrato*, *auto-wah*, *ring modulator* and *Leslie speaker*. Furthermore, we extend the applications of the model by including nonlinear time-invariant audio effects with long temporal dependencies such as *compressor* and *multiband compressor*.

The previous chapters have focused on modeling several linear and nonlinear time-varying and time-invariant digital audio effects. Thus, in Chapter 7, through objective perceptual-based metrics and subjective listening tests we explore the performance of each of the architectures from Chapters 5-6 when modeling various analog audio effects. We show virtual analog models of nonlinear effects, such as a *vaccum-tube preamplifier*; nonlinear effects with long-term memory, such as a *transistor-based limiter*; and nonlinear time-varying effects, such as the rotating *horn* and rotating *woofer* of a *Leslie speaker cabinet*.

Drawing further connections between DNNs and signal-processing systems in Chapter 8, we propose a DSP-informed deep learning architecture for the modeling of artificial reverberators. We show the model matching *plate* and *spring* reverberators and we explore the capabilities of DNNs to learn such highly nonlinear electromechanical responses. In order to measure the performance of the model, we conduct a perceptual evaluation experiment and we also analyze how the given task is accomplished and what the model is actually learning.

1.2 CONTRIBUTIONS

The principal contributions of this thesis are:

- **Chapter 4**: a novel deep learning architecture to perform matched equalization: *Convolutional EQ modeling network (CEQ)*. To the best of our knowledge, this work represents the first DNN for end-to-end black-box modeling of linear audio effects.
- **Chapter 5**: a novel deep learning architecture which represents the state-of-the-art for black-box modeling of nonlinear and linear audio effects. Thus, in this chapter we introduce the *Convolutional Audio Effects modeling network (CAFx)* and we also review the *Feedforward WaveNet Audio Effects modeling network (WaveNet)*, which is based on active research (Rethage et al., 2018).
- **Chapter 6**: two general-purpose deep learning architectures to model audio effects with long-term memory: the *Convolutional Recurrent Audio Effects modeling network (CRAFx)* and the *Convolutional and WaveNet Audio Effects modeling network (CWAFx)*. These architectures build on Chapter 5 and are based on RNNs and temporal dilated convolutions respectively. In addition, they represent state-of-the-art DNN architectures to model nonlinear time-varying audio effects, and in general, to learn long temporal dependencies for end-to-end audio processing tasks.
- **Chapter 7**: through objective perceptual-based metrics and subjective listening tests we perform a systematic comparison between the previous architectures. We report that models containing recurrent layers are able to learn long temporal dependencies and therefore outperform other architectures.
- **Chapter 8**: a novel signal processing-informed DNN to model reverberation: the *Convolutional Recurrent and Sparse Filtering Audio Effects modeling network (CSAFx)*. This network uses domain specific insights, such as sparse FIR filters, within a deep learning framework. Thus introducing a more explainable network which also outperforms the previous RNN-based model. The proposed architecture represents the state-of-the-art of deep learning for black-box modeling of artificial reverberators.

1.3 PUBLICATIONS

The main contributions of this thesis have been peer-reviewed and published. Some ideas and figures have appeared previously in the following publications, which are referred to in the text by their Roman numerals. The author of this thesis contributed the majority of work towards all the publications.

I **"End-to-end equalization with convolutional neural networks."** Martínez Ramírez, M.A.; Reiss, J.D. In Proceedings of the 21st International Conference on Digital Audio Effects (DAFx-18), Aveiro, Portugal, 4–8 September 2018. http://dafx2018.web.ua.pt/papers/DAFx2018_paper_27.pdf

Publication **I** forms the basis of Chapter 4. The chapter contains a more detailed derivation of *CEQ*.

II **"Modeling nonlinear audio effects with end-to-end deep neural networks."** Martínez Ramírez, M.A.; Reiss, J.D. In Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Brighton, UK, 12–17 May 2019. https://ieeexplore.ieee.org/document/8683529

Publication **II** forms the basis of Chapter 5. The chapter contains a more detailed derivation of *CAFx*.

III **"A general-purpose deep learning approach to model time-varying audio effects."** Martínez Ramírez, M.A.; Benetos, E.; Reiss, J.D. In Proceedings of the 22nd International Conference on Digital Audio Effects (DAFx-19), Birmingham, UK, 2–6 September 2019. http://dafx2019.bcu.ac.uk/papers/DAFx2019_paper_12.pdf

Publication **III** forms the basis of Chapter 6, which contains a more detailed derivation of *CRAFx* and also introduces *CWAFx* from Publication **IV**.

IV **"Deep learning for black-box modeling of audio effects."** Martínez Ramírez, M.A.; Benetos, E.; Reiss, J.D. *Applied Sciences* 10, no. 2, p.638, January 2020. https://www.mdpi.com/2076-3417/10/2/638

Publication **IV** forms the basis of Chapter 7.

V **"Modeling plate and spring reverberation using a DSP-informed deep neural network."** Martínez Ramírez, M.A.; Benetos, E.; Reiss, J.D. In Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Barcelona, Spain, 4–8 May 2020. https://ieeexplore.ieee.org/document/9053093

Publication **V** forms the basis of Chapter 8, which contains a more detailed derivation of *CSAFx*.

2 AUDIO EFFECTS

In this chapter and in the next chapter, we present an overview of the necessary background that is required to motivate a deep learning approach to the modeling of audio effects. We introduce the signal processing characteristics of the different types of audio effects followed by the different modeling methods and their technical challenges. Furthermore, Appendix A presents the relevant audio representations used throughout this thesis.

2.1 EQUALIZATION - EQ

EQ is an audio effect widely used in the production and consumption of music (Välimäki and Reiss, 2016). It consists of the modification of frequency content through positive or negative gains which change the harmonic and timbral characteristics of the audio. This is performed for different purposes, such as a corrective/technical filter to reduce masking or leakage within a mixing task, to modify the frequency response of a speaker system, or as an artistic or creative tool when recording a specific audio source.

An equalizer is normally implemented via a filter bank whose coefficients are obtained from the designed cut-off frequency f_c and quality factor Q. In general, EQ is performed through a gain G at a given f_c and Q, and it can be applied in the time-domain and frequency-domain (Verfaille et al., 2006). The filter bank consists of Finite Impulse Response (FIR) or Infinite Impulse Response (IIR) filters, whose main difference corresponds to the absence or presence of feedback loops (Gaydecki, 2004). The non-recursive FIR discrete difference equation is

$$y(n) = \sum_{k=0}^{M-1} a_k \cdot x(n-k), \tag{2.1}$$

where $y(n)$ is the output signal, $x(n)$ is the input signal and a_k correspond to the M filter coefficients. This non-recursive system is characterized by being a weighted sum of delayed inputs, thus a general FIR filter consists of a feed-forward system with $M-1$ delay lines. The recursive IIR discrete difference equation is

$$y(n) = \sum_{k=0}^{M-1} a_k \cdot x(n-k) - \sum_{k=1}^{N} b_k \cdot y(n-k). \tag{2.2}$$

The IIR systems are characterized by a weighted sum of delayed inputs and outputs. This feedback system consists of an input delay line of $M-1$ elements and an output delay line of N elements. The recursive coefficients are b_k.

The frequency-domain behaviour of digital filters is commonly described with the Z-transform of the input signal $X(z)$ and output signal $Y(z)$, i.e. the transfer function $H(z)$ (Gaydecki, 2004). The main characteristic of the Z-Transform is the time-shifting property, which means that a multiplication by z^{-n} delays a signal sample by n intervals. The transfer function of the IIR filter from Eq. (2.2) is

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M-1} a_k \cdot z^{-k}}{1 + \sum_{k=1}^{N} b_k \cdot z^{-k}}. \tag{2.3}$$

The various types of filters can be classified into the following classes (Zölzer, 2011).

- **Lowpass**: attenuates frequencies above f_c and Q determines a boost or resonance of frequencies around f_c.
- **Highpass**: attenuates frequencies below f_c and Q determines a boost or resonance of frequencies around f_c.
- **Bandpass**: attenuates frequencies above and below a frequency band with bandwidth f_b and center frequency f_c. Q is the relative bandwidth f_c/f_b. A bandreject filter attenuates frequencies within the frequency band.
- **Shelving**: a low-frequency shelving filter (*lowshelf*) boosts or cuts frequencies below f_c while preserving frequencies above f_c. A high-frequency shelving filter (*highshelf*) boosts or cuts frequencies above f_c while preserving frequencies below f_c. Q determines the resonance of frequencies around f_c.
- **Peaking**: boosts frequencies between f_{lc} and f_{hc} while preserving frequencies outside this frequency band. A notch filter cuts frequencies within the frequency band.
- **Allpass**: allows all frequencies, but modifies the phase relationships of the frequencies of the input signal. In general, the cut-off frequency f_c is where the phase response reaches $-90°$and Q sets the bandwidth.

These types of filters can be implemented via FIR or IIR filters whose coefficients determine G, f_c and Q.

Linear Time Invariant (LTI) systems such as FIR or IIR filters can be characterized by their impulse response, frequency response or transfer function (Smith, 2010). Thus, if the impulse response $h(n)$ of a system is known, the output signal $y(n)$ can be obtained via a discrete convolution with the input signal $x(n)$. Taking into account that convolving time-domain signals is the same as multiplying their frequency representation (Pestana, 2013), filtering is described by

$$y(n) = \sum_{k=-\infty}^{\infty} x(k) \cdot h(n-k) = x(t) * h(t) \rightarrow Y(f) = X(f) \cdot H(f), \tag{2.4}$$

where $*$ corresponds to the convolution operator, f denotes frequency, $H(f)$ is the frequency response of the system and $X(f)$, and $Y(f)$ are the frequency-domain representations of the input and output signals respectively.

In general, the convolution of two signals means filtering one with the other (Zölzer, 2011). In this manner, EQ can be described with a set of time-domain convolutions, where the frequency response of the filter bank can be expressed through various signals in the time-domain and the equalized audio signal is obtained through the respective convolutions.

2.2 MATCHED EQUALIZATION

Most existing methods for matched EQ show effective performance and have been implemented to obtain the filter coefficients in order to match a specific frequency response. Välimäki and Reiss (2016) provide a review of the different state-of-the-art approaches. These methods apply numerical optimization to find a transfer function that corresponds to a given complex or magnitude frequency response. Most common techniques are based on the equation error method (Smith, 2007), the Yule-Walker algorithm (Friedlander and Porat, 1984), the Steiglitz-McBride method (Jackson, 2008) and the frequency warped method (Härmä et al., 2000).

Within an automatic mixing framework (De Man et al., 2017), automatic EQ corresponds to automatically equalizing a mix based on music production criteria or the spectrum of a target mix. Perez-Gonzalez and Reiss (2009, 2011) explored multitrack EQ as a cross-adaptive audio effect, where the processing of an individual track depends on the content of all the tracks involved, then, the gains of a five filter, first order, filter bank were obtained based on a perceptual loudness weighing.

Given the raw multitrack recording and the final mixture, Barchiesi and Reiss (2010) used least-squares optimization to estimate the gains and f_0 of FIR filters. Mimilakis et al. (2013) proposed a pitch tracking system to perform automatic EQ within a mastering task, where the selected pitches are considered as center frequencies for a set of second order peaking filters. Ma et al. (2013) used least squares fitting to equalize an audio signal by using IIR filters with arbitrary frequency responses. A cross-adaptive EQ was implemented in (Matz et al., 2015), where center and cut-off frequencies of peaking and shelving filters were obtained through the minimization of spectral masking and source separation. Similarly, based on unmasking, Hafezi and Reiss (2015) obtained the center frequencies and gains of peaking filters and Ronan et al. (2018) attains the gains of a six-band equalizer based on second-order IIR filters.

Based on a perceptual task, Reed (2000) proposed a method where the model is trained manually by the users and through nearest neighbor techniques the equalizer gains are obtained in order to match the training data. In a similar approach, Sabin and Pardo (2009); Pardo et al. (2012) investigated a model that associates the gain of each frequency band with the user's training data. Vairetti et al. (2018) proposed an optimization method to match the complex frequency response of a target loudspeaker or room EQ.

In order to obtain optimal results, most automatic EQ implementations rely on fixed architectures of filter banks or require prior knowledge of the type of filters to be modeled. Therefore, in Chapter 4 we explore a general-purpose architecture capable of performing matched EQ given an arbitrary frequency response.

2.3 NONLINEAR AUDIO EFFECTS

Nonlinear processors correspond to all the signal processing systems that do not satisfy the linearity condition, i.e. systems that do not have the homogeneous or additive property (Chen, 1998). These systems are characterized by a harmonic and intermodulation distortion consisting of the introduction of intentional or unintentional harmonic or inharmonic frequency components that are not present in the input signal (Zölzer, 2011). Harmonic distortion is when the system introduces energy at every multiple, or harmonic, of each input frequency. Intermodulation distortion occurs when the system inserts frequency addition or subtraction between the frequency components of the input signal (Reiss and McPherson, 2014).

For example, given $x(n) = \sin(2\pi f_1 Tn) + \sin(2\pi f_2 Tn)$ where $x(n)$ is the input signal containing the frequency components f_1 and f_2 and T is the sampling period, harmonic distortion occurs when the output signal of a nonlinear system $y(n)$ contains $\sin(2\pi(Kf_1)Tn)$ where $K \in \mathbb{Z}$. Likewise, intermodulation distortion occurs when $\sin(2\pi(f_1 + f_2)Tn)$ or $\sin(2\pi(f_1 - f_2)Tn)$ are present in the output signal.

In the case of musical signals, due to aesthetic reasons, harmonic distortion is a desirable property of nonlinear audio processors, while intermodulation distortion is generally avoided. (Reiss and McPherson, 2014). Thus, nonlinear audio effects are extensively used by musicians and sound engineers and can be classified into two main types of effects: dynamic processors such as compressors or limiters; and distortion effects such as tube amplifiers (Zölzer, 2011).

2.3.1 *Distortion processors*

Distortion effects are mainly used for aesthetic reasons and are usually applied to electric musical instruments such as electric guitar, bass guitar, electric piano or synthesizers. The main sonic characteristic of these effects is due to their nonlinearities and the most common processors are overdrive, distortion pedals, tube amplifiers and guitar pickup emulators. These effects can be described by their characteristic curve or waveshaping nonlinearity. For example, a waveshaping transformation depends on the amplitude of the input signal and consists in using a nonlinear function, such as an hyperbolic tangent, to distort the shape of the incoming waveform (Puckette, 2007). An instance of a distortion waveshaping curve is

$$y(n) = \begin{cases} 1, & x(n) > 0.6 \\ \tanh(x(n)), & -0.6 \leqslant x(n) \leqslant 0.6 \\ -1, & x(n) < -0.6 \end{cases}, \tag{2.5}$$

where $x(n)$ and $y(n)$ are the input and output signals, respectively.

Eq. (2.5) can be considered as a memoryless or static nonlinear system. Although most distortion analog circuits are characterized by a short-term memory behavior, we consider distortion effects as time-invariant, where the output samples do not depend on time but only on the current input samples (Reiss and McPherson, 2014).

Distortion effects can be classified into the following types (Zölzer, 2011).

- **Overdrive**: linear audio effect at low input levels which becomes nonlinear when driven by higher input levels. It has a warm and smooth sound e.g. tube amplifiers.

- **Distortion**: mainly operates in the nonlinear region of the waveshaping curve and saturates or clips when reaching the upper or lower input levels. It covers a tonal area from warmth to crunch.
- **Fuzz**: operates completely in the nonlinear region. It is characterized by being a more aggressive style distortion.

2.3.2 *Dynamic range processors*

Dynamic range processors are nonlinear time-invariant audio effects with long temporal dependencies, and their main purpose is to alter the variation in volume of the incoming audio. This is achieved with a varying amplification gain factor, which depends on an envelope follower along with a waveshaping nonlinearity. Long-term memory behavior describes these effects, since the output depends on the current and previous values of the input samples. These effects tend to introduce a low amount of harmonic distortion, while for tube amplifiers a strong distortion is desired (Zölzer, 2011).

Usually, the amount of gain depends on different parameters such as *threshold*, *ratio*, *attack* and *release times* and *knee*. For a compressor: *threshold* is the level above which compression starts, *ratio* determines the amount of compression applied, normally is set as the input/output rate for levels larger than the *threshold*. Once the signal level rises above or falls below the threshold level, *attack* and *release times* establish how fast the compressor starts and stops applying compression, respectively. *Knee* controls the smoothness of the transition at the threshold point.

The various types of dynamic range processors can be classified into the following types of effects (Zölzer, 2011; Reiss and McPherson, 2014).

- **Compressor**: reduces the dynamics of the input signal by applying a variable gain. Based on the *ratio*, this decreased gain is applied to the input signals that are above the *threshold*. This can further increase the overall levels of the output signal thus boosting the loudness.
- **Expander**: does the opposite of the compressor by increasing the dynamics of the input signals that are above the *threshold*.
- **Multiband Compressor**: applies compression to selected frequency bands via a filter bank which splits the input signal, so each band is individually compressed.
- **Limiter**: eliminates any dynamics above the *threshold* by keeping the output level constant, i.e. ratio is $\infty/1$.

- **Noise gate**: eliminates any dynamics below the *threshold* by keeping the output level muted.

Overall, distortion effects and dynamic range processors are based on the alteration of the waveform which leads to various degrees of harmonic distortion. The nonlinear behavior of certain components of the effects' circuit performs this alteration, which can be seen as a waveshaping nonlinearity applied to the amplitude of the incoming audio signal in order to add harmonic and inharmonic overtones.

See Zölzer (2011) for a further review of other types of nonlinear audio processors such as *de-esser*, *tape saturation*, *exciters* and *enhancers*.

2.4 MODELING OF NONLINEAR AUDIO PROCESSORS

Since a nonlinear system cannot be characterized by its impulse response, frequency response or transfer function (Smith, 2010), digital emulation of distortion effects has been extensively researched (Pakarinen and Yeh, 2009).

Different methods have been proposed such as *memoryless static waveshaping* (Möller et al., 2002), where input-output measurements are used to approximate the nonlinearity; *dynamic nonlinear filters* (Karjalainen et al., 2006), where the waveshaping curve changes its shape as a function of the input signal or system-state variables; *circuit simulation* techniques (Yeh et al., 2008; Yeh and Smith, 2008; Yeh et al., 2010), where a complete study of the analog circuitry is performed and nonlinear filters are derived from the differential equations that describe the circuit; and *analytical methods* (Abel and Berners, 2006; Hélie, 2006), where the nonlinearity is modeled via Volterra series theory or nonlinear black-box approaches such as Wiener and Hammerstein models (Gilabert Pinal et al., 2005; Eichas and Zölzer, 2016, 2018).

Modeling of dynamic range processors, such as *compressors*, has been based on black-box methods such as *system identification* techniques, where a model is structured using only the measurements of the input and output signals; and white-box methods where a complete study of the internal circuit is carried out, such as *circuit simulation*.

In (Kröning et al., 2011), state-space models are used to simulate the circuit of a specific analog guitar *compressor*. Black-box (Eichas et al., 2017) and gray-box (Gerat et al., 2017) modeling of general-purpose dynamic range compressors has been investigated via input-output measurements and optimization routines.

The latter differs from black-box modeling, since gray-box approaches use some information about the circuit together with input-output signals.

Generalization among different audio effect units is usually difficult since these modeling methods are often either simplified or optimized to a very specific circuit. This lack of generalization is accentuated when we consider that each audio processor is also composed of components other than the nonlinearity. These components also need to be modeled and often involve filtering before and after the nonlinearity, as well as short and long temporal dependencies such as hysteresis or attack and release gates.

Automatic dynamic range compression has also been investigated recently. Similar to automatic EQ, it corresponds to the use or modeling of dynamic range processors in order to match a target sound (Sheng and Fazekas, 2017) or to perform certain audio engineering tasks, such as mixing (Maddams et al., 2012; Giannoulis et al., 2013; Ma et al., 2015) or mastering (Hilsamer and Herzog, 2014). This is achieved based on the selection of low-level audio features to cross-adaptively (Verfaille et al., 2006) control or estimate the parameters of the respective dynamic range processors. These implementations are based on fixed architectures of the processors and also on predetermined low-level audio features, which makes the models too specific for a given task. Thus, the specificity of these methods certainly represents a restriction since mixing and mastering are carried out with a wide variety of audio processors. Besides, the models cannot be extended to complex or unusual sounds.

In Chapter 5 and Chapter 6 we explore general-purpose architectures to model nonlinear audio effects with short-term and long-term memory, respectively. Table 2.1 includes a summary of the different approaches for virtual analog modeling of noninear audio effects.

2.5 TIME-VARYING AUDIO EFFECTS

Audio processors whose parameters are modified periodically over time are often referred as time-varying or modulation based audio effects. Thus, time-varying audio effects involve audio effect units that include a modulator signal within their analog or digital implementation (Reiss and McPherson, 2014). These modulator signals are in the low frequency range (usually below 20 Hz). Their waveforms are based on common periodic signals such as sinusoidal, squarewave or sawtooth oscillators and are often referred to as Low Frequency Oscillators (LFOs).

The LFO periodically modulates certain parameters of the audio processors to alter the timbre, frequency, loudness or spatialization characteristics of the audio. This differs from time-invariant audio effects which do not change their behavior over time. Based on how the LFO is employed and the underlying signal processing techniques used when designing the effect units, we can classify modulation based audio effects into *time-varying filters* such as phaser or wah-wah; *delay-line based* effects such as flanger or chorus; and *amplitude modulation* effects such as tremolo or ring modulator (Zölzer, 2011).

2.5.1 *Time-varying filters*

Time-varying filters correspond to filters whose parameters, such as gain, cut-off frequency and Q factor, can be controlled by modulator signals. Special sonic characteristics are obtained when different types of modulations are applied to the various types of filters, as in the following effect units.

- **Phaser**: or *phase shifter* is implemented through a cascade of allpass or notch filters. The characteristic sweeping sound of this effect is obtained by modulating the center frequency of the filters, which creates phase cancellations or enhancements when combining the filter's output with the input audio. This phase shifting creates a series of notches in the frequency domain which result from destructive interference. Thus, the sound of the phaser results from the time-varying control of these notch frequencies, which can be designed or placed arbitrarily along the frequency spectrum.
  The transfer function of a phaser implemented via a first-order IIR allpass filter is

$$H(z) = \frac{z^{-1} + c}{1 + cz^{-1}}, \tag{2.6}$$

  where

$$c = \frac{\tan(\pi f_c T) - 1}{\tan(\pi f_c T) + 1}, \tag{2.7}$$

  and f_c is the cut-off frequency (Zölzer, 2011). Phasers are often modulated via exponential motion of the notch frequencies (Reiss and McPherson, 2014), thus, the time-varying behaviour can be described with the following modulation:

$$f_c = \alpha^{\mathrm{LFO}(2\pi f_{\mathrm{LFO}} T n)}, \tag{2.8}$$

  where α is the *sweep width* or frequency range in Hz of the modulation and LFO is the periodic signal with *rate* f_{LFO}.

- **Wah-wah**: is implemented with a bandpass, peaking, or resonant lowpass filter with a variable center or cutoff frequency f_c, usually controlled by a pedal. If f_c is modulated by an LFO or an envelope follower, the effect is commonly called *auto-wah*. The effect is characterized by introducing a vowel-like sound to the input signal, since the bandwidth f_b corresponds to vocal formants in the spectrum. Thus, the typical ranges of the designed frequency bands are between 400 Hz and 1200 Hz (Reiss and McPherson, 2014).
  The transfer function of an auto-wah modulated by an LFO and implemented via a second-order bandpass filter is (Zölzer, 2011)

$$H(z) = \frac{1}{2}\left[1 - \frac{-c + d(1-c)z^{-1} + z^{-2}}{1 + d(1-c)z^{-1} - cz^{-2}}\right], \tag{2.9}$$

  where

$$c = \frac{\tan(\pi f_b T) - 1}{\tan(\pi f_b T) + 1}, \tag{2.10}$$

  and

$$d = -\cos(2\pi f_c T). \tag{2.11}$$

  The LFO that controls the center frequency f_c is

$$f_c = f_{min} + (1 + \alpha \mathrm{LFO}(2\pi f_{\mathrm{LFO}} T n)), \tag{2.12}$$

  where f_{min} is the lowest center frequency.
  The second type of auto-wah is via an envelope follower. Thus, f_c is controlled by the amplitude of the signal. This modulation works similarly and has the same controls and long-term memory behaviour as the varying-gain of the compressor in Section 2.3.2.

2.5.2 *Delay-line based effects*

Delay lines have the function of introducing a time delay between their input and output. Accordingly, delay-line based effects rely on the modulation of the length of the delay lines. Multiple variations of this modulation result in different sonic characteristics and audio effects, such as the following (Smith, 2010).

- **Flanger**: or *flanging* is an effect based on the mixing of two identical signals, where one of the signals is delayed with a modulated delay time. Since a comb filter consists of adding a delayed version of a signal to itself, a flanger is implemented in the digital domain via a modulated feedforward comb filter which causes constructive and destructive interference. Unlike the phaser, the notch

and peak frequencies caused by the flanger's sweep comb filter effect are equally spaced in the spectrum, thus causing the known metallic sound associated with this effect.
The difference equation of a flanger implemented via a first-order FIR comb filter is (Smith, 2010)

$$y(n) = x(n) + gx\big(n - (M + \alpha \mathrm{LFO}(2\pi f_{\mathrm{LFO}} T n))\big), \tag{2.13}$$

where the delay M is generally within the range of 1 to 10 ms and is modulated according to a periodic LFO, typically triangular or sinusoidal. α is the *width* of the delay modulation and the *depth* g sets the amount of delayed signal which is mixed with the input.

- **Chorus**: occurs when mixing the input audio with delayed and pitch modulated copies of the original signal. This is similar to various musical sources playing the same instrument but slightly shifted in time. This differs from a flanger since a chorus uses longer delay times, usually from 20 to 30 ms, thus creating a more subtle effect (Smith, 2010).
The chorus implementation is identical to Eq. (2.13), the only difference is a longer delay M. Also, since the effect recreates various sources playing in unison, it is usual to implement the chorus with several delayed copies of the input signal, whose delay times vary in a small and random way.
- **Vibrato**: consists of a periodic frequency shift. This is achieved by modulating the input delay time, since variable delay lines can alter the pitch of the input signal (Puckette, 2007). Thus, to create this pitch shift, the delay is periodically lengthened and shortened thereby changing the playback speed of the sampled audio (Reiss and McPherson, 2014).
Unlike a chorus or flanger, the input signal is not mixed with the output of the delay line. The difference equation of a vibrato implemented with a modulated delay line is shown below.

$$y(n) = x\big(n - (M + \alpha \mathrm{LFO}(2\pi f_{\mathrm{LFO}} T n))\big) \tag{2.14}$$

Typical values for the delay time are 5 to 10 ms and a sinusoidal LFO is often used to emulate musical vibrato (Reiss and McPherson, 2014).

2.5.3 *Amplitude modulation effects*

Effect units based on amplitude modulation consist of audio processors where the amplitude or gain of the input signal varies periodically. This is achieved with a modulator signal and various techniques have been designed to accomplish this transformation. The main audio effects based on this principle are the following.

- **Tremolo**: consists of an amplitude modulation where an LFO is used to directly vary the amplitude of the incoming audio, creating in this way a perceptual temporal fluctuation. This modulation is achieved by directly multiplying the input signal with the modulator LFO.

$$y(n) = x(n)\,(1 + \alpha \mathrm{LFO}(2\pi f_{\mathrm{LFO}} T n)) \tag{2.15}$$

  Where α is the *depth* of the amplitude modulation. Typical values for the rate f_{LFO} are from 0.5 to 20 Hz and sine, triangular and square waves are usually the LFO waveform.

- **Ring Modulation**: is similar to the amplitude modulation of the tremolo, although the modulation is achieved by having the input audio multiplied by a signal with higher carrier frequencies. The characteristic sound of this effect is based on the multiplicative properties of its input and carrier signal. Given a modulating carrier signal $m(n)$, which is usually a sinusoidal signal, the output of a ring modulator is based on the following equation.

$$y(n) = x(n)m(n) \tag{2.16}$$

  Thus, with $x(n)$ and $m(n)$ as sinusoids of two frequency components f_1 and f_2, respectively, and taking into account that the multiplication of two sinusoids results in the sum and difference of its frequencies (Puckette, 2007), the ring modulation between these signals is shown below.

$$y(n) = \sin(2\pi f_1 T n)\cos(2\pi f_2 T n) \tag{2.17}$$

$$y(n) = \frac{1}{2}\left[\sin(2\pi(f_1 + f_2)Tn) + \sin(2\pi(f_1 - f_2)Tn)\right] \tag{2.18}$$

  This type of modulation is the same as the intermodulation distortion described in Section 2.3. If the carrier signal comprises several spectral components, the same effect is produced in each of the components, thus obtaining a quite particular and complex audible effect (Zölzer, 2011).

In the analog domain, this effect is commonly implemented with a diode bridge, which adds a nonlinear behavior and a distinct sound to this effect unit.

- **The Leslie speaker**: is a type of modulation based effect that combines amplitude, frequency and spatial modulation. It consists of a vacuum-tube amplifier and crossover filter followed by a rotating *horn* and rotating *woofer* inside a wooden cabinet. This effect can be interpreted as a combination of tremolo, Doppler effect and reverberation (see Section 2.7).

  A crossover filter is a filter bank that splits the incoming audio into two or more adjacent frequency bands. It is usually implemented with lowpass, bandpass and highpass IIR filters. The rotating speakers are preceded by a 800 Hz passive crossover filter, i.e. the *horn* and *woofer* amplify frequencies above and below 800 Hz respectively. The rotation of both speakers is achieved with slow and fast A.C. induction motors (Henricksen, 1981).

  Two speeds are available for each rotating speaker; *tremolo* for a fast rotation and *chorale* for a slow rotation. The rotation frequency of the *horn* is approximately 7 Hz and 0.8 Hz for the *tremolo* and *chorale* settings respectively, while the *woofer* has slower speed rotations (Herrera et al., 2009).

  Amplitude modulation is achieved by the increase and decrease of sound intensity due the rotation of the speakers. Frequency modulation occurs due to the Doppler effect. Thus, as the speakers rotate toward the listener; the pitch increases, as they rotate away, the pitch decreases. Furthermore, since musical sources are projected throughout the respective listening room, spatial modulation occurs as the musical sources undergo multiple reflections (Henricksen, 1981). In addition, by moving closer to the rotating speaker, reverberation occurs along the walls of the wooden cabinet, which also acts as a resonant body thus increasing the wide and unique sonic characteristics of this audio processor.

  Therefore, the Leslie speaker cabinet is an electromechanical audio effect consisting of a highly complex nonlinear and time-varying spatial system.

2.6 MODELING OF TIME-VARYING AUDIO PROCESSORS

Most of the time-varying processors can be implemented directly in the digital domain through the use of digital filters and delay lines. Nevertheless, specific perceptual qualities of the analog implementations of these effect units are due to the nonlinearities introduced by certain circuit components, such as operational amplifiers, diodes, transistors or integrated chips. Methods for modeling such audio processors remain an active field of research, which mainly involve circuit modeling and optimization for specific analog components.

Therefore, modeling time-varying audio effects has been explored mostly via white-box methods. In (Huovilainen, 2005), *phasers* implemented via Junction Field Effect Transistors (JFET) and Operational Transconductance Amplifiers (OTA) are modeled using circuit simulation techniques that discretize the differential equations that describe these components. Using a similar circuit modeling procedure, delay-line based effects are also modeled, such as *flanger* and *chorus* as implemented with Bucket Brigade Delay (BBD) chips.

BBD circuits have been widely used in analog delay-line based effect units and several digital emulations have been investigated. Raffel and Smith (2010) emulated BBD devices through circuit analysis and electrical measurements of the linear and nonlinear elements of the integrated circuit. Holters and Parker (2018) modeled BBDs as delay-lines with fixed length but variable sample rate.

Based on BBD circuitry, a *flanger* effect was modeled in (Mačák, 2016) via the nodal DK-method. This is a common method in virtual analog modeling (Yeh, 2012) where nonlinear filters are derived from the differential equations that describe an electrical circuit. In (Holters and Zölzer, 2011), a *wah-wah* pedal is implemented using the nodal DK-method and the method is extended to model the temporal fluctuations introduced by the continuous change of the pedal. In (Eichas et al., 2014), the *MXR Phase 90 phaser* effect is modeled via a thorough circuit analysis and the DK-method. This effect unit is based on JFETs, and voltage and current measurements were performed to obtain the nonlinear characteristics of the transistors.

Amplitude modulation effects such as an analog *ring modulator* were modeled in (Parker, 2011b), where the diode bridge is emulated as a network of static nonlinearities. The *Leslie speaker* cabinet represents a special case of modulation based audio effects, since amplitude and frequency modulation occurs along with the reverberation and structural resonance of the wooden cabinet. In (Smith et al., 2002), the rotating *horn* of the *Leslie speaker* is modeled via varying delay-lines, artificial reverberation and physical measurements from the rotating loudspeaker. Likewise, (Pekonen et al., 2011; Herrera et al., 2009) modeled the *Leslie speaker horn* and *woofer* through time-varying spectral delay filters and time-varying FIR filters respectively. In these *Leslie speaker* emulations, various physical characteristics of the effect are not taken into account, such as the frequency-dependent directivity of the loudspeakers or the effect of the wooden cabinet.

In (Kiiski et al., 2016), based on all-pass filters and multiple measurements of impulse responses, a gray-box modeling method for linear time-varying audio effects is proposed. The method was based on input-output measurements but the time-varying filters were based on knowledge of analog *phasers*. Another method to model time-varying audio effects is discretizing electrical circuit elements via Wave Digital Filters (WDF) (De Sanctis and Sarti, 2009). The Hammond organ vibrato/chorus was modeled using WDFs in (Werner et al., 2016), and Bogason and Werner (2017) performed circuit modeling through WDFs to emulate modulation based effects that use OTAs.

In Chapter 6 we explore general-purpose architectures to model several linear and nonlinear time-varying audio effects. In Chapter 7 we show virtual analog models of electromechanical nonlinear time-varying processors such as the rotating *horn* and rotating *woofer* of a *Leslie speaker* cabinet. Table 2.1 includes a summary of the different approaches for virtual analog modeling of time-varying audio effects.

2.7 ARTIFICIAL REVERBERATION

Reverberation occurs when delayed and attenuated copies of the direct sound appear as reflections. Each reflection is frequency dependent and defined by the directivity of the sound source and the physical attributes of the reflecting surfaces (Zölzer, 2011). The reflections that occur immediately after the sound event are considered *early reflections*. These are not perceived individually, but as a combination of all the reflections. *Late reflections* appear more randomly and are characterized by increasing the echo density, that is, the number of echoes per time unit, which generates a diffuse reverberation. Perceptually, the reflections merge into a sound of continuous decay (Reiss and McPherson, 2014). Therefore, the impulse response of an acoustic space is often divided into: direct sound, early reflections and late reflections.

In the music and film industry, artificial reverberation was initially researched as a way of approximating the reflections occurring in room acoustics. This led to techniques that simulate reverberation, such as chamber, plate, spring and digital reverberators (Välimäki et al., 2012).

2.7.1 *Digital reverberators*

Reverberation is approximately linear and time-invariant. Thus, most of the digital methods that emulate reverberation attempt to match the perceptual characteristics of the respective impulse responses. There are various methods which rely on digital filters, delay networks and convolution-based algorithms.

- **Comb and Allpass filters**: Schroeder and Logan (1961) proposed a digital reverberator based on a parallel filter bank of feedback comb filters and a series of allpass filters. The difference equation of a feedback comb filter is

  $$y(n) = x(n-M) + gy(n-N), \tag{2.19}$$

  where M and N are the input and output delay times respectively, and g is the feedback coefficient. When M is large, the impulse response of the filter is heard as discrete echoes or reflections which are exponentially decaying and uniformly spaced in time. Thus, by adjusting the delay time of each feedback comb filter, the parallel filter bank simulates the reverberant reflections of an acoustic environment.

  The allpass filters emulate the late reflections of the reverberator, for which the difference equation is

  $$y(n) = x(n-M) - gx(n) + gy(n-M). \tag{2.20}$$

  Each allpass filter also produces a series of decaying echoes, thus increasing overall echo density, and expanding the previous single reflections into many reflections. This simulates the diffuse reverberation.

  Moorer (1979) included an FIR lowpass into the feedback loop of each comb filter to model more accurately the early reflections of the impulse response. This is since the feedback comb filters do not attenuate the high frequencies and this do not replicate typical physical attributes of the reflecting surfaces. More complex structures can be built on the above methods, such as feedback delay networks (Jot and Chaigne, 1991; Smith, 2007).

- **Sparse pseudo-random algorithms**: Sparse FIR filtering has proven to be an efficient digital reverberation method (Rubak and Johansen, 1999; Välimäki et al., 2012). Dispersive reflections are approximated via FIR filters within feedback loops with sparsely placed coefficients. These coefficients are often determined by a pseudo-random number sequence such as *velvet noise* (Järveläinen and Karjalainen, 2007).

Thus, sparse pseudo-random reverberation algorithms use discrete coefficient values such as -1 and +1, where each one of the coefficients follows an interval of T_s samples while all the other samples are zero. This greatly improves computational complexity, since this type of filtering can be implemented without multiplications (Välimäki et al., 2012).

In order to have a high-quality reverberation, 2000 to 4000 coefficients per second are needed (Rubak and Johansen, 1999). Also, different types of envelopes are applied to the output of the FIR filters, this in order to avoid audible artifacts due to repetition in the feedback loop (Järveläinen and Karjalainen, 2007).

- **Convolutional**: consists in convolving the input signal with a recorded or estimated impulse response of an acoustic environment. The impulse response can be measured via sinusoidal sweeps, where a linear or logarithmic sweep of constant amplitude is recorded in the respective acoustic space (Farina, 2000). Convolutional reverb is implemented via FIR filtering, where each coefficient corresponds to each value of the impulse response, resulting in a very high-order filter (Välimäki et al., 2012).

  A common approach to overcome this computational load is to use block-based convolutions and comb and allpass filters (Reiss and McPherson, 2014). Thus, convolution via FIR filtering generates the early reflections of the impulse response, and IIR filters, such as feedback comb and allpass filters, generate the diffuse reverberation.

2.7.2 *Electromechanical reverberators*

Electromechanical reverberators such as *plate* and *spring* were first used and researched as means to substitute real room reverberation. Currently, they are often used in music production for aesthetic reasons due to their particular sonic characteristics.

- **Plate reverb**: is based on a large metal plate which vibrates due to a moving-coil transducer attached to its centre. This transducer is fed with an amplified dry input signal and the plate vibrations are read by a pickup sensor and further amplified (Kuhl, 1958). The plate reverb sound is different from room acoustic reverberation as the speed of sound is faster in metal than in air, increasing the echo density (Bilbao, 2009).

  This particular response is due to the mechanical and physical characteristics of the plate. Unlike wave propagation in air, phase and group velocities are

not constant within a plate. Thus, high frequencies travel faster than low frequencies, which generate the diffused and noisy response. In addition, several reflections are missing since the mode density of the plate is constant, whereas in an acoustic space the mode density is frequency dependent (Zölzer, 2011). The higher echo density and constant mode density characterizes the "smooth" late reflections of plate reverberators.
Typically, the reverberators are constructed with steel or gold plates, with thickness and surface area of 0.5 mm and 2 m^2, respectively.

- **Spring reverb**: is based on one or various helical springs suspended under low tension, attached to a magnetic bead and driven via an electromagnetic coupling (Parker and Bilbao, 2009). The input audio source is transduced to spring vibrations which are read through a pickup sensor at the opposite end. The distinct sound of spring reverb is due to the various types of vibrations that occur, transverse and longitudinal, which cause a peculiar combination of wave and dispersive propagation (Zölzer, 2011).
  Due to the physical characteristics of the helical structures, the frequency response is characterized by a cutoff frequency f_c which depends on the wavenumber, i.e. the spatial frequency of a wave. Thus, the spring exhibits a group velocity increment for wavenumbers above f_c, which is perceived as distorted echoes at higher rates. Also, the group velocity decreases as the wavenumber approaches f_c, thus respective frequency components are distorted at slower rates. The group velocity is constant for low wavenumbers, which is perceived as strong dispersive echoes (Bilbao, 2013).
  Common spring reverberators are built with steel helical springs of uncoiled length of 5 m, coil radius of 4 mm and wire diameter of 0.2 mm.

2.8 MODELING OF ELECTROMECHANICAL REVERBERATORS

Although originally developed as substitutes for room reverberators, digital implementations of these devices have been widely researched due to their distinctive sounds which has been of great interest among musicians, music producers and sound engineers.

Plate reverberation has been emulated with different approaches such as numerical simulation techniques, where a finite difference scheme (Bilbao et al., 2006; Bilbao, 2007; Arcas and Chaigne, 2010) or a modal description (Willemsen et al., 2017; Ducceschi and Webb, 2016) is derived from the differential equations that describe the motion of the plate; and hybrid digital filter-based algorithms (Abel et al., 2009; Greenblatt et al., 2010; Lee et al., 2010), where convolutional impulse responses and feedback delay networks are used to model the desired impulse response.

Similarly, modeling of *spring* reverberation has been explored as wave digital filters (Abel et al., 2006), to explicitly model the wave and dispersive propagation; numerical simulation techniques such as finite difference schemes (Bilbao and Parker, 2009; Parker and Bilbao, 2009; Bilbao, 2013), and nonphysical modeling techniques (Välimäki et al., 2010; Parker, 2011a), where chains of allpass filters and varying delay lines are used to approximate the dispersive and reverberant features of *spring* reverb.

The modeling of these audio processors and their salient perceptual qualities remains an active research field. Their mechanical elements together with their analog circuitry yield a spatial system which is difficult to fully emulate digitally. Most of the methods are based on complete physical models or perceptual simplifications , thus, such models are not easily transferable to different artificial reverberators or cannot capture the full response of the system. In Chapter 8 we explore the capabilities of deep neural networks to learn the respective transformation and perceptual qualities of these electromechanical reverberators.

2.9 conclusion

In this chapter we introduced the signal processing properties of the different types of audio effects as well as their modeling methods and technical challenges. This background is required to motivate a deep learning approach to the modeling of audio effects. Therefore, the technical chapters related to the audio effects and modeling methods presented in this chapter correspond to; Chapter 4 where we explore a deep learning architecture for matched EQ; Chapter 5 where we propose a network for modeling nonlinear audio effects with short-term memory; Chapter 6 where we investigate deep neural networks for modeling audio effects with long temporal dependencies; Chapter 7 where we show virtual analog models of a *tube-amplifier*, a *transistor-based limiter* and a *Leslie speaker* cabinet; and Chapter 8 where we propose a network to model artificial reverberators such as *plate* and *spring* reverb.

Table 2.1: Summary of approaches for virtual analog modeling of nonlinear, time-varying and time-invariant audio effects with short-term and long-term memory.

Type	Audio effect	Approach		Reference
nonlinear with short-term memory	tube amplifier	static waveshaping		Möller et al. (2002)
	tube amplifier	dynamic nonlinear filters		Karjalainen et al. (2006)
	distortion	static waveshaping & numerical methods		Yeh et al. (2008)
	distortion	circuit simulation	K-method & WDF	Yeh and Smith (2008)
	distortion	circuit simulation	Nodal DK	Yeh et al. (2010)
	speaker, amplifier	analytical method	Volterra series	Abel and Berners (2006)
	Moog ladder filter	analytical method	Volterra series	Hélie (2006)
	power amplifier	black-box	Wiener & Hammerstein	Gilabert Pinal et al. (2005)
	distortion	black-box	Wiener	Eichas and Zölzer (2016)
	tube amplifier	black-box	Wiener-Hammerstein	Eichas and Zölzer (2018)
	equalization	black-box	end-to-end DNN	Publication I
	tube amplifier	black-box	end-to-end DNN	Schmitz and Embrechts (2018)
	tube amplifier	black-box	end-to-end DNN	Zhang et al. (2018)
	equalization & distortion	black-box	end-to-end DNN	Publication II
	tube amplifier	black-box	end-to-end DNN	Damskägg et al. (2019)
	tube amplifier, distortion	black-box	end-to-end DNN	Wright et al. (2019)
	distortion	circuit simulation & DNN		Parker and Esqueda (2019)
time-dependent nonlinear	compressor	circuit simulation	state-space	Kröning et al. (2011)
	compressor	black-box	system-identification	Eichas et al. (2017)
	compressor	gray-box	system-identification	Gerat et al. (2017)
	compressor	black-box	end-to-end DNN	Hawley et al. (2019)
time-varying	ring modulator	static waveshaping		Parker (2011b)
	phaser	circuit simulation	numerical methods	Huovilainen (2005)
	phaser	circuit simulation	Nodal DK	Eichas et al. (2014)
	modulation based with OTAs	circuit simulation	WDF	Bogason and Werner (2017)
	flanger with BBDs	circuit simulation	Nodal DK	Mačák (2016)
	modulation based with BBDs	circuit simulation & system-identification		Bogason and Werner (2017)
	Leslie speaker horn	digital filter-based & system identification		Smith et al. (2002)
	Leslie speaker horn & woofer	digital filter-based		Pekonen et al. (2011)
	Leslie speaker horn & woofer	digital filter-based		Herrera et al. (2009)
	flanger, chorus	digital filter-based		Huovilainen (2005)
	modulation based with BBDs	digital filter-based		Holters and Parker (2018)
	modulation based	gray-box	system-identification	Kiiski et al. (2016)
	modulation based & compressor	black-box	end-to-end DNN	Publication III

3 DEEP LEARNING FOR AUDIO

This chapter covers the fundamentals of deep learning for audio and can be considered as a brief tutorial on the field. The chapter starts with an introduction to relevant terms of machine learning and ends with a review of the different applications of deep learning for music, as well as for audio effects modeling. The reader with knowledge of deep learning for audio can skip Sections 3.1 to 3.7 and go directly to Sections 3.8 and 3.9. Nevertheless, it is recommended to review Section 3.3, as it introduces *Smooth Adaptive Activation Functions* which is a specific activation function that is used in most of the proposed models of this thesis.

3.1 MACHINE LEARNING CONCEPTS

Machine learning is a field of Artificial Intelligence (AI) that is based on the construction of mathematical and statistical models directly from the data. Therefore, through different learning algorithms, the models can make the respective predictions or decisions given new examples of input data. Most of the machine learning tasks can be divided into the following types (Bishop, 2006).

- **Classification**: The main objective of this task is to identify to which set of categories or classes a new input belongs. Thus, the model learns a function $f : \mathbb{R}^n \mapsto \{1, ..., k\}$, where k is the number of classes. An example of a classification task is genre recognition, where the input is a representation of a particular song, and the output is a probability distribution over the respective genre classes.
- **Regression**: This task is based on the prediction of a continuous numerical value given a new input. For instance, the model learns a function $f : \mathbb{R}^n \mapsto \mathbb{R}$. An example of a regression task is the prediction of the next audio sample or the estimation of musical tempo. All modeling tasks in this thesis are regression tasks.

To perform these tasks, machine learning methods can be classified according to the arrangement of the dataset or training data. A *dataset* is a collection of examples or data points, where each example consists of a feature vector $\mathbf{x} \in \mathbb{R}^n$, which represents a quantitative measure of the respective input data. For example, the features of an audio signal may be the sample values of the waveform or the respective frequency representation. Most of the learning algorithms can be divided into the following categories (Goodfellow et al., 2016).

- **Supervised learning**: It is based on learning a function from a set of training examples, each of which consists of pairs of an input feature vector $\mathbf{x}$ and its desired output y. The trained models learn a function that maps the input to an output, thus predicting y from $\mathbf{x}$ usually by estimating the conditional probability $p(y|\mathbf{x})$. Overall, most of the machine learning applications for music use this method, e.g. instrument or mood classification, beat tracking, melody extraction, score alignment, etc.
- **Unsupervised learning**: It is based on learning representations or transformations of the input data without any targets or labels. The main objective is to learn useful properties of the structure of the dataset, thus, given various examples $\mathbf{x}$, the trained models often learn the probability distribution $p(\mathbf{x})$. Unsupervised learning algorithms are often used as an initial step before solving a particular task through a supervised learning method, e.g. feature learning is often done before performing audio classification (Hamel and Eck, 2010; Fu et al., 2010).

3.2 deep neural networks

Deep learning is a field of machine learning that is based on *learning* of representations directly from the data. This is usually achieved by stacking successive layers of artificial neural networks. The number of stacked layers determines the depth of the model, thus giving the characteristic name of this field as the models are usually deep.

A deep neural network can be considered as a function approximator $\hat{f}(\mathbf{x};\mathbf{W})$, which learns the parameters or weights $\mathbf{W}$ in order to match the function $f(\mathbf{x})$. Thus, within a supervised learning framework, where each example $\mathbf{x}$ has its corresponding target $y = f(\mathbf{x})$, the objective of the model is to produce an output $\hat{y} = \hat{f}(\mathbf{x};\mathbf{W})$ which closely approximates y (Goodfellow et al., 2016).

Artificial neural networks

Artificial neural networks are systems characterized by the number of units and layers and loosely based on the neurons of biological brains (Goodfellow et al., 2016). Each *layer* within a network is represented as a vector, where each element or scalar value corresponds to a single *unit* or *node*. The layers can be classified into input, hidden and output layers. Input and output layers are the first and last layers, respectively, while the hidden layers are the layers in between input and output layers. The number of units of the hidden layers determines the *width* of the model.

For a hidden or output layer, each unit is connected to the units of the previous layer through the respective weights. Therefore, a weighted sum of the inputs for each unit is often calculated to obtain the numerical value of the vector representing the layer. A bias unit storing the value of 1 is usually added to the input and hidden layers. These units are not connected to the previous layer and represent an additive weight learned for each layer. Moreover, a nonlinear function $f()$, which is generally known as an *activation function* (see Section 3.3), is applied to the weighted sum of each unit in order to produce the final output.

Designing deep neural networks

Before the learning or training process begins, the *hyperparameters* of the network must be defined. Hyperparameters differ from the weights $\mathbf{W}$, since the former are constant and non-trainable parameters that define the architecture of the network. Examples of hyperparameters are the type and structure of the layers, the loss function and the optimizer. (Chollet, 2018).

- **Layers**: When learning representations and solving a specific deep learning task, different layers are appropriate for each type of data transformation. For example, *dense layers* are used when processing simple vector data. *Convolutional layers* are used when the spatial structure of the input is significant, such as one-dimensional or two-dimensional representations of audio. *Recurrent layers* are used when processing sequence data where the temporal structure of inputs is relevant. The number of units or hidden layers or the type of activation functions are examples of hyperparameters when choosing a layer.
- **Loss function**: This is the quantity that represents the performance of the model during the training process. Thus, for the current parameters $\mathbf{W}$, the loss or cost function $J(\mathbf{W})$ measures the difference between the prediction $\hat{y}$ and the target y and its numerical value is the feedback used for learning. For regression tasks, the most common loss functions are the *L1* and *L2* distances; *mean absolute error* (*mae*) and *mean squared error* (*mse*), respectively.

$$mae(y, \hat{y}) = \frac{\sum_{i=1}^{N} |y_i - \hat{y}_i|}{N} \tag{3.1}$$

$$mse(y, \hat{y}) = \frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{N} \tag{3.2}$$

Where N is the number of examples. Various cost functions are used for classification tasks, such as *categorical* or *binary cross-entropy*, *Kullback–Leibler divergence* (KL) or *hinge loss* (see Bishop, 2006; Goodfellow et al., 2016).
In addition, to improve the generalization capabilities of DNNs, weights and activity regularization are introduced. This consists of adding a penalty term to the cost function, which is usually the norm of the *L1* or *L2* of the respective weights or outputs of neurons (Goodfellow et al., 2016).

