Master Thesis on Sound and Music Computing

Universitat Pompeu Fabra

Neural Audio Effect Modelling Strategies for a Spring Reverb

Francesco Papaleo

Supervisor: Xavier Lizarraga

September 2023

Contents

Abstract

Virtual analog modelling emulates the processing characteristics of a given physical device. This has been an active field of research and commercial innovation in which two main perspectives have been historically adopted. The first one: *white-box*, seeks to reproduce the exact behaviour through algorithmic simulation of circuits or physical phenomena. The second one: *black-box*, aims to learn the approximation function from examples recorded at the input and output stages of the target device. In this second approach, deep learning has emerged as a valuable strategy for linear systems, such as filters, as well as nonlinear time-dependent ones like distortion circuits or compressors.

The spring reverb is as audio effect with a very long and rooted history in music production and performance, based on a relatively simple design, this device is an effective tool for artificial reverberation. The electromechanical functioning of this reverb makes it a nonlinear time-invariant spatial system that is difficult to fully emulate in the digital domain with white-box modelling techniques.

This thesis wants to address the modelling of spring reverb, leveraging end-to-end neural audio effect architectures through supervised learning. Recurrent, convolutional and hybrid models have successfully been used for similar tasks, especially compressors and distortion circuits emulations. Using two available datasets of guitar recordings, we evaluate with quantitative metrics, acoustical analysis and signal processing measurements the efficiency and the results of four neural network architectures to model this effect. We present the results and outline different strategies for this modelling task, providing a reproducible experimental environment with code [1].

[1]https://github.com/francescopapaleo/neural-audio-spring-reverb

Chapter 1

Introduction

Audio effects are essential tools in the generation or treatment of an audio signal. The most schematic definition of an audio effect is that of a chain that can be described as: audio input - processing on user defined parameters - output signal. Each audio effect can be described with respect to the backing physical and acoustical effect, the underlying digital signal processing and the audio application. For clarity, we need to differentiate the "sound transformation", i.e. the object or what is perceived, and the "audio effect" that produces it. The first corresponds to the subjective perception and its representation, while the latter uses a subject-related term, effect[1]. Applying signal processing techniques to sounds in order to modify how they will be perceived has a great interest for many applications in audio. Over time, this has led to the development of a vast number of techniques to achieve many different transformations that makes a universal classification impossible. Each different "users community" has developed its own taxonomy to meet their specific needs. The development of audio signal processing in the analog domain has led to the invention of a multitude of devices to transform a signal. Not only with the purpose of imitating an acoustical or physical phenomena, but also with a creative intent. We can enumerate four main analog technologies for audio effects:

1. Mechanic / acoustic
2. Electromechanic
3. Electromagnetic
4. Electronic

Progressively, those analog devices have found their "translation" into the digital domain emulating acoustical or perceptual properties of analog effects like filtering, wah-wah, echo, Leslie and reverberation. One way of building a taxonomy in the digital domain, could adopt the standpoint of the "instrument-maker", the software engineer. Indeed, a classification based on the underlying techniques appears to be the most straightforward. This can be done depending on two criteria: one focuses on the implementation technique itself (i.e. resampling, modulators/demodulators, non-linear processing, spatial effects, etc.) the other on the domain where the signal processing is applied:

- Time domain
- Frequency domain
- Time and frequency domain

Limiting the analysis only to the engineering aspects would miss a substantial dimension of the audio effects realm. In psychoacoustics literature, sound has five main fundamental perceptual attributes [2] listed below.

1. Loudness: the perceived intensity of the sound through time.
2. Time and rhythm: perceived through the duration of sound and its gaps.
3. Pitch: with height (high/low frequency) and chroma as attributes.

4. Spatial hearing: with location, directivity and room effect as attributes.

5. Timbre: with complex relations between spectral centroid, noisiness, texture and formants.

It is worth notice that each audio effects can be seen in relation with the manipulation of one or multiple attributes. Based on these considerations, the attributes become the main classification criteria. In this way, the perspective is flipped and the standpoint of the listener is adopted: which is the one that experiences the alterations made by signal processing. It is important to emphasize that by the combination of multiple principles, like the ones previously mentioned, originates an interdisciplinary classification that links the various layers of each single discipline-specific criteria. Ranging from low-level to high-level features, it can be summarized as follows:

- Digital implementation technique
- Processing domain
- Applied processing
- Control type
- Perceptual attributes
- Semantic descriptors

With this multidisciplinary conceptual framework, we want to address the digital implementation of the spring reverb with deep learning techniques in the time domain. Each of the aforementioned features is involved in the design and the deployment of the project and plays a fundamental role for the quality of the outcome.

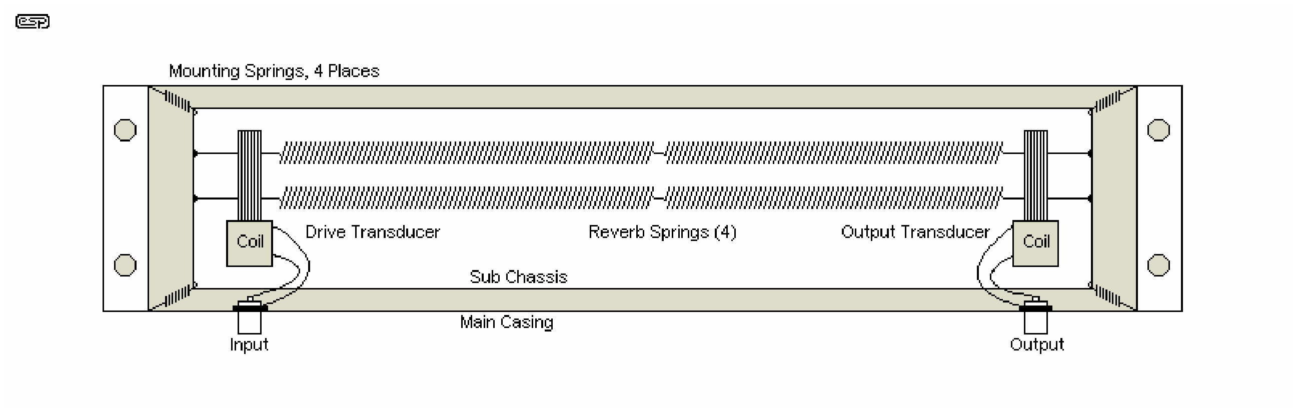


Figure 1: High level schematics of a spring reverb and its main components (source: Elliot Sound Products).

1.1 Motivation

The history of spring reverb can be traced back to the late 1920s, when the first electromechanical reverberation units were developed by Hammond to add acoustic quality to the sound of his electric organ. Its long history has seen many different construction strategies and improvements. Nonetheless, it always includes a metal spring coiled tightly in an enclosure, with a transducer at one end to convert the electrical signal into vibrations in the spring, and a transducer at the other end to convert the vibrations back into an electrical signal. The reverberation effect is created by the sound waves bouncing back and forth between the metal spring coils, creating a dispersive propagation with a complex pattern of reflections and echoes that characterized its unique sound. The signal is transduced from the electrical domain into the mechanical domain before being transduced back into the electrical domain. This electromechanical functioning makes the spring reverb a time-invariant nonlinear system with a number of parameters involved into the shaping of sound. Each unit has its own distinctive sound that is also affected by preamps, number of springs and size of the tank. During the 1950s and the early 1960s, great development and improvement has been done with the introduction of small-diameter springs that paved the way for the widespread use in guitar amplifiers. Towards the end of the 1960s, impedance discontinuities began to be placed along the springs to increase the echo density and achieve a very natural room-like sound, the most iconic example is the: AKG BX20E reverberator[3]. Its simplicity of construction and its versatile and distinctive sound guaranteed the spring reverb its place in the toolbox of musicians and producers along the contemporary music history and in many different genres. The enduring popularity of spring reverb in music history and its central role in music production serve as primary motivations for this thesis.

In recent years, the wide adoption of Digital Audio Workstations (DAW) and the democratization of the music-making process have also increased the demand for plug-ins emulating of analog devices. Knowledge-based approaches have reached very high quality standards, this process often comes with a costly design process and computational cost, in addition it is rarely generalizable to an entire class of effects. Recent developments in data-driven approaches have demonstrated its effectiveness in many audio applications including speech synthesis [4], source separation [5] and neural audio synthesis [6]. The advantages offered by these solutions have created great interest in the application of deep learning techniques for modelling audio effects; previous work on guitar amplifiers has been well documented by Vanhatalo et al. [7] and real-time implementations have been proposed[8].

Third motivation for this project is the possibility of expanding the creative capabilities for music production with the introduction of conditioning elements in the model. Early results have been documented in the literature by Steinmetz et al. [9]. Often an implementation technique comes with its own specific parameters that may expand the ones previously considered from the acoustical or physical phenomena and lead to new creative possibilities [1]. In particular, some previous work by Comunità et al. [10] have proven that time-varying feature-wise linear modulation into existing temporal convolutional systems as a valid approach for the network to adaptively learn.

1.2 Objectives

The core research question this thesis seeks to address is: "which neural network architecture is most adapt at modelling a non-linear, time-invariant effect such as the spring reverb?" To comprehensively address this inquiry, we must traverse the landscape of various neural network architectures, supervised learning methodologies, integrate domain-specific knowledge and deep dive into the nuances specific to the spring reverb's electro-mechanical functioning. Outlined below are the refined objectives that will guide this research:

1. **Data Examination:** Evaluate the dataset proposed by Martinez Ramirez et al. [11] together with the one published by Pedroza et al. [12].The aim is to ensure we train our models on the most relevant and comprehensive data available.

2. **Architectural Exploration:** Delve into the state-of-the-art neural network architectures—both convolutional [9, 4] and recurrent [13, 14, 10]—and assess their potential in virtual analog modelling.

3. **Use Case Analysis:** Undertake a comprehensive analysis to discern the most optimal use case for each examined architecture, ensuring a focused and relevant application of each methodology.

4. **Domain-Specific Knowledge Integration:** We posit that combining traditional deep learning methodologies with domain-specific insights is pivotal for success in music technology research. This objective will revolve around actualizing this blend to ensure the models' effectiveness and robustness.

5. **Open and Reproducible Research:** A commitment to transparency and reproducibility underpins this thesis. While a detailed discussion will be presented in Chapter 3, it's vital to underscore the intention to provide open-source code, data, and comprehensive

documentation. This ensures that the research can be independently verified, built upon, and validated by the wider scientific community.

Chapter 2

State of the Art

When a source emits a sound, it radiates through the medium, and is received directly by the listener. This sound is referred to as the direct sound. However, sound waves also reflect on the surfaces in the environment before reaching the listener. These reflections arrive with a delay and are attenuated in intensity. These reflected and attenuated copies of the direct sound are perceived as reverberation. In audio signal processing, reverberation effects are often introduced synthetically to give the listener a sense of the space where the audio was recorded, or to enhance the richness and sustain of musical tones. Techniques for creating such effects range from convolution with measured or simulated room impulse responses (RIRs) to algorithmic methods that mimic the diffuse nature of reverberation, such as digital reverberators and feedback delay networks. The first idea of artificial reverberation has been introduced by Schroder in 1961 [3].

2.1 Virtual Analog Modelling

Virtual analog modelling (VAM) is a technique that emulates the sound and behaviour of analog audio equipment, such as guitar amplifiers, vintage synthesizers, and effects processors, using digital signal processing methods. The primary aim of VAM is to reproduce the distinct characteristics, including non-linearities, of analog circuits, which can be challenging to achieve using purely digital signal processing techniques. Historically, VAM started in the 1980s with the development of digital signal processing and the need to emulate analog sound. Since then, it has been continually refined, leading to the creation of white-box, gray-box, and black-box modelling techniques. Each presents advantages and disadvantages and is chosen based on the specific requirements of the virtual analog model being coded. The next subsection will outline the main characteristics and differences of these techniques.

2.1.1 White-box

The "white-box" method involves analysing the analog circuitry of the effect and simulating it using discrete-time techniques. This approach is based on a complete study and analysis of the circuit that implies creating a detailed mathematical model of the original device, including all the individual components and their interactions [15]. In some cases, Wave Digital Filters (WDFs), a class of digital filters, have been used. WDFs are *closely related to classical filter networks, preferably lossless filters inserted between resistive terminations [16].* Together with other techniques [17], they proved to be an efficient and accurate way of VAM using digital signal processing techniques. Especially for non-linear devices. One significant downside of the white-box approach is that it requires a high level of expertise and knowledge of the original analog equipment, as the mathematical models are based on circuit diagrams or other technical specifications of the original equipment. This can make the modelling process time-consuming and expensive, particularly for complex equipment. It can also result in a computationally intensive algorithm, particularly for large and complex circuits, which can limit its real-time performance in some applications.