- **Optimizer**: This is the iterative and gradient-based algorithm that updates the weights of the network based on the numerical value of the loss function. Many methods are commonly used, which all are based in a variant implementation of Stochastic Gradient Descent (SGD) (Amari, 1993). Gradient descent is based on the chain rule and computes the directional derivative of a composite function, such as a DNN. It consists of minimizing the loss function $J(\mathbf{W})$ by updating the values of $\mathbf{W}$ towards the opposite direction of its gradient $\nabla J(\mathbf{W})$. SGD is characterized by using the mean gradient of randomly selected batches of m examples (Goodfellow et al., 2016). For a single *iteration* across a selected batch, the weight update can be described as follows.

$$\mathbf{W} \rightarrow \mathbf{W} - \alpha \nabla J(\mathbf{W}) \tag{3.3}$$

The *batch size* m is one of the hyperparameters of the optimizer together with the *learning rate* α and the initial value of $\mathbf{W}$. The initialization of $\mathbf{W}$ is generally done through sampling variations of normal or uniform distributions (see LeCun et al., 2012; Glorot and Bengio, 2010). The learning rate α is usually reduced during training as the loss approaches a local minimum.
Different optimizers based on this principle are commonly used, such as *Adam* (Kingma and Ba, 2015), *RMSprop* (Tieleman and Hinton, 2012) or *Adagrad* (Duchi et al., 2011).

Training and evaluation

In order to train and evaluate the model, the dataset is typically divided into three subsets: *training*, *validation* and *test*. See Bishop (2006) for different methods to split the dataset.

During the learning process the model iterates over the training data in batches of m samples. Each iteration over all the training set is called an *epoch*. After each epoch, the validation set is presented to the model to provide an indication of the model's performance when processing new data. Therefore, weights are not updated during the validation process. The validation set is also commonly used for hyperparameter tuning.

In general, the learning and validation process is computed for a fixed number of epochs, and the model with the lowest error for the validation set is selected. *Early stopping* is a common procedure in which the training stops if there is no improvement in the validation loss. This, after a predefined number of epochs which is referred as *patience*. Once the model is selected, the final evaluation of the model is carried out by processing the test subset only one time.

Another common procedure that takes place during the learning process is *dropout*, which consists of randomly ignoring some of the layer outputs (Srivastava et al., 2014). *Dropout* helps to prevent the network from overfitting the training set, i.e. when the network closely matches the training data while performing poorly for the validation or test subsets.

3.3 ACTIVATION FUNCTIONS

The final numerical value of each unit is obtained after applying a nonlinear function $f()$ to the weighted sum of inputs. This function is often known as *activation function* due to the loose analogy to biological neurons; where each neuron receives inputs from several other neurons and computes its own activation value Goodfellow et al. (2016). Nevertheless, they are of great importance, since by introducing nonlinearities between layers, the model increases the ability to learn complex functions.

The most common types of activation functions are described below, where x is the weighted input of a unit.

$$\textbf{Rectifier Linear Unit (ReLU)}: \quad f(x) = \max(0, x) \tag{3.4}$$

$$\textbf{Sigmoid}: \ \sigma(x) = \frac{1}{1+e^{-x}} \tag{3.5}$$

$$\textbf{Hyperbolic tangent}: \ \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.6}$$

$$\textbf{Softplus}: \ f_{sp}(x) = \ln(e^x + 1) \tag{3.7}$$

Trainable activation functions have been shown to improve generalization capabilities and accelerate the learning process of neural networks, such as parametric logarithmic, linear, and exponential functions (Godfrey and Gashler, 2015), cubic spline functions (Solazzi and Uncini, 2000; Uncini, 2003) and piecewise polynomial functions (Hou et al., 2016, 2017). In this thesis, we focus on a particular case of piecewise polynomial functions, which is defined as follows.

- **Smooth Adaptive Activation Function (SAAF)**: it consists of piecewise second order polynomials which can approximate any continuous function and are regularized under a Lipschitz constant to ensure smoothness (Hou et al., 2017). A function $f(x)$, defined in $[a, b]$, is Lipschitz continuous if and only if its first derivative is bounded, i.e. there is a constant $C \in \mathbb{R}$ such that $|f'(x)| \leqslant C$, $\forall x \in [a, b]$ (Phillips and Taylor, 1996). Thus, differentiability and bounded smoothness are characteristics of SAAFs.
  Hou et al. (2016) shows that SAAFs can approximate any one-dimensional function given a sufficient number of polynomial segments. Thus, SAAFs can be defined as follows, where n is the number of segments outlined by the breakpoints $a_1, ..., a_{n+1}$, which are real numbers in ascending order.

$$f(x) = w_0 x + \sum_{k=1}^{n} w_k b_k^2(x) \tag{3.8}$$

Where w_0 and w_k are the weights of the activation function and are learned during the training process. $b_k^2(x)$ are the basis functions, which are defined as the double integral of the boxcar function $b_k^0(x)$.

$$b_k^2(x) = \int_0^x \int_0^x b_k^0(\alpha)\partial^2\alpha \tag{3.9}$$

$$b_k^0(x) = \begin{cases} 1, & a_k \leqslant x < a_{k+1} \\ 0, & \text{otherwise} \end{cases} \tag{3.10}$$

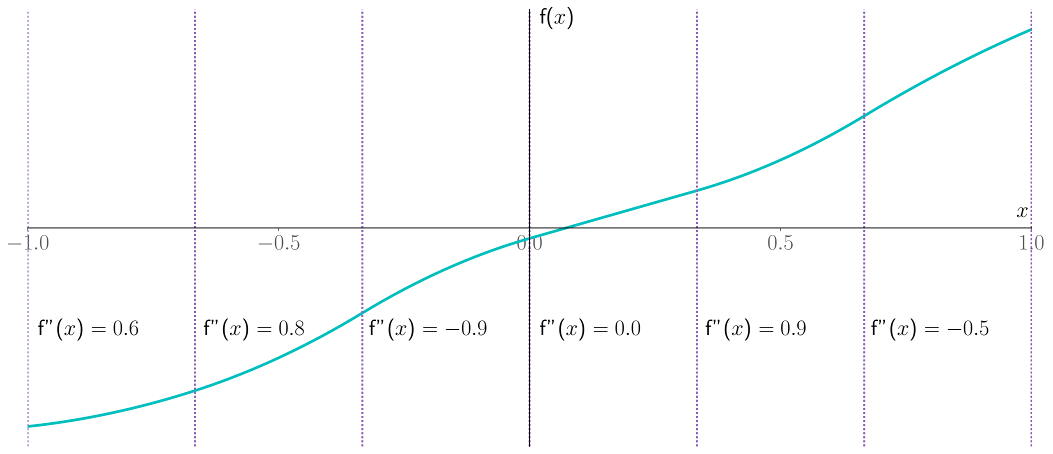


Figure 3.1: SAAF with $n = 6$ and equally spaced breakpoints between -1 and 1. The parameters w_k, or second derivatives, are displayed for each segment and $w_0 = 0.1$. Axes are unitless.

Thus, for $x \in [a_k, a_{k+1}]$, the numerical value of an SAAF is as follows.

$$f(x) = w_0 x + \frac{1}{2} w_k (x - a_k)^2 \tag{3.11}$$

Following this definition SAAFs are guaranteed to be continuous, including at the locations of the $n + 1$ breakpoints. For each segment, there is only one parameter w_k which controls the 2th order derivative within the segment. In addition, the Lipschitz constant C for piecewise second order polynomials is defined in terms of the parameters w_0 and w_k and corresponds to the maximum derivative magnitude of $f(x)$. Therefore the smoothness of an SAAF can be regularized by applying *L2* regularization on its parameters (Hou et al., 2016). Fig. 3.1 shows a example of an SAAF.

$$C = \max \left| w_0 + \int_0^x \sum_{k=1}^{n} w_k b_k^2(\alpha) \partial \alpha \right| \tag{3.12}$$

3.4 DENSE LAYERS

Densely connected layers, also called dense layers, fully connected (FC) networks, deep feedforward networks, or multilayer perceptrons, are the most common type of layers in DNNs (Chollet, 2018). Their name is due to the fact that each neuron is connected to all the outputs of the previous layer, thus, densely connected. Models based on FC layers are considered feedforward networks because there is no feedback or recursive connections within the model. Therefore, information flows directly from the input to the output, via the hidden layers (Goodfellow et al., 2016).

The output feature map $\mathbf{Y}_k$ of the k*th* layer of an FC network is described as

$$\mathbf{Y}_k = f_k(\mathbf{W}_k \cdot \mathbf{X}_k), \tag{3.13}$$

where $f_k()$ is the element-wise activation function and $\mathbf{W}_k$ and $\mathbf{X}_k$ are the matrices corresponding to the weights and input, respectively.

Throughout this thesis, the operator $\cdot$ is defined as follows: If $\mathbf{W}_k$ and $\mathbf{X}_k$ are matrices, the operator $\cdot$ denotes their matrix multiplication; and if $\mathbf{W}_k$ is a matrix and $\mathbf{X}_k$ is a one-dimensional array, the operator $\cdot$ denotes a sum product over the last axis of $\mathbf{W}_k$ and $\mathbf{X}_k$. Also, for the sake of clarity and across this thesis, the bias weights are contained in the corresponding matrix of weights $\mathbf{W}_k$.

Dense layers learn deterministic nonlinear transformations from input to output. Thus, learning a function $f : \mathbb{R}^{D_{in}} \to \mathbb{R}^{D_{out}}$, where D_{in} and D_{out} are the dimensions of the input and output vectors, respectively. As for the design, the number of layers, as well as the number of units per layer determine how this transformation is achieved. For example, when learning compressed representations of the input, i.e. dimensionality reduction, it is common to use a bottleneck architecture where $D_{out} < D_{in}$.

The mapping learned by FC layers is usually very effective when processing simple vector data, i.e. feature vectors without spatial nor temporal structure. Therefore, since dense layers are not shift nor scale invariant, when processing data with these characteristics, these are placed after more specialized layers such as convolutional or recurrent.

3.5 CONVOLUTIONAL LAYERS

Convolutional neural networks (CNN), also called convnets, are a type of feedforward network which is characterized by using the convolution operation (see Eq. (2.4). This differs from dense layers, which are based on the dot product operation or matrix multiplications.

The main advantage of convolutional layers over dense layers is that, when processing their input vector or feature space, the convolution operation allows them to learn local patterns, while dense layers can only learn global patterns. Therefore, convolutional layers are shift invariant, since after learning a specific pattern, CNNs can recognize the pattern at any other location within the input feature space. In addition, convnets can process inputs of variable size and are also more computationally efficient than dense matrix multiplications (Chollet, 2018).

Thus, CNNs are used when the spatial structure of the input is relevant, e.g. audio waveforms or two-dimensional time-frequency representations of audio. The operation performed by a convolutional layer is

$$\mathbf{Y}_k = f_k(\mathbf{X}_k * \mathbf{W}_k), \tag{3.14}$$

where $\mathbf{Y}_k$ and $\mathbf{X}_k$ are the matrices corresponding to the output and input feature maps, respectively. $\mathbf{W}_k$ is the matrix of convolutional kernels or filters and $f_k()$ is the element-wise activation function.

Eq. (3.14) illustrates the general operation performed by convnets, which also assumes convolutions over more than one axis at a time. i.e. multidimensional convolutions, where the kernel and input feature map can be one, two or three-dimensional. In this thesis we focus on one-dimensional convolutions, also called temporal convolutions, as we investigate end-to-end architectures that directly process raw audio. Thus, across this thesis, $\mathbf{X}_k$, $\mathbf{W}_k$ and $\mathbf{Y}_k$ generally represent matrices of multichannel one-dimensional audio representations, e.g. multichannel audio waveforms, filter banks or envelopes, and the convolution operation is defined over the rows of these matrices.

The following are the various hyperparameters when designing CNNs.

- **Number of filters**: corresponds to the total of convolutional kernels or rows in the matrix $\mathbf{W}_k$ and also establishes the dimensionality of the output feature map $\mathbf{X}_k$.
- **Kernel size**: consists of the length of each filter, thus it represents the number of columns in the matrix $\mathbf{W}_k$.
- **Padding**: sets the way the convolutional layer processes the border values of $\mathbf{X}_k$. It is common to pad with zeros on each sides of the input feature map, at the beginning and at the end, so the sliding convolutional kernel can process each numerical value of the input. When padding is not performed, the length of the output feature map is reduced. Also, *causal* padding consists in inserting zeros only at the beginning of the input.
- **Stride**: corresponds to the hop size of the sliding convolution window, i.e. the number of positions that the filters move when processing the next input value. When the stride is larger than 1, the length of $\mathbf{Y}_k$ is less than the length of $\mathbf{X}_k$.

- **Dilation**: consists of increasing the receptive field of each filter by matching the kernel values to non-subsequent elements of the input, thus, introducing gaps in the input feature map. When dilation is larger than 1, the length of the input feature map, or resolution, is preserved and the field of view of the kernels is increased.

Furthermore, It is common practice to use *subsampling* or *pooling* layers when stacking various convolutional layers. Subsampling functions, such as *max-pooling* or *average-pooling* layers, consist of moving windows of size n, where the maximum or average value within each window corresponds to the output feature map. When n is equal to the length of the input, these layers are called *global-max-pooling* and *golbal-average-pooling*, respectively.

The operations performed by a convolutional layer can be divided into three steps; convolution between the input feature map and the filters; the nonlinear function, which is applied element-wisely to each resulting convolution and produces a set of nonlinear activations; and the pooling operation, which reduces the dimensions of the output feature map and also induces this representation to be translation invariant (Goodfellow et al., 2016). It is worth noting that the input feature map for CNN-based architectures that process raw audio, such as those presented in this thesis, corresponds to a sliding window across the audio waveform. Commonly, the input audio frame is multiplied by a window function, such as rectangular or *hann* functions and with or without overlap, depending on the application.

3.6 RECURRENT LAYERS

Recurrent neural networks (RNN) are extended feedforward networks by means of feedback connections. Dense and convolutional layers are characterized by learning deterministic mappings and having no memory. Therefore, each input feature map is always processed independently and always generates the same output. When feedback connections are incorporated into the layers, the resulting networks learn to maintain states between inputs. This allows RNNs to learn from the temporal structure of the input, which is essential when processing data with long temporal dependencies (Chollet, 2018).

Vanilla RNN

The most simple type of RNN, often called *vanilla RNN*, is characterized by having recurrent connections between their hidden units and for generating an output at each time step. These networks are described as

$$\mathbf{y}^t = f_{out}(\mathbf{W}_{out} \cdot \mathbf{h}^t), \tag{3.15}$$

where

$$\mathbf{h}^t = f_h(\mathbf{W}_{in} \cdot \mathbf{x}^t + \mathbf{W}_h \cdot \mathbf{h}^{t-1}), \tag{3.16}$$

where $\mathbf{x}^t$, $\mathbf{h}^t$ and $\mathbf{y}^t$ are the input, hidden state and output matrices at the current time step t, respectively. Likewise, $f_{out}()$ and $f_h()$ are the activation functions for the output and hidden layers. The matrices $\mathbf{W}$ represent the weight of the recurrent layer, where $\mathbf{W}_{in}$ is the weight matrix for the input-to-hidden connections, and correspondingly $\mathbf{W}_h$ for the hidden-to-hidden connections and $\mathbf{W}_{out}$ for the hidden-to-output connections (Chollet, 2018).

It is common to reset the state of the hidden units when processing independent sequences or batches. Thus, the RNN learns to map the input sequence $\mathbf{x}^t, ..., \mathbf{x}^0$ to a hidden state $\mathbf{h}^t$, which contains relevant features of the input up to the current time step t. Furthermore, the network learns how to update $\mathbf{h}^t$ based on the previous hidden state $\mathbf{h}^{t-1}$ and, finally, the network learns how to use $\mathbf{h}^t$ in order to obtain the output feature map $\mathbf{y}^t$.

Bidirectional RNN

Schuster and Paliwal (1997) proposed bidirectional recurrent layers in order to improve the long-term memory capabilities of RNNs. A *bidirectional RNN* consists of two RNNs, one that moves from the start of the sequence, while the other moves from the end of the sequence. Thus, bidirectional recurrent layers can access long-term context from both backward and forward directions, which improves the capabilities of learning long temporal dependencies when processing sequences where the context of the input is needed (Graves and Schmidhuber, 2005; Graves et al., 2013).

The operation of a bidirectional RNN is described in the following equations:

$$\mathbf{y}^t = f_{out}(\mathbf{W}_{out} \cdot \mathbf{h}^t + \mathbf{V}_{out} \cdot \mathbf{g}^t), \tag{3.17}$$

$$\mathbf{g}^t = f_g(\mathbf{V}_{in} \cdot \mathbf{x}^t + \mathbf{V}_g \cdot \mathbf{g}^{t+1}), \tag{3.18}$$

where $\mathbf{V}_{in}$, $\mathbf{V}_g$ and $\mathbf{V}_{out}$ are the input-to-hidden, hidden-to-hidden and hidden-to-output weight matrices of the RNN moving backwards in time. $\mathbf{g}^t$ and $f_g()$ corresponds to the hidden state and activation function, respectively, and $\mathbf{h}^t$ is defined in Eq. (3.16).

Long short-term memory (LSTM)

In general, the main drawbacks of RNNs are related to the propagation of the gradient, such as *vanishing gradient* or *exploding gradient*. As the gradient propagates across many states, this could lead to a severe increase or decrease of the gradient. Vanishing gradient refers to a gradient that exponentially decreases across the layers, which prevents the weights of the earlier layers from being updated, while an exploding gradient occurs when the gradient grows exponentially, so that the network becomes unstable (Pascanu et al., 2013).

Various types of RNNs have been explored in oder to overcome vanishing or exploding gradients, such as *long short-term memory* networks (Hochreiter and Schmidhuber, 1997). LSTM networks correspond to gated RNNs where the neural network learns when to reset or regulate the flow of information within the recurrent layer. This is achieved via gates implemented with the sigmoid function $\sigma()$, which are called *input gate*, *output gate* and *forget gate* (Gers et al., 1999). LSTMs incorporate an internal hidden state, often called cell or memory $\mathbf{c}^t$, which is able to keep states over arbitrary time intervals and is regulated by the three gates.

The gates within an LSTM network are described as follows:

$$\mathbf{g}^t_{in} = \sigma(\mathbf{W}_{in} \cdot \mathbf{x}^t + \mathbf{W}_h \cdot \mathbf{h}^{t-1}), \tag{3.19}$$

$$\mathbf{g}^t_{out} = \sigma(\mathbf{U}_{in} \cdot \mathbf{x}^t + \mathbf{U}_h \cdot \mathbf{h}^{t-1}), \tag{3.20}$$

$$\mathbf{g}^t_{forget} = \sigma(\mathbf{V}_{in} \cdot \mathbf{x}^t + \mathbf{V}_h \cdot \mathbf{h}^{t-1}), \tag{3.21}$$

where $\mathbf{g}^t_{in}$, $\mathbf{g}^t_{out}$ and $\mathbf{g}^t_{forget}$ are the matrices corresponding to the input, output and forget gates, respectively. $\mathbf{W}_{in}$, $\mathbf{U}_{in}$ and $\mathbf{V}_{in}$ represent their input weight matrices and $\mathbf{W}_h$, $\mathbf{U}_h$, and $\mathbf{V}_h$ their recurrent weight matrices. $\mathbf{h}^{t-1}$ is the output feature map of the LSTM at the previous time step $t-1$.

The LSTM cell state is updated with

$$\mathbf{c}^t = \mathbf{g}^t_{\text{forget}} \times \mathbf{c}^{t-1} + \mathbf{g}^t_{\text{in}} \times \tanh(\mathbf{C}_{\text{in}} \cdot \mathbf{x}^t + \mathbf{C}_{\text{h}} \cdot \mathbf{h}^{t-1}), \tag{3.22}$$

where $\times$ denotes element-wise multiplication. $\mathbf{C}_{\text{in}}$ and $\mathbf{C}_{\text{h}}$ are the input and recurrent weight matrices of the cell, respectively. tanh is used as the element-wise gating function of the cell unit to allow negative values for $\mathbf{c}^t$. Finally, the output $\mathbf{h}^t$ of the LSTM is computed as

$$\mathbf{h}^t = \tanh(\mathbf{c}^t) \times \mathbf{g}^t_{\text{out}}. \tag{3.23}$$

LSTMs have been shown to learn long-term dependencies more efficiently than other types of RNNs. This is due to their ability to maintain states without vanishing during the processing of the input sequences (Chollet, 2018). In addition, variant implementations of LSTMs have been explored, such as gated recurrent units (GRUs) (Cho et al., 2014).

3.7 SQUEEZE-AND-EXCITATION LAYERS - SE

Squeeze-and-Excitation networks (Hu et al., 2018) explicitly model interdependencies between channels, i.e. the rows of input matrices, by adaptively scaling the channel-wise information of feature maps. The SE blocks have been shown to provide significant improvements to the representational power of DNNs and to generalize effectively across different tasks and datasets (Hu et al., 2018).

The SE layers correspond to a *global-average-pooling* (GAP) operation followed by two dense layers and a sigmoid function. The pooling layer *squeezes* the global spatial information of the input feature map into a descriptor per channel or filter, whereas the dense layers act as an adaptive gating mechanism via the σ activation function. The latter denotes an *excitation* operation, where the channel descriptors are mapped to a set of channel weights se. Subsequently, the channel weights se are used to scale the input feature map in order to perform feature recalibration.

A block diagram can be seen in Fig. 3.2 and its function is described in detail in the following equations (Hu et al., 2018):

$$se = \sigma(\mathbf{W}_2 \cdot \text{ReLU}(\mathbf{W}_1 \cdot \text{GAP}(\mathbf{X}_1))), \tag{3.24}$$

$$\mathbf{X}_2 = \mathbf{X}_1 \times se, \tag{3.25}$$

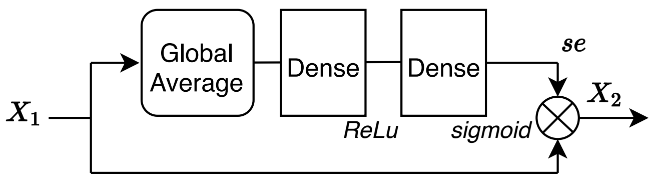


Figure 3.2: Block diagram of SE layers.

where $\mathbf{X}_1$ and $\mathbf{X}_2$ are the matrices corresponding to the input and output feature maps and $\mathbf{W}_1$ and $\mathbf{W}_2$ are the weight matrices of the first and second dense layers. The dense layers are followed by *ReLU* and σ activation functions, respectively. The output of the GAP operation corresponds to a vector of size equal to the number of channels in $\mathbf{X}_1$ and, consequently, the resulting se is also a vector whose size is the number of channels of the input.

3.8 DEEP LEARNING FOR MUSIC

In recent years, deep learning for music has been a constantly growing field, where a large percentage of current research has been devoted to extract information and understand its content. Most of the examples are in the fields of music information retrieval (Sigtia and Dixon, 2014; Sigtia et al., 2016), music recommendation (Van den Oord et al., 2013; Wang and Wang, 2014), and audio event recognition (Lee et al., 2009; Stowell and Plumbley, 2014).

For example, dense layers were initially used as feature estimators for various music information retrieval tasks, such as audio classification (Sigtia and Dixon, 2014; Hamel et al., 2013), downbeat tracking (Durand et al., 2015) or chord estimation (Korzeniowski and Widmer, 2016; Deng and Kwok, 2016). Convolutional layers have also been extensively researched for similar tasks, such as chord estimation (Humphrey and Bello, 2012, 2014), music transcription (Sigtia et al., 2016), downbeat tracking (Schlüter and Böck, 2014), onset detection (Schlüter and Böck, 2013) and audio classification (Han et al., 2016). Similarly, recurrent layers have been investigated for chord and melody estimation (Eck and Schmidhuber, 2002), note prediction (Huang and Wu, 2016) and music transcription (Sigtia et al., 2015; Sturm et al., 2016).

Most deep learning applications for music are based on frequency representations of audio such STFT, or Mel-spectogram frames. Nevertheless, end-to-end networks where the audio waveform is the input have also experienced significant growth. Automatic audio tagging tasks have been explored within this framework (Dieleman and Schrauwen, 2014; Pons et al., 2017). The networks autonomously learn features related to the frequency and phase of the raw waveforms, although architectures based on spectrograms still yielded better results. In (Sigtia et al., 2016) an end-to-end neural network is investigated for the transcription of polyphonic piano music. In the context of end-to-end supervised source separation, Venkataramani et al. (2017) proposed an adaptive convolutional architecture capable of learning a latent representation from the raw waveform.

Furthermore, end-to-end deep learning applied to the generation of music has also become a growing field with emerging architectures such as *wavenet* (Oord et al., 2016), which is an autoregressive network of stacked temporal convolutions and characterized by dilation factors that increase exponentially. Engel et al. (2017) proposed *NSynth*, a *wavenet* architecture that generates audio sample by sample and allows instrument morphing from a dataset of short musical notes. This was achieved using an end-to-end architecture, where raw audio is both the input and the output of the system. Similarly, Mehri et al. (2017) obtained raw audio generation without the need of handcrafted features and Blaauw and Bonada (2017) accomplished singing voice synthesis.

3.9 DEEP LEARNING FOR AUDIO EFFECTS MODELING

Deep learning architectures for audio processing tasks, such as audio effects modeling, have been investigated as end-to-end methods or as parameter estimators of audio processors. Rämö and Välimäki (2019) predicted the filter gains of an EQ using a multilayer perceptron and Sheng and Fazekas (2019) explored CNNs and FC layers to estimate multiple parameters of a dynamic range processor when matching a compressed audio target. Also, Mimilakis et al. (2016) investigated dense layers to predict gain coefficients in order to perform automatic dynamic range compression for mastering applications.

The aforementioned methods differ from end-to-end deep learning architectures, where raw audio is both the input and the output of the DNN. End-to-end deep learning is based on the idea that an entire problem can be taken as a single indivisible task which must be learned from input to output.

Following this approach, Damskägg et al. (2019) explored variants of the *wavenet* architecture in order to model nonlinear effects such as a tube amplifier. Thus, based on a *wavenet* for speech denoising (Rethage et al., 2018), a feedforward architecture is proposed in (Damskägg et al., 2019), where a nonlinear audio effect and its controls are emulated. This network outperforms current state-of-the-art analytical methods for nonlinear black-box modeling such as the block-oriented Wiener models presented in (Eichas and Zölzer, 2016).

In (Hawley et al., 2019), black-box modeling is proposed for nonlinear effects with long temporal dependencies such as compressors. The architecture is based on the *U-Net* (Ronneberger et al., 2015) and *Time-Frequency* (Lim et al., 2018) networks, where input-output measurements and knowledge of the attack and release gate times are used to emulate different compressors and their respective controls. Similarly, various RNNs for real-time black-box modeling of tube amplifiers and distortion pedals were explored in (Wright et al., 2019), and LSTM networks were investigated to learn static configurations of tube amplifiers in (Schmitz and Embrechts, 2018; Zhang et al., 2018). Also, a gray-box method is explored in (Parker and Esqueda, 2019), where a DNN is used to model the state-space system of nonlinear distortion circuits.

Table 2.1 outlines the different deep learning approaches for audio effects modeling. Most of the active research has focused on modeling nonlinear audio processors without long-term memory, so it omits a large percentage of the other types of audio effects. In addition, the methods based on parameter estimation are based on fixed audio processing architectures, resulting in a lack of generalization.

3.10 CONCLUSION

Therefore, in the following chapters, we explore general-purpose end-to-end deep learning architectures for all types of audio effects. Equalization matching is achieved in Chapter 4 and Publication **I**. Nonlinear modeling is explored in Chapter 5 and Publication **II**, where different models are capable of modeling an arbitrary combination of linear and nonlinear audio effects with short-term memory. The latter architectures do not generalize to transformations with long temporal dependencies such as modulation based audio effects. Thus, several linear and nonlinear time-varying and time-invariant audio effects were modeled in Chapter 6 and Publication **III**. Virtual analog experiments for the previous architectures are conducted in Chapter 7 and Publication **V**. Finally, artificial reverberators are modeled in Chapter 8 and Publication **V**.

4 MODELING EQ EFFECTS

In this chapter we introduce a novel deep learning architecture to model linear effects in the context of matched EQ. As discussed in Section 2.2, matched equalization corresponds to methods that match a target frequency response by optimizing the filter coefficients of specific types of filters. Nevertheless, most of these methods rely on fixed architectures of filter banks or require prior knowledge of the type of filters to be modeled.

Thus, by using an end-to-end approach, we implement a deep learning architecture to perform black-box modeling of linear audio effects, such as EQ matching. The model approximates the EQ target as a content-based transformation without directly finding the transfer function, i.e. without explicitly obtaining the parameters of the filters: G, f_c and Q (see Section 2.1).

We show that a procedure based on convolutional and dense layers, via time-domain convolutions and latent-space modifications, can lead us to perform EQ matching. Therefore, we investigate EQ as a time-domain convolution transformation, where the inherent content of the input and filtered signals can lead a CNN to match a target frequency response.

Given an arbitrary EQ configuration or target, our regression task is to train a DNN to learn the specific transformation. In this way, a filter bank decomposition and its latent representation are learned from the input data, and these are transformed respectively to obtain an audio signal that matches the target. The network learns how to process the audio directly in order to match the equalized target audio.

Accordingly, we explore whether the model can be used for EQ matching using an end-to-end architecture, where raw audio is both the input and the output of the system. We analyze what the model is actually learning and how the given task is accomplished, and we use a relevant loss function in the time and frequency domains in order to achieve the equalizer task. We show the model performing matched equalization for *shelving*, *peaking*, *lowpass* and *highpass* IIR and FIR equalizers. Audio samples can be found in Appendix B.

4.1 CONVOLUTIONAL EQ MODELING NETWORK - CEQ

In order to design the network, we follow a similar procedure as in Venkataramani et al. (2017), although based entirely on the time-domain. The model can be divided into three parts: adaptive front-end, synthesis back-end and latent-space DNN. The model is depicted in Fig. 4.1 and its structure is described in detail in Table 4.1. To the best of our knowledge, this model represents the first deep learning architecture for black-box modeling of EQ effects. Appendix C shows the number of parameters and processing times of the model. Code is availabe online[1]. We use an input frame of size 1024, sampled with a hop size of 256 samples and windowed by a *hann* function. All convolutions are along the time dimension and all strides are of unit value. This means, during convolution, we move the filters one sample at a time. In addition, padding is done on each side of the input feature maps so that the output maintains the resolution of the input. Dilation is not introduced.

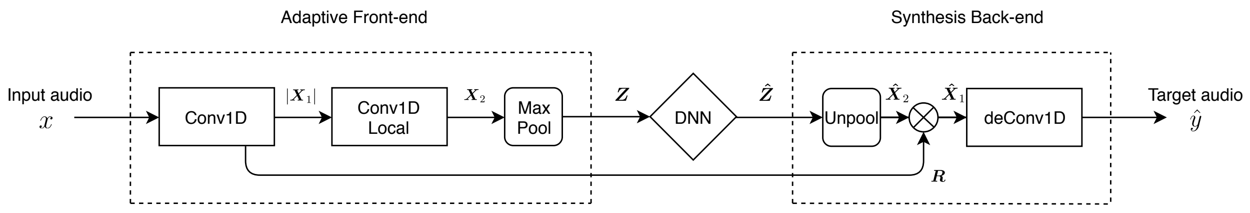


Figure 4.1: Block diagram of *CEQ*; adaptive front-end, synthesis back-end and latent-space DNN based on LC and FC layers.

For a specific EQ configuration or arbitrary combination of filters, consider x and y the raw and equalized audio signals respectively. We train a CNN which operates as a filter bank and produces a latent representation $\mathbf{Z}$ of the given EQ matching task. The latent representation corresponds to compressed or subsampled audio waveforms or learned audio features relevant to the regression task. From Section 3.5, a single one-dimensional convolutional layer can be described as

$$\mathbf{X}_k = \sum_{i=0}^{N-1} \mathbf{X}_{k-1}(n-i) \cdot \mathbf{W}_k(i), \tag{4.1}$$

where $\mathbf{X}_k$ is the output feature map of the k*th* layer. N represents the size of the input feature map $\mathbf{X}_{k-1}$ or input frame x in the case of the first layer. Throughout

1 https://mchijmma.github.io/end-to-end-equalization/

Table 4.1: Detailed architecture of *CEQ* with an input frame size of 1024 samples. Output shape = (m,n) denotes m columns and n rows. Weights = $m(n)$ denotes m kernels of size n, and Weights = n denotes n hidden units.

Layer	Output shape	Weights	Output
Input	(1024, 1)	.	x
Conv1D	(1024, 128)	128(64)	$\mathbf{X}_1$
Residual	(1024, 128)	.	$\mathbf{R}$
Abs	(1024, 128)	.	.
Conv1D-Local	(1024, 128)	128(128)	$\mathbf{X}_2$
MaxPooling	(64, 128)	.	$\mathbf{Z}$
Dense-Local	(64, 128)	64	$\hat{\mathbf{Z}}_h$
Dense	(64, 128)	64	$\hat{\mathbf{Z}}$
Unpooling	(1024, 128)	.	$\hat{\mathbf{X}}_2$
$\mathbf{R} \times \hat{\mathbf{X}}_2$	(1024, 128)	.	$\hat{\mathbf{X}}_1$
deConv1D	(1024, 1)	.	$\hat{y}$

this thesis, these feature maps represent matrices of multichannel one-dimensional audio representations, i.e. each row is an audio waveform or envelope. $\mathbf{W}_k$ is the kernel matrix with M_k filters where each filter correspond to a row in the matrix. The convolution operation is computed between the rows of the input and kernel matrices.

The latent representation $\mathbf{Z}$ is obtained after the designated number of convolutional and subsampling layers of the adaptive front-end and corresponds to a compressed or learned representation

Thus, in order to obtain a $\hat{y}$ that matches the EQ target y, we implement a latent-space DNN to modify $\mathbf{Z}$ based on the EQ matching regression task. Finally, the synthesis back-end implements the deconvolution operation and reconstructs the time-domain signal by inverting the operations of the encoder. We train the whole network within an end-to-end learning framework and we minimize a suitable metric between the target and the output of the network.

Based on an EQ matching task, we expect the network to learn the relevant filters $\mathbf{W}_k$, latent representation $\mathbf{Z}$ and further manipulation. We attempt to find a general architecture that can serve as a matching equalizer based on an arbitrary time-invariant EQ target.

The general structure of *CEQ* corresponds to the main or base architecture of the subsequent models proposed throughout this thesis. Overall, the adaptive front-end learns a filter bank decomposition and its correspondingly latent representation. The front-end also generates a frequency band decomposition of the input audio as a residual connection. The latent representation is modified by the latent-space DNN and fed to the synthesis back-end along with the residual connection. Finally, the back-end transforms the residual connection via the modified latent representation and thus synthesizes a waveform based on the specific audio effects modeling task.

Adaptive front-end

The adaptive front-end consists of a convolutional encoder. It contains two convolutional layers, one pooling layer and one residual connection. The front-end is considered adaptive since its convolutional layers learn a filter bank for each modeling task and directly from the audio.

The first convolutional layer is followed by the *absolute value* as nonlinear activation function and the second convolutional layer is locally connected (LC). This means we follow a filter bank architecture since each filter is only applied to its corresponding row in the input feature map. The latter layer is followed by the *softplus* nonlinearity. The *max-pooling* layer is a moving window of size 16, where the maximum value within each window corresponds to the output and the positions of the maximum values are stored and used by the back-end. From Eq. (4.1), the operation performed by the first layer can be described as

$$\mathbf{X}_1 = x * \boldsymbol{W}_1, \tag{4.2}$$

$$\mathbf{R} = \mathbf{X}_1, \tag{4.3}$$

where $\boldsymbol{W}_1$ is the kernel matrix from the first layer, and $\mathbf{X}_1$ is the feature map after the input audio x is convolved with the rows of $\boldsymbol{W}_1$. The weights $\boldsymbol{W}_1$ consist of 128 one-dimensional filters of size 64, i.e. a matrix of 64 columns and 128 rows. The residual connection $\mathbf{R}$ is equal to $\mathbf{X}_1$, which corresponds to a matrix of 128 channels of 1024 samples, i.e. a matrix of 1024 columns and 128 rows. $\mathbf{R}$ and $\mathbf{X}_1$ correspond to the frequency band decomposition of the input x. This, since *Conv1D* can be seen as a filter bank.

The operation performed by the second layer is described as

$$\mathbf{X}_2^{(i)} = f_{sp}(|\mathbf{X}_1^{(i)}| * \boldsymbol{W}_2^{(i)}), \ \forall i \in [1, 128], \tag{4.4}$$

where $\mathbf{X}_2^{(i)}$ and $\boldsymbol{W}_2^{(i)}$ are the i*th* row of the feature map $\mathbf{X}_2$ and kernel matrix $\boldsymbol{W}_2$, respectively. Thus, $\mathbf{X}_2$ is obtained after the LC convolution between $\mathbf{X}_1$ and $\boldsymbol{W}_2$, the weight matrix of *Conv1D-local*. Each row or channel of the input feature map is convolved only with its respective filter in $\boldsymbol{W}_2$, which consist of 128 one-dimensional filters of size 128, i.e. a matrix of 128 columns and 128 rows. $\mathbf{X}_2$ corresponds to a matrix of 128 channels of 1024 samples. $f_{sp}()$ is the *softplus* function.

The adaptive front-end performs time-domain convolutions with the raw audio and is designed to learn a latent representation for each audio effect modeling task. It also generates a residual connection which is used by the back-end to facilitate the synthesis of the waveform based on the specific audio effect transformation.

This differs from traditional encoding practices, where the complete input data is encoded into a latent-space, which causes each layer in the decoder to solely generate the complete desired output (He et al., 2016). Furthermore, a full encoding approach such as Engel et al. (2017); Oord et al. (2016) will require very deep models, large data sets and difficult training procedures.

By using the *absolute value* as activation function of the first layer and by having larger filters $\boldsymbol{W}_2$, we expect the front-end to learn smoother representations of the incoming audio, such as envelopes as shown in Venkataramani et al. (2017).

Latent-space DNN

The latent-space DNN contains two dense layers. Following the filter bank architecture, the first layer is based on LC dense layers and the second layer consists of an FC layer. The DNN modifies the latent representation $\mathbf{Z}$ into a new latent representation $\hat{\mathbf{Z}}$ which is fed into the synthesis back-end. The first layer applies a different dense layer to each row of the matrix $\mathbf{Z}$ and the second layer is applied to each row of the output matrix from the first layer. In both layers, all dense layers have 64 hidden units and are followed by the *softplus* function (f_{sp}).

The operation performed by the latent-space DNN is

$$\hat{\mathbf{Z}}_h^{(i)} = f_{sp}(\mathbf{V}_1^{(i)} \cdot \mathbf{Z}^{(i)}), \ \forall i \in [1, 64], \tag{4.5}$$

$$\hat{\mathbf{Z}} = f_{sp}(\mathbf{V}_2 \cdot \hat{\mathbf{Z}}_h), \tag{4.6}$$

where $\hat{\mathbf{Z}}_h^{(i)}$ is the i*th* row of the output feature map $\hat{\mathbf{Z}}_h$ of the LC layers. Likewise, $\mathbf{V}_1^{(i)}$ is the i*th* dense layer corresponding to the weight matrix $\mathbf{V}_1^{(i)}$ of the LC layer. $\mathbf{V}_2$ corresponds to the weights of the FC layer. $\mathbf{Z}$, $\hat{\mathbf{Z}}_h$ and $\hat{\mathbf{Z}}$ correspond to matrices of 128 channels of 64 samples. In both layers, the weights $\mathbf{V}_1$ and $\mathbf{V}_2$ are applied to the complete latent representation rather than to the channel dimension of $\mathbf{Z}$ and $\hat{\mathbf{Z}}_h$, respectively, i.e. the matrix multiplication is computed between the weights and the rows of the input feature maps which correspond to 128 learned envelopes of size 64 samples.

The output of the max pooling operation $\mathbf{Z}$ corresponds to a matrix that represents the latent representation of the input audio given the EQ task, such as envelopes. The DNN is trained to modify these envelopes, thus, a new latent representation or set of envelopes $\hat{\mathbf{Z}}$ is fed into the synthesis back-end in order to reconstruct an audio signal that matches the target task.

Synthesis back-end

In order to invert the operations performed by the front-end, the decoder consists of one CNN layer and one unpooling layer. Since the *max-pooling* function is non-invertible, the inverse can be approximated by recording the locations of the maximum values in each pooling window (Zeiler and Fergus, 2014) and only upsampling $\mathbf{Z}$ at these time indices. Thus the discrete approximation $\hat{\mathbf{X}}_2$ is obtained.

The approximation $\hat{\mathbf{X}}_1$ of matrix $\mathbf{X}_1$ is obtained through the element-wise multiplication of the residual $\mathbf{R}$ and $\hat{\mathbf{X}}_2$:

$$\hat{\mathbf{X}}_1 = \mathbf{R} \times \hat{\mathbf{X}}_2. \tag{4.7}$$

Depending on whether $\mathbf{Z}$ has been modified or not, Eq. (4.7) can be seen as a sampling or transformation of $\mathbf{X}_1$.

The final layer *deConv1D* corresponds to the deconvolution operation, which can be implemented by transposing the first layer transform. This layer is not trainable since its kernels are transposed versions of $\boldsymbol{W}_1$. In this way, the synthesis layer reconstructs the audio signal in the same manner the front-end decomposed it. This layer does not apply an activation function, thus is considered linear and is depicted as

$$\hat{y} = \hat{\mathbf{X}}_1 * \boldsymbol{W}_1^T, \tag{4.8}$$

where $\hat{y}$ corresponds to the output frame of 1024 samples.

Loss function

The loss function to be minimized is based in time and frequency and described as follows:

$$\text{loss} = \alpha_1 KL(Y, \hat{Y}) + \alpha_2 MSE(Y, \hat{Y}) + \alpha_1 MAE(y, \hat{y}), \tag{4.9}$$

where *KL* is the normalized Kullback-Leibler divergence, the mean squared error is *MSE*, and *MAE* is the mean absolute error (see Section 3.2). Y and $\hat{Y}$ are the frequency magnitude of the target and output respectively, and y and $\hat{y}$ their respective waveforms. We use a 1024-point FFT in order to obtain Y and $\hat{Y}$. The *KL* divergence is computed after normalizing Y and $\hat{Y}$ as probability distributions.

In order to scale the time and frequency losses, we empirically set $\alpha_1 = 1.0$, $\alpha_2 = 1.0$ and $\alpha_3 = 0.1$. We selected a more specialized loss function since by introducing spectral terms in a frequency related task, such as EQ, fewer training iterations were required. Since small amplitude errors are as important as large ones, the loss function to be minimized in the time-domain is the mean absolute error between the target and output waveforms.

4.2 EXPERIMENTS

4.2.1 *Training*

The training of the model includes an initialization step. This pretraining stage consists in optimizing a network formed solely by the convolutional and pooling layers of the front-end and back-end. This pretraining allows to have a better fitting when training for the EQ matching tasks. Within an unsupervised learning task, the network is trained to process and reconstruct both the dry audio x and target audio y. Only during this step the unpooling layer of the back-end uses the time positions of the maximum values recorded by the *max-pooling* operation.

During the pretraining only the weights $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are optimized. This means the model is being prepared to reconstruct the input and target data in order to have a better fitting when training for the EQ task.

Once the front-end and back-end are pretrained, the latent-space DNN is incorporated in the model. The second step consists of an end-to-end supervised learning task based on a given EQ target. Hence, the second training procedure consists in using as objectives of the model x and y as input and target respectively. During the end-to-end learning, all the weights of the convolutional and dense layers are updated. This is done independently for each EQ task.

In both training procedures the input and target audio is windowed by a *hann* function. The batch size consisted of the total number of frames per audio sample and 100 epochs were carried out in each training step. *Adam* (Kingma and Ba, 2015) is used as optimizer and the initial learning rate is $1e-4$.

4.2.2 *Dataset*

The raw audio x is obtained from the *Salamander Grand Piano V3* dataset[2], which consists of a *Yamaha C5* grand piano sampled in minor thirds from the *A0* note and with 16 velocities for each note. The dataset is augmented by pitch shifting each note by $+1$ and $+2$ semitones until all the 88 notes of the piano are obtained. This gives us a total of 1408 samples. The piano notes are downsampled to 16 kHz and trimmed to 4 seconds. The test and validation subsets correspond to 4.55% of the dataset each, and only contain the *B* notes from *B0* to *B7*.

The EQ targets y are obtained by applying the filters described in Table 4.2.

Table 4.2: Filter parameters of the EQ targets.

EQ	filter type	order	gain (dB)	f_c (Hz)	Q
shelving	IIR	2	10	500	0.707
peaking	IIR	2	10	500	0.707
lowpass	FIR	50	0	500	.
highpass	FIR	50	0	500	.

4.3 RESULTS

The training steps were performed for each type of EQ target. Then, the models were tested with samples from the test dataset. Audio is available online[3].

Fig. 4.2 shows various visualizations from the front-end and back-end of *CEQ* after the pretraining step. Fig. 4.2a displays the waveform and frequency magnitude

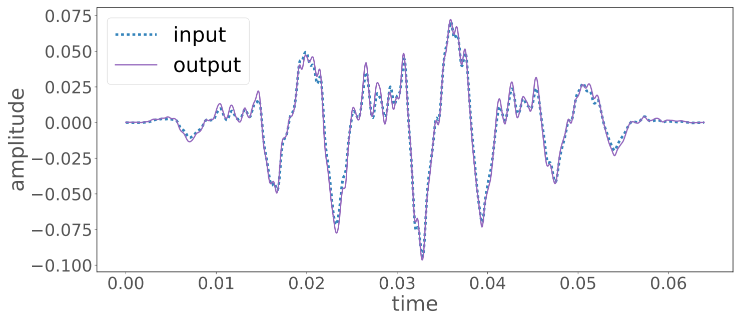
2 https://archive.org/details/SalamanderGrandPianoV3, accessed on 02/03/2020
3 https://mchijmma.github.io/end-to-end-equalization/

of a test frame x of 1024 samples and its respective reconstruction $\hat{x}$. The weights $\mathbf{W}_1$ of *Conv1D* can be seen in Fig. 4.2c, where the first 32 filters are shown.

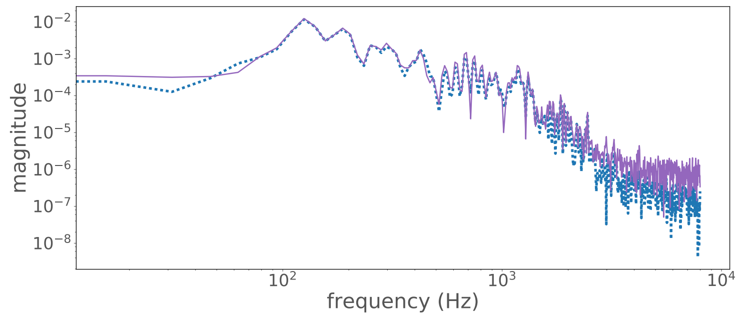
Also, in order to obtain $\hat{x}$, different plots from the front-end, latent-space and back-end are shown in Figs. 4.2d to 4.2f. The results of Eq. (4.2) can be seen in Fig. 4.2d where the first 32 rows of $\mathbf{X}_1$ are displayed. Fig. 4.2e presents their latent-space representation $\mathbf{Z}$, which is obtained through the second convolutional and subsampling layers. Fig. 4.2f shows $\hat{\mathbf{X}}_1$, which is the result of Eq. (4.7). This is the input to the deconvolution layer prior to obtaining the output frame $\hat{x}$.

Following the pretraining of the network, the model is trained through an end-to-end supervised learning method. For each EQ task, Figs. 4.3 and 4.4 show the results of selected samples from the test dataset. For a specific frame of 1024 samples, the input, target and output waveforms as well as their FFT magnitudes are displayed. The power spectrogram of the respective 4-second samples is also shown. Finally, together with the input and the target, the complete reconstructed output waveform of a *shelving* EQ task is presented in Fig. 4.5.

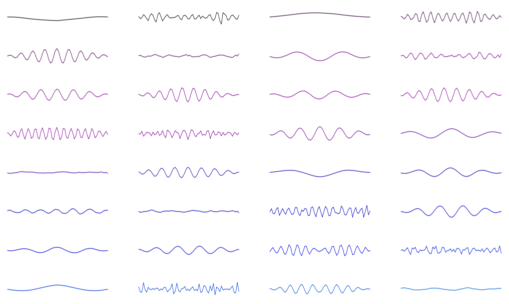
The performance of the models, and their respective losses in time and frequency can be seen in Table 4.3. This based on Eq. (4.9).



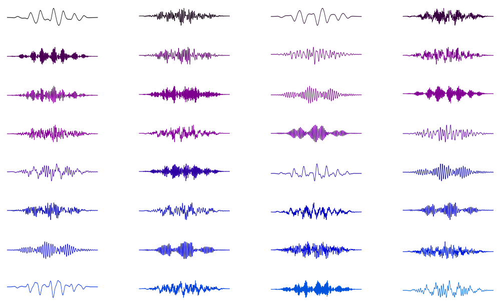
(a)



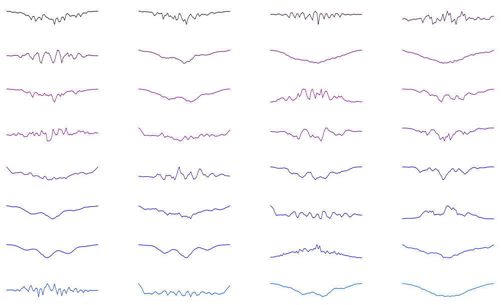
(b)



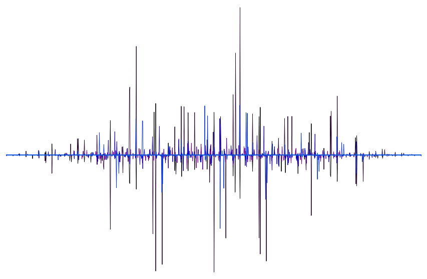
(c)



(d)

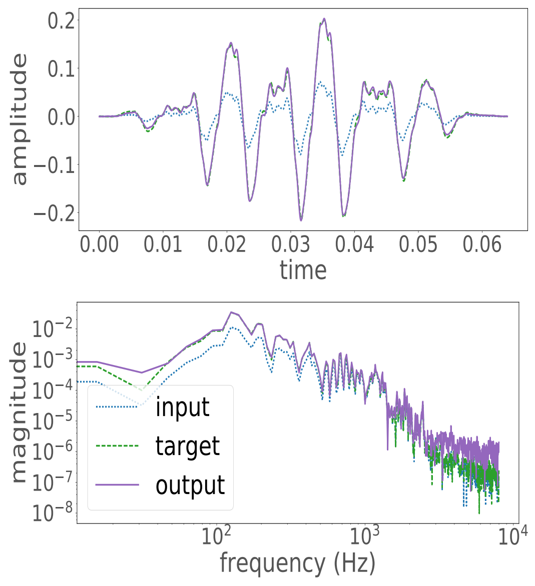


(e)

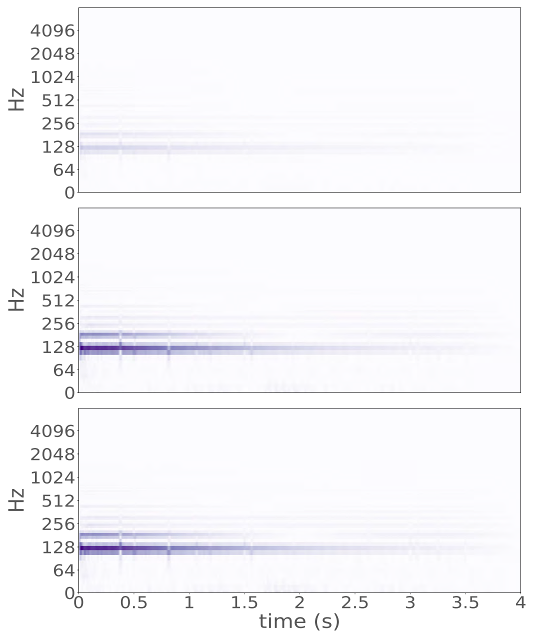


(f)

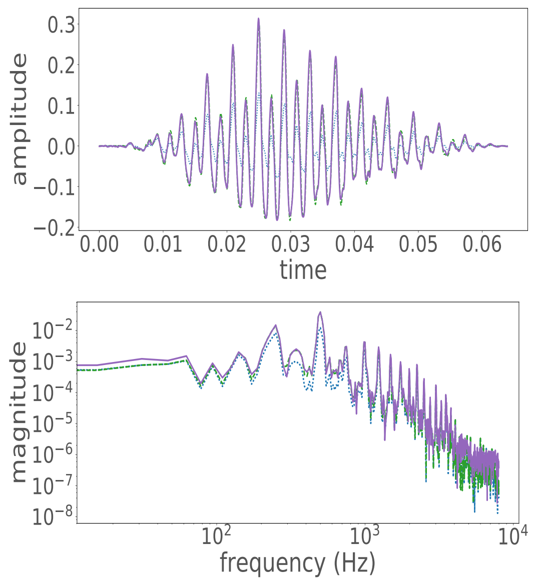
Figure 4.2: Various plots after the pretraining step. Figs. 4.2a and 4.2b show the input x and output $\hat{x}$ frames and their respective FFT magnitude. Fig. 4.2c shows 32 filters from $\mathbf{W}_1$; Figs. 4.2d and 4.2e the respective 32 rows of $\mathbf{X}_1$ and $\mathbf{Z}$, accordingly; Fig. 4.2f the resulting element-wise multiplication between $\mathbf{R}$ and $\hat{\mathbf{X}}_2$. Vertical axes in Figs. 4.2c to 4.2f are unitless and horizontal axes correspond to time.



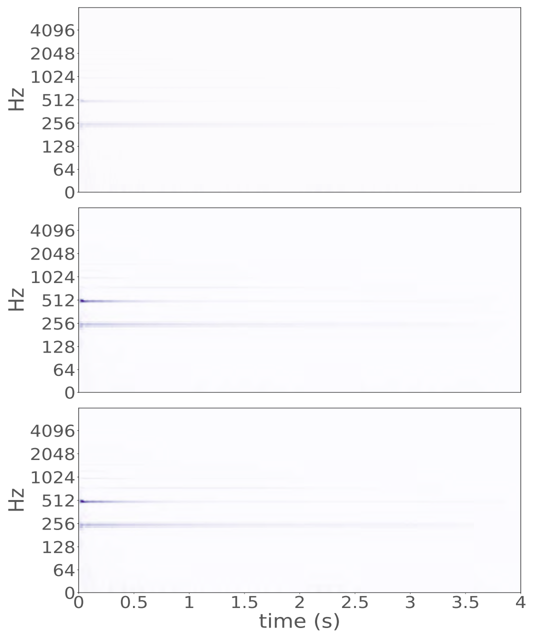
(a)



(b)

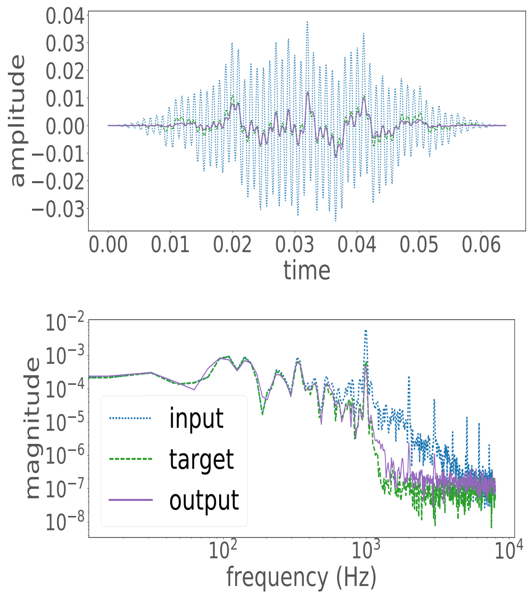


(c)

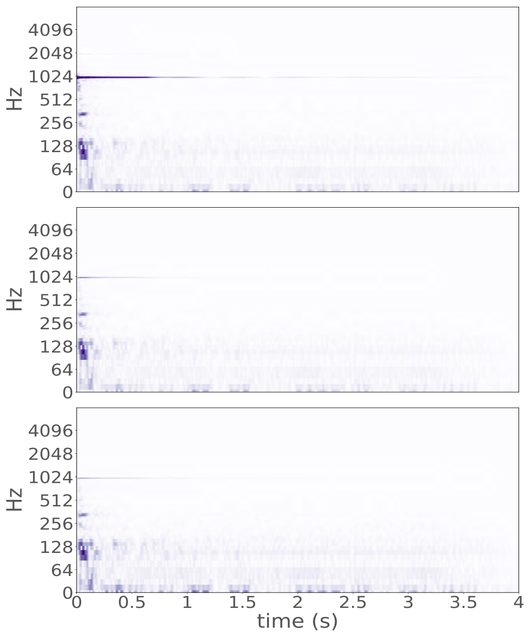


(d)

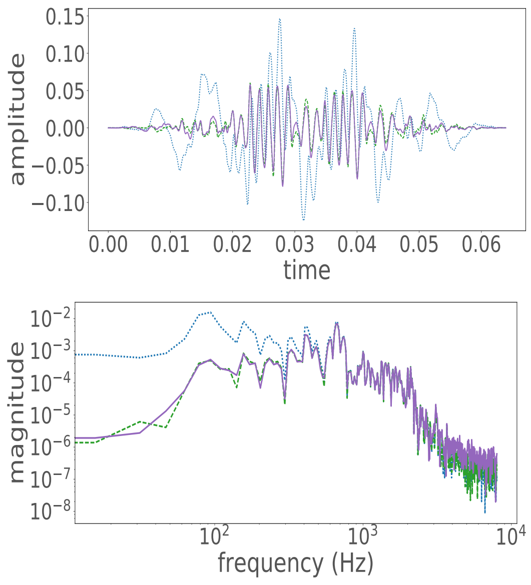
Figure 4.3: Results with the test dataset for the following EQ tasks. Figs. 4.3a and 4.3b: *shelving*, Figs. 4.3c and 4.3d: *peaking*. Figs. 4.3a and 4.3c show the input, target and output frames of 1024 samples and their respective FFT magnitudes. Figs. 4.3b and 4.3d show from top to bottom: input, target and output power spectrograms of the 4-second test samples. Colour intensity represents higher energy.



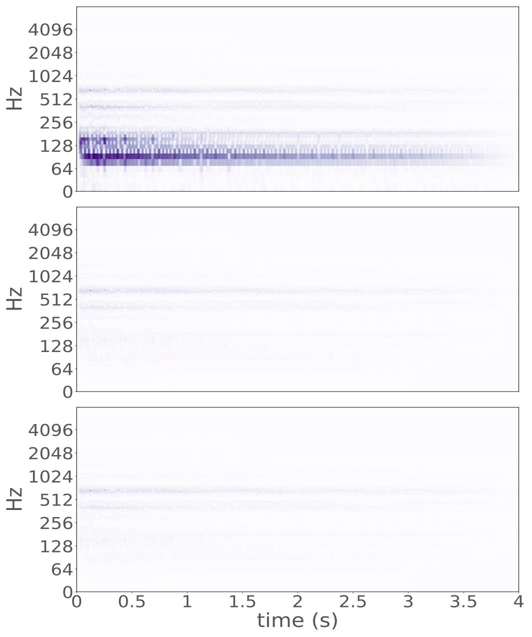
(a)



(b)



(c)



(d)

Figure 4.4: Results with the test dataset for the following EQ tasks. Figs. 4.4a and 4.4b: *lowpass*, Figs. 4.4c and 4.4d: *highpass*. Figs. 4.4a and 4.4c show the input, target and output frames of 1024 samples and their respective FFT magnitudes. Figs. 4.4b and 4.4d show from top to bottom: input, target and output power spectrograms of the 4-second test samples. Color intensity represents higher energy.

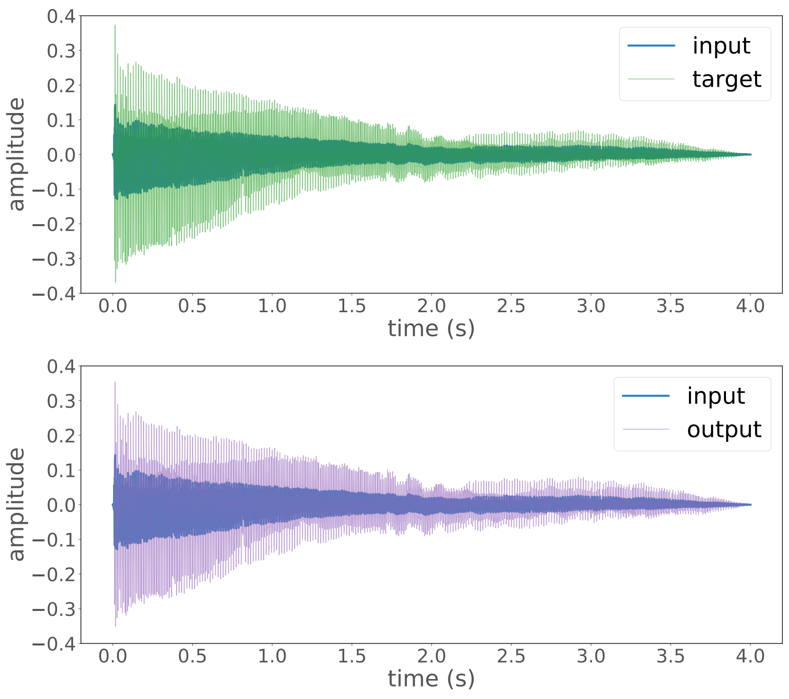


Figure 4.5: For a test sample of the *shelving* EQ task, complete waveform reconstruction of the output and comparison with the input and target. See Fig. 4.3b for the power spectrogram of these waveforms.

Table 4.3: *Evaluation of the models with the test datasets. Loss values for each EQ task.*

EQ	*KL*	*MSE*	*MAE*	*loss*
shelving	0.021845	0.007764	0.02474	0.032083
peaking	0.022038	0.007847	0.02521	0.032406
lowpass	0.025365	0.005345	0.02710	0.033420
highpass	0.021463	0.000951	0.01293	0.023708

4.4 DISCUSSION

Adaptive front-end and synthesis back-end

From the results of the pretraining step, Fig. 4.2 shows that the model manages to reconstruct the input frame almost perfectly. There are minor differences between the magnitudes of the lower and higher frequencies, but it is worth mentioning that the network achieves this by optimizing only two convolutional layers.

During this first training step, the model learns the $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ weight matrices with 128 filters each. These filters correspond to the weights of the front-end for the decomposition and reconstruction of the training data. As expected, from the $\boldsymbol{W}_1$ kernels shown in Fig. 4.2c, it can be observed the filters represent sinusoids and distributions of different frequencies. Also, upon examination of all the weights, we find some redundancy between the filters. This can be improved by adding kernel or activity regularizations, such as the L1 or L2 norm regularizers. In addition, some weights follow the shape of the *hann* window, which is expected since all input frames were windowed.

From the output feature map matrix $\mathbf{X}_1$ in Fig. 4.2d it can be seen the filters $\boldsymbol{W}_1$ are actively acting as a filter bank or frequency selectors. Thus, $\mathbf{X}_1$ correspond to the decomposition of the input data into different frequency bands. Since $\mathbf{X}_1$ is also the residual matrix $\mathbf{R}$, this feature map consists of the frequency band decomposition that is fed to the back-end to be modified and resynthesized according to the EQ task.

The second convolutional layer is acting as a smoothing layer, since $\mathbf{X}_2$ corresponds to envelopes from $\mathbf{X}_1$. This is due to the learned averaging filters and the *absolute* and *softplus* activation functions. The subsampled feature map $\mathbf{Z}$ is shown in Fig. 4.2e, where different types of envelopes are evident. Therefore, the model is learning a latent-space representation based on the envelopes of selected frequencies.

The resulting feature map of the unpooling layer $\hat{\mathbf{X}}_2$ corresponds to the values of $\mathbf{Z}$ at the time positions registered by the *max-pooling* layer and padded with zeros between each maximum value. Therefore the element-wise multiplication of $\hat{\mathbf{X}}_2$ with $\mathbf{R}$ generates a discrete version of the latter, which indicates the amplitudes and positions in time that the deconvolution layer should use to reconstruct the input signal (see Fig. 4.2f). Thus, convolving $\hat{\mathbf{X}}_1$ with $\boldsymbol{W}_1^T$ generates the output frame presented in Fig. 4.2a.

The front-end and back-end manage to match closely and reconstruct the test piano notes. Since the training was performed with a hop size of 256 samples, an ideal unit sample hop size would decrease the loss value, although the training time will increase notably. Furthermore, it is worth noting that the convolutional layers have a relatively small number of trainable parameters (24,832), which achieves a low-error reconstruction.

EQ task

The model was trained in a frame-by-frame basis and the input frames were windowed. So the model learned the windowing procedure and the output frames followed the *hann* window shape. Therefore, in order to reconstruct the complete audio signal (see Fig. 4.5), no further windowing was needed, as the model generates windowed frames. The overlapping procedure was carried out by applying a gain in order to ensure a Constant Overlap-Add (Antoni and Schoukens, 2007), which is specific to the type of window and hop size.

Table 4.3 shows the model performance for each EQ task. To provide a reference, the mean loss value between the inputs and targets of the *shelving* testing samples is 1.21. The *KL* is fairly uniform across the four types of equalizers, with a minor increase for the *lowpass* EQ. The same can be said about the *MSE* and *MAE* with the exception of a significant decrease for the *highpass* EQ. Therefore, loss function values were minimal and the model is capable of matching the most common types of EQ, whether these are based on FIR or IIR filters.

The model achieved the best results during the *highpass* task, which could be an indication of the frequency distribution among the training data. Since only piano notes where used, and most spectral energy of acoustic pianos is within 250 Hz to 1 kHz with higher frequencies responsible for the perceived timbral quality of the notes (Koenig, 2014). Thus, having a 500 Hz cut-off frequency could signify that the model more effectively filters out the lower-end of the piano notes due to the frequency content of the dataset. The slightly worse performance for the *lowpass* task could be further explored by extending the dataset to include lower frequency samples.

Figs. 4.3 and 4.4 confirm the correct EQ matching for the different types of equalizers. The spectral and waveform comparison between input, target and output shows how accurate the model is at reconstructing an audio signal that matches the EQ task. For individual frames and complete piano notes, the different types of filtering are evident from the FFT magnitude and power spectogram, respectively.

For the *shelving* EQ in Fig. 4.3b, the effect of the equalizer can be seen in the target and output spectral plots. The power spectrogram shows how the spectral energy was boosted for frequencies lower than 500 Hz. From the FFT magnitude it can be noticed a minor deviation in the lower-end of the target, where there is a boost increment around 20 Hz. This could indicate a weak generalization around these frequencies, which could be improved by using a loss function with higher frequency resolution in the lower-end (Härmä et al., 2000).

The *peaking* equalizer can be seen in Fig. 4.3d. The selective boost at 500 Hz can be seen in both in the FFT magnitude and in the power spectrogram. There is a minor boost in the lower-end which is a consequence of the reasons discussed above. Overall the results indicate a significant fitting for the *peaking* EQ task. Accordingly, the model is able to match EQ tasks based on *peaking* and *shelving* IIR filters.

Likewise, the *lowpass* and *highpass* EQ targets were correctly accomplished. The spectrograms in Figs. 4.4b and 4.4d show the cut of frequencies higher than 500 Hz for the *lowpass* and the opposite for the *highpass*. As discussed, it can be seen that the model performs the best for the *highpass* EQ task, obtaining a highly accurate matching between target and output in both time and frequency domains.

4.5 CONCLUSION

In this chapter, we proposed *CEQ*: a deep learning architecture capable of performing EQ matching. To achieve this, based on the universal approximation capabilities of neural networks, we explored a model based on a convolutional adaptive front-end and back-end together with a latent-space DNN. Thus, we introduced a general-purpose architecture for EQ matching able to model different types of equalizers and filters.

We showed the model matching *shelving*, *peaking*, *lowpass* and *highpass* IIR and FIR equalizers. For each EQ task the model was trained via an end-to-end learning approach, which presents and advantage towards common methods of automatic EQ since no prior knowledge of the type of filters nor fixed filter bank architecture is required. Accordingly, the proposed model approximated the target as a content-based transformation without using or obtaining filter parameters. Therefore, the model learned a filter bank decomposition and latent representation from the training data, and correspondingly, how to modify it in order to obtain an audio signal that matches the EQ task.

As future work, the architecture can be further tested by matching analog equalizers or other linear audio effects with short-term memory. An exploration of the latent-space DNN, or deeper convolutional layers within the encoder and decoder could improve the results of the model. In addition, the modeling capabilities of the model can be explored by the use of regularizers, loss functions based on frequency wrappers or different loss weights α_1, α_2 and α_3.

Also, since training on piano semitones provides only a sparse sampling of the frequency dimension, the generalization capability of the model should be extended for much more complex audio signals, such as noise, human voice or non-musical sounds. Therefore, a further exploration with a less homogeneous dataset together with an analysis of the type of filters learned by the model could benefit the design of a general-purpose architecture for modeling audio effects.

Possible applications for this architecture are within the fields of automatic mixing and audio effect modeling. For example, style-learning of a specific sound engineer could be explored, where the model is trained with several tracks equalized by the engineer and finds a generalization from the engineer's EQ practices. Also, automatic EQ for a specific instrument across one or several genres could be analyzed and implemented by the model.

Perceptually, most output waveforms are indistinguishable from their EQ target, although a further subjective study or listening test is required. The network implemented in this chapter serves as a base model for deep learning architectures for audio effects modeling. Therefore we can conclude that, as proof of concept, the model achieved low-error matching with various EQ matching tasks. Nevertheless, a further comparative evaluation with existing methods is required to test the performance of the proposed model.

In the following chapters, we build on this architecture and explore whether adaptive activation functions or RNNs can improve the capabilities of the network to model much more complex audio effects. In this case, we will explore transformations involving long temporal dependencies such as compression or different modulation effects, as well as complicated distortion effects or electromechanical reverberators. In addition, perceptually-based objective functions and relevant listening tests will also be performed.

5 MODELING NONLINEAR AUDIO EFFECTS

In this chapter we build on the *CEQ* modeling network from Chapter 4 in order to emulate much more complex transformations, such as distortion effects. Therefore we introduce *CAFx*: a novel deep learning architecture for modeling nonlinear and linear audio effects with short-term memory. In addition, we also compare our model with the proposed architecture in Rethage et al. (2018), which is a feedforward variation of the original autoregressive model from Oord et al. (2016).

As mentioned in Sections 2.3 and 2.4, distortion effects are mainly used for aesthetic reasons and are usually applied to electric musical instruments. Most existing methods for nonlinear modeling are often either simplified or optimized to a very specific circuit. Thus, in this chapter we investigate end-to-end DNNs for black-box modeling of nonlinear audio effects.

For an arbitrary combination of linear and nonlinear audio effects with short-term memory, the models learn how to process the audio directly in order to match the target audio in a regression task. Given a nonlinearity, consider x and y the raw and distorted audio signals respectively. In order to obtain a $\hat{y}$ that matches the target y, we train a DNN to modify x based on the nonlinear task.

We explore nonlinear emulation as a content-based transformation without explicitly obtaining the solution of the nonlinear system. *CAFx*, a new model based on convolutional and dense layers can incorporate trainable activation functions, such as smooth adaptive activation functions (SAAFs, see Section 3.3), this in order to explicitly train SAAFs to act as waveshapers in audio processing tasks such as nonlinear modeling. Thus, since distortion effects are characterized by their waveshaping nonlinearity, we rely on the smooth attributes of SAAFs, which can approximate any continuous function, to act as trainable waveshapers within a DNN modeling framework.

In this manner, we explore the capabilities of DNNs as audio processing blocks in the context of modeling nonlinear audio effects. Through the use of specific domain knowledge, such as waveshaping nonlinearities, we increase the function approximation capabilities of DNNs when performing nonlinear audio processing tasks with short-term memory.

Through the same nonlinear modeling tasks we analyse a model solely based on temporal dilated convolutions. We measure the performance of the models via a perceptually-based objective metric and we report that both models perform similarly when modeling *distortion*, *overdrive*, *amplifier emulation* and *combinations of linear and nonlinear* digital audio effects.

In the following sections we present the architecture of the different modeling networks. All the models are based entirely in the time-domain and are end-to-end; with raw audio as the input and processed audio as the output. Code is availabe online[1] and the number of parameters and processing times are shown in Appendix C. Audio samples for *CAFx* can be found in Appendix B.

5.1 CONVOLUTIONAL AUDIO EFFECTS MODELING NETWORK - CAFX

Following the *CEQ* architecture, the model is divided into three parts: adaptive front-end, synthesis back-end and latent-space DNN. The architecture is designed to model nonlinear audio effects with short-term memory and is based on a parallel combination of cascade input filters, trainable waveshaping nonlinearities, and output filters. All convolutions are along the time dimension, all strides are of unit value and no dilation is incorporated. We use the same padding as in *CEQ*.

The model is depicted in Fig. 5.1 and its structure is described in detail in Table 5.1. We use an input frame of size 1024, sampled with a hop size of 256 samples and rectangular windows.

The *adaptive front-end* and *latent-space DNN* are exactly the same as in *CEQ* (see Section 4.1). The main difference is the incorporation of dense layers and SAAFs into the back-end. This in order to allow the model to learn the waveshaping nonlinearities that characterize distortion effects.

Synthesis back-end

The synthesis back-end accomplishes the nonlinear task by the following steps. First, $\hat{\mathbf{X}}_2$ corresponds to a matrix of 128 channels of 1024 samples and is obtained via unpooling the modified envelopes $\hat{\mathbf{Z}}$. Then the feature map $\hat{\mathbf{X}}_1$ is the result of the element-wise multiplication of the residual connection $\mathbf{R}$ and $\hat{\mathbf{X}}_2$. This can be seen as an input filtering operation, since a different envelope gain is applied to each of the frequency band decompositions obtained in the front-end.

1 https://github.com/mchijmma/DL-AFx/tree/master/src

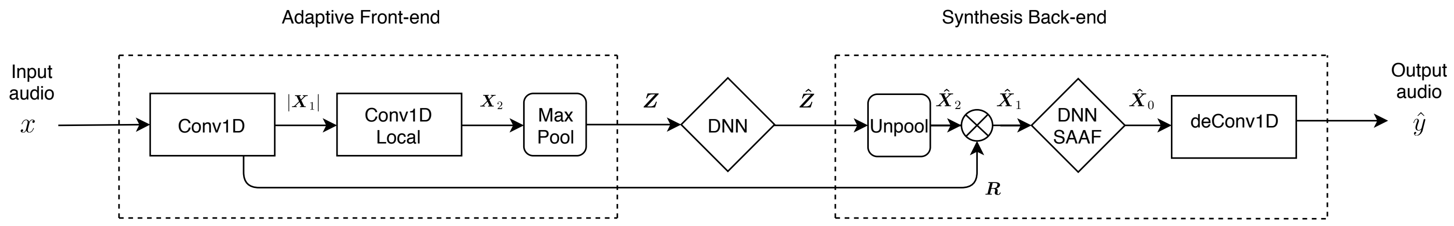


Figure 5.1: Block diagram of *CAFx*; adaptive front-end, synthesis back-end and latent-space DNN.

Table 5.1: Detailed architecture of *CAFx* with an input frame size of 1024 samples. Output shape = (m,n) denotes m columns and n rows. Weights = $m(n)$ denotes m kernels of size n, and Weights = n denotes n hidden units.

Layer	Output shape	Weights	Output
Input	(1024, 1)	.	x
Conv1D	(1024, 128)	128(64)	X_1
Residual	(1024, 128)	.	R
Abs	(1024, 128)	.	.
Conv1D-Local	(1024, 128)	128(128)	X_2
MaxPooling	(64, 128)	.	Z
Dense-Local	(64, 128)	64	.
Dense	(64, 128)	64	$\hat{Z}$
Unpooling	(1024, 128)	.	$\hat{X}_2$
$R \times \hat{X}_2$	(1024, 128)	.	$\hat{X}_1$
Dense	(1024, 128)	128	.
Dense	(1024, 64)	64	.
Dense	(1024, 64)	64	.
Dense	(1024, 128)	128	.
SAAF	(1024, 128)	128(25)	$\hat{X}_0$
deConv1D	(1024, 1)	.	$\hat{y}$

The second step is to apply various waveshapping nonlinearities to $\hat{X}_1$. This is achieved with a processing block containing dense layers and smooth adaptive activation functions (DNN-SAAF). The DNN-SAAF consists of 4 FC layers. The weights of the dense layers are applied to the channel dimension of the input feature maps, i.e. the matrix multiplication is computed between the weights and the columns of the input matrix which correspond to 1024 samples of 128 channels. All dense layers are followed by the *softplus* function with the exception of the last layer. Locally connected SAAFs are used as the nonlinearity for the last layer. Overall, each SAAF is locally connected and composed of 25 intervals between -1 to $+1$.

We tested different standard and adaptive activation functions, such as the parametric and non parametric *ReLU*, hyperbolic tangent, sigmoid and fifth order polynomials. Nevertheless, we found stability problems and non optimal results when modeling nonlinear effects. Since each SAAF explicitly acts as a waveshaper, the DNN-SAAF is constrained to behave as a set of trainable waveshaping nonlinearities, which follow the filter bank architecture and are applied to the channel dimension of the modified frequency decomposition $\hat{\mathbf{X}}_1$.

Finally, the last layer corresponds to the deconvolution operation, which can be implemented by transposing the first layer transform. As in *CEQ*, this layer is not trainable since its kernels are transposed versions of $\mathbf{W}_1$. In this way, the back-end reconstructs the audio waveform in the same manner that the front-end decomposed it. The complete waveform is synthesized using a *hann* window and constant overlap-add gain.

5.2 FEEDFORWARD WAVENET AUDIO EFFECTS MODELING NETWORK - WAVENET

The *WaveNet* model corresponds to a feedforward variation of the original autoregressive model from Oord et al. (2016). Recently it has been shown that temporal dilated convolutions can be effective for modeling raw audio (Rethage et al., 2018; Mor et al., 2019; Chorowski et al., 2019). For a regression task, such as nonlinear modeling, the predicted samples are not fed back into the model, but through a sliding input window, where the model predicts a set of samples in a single forward propagation. The *WaveNet* model corresponds to the architecture proposed in Rethage et al. (2018). The model is divided into two parts: stack of dilated convolutions and a post-processing block. The model is depicted in Fig. 5.2 and its structure is described in Table 5.2.

We use 2 stacks of 6 dilated convolutional layers with a dilation factor of 1,2,...,32 and 16 filters of size of 3. This model differs from the network in Damskägg et al. (2019), where their implementation uses 1 stack of dilated convolutions with a dilation factor of 1,2,...,512 and a post-processing block based solely on FC layers.

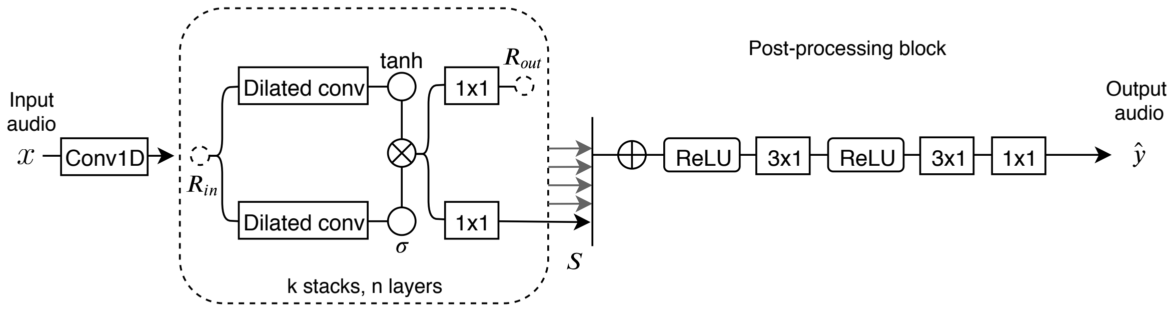


Figure 5.2: Block diagram of the feedforward *WaveNet*; stack of dilated convolutional layers and the post-processing block.

From Figure 5.1, prior to the stack of dilated convolutions, the input x is projected into 16 channels via a 3x1 convolution. This is in order to match the number of channels within the feature maps of the dilated convolutions.

The *stack of dilated convolutions* processes the input feature map $\mathbf{R}_{in}$ with 3x1 gated convolutions and exponentially increasing dilation factors. This operation can be described by:

$$z = \tanh(\boldsymbol{W}_f * \mathbf{R}_{in}) \times \sigma(\boldsymbol{W}_g * \mathbf{R}_{in}) \tag{5.1}$$

Where $\boldsymbol{W}_f$ and $\boldsymbol{W}_g$ are the filter and gated convolutional kernels, tanh and σ the hyperbolic tangent and sigmoid functions and $*$ and $\times$ the operators for convolution and element-wise multiplication. The residual output connection $\mathbf{R}_{out}$ and the skip connection $\mathbf{S}$ are obtained via a 1x1 convolution applied to z. Therefore $\mathbf{S}$ is sent to the post-processing block and $\mathbf{R}_{out}$ is added to the current input matrix $\mathbf{R}_{in}$, thus, resulting in the residual input feature map of the next dilated convolutional layer.

The *post-processing block* consists in summing all the skip connections $\mathbf{S}$ followed by a *ReLU*. Two final 3x1 convolutions are applied to the resulting feature map, which contain 2048 and 256 filters and are separated by a *ReLU*. As a last step, a 1x1 convolution is introduced in order to obtain the single-channel output audio $\hat{y}$.

The receptive field *rf* corresponds to the largest range of input samples to which the network can be exposed, and for a model based on time dilated convolutions it can be calculated with the following equation (Oord et al., 2016):

$$rf = 1 + n(f_k - 1) \sum_{i=1}^{D} d_i, \tag{5.2}$$

Table 5.2: Detailed architecture of *WaveNet* with input and output frame sizes of 1277 and 1024 samples respectively. Output shape = (m,n) denotes m columns and n rows. Weights = $m(n)$ denotes m kernels of size n.

Layer - Output shape - Weights		Output
Input (1276, 1)		x
Conv1D (1276, 16) - 16(3)		$\mathbf{R}_{\text{in}}$
Dilated conv (1276, 16) - 16(3)	Dilated conv (1276, 16) - 16(3)	.
Tanh (1276, 16)	Sigmoid (1276, 16)	. .
Multiply (1276, 16)		z
Conv1D (1276, 16) - 16(1)	Conv1D (1276, 16) - 16(1)	$\mathbf{R}_{\text{out}}$ $\mathbf{S}$
Add (1024, 16)		.
ReLU (1024, 16)		.
Conv1D (1024, 2048) - 2048(3)		.
ReLU (1024, 16)		.
Conv1D (1024, 256) - 256(3)		.
Conv1D (1024, 1) - 1(1)		$\hat{y}$

where n is the number of stacks, f_k is the size of the filters, D is the number of dilated layers and d_i corresponds to each dilation factor. For this architecture, the receptive field of the model is of 253 samples and the target field *tf* is 1024 samples. Therefore the input frame *if* presented to the model consists of sliding windows of 1276 samples and is calculated as follows (Rethage et al., 2018).

$$if = rf + (tf - 1) \tag{5.3}$$

5.3 EXPERIMENTS

5.3.1 *Training*

The training of *CAFx* includes the same initialization step as *CEQ* (see Section 4.2.1) Once the model is pretrained, the latent-space DNN and DNN-SAAF are incorporated into the model, and all the weights of the convolutional, dense and activation layers are trained following an end-to-end supervised learning task. The *WaveNet* model is trained directly following this second step.

Since small amplitude errors are as important as large ones, the loss function to be minimized is the mean absolute error, see Eq. (3.1), between the target and output waveforms. In addition, in both training procedures, the input and target audio segments are framed with a rectangular window. The batch size consisted of the total number of frames per audio sample and 1000 iterations were carried out in each training step. The learning rate is 10^{-4}. We select the model with the lowest error for the validation subset and *Adam* (Kingma and Ba, 2015) is used as optimizer.

5.3.2 *Dataset*

The audio is obtained from the *IDMT-SMT-Audio-Effects* dataset (Stein et al., 2010), which is a large library of 20,592 monophonic recordings of electric guitar and bass guitar respectively. The samples correspond to individual 2-second notes and cover the common pitch range of 2 different 6-string electric guitars and 2 different 4-string bass guitars.

The recordings include the raw notes and their respective effected versions after 3 different settings for each effect. We use unprocessed and processed audio with *distortion*, *overdrive*, and *EQ*. These effects correspond to digital implementations of audio processors, such as Virtual Studio Technology (VST) audio plug-ins (Steinberg, 1999).

In addition, we also apply a custom audio effects chain (*FxChain*) to the raw audio. The *FxChain* consist of different cascade configurations of *lowshelf* and *highshelf* filters (see Section 2.1) together with an *overdrive*. We use *SoX*[2] to implement the nonlinear audio effect and the filters.

In total we use 624 raw and distorted notes for each audio effect setting. The test and validation samples are randomly selected and correspond to 5% of this dataset each. The recordings were downsampled to 16 kHz and Table 5.3 shows the details of the settings for each audio effect.

For the *FxChain* task we further evaluate the generalization capabilities of the trained modeling networks. We test the models with recordings from a broader dataset, i.e. the NSynth dataset (Engel et al., 2017). This dataset consists of individual notes of 4 seconds from more than 1000 instruments.

2 `http://sox.sourceforge.net/`

5.3.3 *Evaluation*

Two metrics are used when testing the models with the various nonlinear modeling tasks. Since the mean absolute error depends on the amplitude of the output and target waveforms, before calculating this metric, we normalize the energy of the target and the output and define it as the energy-normalized mean absolute error (*mae*).

As a perceptually-based objective metric, we measure the MFCCs which are based on a model of human auditory perception (see **??**). The *mfcc_cosine* corresponds to the mean cosine similarity of the MFCCs. The cosine similarity or distance computes the cosine of the angle between two vectors in order to measure their similarity. Therefore it consists of the dot product between two vectors divided by the product of their *L*2 norms. The cosine distance indicates whether the vectors are pointing in the same direction (Han et al., 2011).

The *mfcc_cosine* is calculated as follows:

- A log-power-melspectogram is computed from the energy-normalized audio waveforms. This is calculated with 40 mel-bands and audio frames of 4096 samples and 50% hop size.
- 13 MFCCs are computed using the DCT and the *mfcc_cosine* metric is the mean cosine similarity across the MFCC vectors.

5.4 RESULTS

The training procedures were performed for each type of nonlinear effect and for bass guitar and electric guitar, i.e. we train a model for each audio effect and instrument, respectively. Then, the models were tested with samples from the test dataset. Audio examples for *CAFx* are available online[3].

Figs. 5.3 and 5.4 show the *mae* and *mfcc_cosine* for the different models when tested with the test subset of each modeling task. For each *FxChain* setting and for each instrument, Fig. 5.10 shows the *mae* and *mfcc_cosine* values when the models are tested with the test samples of the NSynth dataset.

For selected test samples of the *distortion* and *overdrive* tasks and for both *CAFx* and *WaveNet*, Figs. 5.5 and 5.6 show the input, reference, and output waveforms together with their respective spectrogram. To more closely display the performance of these nonlinear tasks, Fig. 5.7 shows a segment of the respective waveforms.

3 https://github.com/mchijmma/modeling-nonlinear

In addition, for both *CAFx* and *WaveNet*, Figs. 5.8 and 5.9 show the target and obtained waveshaping nonlinearities for various bass guitar and electric guitar modeling tasks. This for the *distortion* and *overdrive* tasks, respectively.

Table 5.3: Settings for each audio effect modeling task.

Fx	#	Bass	Guitar
distortion	1	**Sawing**: 'Edge - 9 o clock', 'Gain - 3 o clock', 'Level - 12 o clock', 'Tone - 3 o clock'	**Sawing**: 'Edge - 10 o clock', 'Gain - 2 o clock', 'Level - 12 o clock', 'Tone - 1 o clock'
	2	**Screaming**: 'Contour - 80%', 'Drive - 90%', 'Factory Preset - evil', 'Input - 0dB', 'Output - -17dB', 'Shape - 3'	**Screaming**: 'Contour - 90%', 'Drive - 95%', 'Factory Preset - wracky', 'Input - 0dB', 'Output - -17dB', 'Shape - 3'
	3	**Muffled**: 'Drive - 30%', 'Muffle - 60%', 'Output - -2dB'	**Muffled**: 'Drive - 40%', 'Muffle - 40%', 'Output - 0dB'
overdrive	1	**Rock Bass** 'Drive - 3', 'High Frequency - 5 kHz', 'High Q - 0.5', 'Input - 0dB', 'Low Cut - 40 Hz', 'Mid Frequency - 250 Hz', 'Mid Gain - -3dB', 'Output - -7dB', 'Shape - 4', 'Tone - 3'	**Rocking Blues** 'Drive - 7', 'High Frequency - 5 kHz', 'High Q - 1.4', 'Input - 0dB', 'Low Cut - 325 Hz', 'Mid Frequency - 600 Hz', 'Mid Gain - 0dB', 'Output - -4dB', 'Shape - 7', 'Tone - 4'
	2	**Tubish**: 'Limiter - Off', 'Output - -20dB', 'Shape - 3 o clock', 'Shaper - On'	**Tubish**: 'Limiter - Off', 'Output - -10dB', 'Shape - 5 o clock', 'Shaper - On'
	3	**Drive**: 'Bass - 3dB', 'Factory Preset - magic OD', 'High - -12dB', 'Input - -2dB', 'Mid - 6dB', 'Output - -15dB'	**Drive**: 'Bass - 3dB', 'Factory Preset - magic OD', 'High - -3dB', 'Input - 0dB', 'Mid - 6dB', 'Output - -10dB'
EQ	1	**Speaker Simulation**: 'Bias - 0', 'Drive - 0', 'Model - Spkr Sim', 'Output - 0 dB', 'Process - Mono'	**Fender Twin**: 'Bright - On', 'High - 5', 'Low - 5', 'Mid - 5', 'Tr Intens - 0', 'Tr Speed - 5', 'Volume - 8.8'
	2	**Rock Bass EQ**: 'Band 1 Frequency - 60 Hz', 'Band 1 Gain - 0 dB', 'Band 1 Q - 1', 'Band 1 Type - Shelve', 'Band 2 Frequency - 400 Hz', 'Band 2 Gain - 3 dB', 'Band 2 Q - 1', 'Band 2 Type - Peak', 'Band 3 Frequency - 1500 Hz', 'Band 3 Gain - 4 dB', 'Band 3 Q - 2', 'Band 3 Type - Peak', 'Band 4 Frequency - 4500 Hz', 'Band 4 Gain - -3 dB', 'Band 4 Q - 1', 'Band 4 Type - Peak'	**Speaker Simulation**: 'Bias - 0', 'Drive - 0', 'Model - 4x12 1', 'Output - 0dB', 'Process - Mono'
FxChain	1	**lowshelf**: 'Gain - +20 dB', 'Frequency - 500 Hz' **highshelf**: 'Gain - -20 dB', 'Frequency - 500 Hz' **overdrive**: 'Gain - +30 dB', 'Colour - 0'	**lowshelf**: 'Gain - +20 dB', 'Frequency - 500 Hz' **highshelf**: 'Gain - -20 dB', 'Frequency - 500 Hz' **overdrive**: 'Gain - +30 dB', 'Colour - 0'
	2	**lowshelf**: 'Gain - +20 dB', 'Frequency - 500 Hz' **overdrive**: 'Gain - +30 dB', 'Colour - 0' **highshelf**: 'Gain - -20 dB', 'Frequency - 500 Hz'	**lowshelf**: 'Gain - +20 dB', 'Frequency - 500 Hz' **overdrive**: 'Gain - +30 dB', 'Colour - 0' **highshelf**: 'Gain - -20 dB', 'Frequency - 500 Hz'
	3	**overdrive**: 'Gain - +30 dB', 'Colour - 0' **lowshelf**: 'Gain - +20 dB', 'Frequency - 500 Hz' **highshelf**: 'Gain - -20 dB', 'Frequency - 500 Hz'	**overdrive**: 'Gain - +30 dB', 'Colour - 0' **lowshelf**: 'Gain - +20 dB', 'Frequency - 500 Hz' **highshelf**: 'Gain - -20 dB', 'Frequency - 500 Hz'

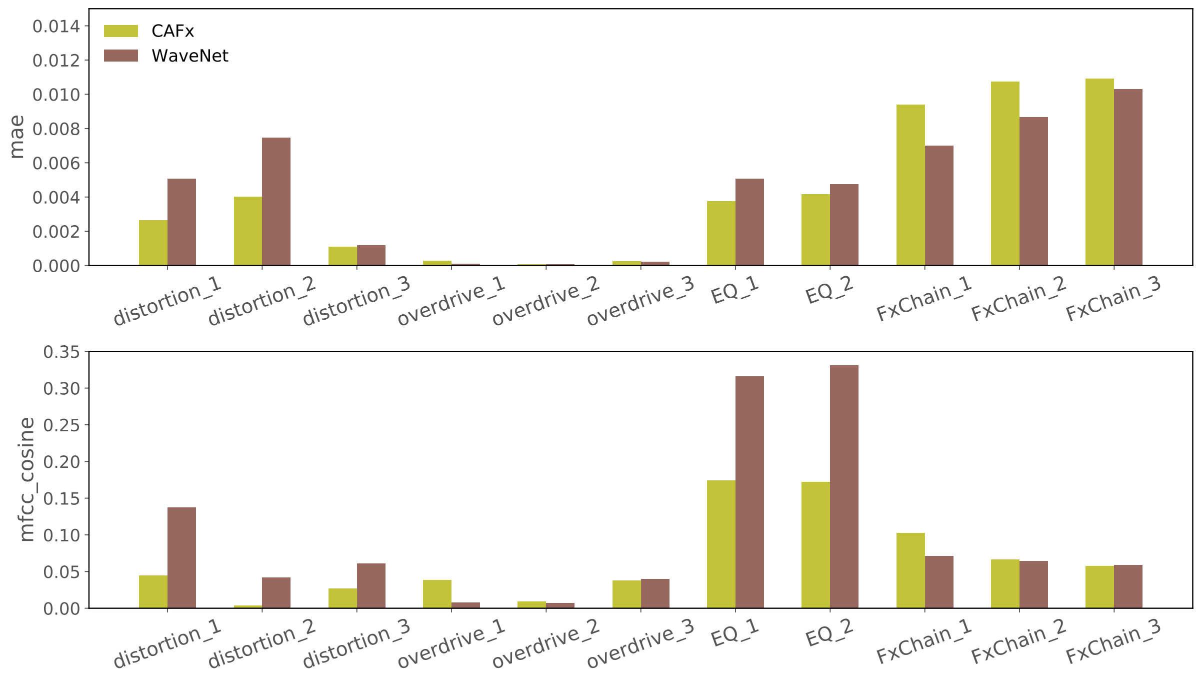


Figure 5.3: For both architectures and for all modeling tasks, the **mae** and **mfcc_cosine** values of *bass guitar* models when evaluated with test datasets. Lower is better for all metrics.

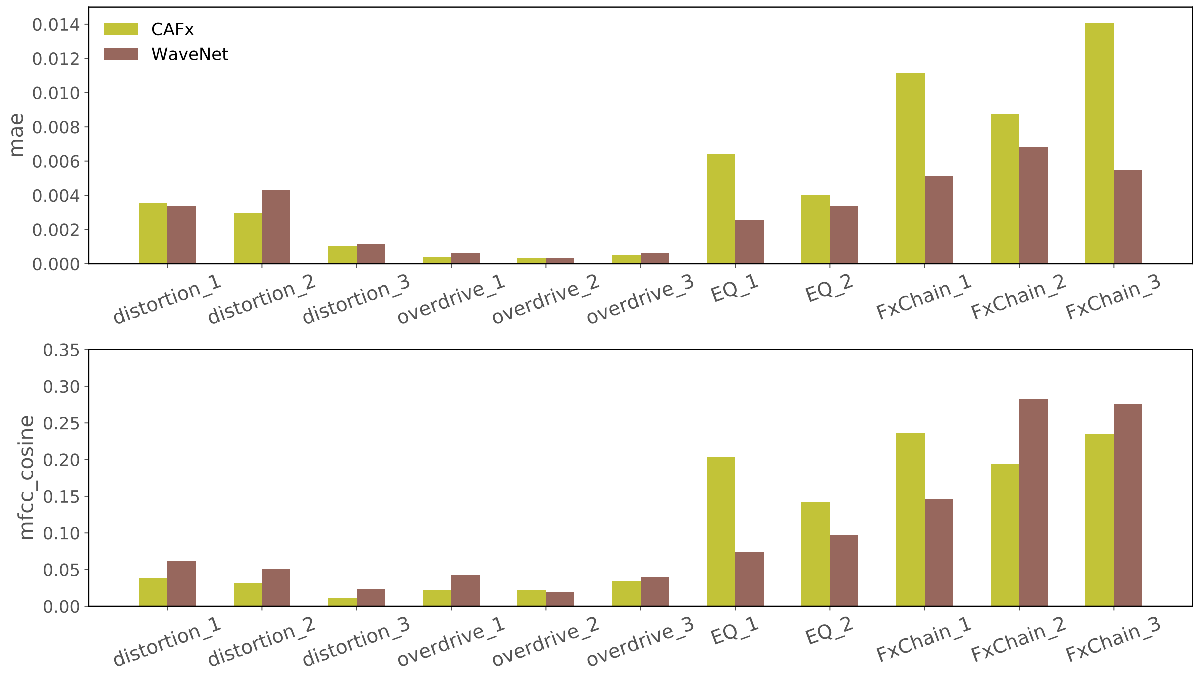
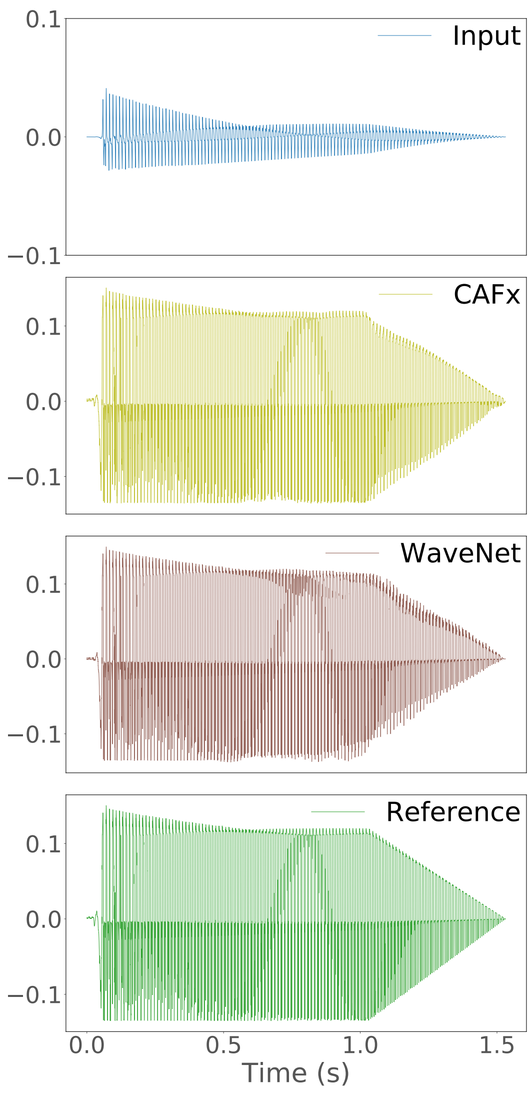
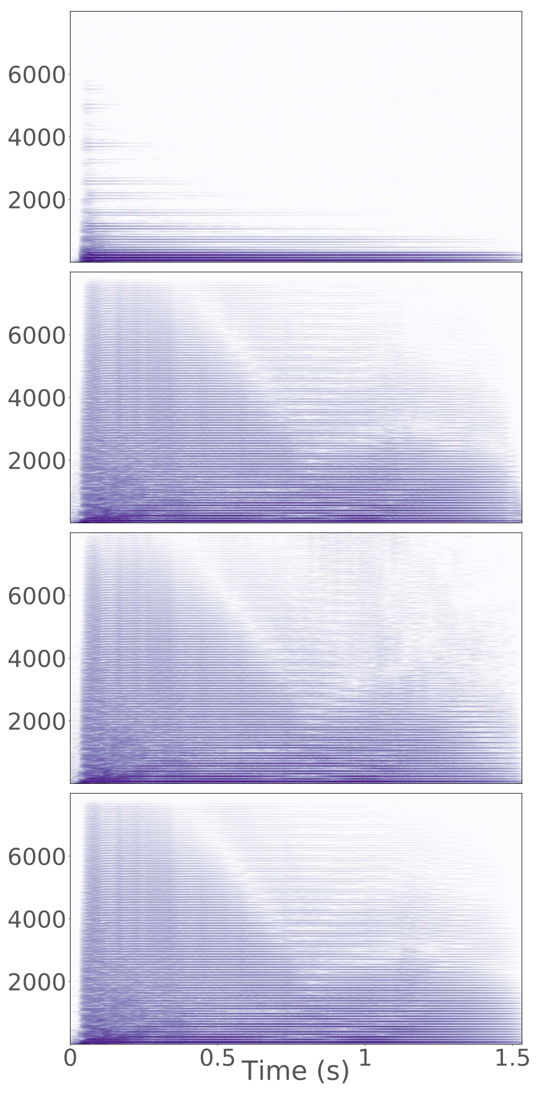


Figure 5.4: For both architectures and for all modeling tasks, the **mae** and **mfcc_cosine** values of *electric guitar* models when evaluated with test datasets. Lower is better for all metrics.