2.1.2 Black-box

On the other hand, the "black-box" method treats the circuit as a whole and attempts to learn its behaviour through the use of "examples". This approach consists on measuring the circuit's response to some input signals, and creating a model which replicates the observed input-output mapping. Unlike white-box, which divide the device into blocks and components that can be individually interpreted, this approach reproduces the target device with a mathematical representation that is not strictly derived by its internal functioning [7]. Gray-box, is a hybrid technique that tries to combine the advantages of the previous two. In gray-box approach, a mix of circuit analysis and neural networks to create a model that emulates the behaviour of the original effect. The circuit analysis is used to identify the main non-linearities and other important features, then the effect's behaviour under various input signals is predicted [18]. While requiring less in-depth analysis of the original device, gray-box as well as black-box approaches may have difficulties into generalization since the resulting models are often very specific to the analog circuit used as reference [19]. In recent years, the use of machine learning and Artificial Neural Networks (ANNs) is becoming increasingly popular for black-box modelling, thanks to the availability of large amount of data and the increased computational capabilities. From a machine learning perspective, the process of modelling an audio effect can be regarded as a sequence modelling task. This is also referred to as data-driven approach: the process is guided by the insights and the conclusions inferred from the data. From the study of the existing literature emerges that a few datasets have been made and used for VA modelling through neural networks for audio effects in general. Each of them includes different dry and wet sounds of different devices. Some of them use as a reference a digital plugin (VST or AU). Table 1 summarize the most relevant dataset and gives an overview of their main characteristics.

Name	**Year**	**Sources**	**Samples**	**Hours**	**Sample Rate**	**Reference**
IDMT-SMT-Audio-Effects	2010	Bass/Guitar	55044	30	44.1 kHz	[20]
IDMT-SMT-Bass	2013	Bass (dry)	6000	-	44.1 kHz	[21]
IDMT-SMT-Guitar	2014	Guitar (dry)	849	120	44.1 kHz	[22]
SignalTrain LA2A	2019	Noise	18179	43	44.1 kHz	[23]
DL-AFx	2019	Bass/Guitar	2500	-	16.0 kHz	[24]
SpringSet	2020	Bass/Guitar	2400	1.5	16.0 kHz	[25]
EGFxSet	2022	Guitar	8970	11.5	44.1 kHz	[12]

Table 1: Main datasets used for audio effects modelling, their characteristics and the papers that introduced them.

2.2 Neural Audio Effects

Sequence modelling consists on predicting a series of outputs based on a series of inputs over time. For a given input sequence $x_0, x_1, \ldots, x_T$, the aim is to estimate the corresponding outputs $y_0, y_1, \ldots, y_T$. A fundamental principle of sequence modelling is causality: the prediction at a specific time t, represented as y_t, should be contingent solely on the inputs seen up to that time $x_0, \ldots, x_t$ and not be influenced by future inputs $x_{t+1}, \ldots, x_T$.

Mathematically, a sequence model f can be expressed as:

$$\hat{y}_0, \ldots, \hat{y}_T = f(x_0, \ldots, x_T)$$

where it's essential that each y_t is reliant only on $x_0, \ldots, x_t$.

The primary objective in sequence modelling is to pinpoint a model f that reduces the discrepancy between the predicted and actual outputs of a sequence. This discrepancy is typically quantified using a loss function. Within the domain of virtual analog, this method is applicable to tasks like autoregressive prediction, where the subsequent sample of an audio signal is forecasted using its preceding samples. Artificial Neural Networks (ANNs) are a sub-category of machine learning algorithms inspired by the structure and function of the human brain. ANNs consist of multiple interconnected processing nodes that work together to learn from data and perform tasks. There are several types of ANNs architectures, each designed to solve specific problems. In this thesis, we will focus on two main types: Deep Neural Network (DNN) and Recurrent Neural Network (RNN). It is difficult to formulate a consensual definition of deep learning, here we adopt the Briot and Pachet formulation in [26]: *it is a repertoire of machine learning techniques, based on artificial neural networks. The common ground is the term deep, which means that there are multiple layers processing multiple levels of abstractions, which are automatically extracted from data, as a way to express complex representations in terms of simpler representations.* From there we can outline the Deep Neural Network, as one sub-set of ANN that rely on deep learning for patterns identification across input data. This architecture suits well the processing of static, non-sequential data and can be applied to problems where the input and output have a fixed size. For tasks of virtual analog modelling, this involves transforming the raw input data into the frequency domain using Fast Fourier Transform. The DNN is a feedforward type of NN where the input flows in one direction through the network's layers, without feedback connections. This is a radical element that separates it from the RNN.

2.2.1 Recurrent Architectures

This architecture has feedback connections: the output of a node in one layer can be used as an input for the same or a different node in another layer. This makes the RNN an architecture designed to handle sequential data. Unlike feedforward networks, which process input data in a single pass, it maintains an internal state that allows to process input sequences of arbitrary length. Technically, a Recurrent Neural Network (RNN) is a type of Deep Neural Network (DNN), as it contains multiple layers of neurons and uses non-linear activation functions. However, in practice, the term "DNN" is often used to refer to feedforward neural network, which is the most common type of deep neural network and this is why the RNN very often treated as a separate category. Historically, recurrent neural networks (RNNs) have been the preferred

Source	**Year**	**Audio Effect**	**Architecture**	**Loss Function**
Covert et al. [27]	2013	Tube Amp	mod. RNN	RMS
Zhang et al. [28]	2018	Tube Amp	Multi-Layer LSTM	MSE
Damskagg et al. [29]	2019	Distortion	WaveNet	ESR
Wright et al. [13]	2019	Tube + Dist	LSTM, GRU, WaveNet	ESR + DC
Damskagg et al. [30]	2019	Tube Amp	WaveNet, MLP	ESR
M. Ramirez et al. [31]	2019	Various	CNN + DNN	MAE
M. Ramirez et al. [19]	2019	Var. Time-Varying	LSTM + Conv	MAE, MSED
M. Ramirez et al. [24]	2020	Amp, Leslie	LSTM + Conv	MAE
Wright et al. [8]	2020	Valve Amp	WaveNet, LSTM	ESR + DC
M. Ramirez et al. [11]	2020	Reverb	LSTM + Conv	MAE + MSE
Steinmetz et al. [9]	2020	Reverb, Comp	TCN	STFT
M. Ramirez et al. [32]	2021	Various	DeepAFX	Delay-invariant
Comunità et al. [10]	2022	Fuzz, Comp	GCN	MAE, MR STFT
Wright et al. [33]	2021	Phaser, Flanging	LSTM + Conv	ESR + DC
Steinmetz et al. [14]	2022	Dyn Range Comp	TCN	MAE, STFT, LUFS
Simionato et al. [34]	2022	Opt Comp	Encoder-Decoder	MSE

Table 2: Selected literature for black-box modelling, with audio effects, architectures and loss functions.

framework for handling sequential data: they enhance feed-forward neural networks by using previous states to inform current predictions. Initial versions of RNNs faced several obstacles during training, like the issues of exploding and vanishing gradients [35]. Early introduction of a feedforward neural network model for VA emulation can be traced back to 2013 by Covert and Livingston [27]. Despite the inaccurate results (100% of root-mean-square error), it can be considered as a first attempt to emulate tube amplifiers by neural networks.

The vanishing/exploding gradient descent problem in neural networks refers to a phenomenon that can occur during the training phase, where the gradients of the loss function with respect to the weights of the network become either too small (vanishing gradient) or too large (exploding gradient) to allow effective training. The gradient is used to update the weights of the neural network during backpropagation. During training, the gradients are propagated backwards through the layers of the network, and the weight updates are determined by the product of the gradient and the learning rate. If the gradient becomes very small, then the weight updates become negligible, which leads to slow convergence and may cause the network to get stuck in a local minimum. This is known as the vanishing gradient problem. Conversely, if the gradient becomes very large, the weight updates become too large, and the optimization algorithm may overshoot the minimum and fail to converge. This is known as the exploding gradient problem. The vanishing gradient problem typically occurs in DNN, which have many layers. As the gradients are propagated backwards through the layers, they are multiplied together, and if the gradients are small, the product of many small numbers becomes very small. The exploding gradient problem is more common in RNNs, which have loops in the network. If the weights in the loops are too large, the gradients may become very large and cause the optimization algorithm to diverge. Several techniques have been developed to address the vanishing and exploding gradient problem. One approach is to use different activation functions, such as the rectified linear unit (ReLU), which can help mitigate the vanishing gradient problem by allowing gradients to flow more easily through the network. Another approach is to use weight initialization techniques that can help prevent the gradients from becoming too small or too large. Additionally, gradient clipping can be used to limit the size of the gradients during training, which can help prevent the exploding gradient problem.

Long Short-Term Memory (LSTM) is a variant of the recurrent architecture that was introduced to overcome the vanishing/exploding gradient problem that was afflicting "vanilla" RRNs [36]. In 2018, Schmitz and Embrechts, introduced a model based on the LSTM, enabling the emulation of tube guitar amplifiers with a small root-mean-square error and an accurate perceptual accuracy [37]. Similarly to RNNs, these networks are specifically designed to handle long-term dependencies. Unlike standard RNNs, LSTMs use gated units to selectively remember and forget information over time. The unit consists of two vectors: the cell state is the

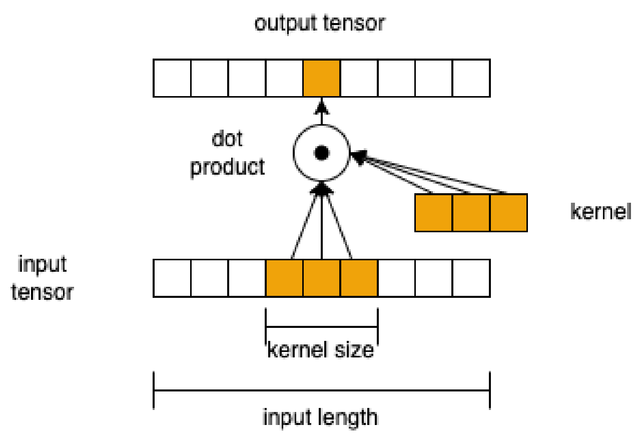


Figure 2: 1D convolution

long-term memory, while the hidden state keeps the short-term one.

2.2.2 Convolutional Architectures

Despite these advances, RNNs, known for capturing long-term temporal structures, are limited by their inherent inability to be parallelized due to their autoregressive nature. This results in slower processing speed when compared to feed-forward approaches. In contrast, convolutional neural networks (CNNs), which were originally used in computer vision, have demonstrated superior performance to analogous RNNs across various sequence modelling tasks, offering improved efficiency [4]. Consequently, our research prioritizes CNNs for spring reverb neural audio effect modelling. Many relevant scientific contributions are available on general VA modelling with neural network, a selection of which is summarized in Table 2.

Digital audio signals, at their core, are 1-dimensional sequences of samples representing changes in air pressure over time. To capture patterns in these sequences, the operation of convolution can be particularly effective. A filter, also known as *kernel*, with its own set of weights, slides over the audio signal (the sequence). At each position of this filter over the signal, the element-wise product of the kernel's weights with the current section of the audio signal is taken, and then sum the results. This sum forms the signal point in the output sequence. Sliding the kernel over by one position and repeating until the entire signal is covered is what is called a 1D convolution (see Figure 2). 1D convolution can be formalized as follows [38]:

$$\text{out}(N_i, C_{\text{out}j}) = \text{bias}(C_{\text{out}j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}j}, k) \star \text{input}(N_i, k) \tag{2.1}$$

Where: $\star$ is the valid cross-correlation operator, N is a batch size, C denotes a number of channels and L is a length of a signal sequence.

The causality of convolutional architectures is the innovation that led to perform extremely well on audio signals: each output sample predicted by the model depends only on r previous input samples: r is known as the receptive field of the model. This is achieved using dilated convolution that can exponentially increase the receptive field of a convolutional unit without increasing the number of parameters. The receptive field of a model is a function of the kernel size (k_i), stride (s_i), dilation factor (d_i), for the i^th layer of the network is:

$$r_i = r_{i-1} + d_i \times (s_i \times (k_i - 1)) \tag{2.2}$$

The total receptive field of a convolutional model with I layers represents the span of input data influencing the output of the final layer. Starting with the receptive field of the first layer, which is the size of its kernel, the receptive field expands at each subsequent layer according to the layer's kernel size, stride, and dilation factor. By the time we reach the I^th layer, the total receptive field encapsulates influences from a range of input data, governed by the cumulative effects of all previous layers.

WaveNet - A prominent architecture in audio processing, WaveNet was introduced by van den Oord et al. in 2016 [4], it stands out as a deep generative model for raw audio waveforms. Its speciality lies in using dilated causal convolutions, enabling it to generate coherent, high-quality waveforms over extended periods, solidifying its position in tasks such as speech synthesis, music generation, and virtual analog modelling.

Temporal Convolutional Network — A convolutional deep neural network variant, the Temporal Convolutional Network (TCN) was first proposed by Lea et al. [39] for video action segmentation. TCN, equipped with dilated, causal 1D convolutional layers, maintains consistent input-output lengths. Its architecture, leveraging dilation rates, ensures a robust receptive field, thus proficiently handling long-term input sequence dependencies.

2.3 Building Blocks

With this section we want to outline some important elements that connected together determine the functioning, hence the performance, of a neural network architecture. From the selected literature (see Table 2, we summarize the most recurrent ones that are relevant in the field of audio effects virtual analog modelling.

2.3.1 Activation Functions

Activation functions are a key component in a network architecture, there is a great number of them used in different branches of deep learning. Here follows a non-exhaustive selection of the ones relevant for the purpose of this work [13, 4].

Sigmoid - Also called the "logistic" function, is probably the most used activation function. Takes any real values as input and outputs values in the range of 0 to 1. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0. Mathematically, its formula is:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

Tanh (Hyperbolic Tangent) - Also a very commonly used activation function in neural networks. It is a non-linear function that squashes the input values between -1 and 1. It is commonly used in neural networks because it is non-linear, and it is symmetric around zero, meaning that it can model both positive and negative inputs. For higher inputs, its output will be closer to 1.0, whereas for a smaller input (negative), the closer it will be to -1.0. It is mathematically represented as:

$$f(x) = \frac{(exp(x) - exp(-x))}{(exp(x) + exp(-x))} \tag{2.4}$$

ReLU (Rectified Linear Unit) - It is a simple yet efficient function that returns zero for negative inputs and the input value itself for positive inputs. ReLU is widely used because it is computationally efficient and does not suffer from the vanishing gradient problem, as it allows the gradient to flow more easily through the network. Mathematically, it is defined as:

$$f(x) = max(0, x) \tag{2.5}$$

PReLU (Parametric ReLU) - Although widely used, the ReLU has a known limitation termed as the "Dying ReLU" problem. This problem arises because the gradient value becomes zero for all negative inputs, leading to certain neurons becoming inactive or "dead" during the backpropagation process. Consequently, these neurons do not update their weights and biases, reducing the model's capacity to fit or train effectively on the data.

To address this limitation, the Parametric ReLU (PReLU) was introduced with a slope parameter, a, for negative input values. The function is mathematically represented as:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ a \cdot x & \text{if } x \leq 0 \end{cases} \tag{2.6}$$

Here, the slope a isn't fixed but is learned through backpropagation, allowing the model to determine the most suitable value. PReLU is particularly useful when the more simplistic Leaky ReLU variant fails to address the Dying ReLU issue and vital information isn't effectively transmitted to subsequent layers. However, it's worth noting that PReLU's performance can vary across different tasks, primarily influenced by the value of the slope parameter.

SoftPlus - Is part of a class: the gated activation functions, these allow the network to learn continuous gating mechanisms. The softplus is used in neural networks to model continuous and time-varying phenomena. It has a gating mechanism that is based on a smooth version of the ReLU function, which allows the network to learn continuous gating and its output is scaled by another function that ranges between zero and one, allowing for continuous gating. It is defined as:

$$f(x) = log(1 + exp(x)) \tag{2.7}$$

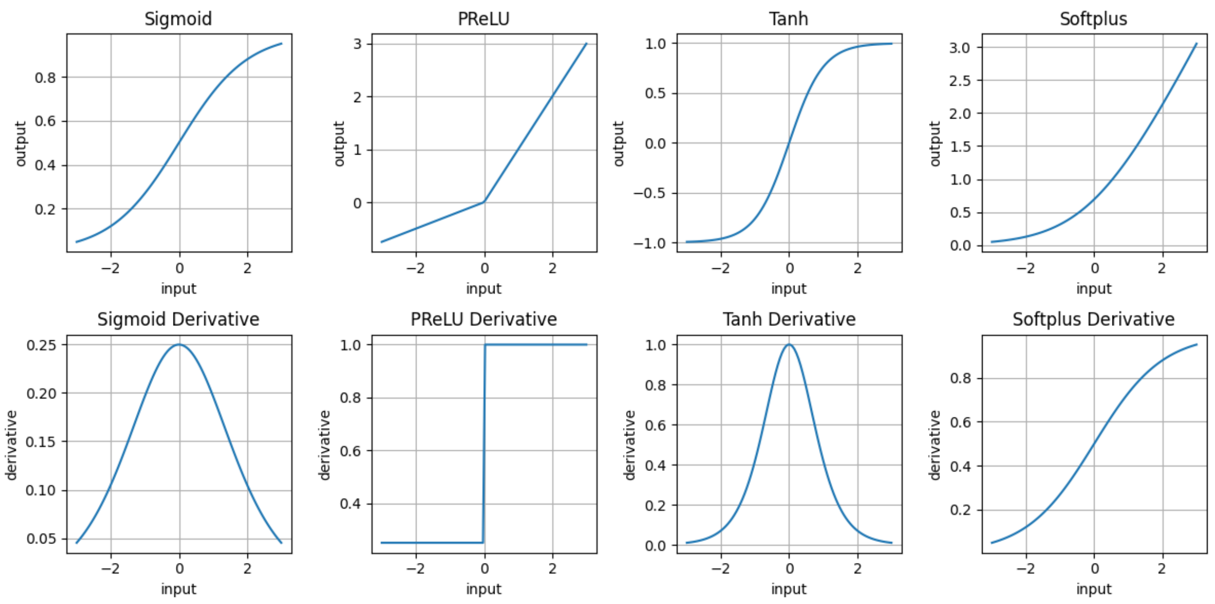


Figure 3: Plot of the output and the derivative for the Sigmoid, PReLU, Tanh and SoftPlus non-linearities.

2.3.2 Loss Functions

A loss function, also known as cost function, measures the difference between the predicted output of the model and what the output *should be*, the target or ground truth. It quantifies how the model is performing by calculating the error or loss. The goal of the model is to minimize the value of the loss function, indicating that the predicted output is as close as possible to the expected/desired output for the task.

Mean-Absolute-Error (MAE) . In supervised learning regression tasks and in the context of audio effect modelling, calculates the absolute value of the distance in the samples between the predicted signal and the ground-truth [24]. The model output being y and target (ground truth) being $\hat{y}$, the MAE is defined as:

$$L_{MAE} = \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}_i| \tag{2.8}$$

where N is the number of samples used to calculate the MAE. In some cases, the difference between the observed ground truth and the predicted samples is squared, as in the previous formula the result is then divided by the number of samples. This results in the average squared error across all samples, named: the Mean Square Error [11].

Error-to-Signal Ratio (ESR) Measures the difference in magnitude between the predicted value and the true value. The ESR is computed by dividing the absolute difference between the true value and the predicted value by the magnitude of the true value. The resulting ratio indicates how much the error deviates from the true value, relative to the true value itself. Its mathematical formalization is:

$$L_{ESR} = \frac{\sum_{i=0}^{N-1} |y_i - \hat{y}_i|^2}{\sum_{i=0}^{N-1} |y_i|^2} \tag{2.9}$$

where $|y|$ is the magnitude of the true value (also called target) and $|\hat{y}|$ is the prediction by the neural network. The denominator in the ESR normalises the loss with regard to the energy of the target signal, preventing it from being dominated by the segments with higher energy.

Difference in DC offset Wright and Valimaki [40], proposed to measure the difference in DC offset between the target and the output of the neural network, it is given by:

$$l_{DC} = \frac{\left| \frac{1}{N} \sum_{n=0}^{N-1} (y[n] - \hat{y}[n]) \right|^2}{\frac{1}{N} \sum_{n=0}^{N-1} |y[n]|^2} \tag{2.10}$$

In previous work on distortion and compressors modelling with deep neural networks, a combination of Error-to-Signal-Ratio (ESR) and DC offset as loss function has been proposed. These two terms combined, address the low-frequency artefacts that are often introduced by deep neural network at their output stage and keeps a perceptual relevance given by the use of the ESR [40]. the overall error is then simply the sum of the two terms:

$$\varepsilon = \varepsilon_{ESR} + \varepsilon_{DC} \tag{2.11}$$

Pre-Emphasis Filter In many implementations of compressor and distortion circuits [29], [13], [30], the computation of the ESR between target and output is preceded by a pre-emphasis filter. This filter doesn't affect the DC offset. Pre-emphasis is a filtering technique that amplifies the high frequencies of a signal, in VAM, a pre-emphasis filter is often applied before the extraction of features to balance the frequency spectrum. Higher frequencies in the spectrum have lower energy magnitudes compared to lower frequencies, applying a filter compensates for this difference, by doing so, the neural network becomes more sensitive to the changes in the high-frequency range, which is crucial for a realistic VAM. As presented by Wright and Valimaki. in [40], high-pass pre-emphasis filters with different orders and weightings are intended to compensate for human-ear particular perception of loudness at different frequencies, as an example in PyTorch library, a pre-emphasis filter is defined as:

$$y[i] = x[i] - coeff \cdot x[i-1] \tag{2.12}$$

The Short-time Fourier Transform (STFT) implemented by Steinmetz and Reiss in [41] loss is composed of two main terms: spectral convergence and spectral log-magnitude. Spectral convergence (2.13) and spectral log-magnitude (2.14) are computed as follows, where $\| \cdot \|_F$ is the Frobenius norm, $\| \cdot \|_1$ is the L1 norm, and N is the number of STFT frames:

$$l_{SC}(\hat{y}, y) = \frac{\| |STFT(y)| - |STFT(\hat{y})| \|_F}{\| |STFT(y)| \|_F} \tag{2.13}$$

$$l_{SM}(\hat{y}, y) = \frac{1}{N} \| log(|STFT(y)|) - log(|STFT(\hat{y})|) \|_1 \tag{2.14}$$

The overall STFT loss is defined as the sum of these two terms.

Multi-Resolution STFT Is an extension of the STFT loss that increases robustness and limits the potential bias. The spectral convergence (SC) and the log magnitude (LogMag or LM) loss for a single resolution are defined as:

1. Spectral Convergence (SC):

$$l_{SC}(\hat{y}, y) = \frac{\| |\mathrm{STFT}(\hat{y})| - |\mathrm{STFT}(y)| \|_F}{\| |\mathrm{STFT}(y)| \|_F} \tag{2.15}$$

2. Log Magnitude (LogMag or LM):

$$l_{SM}(\hat{y}, y) = \frac{1}{S} \| \log(|\mathrm{STFT}(\hat{y})|) - \log(|\mathrm{STFT}(y)|) \|_1 \tag{2.16}$$

For multiple resolutions, the combined loss is:

$$l_{MR}(\hat{y}, y) = \sum_{m=1}^{M} (l_{SC}^{m}(\hat{y}, y) + \alpha l_{SM}^{m}(\hat{y}, y)) \tag{2.17}$$

Where:

- M is the total number of resolutions.
- α is a weighting factor for the log magnitude loss.

2.3.3 Optimizers

Backpropagation is the algorithm used to compute the gradient of the loss function with respect to the weights and biases of the neural network. The gradient tells us the direction in which the weights and biases should be adjusted to decrease the value of the loss function. This adjustment is then performed by an optimizer algorithm, such as gradient descent, SGD, Adam, or RMSProp. The loss function is a function that measures the error between the predicted output of the neural network and the ground truth, while backpropagation is an algorithm used to compute the gradient of the loss function with respect to the weights and biases of the network. The gradient is then used by the optimizer to adjust the weights and biases to minimize the loss and increase performance. An optimizer is an algorithm that is used to update the weights and biases of the network during the training process in order to minimize the error between the predicted output and the true output. The goal of an optimizer is to find the optimal set of weights and biases that will minimize the cost or loss function of the network. This is achieved through an iterative process in which the optimizer adjusts the weights and biases in small steps, based on the gradient of the cost function with respect to the weights and biases. There are several types of optimizers used in NNs, including:

- **Gradient Descent** is the most basic optimizer that updates the weights and biases in the direction of the negative gradient of the cost function. It works by calculating the gradient of the loss function with respect to the weights and biases, and updating them in the direction of the negative gradient to minimize the loss. The learning rate determines the size of the steps taken in the direction of the gradient, and needs to be carefully chosen to avoid overshooting the minimum or getting stuck in a local minimum.

- **Stochastic Gradient Descent (SGD)** is a variant of gradient descent that updates the weights and biases based on a randomly selected subset (or batch) of the training data, instead of the entire dataset. This can speed up training and avoid getting stuck in a local minimum, but it may also introduce more noise into the optimization process.

- **Adam** is an adaptive optimizer that adjusts the learning rate for each weight and bias based on the magnitude of their gradients. It combines elements of both momentum-based and adaptive learning rate-based optimization algorithms. Adam adapts the learning rate for each weight based on the first and second moments of the gradient, and uses a bias correction to correct for initialization bias. This allows Adam to converge faster and with more accuracy than other optimization algorithms. ADAM is a popular choice because it can handle large datasets and high-dimensional data efficiently, it does this combining the strengths of both SGD and root-mean-square propagation (RMSProp). An adaptive learning rate is used for each parameter, which means that it adjusts the size of the updates to the model's parameters based on the gradient of the loss function. This allows to converge to the optimal solution faster and more reliably than other optimization algorithms, such as SGD.

- **RMSProp** is another adaptive optimization algorithm that scales the learning rate for each weight and bias based on the magnitude of their gradients. It is similar to Adam, but does not use the second moment of the gradient. Instead, it calculates an exponentially weighted moving average of the squared gradients and uses this to adjust the learning rate. This can help RMSProp to converge faster and with more accuracy than standard gradient descent or stochastic gradient descent.