(a)



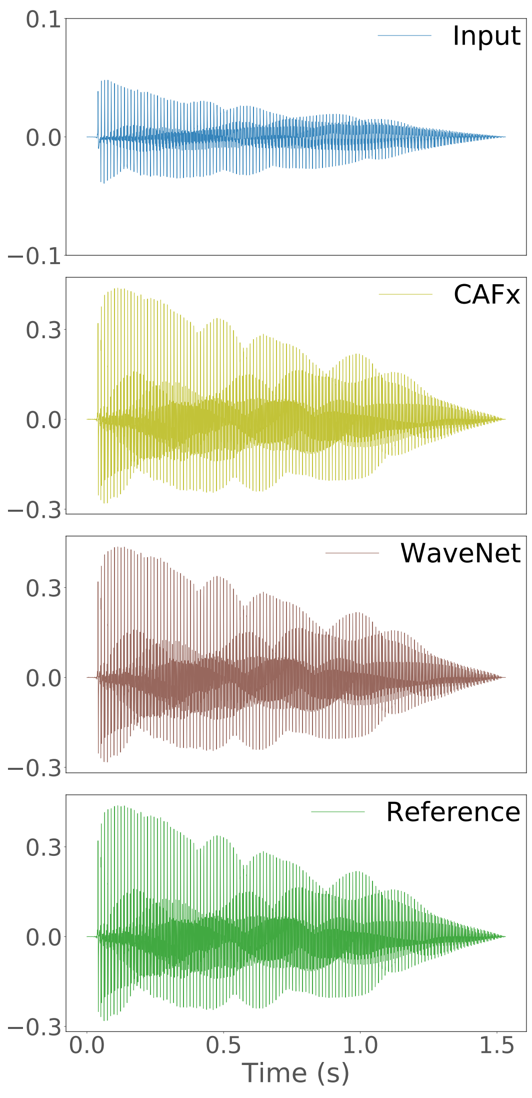
(b)

Figure 5.5: Results with selected samples from the test dataset for the bass guitar *distortion* task #1. The waveforms and their respective spectrograms are shown and vertical axes represent amplitude and frequency (Hz) respectively.

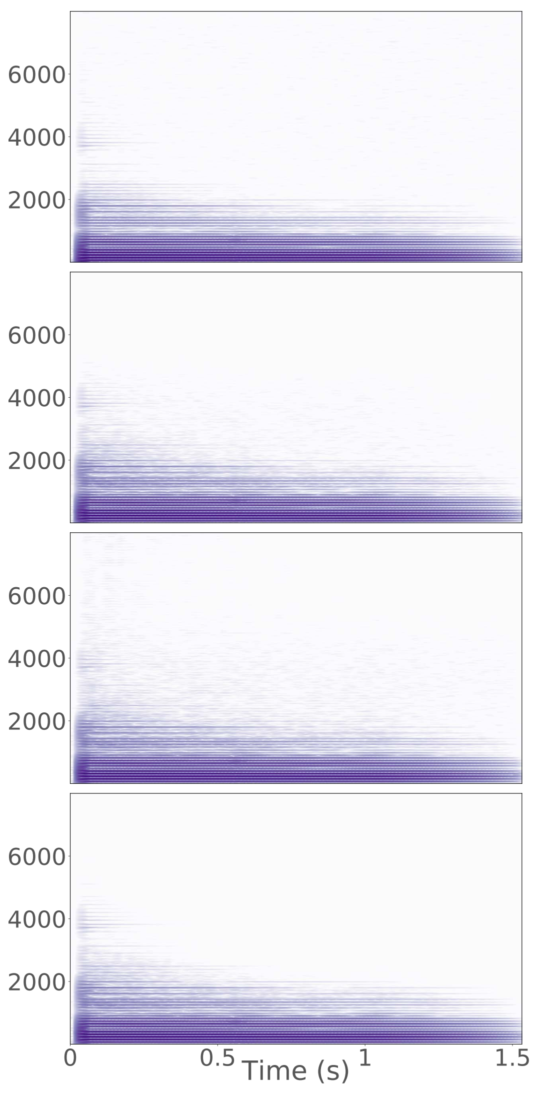
5.5 DISCUSSION

Nonlinear modeling tasks

Figs. 5.3 and 5.4 show the performance of the networks for each nonlinear modeling task and for both bass guitar and electric guitar models. To provide a reference,



(a)

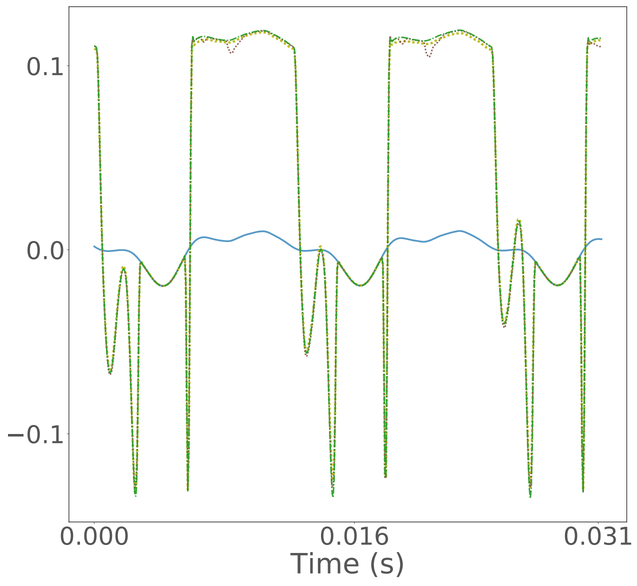


(b)

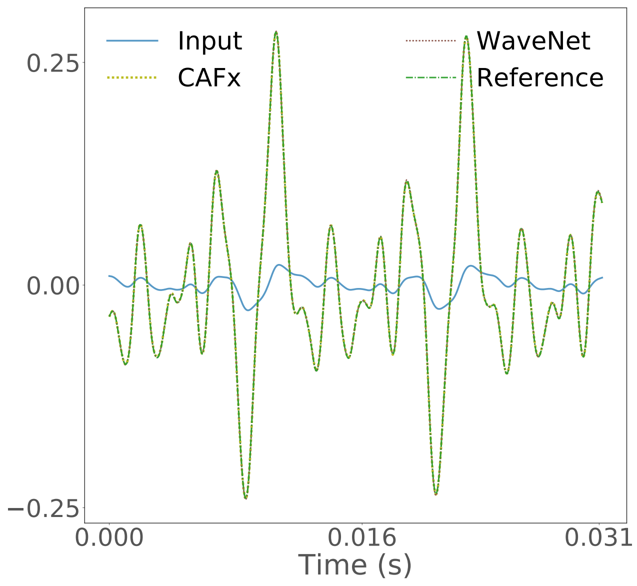
Figure 5.6: Results with selected samples from the test dataset for the electric guitar *overdrive* task #2. The waveforms and their respective spectrograms are shown and vertical axes represent amplitude and frequency (Hz) respectively.

across all modeling tasks, the mean *mae* and *mfcc_cosine* values between inputs and target waveforms are 0.1 and 0.9, respectively.

Overall, *CAFx* and *WaveNet* achieved low-error objective metrics. From Fig. 5.7, it can be seen that, both in time and frequency, the models accomplished the nonlinear target with high and almost identical accuracy. Furthermore, from Figs. 5.8 and 5.9, the models were able to match precisely the input-target waveshaping curve or ratio for selected settings. Therefore the models correctly accomplished



(a)



(b)

Figure 5.7: For the test samples from Figs. 5.5 and 5.6, a segment of the respective waveforms: 5.7a bass guitar *distortion* task #1 and Fig. 5.7b electric guitar *overdrive* task #2. Vertical axes represent amplitude. The reference, *CAFx* and *WaveNet* waveforms are mostly overlapping.

the timing settings from the nonlinear effects, such as attack and release, which are evident in the hysteresis behavior present in Figs. 5.8 and 5.9.

We obtained the best results with the overdrive task #2 for both instruments. This is due to the waveshaping curves from Fig. 5.9b where it can be seen that

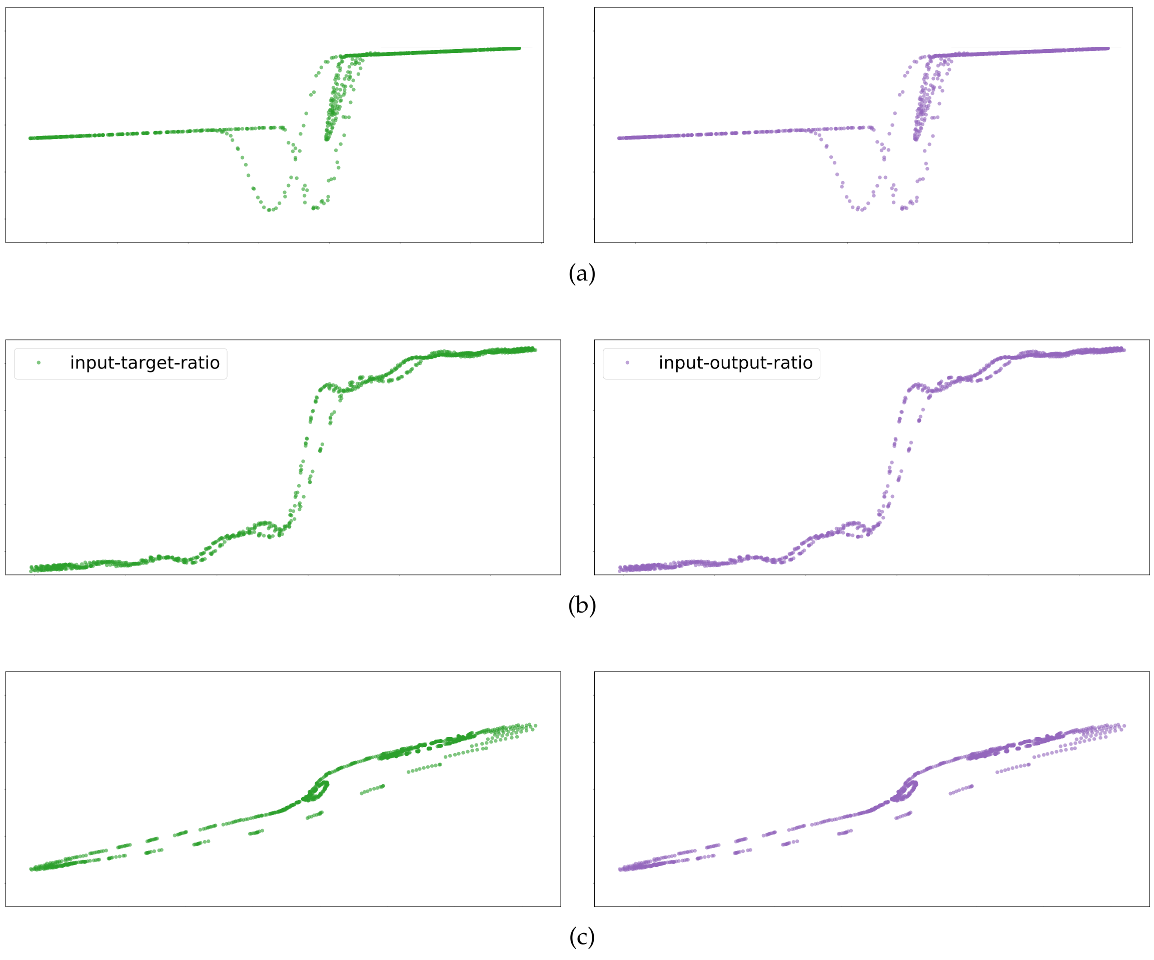
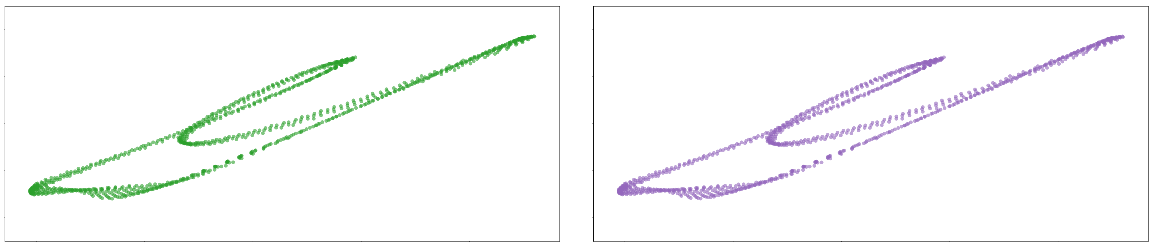


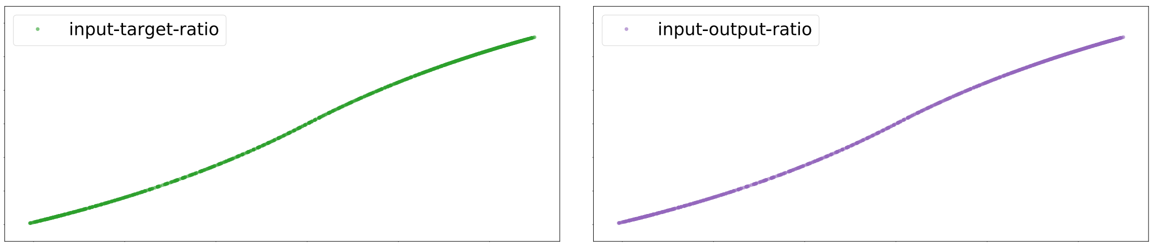
Figure 5.8: For *CAFx*, input-target and input-output waveshaping ratio for each *distortion* setting. Fig. 5.8a bass guitar *distortion* task #1. Fig. 5.8b electric guitar *distortion* task #2. Fig. 5.8c electric guitar *distortion* task #3. Horizontal axes represent input amplitude and vertical axes represent target/output amplitude.

the transformation does not involve timing nor filtering settings. We obtained the largest error for *FxChain* setting #3. Due to the extreme filtering configuration after the *overdrive*, it could be more difficult for the networks to model both the nonlinearity and the filters.

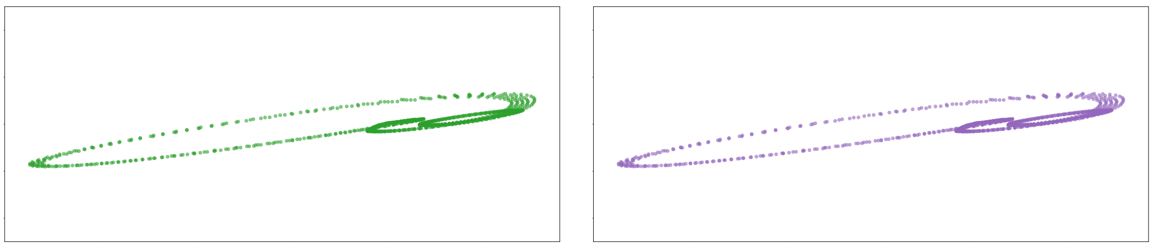
For the *distortion* and *overdrive* tasks and for both instruments, *CAFx* outperformed *WaveNet* based on the obtained objective metrics. In general, the tasks that involve filtering, such as *EQ* and *FxChain*, introduced the largest errors. In particular, *CAFx* and *WaveNet* decreased their performance when modeling electric guitars for theses tasks, e.g. the larger **mfcc_cosine** values for the electric guitar *FxChain* task.



(a)



(b)



(c)

Figure 5.9: For *WaveNet*, input-target and input-output waveshaping ratio for each *overdrive* setting. Fig. 5.9a electric guitar *overdrive* task #1. Fig. 5.9b electric guitar *overdrive* task #2. Fig. 5.9c bass guitar *overdrive* task #3. Horizontal axes represent input amplitude and vertical axes represent target/output amplitude.

Therefore for the *FxChain* task we further test the trained models. From Fig. 5.10 it can be seen that, when tested with different instrument recordings, both architectures decreased their performance with *WaveNet* obtaining better results than *CAFx*. This higher performance obtained by the *WaveNet* architecture might be due to the smaller size of the filters, which compared to larger *CAFx* filters are less prone to overfitting and could generalize better (Lee et al., 2018).

Furthermore, taking into account that in the other modeling tasks, *CAFx* tend to reach lower metric values, we could point towards a trade-off between optimization for a specific instrument and generalization among different instruments. This also means the convolutional and dense layers within *CAFx* are being tuned to find certain feature patterns of the respective instrument recordings. A training subset

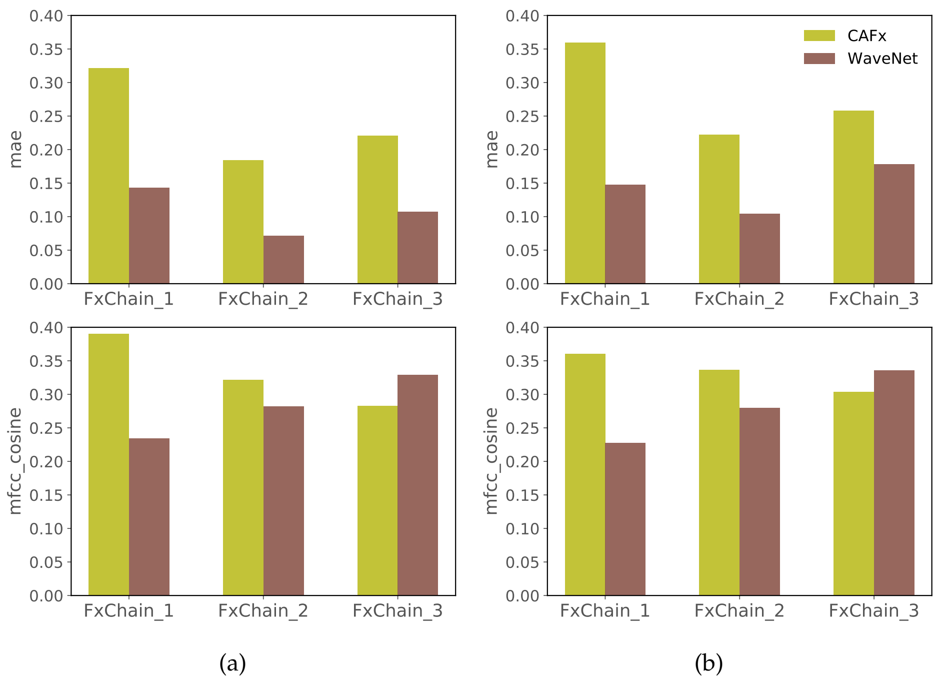


Figure 5.10: Evaluation of the generalization capabilities of the models. **mae** and **mfcc_cosine** values for *CAFx* and *WaveNet* when tested with a different test dataset (NSynth). Fig. 5.10a shows the metrics for the bass guitar models and Fig. 5.10b shows the metrics for the electric guitar models. Lower is better for all metrics.

with broader types of musical recordings could further improve the generalization capabilities of the models as well as regularization methods, such as dropout layers as shown in Publication **II**.

It is worth mentioning that the *EQ* task is also nonlinear, since the effects that were applied include amplifier emulation, which involves nonlinear modeling. Therefore, for this task, the models are also achieving linear and nonlinear modeling, although with lesser extent in terms of objective metric values. This can be further improved by exploring a loss function that considers the frequency-domain, such as Eq. (4.9) in the case of the *CEQ* model.

Also, the audio samples for all the effects from the *IDMT-SMT-Audio-Effects* dataset have a fade-out applied in the last 0.5 seconds of the recordings. Thus, when modeling nonlinear effects related to dynamics, this represents an additional challenge to the networks. We found that the *CAFx* might capture more closely this amplitude modulation via visual inspection of the waveforms, although additional tests are required.

Other black-box modeling methods suitable for the *FxChain* task, such as Wiener and Hammerstein models (Gilabert Pinal et al., 2005; Eichas and Zölzer, 2016, 2018), would require additional optimization in order to find the optimal combination of linear/nonlinear components. Moreover, further assumptions on the Wiener and Hammerstein static nonlinearity functions (i.e. invertibility) are needed and common nonlinearities which are not invertible are for example a dead-zone and a saturation (Hagenblad, 1999). Therefore, the proposed end-to-end deep learning architecture *CAFx* represents an improvement of the state-of-the art in terms of flexibility, although a further analysis of the trade-off between instrument specificity and generalization is required. Compared to the analytical methods, *CAFx* makes less assumptions about the modeled audio system and is thus more suitable for generic black-box modeling of nonlinear and linear audio effects.

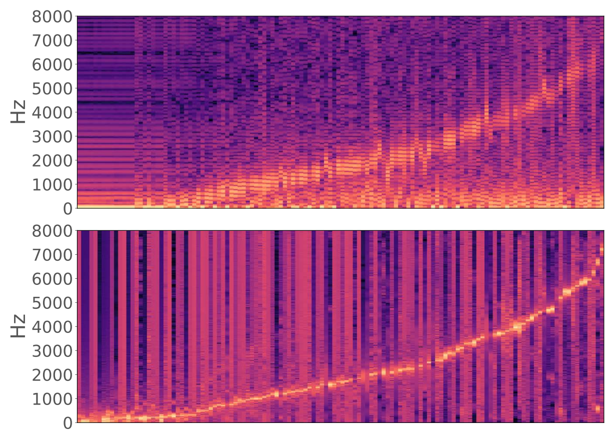
Gradient Ascent

We perform gradient descent analysis in order to obtain a further insight of the filters learned by *CAFx* in the layers *Conv1D* and *Conv1D-Local*. This method allows us to compute the input waveform that activates the most each convolutional filter and it has been shown to provide more meaningful interpretations of the features learned by CNNs (Lee et al., 2018; Erhan et al., 2009; Zeiler and Fergus, 2014).

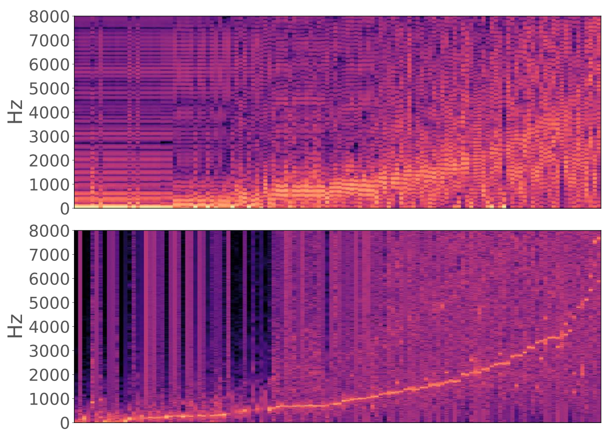
We start with a random noise frame as an input and a loss function that maximizes the activation of a specific filter. Then we use SGD to modify iteratively the input frame, thus, obtaining the input waveform that activates the most the respective filter. For selected nonlinear tasks, various spectrum visualizations of the obtained waveforms are displayed in Fig. 5.11.

From the spectrum visualizations for the *Conv1D* filters, it can be seen that the *distortion* and *EQ* modeling tasks introduce features with a narrow bandwidth, whereas for the *overdrive* and *FxChain* tasks, the computed features represent filters with a wider and less defined bandwidth. We can see that most filters contain high magnitudes for lower frequencies (<1000 Hz), which is similar to logarithmic compressed representations such as MFCCs.

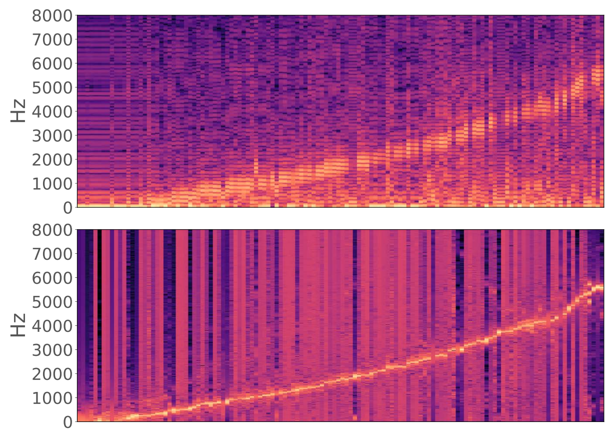
For the *Conv1D-Local* filters, the peak frequencies of the obtained features only reached the half of the sampling frequency for the *distortion* task. This could be due to the fact that this task represents the most severe nonlinearity and, therefore, implies greater harmonic distortion at high frequencies. In addition, the obtained



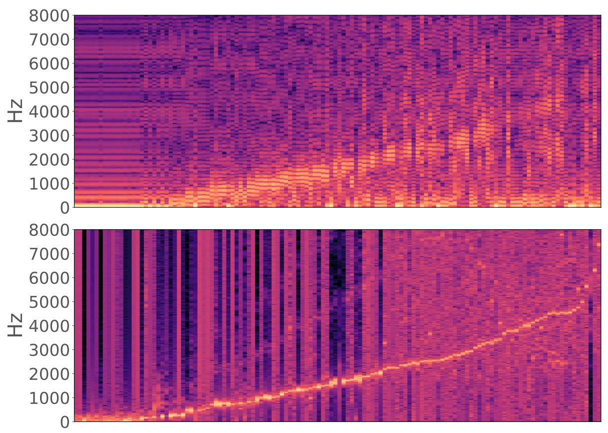
(a)



(b)



(c)



(d)

Figure 5.11: Spectrum visualization of the input waveforms that maximize the activation of each filter. Filters are ordered in ascending peak frequency and x-axis represents the index of the filter. From top to bottom: *Conv1D* and *Conv1D-Local* for the following modeling tasks: Fig. 5.11a bass guitar *distortion* task #1, Fig. 5.11b electric guitar *overdrive* task #2, Fig. 5.11c electric guitar *EQ* task #1 and Fig. 5.11c electric guitar *FxChain* task #3.

features represent more selective peak filters with narrower bandwidths. Logarithmic compression at the lower frequencies is also present.

5.6 CONCLUSION

In this chapter, we introduced *CAFx*, a general-purpose deep learning architecture for audio processing in the context of nonlinear modeling. Complex nonlinearities with attack, release and filtering settings were correctly modeled by the network due to the low-errors obtained in the objective metrics. Since the model was trained on a frame-by-frame basis, we can conclude that most transformations that occur within the frame-size will be captured by the network.

To achieve this, we built on *CEQ* and we explored an end-to-end network based on an adaptive convolutional front-end and back-end, latent-space DNNs and smooth adaptive activation functions. We showed the model matching *distortion*, *overdrive*, *amplifier emulation* and *combination of linear and nonlinear* audio effects. To the best of our knowledge, prior to this work, deep learning architectures have not been successfully implemented to model nonlinear and linear audio effects.

Through the use of specific domain knowledge, we explored trainable wave-shaping nonlinearities, such as SAAFs, along with a powerful DNN audio effects modeling network. Therefore we have shown the capabilities of DNNs as audio processing blocks in the context of modeling nonlinear audio effects with short-term memory.

We also compare our model to *WaveNet*, a nonlinear modeling network that corresponds to the feedforward variant of Oord et al. (2016) and implemented by Rethage et al. (2018). It is worth noting that this method was designed for speech denoising and not for audio effects modelling. Therefore, since Damskägg et al. (2019) proposed a similar architecture for tube amplifier emulation, further comparison with their model is required, which is shown to outperform the current state-of-the-art analytical method for nonlinear black-box modeling.

Overall, we measure the modeling capabilities of the models via a perceptually-based objective metric and we conclude that both architectures perform similarly. We show that the models accomplished the nonlinear modeling tasks with almost identical accuracy. Perceptually the obtained waveforms are virtually indistinguishable from their target counterparts although a listening test is required (see Chapter 7).

The *CAFx* architecture performed better when modeling nonlinear nonlinear tasks such as *distortion* and *overdrive*, while *WaveNet* generalized better when matching tasks that involve cascades of filters and nonlinearities. Generalization capabilities among instruments and optimization towards an specific instrument were found among the trained models. As future work, further generalization could be explored with the use of dropout layers or weight regularizers as well as training data with a wider range of instruments.

Although the models currently run on a GPU, real-time implementations could be further explored since processing times are already close to real-time temporal constraints. Table C.1 displays processing times using the non optimized python implementation. In addition, shorter input frames can also be explored for low-latency applications. Since the model is learning a static representation of the audio effect, parametric models could also be explored by introducing, during training, effect controls as inputs as shown in Hawley et al. (2019).

Possible applications for these architectures are digital emulations of nonlinear audio effects that indistinguishably match the sonic behavior of the analog counterparts. Also, automatic linear and nonlinear processing within an automatic mixing task can be investigated. Specific instrument recordings and their respective mixing processing could be analyzed and implemented by the models.

In the following chapter, we build on these architectures and we explore RNNs and latent-space temporal dilated convolutions to model transformations involving long term memory such as dynamic range compression or different modulation effects.

6 MODELING TIME-VARYING AUDIO EFFECTS

As mentioned in Section 2.5, audio effects whose parameters are modified periodically over time are often referred as time-varying or modulation based audio effects. Furthermore, as discussed in Section 2.3.2, a broad family of time-invariant audio effects is based on long-term dependencies, such as compressors. By assuming linear behaviour or by omitting certain nonlinear circuit components, most of these effects can be implemented directly in the digital domain through the use of digital filters and delay lines.

Nevertheless, modeling of this type of effects remain an active field, since musicians tend to prefer analog counterparts and current methods are often optimized to a very specific circuit. Therefore such models are not easily transferable to different effects units since expert knowledge of the type of circuit being modeled is always required and cannot be efficiently generalized to other time-varying or time-invariant audio effects with long-term memory.

Since the architectures from previous chapters do not generalize to transformations with long temporal dependencies, in this chapter we explore the capabilities of end-to-end DNNs to learn the long-term memory which characterizes these effect units. We build on the *CAFx* and *WaveNet* architectures and we propose two novel general-purpose modeling networks: *CRAFx* and *CWAFx*. Based on the adaptive front-end and back-end structures from previous models, we explore whether a latent-space based on Bidirectional Long Short-Term Memory (Bi-LSTM) layers or temporal dilated convolutions is able to learn time-varying transformations. To the best of our knowledge, these models represent the first DNNs for modelling time-varying audio effects with long temporal dependencies. Code is available online[1] and the number of parameters and computational complexity are shown in Appendix C. Audio samples for *CRAFx* can be found in Appendix B.

Therefore we introduce deep learning architectures for generic black-box modeling of audio processors with long-term memory. We show the models matching digital implementations of modulation based audio effects such as *chorus*, *flanger*, *phaser*, *tremolo*, *vibrato*, LFO-based *auto-wah*, *ring modulator* and *Leslie speaker*. Fur-

1 https://github.com/mchijmma/DL-AFx/tree/master/src

thermore, we extend the applications of the models by including nonlinear time-invariant audio effects with long temporal dependencies such as *auto-wah* with envelope follower, *compressor* and *multiband compressor*. We also introduce nonlinearities such as *overdrive* into linear time-varying effect units, in order to test the capabilities of the networks when modeling nonlinear time-varying audio transformations.

We explore linear and nonlinear time-varying emulation as a content-based transformation without explicitly obtaining the solution of the time-varying system. In order to measure the performance of the model, we propose an objective metric based on the psychoacoustics of modulation frequency perception. We also analyze what the model is actually learning and how the given task is accomplished.

6.1 CONVOLUTIONAL RECURRENT AUDIO EFFECTS MODELING NETWORK - CRAFX

The *CRAFx* model builds on the *CAFX* architecture and is also divided into three parts: adaptive front-end, latent-space and synthesis back-end. A block diagram can be seen in Fig. 6.1 and its structure is described in detail in Table 6.1. The main difference is the incorporation of Bi-LSTMs (see Section 3.6) into the latent-space and the modification of the synthesis back-end structure. We explore bidirectional recurrent layers as they can access long-term context from backward and forward directions (Graves et al., 2013), and Bi-LSTMs have been shown to be able to learn long time dependencies when processing time series where the input context is needed (Graves and Schmidhuber, 2005).

Thus, we incorporate Bi-LSTMs to allow the model to learn nonlinear transformations with long temporal dependencies, although further exploration of other gated recurrent layers such as GRUs is needed. Also, instead of 128 channels, due to the training time of the recurrent layers, this model uses a filter bank structure of 32 channels or filters. As mentioned in Section 6.1, all convolutional layers use strides of unit value, no dilation is incorporated and we follow the same padding as in *CAFx*.

In order to allow the model to learn long-term memory dependencies, the input consists of the audio frame x at the current time step t, concatenated with the k previous and k subsequent frames. These frames are of size N and sampled with a hop size τ. The concatenated input $\mathbf{x}$ is described as

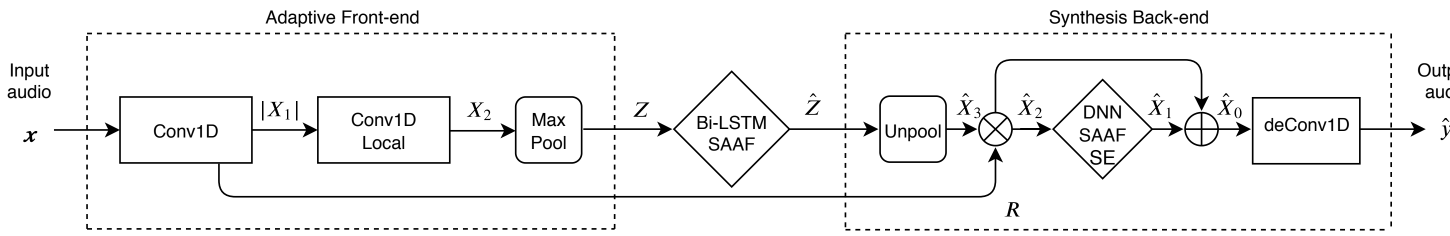


Figure 6.1: Block diagram of *CRAFx*; adaptive front-end, latent-space Bi-LSTM and synthesis back-end.

$$\mathbf{x}^{(j)} = x(t + j\tau), \; j = -k, ..., k. \tag{6.1}$$

The *adaptive front-end* is exactly the same as the one from *CAFx*, but its layers are time distributed, i.e. the same convolution or pooling operation is applied to each of the 2k+1 input frames. The *max-pooling* operation is a moving window of size N/64. In this model, $\mathbf{R}$ is the corresponding row in $\mathbf{X}_1$ for the frequency band decomposition of the current input frame $\mathbf{x}^{(0)}$. Thus, the back-end does not directly receive information from the past and subsequent context frames.

Latent-space Bi-LSTM

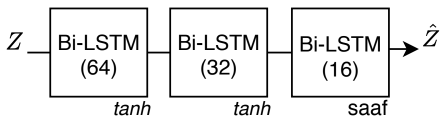


Figure 6.2: Block diagram of the latent-space of *CRAFx*.

The latent-space is depicted in Fig. 6.2 and consists of three Bi-LSTM layers of 64, 32, and 16 units respectively (see Section 3.6). The Bi-LSTMs process the latent-space representation $\mathbf{Z}$, which is learned by the front-end and contains information regarding the 2k+1 input frames. With an input frame size of 4096 samples and ±4 context frames, the latent representation $\mathbf{Z}$ from the front-end corresponds to 9 frames of 64 samples and 32 channels, which can be unrolled into a feature map of 64 samples and 288 channels. The input-to-hidden weights of the Bi-LSTM layers are applied to the channel dimension of the input feature maps. For instance, for the first Bi-LSTM layer, the matrix multiplication is com-

Table 6.1: Detailed architecture of *CRAFx* with input frame size of 4096 samples and ±4 context frames. Output shape = (k,m,n) denotes k frames of m columns and n rows, and Output shape = (m,n) denotes m columns and n rows. Weights = m(n) denotes m kernels of size n, and Weights = n denotes n hidden units.

Layer	Output shape	Weights	Output
Input	(9, 4096, 1)	.	x
Conv1D	(9, 4096, 32)	32(64)	X_1
Residual	(4096, 32)	.	R
Abs	(9, 4096, 32)	.	.
Conv1D-Local	(9, 4096, 32)	32(128)	X_2
MaxPooling	(9, 64, 32)	.	Z
Bi-LSTM	(64, 128)	64	.
Bi-LSTM	(64, 64)	32	.
Bi-LSTM	(64, 32)	16	.
SAAF	(64, 32)	32(25)	$\hat{Z}$
Unpooling	(4096, 32)	.	$\hat{X}_3$
Multiply	(4096, 32)	.	$\hat{X}_2$
Dense	(4096, 32)	32	.
Dense	(4096, 16)	16	.
Dense	(4096, 16)	16	.
Dense	(4096, 32)	32	.
SAAF	(4096, 32)	32(25)	$\hat{X}'_1$
Abs	(4096, 32)	.	.
Global Average	(1, 32)	.	.
Dense	(1, 512)	512	.
Dense	(1, 32)	32	se
$\hat{X}'_1 \times se$	(4096, 32)	.	$\hat{X}_1$
$\hat{X}_1 + \hat{X}_2$	(4096, 32)	.	$\hat{X}_0$
deConv1D	(4096, 1)	.	$\hat{y}$

puted between the input-to-hidden weights and the columns of the input matrix, which correspond to 64 samples and 288 channels.

These recurrent layers are trained to reduce the dimension of Z, while also learning a set of nonlinear modulators $\hat{Z}$. This new latent representation or modulators corresponds to a matrix of 32 channels of 64 samples and is fed into the synthesis back-end in order to reconstruct an audio signal that matches the time-varying

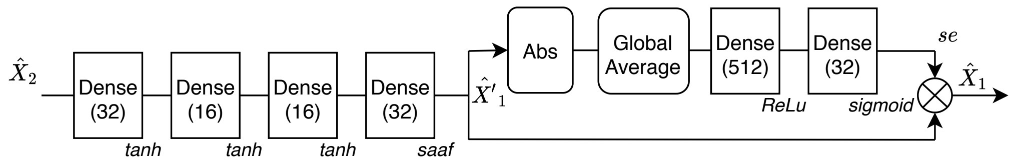


Figure 6.3: Block diagram of DNN-SAAF-SE.

modeling task. Each Bi-LSTM has dropout and recurrent dropout rates of 0.1 and the first two layers have tanh as activation function. Also, the nonlinearities of the last recurrent layer are locally connected SAAFs.

As shown in Section 5.1, locally connected SAAFs are used as the nonlinearity for the last layer. This in order to make use of the smooth characteristics of SAAFs, which can approximate any continuous function such as the modulators of the respective time-varying effect units. Each SAAF is composed of 25 intervals between -1 to $+1$.

Synthesis back-end

The synthesis back-end accomplishes the reconstruction of the target audio by processing the frequency band decomposition $\mathbf{R}$ and the nonlinear modulators $\hat{\mathbf{Z}}$. Similarly to *CAFx*, The back-end consists of an unpooling layer, a DNN-SAAF block and a final convolutional layer. The DNN-SAAF block consists of four dense layers of 32, 16, 16 and 32 hidden units respectively. Each dense layer is followed by the tanh function except for the last one, which is followed by a SAAF layer. The new structure of the back-end of *CRAFx* incorporates *Squeeze-and-Excitation* layers (see Section 3.7) after the DNN-SAAF block (DNN-SAAF-SE).

We propose a SE block which applies a dynamic gain to each of the feature map channels of $\hat{\mathbf{X}}'_1$, the output of DNN-SAAF. Based on the structure from Kim et al. (2018), the SE block consists of a global average pooling operation followed by two FC layers. The FC layers are followed by *ReLU* and *sigmoid* activation functions accordingly.

Since the feature maps within the back-end are based on time-domain waveforms, we incorporate an *absolute value* layer before the global average pooling operation. Fig. 6.3 depicts the block diagram of DNN-SAAF-SE, which input and output are the feature maps $\hat{\mathbf{X}}_2$ and $\hat{\mathbf{X}}_1$, respectively.

Following the filter bank architecture, the back-end matches the time-varying task by the following steps. First, an upsampling operation is applied to the learned modulators $\hat{\mathbf{Z}}$ which is followed by an element-wise multiplication with the residual connection $\mathbf{R}$. This can be seen as a frequency dependent amplitude modulation to each of the channels or frequency bands of $\mathbf{R}$:

$$\hat{X}_2 = \hat{X}_3 \times \mathbf{R}, \tag{6.2}$$

where $\hat{X}_3$ corresponds to a matrix containing the upsampled modulators $\hat{\mathbf{Z}}$. Both $\hat{X}_3$ and $\mathbf{R}$ feature maps have 32 rows or channels of 4096 samples.

This is followed by the nonlinear waveshaping and channel-wise scaled filters from the DNN-SAAF-SE block. Thus, the modulated frequency band decomposition $\hat{X}_2$ is processed by the learned waveshapers from the DNN-SAAF layers, resulting in the feature map $\hat{X}'_1$. This is further scaled by se, the frequency dependent gains from the SE layer. The resulting feature map $\hat{X}_1$ can be seen as modeling the nonlinear short-term memory transformations within the audio effects modeling tasks:

$$\hat{X}_1 = \hat{X}'_1 \times se, \tag{6.3}$$

where se corresponds to a vector of 32 scalars and $\hat{X}'_1$ is a matrix of 32 channels of 4096 samples.

Then, $\hat{X}_1$ is added back to $\hat{X}_2$, acting as a nonlinear feedforward delay line:

$$\hat{X}_0 = \hat{X}_1 + \hat{X}_2, \tag{6.4}$$

where $\hat{X}_0$, $\hat{X}_1$ and $\hat{X}_2$ correspond to matrices of 32 channels of 4096 samples.

Therefore the structure of the back-end is informed by the general architecture in which the modulation based effects are implemented in the digital domain, through the use of LFOs, digital filters and delay lines. By using this architecture, we interpret that $\hat{X}_2$ corresponds to the linear component with long temporal dependencies and $\hat{X}_1$ to the nonlinear component with short-term memory of time-varying audio effects. Nevertheless, a further exploration is required to determine if the model uses these feature maps exclusively in this way.

Finally, the complete waveform is synthesized in the same way as in *CAFx*, where the last layer corresponds to the transposed and non-trainable deconvolution operation.

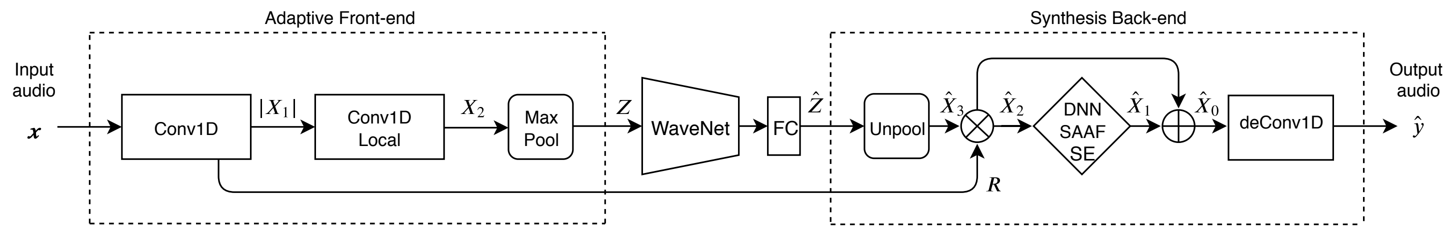


Figure 6.4: Block diagram of *CWAFx*; adaptive front-end, latent-space WaveNet and synthesis back-end.

Table 6.2: Detailed architecture of the latent-space WaveNet. This is for a *CWAFx* with input frame size of 4096 samples and ±4 context frames. Output shape = (m,n) denotes m columns and n rows. Weights = m(n) denotes m kernels of size n.

Layer - Output shape - Weights		Output	
Z (576, 32)		.	
Conv1D (576, 32) - 32(3)		$\mathbf{R}_{\text{in}}$	
Dilated conv (576, 32) - 32(3)	Dilated conv (576, 32) - 32(3)	.	
Tanh (576, 32)	Sigmoid (576, 32)	.	.
Multiply (576, 32)		.	
Conv1D (576, 32) - 32(1)	Conv1D (576, 32) - 32(1)	$\mathbf{R}_{\text{out}}$	**S**
Add (576, 32)		.	
ReLU (576, 32)		.	
Conv1D (576, 32) - 32(3)		.	
ReLU (576, 32)		.	
Conv1D (576, 32) - 32(3)		.	
FC (64, 32) - 64		$\hat{\mathbf{Z}}$	

6.2 CONVOLUTIONAL AND WAVENET AUDIO EFFECTS MODELING NETWORK - CWAFX

We propose a new model based on the combination of the convolutional and dense architectures from *CRAFx* with the dilated convolutions from *WaveNet*. Since the Bi-LSTM layers in the former were in charge of learning long temporal dependencies from the input and context audio frames, we replace these recurrent layers with temporal dilated convolutions. This is due to the fact that dilated convolutions have been shown to outperform recurrent approaches when learning sequential problems (Bai et al., 2018), such as in MatthewDavies and Böck (2019), where Bi-LSTMs are successfully replaced with this type of temporal convolutions.

Thus, we investigate whether a latent-space based on stacked dilated convolutions can learn frequency-dependent amplitude modulation signals. The model is depicted in Fig. 6.4. The *adaptive front-end* and *synthesis back-end* are the same as the ones presented in *CRAFx*.

Latent-space WaveNet

The structure of the latent-space WaveNet is described in detail in Table 6.2.

With *CWAFx* with input frame size of 4096 samples and ±4 context frames, the latent representation $\mathbf{Z}$ from the front-end corresponds to 9 frames of 64 samples and 32 channels, which can be unrolled into a feature map of 576 samples and 32 channels. Thus, we approximate these input dimensions with a latent-space WaveNet with receptive and target fields of 510 and 64 samples respectively. Thus, based on Eq. (5.2), we use 2 stacks of 7 dilated convolutional layers with a dilation factor of 1,2,...64 and 32 filters of size 3.

Also, we achieved better fitting by keeping the dimensions of the skip connections $\mathbf{S}$ and by replacing the final 1x1 convolution with a FC layer. The latter has 64 hidden units followed by the tanh activation function. The FC layer is applied along the latent dimension rather than to the channel dimension, i.e. the matrix multiplication is computed between the FC weights and the rows of the input matrix, which corresponds to 576 samples and 32 channels. The new latent representation $\hat{\mathbf{Z}}$ corresponds to a matrix of 32 channels of 64 samples

6.3 EXPERIMENTS

6.3.1 *Training*

The training of *CRAFx* and *CWAFx* includes the same initialization step as *CEQ* and *CAFx* (see Section 4.2.1). Once the convolutional layers of the front-end and back-end are pretrained, the DNN-SAAF-SE block and the latent-space Bi-LSTM and Wavenet layers are incorporated into the respective models, and all the weights are trained following an end-to-end supervised learning task.

The loss function to be minimized is the mean absolute error between the target and output waveforms, see Eq. (3.1). We explore input size frames from 1024 to

8192 samples and we always use a rectangular window with a hop size of 50%. The batch size consisted of the total number of frames per audio sample.

Adam (Kingma and Ba, 2015) is used as optimizer and we perform the pre-training for 200 epochs and the supervised training for 500 epochs. In order to speed convergence, during the second training step we start with a learning rate of $5 \cdot 10^{-5}$ and we reduce it by 50% every 150 epochs. We select the model with the lowest error for the validation subset.

6.3.2 *Dataset*

Modulation based audio effects such as *chorus*, *flanger*, *phaser*, *tremolo* and *vibrato* were obtained from the *IDMT-SMT-Audio-Effects* dataset (Stein et al., 2010). The recordings correspond to individual 2-second notes which include electric guitar and bass guitar raw notes and their respective effected versions. These effects correspond to digital implementations of effect units, such as VST audio plug-ins.

For our experiments, for each of the above effects, we only use the setting #2 from where we obtained the unprocessed and processed audio for bass guitar. In addition, processing the bass guitar raw audio, we implemented an LFO-based *auto-wah* with a peak filter whose center frequency ranges from 500 Hz to 3 kHz and modulated by a 5 Hz sinusoidal.

Since the previous audio effects are linear time-varying, we further test the capabilities of the model by adding a nonlinearity to each of these effects. Thus, using the bass guitar wet audio, we use *SoX*[2] to apply an *overdrive* (gain$= +10$ dB) after each modulation based effect.

We also use virtual analog implementations of a *ring modulator* and a *Leslie speaker* to process the electric guitar raw audio. The *ring modulator* implementation[3] is based on Parker (2011b) and we use a modulator signal of 5 Hz. The *Leslie speaker* implementation[4] is based on Smith et al. (2002) and we model each of the stereo channels.

Finally, we also investigate the capabilities of the model with nonlinear time-invariant audio effects with long temporal dependencies, such as *compressors* and *auto-wah* based on an envelope follower. We use the *compressor* and *multiband compressor* from *SoX*[5] to process the electric guitar raw audio.

2 `http://sox.sourceforge.net/`
3 `https://github.com/nrlakin/robot_voice/blob/master/robot.py`
4 `https://ccrma.stanford.edu/software/snd/snd/leslie.cms`
5 See footnote 2.

Similarly, we use an *auto-wah* implementation[6] with an envelope follower and a peak filter which center frequency modulates between 500 Hz to 3 kHz.

For each time-varying task we use 624 raw and effected notes and both the test and validation samples are randomly selected and correspond to 5% of this subset each. The recordings were downsampled to 16 kHz and amplitude normalization was applied with exception to the time-invariant audio effects. Table 8.3 shows the details of the settings for each audio effect.

6.3.3 *Evaluation*

Three metrics are used when testing the models with the various modeling tasks. As shown in Chapter 5, we use the energy-normalized mean absolute error (*mae*).

As an objective evaluation for the time-varying tasks, we propose an objective metric which mimics human perception of amplitude and frequency modulation. The so-called modulation spectrum uses time-frequency theory integrated with the psychoacoustics of modulation frequency perception, thus, providing long-term knowledge of temporal fluctuation patterns (Sukittanon et al., 2004). The modulation spectrum mean squared error (*ms_mse*) is based on the audio features from McDermott and Simoncelli (2011) and McKinney and Breebaart (2003) and is defined as follows:

- A Gammatone filter bank (see **??**) is applied to the target and output entire waveforms. In total we use 12 filters, with center frequencies spaced logarithmically from 26 Hz to 6950 Hz.
- The envelope of each filter output is calculated via the magnitude of the Hilbert transform (Hahn, 1996) and downsampled to 400 Hz.
- A Modulation filter bank is applied to each envelope. In total we use 12 filters, with center frequencies spaced logarithmically from 0.5 Hz to 100 Hz.
- The FFT is calculated for each modulation filter output of each Gammatone filter. The energy is summed across the Gammatone and Modulation filter banks and the *ms_mse* metric is the mean squared error of the logarithmic values of the FFT frequency bins.

The evaluation for the nonlinear time-invariant tasks (*compressor* and *multiband compressor*) corresponds to *mfcc_cosine*: the mean cosine distance of the MFCCs (see Section 5.3.3).

6 https://github.com/lucieperrotta/ASP

Table 6.3: Settings for each audio effect modeling task.