The choice of optimizer can have a significant impact on the training performance of a neural network, and it is often a matter of experimentation to determine which optimizer works best for a given problem.

Chapter 3

Methods

In this section will be described the methodology applied for the development of the Neural Spring Reverb. We will start by describing the architectures selected for comparison, to then analyse the available datasets and finally outline the design of the experimental environment. The Python code together with the data pipeline, pre-trained models and Jupyter Notebooks is publicly available in a GitHub Repository [1]. The code is designed with great attention to the reproducibility of the experiments, a command line interface has been implemented.

3.1 Models

Previous work in this field has shown that convolutional (WaveNet, TCN, GCN) models can be very effective for 'style transfer' tasks and at the same time there is evidence and studies on the effectiveness of 'recurrent' type architectures (LSTM, GRU). We therefore propose here five architectural patterns that revolve around these two principles of operation and, in some cases, attempt to optimise performance with hybrid-type systems.

[1]https://github.com/francescopapaleo/neural-audio-spring-reverb

3.1.1 Temporal Convolutional Network

The TCN model presented here was introduced by Steinmetz and Reiss as a style-transfer architecture for audio effects [42]. As seen in Chapter 2, this architecture is designed for sequence modelling tasks and often outperforms traditional recurrent networks. It uses causal (non-future-looking) convolutions. The model consists of a stack of multiple TCN blocks, each increasing in dilation as we go deeper. The first block takes in the input sequence and passes it through a TCN block, transforming it to a specified number of channels. Intermediate blocks are identical in structure and simply transform the input without changing its channel count. Finally, the last block maps the transformed features into the desired output channels. Figure 4 shows the inner structure of those blocks.

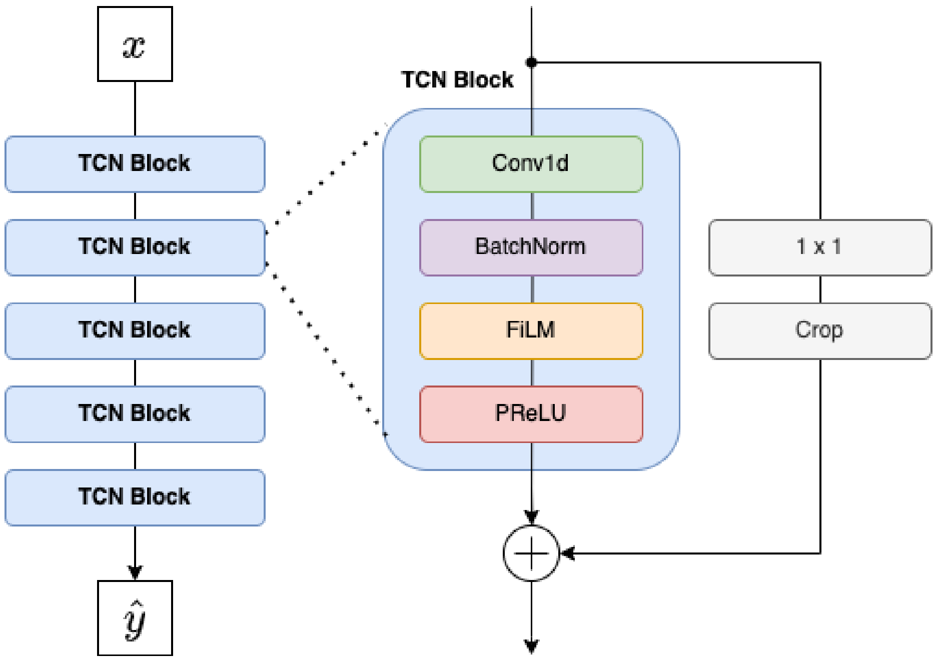


Figure 4: Block diagram representation of the baseline TCN architecture

The core component is the `1DConvolution` that forms the backbone of the TCN, working as a sliding filter across the time axis of the input signal. Additionally, there's a skip connection (residual) added to the output of the convolution, which helps training for deeper networks. `BatchNorm`, or Batch Normalization, is an essential normalization technique aimed at accelerating training and stabilizing the activations of neural networks. In this architecture, `BatchNorm` is introduced to normalize the output from each layer, ensuring that the activations do not reach extremely high or low values, thus addressing the internal covariate shift. By maintaining the activations within a certain scale, it promotes smoother optimization landscapes and a more efficient training process. Furthermore, FiLM (Feature-wise Linear Modulation) conditioning mechanism [43]. Following this, the architecture introduces the PReLU (Parametric Rectified Linear Unit) as its activation function. The inclusion of PReLU is noteworthy due to its ability to introduce non-linearity into the model, enabling the network to capture complex patterns and relationships in the data. Unlike the standard ReLU, which nullifies negative values, the PReLU allows a small gradient when the unit is inactive (i.e., for negative values), making it more flexible and often leading to better performance in deeper networks. This small gradient, determined by a learned parameter, ensures that even "inactive" units can adapt during the training process, reducing the risk of neuron "death" seen in some deep network trainings with standard ReLUs.

3.1.2 Feed-Forward WaveNet

The second architecture implemented is a simplified, feed-forward variant of the WaveNet model, employing the principle of causal convolutional layers. Each `CausalConv1d` ensures that the output at time t is only influenced by inputs from time t and earlier. This is achieved by adding specific padding, ensuring that no future information leaks into the current computation. Per each layer, a number of convolutional channel is established with the hyperparameters of the model, this represents the number of feature channels (or filters) used in the causal convolutional layers. Ultimately, it dictates the depth or richness of feature representation at each layer of the network. It's a design choice that allows to trade off between expressiveness and computational efficiency, this parameter can have a dramatic impact on the model's performance.

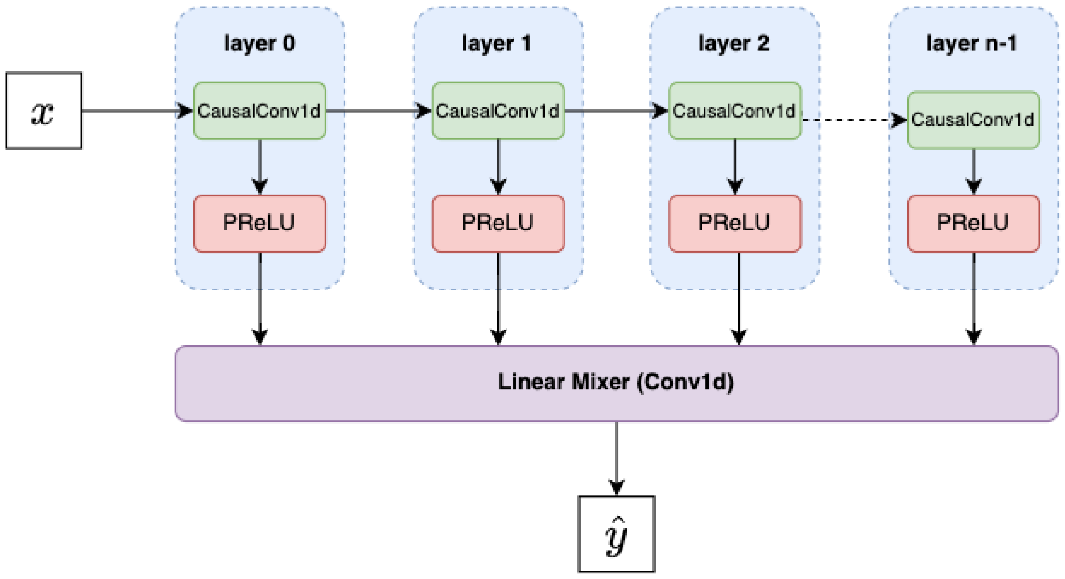


Figure 5: High-level block representation of the feed-forward WaveNet architecture.

The output from each convolutional layer (after the PReLU activation function) is stored. At the end of the network, the outputs of all the convolutional layers are concatenated together to form a combined feature map. The concatenated feature map is then passed through a 1x1 convolution, called "linear mix" in the code. This acts as a mixing function, allowing the network to weight the contribution of each feature map before producing the final output, as described by Wright et al. [8]. Finally, a function calculates the receptive field of the network, which is essentially the number of input samples that influence a single output sample. The receptive field grows with each layer due to the dilation growth in the convolutional layers.

3.1.3 Gated Convolutional Network

Similar to the feed-forward WaveNet architecture and the Temporal Convolutional Network, this architecture can be briefly described as a TCN that leverages the power of dilated convolutions combined with gating mechanisms to capture long-range dependencies without significantly increasing computational costs. The main GCN structure consists of a list of `GatedConv1d`

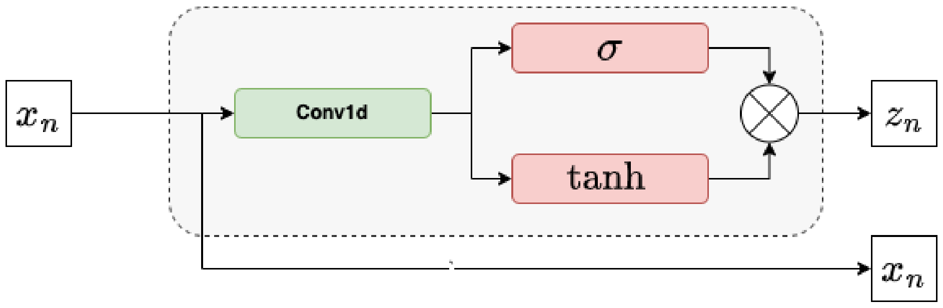


Figure 6: Block Diagram of the dilated 1 dimensional gated convolution.

layers. The number of such layers depends on the hyperparameters selected. Dilation increases exponentially with the layer depth, allowing the network to capture wider ranges of input context as depth increases. After reaching the maximum dilation at the final layer, it resets and starts again, marking the beginning of a new block. This structure helps in capturing multiscale patterns in the data. The outputs from all layers are stored and combined in a single tensor (`z_storage`) to which a final mixing convolution of kernel size 1 is applied and generates the network's output. This mixing ensures the network can use information from all layers in its final decision. The core component of the network is the `GatedConv1d` layer, each one has two convolutional operations. The first operation has its output channels doubled in size, and this output is split into two equal parts. One part undergoes a *tanh* activation, while the other undergoes a *sigmoid* activation. The element-wise product of these two activations creates the gating mechanism, effectively controlling the flow of information through the network. To maintain consistent tensor sizes throughout the network and enable residual connections, zero-padding is added at the beginning of the tensor after the gated operation. A mixing convolution of kernel size 1 follows the gating mechanism to control the number of output channels. The layer's output is a sum of the original input (residual connection) and the result from the mixing convolution.

3.1.4 Long Short-Term Memory Network

The provided code includes a basic implementation of an LSTM (Long Short-Term Memory) model using PyTorch's neural network library (`nn`). In this design, the model starts by receiving audio data, which is permuted to the preferred input shape for LSTM layers: batch, sequence, channel. It's worth noting that the LSTM layer is set up with a batch first configuration, optimizing for batch processing of sequential data. The number of LSTM layers and the number of hidden units in each are determined by the hyperparameters chosen [34]. By design, the model is able to handle bidirectional sequences, though in this instance, it's set to process sequences in a single direction. After the LSTM layer processes the sequences, the model employs a fully connected layer to map the output of the LSTM to the desired output size.