Fx	settings
chorus	**Heavy Chorus**: 'Delay Time - 12 ms', 'Rate - 0.501 Hz', 'Spread - Off', 'Depth - 70%', 'Mix - 1:1', 'Level - 0 dB'
flanger	**Standard Flange**: 'Time - 3.98 ms', 'Rate - 1.2 Hz', 'Sync - Off', 'Depth - 50%', 'Spread - Off', 'Feedback - 70%','Mix - 1:1', 'Level - 0 dB'
phaser	**Metallic Space**: 'Stages - 6', 'Upper - 1933 Hz', 'Rate - 0.24 Hz', 'Sync - Off', 'Depth - 100%', 'Spread - Off', 'Feedback - 0%','Mix - 1:1', 'Level - 0 dB'
tremolo	**Slow Pulsing**: 'Frequency - 2 Hz', 'Oscillator - Sine', 'Output - 100%', 'Level - 100%'
vibrato	**Slow Expressive Vibrato**: 'Delay Time - 6 ms' , 'Dry Mix - 0%' , 'Wet Mix - 100%' , 'Feedback - 0%' , 'Modulation Rate - 2 Hz' , 'Modulation Depth - 1.1 ms'
auto-wah	**Peak filter**: 'Oscillator - Sine', 'Modulation Rate - 5 Hz' , 'Filter - IIR 2nd order' , 'Q - 1' , 'Peak Height - 0.8' , 'Upper - 3000 Hz' , 'Lower - 500 Hz'
auto-wah envelope follower	**Peak filter**: 'Moving Average Width - 2000' , 'Filter - IIR 2nd order' , 'Q - 1' , 'Peak Height - 0.8' , 'Upper - 3000 Hz' , 'Lower - 500 Hz'
ring modulator	**Diode-based**: 'Diode Constants - 0.2, 0.4', 'Gain Modulator - 0.5' , 'Oscillator - Sine', 'Modulation Rate - 5 Hz'
leslie speaker	**Doppler Simulation and the Leslie**: 'Speed Source Listener - 3.33 ms' , 'Gain Output - 0.35' , 'Reverb Amount - 0.025' , 'Horn Angular Velocity - 1.0' , 'Baffle Angular Velocity - 1.0" , 'Horn Angle - 0.0' , 'Baffle Angle - 0.0' , 'Horn Radius - 0.18' , 'Baffle Radius - 0.19050', 'Cabinet Length - 0.71', 'Cabinet Width - 0.52'
compressor	**SoX**: 'Attack Time - 10 ms' , 'Release Time - 100 ms' , 'Knee - 1 dB' , 'Ratio - 4:1' , 'Threshold - -40 dB'
multiband compressor	**SoX**: 'Frequency bands - 2' , 'Crossover Frequency - 500 Hz' , 'Attack Time 1 - 5 ms' , 'Release Time 1 - 100 ms' , 'Knee 1 - 0 dB' , 'Ratio 1 - 3:1' , 'Threshold 1 - -30 dB' , 'Attack Time 2 - 625 µs' , 'Release Time 2 - 12.5 ms' , 'Knee 2 - 6 dB' , 'Ratio 2 - 6:1' , 'Threshold 2 - -60 dB'

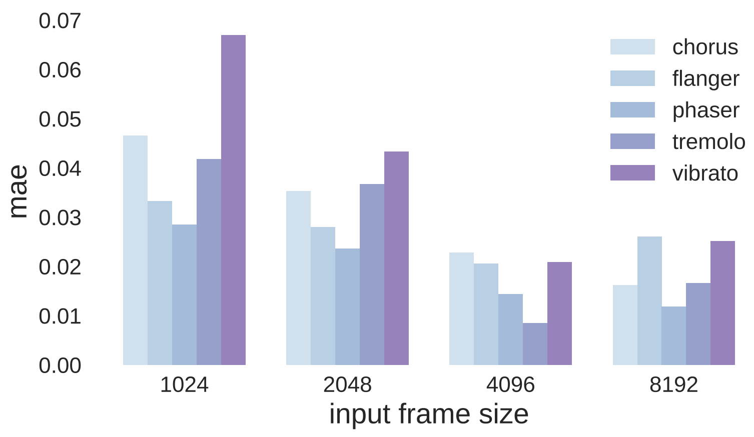


Figure 6.5: **mae** values for *CRAFx* when modeling linear time-varying tasks with different input size frames. Lower is better.

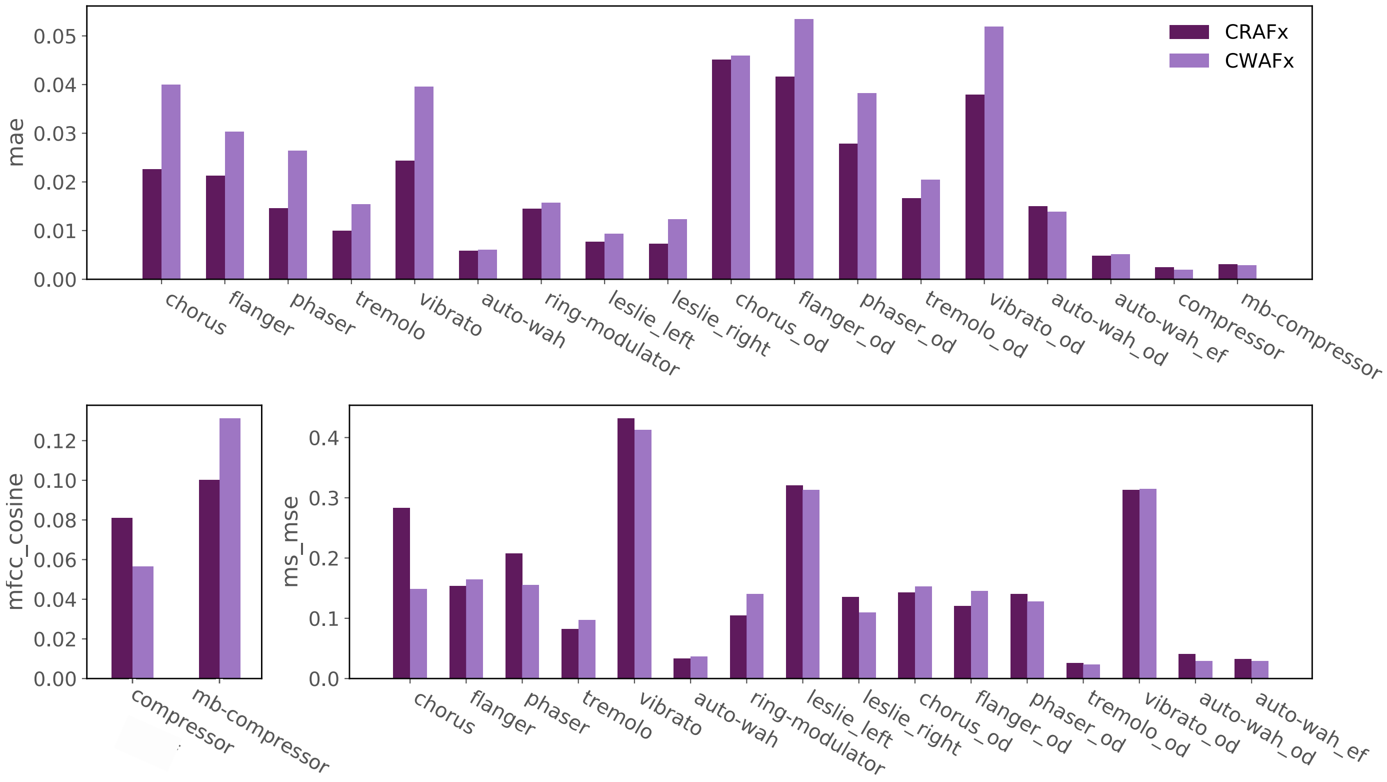


Figure 6.6: **mae**, **mfcc_cosine** and **ms_mse** values with the test dataset for all the modeling tasks. **od**, **ef** and **mb** mean *overdrive*, *envelope follower* and *multiband* respectively. Lower is better for all metrics.

6.4 RESULTS & ANALYSIS

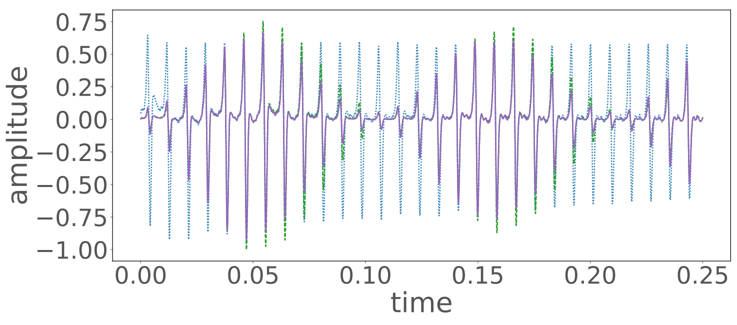
First, we explore the capabilities of Bi-LSTMs to learn long-term temporal dependencies. For *CRAFx*, Fig. 6.5 shows the *mae* results of the test dataset for different input frame sizes and various linear time-varying tasks. The results with the lowest *mae* are the ones with an input size of 4096 samples, since shorter frame sizes result in a higher error and 8192 samples do not yield a significant improvement. Since the average modulation frequency in our tasks is 2 Hz, for each input size we select a k that covers one period of this modulator signal. Thus, for the rest of our experiments, we use an input size of 4096 samples and $k = 4$ for the number of past and subsequent frames.

For each model, the training procedures were performed for each type of time-varying and time-invariant audio effect. Then, the models were tested with samples from the test dataset. Audio examples for *CRAFx* are available online[7]. Fig. 6.6 shows the different objective metrics for all the test subsets. To provide a reference, the mean *mae* and *ms_mse* and values between input and target waveforms are 0.13, 0.83 respectively. For the compressor and multiband compressor, the mean *mfcc_cosine* value is 0.15.

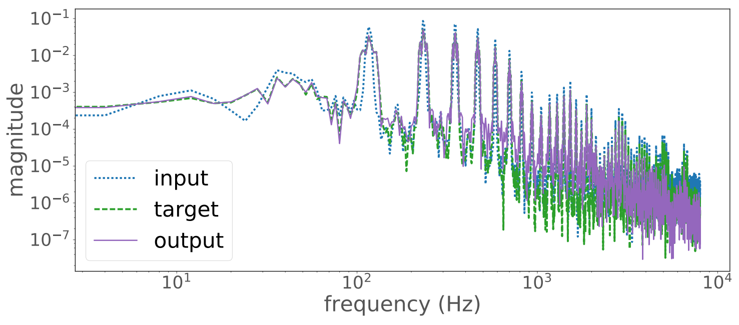
It can be seen that the models achieved low-error metrics in each linear time-varying modeling task, and higher errors when modeling effects with added nonlinearities. Overall, the models achieved better results with amplitude modulation and time-varying filter audio effects, although the modeling of delay-line based effects also yielded low-error results. Based on the perceptual-based objective metrics, *CRAFx* and *CWAFx* closely match each audio effect modeling task with great similarity. On the other hand, based on the *mae* results, the waveform of the modeling targets is approximated with lower errors by *CRAFx*.

Fig. 6.7 displays the functioning of *CRAFx* for the *tremolo* task, where the selected rows in Figs. 6.7c to 6.7f were chosen in order to illustrate how the model accomplishes the amplitude modulation task. It shows how the model processes the input frame $x^{(0)}$ into the feature maps X_1 and X_2, learns a set of modulator signals $\hat{Z}$, and applies the respective amplitude modulation. This linear time-varying audio effect is easy to interpret. For more complex nonlinear time-varying effects, a more in-depth analysis of the model is required.

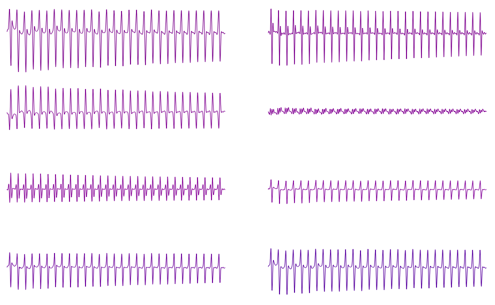
7 https://mchijmma.github.io/modeling-time-varying/



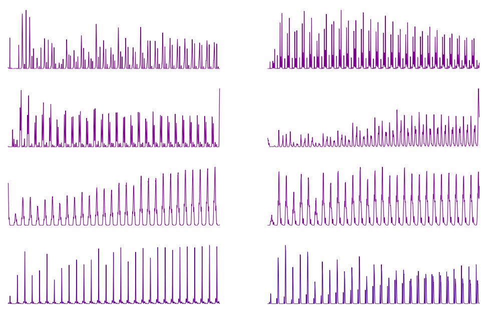
(a)



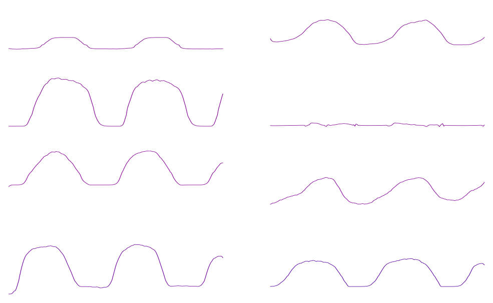
(b)



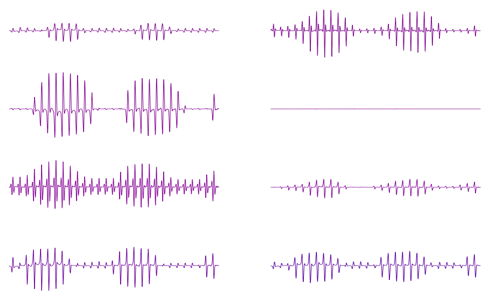
(c)



(d)

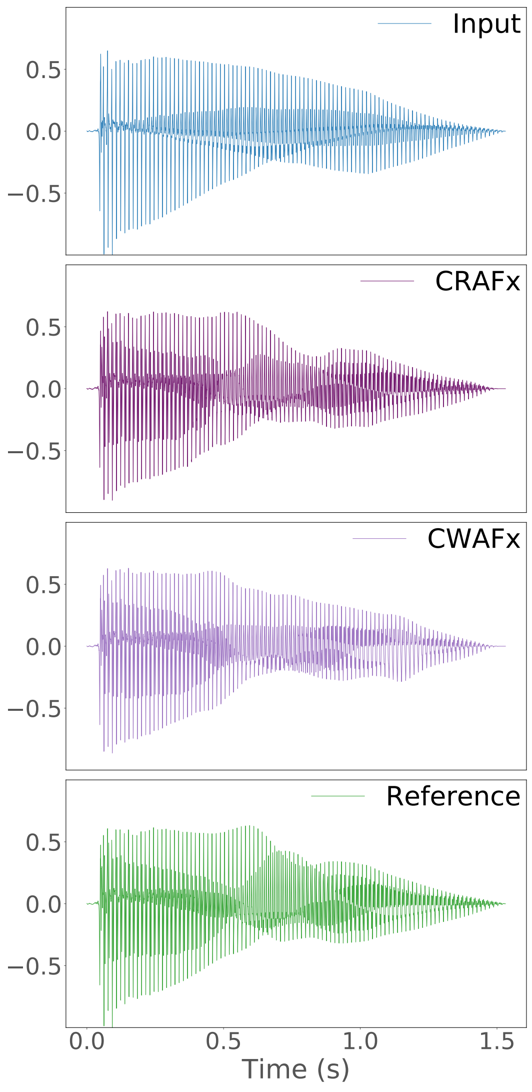


(e)

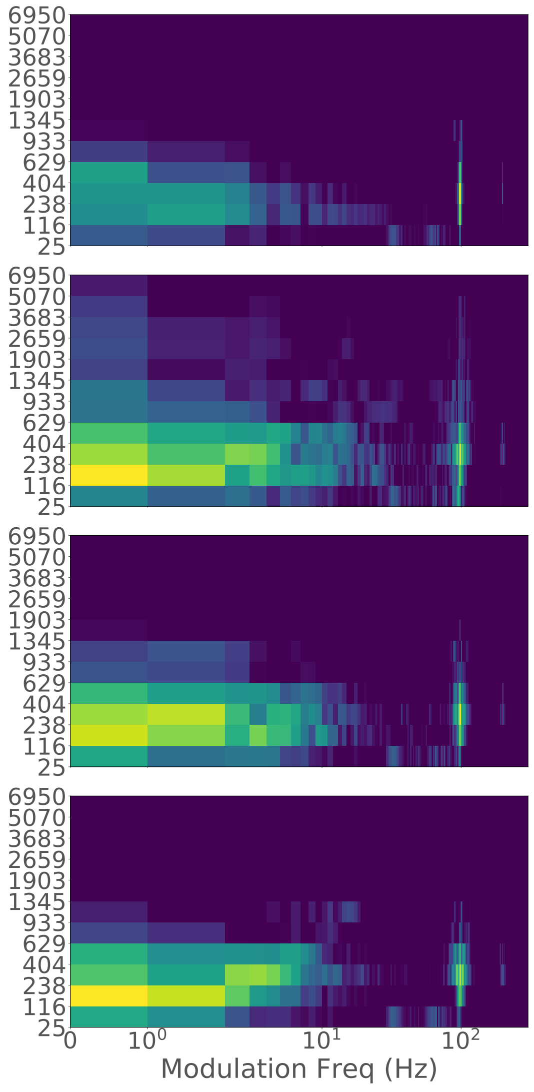


(f)

Figure 6.7: For *CRAFx*, various internal plots for the test dataset of the **tremolo** modeling task. Fig. 6.7a input, target and output frames of 4096 samples and their respective FFT magnitudes. Fig. 6.7c, for the input frame $\mathbf{x}^{(0)}$, 8 selected rows from $\mathbf{R}$. Fig. 6.7d, following the filter bank architecture, respective 8 rows from $\mathbf{X}_2$. Fig. 6.7e, from $\hat{\mathbf{Z}}$, corresponding 8 modulator signals learned by the Bi-LSTM layer. Fig. 6.7f, in the same manner, 8 rows from $\hat{\mathbf{X}}_0$, which is the input to the deconvolution layer prior to obtaining the output frame $\hat{\mathrm{y}}$. Vertical axes in Figs. 6.7c to 6.7f are unitless and horizontal axes correspond to time.



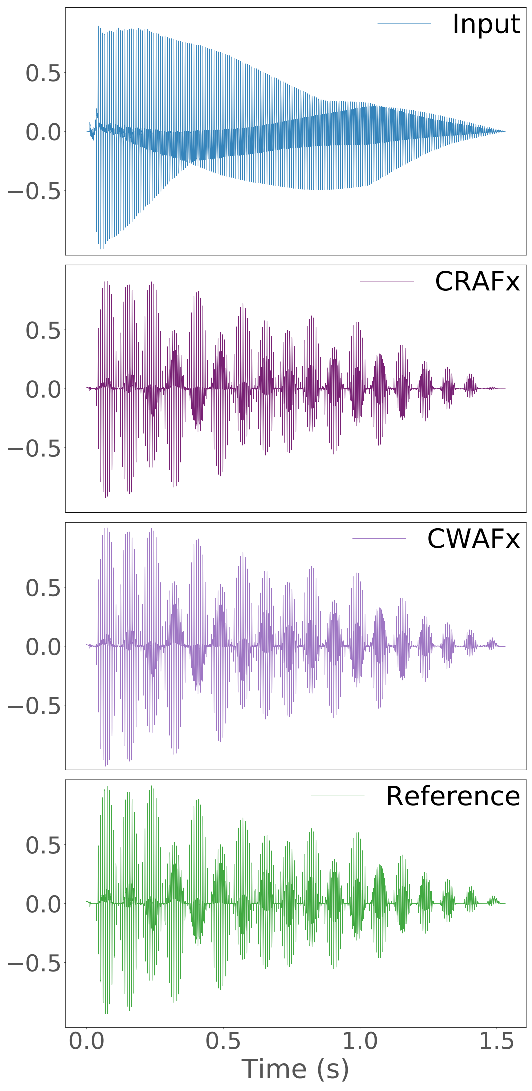
(a)

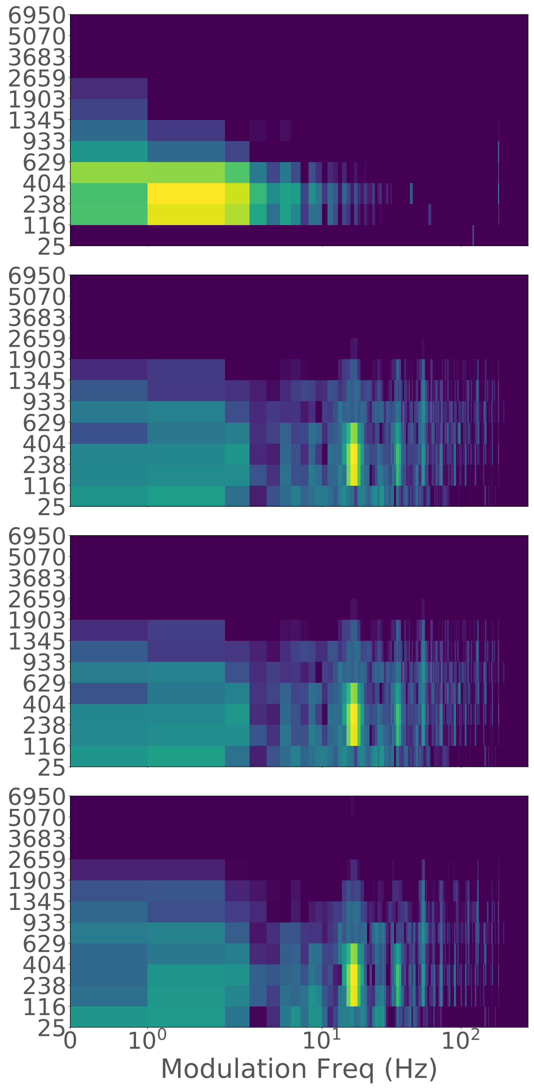


(b)

Figure 6.8: Results with selected samples from the test dataset for the **chorus** task. Figs. 6.8a and 6.8b show the waveforms and their respective modulation spectra. Vertical axes represent amplitude and Gammatone center frequency (Hz) respectively.

For selected linear and nonlinear time-varying tasks, Figs. 6.8 to 6.11 show the input, target, and output waveforms together with their respective modulation spectra. In the time-domain, it is evident that the models are matching the target waveform in a similar manner. From the modulation spectrum it is noticeable that the models equally introduce different modulation energies into the output which



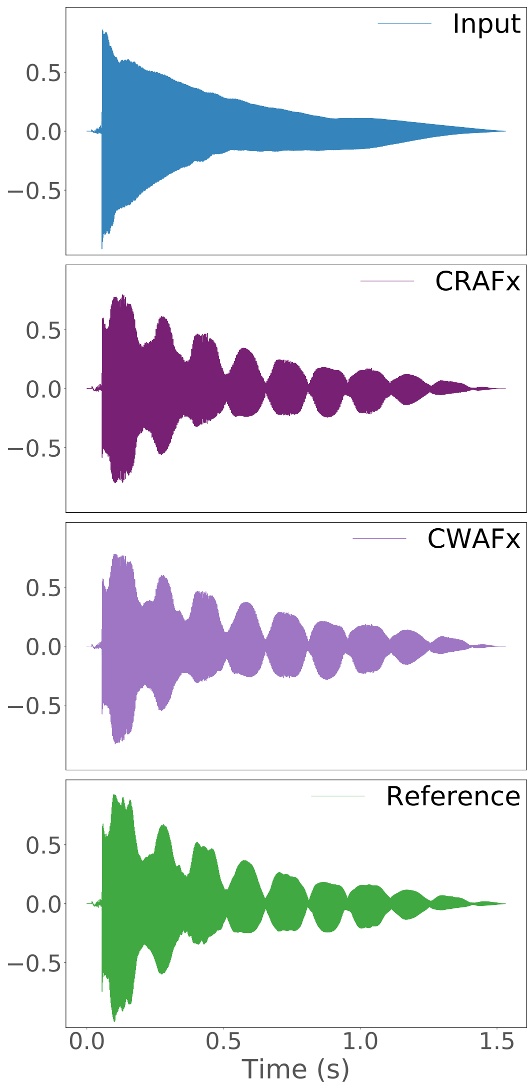


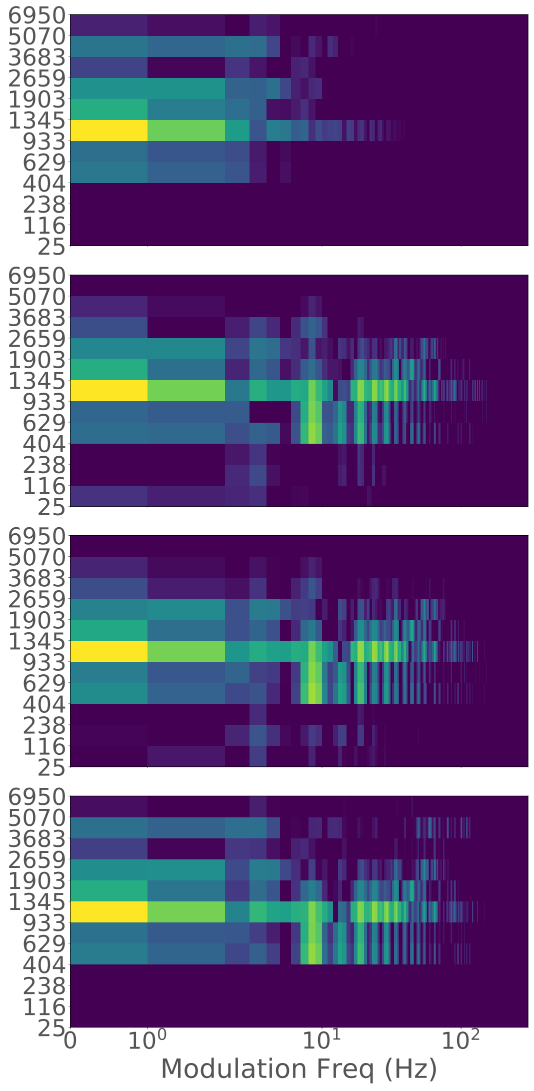
(a) (b)

Figure 6.9: Results with selected samples from the test dataset for the **ring modulator** task. Figs. 6.9a and 6.9b show the waveforms and their respective modulation spectra. Vertical axes represent amplitude and Gammatone center frequency (Hz) respectively.

were not present in the input and which closely match those of the respective targets.

The task becomes more challenging when a nonlinearity is added to a linear time-varying transformation. Fig. 6.11 depicts results for the *phaser-overdrive* task. Given the large overdrive gain, the resulting audio has a lower-frequency modulation. It can be seen that the models introduce modulations as low as 0.5 Hz.



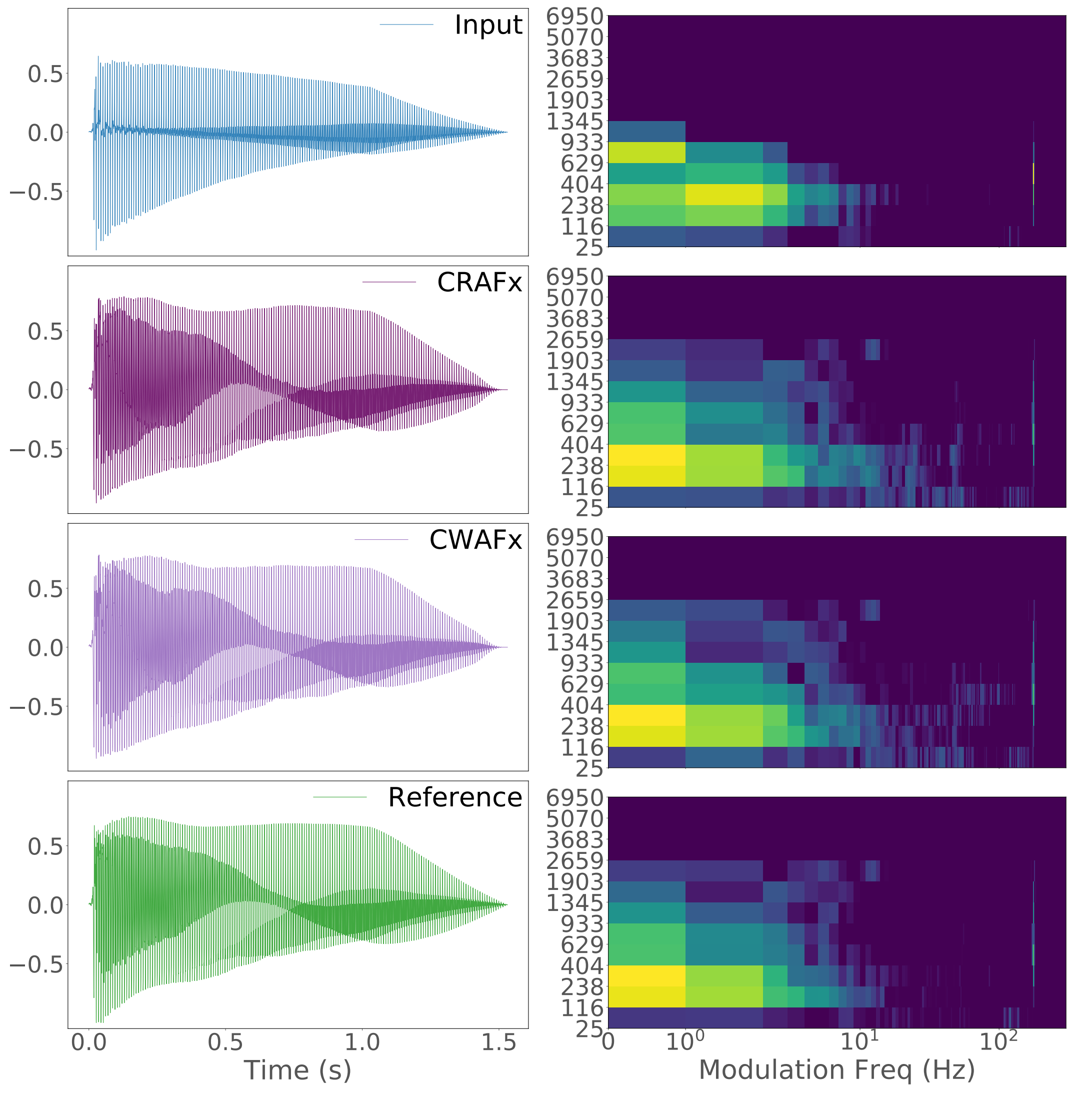


(a) (b)

Figure 6.10: Results with selected samples from the test dataset for the **Leslie speaker** task (right channel). Figs. 6.10a and 6.10b show the waveforms and their respective modulation spectra. Vertical axes represent amplitude and Gammatone center frequency (Hz) respectively.

But the waveforms are not as smooth as the target, hence the larger *mae* values in Fig. 6.6. Although the *mae* increases for modeling tasks that include the *overdrive* nonlinearity, the models do not significantly reduce performance and are able to match the combination of nonlinear and modulation based audio effects.

Much more complicated time-varying tasks, such as the *ring modulator* and *Leslie speaker* virtual analog implementations were also modeled with low-error metrics.



(a) (b)

Figure 6.11: Results with selected samples from the test dataset for the **phaser-overdrive** task. Figs. 6.11a and 6.11b show the waveforms and their respective modulation spectra. Vertical axes represent amplitude and Gammatone center frequency (Hz) respectively.

This represents a significant result, since these implementations include emulation of the modulation introduced by nonlinear circuitry; as in the case of the *ring modulator*, or varying delay lines together with artificial reverberation and Doppler effect simulation; as in the *Leslie speaker* implementation.

The models are also able to perform linear and nonlinear time-invariant modeling. From waveform inspection, the long temporal dependencies of an envelope driven *auto-wah*, *compressor* and *multiband compressor* are matched by the networks. From Fig. 6.6, the *mfcc_cosine* values for the *multiband compressor* task indicate a lesser fitting, nevertheless the low *mae* values represent that the models are performing well when modeling these type of processors. Furthermore, in this latter case, the crossover filters denote a more complicated task which is being correctly modeled. More specific perception-based metrics for compressors can be further investigated, as well as subjective tests.

Overall, based on low-error objective metrics, the models performed better when modeling effect units based on amplitude modulation, such as *tremolo* or *ring modulator*, and time-varying filters, such as *phaser*. Likewise, similar performance was achieved for delay-line effects based on frequency modulation as in the case of *flanger* or the *Leslie speaker* stereo channels. Nevertheless, *vibrato* and *vibrato-overdrive* represent the modeling tasks with highest errors. This might be because *vibrato* is an effect based solely on frequency modulation whose rate is around 2 Hz. Since this represents a modulation rate higher than the rotation horn of the *Leslie speaker*, this indicates that the performance of the models decreases when matching effects based on low-frequency modulation such as the slow rotating setting of the *Leslie speaker* (see Chapter 7). This could be improved by increasing the frequency resolution by introducing more filters or channels, e.g. a filter bank architecture of 128 filters, or by increasing the size of the latent-space through smaller max pooling.

6.5 conclusion

In this chapter, we introduced *CRAFx* and *CWAFx*, two general-purpose deep learning architectures for modeling audio effects with long temporal dependencies. Through these two architectures, we explored the capabilities of end-to-end DNNs with Bi-LSTM layers and temporal dilated convolutions to learn long temporal dependencies such as low-frequency modulations and to process the audio accordingly. We can conclude that both models achieved similar performance in terms of objective metrics and were able to match digital implementations of linear and nonlinear time-varying audio effects, time-varying and time-invariant audio effects with long-term memory.

Based on the *mae*, *CRAFx* accomplished a closer match of the target waveforms. Nevertheless, both models performed equally well when tested with perceptual-based metrics such as *mfcc_cosine* and *ms_mse*. It is worth to mention that the computational processing times on GPU are significantly lower for *CWAFx* (see Appendix C). This is due to GPU-accelerated libraries such as *cuDNN* (Chetlur et al., 2014), which are highly optimized for convolutional layers.

In both architectures, we incorporated SE layers in order to learn and apply a dynamic gain to each of the feature map channels or frequency band decompositions. This allowed the models to apply the respective modulator signals to each channel and then further scale them through the SE layers. The introduction of this dynamic gain provided a better fitting when modeling the various time-varying tasks.

Other white-box or gray-box modeling methods suitable for these time-varying tasks would require expert knowledge such as specific circuit analysis and discretization techniques. Moreover, these methods cannot easily be extended to other time-varying tasks, and assumptions are often made regarding the nonlinear behavior of certain components. To the best of our knowledge, this work represents the first architectures for black-box modeling of linear and nonlinear, time-varying and time-invariant audio effects.

Using a small amount of training examples we showed the model matching *chorus*, *flanger*, *phaser*, *tremolo*, *vibrato*, LFO-based and envelope follower-based *auto-wah*, *ring modulator*, *Leslie speaker* and *compressors*. We proposed *ms_mse*, an objective perceptual metric to measure the performance of the model. The metric is based on the modulation spectrum of a Gammatone filter bank, thus measuring the human perception of amplitude and frequency modulation.

We demonstrated that the models process the input audio by applying different modulations which closely match with those of the time-varying target. Perceptually, most output waveforms are indistinguishable from their target counterparts, although there are minor discrepancies at the highest frequencies and noise level. This could be improved by using more convolution filters, as in *CAFx*, which means a higher resolution in the filter bank structures. Moreover, as shown in Chapter 4, a loss function based on time and frequency can be used to improve this frequency related issue, though listening tests may be required (see Chapter 7).

The generalization can also be studied more thoroughly, since the models learn to apply the specific transformation to the audio of a specific musical instrument, such as the electric guitar or the bass guitar. In addition, considering that the models use 256 ms input and output frames and strive to learn long temporal dependencies with shorter input frames, more research is needed on how to adapt these architectures to low-latency and high-resolution audio implementations.

Real-time applications would benefit significantly from the exploration of RNNs or temporal dilated convolutions to model transformations that involve long-term memory without resorting to large input frame sizes and the need for past and future context frames. Therefore, uni-directional LSTMs can be used to explore causal models where only past context frames are needed. The latter could improve the inherent latency of these architectures and the open the way to higher sampling rates, such as 44.1 kHz.

Although the models were able to match the artificial reverberation of the *Leslie speaker* implementation, a thorough exploration of reverb modeling is needed, such as plate, spring or convolution reverberation (see Chapter 8). In addition, since the models are learning a static representation of the audio effect, ways of devising a parametric model could also be explored. Finally, applications beyond virtual analog can be investigated, for example, in the field of automatic mixing the model could be trained to learn a generalization from mixing practices.

7 VIRTUAL ANALOG EXPERIMENTS

The previous chapters have focused on modeling several linear and nonlinear time-varying and time-invariant digital implementations or approximations of analog effect units. Furthermore, hitherto we have only evaluated the models with objective metrics. Thus, in this and the following chapters, we extend the evaluation of previous architectures by including perceptual listening tests and by modeling various analog audio effects.

Virtual analog modeling of audio effects consists of emulating the sound of an analog audio processor reference device. Therefore, we show virtual analog models of nonlinear effects, such as the *Universal Audio vacuum-tube preamplifier 610-B*; nonlinear effects with long-term memory, such as the *Universal Audio transistor-based limiter amplifier 1176LN*; and electromechanical nonlinear time-varying processors, such as the rotating *horn* and rotating *woofer* of a *145 Leslie speaker cabinet*. Audio samples can be found in Appendix B.

Through objective perceptual-based metrics and subjective listening tests we explore the performance of each of the architectures from Chapters 5 and 6: *CAFx*, *WaveNet*, *CRAFx* and *CWAFx*, when modeling these analog processors. We perform a systematic comparison between these architectures and we report that *CAFx* and *WaveNet* perform similarly when modeling nonlinear audio effects without memory and with long temporal dependencies, but fail to model time-varying tasks such as the *Leslie speaker*. On the other hand, and across all tasks, the models that incorporate latent-space RNNs or latent-space temporal dilated convolutions to explicitly learn long temporal dependencies, such as *CRAFx* and *CWAFx*, tend to outperform objectively and subjectively the rest of the models.

7.1 EXPERIMENTS

7.1.1 *Models*

For the experiments of this chapter we use the *CAFx*, *WaveNet*, *CRAFx* and *CWAFx* architectures. In order to provide a fairer comparison, *CAFx* and *WaveNet* are adapted to process input frames of size 4096 and sampled with a hop size of 2048 samples. *CRAFx* and *CWAFx* are used exactly as described in Sections 6.1 and 6.2, respectively.

The main modification to *CAFx* is in the adaptive front-end where we increase the *max-pooling* layer to a moving window of size 64. The rest of the model is as depicted in Section 5.1.

With regards to *WaveNet*, we extend the model to 2 stacks of 8 dilated convolutional layers with a dilation factor of 1,2,...,128. Based on Eq. (5.2), the receptive field of this architecture is of 1021 samples. The target field is 4096 samples, thus the input frame presented to the model consists of sliding windows of 5116 samples (see Eq. (5.3)). The rest of the architecture is as presented in Section 5.2.

Code is available online[1]. Also, Appendix C shows the number of parameters and processing times across all models.

7.1.2 *Training*

As mentioned in previous chapters, the training of the *CAFX*, *CRAFx* and *CWAFx* architectures includes an initialization step (see Section 4.2.1). Once the front-end and back-end are pretrained, the rest of the convolutional, recurrent, dense and activation layers are incorporated into the respective models, and all the weights are trained following an end-to-end supervised learning task. The *WaveNet* model is trained directly following this second step.

The loss function to be minimized is the mean absolute error and *Adam* (Kingma and Ba, 2015) is used as optimizer. For these experiments and for each model, we carried out the same supervised learning training procedure.

We use an early stopping patience of 25 epochs, i.e. training stops if there is no improvement in the validation loss. The model is fine-tuned further with the learning rate reduced by a factor of 4 and also a patience of 25 epochs. The initial learning rate is $1e{-4}$ and the batch size consists of the total number of frames per audio sample. On average, the total number of epochs is approximately 750.

We select the model with the lowest error for the validation subset (see Section 7.1.3). For the *Leslie speaker* modeling tasks, the early stopping and model selection procedures were based on the training loss. This is explained in more detail in Section 7.3.

1 https://github.com/mchijmma/DL-AFx/tree/master/src

Table 7.1: Settings for each analog audio effect modeling task.

Fx	settings
preamp	'Gain - +10 dB', 'Level - 6', 'Impedance - Line', 'High Boost/Cut - 0 dB', 'Low Boost/Cut - 0 dB'
limiter	'Attack Time - 800 µs', 'Release Time - 1100 ms', 'Input Level - 4', 'Output Level - 7', 'Ratio - *ALL*'
Leslie speaker	**Tremolo**: 'Rotation Speed - Fast' - **Chorale**: 'Rotation Speed - Slow'

7.1.3 *Dataset*

Raw recordings of individual 2-second notes of various 6-string electric guitars and 4-string bass guitars are obtained from the *IDMT-SMT-Audio-Effects* dataset (Stein et al., 2010). We use the 1250 unprocessed recordings of electric guitar and bass to obtain the wet samples of the respective audio effects modeling tasks. The raw recordings are amplitude normalized and for each task the test and validation samples are randomly selected and correspond to 5% of this dataset each. After the analog audio processors were sampled with the raw notes, all the recordings were downsampled to 16 kHz. The dataset is available online[2].

Universal Audio vacuum-tube preamplifier 610-B

This microphone tube preamplifier (*preamp*) is sampled from a *6176 Vintage Channel Strip* unit. In order to obtain an output signal with high harmonic distortion, the *preamp* is overdriven with the settings from Table 7.1.

Universal Audio transistor-based limiter amplifier 1176LN

Similarly, the wildly used field-effect transistor *limiter 1176LN* is sampled from the same *6176 Vintage Channel Strip* unit. The *limiter* samples are recorded with with the settings from Table 7.1. We use the slowest attack and release settings in order to further test the long-term memory of the models. The compression ratio value of *ALL* corresponds to all the ratio buttons of an original 1176 being pushed simultaneously. Thus, this setting also introduces distortion due to the variation of attack and release times.

2 https://zenodo.org/record/3562442

145 Leslie speaker cabinet

The output samples from the rotating *horn* and *woofer* of a *145 Leslie speaker* cabinet are recorded with an *AKG-C451-B* microphone. Each recording is done in mono by placing the condenser microphone perpendicularly to the *horn* or *woofer* and 1 meter away. Two speeds are recorded for each rotating speaker; *tremolo* for a fast rotation and *chorale* for a slow rotation. The rotation frequency of the *horn* is approximately 7 Hz and 0.8 Hz for the *tremolo* and *chorale* settings respectively, while the *woofer* has slower speed rotations (Herrera et al., 2009).

Since the *horn* and *woofer* speakers are preceded by a 800 Hz crossover filter, we apply a highpass FIR filter with the same cutoff frequency to the raw notes of the electric guitar and use only these samples as input for the *horn* speaker. Likewise, for the *woofer* speaker we use a lowpass FIR filter to preprocess the raw bass notes. The audio output of both speakers is filtered with the respective FIR filters. This in order to reduce mechanical and electrical noise and also to focus the modeling tasks on the amplitude and frequency modulations. Also, the recordings are amplitude normalized.

7.1.4 *Objective Metrics*

Three metrics are used when testing the models with the various modeling tasks; *mae*, the energy-normalized mean absolute error; *mfcc_cosine*, the mean cosine distance of the MFCCs (see Section 5.3.3); and *ms_mse*, the modulation spectrum mean squared error (see Section 6.3.3).

7.1.5 *Listening Test*

Thirty participants between the ages of 23 and 46 took part in the experiment[3] which was conducted at a professional listening room at Queen Mary University of London. The *Web Audio Evaluation Tool* (Jillings et al., 2015) was used to set up the test and participants used *Beyerdynamic DT-770 PRO* studio headphones.

The subjects were among musicians, sound engineers or experienced in critical listening. The listening samples were obtained from the test subsets and each page of the test contained a reference sound, i.e. a recording from the original analog device. The aim of the test was to identify which sound is closer to the reference,

3 The Queen Mary Ethics of Research Committee approved the listening test with reference number QMREC2165.

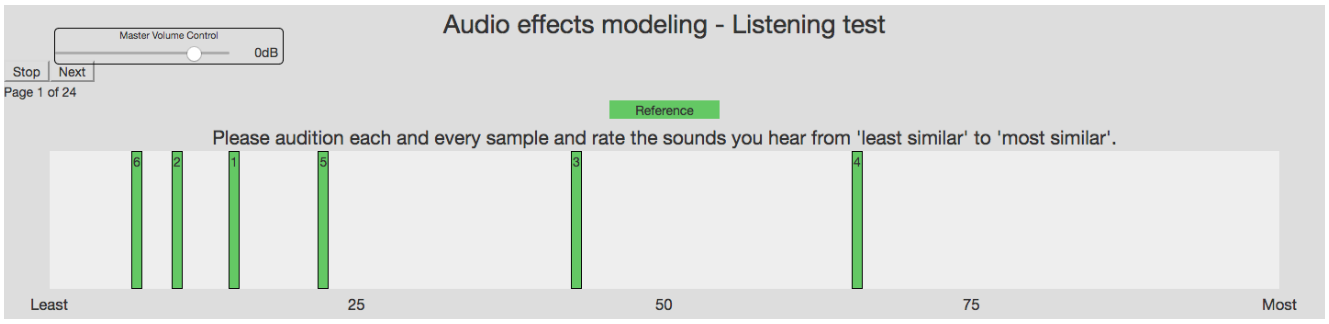

Audio effects modeling - Listening test
Master Volume Control
0dB
Stop
Next
Page 1 of 24
Reference
Please audition each and every sample and rate the sounds you hear from 'least similar' to 'most similar'.
6
2
1
5
3
4
Least
25
50
75
Most

Figure 7.1: User interface used by the participants for the listening test.

and participants rated 6 different samples according to the similarity of these in relation to the reference sound.

Therefore, participants were informed what modeling task they were listening to, and were asked to rate the samples from 'least similar' to 'most similar'. This in a scale of 0 to 100, which was then mapped into a scale of 0 to 1. The samples consisted of a dry sample as anchor, outputs from the 4 different models and a hidden copy of the reference. The test is based on the MUSHRA method (Union, 2003) and a screenshot of the user interface is shown in Fig. 7.1.

7.2 RESULTS

The training procedures were performed for each architecture and each modeling task: *preamp* corresponds to the vacuum-tube preamplifier, *limiter* to the transistor-based limiter amplifier, *horn tremolo* and *horn chorale* to the *Leslie speaker* rotating horn at fast and slow speeds respectively, and *woofer tremolo* and *woofer chorale* to the rotating woofer at the corresponding speeds. Then, the models were tested with samples from the test subset and the audio results are available online[4].

Fig. 7.2 shows the *mae*, *mfcc_cosine* and *ms_mse* for all the test subsets. It can be seen that the *mae* models' performance is similar within each modeling tasks with *limiter* having the lowest error. Also, *CAFx* presents the largest errors, with the *Leslie speaker chorale* settings being the highest.

In terms of perceptually-based metrics such as the *mfcc_cosine* and *ms_mse*, the *CRAFx* and *CWAFx* models achieved the best scores. This with the exception of the *woofer chorale* task, where the *CWAFx* model did not manage to accomplish the task. Overall, *CRAFx* and *CAFx* correspond to the highest and lowest scoring models respectively.

4 https://mchijmma.github.io/DL-AFx/

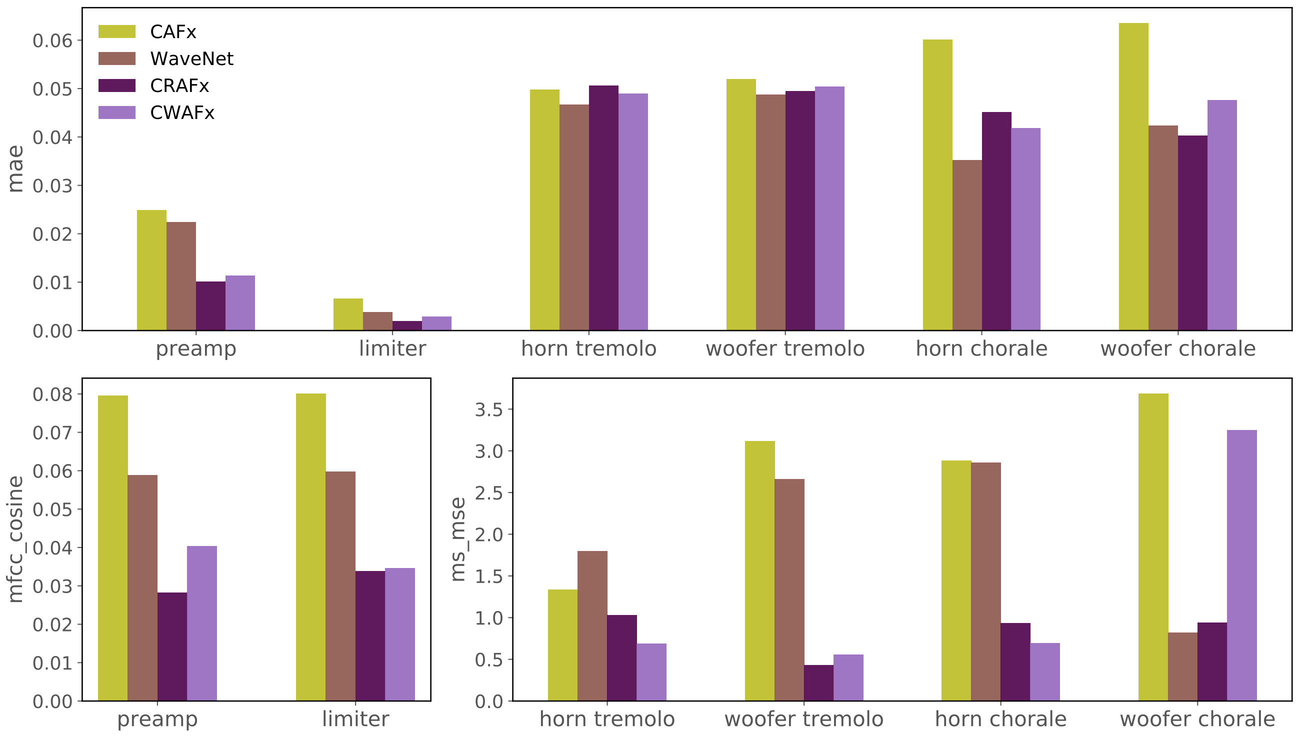
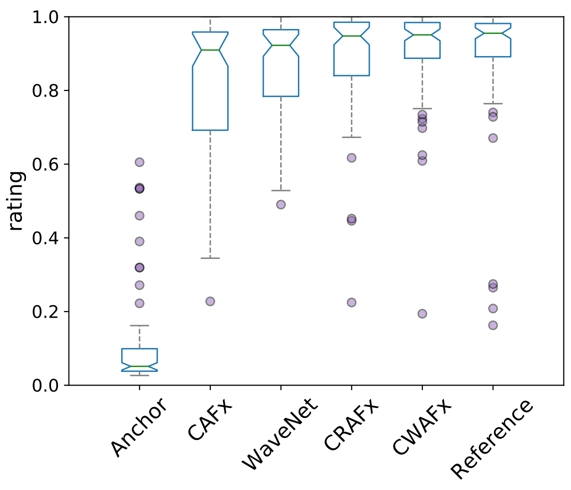


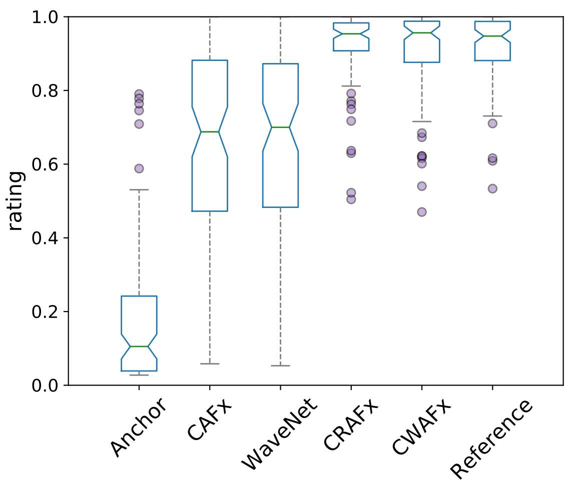
Figure 7.2: **mae**, **mfcc_cosine** and **ms_mse** values with the test dataset for all modeling tasks. Lower is better for all metrics.