From the analysis of previous work, [33] and [8] emerges the use of a residual layer, also referred to as "skip connection". When enabled, the model selectively bypasses certain parts of the data, ensuring these initial features are directly carried to the output without undergoing transformations. This is useful for preserving specific characteristics or nuances in the audio. As in the previously exposed implementation, the subsequent linear layer maps the LSTM's outputs to the desired output dimensions. The Convolutional LSTM model with skip connection presented in this work (`LstmConvSkip`), is a hybrid, recurrent and convolutional, architecture design. The design aims to capitalize on the spatial feature extraction capabilities of CNNs and the temporal sequence modelling capabilities of LSTMs [19], [31]. The 1D convolutional layer acts as the feature extractor. By sliding the convolutional kernel across the input sequence, it captures localized spatial patterns. The kernel size and padding are set such that the output retains the same sequence length as the input. As a recurrent network layer, the LSTM processes the sequence step-by-step, maintaining an internal memory state. Additionally, it introduces an optional skip (or residual) connection, inspired by ResNet architectures [44], to improve learning and convergence in deeper networks. This linear transformation projects the input sequence directly to the output size, providing a shortcut or direct path. It can assist the

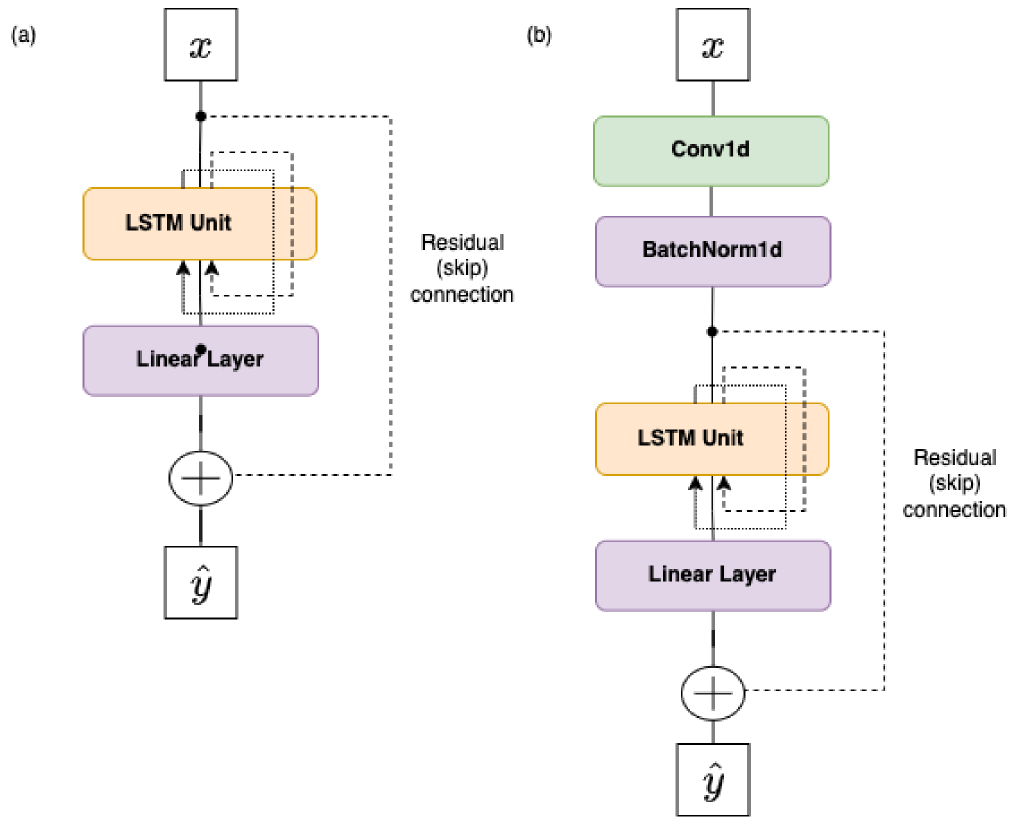


Figure 7: Block diagram of the LSTM architectures, (a): the basic model with the skip connection, (b): the convolutional feature extraction layer with Batch Normalization, then connected to a LSTM cell with skip connection.

model in learning identity mappings and, in some cases, can accelerate convergence.

3.2 Datasets

From Table 1 we see that specific to the spring reverb, there are only 2 datasets publicly available. Both those datasets use physical devices to record wet samples of guitar and bass notes. They are quite different either for the sampling rates and bit depths and for the format in which they are distributed. In this section, we analyse their characteristics and differences, providing some audio descriptors and visualizations.

3.2.1 SpringSet

The first dataset, named here: "SpringSet", introduced by Martinez Ramirez et al. [24], is derived from the clean samples of the IDMT-SMT-Audio-Effects [20] a large dataset of guitar and bass notes processed with effects other than the spring reverb. The dry samples selected from this parent dataset are processed with a spring reverb tank Accutronics 4EB2C1B, the dry mix is set to 0 %, and the wet mix to 100 %. All the recordings are downsampled to 16 kHz, each one has a length of 2 seconds and a fade-out is applied in the last 0.5 seconds of the recordings. Table 3 summarizes the main characteristics.

Data	**Samples**	**Length**	**Size**	**Min**	**Max**	**DC offset**	**SNR**
Train Dry	1122	37’24”	287 MB	-1.019	1.018	$-6.685 \cdot 10^{-7}$	0.201 dB
Train Wet	1122	37’24”	287 MB	-1.006	1.007	$-4.383 \cdot 10^{-6}$	0.196 dB
Test Dry	64	2’8”	16 MB	-1.003	1.003		
Test Wet	64	2’8”	16 MB	-1.006	1.007		
Total	2372	79’4”	606 MB				

Table 3: SpringSet overview: dataset characteristics and statistics, Min, Max, DC offset and SNR values represent the mean of the subset.

All the samples are compressed into four HDF5 archives and encoded with the data type NumPy "float64". While this dataset is distributed with only two parts: train and test, in our experiments we split it into 3 parts, adding one for the validation step. In terms of total length, this dataset appears to be more than sufficient for the task, as demonstrated in the available literature documenting similar research [8, 13, 19, 9, 14].

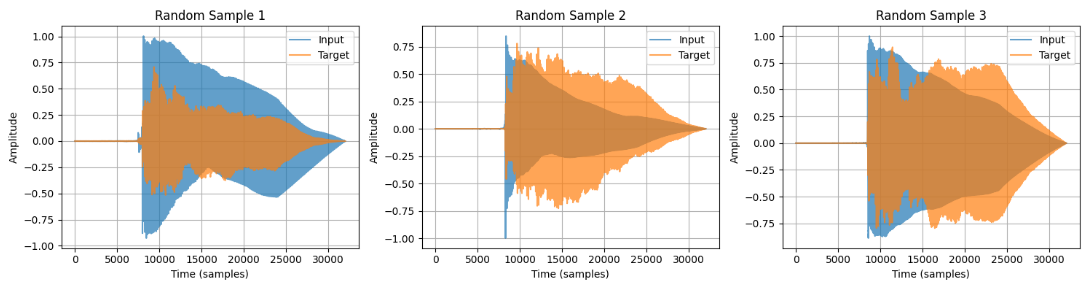


Figure 8: SpringSet: visualization of three random waveforms for input-target pairs overlapped.

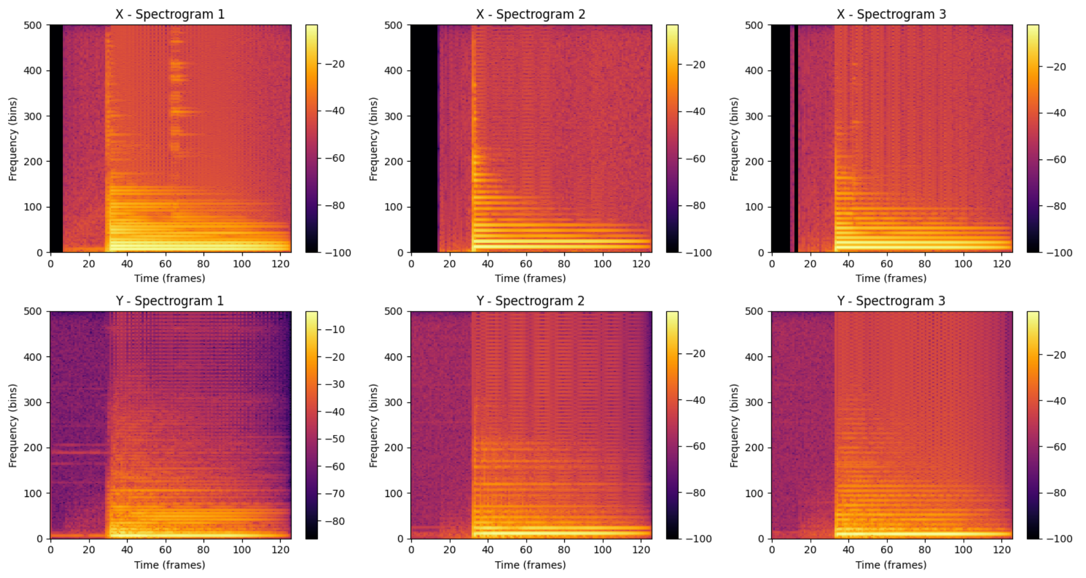


Figure 9: SpringSet: visualization of 3 spectrograms, comparing input (top row) - target (bottom row).

No metadata is provided with this dataset, and the original pipeline for the pre-processing is not publicly available. Given that, a preliminary analysis is performed on the data. For the scope of this work, only the subset related to the spring reverb samples is considered, discarding the samples related to plate reverb. Figure 8 and 9 show visual representations of the samples in the time and frequency domains. The review of a certain number of waveforms, confirms the irregularity in the dry-wet loudness level of the SpringSet, motivating further analysis. From listening to the samples a perceptible difference in loudness between dry and wet samples is present, some background noise is also, occasionally, present in the data. To verify these hypotheses and check the quality of the data, we compute the minimum and maximum array values, which confirm a maximum range going from -1 to 1. The DC offset values are quite low across the entire data, well below the minimum quantization value at 16 bits[2]. These figures are also be beneficial to ascertain the uniformity between the training and testing subsets.

3.2.2 EGFxSet

From the theorems at the foundations of digital signal processing [45] , [46], we know that at a given sampling rate SR the highest frequency represented without artefacts will have frequency $f = SR/2$ Hz, for this reason the current recording standard is at 44100Hz. Nevertheless, for some deep learning applications downsampling and upsampling might be appropriate, in general, working with neural networks at a lower resolution than the Nyquist rate can significantly shorten the training process and reduces the computational resources needed, facilitating the development process, especially in the early stages, This lightweight and flexibility come at the cost of a reduced spectrum, introduction of artefacts due to downsampling / upsampling operations and results in networks trained that can't operate at higher sampling rates successively (limiting their possibile inter-operability in a digital signal chain). These limitations become particularly relevant in the context of a reverb emulation, such as the spring reverb. High-frequencies play a key role in the representation of artificial reverberation, they define and characterize resonances and spatial perception in general, for a human listener those elements are essential for him to recognize the natural psychoacoustic and physical phenomenon.

The Electric Guitar Effects Dataset (abbreviated: "EGFxSet"), introduced at the end of 2022 by Pedroza et al. [12], is a collection of dry-wet pairs recorded from 12 popular guitar effects,

[2]The minimum quantization value for a 16-bit resolution is $3.05 \cdot 10^{-5}$

Data	**Samples**	**Length**	**Size**	**Min**	**Max**	**DC offset**	**SNR**
Dry	690	57'30"	502 MB	-1.000	1.000	$8.121 \cdot 10^{-6}$	0.596 dB
Wet	690	57'30"	502 MB	-0.891	0.899	$-1.39 \cdot 10^{-7}$	0.596 dB
Total	1380	115'	1004 MB				

Table 4: EGFxSet: overview of the dataset with its characteristics and some audio statistics. For Min Max and DC offset the mean values are shown.

including the spring reverb. A total of 8970 unique, annotated guitar tones, each of five seconds, it is published with full open-access rights, the main advantage of this dataset resides into being recorded at 48kHz and 24 bit resolution, the standard for broadcasting and film industry. For the purpose of this project, only the Clean and the Spring-Reverb sections are considered here. The device used is a CR60C digital, reverb emulation of an Orange Crush Pro 60 with a 12-inch speaker combo amplifier, it integrates Plate, Hall and Spring reverberations. To record the spring reverb clips, the volume dial has been set at 50%, the dry mix at 0% and the wet at 100% [12]. This dataset is not split at the origin, all the clean samples have their correspondent wet sample for each of the 12 audio effect. There is little difference in the peak level of wet samples compared with the peak of dry ones. The DC offset values measured are also very low.

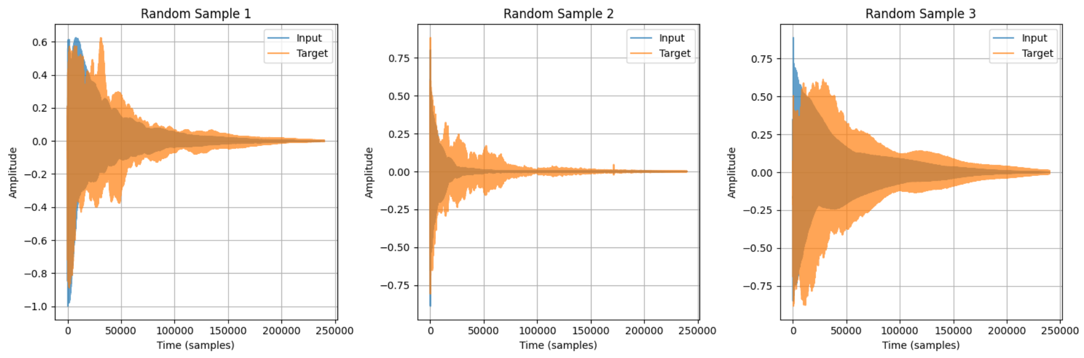


Figure 10: EGFxSet: visualization of 3 waveforms of input-target pairs.

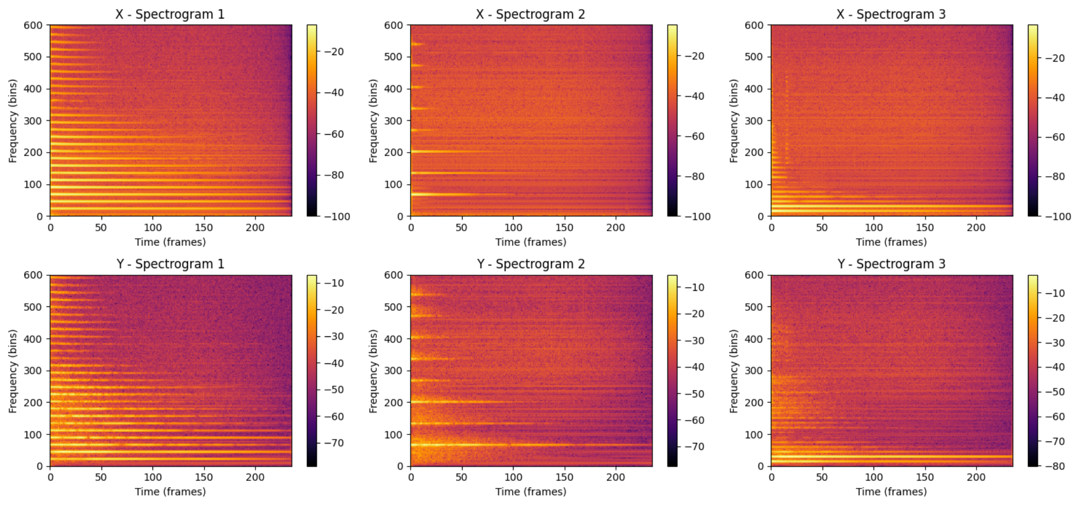


Figure 11: EGFxSet: visualization of 3 spectrograms comparing input-target pairs.

3.2.3 Audio Features Extraction

While basic numerical statistics may offer a valuable initial inspection, they only give a high-level view on the analysed data. This initial insight acquired can be further deepened by extracting audio descriptors. A preliminary analysis of loudness, pitch and noise level is done with `Librosa` and `Torchaudio`, then we use a selection of 9 algorithms from the `Essentia`[47] library to extract low level audio features across both datasets. These features are stored in json files that are available in the GitHub repository with the rest of the code. The algorithms selected are essentially aimed to validate the data pipeline and the split of the data into train, validation and test based on randomness, it also aims at checking that the no other audio problems might be present, here follows a summarization of them.

- **ReplayGain**: the distance to the suitable average replay level (-31dbB) defined by SMPTE standard and is expressed in decibels (dB). As a pre-processing step, the algorithm applies equal-loudness filtering to the input signal.

- **Leq**: the equivalent sound level estimate of the audio signal. This measure can be derived from the Revised Low-frequency B-weighting (RLB) or from the raw signal. If the signal contains no energy, the default value for silence is set at -90dB.

- **Loudness**: the loudness of the audio signal as defined by Steven's power law: i.e. as the energy of the signal raised to the power of 0.67.
- **FlatnessDB**: this algorithm computes the ration between the geometric mean and the aritmetic mean converted to dB scale of an input signal. This is useful to measure how a sound is "noise-like" or on the contrary "tone-like". White-noise would theoretically have a value of 1.0, a pure tone, on the contrary, would present a value close to 0.0.
- **HFC**: the High-Frequency-Content of a spectrum computed according to the 'Masri' technique. This measure is purely mathematical and have a perceptual relevance.
- **SpectralCentroidTime**: spectral centroid of a signal in the time domain. A difference filter is applied to the input, then the centroid is computed by dividing the norm of the resulting signal by the norm of the input. It is expressed in Hertz (Hz).
- **ZCR**: Zero-Crossing-Rate, i.e.: the rate at which the input signal changes of sign. Often used to detect percussive sounds.
- **PitchYin**: 'implementation of the Yin algorithm for the estimation of the fundamental frequency in the time domain. The algorithm returns two values, one for the estimated f0 and one for the pitch confidence.

Upon examining the extracted audio features in Figure 12, we can derive a few observations about the SpringSet:

- Wet samples ("y") in the SpringSet, are generally lower in Leq level than dry samples ("x"), Loudness standard deviation also shows higher values for y samples.

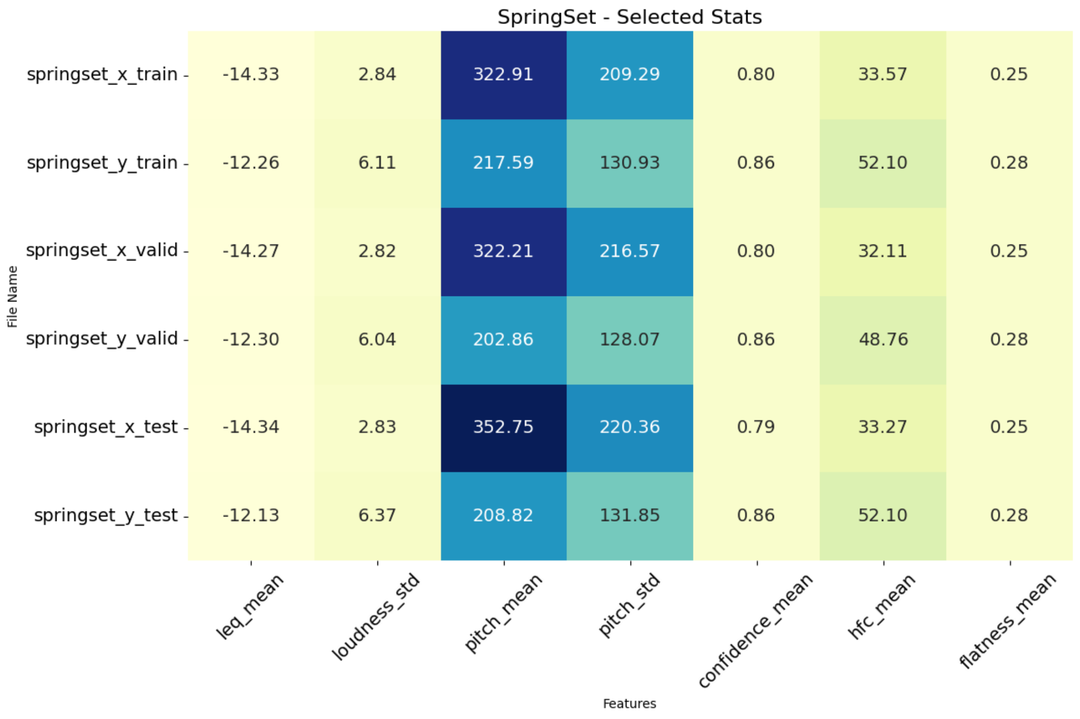


Figure 12: SpringSet: selection of mean and standard deviation values from the extracted features.

- There's a high confidence score on the pitch estimation that allows the classification of guitar and bass notes. This results into approximately 10% of bass notes over the ensemble of the samples for the three splits.

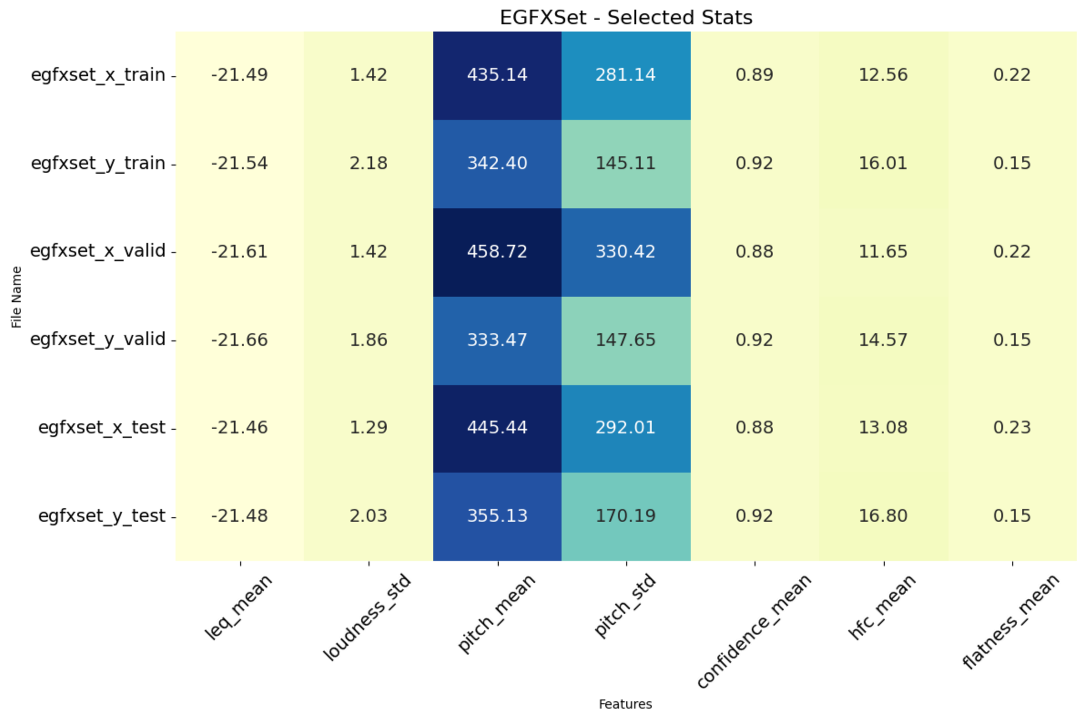
- In general, processed sounds have lower pitches and tend to be more clustered, the effect of the spring reverb samples acts also a low-pass filter, making the y samples "darker" compared to their x pairs.

- The higher values of the HFC coefficient, for the processed sounds, might correspond to the resonances and the reflections added by the springs during the processing of the input samples. This descriptor isn't necessarily correlated with the pitch.

- Flatness is fairly low all across the data, this is coherent with the fact that the dataset is composed essentially of guitar and bass notes with a low amount of noise and no particular sound with a noisy or percussive envelope.

The same algoritms are computed for the EGFxSet, the results are presented in Figure 13.

- The loudness across the data presents a standard deviation between 1 and 2dB. Compared with the SpringSet here, the difference is lower, the data seems more consistent.
- Pitch is generally higher as well as its standard deviation. The recordings used for this dataset are divided into 5 different positions of execution on the guitar: Bridge, Bridge-Middle, Middle, Middle-Neck, Neck. This corresponds into a more distributed selection of notes that includes more high-pitched ones.
- As with the other dataset, wet samples have a higher HFC coefficient.


EGFXSet - Selected Stats

File Name	leq_mean	loudness_std	pitch_mean	pitch_std	confidence_mean	hfc_mean	flatness_mean
egfxset_x_train	-21.49	1.42	435.14	281.14	0.89	12.56	0.22
egfxset_y_train	-21.54	2.18	342.40	145.11	0.92	16.01	0.15
egfxset_x_valid	-21.61	1.42	458.72	330.42	0.88	11.65	0.22
egfxset_y_valid	-21.66	1.86	333.47	147.65	0.92	14.57	0.15
egfxset_x_test	-21.46	1.29	445.44	292.01	0.88	13.08	0.23
egfxset_y_test	-21.48	2.03	355.13	170.19	0.92	16.80	0.15

Features

Figure 13: EGFxSet: selection of mean and standard deviation values from the extracted features.

3.3 Experimental design

In developing an experimental environment, our primary goal is to ensure both clarity of results and ease of replicability, the design of such environment is structured in several steps to achieve this goal:

Reproducibility - With a commitment to transparent and open research, we place great emphasis on the ease of replicating our experiments. Through consistent setups, fixed random seeds, and comprehensive documentation, we strive to ensure that our results are both verifiable and easily adoptable by the broader research community.

Preliminary Steps - The explorative phase of the project laid the groundwork for our experiments. Initial tests were vital to understanding the potential and limits of our proposed methods. This initial phase is instrumental in setting the stage for more in-depth experimentation and to gain insight on the context into which to develop a comparative study.

Benchmarks and Baselines - To judge the efficiency and accuracy of the models considered, it is crucial to have benchmarks. We identify three baselines to provide diverse performance metrics, further detail on those will be given in the next chapter.

Experiments on Different Sampling Rate and Bit Depth - For computational resource optimization, the entire first part of the work has been done on 16kHz, 16 bits data. Successively a higher resolution of 48kHz, 24 bits have been adopted to match the industry standard for audio processing.

Code Structure - Our project's codebase is organized systematically, ensuring modularity and scalability. This structure facilitated both the integration of new models and the tweaking of existing ones.

Command Line Interface - For ease of interaction with our models and experiments, a command-line interface (CLI) is designed. This CLI not only streamline the process of model training and evaluation, but also ensures a consistent and easy-to-use environment for all our experiments.

Documentation - Comprehensive documentation is maintained throughout the project. This ensures that any researcher, either extending our work or trying to reproduce our results, has a clear and detailed guide, minimizing the knowledge gap and maximizing clarity.

Hyperparameters	**TCN**	**WaveNet**	**GCN**	**LSTM**
Channels	8 - 64	8 - 64	8 - 32	16 - 128
Dilation	1 - 12	1 - 12	1 - 8	-
Kernel Size	1 - 19	1 - 19	3 - 41	0 - 3
Layers	1 - 10	10 - 24	1 - 10	1 - 4
Common Ranges for All Models				
Batch Size	1 - 32			
Learning Rate	$10^{-1} - 10^{-4}$			
Total Configurations	**30**	**25**	**15**	**15**

Table 5: The "search space": hyperparameters and ranges assessed during the initial scaling phase.