The results of the listening test for all modeling tasks can be seen in Fig. 7.3 as notched box plots. The end of the notches represents a 95% confidence interval and the end of the boxes represent the first and third quartiles. Also, the green lines illustrate the median rating and the purple circles represent outliers. In general, both anchors and hidden references have the lowest and highest median respectively. Listeners who did not rate these samples as expected were not eliminated from the presented results, as they represent outliers; nevertheless, additional analysis and post-experiment selection of subjects may be required.

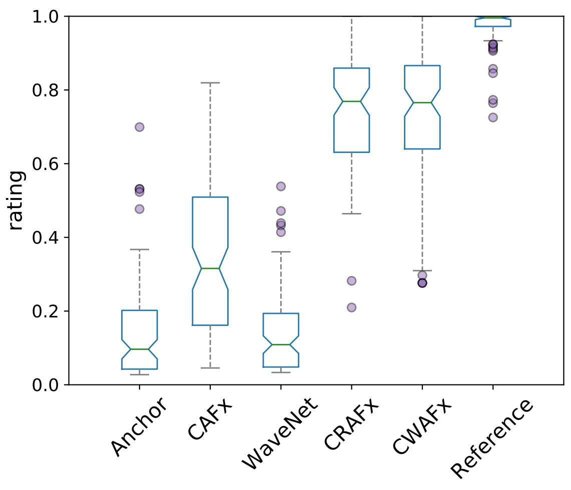
The perceptual findings match closely the objective metrics from Fig. 7.2, since the architectures that explicitly learn long-temporal dependencies, such as *CRAFx* and *CWAFx* outperform the rest of the models. Furthermore, for the *woofer chorale* task, the high-error performance of the latter is also evidenced in perceptual ratings. This indicates that the latent-space WaveNet fails to learn low-frequency modulations such as the *woofer chorale* rotating rate.



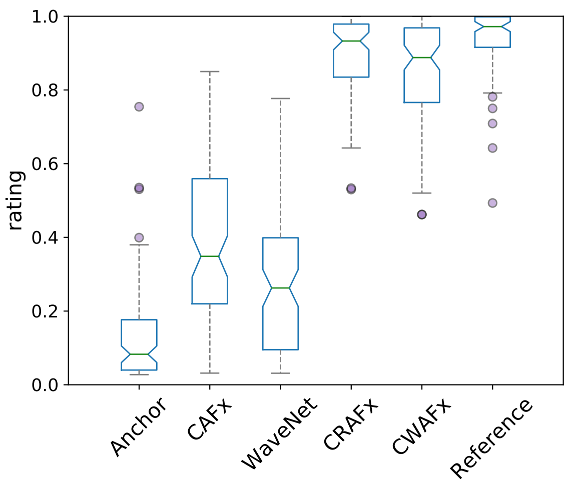
(a)



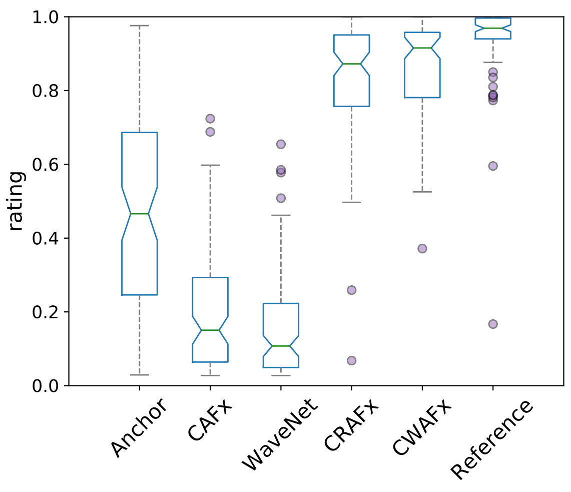
(b)



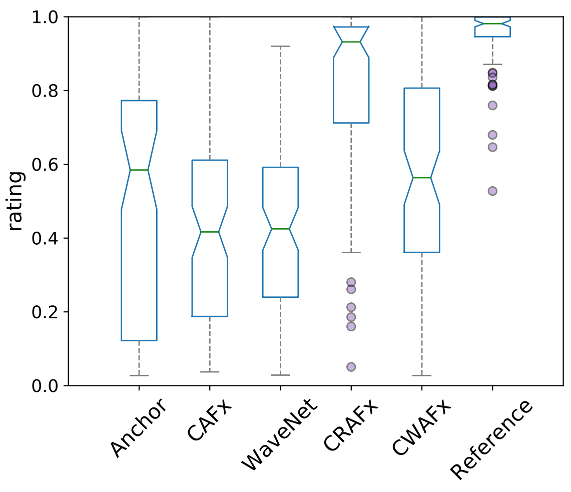
(c)



(d)



(e)

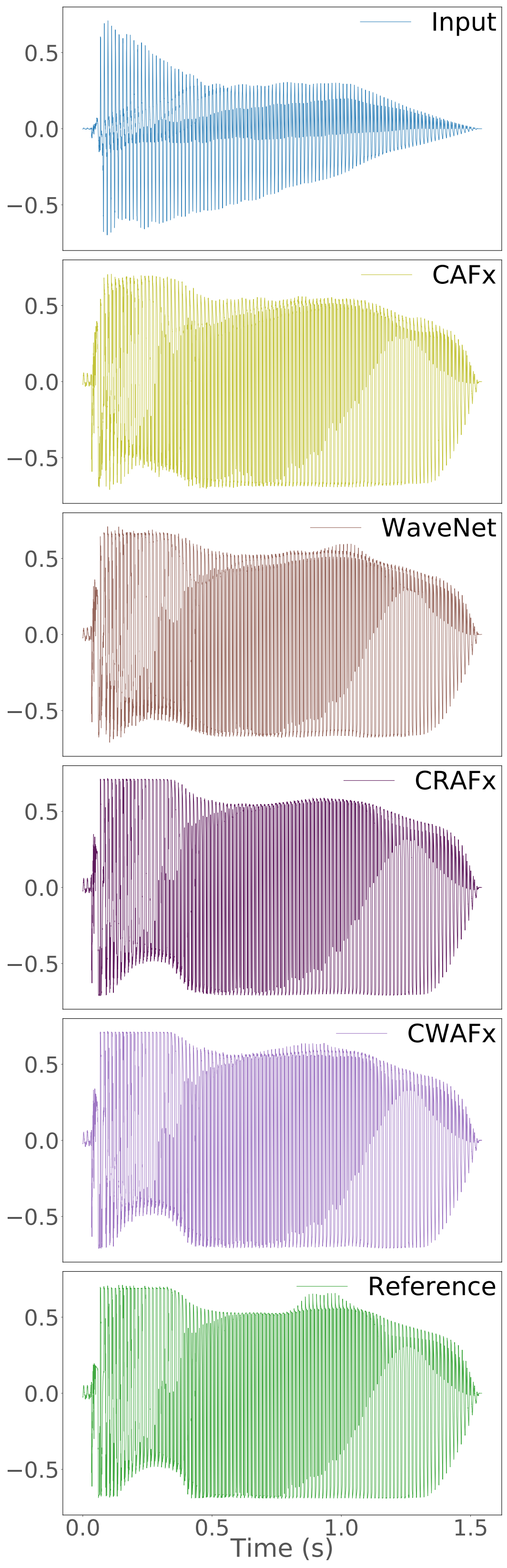


(f)

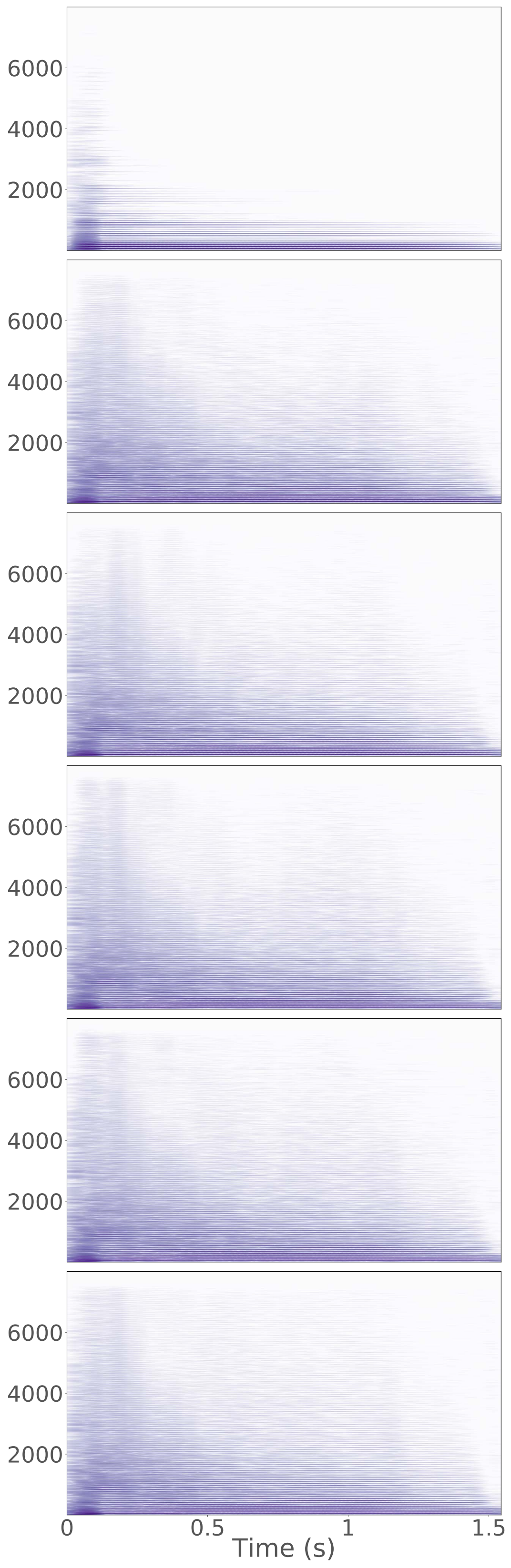
Figure 7.3: Box plot showing the rating results of the listening tests. Fig. 7.3a **preamp**, Fig. 7.3b **limiter**, Fig. 7.3c **Leslie speaker horn-tremolo**, Fig. 7.3d **Leslie speaker woofer-tremolo**, Fig. 7.3e **Leslie speaker horn-chorale** and Fig. 7.3f **Leslie speaker woofer-chorale**.

For selected test samples of the *preamp* and *limiter* tasks and for all the different models, Figs. 7.4 and 7.5 show the input, reference, and output waveforms together with their respective spectrogram. Both in the time-domain and in the frequency-domain, it is observable that the waveforms and spectrograms are in line with the objective and subjective findings. To more closely display the performance of these nonlinear tasks, Fig. 7.6 shows a segment of the respective waveforms. It can be seen how the different models match the waveshaping from the overdriven *preamp* as well as the attack waveshaping of the *limiter* when processing the onset of the test sample.

Regarding the *Leslie speaker* modeling task, Figs. 7.7 to 7.10 show the different waveforms together with their respective modulation spectrum and spectrogram: Fig. 7.7 *horn-tremolo*, Fig. 7.8 *woofer-tremolo*, Fig. 7.9 *horn-chorale* and Fig. 7.10 *woofer-chorale*. From the spectra, it is noticeable that *CRAFx* and *CWAFx* introduce and match the amplitude and frequency modulations of the reference, whereas *CAFX* and *WaveNet* do not introduce these modulations when modeling these time-varying tasks.

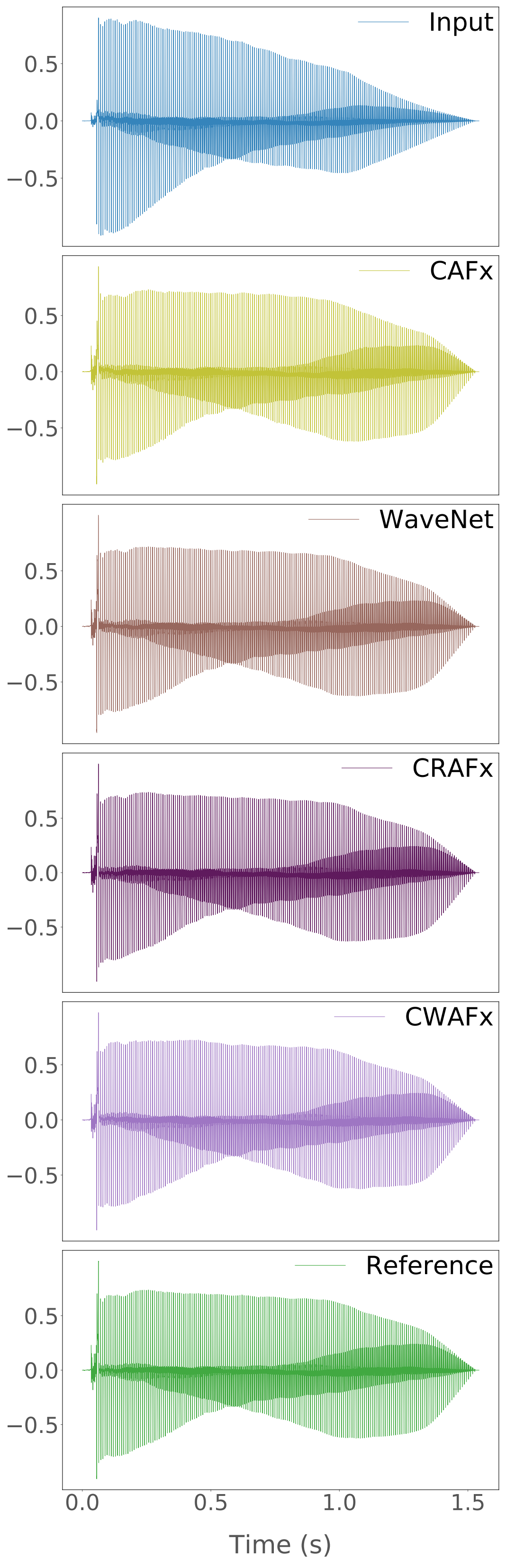


(a)

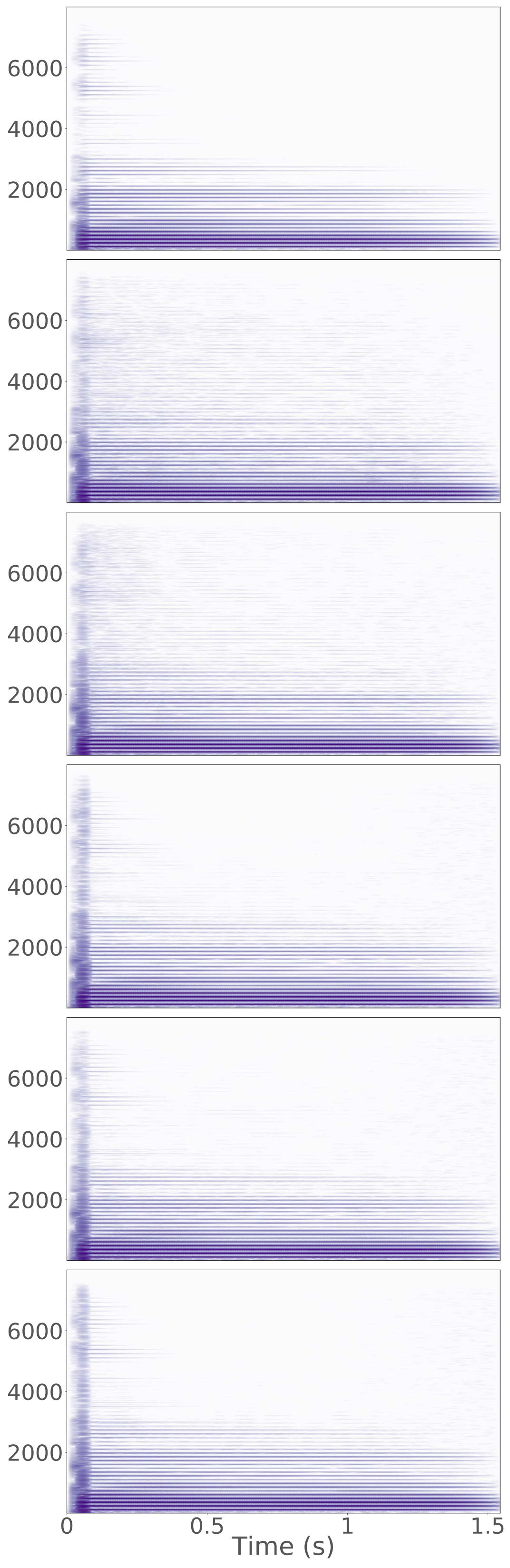


(b)

Figure 7.4: Results with selected samples from the test dataset for the **preamp** task. Figs. 7.4a and 7.4b show the waveforms and their respective spectrograms. Vertical axes represent amplitude and frequency (Hz) respectively.

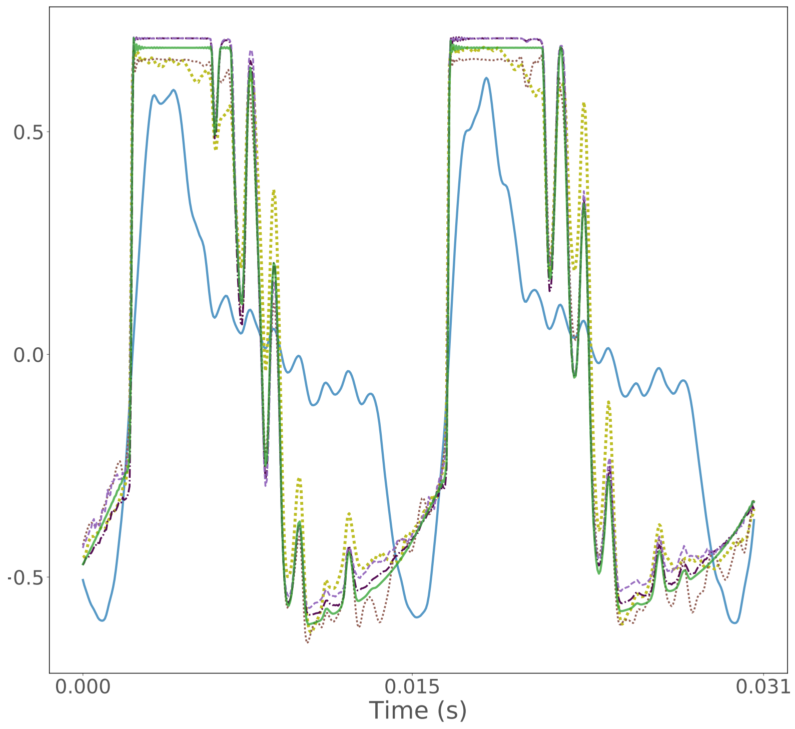


(a)

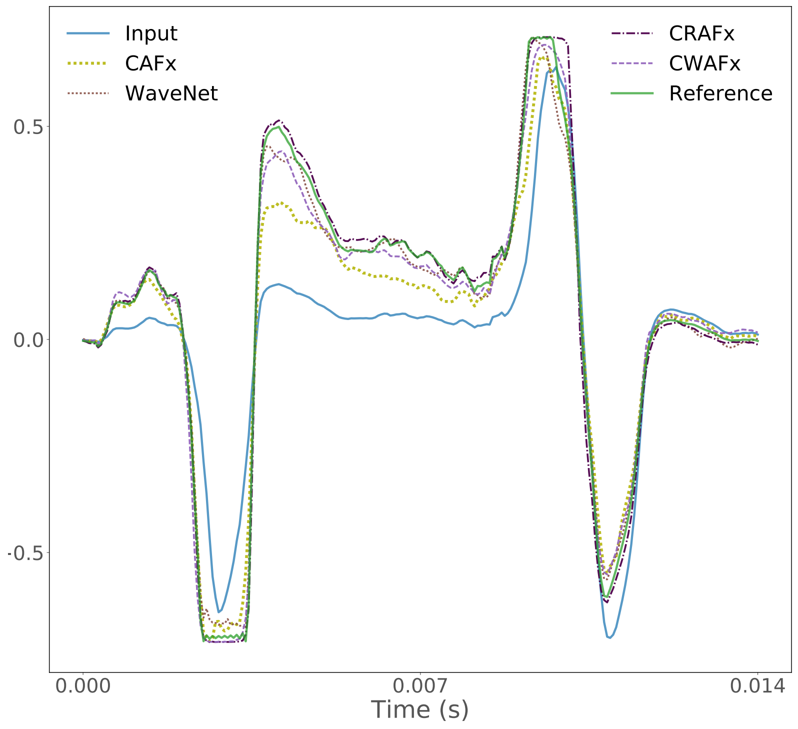


(b)

Figure 7.5: Results with selected samples from the test dataset for the **limiter** task. Figs. 7.5a and 7.5b show the waveforms and their respective spectrograms. Vertical axes represent amplitude and frequency (Hz) respectively.

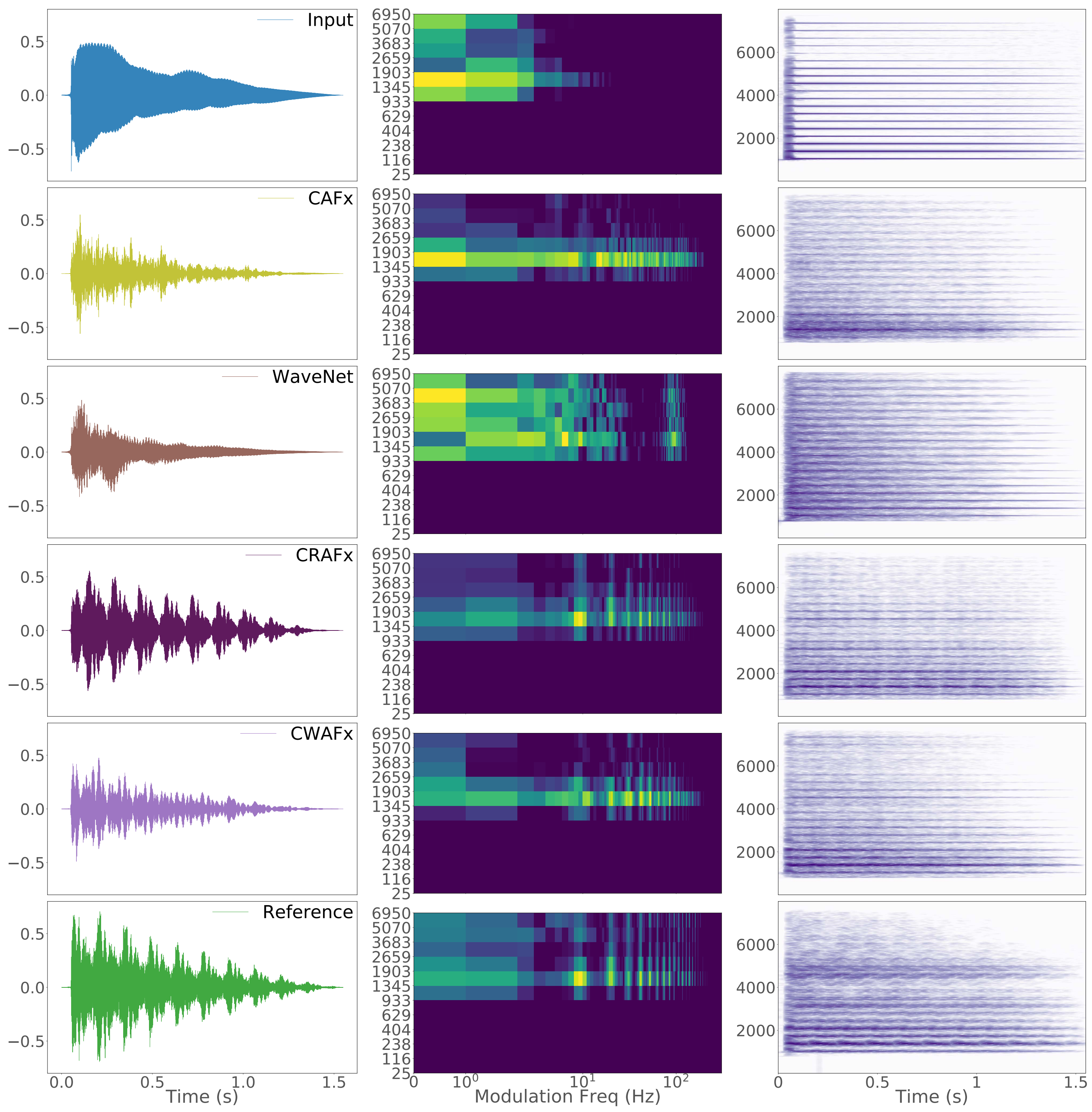


(a)



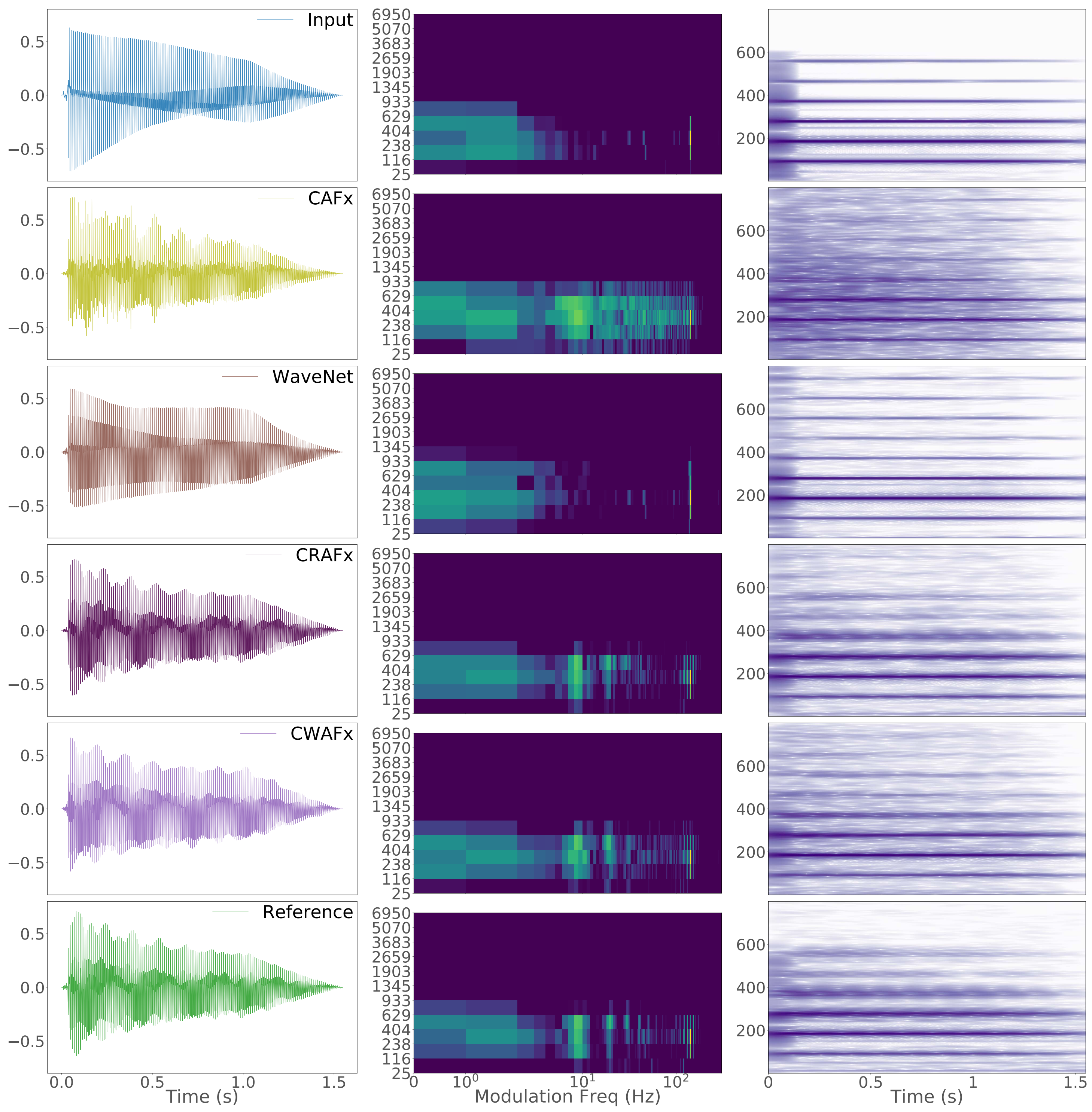
(b)

Figure 7.6: For the test samples from Figs. 7.4 and 7.5, a segment of the respective waveforms: Fig. 7.6a **preamp** task and Fig. 7.6b **limiter** task when processing the onset of the input audio. Vertical axes represent amplitude.



(a) (b) (c)

Figure 7.7: Results with selected samples from the test dataset for the **Leslie speaker horn-tremolo** tasks. Fig. 7.7a waveform, Fig. 7.7b modulation spectrum and Fig. 7.7c spectrogram. Vertical axes represent amplitude, Gammatone frequency (Hz) and FFT frequency (Hz) respectively.



(a) (b) (c)

Figure 7.8: Results with selected samples from the test dataset for the **Leslie speaker woofer-tremolo** tasks. Fig. 7.8a waveform, Fig. 7.8b modulation spectrum and Fig. 7.8c spectrogram. Vertical axes represent amplitude, Gammatone frequency (Hz) and FFT frequency (Hz) respectively.

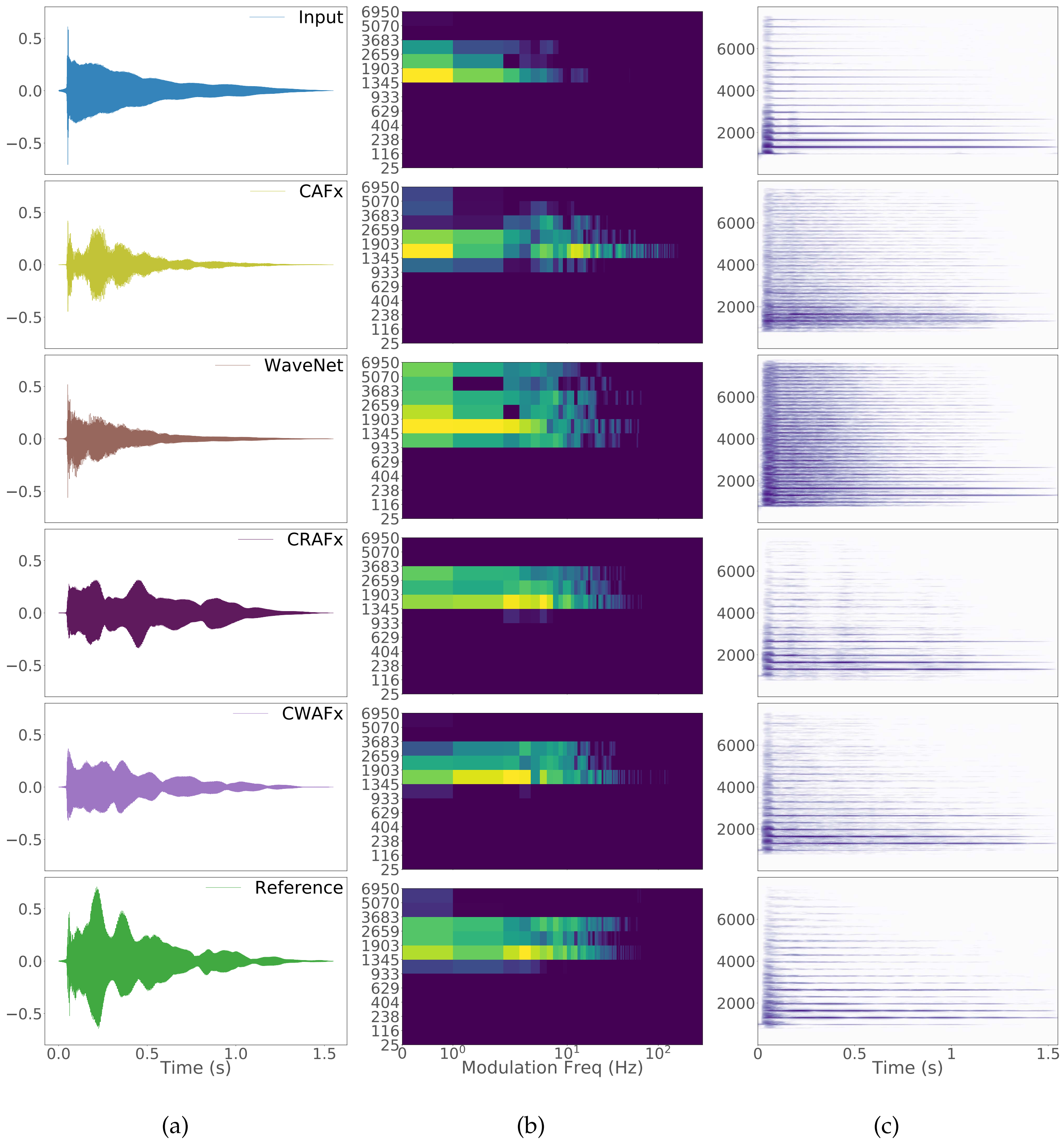


Figure 7.9: Results with selected samples from the test dataset for the **Leslie speaker horn-chorale** tasks. Fig. 7.9a waveform, Fig. 7.9b modulation spectrum and Fig. 7.9c spectrogram. Vertical axes represent amplitude, Gammatone frequency (Hz) and FFT frequency (Hz) respectively.

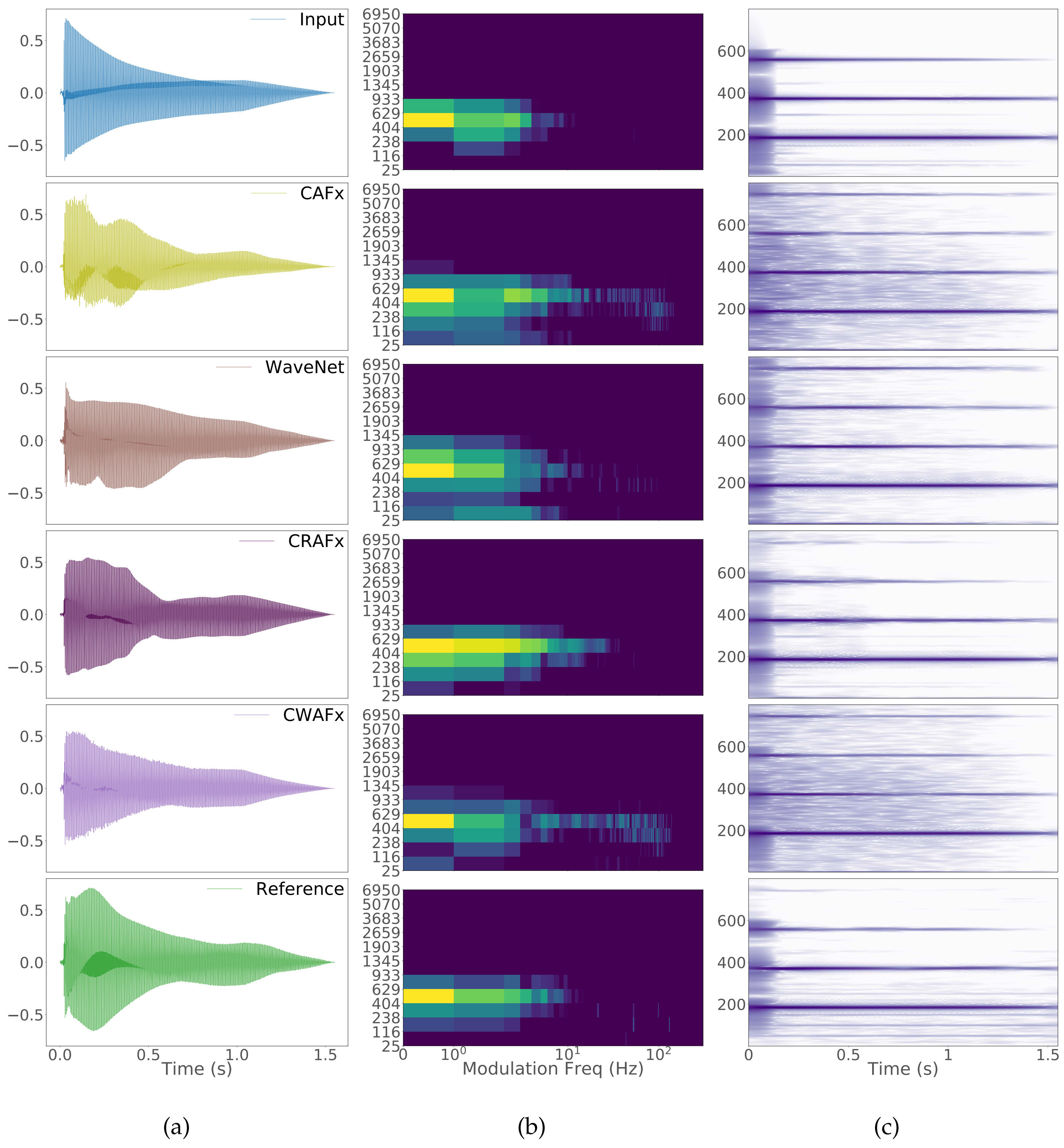


Figure 7.10: Results with selected samples from the test dataset for the **Leslie speaker woofer-chorale** tasks. Fig. 7.10a waveform, Fig. 7.10b modulation spectrum and Fig. 7.10c spectrogram. Vertical axes represent amplitude, Gammatone frequency (Hz) and FFT frequency (Hz) respectively.

7.3 DISCUSSION

Nonlinear task with short-term memory - preamp

The architectures that were designed to model nonlinear effects with short-term memory, such as *CAFx* and *WaveNet*, were outperformed by the models that incorporate temporal dependencies. With *CRAFx* and *CWAFx* being the highest scoring models both objectively and perceptually. Although this task does not require a long-term memory, the context input frames and latent-space recurrent layers and temporal dilated convolutions from *CRAFx* and *CWAFx* respectively, benefited the modeling of the *preamp*. This performance improvement could be on account of the temporal behaviour present on the vaccum-tube amplifier, such as hysteresis or attack and release timings, although additional tests on the *preamp* might be required.

Given the results reported in Chapter 5 it is remarkable that the performance of these architectures (*CAFx* and *WaveNet*) is exceeded by *CRAFx* and *CWAFx*. It is worth noting that *CAFx* and *WaveNet* from Chapter 5 are trained with input frame sizes of 1024 samples, which could indicate a decrease in modeling capabilities when handling larger input frame sizes, such as 4096 samples.

Nevertheless, from Fig. 7.3a, we can conclude that all models accomplished the modeling of the *preamp*. Most of the output audio is only slightly discernible from their target counterparts, with *CRAFx* and *CWAFx* being virtually indistinguishable form the real analog device.

Time-dependent nonlinear task - limiter

Since the *limiter* task includes long temporal dependencies such as a 1100 ms release gate, as expected, the architectures that include memory achieved a higher performance both objectively and subjectively. From Fig. 7.5b it can be seen that *CAFx* and *WaveNet* introduce high frequency information that is not present in the reference spectrogram. This could be an indication that the models compensate for their limitations when modeling information beyond one input frame, such as the distortion tone characteristic due to the long release time together with the variable ratio of the *limiter*. Furthermore, from Fig. 7.6b it is noticeable how each architecture models the attack behavior of the *limiter*.

We can conclude that although all networks closely matched the reference target, it is *CRAFx* and *CWAFx* which achieved the lowest objective metrics and highest perceptual ratings when modeling the saturation waveshaping characteristic of the audio processor. The latter is accentuated with the perceptual results from Fig. 7.3b, where *CRAFx* and *CWAFx* are again virtually indistinguishable from the reference target. While *CAFx* and *WaveNet* are ranked behind due to the lack of long-term memory capabilities, it is noteworthy that these models closely rendered the desired waveform.

Time-varying task - Leslie speaker

With respect to the *horn tremolo* and *woofer tremolo* modeling tasks, it can be seen that for both rotating speakers, *CRAFx* and *CWAFx* are rated highly whereas *CAFx* and *WaveNet* are rated poorly when modeling these tasks. Thus, the perceptual findings from Figs. 7.3c and 7.3d confirm the results obtained with the *ms_mse* metric and overall, the *woofer* task has a better matching that the *horn* task. Nevertheless, for *CRAFx* and *CWAFx*, the objective and subjective ratings for the *horn tremolo* task do not represent a significant decrease of performance and it can be concluded that both time-varying tasks were modeled by these architectures.

CRAFx is perceptually ranked slightly higher than *CWAFx*. Although a statistical analysis is required, this might indicate a closer matching of the reference amplitude and frequency modulations, which can be seen in the respective modulation spectra and spectrograms from Fig. 7.7 and Fig. 7.8.

Based on the perceptual ratings, for the *horn chorale* and *woofer chorale* modeling tasks, *CRAFx* and *CWAFx* modeled the former while only *CRAFx* accomplished the *woofer chorale* task. Since the *woofer chorale* task corresponds to modulations lower than 0.8 Hz, we can conclude that Bi-LSTMs are more adequate than a latent-space WaveNet when modeling such low-frequency modulations. Furthermore, this is closely associated with the objective metrics reported in Section 6.4, where *CWAFx* obtained the highest *mae* values when modeling effects based on low-frequency modulation, such as *vibrato*.

In general, from Fig. 7.7 to Fig. 7.10, it is observable that the output waveforms do not match the waveforms of the references. This shows that the models are not overfitting to the waveforms of the training data and, taking into account the perceptual ratings, this indicates that the highest rated models are learning to introduce the respective amplitude and frequency modulations. The models cannot replicate the exact reference waveform since the phase of the rotating speakers varies across the whole dataset. For this reason, the early stopping and model selection procedures of these tasks were based on the training loss rather than the validation loss. This is also the reason of the high *mae* scores across the *Leslie speaker* modeling tasks, due to these models applying the modulations yet without exactly matching their phase in the target data. Further exploration of a phase-invariant cost function could improve the performance of the different architectures.

CAFx and *WaveNet* were not able to accomplish these time-varying tasks. It is worth noting that both architectures try to compensate for long-term memory limitations with different strategies. It is suggested that *CAFx* wrongly introduces several amplitude modulations, whereas *WaveNet* tries to average the waveform envelope of the reference. This results in output audio significantly different from the reference, with *WaveNet* being perceptually rated as the lowest for the *horn tremolo* and *horn chorale* tasks. This also explains the *ms_mse* results from Fig. 7.2 for the *woofer chorale* task, where *WaveNet* achieves the best score since averaging the target waveform could be introducing the low-frequency amplitude modulations present in the reference audio.

7.4 CONCLUSION

In this chapter, we explored the different deep learning architectures from Chapters 5 and 6. We tested the models when modeling nonlinear effects with short-term and long-term memory such as a tube *preamp* and a transistor-based *limiter*; and nonlinear time-varying processors such as the rotating *horn* and *woofer* of a *Leslie speaker* cabinet.

Through objective perceptual-based metrics and subjective listening tests we found that across all modeling tasks, the architectures that incorporate Bi-LSTMs or, to a lesser extent, latent-space dilated convolutions to explicitly learn long temporal dependencies, outperform the rest of the models. With these architectures we obtain results that are virtually indistinguishable from the analog reference processors, although a statistical analysis of the results of the listening tests is required in order to confirm their validity. Also, state-of-the-art DNN architectures for modeling nonlinear effects with short-term memory perform similarly when matching the *preamp* task and achieve low-error and high perceptual similarity when modeling the *limiter* task, but fail to yield the same results when modeling the time-varying *Leslie speaker* tasks.

Based on the objective metrics and perceptual ratings, we can conclude that the nonlinear amplifier, rotating speakers and wooden cabinet from the *Leslie speaker* were emulated by *CRAFx* and to a certain extent by *CWAFx*. Nevertheless, the crossover filter was bypassed in the modeling tasks and the dry and wet audio were filtered accordingly. This was due to the limited frequency bandwidth of the bass and guitar samples, thus, this modeling task could be further explored with a more appropriate dataset such as Hammond organ recordings.

As future work, a cost function based on both time and frequency can be used to further improve the modeling capabilities of the models. In addition, since the highest ranked architectures use past and subsequent context input frames, more research is needed on how to adapt these architectures to overcome the resulting latency. Thus, real-time applications would benefit significantly from the exploration of end-to-end DNNs that include long-term memory without resorting to large input frame sizes and the need for past and future context frames. Also, an end-to-end architecture of temporal dilated convolutions with a receptive field as large as the context input frames from *CRAFx* and *CWAFx* could also be explored for the time-varying modeling tasks.

Moreover, as shown in Damskägg et al. (2019), the introduction of controls as a conditioning input to the networks can be investigated, since the models are currently learning a static representation of the audio effect. Finally, applications beyond virtual analog can be investigated, for example, in the field of automatic mixing the models could be trained to learn a generalization from mixing practices.

8 MODELING ARTIFICIAL REVERBERATION

In this chapter we present a deep learning architecture to model artificial reverberators such as *plate* and *spring*. As mentioned in Section 2.7.2, *plate* and *spring* reverberators are electromechanical audio processors mainly used for aesthetic reasons and characterized by their particular sonic qualities. The modeling of these reverberators remains an active research field due to their their mechanical elements which frequency response is difficult to fully emulate digitally (see Section 2.8).

We explore the capabilities of DNNs to learn the respective transformation and perceptual qualities of these electromechanical reverberators. Therefore based on digital reverberators that use sparse FIR (SFIR) filters, we use domain knowledge from signal-processing systems and we propose the *Convolutional recurrent and Sparse filtering audio effects modeling network (CSAFx)*.

CSAFx represents a DSP-informed DNN for modeling artificial reverberators since we extend previous architectures by incorporating trainable FIR filters with sparsely placed coefficients in order to model diffused and noisy responses, such as those present in *plate* and *spring* devices. We also modify the *Squeeze-and-Excitation* (SE) blocks from *CRAFx* (see Sections 3.7 and 6.1) in order to act as time-varying mixing gains between the direct sound and the reflections. The SE blocks explicitly scale the channel-wise information of input feature maps, i.e. the SE blocks apply a scalar gain to each row or channel. Thus, we extend this block by placing LSTMs to allow the model to learn long temporal dependencies, such as the time-varying mixing gains for the direct sound and early and late reflections.

Based on the results of the virtual analog experiments from Chapter 7, we use *CRAFx* as baseline model and we also test its capabilities when modeling artificial reverberation. In order to measure the performance, we conduct a perceptual listening test and we also analyze how the given task is accomplished and what the model is actually learning.

Prior to this work, end-to-end DNNs have not yet been implemented to model artificial reverberators, i.e. learning from input-output data and applying the reverberant effect directly to the dry input audio. Although deep learning for dereverberation has become a heavily researched field (Feng et al., 2014; Han et al.,

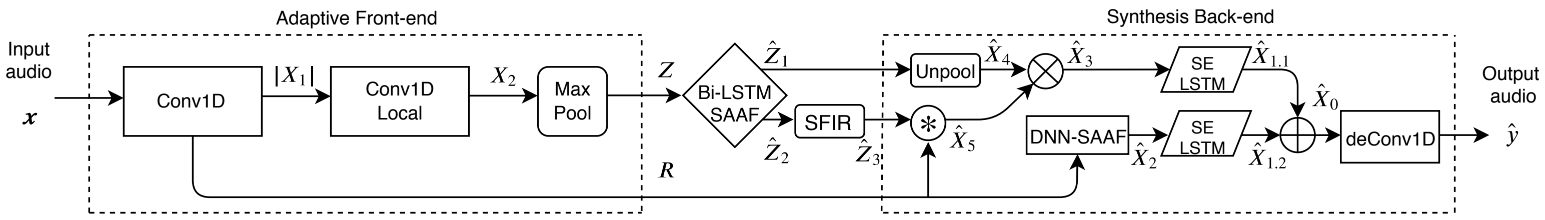


Figure 8.1: Block diagram of *CSAFx*; adaptive front-end, latent-space and synthesis back-end.

2015), applying artificial reverberation or modeling *plate* and *spring* reverb with DNNs has not been explored yet.

We report that *CSAFx* outperforms *CRAFx* for this task. Perceptual and objective evaluations indicate that the proposed model accomplishes the emulation of electromechanical artificial reverberators. In addition, it achieves better results than a proven DNN for black-box modeling of audio effects. Audio samples can be found in Appendix B.

8.1 CONVOLUTIONAL RECURRENT AND SPARSE FILTERING NETWORK - CSAFX

The model builds on *CRAFx* and is also completely based on the time-domain; the model uses raw and processed audio as input and output, respectively. It is divided into three parts: adaptive front-end, latent-space and synthesis back-end. A block diagram is depicted in Fig. 8.1, code is available online[1] and Table C.1 displays the number of parameters and computational processing times.

The *adaptive front-end* is exactly the same as the one from *CRAFx* (see Table 6.1). It follows the same time distributed convolutional and pooling layers, yielding a filter bank architecture of 32 channels which learns the latent representation $\mathbf{Z}$. Likewise, the model learns long-term memory dependencies by having an input $\mathbf{x}$ which consists of the current audio frame x concatenated with the ±4 previous and subsequent frames. The input is described by Eq. (6.1). These frames are of size 4096 (256 ms) and sampled with a hop size of 50%.

Latent-space

A block diagram of the latent-space can be seen in Fig. 8.2 and its structure is described in detail in Table 8.1. The latent-space has as its main objective to process

1 https://github.com/mchijmma/modeling-plate-spring-reverb/tree/master/src

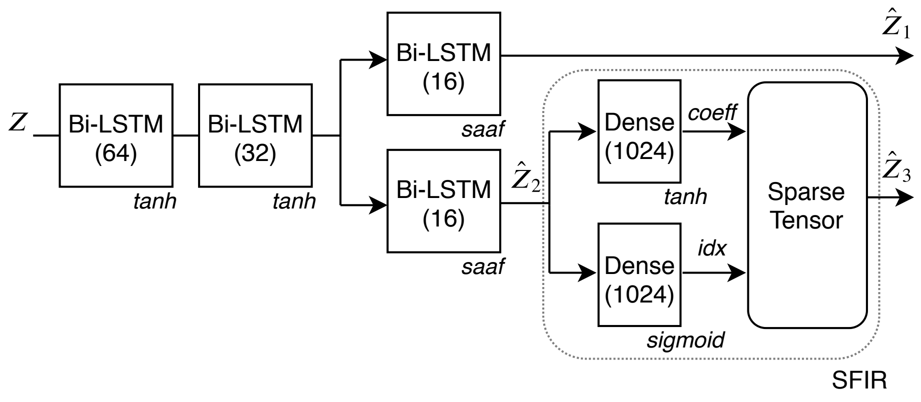


Figure 8.2: Block diagram of the latent-space of *CSAFx*.

Table 8.1: Detailed architecture of the latent-space of *CSAFx*. This with an input frame size of 4096 samples and ±4 context frames. Output shape = (m,n) denotes m columns and n rows.

Layer - Output shape			Output	
$\mathbf{Z}$ (64, 288)				
Bi-LSTM (64, 128)			.	
Bi-LSTM (64, 64)			.	
Bi-LSTM (64, 32)	Bi-LSTM (64, 32)		.	
SAAF (64, 32)	SAAF (64, 32)		$\hat{\mathbf{Z}}_1$	$\hat{\mathbf{Z}}_2$
.	Dense (1024, 32)	Dense (1024, 32)	.	.
.	Sparse Tensor (4096, 32)		.	$\hat{\mathbf{Z}}_3$

$\mathbf{Z}$ into two latent representations, $\hat{\mathbf{Z}}_1$ and $\hat{\mathbf{Z}}_2$. The former corresponds to a set of envelope signals and the latter is fed to the next layer in order to obtain a set of sparse FIR filters $\hat{\mathbf{Z}}_3$.

The latent representation $\mathbf{Z}$ from the front-end corresponds to 9 frames of 64 samples and 32 channels, which can be unrolled into a feature map of 64 samples and 288 channels. The latent-space consists of two shared Bi-LSTM layers of 64 and 32 units with tanh as activation function. The output feature map from these Bi-LSTM layers is fed to two independent Bi-LSTM layers of 16 units. Each of these layers is followed by locally connected SAAFs as the nonlinearity, obtaining in this way $\hat{\mathbf{Z}}_1$ and $\hat{\mathbf{Z}}_2$, which correspond to matrices of 32 channels of 64 samples. As

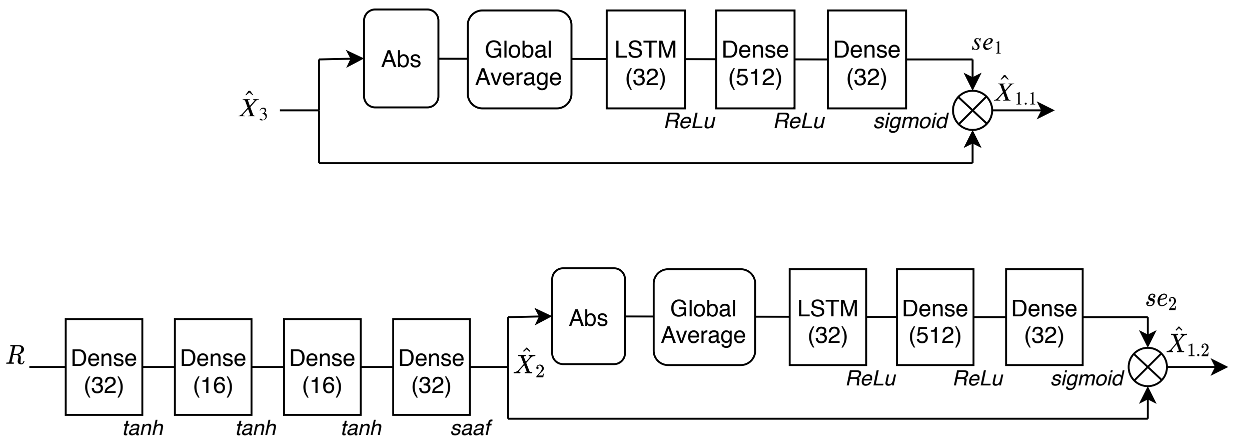


Figure 8.3: Block diagram of the synthesis back-end of *CSAFx*.

shown in previous chapters, SAAFs can be used as nonlinearities or waveshapers in audio processing tasks. The input-to-hidden weights of the Bi-LSTM layers are applied to the channel dimension of the input feature maps.

We propose a SFIR layer where we follow the constraints of sparse pseudo-random reverberation algorithms (Välimäki et al., 2012). As mentioned in Section 2.7.1, early reflections are modeled via FIR filters with sparsely placed coefficients. These coefficients are usually obtained through a pseudo-random number sequence (e.g. velvet noise), which is based on discrete coefficient values such as -1 and +1, where each one of the coefficients follows an interval of T_s samples while all the other samples are zero.

Nevertheless, in SFIR, instead of using discrete coefficient values, each coefficient can take any continuous value within -1 to $+1$. Accordingly, each one of the coefficients is placed at a specific index position within each interval of T_s samples while the rest the samples are zero.

Thus, the SFIR layer processes $\hat{\mathbf{Z}}_2$ by two independent dense layers of 1024 units each. The dense layer is applied along the latent dimension rather than to the channel dimension, i.e. the matrix multiplication is computed between the FC weights and the rows of the input matrix. The dense layers are followed by a tanh and *sigmoid* function, whose outputs are the coefficient values (*coeff*) and their index position (*idx*) respectively. To obtain the specific *idx* value, the output of the *sigmoid* function is multiplied by T_s and a rounding down to the nearest integer

Table 8.2: Detailed architecture of the synthesis back-end of *CSAFx*. This with input frame size of 4096 samples and ±4 context frames.

Layer - Output shape		Output	
$\hat{Z}_1$ (4096, 32)	$\hat{Z}_3$ (4096, 32)		
Unpooling (4096, 32)	$\mathbf{R} * \hat{Z}_3$ (4096, 32)	$\hat{X}_4$	$\hat{X}_5$
$\hat{X}_4 \times \hat{X}_5$ (4096, 32)		$\hat{X}_3$	
$\hat{X}_3$ (4096, 32)	$\mathbf{R}$ (4096, 32)		
.	Dense (4096, 32)	.	.
.	Dense (4096, 16)	.	.
.	Dense (4096, 16)	.	.
.	Dense (4096, 32)	.	.
.	SAAF (4096, 32)	.	$\hat{X}_2$
Abs (4096, 32)	Abs (4096, 32)	.	.
G-Avg (1, 32)	G-Avg (1, 32)	.	.
LSTM (1, 32)	LSTM (1, 32)	.	.
Dense (1, 512)	Dense (1, 512)	.	.
Dense (1, 32)	Dense (1, 32)	$se1$	$se2$
$se1 \times \hat{X}_3$ (4096, 32)	$se2 \times \hat{X}_2$ (4096, 32)	$\hat{X}_{1.1}$	$\hat{X}_{1.2}$
$\hat{X}_{1.1} + \hat{X}_{1.2}$ (4096, 32)		$\hat{X}_0$	
deConv1D (4096, 1)		$\hat{y}$	

is applied. This operation is not differentiable so we use an identity gradient as a backward pass approximation (Athalye et al., 2018). In order to have a high-quality reverberation, we use 2000 coefficients per second (see Section 2.7.1), thus, $T_s = 8$ samples for a sampling rate of 16 kHz.

Synthesis back-end

The synthesis back-end can be seen in more detail in Fig. 8.3 and Table 8.2. The back-end uses the SFIR output $\hat{Z}_3$, the envelopes $\hat{Z}_1$ and the residual connection $\mathbf{R}$ to synthesize the waveform and accomplish the reverberation task. It consists of an unpooling layer, a convolution and multiplication operation, a DNN with SAAFs

(DNN-SAAF), two modified Squeeze-and-Excitation blocks (Hu et al., 2018) that incorporate LSTM layers (*SE-LSTM*) and a final convolutional layer.

Following the filter bank architecture: $\hat{\mathbf{X}}_4$ is obtained by upsampling $\hat{\mathbf{Z}}_1$ and the feature map $\hat{\mathbf{X}}_5$ is accomplished by the locally connected convolution between $\mathbf{R}$ and $\hat{\mathbf{Z}}_3$. As in *CRAFx*, $\mathbf{R}$ is obtained from $\mathbf{X}_1$ and corresponds to the frequency band decomposition of the current input frame $\mathbf{x}^{(0)}$. $\hat{\mathbf{X}}_5$ is obtained with the following equation:

$$\hat{\mathbf{X}}_5^{(i)} = \mathbf{R}^{(i)} * \hat{\mathbf{Z}}_3^{(i)} \; \forall i \in [1, 32], \tag{8.1}$$

where i denotes the i*th* row of the feature maps. The matrix $\mathbf{R}$ contains one-dimensional audio representations, where each row is the resulting audio waveform after the filter bank in *Conv1D*, and $\hat{\mathbf{Z}}_3$ is a matrix where each row contains a learned sparse FIR filter. $\mathbf{R}$, $\hat{\mathbf{Z}}_3$ and $\hat{\mathbf{X}}_5$ correspond to matrices of 32 channels of 4096 samples. The convolution operation is computed between the rows of the feature maps and follows a filter bank architecture of 32 channels.

The result of this convolution can be seen as explicitly modeling a frequency dependent reverberation response with the incoming audio. Furthermore, due to the temporal dependencies learnt by the Bi-LSTMs, $\hat{\mathbf{X}}_5$ is able to represent from the onset response to the late reflections of the reverberation task.

Then the feature map $\hat{\mathbf{X}}_3$ is the result of the element-wise multiplication of the reverberant response $\hat{\mathbf{X}}_5$ and the learnt envelopes $\hat{\mathbf{X}}_4$. The envelopes are applied in order to avoid audible artifacts between input frames (Järveläinen and Karjalainen, 2007). $\hat{\mathbf{X}}_3$ is computed with the following equation:

$$\hat{\mathbf{X}}_3 = \hat{\mathbf{X}}_5 \times \hat{\mathbf{X}}_4, \tag{8.2}$$

where $\hat{\mathbf{X}}_4$ and $\hat{\mathbf{X}}_3$ correspond to matrices of 32 channels of 4096 samples.

Secondly, the feature map $\hat{\mathbf{X}}_2$ is obtained when the waveshaping nonlinearites from the DNN-SAAF block are applied to $\mathbf{R}$. The result of this operation consists of a learnt nonlinear transformation or waveshaping of the direct sound (see Section 5.1). As used in *CRAFx*, the DNN-SAAF block consists of 4 dense layers of 32, 16, 16 and 32 hidden units respectively. Each dense layer uses tanh as nonlinearity except for the last one, which uses a SAAF layer.

We propose an SE-LSTM block to act as a time-varying gain for $\hat{\mathbf{X}}_2$ and $\hat{\mathbf{X}}_3$. Since SE blocks explicitly and adaptively scale the channel-wise information of feature maps (Hu et al., 2018), we incorporate an LSTM layer in the SE architecture in order to include long-term context from the input. Each SE-LSTM builds on the

SE blocks from Section 6.1 which are based on the architecture from Kim et al. (2018).

The SE-LSTMs blocks consist of an *absolute value* operation and global average pooling operation followed by one LSTM and two dense layers of 32, 512 and 32 hidden units respectively. The LSTM and first dense layer are followed by a *ReLu*, while the last dense layer uses a *sigmoid* activation function. The weights of the dense and recurrent layers are applied to the channel dimension of the input feature maps, i.e. the matrix multiplication is computed between the weights and the columns of the input matrix. As depicted in Fig. 8.3, each SE-LSTM block processes each feature map $\hat{\mathbf{X}}_2$ and $\hat{\mathbf{X}}_3$, thus, applying a frequency dependent time-varying mixing gain $se1$ and $se2$. The resulting feature maps $\hat{\mathbf{X}}_{1.1}$ and $\hat{\mathbf{X}}_{1.2}$ are added together in order to obtain $\hat{\mathbf{X}}_0$:

$$\hat{\mathbf{X}}_{1.1} = se1 \times \hat{\mathbf{X}}_3, \tag{8.3}$$

$$\hat{\mathbf{X}}_{1.2} = se2 \times \hat{\mathbf{X}}_2, \tag{8.4}$$

$$\hat{\mathbf{X}}_0 = \hat{\mathbf{X}}_{1.1} + \hat{\mathbf{X}}_{1.2}, \tag{8.5}$$

where $\hat{\mathbf{X}}_0$, $\hat{\mathbf{X}}_{1.1}$, $\hat{\mathbf{X}}_{1.2}$ and $\hat{\mathbf{X}}_2$ correspond to matrices of 32 channels of 4096 samples, and $se1$ and $se2$ correspond to column vectors of 32 channels.

As in the previous deep learning architectures, the last layer corresponds to the *deconvolution* operation which is not trainable since its filters are the transposed weights of the first convolutional layer. The complete waveform is synthesized using a *hann* window and constant overlap-add gain. As shown in the previous *CEQ*, *CAFx*, *CRAFx* and *CWAFx* architectures, all convolutions are along the time dimension and all strides are of unit value. For each convolutional layer we use the same padding and dilation is not incorporated.

Overall, each SAAF is locally connected and each function consists of 25 intervals between −1 to +1 and each Bi-LSTM and LSTM have dropout and recurrent dropout rates of 0.1.

8.1.1 *Loss function*

The loss function to be minimized is based in time and frequency and described by:

$$\text{loss} = \alpha_1 MAE(y, \hat{y}) + \alpha_2 MSE(Y, \hat{Y}) \quad (8.6)$$

Where *MAE* is the mean absolute error and *MSE* is the mean squared error. Y and $\hat{Y}$ are the logarithmic power magnitude spectra in dBs of the target and output respectively, and y and $\hat{y}$ their respective waveforms. Prior to calculating the *MAE*, the following pre-emphasis filter is applied to y and $\hat{y}$ using the following equation:

$$H(z) = 1 - 0,95z^{-1}. \quad (8.7)$$

As shown in Damskägg et al. (2019), $H(z)$ is a highpass filter that we apply in order to add more weight to the high frequencies. We use a 4096-point FFT to obtain Y and $\hat{Y}$. In order to scale the time and frequency losses, we empirically set 1.0 and 1e−4 as the loss weights α_1 and α_2 respectively. We found empirically that explicit minimization in the frequency and time domains resulted crucial when modeling such complex reverberant responses. The attention to the high frequencies is further emphasized by incorporating the pre-emphasis filter and the logarithmic power spectrum in the time and frequency domain, respectively.

8.2 EXPERIMENTS

8.2.1 *Training*

We follow the same pretraining initialization step as in Section 4.2.1. Once the convolutional layers of the front-end and back-end are initialized, the latent-space Bi-LSTMs, SFIR, DNN-SAAF and SE-LSTM blocks are incorporated into the model, and all the weights are trained jointly based on the reverberation task.

For both training steps, *Adam* (Kingma and Ba, 2015) is used as optimizer and we use the same early stopping procedure from Section 7.1.2. We use a patience value of 25 epochs if there is no improvement in the validation loss. Similarly, afterwards the model is fine-tuned further with the learning rate reduced by 25% and also a patience value of 25 epochs. The initial learning rate is $1e-4$ and the batch size consists of the total number of frames per audio sample. We select the model with the lowest error for the validation subset.

Table 8.3: Settings for each artificial reverberation modeling task.

Fx	settings
plate	**Smaertelectronix ambience**: 'Gating Amount - 0', 'Gating Attack" - 10 ms', 'Gating Release - 10 ms', 'Decay Time - 2225 ms', 'Decay Diffusion - 50%', 'Decay Hold - off', 'Shape Size - 16%', 'Shape Predelay - 0 ms', 'Shape Width - 100%', 'Shape Quality - 100%', 'Shape Variation - 0', 'EQ Bass Frequency - 43 Hz', 'EQ Bass Gain - −7.8 dB', 'EQ Treble Frequency - 5044 Hz', 'EQ Treble Gain - −3.7 dB', 'Damping Bass Frequency - 158 Hz', 'Damping Bass Amount - 87%', 'Damping Treble Frequency - 8127 Hz', 'Damping Treble Amount - 32%', 'Dry - −Inf', 'Wet - 0dB'
spring	**Accutronics 4EB2C1B**: 'Dry Mix - 0%', 'Wet Mix - 100%'

8.2.2 *Dataset*

Plate reverberation is obtained from the *IDMT-SMT-Audio-Effects* dataset (Stein et al., 2010), which corresponds to individual 2-second notes and covers the common pitch range of various electric guitars and bass guitars. We use raw and *plate* reverb notes from the bass guitar recordings. *Spring* reverberation samples are obtained by processing the electric guitar raw audio samples with the *spring* reverb tank *Accutronics 4EB2C1B*. It is worth noting that the *plate* reverb samples correspond to a VST audio plug-in, while the *spring* reverb samples are recorded using an analog reverb tank which is based on 2 springs placed in parallel.

For each reverb task we use 624 raw and effected notes and both the test and validation samples correspond to 5% of this subset each. The recordings are downsampled to 16 kHz and amplitude normalization is applied. Also, since the *plate* reverb samples have a fade-out applied in the last 0.5 seconds of the recordings, we process the *spring* reverb samples accordingly. The dataset is available online[2].

8.2.3 *Evaluation*

Two objective metrics are used when testing the models with the various modeling tasks; *mae*, the energy-normalized MAE; and *mfcc_cosine*, the mean cosine distance of the MFCCs (see Section 5.3.3).

2 https://zenodo.org/record/3746119

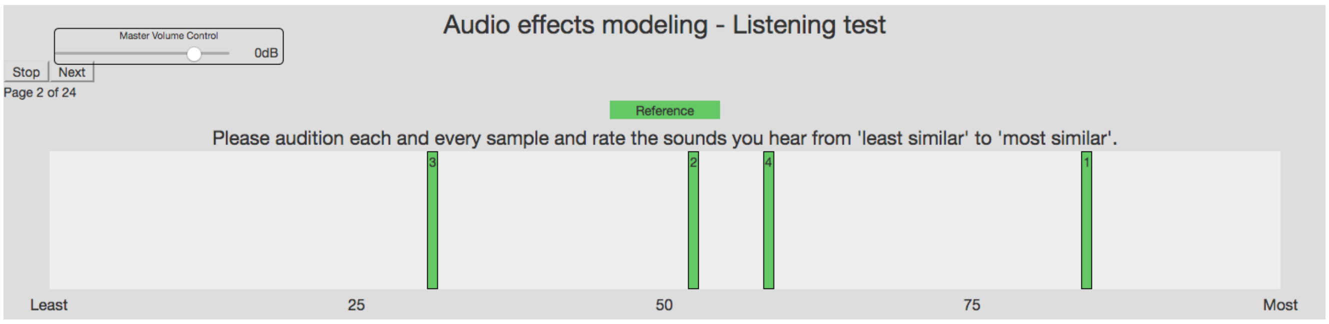

Audio effects modeling - Listening test
Master Volume Control
0dB
Stop
Next
Page 2 of 24
Reference
Please audition each and every sample and rate the sounds you hear from 'least similar' to 'most similar'.
3
2
4
1
Least
25
50
75
Most

Figure 8.4: User interface used by the participants for the listening test.

Table 8.4: **loss** values for *plate* and *spring* reverb models when tested with the test dataset.

Fx	model	*MAE*	*MSE*	*loss*
plate	*CSAFx*	**0.00214**	**7.75815**	**0.00292**
	CRAFx	0.00316	27.08704	0.00587
spring	*CSAFx*	**0.00366**	**9.43629**	**0.00461**
	CRAFx	0.00474	33.09621	0.00805

As described in Section 7.1.5, we also conducted a perceptual listening test[3] to measure the performance of the models. Thirty participants completed the test which took place at a professional listening room at Queen Mary University of London. The subjects were among musicians, sound engineers or experienced in critical listening. The audio was played via *Beyerdynamic DT-770 PRO* studio headphones and the Web Audio Evaluation Tool (Jillings et al., 2015) was used to set up the test.

The participants were presented with samples from the test subset. Each page contained a reference sound, i.e. from the original *plate* or *spring* reverb. Participants were asked to rate 4 different samples according to the similarity of these in relation to the reference sound, i.e. from 'least similar' to 'most similar'. The aim of the test was to identify which sound is closer to the reference. Thus, the test is based on the MUSHRA method (Union, 2003). The samples consisted of outputs from *CSAFx*, *CRAFx*, a hidden copy of the reference and a dry sample as hidden anchor. The user interface is displayed in Fig. 8.4.

3 The Queen Mary Ethics of Research Committee approved the listening test with reference number QMREC2165.

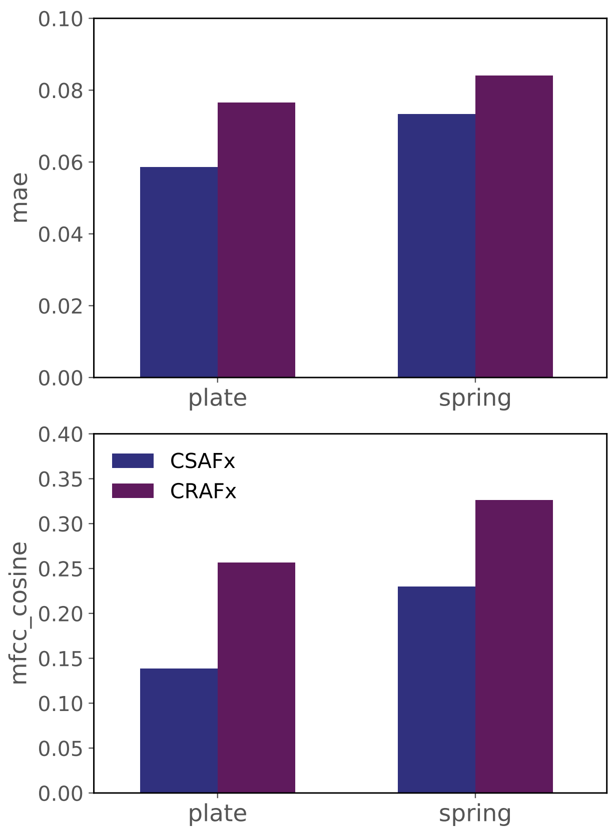


Figure 8.5: **mae** and **mfcc_cosine** values with the test dataset for all the *plate* and *spring* reverb tasks.

8.3 RESULTS & ANALYSIS

In order to compare the reverberation modeling capabilities of *CSAFx*, we use *CRAFx* as baseline, which has proven capable of modeling complex electromechanical devices with long-term memory and low-frequency modulations such as the Leslie speaker (see Chapter 7). The latter presents an architecture similar to *CSAFx*, although its latent-space and back-end have been designed to explicitly learn and apply amplitude and frequency modulations in order to match time-varying audio effects (see Section 2.5). Both models are trained under the same procedure, tested with samples from the test dataset and the audio results are available online[4].

Table 8.4 shows the corresponding *loss* values from Eq. (8.6) and Fig. 8.5 shows the *mae* and *mfcc_cosine* for all the test subsets. The proposed model outperforms *CRAFx* in both tasks. It is worth mentioning that for plate reverb, the mean *mae*

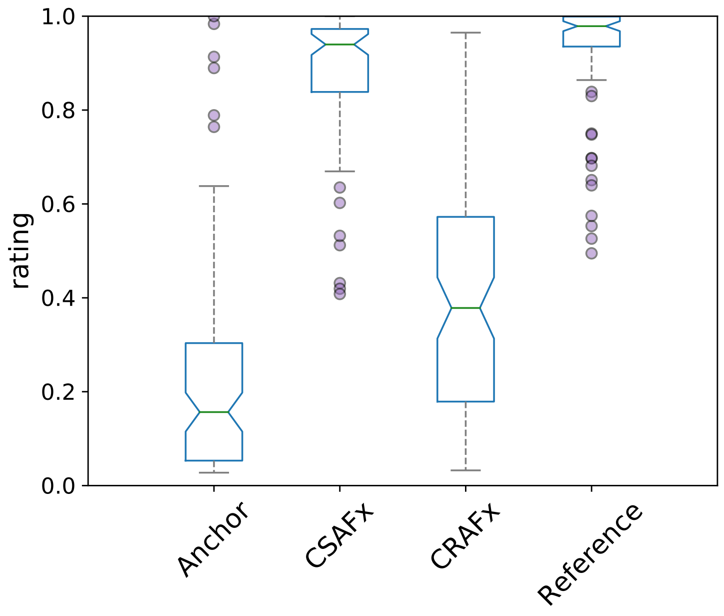
4 https://mchijmma.github.io/modeling-plate-spring-reverb/

and *mfcc_cosine* values between input and target waveforms are 0.16 and 0.15, respectively. Thus, from Fig. 8.5 we can observe that both models perform similarly well in terms of *mae*, with *CSAFx* achieving better results. Nevertheless, in terms of *mfcc_cosine*, the values obtained by *CRAFx* indicate that, perceptually, the dry notes are closer to the target than the outputs from this model.

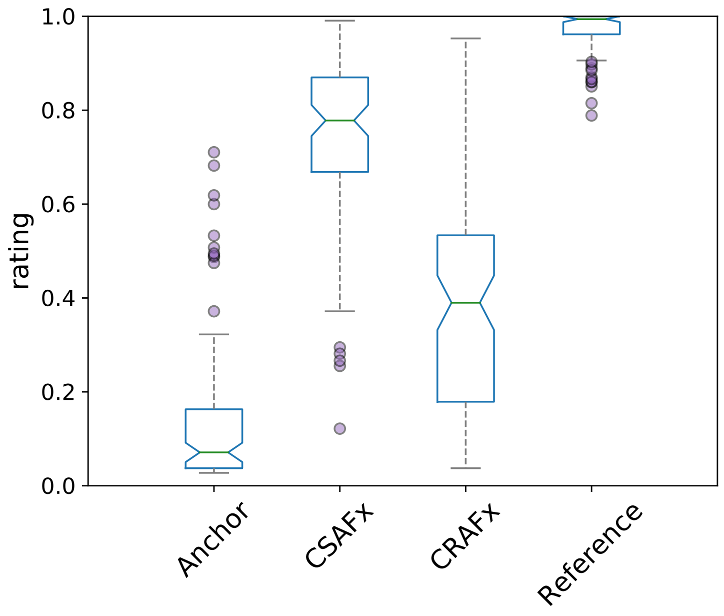
For the spring reverb task, the mean *mae* and *mfcc_cosine* values between input and target waveforms are 0.22 and 0.34, respectively. In the same way, we can see a similar matching to the waveform, this based on the improvement of the *mae* values. Furthermore, based on the results of *mfcc_cosine*, it can be seen that only *CSAFx* is capable of improving the values of the dry recordings. For both *plate* and *spring* reverb tasks, the latter is further confirmed since the mean *MSE* values between input and target waveforms are 9.64 and 41.29, respectively.

The results of the listening test can be seen in Fig. 8.7 as a notched box plot. The end of the boxes represents the first and third quartiles, the end of the notches represents a 95% confidence interval, the green line depicts the median rating and the circles represent outliers. As expected, both anchor and reference have the lowest and highest median respectively. It is evident that for both *plate* and *spring* reverb tasks, *CSAFx* is rated highly whereas *CRAFx* achieves low perceptual ratings when modeling the reverberation tasks.

The perceptual findings confirm the results obtained with the *loss*, *mae* and *mfcc_cosine* metrics and likewise, *plate* models have a better matching that *spring* reverberators. These results are due to the fact that *plate* reverb samples correspond to a digital emulation of a plate reverberator, whereas *spring* reverb samples correspond to an analog reverb tank. Therefore, we interpret that *spring* reverb samples represent a much more difficult task to model.

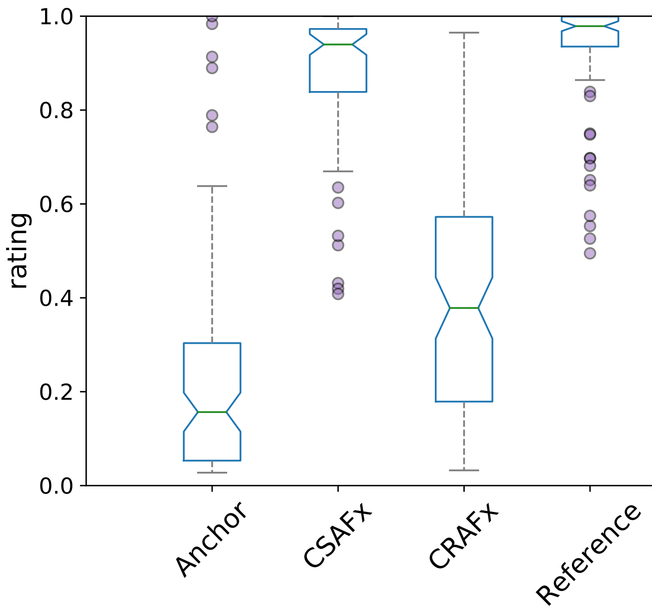


(a)

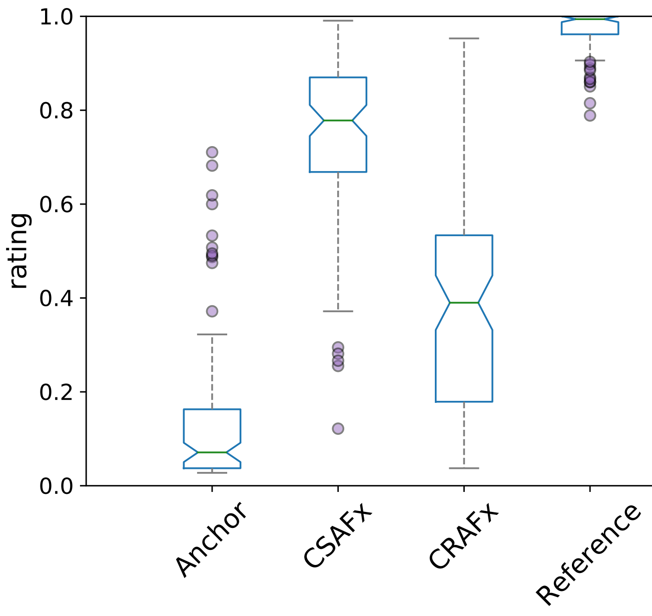


(b)

Figure 8.6: Box plot showing the rating results of the listening tests. From top to bottom: *plate* and *spring* reverb tasks.



(a)



(b)

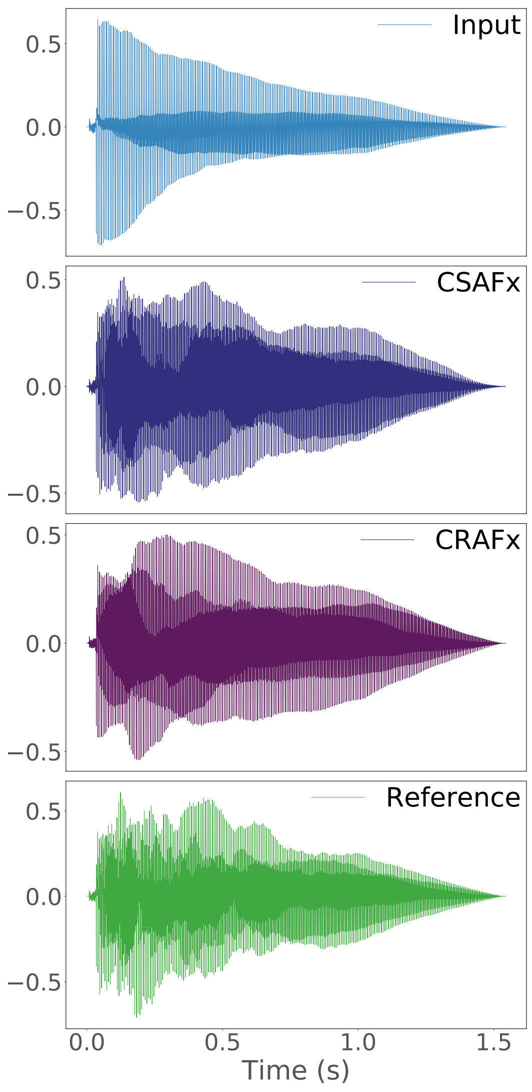
Figure 8.7: Box plot showing the rating results of the listening tests. From top to bottom: *plate* and *spring* reverb tasks.

Furthermore, the perceptual ratings and objective metric values for *spring* do not represent a significant decrease of performance, nevertheless, the modeling of *spring* late reflections could be further explored via a larger number of filters, different loss weights or input frame sizes. Furthermore, as mentioned in Section 7.2, a statistical significance analysis and post-experiment selection of subjects may be required.

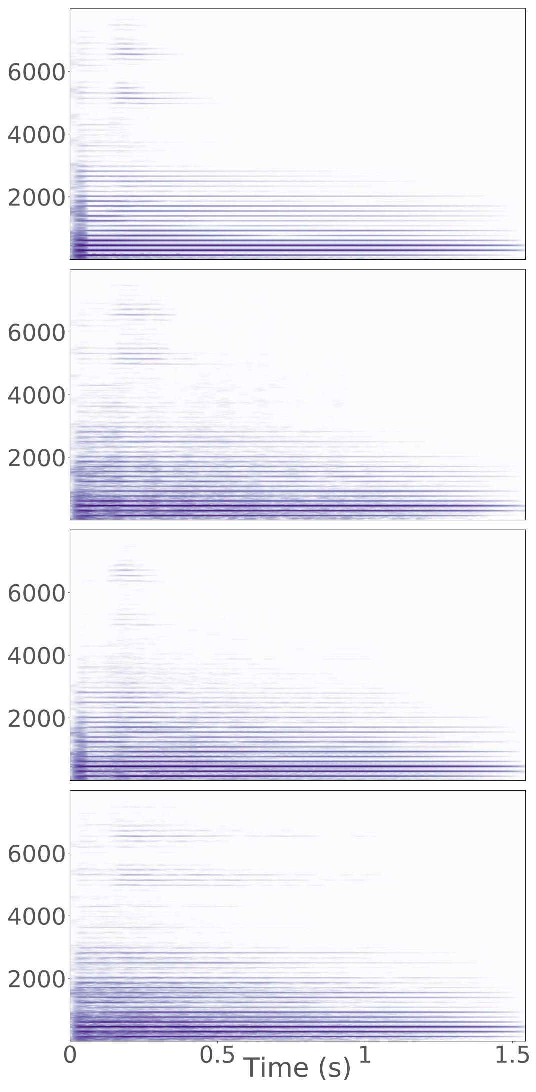
For both reverb tasks and from the test subset, Figs. 8.8 and 8.9 show selected input, target, and output waveforms together with their respective spectrograms. From the spectrograms, the respective noisy and diffused reflections of the *plate* and the *spring* are noticeable. Based on the waveforms from Figs. 8.8a and 8.9a, in order to more closely display the performance of each model and for both modeling tasks, Figs. 8.10 and 8.11 show shorter segments and their corresponding FFT. As expected, it can be seen that *CSAFx* matches very closely the target in the time and frequency domains.

Overall, the initial onset responses are being modeled more accurately, whereas the late reflections differ more prominently in the case of the *spring*, which as mentioned, in all the models it presents a higher loss. From Figs. 8.10 and 8.11 it is remarkable that the models are introducing specific reflections that are not present in the input waveforms which closely match those of the respective targets. Also, from Figs. 8.10b and 8.11b it can be seen that *CRAFx* fails to match the high frequencies of the target, which goes along with the reported objective and perceptual scores. For *CSAFx*, the differences in the time and frequency domains in relation to the target, also correspond to the obtained *loss* values, therefore, a further exploration of the loss weights α_1 and α_2 can be conducted.

Finally, Fig. 8.12 displays the functioning of *CSAFx* by showing internal plots when processing the frame from Fig. 8.11. The selected rows were chosen in order to illustrate how the network models the direct sound and reflections. It shows how the model processes the input frame into the frequency band decomposition $\mathbf{R}$ and learns a set of sparse FIR filters $\hat{\mathbf{Z}}_3$ for each frequency band. Then, the frequency dependent reverberation response $\hat{\mathbf{X}}_{1.1}$ is obtained by applying the learned sparse FIR filters and envelopes to $\mathbf{R}$. The nonlinear transformation of the direct sound $\hat{\mathbf{X}}_{1.2}$ is accomplished through the learnt waveshapers from DNN-SAAF. These two representations are added together via the time-varying mixing gains from SE-LSTM, which is fed to the last layer so the audio waveform is reconstructed in the same manner as the front-end that decomposed it.

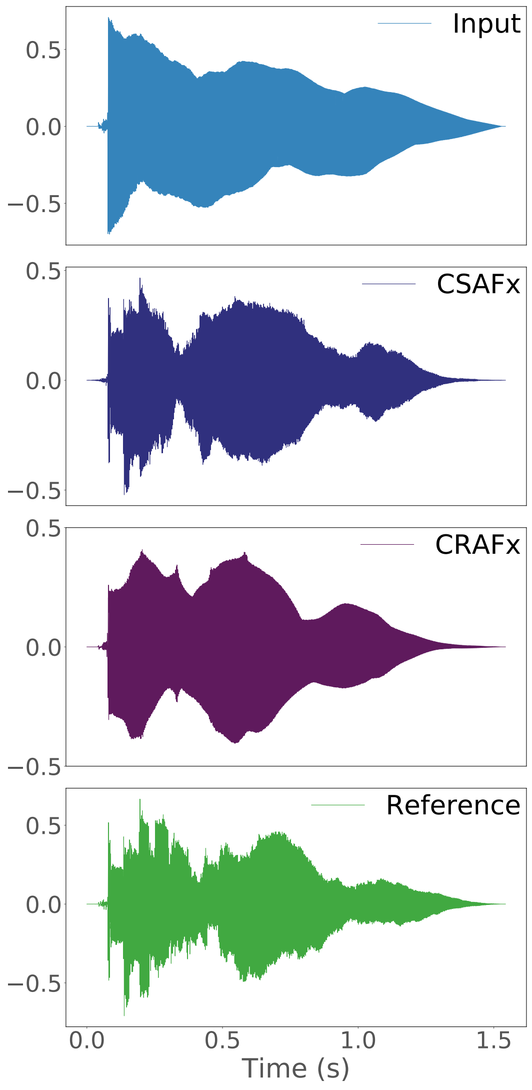


(a)

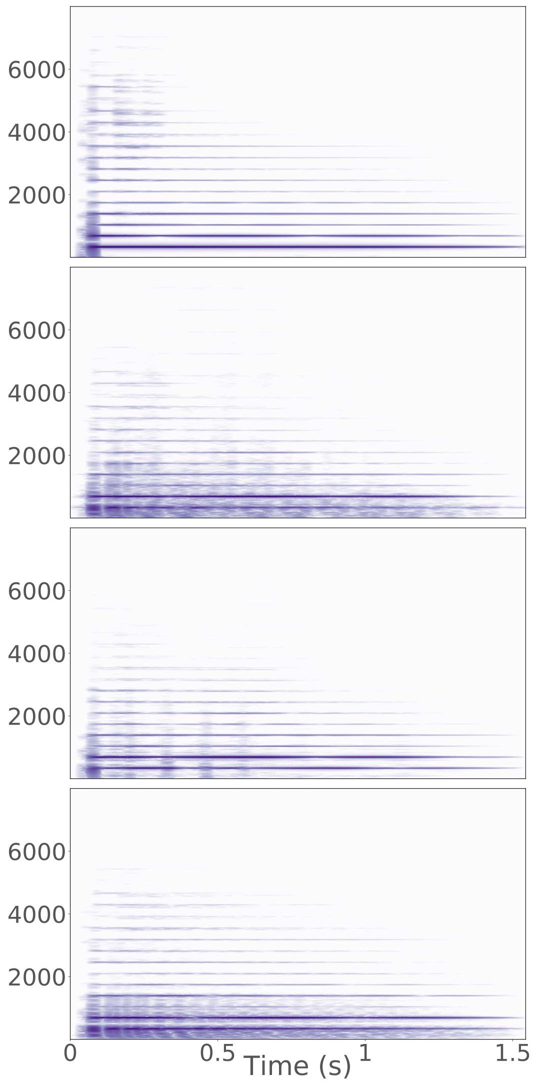


(b)

Figure 8.8: Results with a selected sample from the test dataset for the **plate** reverb task. Figs. 8.8a and 8.8b show the waveforms and their respective spectrograms. Vertical axes represent amplitude and frequency (Hz) respectively.

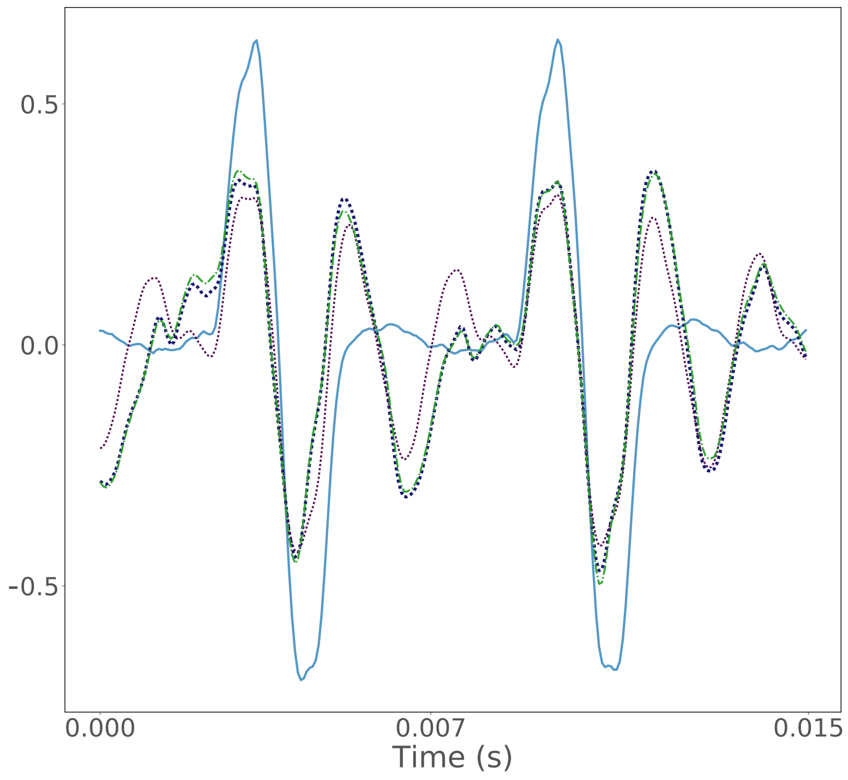


(a)

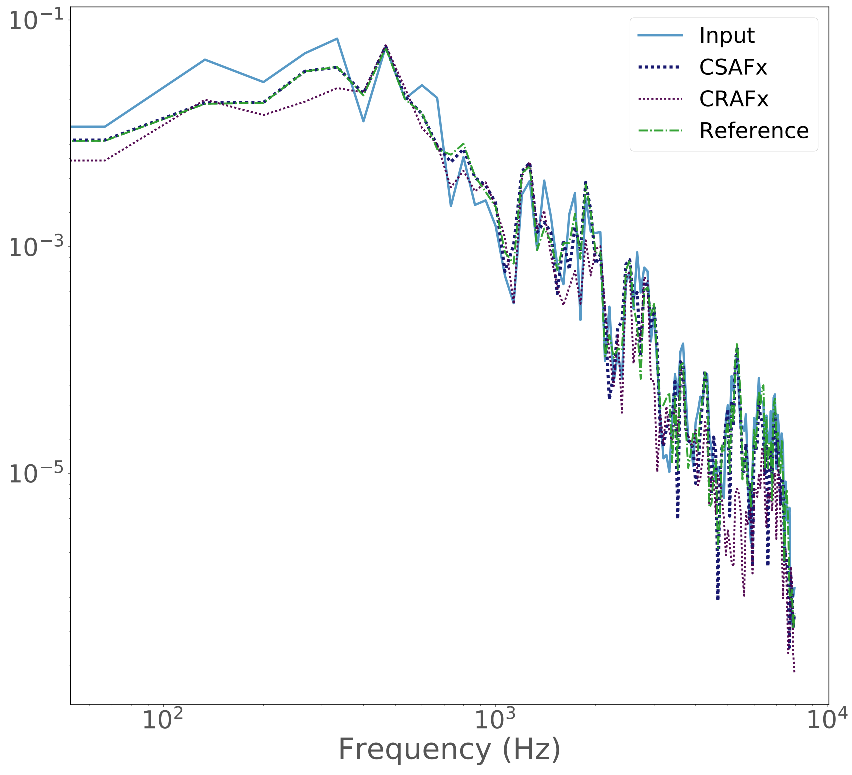


(b)

Figure 8.9: Results with a selected sample from the test dataset for the **spring** reverb task. Figs. 8.9a and 8.9b show the waveforms and their respective spectrograms. Vertical axes represent amplitude and frequency (Hz) respectively.

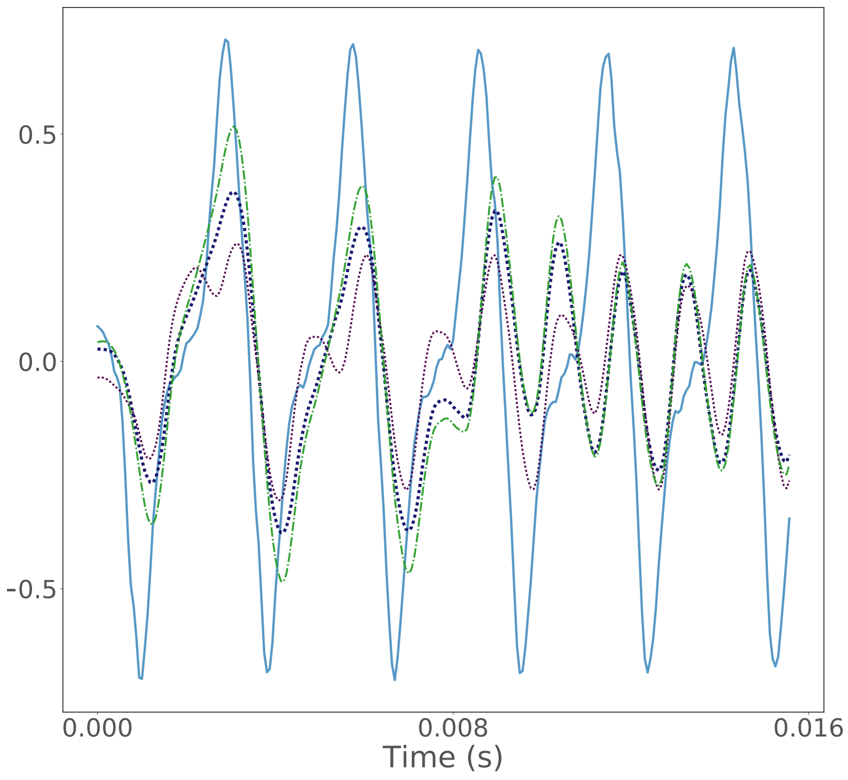


(a)

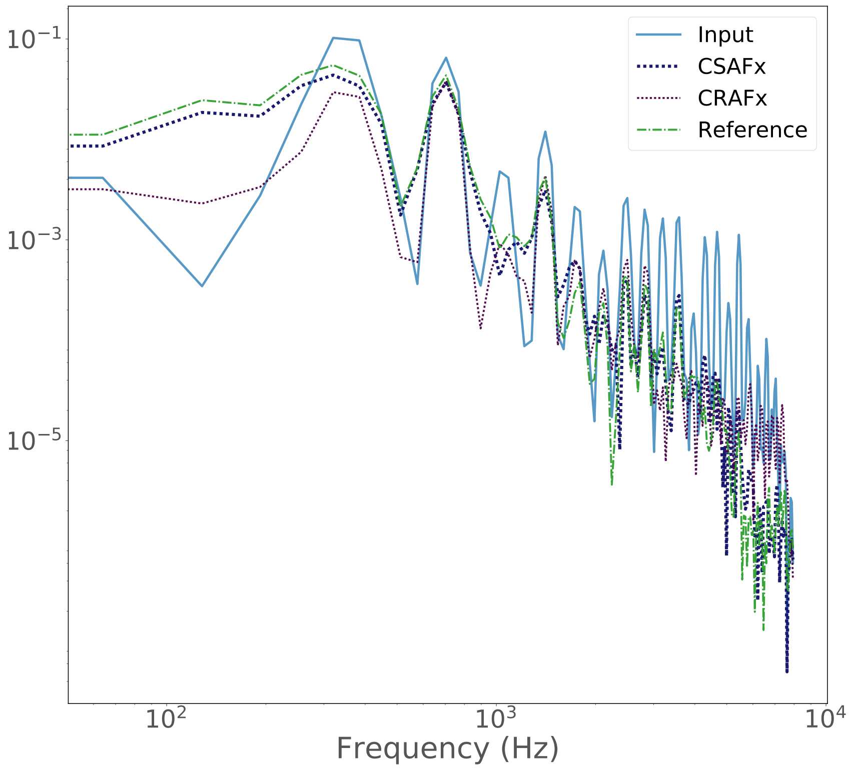


(b)

Figure 8.10: *Plate* reverb. For the test sample from Fig. 8.8, a segment of the respective waveforms and its FFT. Vertical axes represent amplitude and magnitude, respectively.

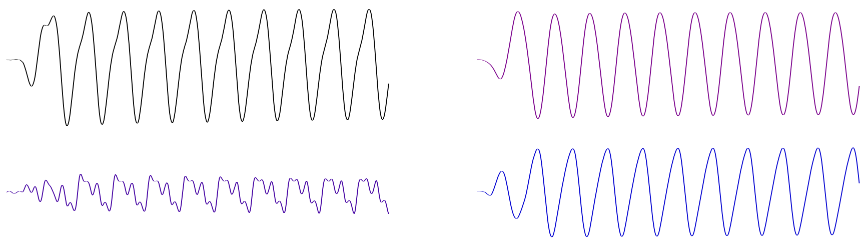


(a)

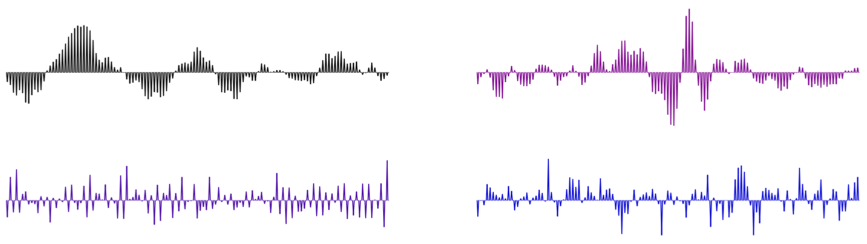


(b)

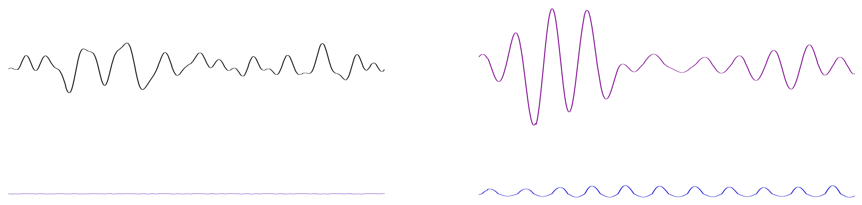
Figure 8.11: *Spring* reverb. For the test sample from Fig. 8.9, a segment of the respective waveforms and its FFT. Vertical axes represent amplitude and magnitude, respectively.