3.3.1 Hyperparameter Scaling

Hyperparameter tuning and model selection, especially across multiple architectures, is a meticulous process that requires a structured approach to ensure optimal outcomes. Once the architectures are implemented, the process of scaling follows these steps:

- **Defining the search space**: specify the range and type of hyperparameters under consideration, Table 5 summarizes the main parameters and their ranges.

- **Progressive pruning:** Systematic removal of the models that underperform, ensuring that only potential contenders proceeded to subsequent stages of assessment.
- **Detailed evaluation of shortlisted models:** a meticulous evaluation process, with a focus on their generalization capabilities, computational efficiency, and robustness.

We proceed to approximately 300 runs of training for short amounts of epochs (below 50) to define the models that respond to the criteria before mentioned. This process leads to the definition of 9 models, detailed in Table 6.

Model	CH / HS	Dilation	Kernel Size	RF (ms)	Parameters	Layers
TCN-1800	32	10	9	5000	31.5k	5
TCN-4000	32	10	19	4000	62.9k	5
WaveNet-10	16	10	3	160	18.6k	10
WaveNet-18	16	10	3	215	33.4k	18
WaveNet-24	16	10	3	47	44.6k	24
GCN-250	16	6	41	250	65.6k	10
GCN-2500	16	3	5	2500	26.4k	10
LSTM-cs-32	32	-	3	-	17.1k	1
LSTM-cs-96	96	-	3	-	75.2k	2

Table 6: Hyperparameters of the selected models after the scaling process. CH: channels (convolutional layers), HS: hidden size of LSTM, RF = Receptive Field.

3.3.2 Training

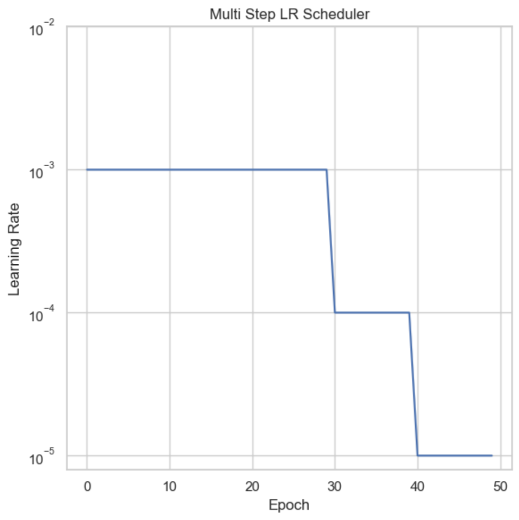
The training process is implemented using PyTorch version 2.0.1, encompassing both training and validation loops across all models. For both datasets, 50% of the samples is used for training, 25% for validation and another 25% is reserved for the evaluation process. For every validation epoch, a checkpoint with the model parameters is saved if the loss has decreased

Dataset	**SR**	**BS**	**Init. LR**	**Epochs (max)**	**ES Patience**
SpringSet	16kHz	16	0.005	1000	25
EGFXSet	48kHz	8	0.005	250	25

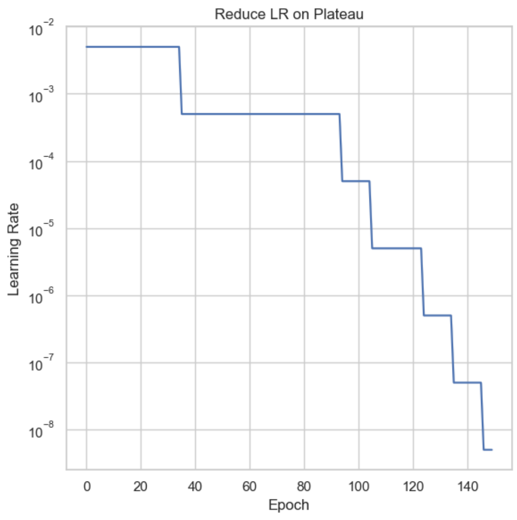
Table 7: Training configurations for each dataset, SR: sample rate, BS: batch size, LR: learning rate, ES: early stopping.

with respect to the previous validation loop. To track metrics like the learning rate, average training loss, average validation loss, example audio files (including input, output, and target), and spectrograms, Tensorboard [48] is employed for logging. Some experiments are conducted on the most common optimizers for similar tasks, Adam together with `ReduceLROnPlateau` as scheduler seems to work better in this context. A patience range is tested, spanning from 50 epochs down to 10. The initial learning rate is another area of exploration, with values tested between $1 \cdot 10^{-5}$ and $5 \cdot 10^{-1}$. The rate of 0.005 seems to guarantee the best balance in our experiments. Batch sizes also underwent rigorous testing [49]. For data at 16kHz, a batch size of 16 is optimal, while the 48kHz data has better results with a batch size of 8. This difference also responds to the limited computing resources available and the difference in necessary memory when working at 48kHz. A single Nvidia Tesla T4 GPU with 16GB of RAM and CUDA version 12.0 is used for the whole experimental process.

From an analysis of the previous literature in this field emerges a consistent set of loss functions widely used for successful modelling of audio effects. In particular, as shown in Table 2, Mean Absolute Error, Error-to-Signal-Ratio and Multi-Resolution STFT, alone or in combination, appear to be quite common as loss functions for tasks similar to the one approached here. To ensure thoroughness in our approach and potentially identify the most effective loss function for our needs, we systematically test various loss functions - MR-STFT, STFT, ESR, MAE, and DC - across all architecture configurations. These tests on baseline models with subsets of the data reveal that a combined loss of MAE and Multi-Resolution STFT represent the optimal



(a)



(b)

Figure 14: An example of two different schedulers adjusting learning rate during training: `MultiStepLR` (a), `ReduceLROnPLateau` (b).

solution. The overall loss is given by:

$$L = \alpha L_{MAE} + (1 - \alpha) L_{MR-STFT} \tag{3.1}$$

with $\alpha = 0.5$. The rationale behind the use of time and frequency domain losses is to ensure that the model not only captures the overall structure and envelope of the audio waveform, but also respects the harmonic complexities in the spectral domain. Loss in the time domain, represented by MAE, is more sensitive to phase discrepancies and temporal structures, while loss in the frequency domain, represented by MR-STFT, focuses on ensuring that the spectral characteristics of the processed audio closely match those of the original [24].

Chapter 4

Results

In this chapter, we delve into the core findings of our research, meticulously examining the results of experiments conducted on 10 distinct models. These models, a mix of convolutional and recurrent architectures, were carefully chosen during the hyperparameter optimization phase discussed in the preceding chapter. We've undertaken these experiments as comparative studies, aiming to discern the most optimal architecture for accurately modelling the spring reverb effect. Our results suggest that hybrid neural network patterns exhibit exceptional performance across various evaluation criteria. Such models not only demonstrate enhanced adaptability to data, but they also seem particularly suited for actual use cases in a production context. Furthermore, we also evaluate the computational efficiency of the trained models, and we propose the adoption of impulse response and RT60 as analytical measurements deeply connected with reverberation either as a physical phenomenon as well as audio effect.

4.1 Objective Evaluation

4.1.1 Baselines

While the loss functions delineated earlier in Chapter 2 primarily guide the training phase, they are also adopted as evaluation metrics during testing. Quantitative metrics don't always fit within predefined bounds: when there's no previous work on the same data or for a similar task, it is difficult to establish without a baseline the range of possible values. Defining and testing a baseline is essential for the evaluation process based on objective metrics. Consequently, we define and test three baseline models, built to ascertain potential metric ranges, thus aiding the subsequent evaluations of the model under comparison, the results are summarized in Table 8.

Baselines	**MAE**	**ESR**	**DC**	**MR-STFT**
TCN-baseline (16kHz)	0.196	4.480	0.1601	2.555
Naive-baseline	0.193	1.396	$5 \cdot 10^{-6}$	1.897
Noise-ceiling	0.219	1.922	$5 \cdot 10^{-6}$	2.914

Table 8: Baselines results for the selected objective metrics.

1. **TCN-baseline**: Serving as our primary baseline, the Temporal Convolutional Network (TCN), as presented in Chapter 3, is trained on a sequence made of concatenated samples belonging to the SpringSet. Its training is carried for 2500 steps without validation, the metrics are computed on a testing sequence, unseen by the model during training.

2. **Naive Baseline**: Drawing inspiration from the `DummyRegressor` in the Python package `Scikit-learn`, this model emulates a prediction that corresponds to the wet sample and a target replaced by the dry sample. The metrics are computed between the dry and wet signals of the dataset. The underlying logic of this regressor is to provide a quantitative measure of the inherent variance of the target device that we are modelling.

3. **Noise Ceiling**: Represents the upper boundary, sometimes referred to as the "topline". It is designed to make a random prediction that is uncorrelated with the input and the target. Metrics are computed between a target and the random tensor (similar to white noise) that represents this baseline.

To ensure meaningful and significant results in the context of our evaluation, the chosen models for further consideration and deployment should consistently outperform these baselines across the different objective metrics.

4.1.2 Testing

16kHz, 16bit - For each of the datasets used in this study, 25% of the samples are reserved solely for the final test on data that the model has not seen during training or validation. Table 9 shows the evaluation metrics for the selected models on the SpringSet. The Gated Convolutional Network performs significantly better on all the evaluation metrics, except for the Multi-Resolution STFT where the LSTM-96 leads. This is a rather simple architecture with a single recurrent layer followed by a linear layer just before the output, its biggest drawback is the training time that is 3 to 5 times longer than other models for the same amount of steps / data. On MAE, ESR and MR-STFT all the selected models perform better than the baselines presented earlier, supporting the selection process carried until this point and the hyperparameters scaling. The only exception to this improvement is DC offset, it is significantly lower for the baselines than for the models under evaluation. This is partially due to the calculation of DC offset itself, as defined by Wright and Valimaki [40], for the Naive baseline and the noise ceiling, since there is no processing through neural networks before the computation of the metrics the average value measured is very close to 0, either on negative or the positive side. With these insights from the results on the SpringSet, we now pivot our attention to the EGFxSet results, this transition allows us to evaluate how our models generalize across different datasets, revealing any architecture-specific nuances or tendencies.

Model	**Parameters**	**MAE**	**ESR**	**DC**	**MR-STFT**
TCN-baseline	$3.15 \cdot 10^4$	0.194	0.101	1.44	0.00066
TCN-4000	$6.29 \cdot 10^4$	0.148	0.0583	0.824	0.000761
GCN-250	$6.56 \cdot 10^4$	**0.0658**	**0.012**	**0.223**	0.000242
WaveNet-18	$3.34 \cdot 10^4$	0.173	0.092	1.31	0.000619
WaveNet-24	$4.46 \cdot 10^4$	0.123	0.0519	0.749	0.000457
LSTM-96	$3.81 \cdot 10^4$	0.146	0.0553	0.772	$\mathbf{7.21 \cdot 10^{-5}}$
LSTM-cs-96	$7.52 \cdot 10^4$	0.178	0.0821	1.13	0.000268

Table 9: Test results for SpringSet (16kHz). Lowest values highlighted in bold.

48kHz, 24bit - Table 10 summarizes the metrics for the EGFxSet test. The GCN-250 model consistently outperforms its counterparts in three of the four metrics. This underscores its ability to generalize well across different datasets, suggesting its architecture is particularly suitable for these audio tasks. This phenomenon is evident from the performances of GCN and WaveNet models. While GCN-250 achieved the best results in terms of MAE, ESR, and DC, the TCN-4000 surpasses in the MR-STFT metric. This highlights the importance of multi-metric evaluations: no single model may be the best across all metrics, and trade-offs must be considered. The results don't show a strict linear relationship between the number of parameters and performance. For example, the GCN-250 with 65.6k parameters outperforms the GCN-2500 with 26.4k in most metrics despite the latter having fewer parameters. The advantage of convolutional architectures, particularly those equipped with gated activation functions, is apparent in these results. Evaluated under the light of objective metrics, they are adapted for capturing and modelling audio effects with prolonged temporal dependencies.

In neural network research, quantitative analysis, supported by proven metrics, is a valuable tool. It provides an objective basis for measuring progress, facilitating comparisons and ensuring consistency of evaluation. However, an exclusive reliance on these metrics is inappropriate, especially when studying neural networks for music and audio. As we pointed out in the first

Model	**Parameters**	**MAE**	**ESR**	**DC**	**MR-STFT**
GCN-2500	26.4k	0.087	0.035	3.1	0.000380
GCN-250	65.6k	**0.029**	**0.004**	**0.616**	0.000058
LSTM-cs-32	17.1k	0.080	0.029	3.0	0.000170
LSTM-cs-96	75.2k	0.036	0.005	0.65	0.001610
TCN-4000	62.9k	0.084	0.031	3.08	**0.000043**
TCN-baseline	31.5k	0.042	0.008	0.842	0.005130
WaveNet-10	18.6k	0.035	0.006	0.683	0.000094
WaveNet-18	33.4k	0.081	0.030	3.22	0.000140
WaveNet-24	44.6k	0.081	0.029	3.21	0.000470

Table 10: Test results for EGFxSet (48kHz). Lowest values in bold.

chapter, the task of modelling an electromechanical reverberation device such as spring reverberation is quite challenging and requires an interdisciplinary approach, capable of adopting multiple points of view. This mentality leads to the need for further measurements and tests that are more related to the sound characteristics captured by the models during the training process.