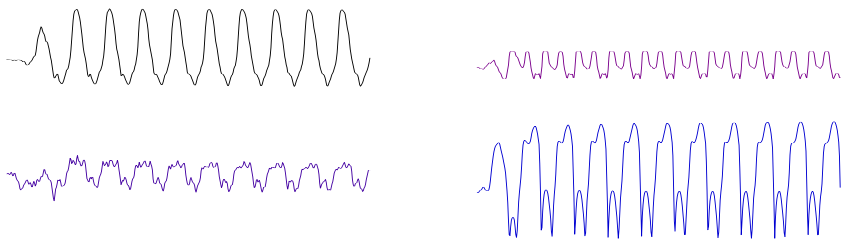
(a)



(b)



(c)



(d)

Figure 8.12: Various internal plots for *CSAFx* from the test sample from Fig. 8.9, which corresponds to the *spring* reverb task. Fig. 8.12a 4 selected rows from the frequency band decomposition $\mathbf{R}$. Fig. 8.12b from $\hat{\mathbf{Z}}_3$, corresponding 4 sparse FIR filters learned by the latent-space. Following the filter bank architecture, Figs. 8.12c and 8.12d show the corresponding 4 rows from $\hat{\mathbf{X}}_{1.1}$ and $\hat{\mathbf{X}}_{1.2}$ respectively. Vertical axes are unitless and horizontal axes are time.

8.4 CONCLUSION

In this chapter, we introduced *CSAFx*: a signal processing-informed deep learning architecture for modeling artificial reverberators.

For this architecture we proposed the SFIR layer, therefore exploring the capabilities of DNNs to learn the coefficients of sparse FIR filters. Likewise, we introduced the SE-LSTM block in order to allow a DNN to learn time-varying mixing gains, which are used by *CSAFx* to dynamically mix the direct sound and the respective reflections. Thus introducing a more explainable network which also outperforms the previous RNN-based model.

We explore whether a deep learning architecture is able to emulate *plate* and *spring* reverberators and we measure the performance of the model through a listening test. We show *CSAFx* matching the reverberant responses of these electromechanical audio processors.

Listening test results and perceptual-based metrics show that the model emulates closely the electromechanical reverberators and also achieves higher ratings than *CRAFx*, although a statistical analysis of the listening test results is required. The latter corresponds to an audio effects modeling network which, in the previous chapter, has been proven to outperform several DNNs for black-box modeling of audio effects. The proposed architecture represents the first deep learning architecture for black-box modeling of artificial reverberators.

From Table C.1, the computational processing times on both GPU and CPU are significantly higher for *CSAFx*. Since these times were computed using the the non real-time optimized *python* implementation, this higher computational cost could be due to the fact that *CSAFx* contains custom layers, such as SFIR, which have not been optimized within machine learning libraries such as *tensorflow*.

For future work, there is a need for additional systematic comparison between the proposed DNN and current analytical methods for modeling *plate* and *spring* reverb, such as numerical simulation or modal techniques. Also, modeling an actual electromechanical *plate* reverb would provide a deeper insight into the CSAFx performance when modeling *plate* and *spring* reverberators.

The modeling of longer decay times and late reflections can also be investigated since the *plate* and *spring* reverb samples have a fade-out applied in the last 0.5 seconds of the recordings. Parametric models can be explored by including the respective controls as new input training data.

Likewise, the architecture can be further tested by modeling vintage digital reverberators or via convolution-based reverb applications. The latter brings applications within the fields of sound spatialization and room acoustics modeling, where accuracy has crucial role as current acoustic modeling methods underestimate the role of phase in early reflections (Välimäki et al., 2012). Moreover, researching causal models, i.e. without subsequent context frames, could open the way to real-time implementations that can be explored alongside applications beyond audio effects modeling, such as automatic reverberation, mixing and mastering.

9 CONCLUSION

9.1 SUMMARY AND DISCUSSION

This research investigated deep learning architectures for black-box modeling of audio effects. Through the modeling capabilities of DNNs together with the domain knowledge from digital audio effects, we show several deep learning architectures that are capable of recreating the sound, behaviour and main perceptual features of various types of audio effects. Based on quantitative evaluation via objective perceptual-based metrics and qualitative evaluation through subjective listening tests, it can be concluded that we demonstrated the feasibility of DNNs as audio processing blocks in the context of audio effects modeling. This represents an improvement of the state-of-the-art for black-box modeling of several types of audio effects.

In order to achieve the latter, we proposed the following deep learning architectures for audio effects modeling: *CEQ*, *CAFX*, *CRAFx*, *CWAFx* and *CSAFX* (see Appendix D). These models correspond to end-to-end networks, where raw audio is both the input and the output of the systems. Thus, given a respective audio effect target, we showed these networks trained via supervised learning procedures and processing and outputting audio that matches the sonic and perceptual characteristics of the reference audio effect. Therefore we can conclude that these models can learn and apply the intrinsic characteristics of the audio effects modeling task.

Based on current active research in the field, we also performed a systematic comparison of the proposed models with the deep learning architecture *WaveNet*. We demonstrated how various models, such as *CAFx*, *CRAFx* and *CWAFx*, outperform *WaveNet* architectures when modeling nonlinear and time-varying audio effects with short and long-term memory. Furthermore, given that the architectures proposed throughout this thesis are based on specific domain knowledge from signal-processing systems and digital audio effects, we can also conclude that we introduced more explainable networks than models solely based on stack of temporal dilated convolutions. Correspondingly, as shown in previous chapters, we analyzed how the modeling tasks are carried out and we explored what various models, such as *CEQ*, *CRAFx* and *CSAFx*, are actually learning.

In Chapter 4, we introduced *CEQ*: a Convolutional EQ modeling network that we show being capable of performing an audio processing task such as modeling different types of equalizers and filters. The overall structure on which *CEQ* is based corresponds to the main or base architecture on which the subsequently proposed models are built. Therefore, we proposed DNNs based on three parts: adaptive front-end, latent-space DNN and synthesis back-end.

The main characteristic of this type of architecture is as follows: First, via the adaptive front-end the model learns an optimal filter bank decomposition and its corresponding latent representation. Second, based on the given modeling task, this latent representation is modified by the latent-space DNN. Finally, since the front-end also generates a residual connection, which corresponds to a frequency band decomposition of the input audio, this feature map is used by the synthesis back-end together with the modified latent representation. Therefore, the back-end aims to modify the residual connection through the modified latent representation and thus synthesize a waveform based on the specific transformation of the audio effect.

By following this type of architecture, we investigated and introduced deep learning architectures that function as audio processing blocks. The main objective of the proposed DNNs is to apply a learnt transformation to the incoming audio, without losing its resolution or intrinsic qualitative characteristics. This differs from traditional DNN encoding-decoding practices, where the complete input data is encoded into a latent-space and each layer in the decoder has to generate the complete desired output.

First, in Chapter 5 we extended the capabilities of this architecture by introducing *CAFx*: a Convolutional audio effects modeling network for nonlinear and linear audio effects with short-term memory. The main novelty of *CAFx* is the incorporation to the back-end of a processing block that contains dense layers and smooth adaptive activation functions (DNN-SAAF). The SAAFs explicitly act as trainable waveshaping nonlinearities which follow the filter bank architecture and are applied to the modified frequency band decomposition.

The use of waveshapers follows most implementations of nonlinear audio effects and through this domain knowledge we introduced a general-purpose DNN for black-box modeling of linear and nonlinear effect units. Hence we demonstrated the feasibility of trainable waveshaping nonlinearities, such as SAAFs, along with a powerful DNN audio effect modeling network.

We also measured the modeling capabilities of *CAFx* alongside with *WaveNet*, a feedforward variation of the original autoregressive model from Oord et al. (2016) and introduced by Rethage et al. (2018). We showed that both architectures perform similarly and each model accomplished the nonlinear modeling tasks with similar accuracy based on a perceptual-based metric.

In Chapter 6, we built on previous architectures, which have focused solely on linear and nonlinear transformations with short-term memory. We confirmed that a latent-space based on Bi-LSTMs or temporal dilated convolutions is able to learn the long temporal dependencies which characterize effect units such as modulation based effects and compressors. This, together with an input consisting of the current audio frame concatenated with previous and subsequent audio frames.

Therefore we introduced; *CRAFx*, a Convolutional Recurrent audio effects modeling network; and *CWAFx*, a Convolutional and WaveNet audio effects modeling network. These models correspond to two general-purpose deep learning architectures for modeling audio effects with long-term memory. We concluded that both models achieved similar performance and, based on a perceptual-based metric, the models were able to match digital implementations of audio effects with long temporal dependencies such as low-frequency modulations, and to process the audio accordingly.

The main innovation of *CRAFx* and *CWAFx* relates to their latent-space and back-end structure. The main objective of their latent-space is to learn a set of frequency-dependent amplitude modulators. These are obtained by modifying the latent representation learned which corresponds to the current, past and future input frames. The structure of the back-end is informed by the general architecture in which the modulation based effects are implemented in the digital domain, i.e. through the use of modulator signals, filters and delay lines.

In the back-end we also introduced a modification of Squeeze-and-Excitation layers. We incorporated SE layers to allow the models to learn and apply a dynamic gain to each of the feature map channels, which in this case correspond to waveforms following a filter bank architecture, i.e. frequency band decompositions. This permitted the back-end to apply a modulator signal to each frequency band and then scale it further through the SE layers.

Hitherto, it has been shown that the proposed architectures can only model digital implementations of linear, nonlinear and time-varying audio effects. Also, their performance was only measured via perceptual-based objective metrics, such as *ms_mse* and *mfcc_cosine*. Thus, in Chapter 7 we further tested the modeling capabilities of the architectures with listening tests and analog modeling tasks; such as a tube *amplifier*; a transistor-based *limiter*; and the rotating *horn* and *woofer* of a *Leslie speaker* cabinet.

We reported that across all modeling tasks, and based on objective perceptual-based metrics and subjective listening tests, *CRAFx*, and to a lesser extent *CWAFx*, outperform the rest of the models. We demonstrated that these architectures are capable of outputting audio that is virtually indistinguishable from the analog reference processors, although a statistical significance analysis is required. Furthermore, the perceptual ratings coincided with the results of the objective metrics, therefore, we can further validate the results reported in previous chapters where only the objective metrics were taken into account.

Finally, in Chapter 8 we introduced *CSAFx*: a Convolutional recurrent and Sparse filtering audio effects modeling network. *CSAFx* is a signal processing-informed deep learning architecture for modeling artificial reverberators. This model extends *CRAFx* by adding a sparse FIR layer to the latent-space, therefore we explored the capabilities of DNNs to learn the coefficients of sparse FIR filters.

Since early reflections can be modeled via FIR filters with sparsely placed coefficients, the main objective of the latent-space is to learn a set of sparse FIR filters and envelopes for each frequency band. Then, the back-end is in charge of processing the residual connection with the learnt filters and envelopes, thus obtaining the frequency-dependent reverberation response. The synthesis back-end also expands with the introduction of LSTMs to the SE layers. SE-LSTM blocks allowed the model to learn time-varying mixing gains which dynamically mix the frequency band decomposition of the direct sound together with the frequency-dependent reflections. This is informed by the overall architecture in which digital reverberators are implemented.

We tested *CSAFx* by modeling a digital implementation of *plate* reverb and an analog *spring* reverb tank. We also compare this model with the previously best-rated architecture, *CRAFx*. We measured the performance of the models through a listening test and we showed *CSAFx* achieving high perceptual ratings when modeling the characteristic noisy and diffused responses of these electromechanical audio processors.

The main conclusions of this dissertation can be summarised as follows:

- We demonstrated the usefulness of deep neural networks for audio processing tasks such as audio effects modeling. In addition, we also showed the benefit that deep learning architectures can obtain from the domain knowledge of audio signal processing systems.
- Deep learning architectures based on an adaptive front-end, latent-space DNN and synthesis back-end can be trained to learn and modify audio accordingly.
- The convolutional layers of the front-end learn a filter bank for each modeling task. This filter bank architecture is maintained across all the networks and yields models that are more explainable and signal-processing informed.
- A latent-space DNN based on locally connected dense layers can learn frequency-dependent envelopes, which can be used to modify the frequency content of the incoming audio, thus modeling EQ.
- SAAFs can be incorporated into DNNs to function explicitly as trainable waveshapers, thus modeling nonlinear audio processing systems with short-term memory.
- A latent-space DNN based on Bi-LSTMs or stacks of temporal dilated convolutions can learn long temporal dependencies such as modulator signals. This is achieved through an input consisting of the current frame concatenated with past and future context frames.
- Bi-LSTMs slightly outperform temporal dilated convolutions when modeling low-frequency modulations.
- SE layers can function as trainable dynamic gains that are applied to each of the channels in the filter bank architecture.
- SFIR layers can be incorporated into DNNs to learn the coefficients of sparse FIR filters.
- Trainable sparse FIR filters and envelopes can be used to match the response of artificial reverberators.
- SE-LSTM layers can be added to a DNN to act as time-varying mixing gains.
- Models that incorporate a latent-space DNN based on Bi-LSTMs or stacks of temporal dilated convolutions outperform the rest of the architectures across all modeling tasks.
- For reverberation tasks, models that incorporate SFIR layers achieve higher ratings than architectures based solely on RNNs.

9.2 FUTURE WORK

Throughout this thesis, we have identified some key areas to improve or explore in the application of deep learning to audio processing tasks, such as modeling audio effects.

In previous chapters, the proposed models learnt to apply the specific audio transformation by being trained with specific musical instruments, such as electric guitar or bass guitar. Moreover, the selected datasets always corresponded to individual notes of short duration. Thus it is not known whether the proposed architectures would also work with different audio signals, such as speech signals or mixtures of music with various instruments. As future work, generalization capabilities among instruments could be explored with the use of a training data with a wider range of instruments or sounds. In addition, dropout layers or weight or activity regularizers can also be investigated in order to improve the modeling capabilities of networks.

Therefore, considering that the models decreased their performance when processing audio from different instruments (see Section 5.5), a further analysis is required on how these models perform when processing much more complex audio waveforms, such as mixtures of various musical sources; or test signals, such as noise or swept-frequency sinusoids. Also, since we only explore long-term memory transformations for individual notes of short duration, more research is needed on how these architectures perform when processing longer or sustained musical sources.

Parametric models could also be explored as the models are learning a static representation of each audio effect modeling task. Therefore the behaviour of the parameters of the effect units can be modeled by including the respective controls as new input training data.

Causal models, i.e. without subsequent context frames, can also be investigated. This is due to *CRAFX*, *CWAFX* and *CSAFx* using both past and subsequent context input frames. Furthermore, considering that models use 256 ms input and output frames to learn long temporal dependencies, more research is needed on how to improve the inherent latency of these architectures.

Moreover, researching causal models that use shorter input frame sizes could open the way to low-latency and real-time implementations. Although the proposed models currently operate via an offline python implementation, real-time models could be further explored since processing times are already close to real-time temporal constraints (see Appendix C). Thus, real-time applications would benefit significantly from the exploration of DNNs to model transformations that involve long-term memory.

More research can be done on the weights learnt by the latent-space DNNs along with a deeper analysis of the filters learnt by the convolutional layers of the front-end. Further investigation of these weights may yield more explainable models. In addition, these weights can be modified during inference to alter the way the input audio is transformed, therefore new transformations could be achieved which would not be possible by using common analog or digital audio processors.

An optimization of the number of filters or channels within the models can be investigated. This, along with more fine-tuned loss functions, such as differentiable perceptual-based metrics. Also, to more closely follow the domain knowledge from modulation based signal processing systems, feedback delay lines within a DNN framework can be researched instead of the feedforward delay lines present in *CRAFx* and *CWAFx*. The latter corresponds to a key area for future work, as differentiable DSP systems, as proposed by Engel et al. (2020), could be incorporated into deep learning models to perform audio processing tasks more effectively.

In this thesis we successfully researched and proposed DNNs that model various types of audio effects, however these architectures could be further tested and extended to model other types of audio processors. For example; nonlinear processors, such as *tape saturation*, *exciters* or *enhancers*; or dynamic range processors, such as *de-esser* or *noise gates*. Also audio effects with long temporal dependencies that are based on echo, such as *feedback delay*, *slapback delay* or *tape-based delay*.

CRAFx and *CWAFx* are designed to model time-varying audio effects driven by low-frequency modulator signals or envelopes, however stochastic effects, i.e. audio processors driven by noise, can also be explored. For instance, a noise generator can be included in the synthesis back-end of these networks which can be scaled via SE or SE-LSTM layers. Furthermore, the modeling capabilities of these architectures can be additionally tested with time-varying effects driven by carrier signals, such as *modulation vocoder*.

Completely different families of effects can also be investigated. This includes *audio-morphing*; time-frequency transformations such as *phase vocoder* effects; time-segment processors such as *time stretching*, *pitch shifting*, *time shuffling* and *granulation*; spatial audio effects such as *3D sound localization* or *room acoustics modeling*.

Adaptive digital audio effects, where low-level and perceptual features are extracted and mapped for the implementation of inter-channel cross-adaptive systems can also be explored. Given an adaptive audio effects task, this mapping of sound features to control the parameters of other processors can be investigated by jointly training various of the proposed architectures.

Possible applications of adaptive audio effects are within the field of automatic mixing and mastering. Automatic linear and nonlinear processing can be investigated for an automatic mixing task, such as automatic EQ, compression, or reverberation. Furthermore, style-learning of a specific sound engineer could be explored, where a network is trained with several tracks mixed by a sound engineer and finds a generalization from the engineer's mixing practices. Also, automatic post-production for a specific instrument across one or several genres could be analyzed and implemented by the models.

It is worth noting the immense benefit that generative music could obtain from deep learning architectures for intelligent music production. The proposed models could be used together with generative music methods in order to function as automatic mixing or mastering systems.

Applications beyond audio effects modeling and intelligent music production can also be investigated, for instance signal restoration methods such as *undistortion*, *denoising* and *dereverberation*.

Overall, this thesis ventured for the application of artificial intelligence to the manipulation of musical signals. We focused on recreating the sound crafted by electronic engineers more than 80 years ago, and while this has enormous merit on its own, I believe this research could be just the start of a new branch of digital sound processors. Analogous to the great transformation that music production had with the dawn of digital signal processing, an exciting future awaits for audio processors based on neural networks, from intelligent assistants to new effects yet to be discovered.

BIBLIOGRAPHY

Jonathan S Abel and David P Berners. A technique for nonlinear system measurement. In *121st Audio Engineering Society Convention*, 2006.

Jonathan S Abel, David P Berners, Sean Costello, and Julius O Smith. Spring reverb emulation using dispersive allpass filters in a waveguide structure. In *121st Audio Engineering Society Convention*, 2006.

Jonathan S Abel, David P Berners, and Aaron Greenblatt. An emulation of the emt 140 plate reverberator using a hybrid reverberator structure. In *127th Audio Engineering Society Convention*, 2009.

Shun-ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4-5):185–196, 1993.

Jérôme Antoni and Johan Schoukens. A comprehensive study of the bias and variance of frequency-response-function measurements: Optimal window selection and overlapping strategies. *Automatica*, 43(10):1723–1736, 2007.

Kevin Arcas and Antoine Chaigne. On the quality of plate reverberation. *Applied Acoustics*, 71(2):147–156, 2010.

Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: circumventing defenses to adversarial examples. In *International Conference on Machine Learning*, 2018.

Shaojie Bai, Jeremy Zico Kolter, and Vladlen Koltun. Convolutional sequence modeling revisited. In *6th International Conference on Learning Representations (ICLR)*, 2018.

Daniele Barchiesi and Joshua D. Reiss. Reverse engineering of a mix. *Journal of the Audio Engineering Society*, 58(7/8):563–576, 2010.

Stefan Bilbao. A digital plate reverberation algorithm. *Journal of the Audio Engineering Society*, 55(3):135–144, 2007.

Stefan Bilbao. *Numerical sound synthesis*. Wiley Online Library, 2009.

Stefan Bilbao. Numerical simulation of spring reverberation. In *16th International Conference on Digital Audio Effects (DAFx-13)*, 2013.

Stefan Bilbao and Julian Parker. A virtual model of spring reverberation. *IEEE Transactions on Audio, Speech and Language Processing*, 18(4):799–808, 2009.

Stefan Bilbao, Kevin Arcas, and Antoine Chaigne. A physical model for plate reverberation. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2006.

Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

Merlijn Blaauw and Jordi Bonada. A neural parametric singing synthesizer. In *Interspeech*, 2017.

Ólafur Bogason and Kurt James Werner. Modeling circuits with operational transconductance amplifiers using wave digital filters. In *20th International Conference on Digital Audio Effects (DAFx-17)*, 2017.

Chi-Tsong Chen. *Linear system theory and design*. Oxford University Press, Inc., 1998.

Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cuDNN: Efficient primitives for deep learning. *CoRR*, abs / 1410.0759, 2014.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Francois Chollet. *Deep Learning with Python*. Manning Publications Co., 2018.

Jan Chorowski, Ron J Weiss, Samy Bengio, and Aäron van den Oord. Unsupervised speech representation learning using wavenet autoencoders. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 27(12):2041–2053, 2019.

Eero-Pekka Damskägg, Lauri Juvela, Etienne Thuillier, and Vesa Välimäki. Deep learning for tube amplifier emulation. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2019.

Brecht De Man, Joshua D Reiss, and Ryan Stables. Ten years of automatic mixing. In *Proceedings of the 3rd Workshop on Intelligent Music Production*, 2017.

Giovanni De Sanctis and Augusto Sarti. Virtual analog modeling in the wave-digital domain. *IEEE Transactions on Audio, Speech, and Language Processing*, 2009.

Junqi Deng and Yu-Kwong Kwok. Automatic chord estimation on seventhsbass chord vocabulary using deep neural network. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016.

Sander Dieleman and Benjamin Schrauwen. End-to-end learning for music audio. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014.

Michele Ducceschi and Craig J Webb. Plate reverberation: Towards the development of a real-time physical model for the working musician. In *International Congress on Acoustics (ICA)*, 2016.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159, 2011.

Simon Durand, Juan P Bello, Bertrand David, and Gaël Richard. Downbeat tracking with multiple features and deep neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015.

Douglas Eck and Juergen Schmidhuber. A first look at music composition using lstm recurrent neural networks. *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*, 103, 2002.

Felix Eichas and Udo Zölzer. Black-box modeling of distortion circuits with block-oriented models. In *19th International Conference on Digital Audio Effects (DAFx-16)*, 2016.

Felix Eichas and Udo Zölzer. Virtual analog modeling of guitar amplifiers with wiener-hammerstein models. In *44th Annual Convention on Acoustics*, 2018.

Felix Eichas, Marco Fink, Martin Holters, and Udo Zölzer. Physical modeling of the mxr phase 90 guitar effect pedal. In *17th International Conference on Digital Audio Effects (DAFx-14)*, 2014.

Felix Eichas, Etienne Gerat, and Udo Zölzer. Virtual analog modeling of dynamic range compression systems. In *142nd Audio Engineering Society Convention*, 2017.

Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Mohammad Norouzi, Douglas Eck, and Karen Simonyan. Neural audio synthesis of musical notes with wavenet autoencoders. *34th International Conference on Machine Learning*, 2017.

Jesse Engel, Lamtharn Hantrakul, Chenjie Gu, and Adam Roberts. DDSP: Differentiable digital signal processing. In *8th International Conference on Learning Representations (ICLR)*, 2020.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

Angelo Farina. Simultaneous measurement of impulse response and distortion with a swept-sine technique. In *108th Audio Engineering Society Convention*, 2000.

Xue Feng, Yaodong Zhang, and James Glass. Speech feature denoising and dereverberation via deep autoencoders for noisy reverberant speech recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2014.

Benjamin Friedlander and Boaz Porat. The modified Yule-Walker method of ARMA spectral estimation. *IEEE Transactions on Aerospace and Electronic Systems*, (2):158–173, 1984.

Zhouyu Fu, Guojun Lu, Kai Ming Ting, and Dengsheng Zhang. A survey of audio-based music classification and annotation. *IEEE Transactions on Multimedia*, 13 (2):303–319, 2010.

Todor Ganchev, Nikos Fakotakis, and George Kokkinakis. Comparative evaluation of various mfcc implementations on the speaker verification task. In *International Conference on Speech and Computer*, 2005.

Patrick Gaydecki. *Foundations of digital signal processing: theory, algorithms and hardware design*, volume 15. Iet, 2004.

Etienne Gerat, Felix Eichas, and Udo Zölzer. Virtual analog modeling of a urei 1176ln dynamic range control system. In *143rd Audio Engineering Society Convention*, 2017.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. *Learning to forget: Continual prediction with* LSTM. IET, 1999.

Dimitrios Giannoulis, Michael Massberg, and Joshua D Reiss. Parameter automation in a dynamic range compressor. *Journal of the Audio Engineering Society*, 61 (10):716–726, 2013.

Pere Lluís Gilabert Pinal, Gabriel Montoro López, and Eduardo Bertran Albertí. On the wiener and hammerstein models for power amplifier predistortion. In *IEEE Asia-Pacific Microwave Conference*, 2005.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *the 13th International Conference on Artificial Intelligence and Statistics*, 2010.

Luke B Godfrey and Michael S Gashler. A continuum among logarithmic, linear, and exponential functions, and its potential to improve generalization in neural networks. In *7th IEEE International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, 2015.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18 (5-6):602–610, 2005.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2013.

Aaron B Greenblatt, Jonathan S Abel, and David P Berners. A hybrid reverberation crossfading technique. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2010.

Sina Hafezi and Joshua D. Reiss. Autonomous multitrack equalization based on masking reduction. *Journal of the Audio Engineering Society*, 63(5):312–323, 2015.

Anna Hagenblad. *Aspects of the identification of Wiener models*. PhD thesis, Linköpings Universitet, 1999.

Stefan L Hahn. *Hilbert transforms in signal processing*, volume 2. Artech House Boston, 1996.

Philippe Hamel and Douglas Eck. Learning features from music audio with deep belief networks. In *11th International Society for Music Information Retrieval Conference (ISMIR)*, 2010.

Philippe Hamel, Matthew EP Davies, Kazuyoshi Yoshii, and Masataka Goto. Transfer learning in MIR: Sharing learned latent representations for music audio classification and similarity. In *14th International Society for Music Information Retrieval Conference (ISMIR)*, 2013.

Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

Kun Han, Yuxuan Wang, DeLiang Wang, William S Woods, Ivo Merks, and Tao Zhang. Learning spectral mapping for speech dereverberation and denoising. *IEEE Transactions on Audio, Speech and Language Processing*, 23(6):982–992, 2015.

Yoonchang Han, Jaehun Kim, and Kyogu Lee. Deep convolutional neural networks for predominant instrument recognition in polyphonic music. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(1):208–221, 2016.

Aki Härmä, Matti Karjalainen, Lauri Savioja, Vesa Välimäki, Unto K Laine, and Jyri Huopaniemi. Frequency-warped signal processing for audio applications. *Journal of the Audio Engineering Society*, 48(11):1011–1031, 2000.

Scott H Hawley, Benjamin Colburn, and Stylianos I Mimilakis. SignalTrain: Profiling audio compressors with deep neural networks. In *147th Audio Engineering Society Convention*, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Thomas Hélie. On the use of volterra series for real-time simulations of weakly nonlinear analog audio devices: Application to the moog ladder filter. In *9th International Conference on Digital Audio Effects (DAFx-06)*, 2006.

Clifford A Henricksen. Unearthing the mysteries of the leslie cabinet. *Recording Engineer/Producer Magazine*, 1981.

Jorge Herrera, Craig Hanson, and Jonathan S Abel. Discrete time emulation of the leslie speaker. In *127th Audio Engineering Society Convention*, 2009.

Marcel Hilsamer and Stephan Herzog. A statistical approach to automated offline dynamic processing in the audio mastering process. In *17th International Conference on Digital Audio Effects (DAFx-14)*, 2014.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Martin Holters and Julian D Parker. A combined model for a bucket brigade device and its input and output filters. In *21st International Conference on Digital Audio Effects (DAFx-17)*, 2018.

Martin Holters and Udo Zölzer. Physical modelling of a wah-wah effect pedal as a case study for application of the nodal dk method to circuits with variable parts. In *14th International Conference on Digital Audio Effects (DAFx-11)*, 2011.

Le Hou, Dimitris Samaras, Tahsin M Kurc, Yi Gao, and Joel H Saltz. Neural networks with smooth adaptive activation functions for regression. *arXiv preprint arXiv:1608.06557*, 2016.

Le Hou, Dimitris Samaras, Tahsin M Kurc, Yi Gao, and Joel H Saltz. Convnets with smooth adaptive activation functions for regression. In *20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Allen Huang and Raymond Wu. Deep learning for music. *CoRR*, abs / 1606.04930, 2016.

Eric J Humphrey and Juan P Bello. Rethinking automatic chord recognition with convolutional neural networks. In *11th International Conference on Machine Learning and Applications*, 2012.

Eric J Humphrey and Juan P Bello. From music audio to chord tablature: Teaching deep convolutional networks to play guitar. In *IEEE international conference on acoustics, speech and signal processing (ICASSP)*, 2014.

Antti Huovilainen. Enhanced digital models for analog modulation effects. In *8th International Conference on Digital Audio Effects (DAFx-05)*, 2005.

Leland B Jackson. Frequency-domain Steiglitz-McBride method for least-squares IIR filter design, ARMA modeling, and periodogram smoothing. *IEEE Signal Processing Letters*, 15:49–52, 2008.

Hanna Järveläinen and Matti Karjalainen. Reverberation modeling using velvet noise. In *30th Audio Engineering Society International Conference*, 2007.

Nicholas Jillings, Brecht De Man, David Moffat, and Joshua D Reiss. Web Audio Evaluation Tool: A browser-based listening test environment. In *12th Sound and Music Computing Conference*, 2015.

Jean-Marc Jot and Antoine Chaigne. Digital delay networks for designing artificial reverberators. In *90th Audio Engineering Society Convention*, 1991.

Matti Karjalainen, Teemu Mäki-Patola, Aki Kanerva, and Antti Huovilainen. Virtual air guitar. *Journal of the Audio Engineering Society*, 54(10):964–980, 2006.

Roope Kiiski, Fabián Esqueda, and Vesa Välimäki. Time-variant gray-box modeling of a phaser pedal. In *19th International Conference on Digital Audio Effects (DAFx-16)*, 2016.

Taejun Kim, Jongpil Lee, and Juhan Nam. Sample-level CNN architectures for music auto-tagging using raw waveforms. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2018.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations (ICLR)*, 2015.

David M Koenig. *Spectral analysis of musical sounds with emphasis on the piano*. OUP Oxford, 2014.

Filip Korzeniowski and Gerhard Widmer. Feature learning for chord recognition: The deep chroma extractor. In *17th International Society for Music Information Retrieval Conference (ISMIR)*, 2016.

Oliver Kröning, Kristjan Dempwolf, and Udo Zölzer. Analysis and simulation of an analog guitar compressor. In *14th International Conference on Digital Audio Effects (DAFx-11)*, 2011.

Walter Kuhl. The acoustical and technological properties of the reverberation plate. *E. B. U. Review*, 49, 1958.

Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. *Neural networks: Tricks of the trade*, pages 9–48, 2012.

Honglak Lee, Peter Pham, Yan Largman, and Andrew Y Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in neural information processing systems*, pages 1096–1104, 2009.

Jongpil Lee, Jiyoung Park, Keunhyoung Luke Kim, and Juhan Nam. SampleCNN: End-to-end deep convolutional neural networks using very small filters for music classification. *Applied Sciences*, 8(1):150, 2018.

Keun Sup Lee, Nicholas J Bryan, and Jonathan S Abel. Approximating measured reverberation using a hybrid fixed/switched convolution structure. In *13th International Conference on Digital Audio Effects (DAFx-10)*, 2010.

Teck Yian Lim, Raymond A Yeh, Yijia Xu, Minh N Do, and Mark Hasegawa-Johnson. Time-frequency networks for audio super-resolution. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

Zheng Ma, Joshua D Reiss, and Dawn AA Black. Implementation of an intelligent equalization tool using yule-walker for music mixing and mastering. In *134th Audio Engineering Society Convention*, 2013.

Zheng Ma, Brecht De Man, Pedro DL Pestana, Dawn AA Black, and Joshua D Reiss. Intelligent multitrack dynamic range compression. *Journal of the Audio Engineering Society*, 63(6):412–426, 2015.

Jaromír Mačák. Simulation of analog flanger effect using BBD circuit. In *19th International Conference on Digital Audio Effects (DAFx-16)*, 2016.

Jacob A Maddams, Saoirse Finn, and Joshua D Reiss. An autonomous method for multi-track dynamic range compression. In *15th International Conference on Digital Audio Effects (DAFx-12)*, 2012.

EP MatthewDavies and Sebastian Böck. Temporal convolutional networks for musical audio beat tracking. In *27th IEEE European Signal Processing Conference (EUSIPCO)*, 2019.

Daniel Matz, Estefanía Cano, and Jakob Abeßer. New sonorities for early jazz recordings using sound source separation and automatic mixing tools. In *16th International Society for Music Information Retrieval Conference (ISMIR)*, 2015.

Josh H McDermott and Eero P Simoncelli. Sound texture perception via statistics of the auditory periphery: evidence from sound synthesis. *Neuron*, 71, 2011.

Martin McKinney and Jeroen Breebaart. Features for audio and music classification. In *4th International Society for Music Information Retrieval Conference (ISMIR)*, 2003.

Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron Courville, and Yoshua Bengio. SampleRNN: An unconditional end-to-end neural audio generation model. In *5th International Conference on Learning Representations*. ICLR, 2017.

Stylianos I Mimilakis, Konstantinos Drossos, Andreas Floros, and Dionysios Katerelos. Automated tonal balance enhancement for audio mastering applications. In *134th Audio Engineering Society Convention*, 2013.

Stylianos I Mimilakis, Konstantinos Drossos, Tuomas Virtanen, and Gerald Schuller. Deep neural networks for dynamic range compression in mastering applications. In *140th Audio Engineering Society Convention*, 2016.

Stephan Möller, Martin Gromowski, and Udo Zölzer. A measurement technique for highly nonlinear transfer functions. In *5th International Conference on Digital Audio Effects (DAFx-02)*, 2002.

Brian CJ Moore. *An introduction to the psychology of hearing*. Brill, 2012.

James A Moorer. About this reverberation business. *Computer music journal*, pages 13–28, 1979.

Noam Mor, Lior Wolf, Adam Polyak, and Yaniv Taigman. A universal music translation network. In *7th International Conference on Learning Representations (ICLR)*, 2019.

M Narasimha and A Peterson. On the computation of the discrete cosine transform. *IEEE Transactions on Communications*, 26(6):934–936, 1978.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *CoRR abs/1609.03499*, 2016.

Douglas O'shaughnessy. *Speech Communications: Human And Machine*. Universities press, 1987.

Jyri Pakarinen and David T Yeh. A review of digital techniques for modeling vacuum-tube guitar amplifiers. *Computer Music Journal*, 33(2):85–100, 2009.

Bryan Pardo, David Little, and Darren Gergle. Building a personalized audio equalizer interface with transfer learning and active learning. In *2nd International*

ACM Workshop on Music Information Retrieval with User-Centered and Multimodal Strategies, 2012.

Julian Parker. Efficient dispersion generation structures for spring reverb emulation. *EURASIP Journal on Advances in Signal Processing*, 2011a.

Julian Parker. A simple digital model of the diode-based ring-modulator. In *14th International Conference on Digital Audio Effects (DAFx-11)*, 2011b.

Julian Parker and Stefan Bilbao. Spring reverberation: A physical perspective. In *12th International Conference on Digital Audio Effects (DAFx-09)*, 2009.

Julian Parker and Fabian Esqueda. Modelling of nonlinear state-space systems using a deep neural network. In *22nd International Conference on Digital Audio Effects (DAFx-19)*, 2019.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, 2013.

Roy D Patterson. Auditory filters and excitation patterns as representations of frequency resolution. *Frequency selectivity in hearing*, 1986.

Jussi Pekonen, Tapani Pihlajamäki, and Vesa Välimäki. Computationally efficient hammond organ synthesis. In *14th International Conference on Digital Audio Effects (DAFx-11)*, 2011.

Enrique Perez-Gonzalez and Joshua D. Reiss. Automatic equalization of multichannel audio using cross-adaptive methods. In *127th Audio Engineering Society Convention*, 2009.

Enrique Perez-Gonzalez and Joshua D Reiss. Automatic mixing. *DAFX: Digital Audio Effects, Second Edition*, pages 523–549, 2011.

Pedro Duarte Leal Gomes Pestana. *Automatic mixing systems using adaptive digital audio effects*. PhD thesis, Universidade Católica Portuguesa, 2013.

George M Phillips and Peter J Taylor. *Theory and applications of numerical analysis*. Elsevier, 1996.

Jordi Pons, Oriol Nieto, Matthew Prockup, Erik Schmidt, Andreas Ehmann, and Xavier Serra. End-to-end learning for music audio tagging at scale. In *31st Conference on Neural Information Processing Systems*, 2017.

Miller Puckette. *The theory and technique of electronic music*. World Scientific Publishing Company, 2007.

Colin Raffel and Julius O Smith. Practical modeling of bucket-brigade device circuits. In *13th International Conference on Digital Audio Effects (DAFx-10)*, 2010.

Jussi Rämö and Vesa Välimäki. Neural third-octave graphic equalizer. In *22nd International Conference on Digital Audio Effects (DAFx-19)*, 2019.

Dale Reed. A perceptual assistant to do sound equalization. In *5th International Conference on Intelligent User Interfaces*, pages 212–218. ACM, 2000.

Joshua D Reiss and Andrew McPherson. *Audio effects: theory, implementation and application*. CRC Press, 2014.

Dario Rethage, Jordi Pons, and Xavier Serra. A wavenet for speech denoising. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

David Ronan, Zheng Ma, Paul Mc Namara, Hatice Gunes, and Joshua D Reiss. Automatic minimisation of masking in multitrack audio using subgroups. *IEEE Transactions on Audio, Speech, and Language processing*, 2018.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2015.

Per Rubak and Lars G Johansen. Artificial reverberation based on a pseudo-random impulse response II. In *106th Audio Engineering Society Convention*, 1999.

Andrew T Sabin and Bryan Pardo. A method for rapid personalization of audio equalization parameters. In *17th ACM International Conference on Multimedia*, 2009.

Jan Schlüter and Sebastian Böck. Musical onset detection with convolutional neural networks. In *6th International Workshop on Machine Learning and Music*, 2013.

Jan Schlüter and Sebastian Böck. Improved musical onset detection with convolutional neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014.

Thomas Schmitz and Jean-Jacques Embrechts. Nonlinear real-time emulation of a tube amplifier with a long short time memory neural-network. In *144th Audio Engineering Society Convention*, 2018.

Manfred R Schroeder and Benjamin F Logan. "Colorless" artificial reverberation. *IRE Transactions on Audio*, (6):209–214, 1961.

Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

Di Sheng and György Fazekas. Automatic control of the dynamic range compressor using a regression model and a reference sound. In *20th International Conference on Digital Audio Effects (DAFx-17)*, 2017.

Di Sheng and György Fazekas. A feature learning siamese model for intelligent control of the dynamic range compressor. In *International Joint Conference on Neural Networks (IJCNN)*, 2019.

Siddharth Sigtia and Simon Dixon. Improved music feature learning with deep neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014.

Siddharth Sigtia, Emmanouil Benetos, Nicolas Boulanger-Lewandowski, Tillman Weyde, Artur S d'Avila Garcez, and Simon Dixon. A hybrid recurrent neural network for music transcription. In *IEEE international conference on acoustics, speech and signal processing (ICASSP)*, 2015.

Siddharth Sigtia, Emmanouil Benetos, and Simon Dixon. An end-to-end neural network for polyphonic piano music transcription. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(5):927–939, 2016.

Julius O Smith. *Introduction to digital filters: with audio applications*, volume 2. W3K Publishing, 2007.

Julius O Smith. *Physical audio signal processing: For virtual musical instruments and audio effects*. W3K Publishing, 2010.

Julius O Smith and Jonathan S Abel. Bark and ERB bilinear transforms. *IEEE Transactions on Speech and Audio Processing*, 7(6):697–708, 1999.

Julius O Smith, Stefania Serafin, Jonathan Abel, and David Berners. Doppler simulation and the leslie. In *5th International Conference on Digital Audio Effects (DAFx-02)*, 2002.

Mirko Solazzi and Aurelio Uncini. Artificial neural networks with adaptive multi-dimensional spline activation functions. In *IEEE International Joint Conference on Neural Networks (IJCNN)*, 2000.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Michael Stein, Jakob Abeßer, Christian Dittmar, and Gerald Schuller. Automatic detection of audio effects in guitar and bass recordings. In *128th Audio Engineering Society Convention*, 2010.

Karl Steinberg. Steinberg virtual studio technology (VST) plug-in specification 2.0 software development kit. *Hamburg: Steinberg Soft-und Hardware GMBH*, 1999.

Stanley Smith Stevens, John Volkmann, and Edwin B Newman. A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, 8(3):185–190, 1937.

Dan Stowell and Mark D Plumbley. Automatic large-scale classification of bird sounds is strongly improved by unsupervised feature learning. *PeerJ*, 2:e488, 2014.

Bob L Sturm, Joao Felipe Santos, Oded Ben-Tal, and Iryna Korshunova. Music transcription modelling and composition using deep learning. In *1st Conference on Computer Simulation of Musical Creativity*, 2016.

Somsak Sukittanon, Les E Atlas, and James W Pitton. Modulation-scale analysis for content identification. *IEEE Transactions on Signal Processing*, 52, 2004.

Tijmen Tieleman and Geoffrey Hinton. RMSprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Aurelio Uncini. Audio signal processing by neural networks. *Neurocomputing*, 55 (3-4):593–625, 2003.

International Telecommunication Union. Recommendation ITU-R BS.1534-1: Method for the subjective assessment of intermediate quality level of coding systems. 2003.

Giacomo Vairetti, Enzo De Sena, Michael Catrysse, Søren Holdt Jensen, Marc Moonen, and Toon van Waterschoot. An automatic design procedure for low-order iir parametric equalizers. *Journal of the Audio Engineering Society*, 66(11):935–952, 2018.

Vesa Välimäki and Joshua D. Reiss. All about audio equalization: Solutions and frontiers. *Applied Sciences*, 6(5):129, 2016.

Vesa Välimäki, Julian Parker, and Jonathan S Abel. Parametric spring reverberation effect. *Journal of the Audio Engineering Society*, 58(7/8):547–562, 2010.

Vesa Välimäki, Julian D Parker, Lauri Savioja, Julius O Smith, and Jonathan S Abel. Fifty years of artificial reverberation. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(5):1421–1448, 2012.

Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In *Advances in Neural Information Processing Systems*, pages 2643–2651, 2013.

Shrikant Venkataramani, Jonah Casebeer, and Paris Smaragdis. Adaptive front-ends for end-to-end source separation. In *31st Conference on Neural Information Processing Systems*, 2017.

Vincent Verfaille, U. Zölzer, and Daniel Arfib. Adaptive digital audio effects (A-DAFx): A new class of sound transformations. *IEEE Transactions on Audio, Speech and Language Processing*, 14(5):1817–1831, 2006.

Xinxi Wang and Ye Wang. Improving content-based and hybrid music recommendation using deep learning. In *22nd International Conference on Multimedia*, pages 627–636. ACM, 2014.

Kurt J Werner, W Ross Dunkel, and François G Germain. A computational model of the hammond organ vibrato/chorus using wave digital filters. In *19th International Conference on Digital Audio Effects (DAFx-16)*, 2016.

Silvin Willemsen, Stefania Serafin, and Jesper R Jensen. Virtual analog simulation and extensions of plate reverberation. In *14th Sound and Music Computing Conference*, 2017.

Alec Wright, Eero-Pekka Damskägg, and Vesa Välimäki. Real-time black-box modelling with recurrent neural networks. In *22nd International Conference on Digital Audio Effects (DAFx-19)*, 2019.

David T Yeh. Automated physical modeling of nonlinear audio circuits for real-time audio effects part II: BJT and vacuum tube examples. *IEEE Transactions on Audio, Speech, and Language Processing*, 20, 2012.

David T Yeh and Julius O Smith. Simulating guitar distortion circuits using wave digital and nonlinear state-space formulations. In *11th International Conference on Digital Audio Effects (DAFx-08)*, 2008.

David T Yeh, Jonathan S Abel, Andrei Vladimirescu, and Julius O Smith. Numerical methods for simulation of guitar distortion circuits. *Computer Music Journal*, 32(2):23–42, 2008.

David T Yeh, Jonathan S Abel, and Julius O Smith. Automated physical modeling of nonlinear audio circuits for real-time audio effects part I: Theoretical development. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(4):728–737, 2010.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*. Springer, 2014.

Zhichen Zhang, Edward Olbrych, Joseph Bruchalski, Thomas J McCormick, and David L Livingston. A vacuum-tube guitar amplifier model using long/short-term memory networks. In *IEEE SoutheastCon*, 2018.

Udo Zölzer. *DAFX: digital audio effects*. John Wiley & Sons, 2011.

APPENDIX

A AUDIO REPRESENTATIONS

In this section we define the audio representations and transformations that are used throughout this thesis.

- **Audio waveform**: or *raw audio*, it consists of the amplitude values of a sampled audio signal. These values are measured with dBFS, i.e. decibels relative to full scale, where 0 dBFS corresponds to the highest amplitude level within a digital system. It is common to represent the amplitude of the audio waveform in the [−1,+1] range.
- **Short-time Fourier Transform (STFT)**: is a time-frequency representation based on the Discrete Fourier Transform (Gaydecki, 2004). It consists of slicing an audio waveform $x(n)$ into overlapping frames of equal length, multiplying each segment by a window signal $w(n)$ and computing the discrete Fourier Transform on each segment. For a single frame this is described as follows.

  $$X(\omega) = \sum_{-\infty}^{\infty} x(n)w(n)e^{-i\omega n} \tag{A.1}$$

  Where $X(\omega)$ is the Fourier Transform and ω is the frequency in radians/sample. It is common to calculate the STFT with the Fast Fourier Transform (FFT) algorithm, which reduces the computational cost. The *spectrogram* is computed by plotting $|X(\omega)|$ as a function of time.
- **Mel-Frequency Cepstral Coefficients (MFCCs)**: they consist of the amplitude coefficients of the mel-frequency cepstrum. The latter is the Discrete Cosine Transform (DCT) (Narasimha and Peterson, 1978) of the log mel-spectrogram. The mel-spectrogram is a time-frequency representation based on a model of human auditory perception. This differs from the STFT since the frequencies are not linearly spaced, but compressed into mel-frequency bands (Stevens et al., 1937). Mel-frequencies m can be calculated with the following equation (O'shaughnessy, 1987), where f is the frequency in Hz.

  $$m = 2595 \log_{10}(1 + f/700) \tag{A.2}$$

  The MFFCs can be computed with the following steps (Ganchev et al., 2005).

- Compute the STFT of an audio signal.
- Compress the powers of the spectrum into mel-frequency bands using triangular overlapping windows.
- Calculate the logarithm value of the energy at each mel-frequency band and correspondingly compute the DCT.
- The MFCCs are the amplitudes of the resulting mel-frequency cepstrum.

• **Gammatone Filter Bank**: Similar to mel-frequency bands, it corresponds to overlapping bandpass filters which simulate the motion of the basilar membrane within the cochlea (Moore, 2012). This is achieved with linear filters equally spaced on the Equivalent Rectangular Bandwidth (ERB) scale. The ERB scale corresponds to a logarithmic filter bank where the centre frequencies are equally spaced. The ERB scale approximates the bandwidths of the auditory filters of the cochlea and each bandwidth ERB can be estimated as follows (Smith and Abel, 1999).

$$\mathrm{ERB}(f) = 24.7(0.00437f + 1) \tag{A.3}$$

Each gammatone filter $g(t)$ is described in the time-domain by the impulse response of a gamma distribution and a sinusoidal tone (Patterson, 1986).

$$g(t) = \alpha t^{n-1} \cos(2\pi f_c t) e^{-2\pi f_b t} \tag{A.4}$$

Where n is the order of the filter, f_b and f_c are the filter bandwidth and center frequency, and α determines the amplitude.

B AUDIO SAMPLES

The audio samples of the experiments carried out in this thesis can be found at the following links:

- **Chapter 4** - *CEQ*: `https://mchijmma.github.io/end-to-end-equalization/`
- **Chapter 5** - *CAFX*: `https://mchijmma.github.io/modeling-nonlinear/`
- **Chapter 6** - *CRAFX*: `https://mchijmma.github.io/modeling-time-varying/`
- **Chapter 7** - All models: `https://mchijmma.github.io/DL-AFx/`
- **Chapter 8** - *CSAFX*: `https://mchijmma.github.io/modeling-plate-spring-reverb/`

C COMPUTATIONAL COMPLEXITY

The computational processing times were calculated with a *Titan XP GPU* and an *Intel Xeon E5-2620* CPU. We use input frames of size 4096 and sampled with a hop size of 2048 samples and it corresponds to the time a model takes to process one batch, i.e. the total number of frames within a 2-second audio sample. GPU and CPU times are reported using the non real-time optimized *python* implementation.

Table C.1 shows the number of trainable parameters and processing times across all the models. Table C.2 presents the versions of the *python* packages used throughout this thesis.

Table C.1: Number of parameters and processing times across various models.

model	number of parameters	GPU time (s)	CPU time (s)
CEQ	561,473	0.0336	0.4811
CAFx	604,545	0.0842	1.2939
WaveNet	1,707,585	0.0508	1.0233
CRAFx	275,073	0.4066	2.8706
CWAFx	205,057	0.0724	2.9552
CSAFx	410,977	0.752	4.5681

Table C.2: *python* packages.

python package	version
numpy	1.16.1
scipy	1.2.0
sacred	0.7.7
tensorflow-gpu	1.11.0
keras	2.2.4
librosa	0.6.0
sklearn	0.20.2
brian	1.4.4
json	2.0.9

D

ARCHITECTURES OF THE PROPOSED MODELS

Table D.1: Layers across all models.

model	adaptive front-end	latent-space	synthesis back-end
CEQ	Conv1D, Conv1D-Local, max-pooling	Dense-Local, Dense	unpooling, $\times$, deConv1D
CAFx	Conv1D, Conv1D-Local, max-pooling	Dense-Local, Dense	unpooling, $\times$, DNN-SAAF, deConv1D
CRAFx	(time-distributed) Conv1D, Conv1D-Local, max-pooling	Bi-LSTM	unpooling, $\times$, DNN-SAAF, SE, $+$, deConv1D
CWAFx	(time-distributed) Conv1D, Conv1D-Local, max-pooling	WaveNet, Dense	unpooling, $\times$, DNN-SAAF, SE, $+$, deConv1D
CSAFx	(time-distributed) Conv1D, Conv1D-Local, max-pooling	Bi-LSTM, SFIR	unpooling, $*$, $\times$, DNN-SAAF, SE-LSTM, $+$, deConv1D