4.2 Efficiency

Utilizing neural networks for real-time music applications presents a challenge influenced by the trade-off between computational efficiency and model performance. There are two main numerical measures for assessing a model's suitability for real-time deployment within a common digital audio workstation (DAW) workflow.

$$RTF = \frac{time(decode(a))}{length(a)} \tag{4.1}$$

Real Time Factor (RTF) - As introduced above, this is an intuitive metric to ascertain the possible deployment real-time of a system. It measures the time it takes the system to process an input compared to the actual duration of that input. As a rule of thumb: an RTF less than or equal to 1 indicates that the system is employable in real time. On the other hand, an RTF greater than 1 indicates a delay in processing. For example, an RTF of 2 implies that the system requires twice the duration of the input to process it. Using the inference on CPU, it is possible to measure the time required by the models during testing to process the data and compute the RTF. Although this approach may not offer a completely objective assessment of the deployment in a framework like Neutone or JUCE, it is still valuable as it sheds light on the computational demands posed by the model and allows a better tuning to improve its performance. From the results summarized in table 11, when operating at 16kHz the slowest model (WaveNet-18) takes 0.438 seconds to process 1 second of audio at the same resolution, the fastest model being the WaveNet-24 taking 0.157 seconds per second of audio processed.

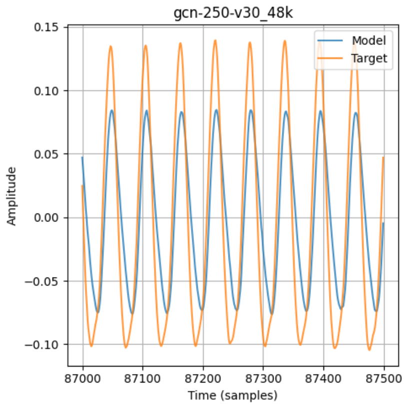
Computational Load - Is the amount of memory required by a model to run an inference. This is another critical aspect to consider in the design and the development of a neural network architecture, particularly in audio applications. The values, reported in Table 11, refer to an inference running on an Apple MacBookPro laptop with CPU Intel i7 and 16Gb of RAM. Despite having a similar number of parameters to other models: WaveNet exhibit a high memory consumption, probably due to the stratification of many hidden layers and connections. In contrast, GCN models, even with higher parameter counts, seem to be more memory efficient.

		48kHz		16kHz	
Model	**Parameters**	**RTF**	**CL**	**RTF**	**CL**
GCN-2500	26.4k	0.713	3.58GB		
GCN-250	65.6k	**0.059**	1.48GB	0.176	187MB
LSTM-cs-32	17.1k	0.665	**756MB**		
LSTM-cs-96	75.2k	1.349	2.23GB	0.422	297MB
LSTM-96	38.1k	-	-	**0.036**	99MB
TCN-4000	62.9k	1.114	2.98GB	0.091	764MB
TCN-1800	31.5k	0.606	2.98GB	0.185	396MB
WaveNet-10	18.6k	0.851	3.82GB		
WaveNet-18	33.4k	1.411	6.67GB	0.479	6.77GB
WaveNet-24	44.6k	1.810	8.98GB	0.161	8.98GB

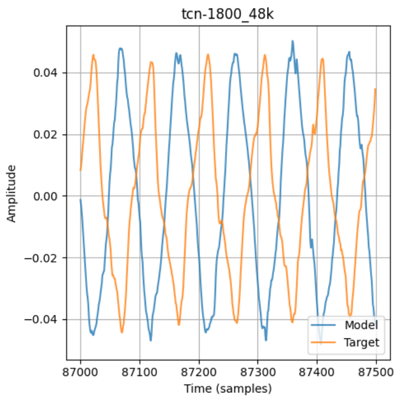
Table 11: Real Time Factor (RTF) and Computational Load (CL) of 48kHz and 16kHz models. Lowest score in boldface.

4.3 Techniques for Neural Audio Effect Analysis

Every audio effect can be associated with one or more perceptual attributes of sound. As an example: compression is related to loudness, distortion to the distribution and amplitude of the perceived harmonics, while reverberation to spatial perception and timbre. Digital modelling of a given effect has to somehow reproduce the related transformation in the sound signal in a manner that will cause in the listener the perception of that effect. Translating this process to neural audio modelling demands a specific set of tools in order to "steer" the learning process of the neural network towards the best possible approximation of the effect that we want to emulate. From Acoustics and digital signal processing, the study of artificial reverberation rely mainly on two techniques to measure the characteristics of reverberation: the Impulse Response (IR) and the Reverberation Time (RT60)[3].



(a)



(b)

Figure 15: The model prediction, in blue, and the target sample, in orange. In many cases, the simple visual inspection of random samples retrieved during training and testing can help to diagnose problems in the experimental design or in the learning process of the network. The GCN-250 (a), seem to attenuate the level compared to the target, despite normalization. On the right, the TCN-1800 (b) has repeatedly shown problems in the alignment of the predicted signal with the target.

4.3.1 Impulse Response (IR)

Angelo Farina in the year 2000 [50], standardized a method that, using a sinusoidal sweep, with a single measurement is able to measure and describe a system in the time and frequency domains. This process can be breakdown in these steps:

1. Generate a sine sweep tone, i.e. a sinusoidal sound that goes logarithmically from a $f0$ to $fmax$, together with that we generate its inverse filter, which is a time-reversed version of the first sweep. Both have the same length and the same $f0$ and $fmax$.
2. The first sine sweep is played through the system under measurement, which could be a room, an analog device or the inference of a neural network, its output is recorded.
3. The processed sweep is convolved with the inverse filter: with this operation, we isolate the system impulse response (IR) from the noise that is present in the captured version of the impulse response.

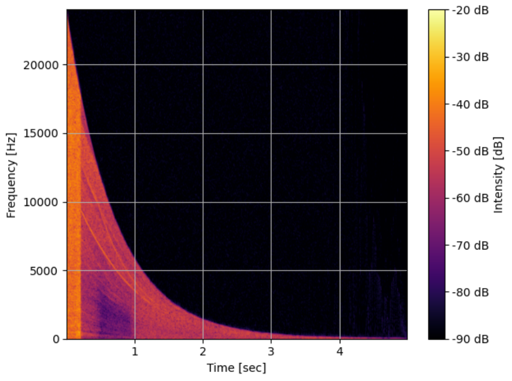


Figure 16: 2D plot of the IR recorded from the GCN-250 48kHz, only the second half of the recordeding is selected.

The significant advantage of this technique is providing a representation of the IR of the device under study using just a single, relatively straightforward measurement. In this study, we measure impulse responses for all our models, employing these measurements for fine-tuning and evaluating perceptual aspects of the trained models, this step is also essential for analysing reverberation time.

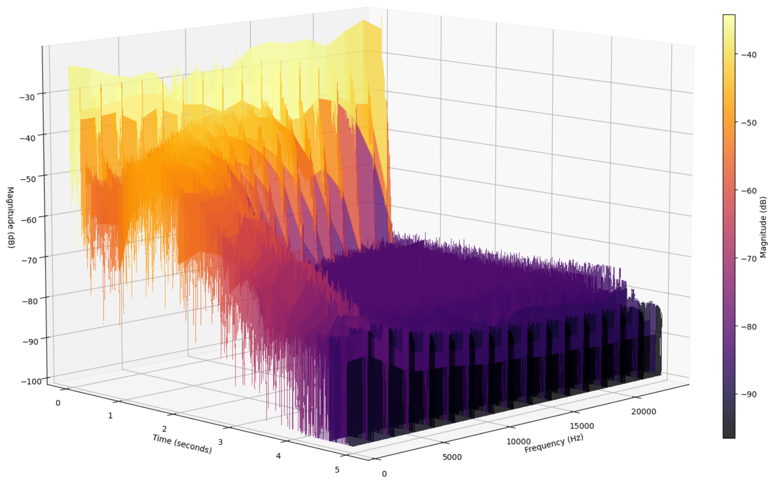


Figure 17: 3D visualization, "waterfall", of the impulse response obtained from the GCN-250 (48kHz) model, showing the magnitude (z) of the different frequencies (x) over time (y). Such tools are essential to interpret the information gathered by the measurements.

Model	RT60
GCN-250-48.0k	3946 ms
GCN-2500-48.0k	3003 ms
LSTM-CS-32-48.0k	4927 ms
LSTM-CS-96-48.0k	3538 ms
TCN-4000-48.0k	1898 ms
TCN-baseline-48.0k	1851 ms
WaveNet-10-48.0k	3449 ms
WaveNet-24-48.0k	3735 ms
WaveNet-18-48.0k	3580 ms

Figure 18: RT_{60} values for each model.

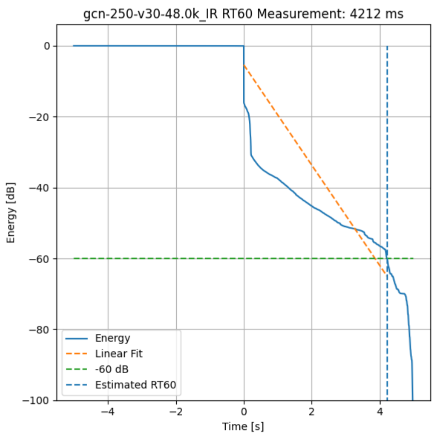


Figure 19: RT_{60} computed of the GCN-250 (48kHz).

4.3.2 Reverberation Time (RT)

In 1965, M.R. Schroeder, introduced a new methodology for measuring reverberation time [51], this is based on the use of short pulses of acoustic energy (traditionally tone bursts or gun shots) to excite the space being studied to then measure the decay time of the sound between two predefined levels. From this definition there are different deltas commonly used for this measurement: 20dB, 30dB and 60dB, these differences are stated as RT_δ, in which δ is the difference in decibels. As an example, the measure of the RT_{60} is done calculating the decay time from the point in which the sound has -5dB with respect to its peak to the level of -65dB, the -5dB threshold is introduced because in acoustic measurements this part of the signal consist mainly of early reflections and other types of perturbations that are not relevant to compute the actual reverberation.Measuring the IR is a preliminary step for determining reverberation time. In our experiments, we consistently measured the RT_{60} of the trained models with this technique, the results of our measurements are shown in Table 11 together with an example of 2D plot visualizing the RT_{60}.

Chapter 5

Discussion

This final chapter provides a comprehensive review of the research findings presented in this thesis. It begins with a summary of the main conclusions and our contributions, placing them in the context of the overall research objectives. The subsequent section will address the inherent limitations encountered in the course of this research, ensuring a transparent and thorough understanding of the scope and boundaries of the results presented. Finally, an overview of potential future work will be presented, highlighting areas that merit further investigation based, also, on the knowledge gained with this work.

5.1 Conclusions

Modelling a nonlinear, time-invariant audio effect such as the spring reverb using neural networks represents a significant challenge, that is at the core of this study. The strategies underscored in the title emphasize the intricacy of the problem and the necessity for meticulous attention throughout the entire process of designing a neural audio effect.

Our experimental findings highlight that all tested models surpass the benchmarks across most metrics, with the sole exception of DC offset. Convolutional networks, especially those leveraging gated activation function, like the GCN-250, standout in terms of quantitative evaluation. LSTM models, while advantageous in computational efficiency, face longer training and inference times compared to the other networks studied and don't seem to outperform the other models in any of the quantitative metrics here considered. We've improved significantly their performance applying a convolutional layer and batch normalization as feature extractors, and we could observe much less sensitivity to alignment issues. WaveNets, the most resource-intensive models considered here, also showed good predictions with low artefacts and rare alignment issues, on the other hand, this architecture can be very complex to train on limited GPUs without necessarily yielding higher evaluation scores. As we approach the end of this study, the synthesis of these elements suggests that tackling the intricacies of this complex challenge may indeed require a hybrid approach under the neural network architecture point of view. Ideally, a system that would combine the strengths of both recurrent and convolutional neural network classes. Throughout our exploration, it's clear that all the models under scrutiny exhibit superior performance when compared with the baselines. However, a common feature emerges: the resulting impulse responses tend not to have the desired reverberation, this aspect requires further investigation.

5.2 Contributions

- Among our contributions, a key one in this research is the creation of a complete experimental pipeline with fully reproducible steps and an extensive documentation. This constitutes the core of the background work that has been done during this thesis.
- A notable result of this effort is the ability of the models to operate at the industry-standard resolution of 48 kHz, 24-bit. In perspective, this achievement amplifies the practicality and real-world application of the developed models.
- Incorporating metrics such as impulse response (IR) and reverberation time (RT60) enhances our ability to address these challenges. While uncommon in the deep learning

literature, these metrics prove to be indispensable tools.

- Other critical indicators such as real time factor (RTF) and memory consumption offer insights that go beyond mere performance, shaping the very architecture of these systems.

5.3 Limitations

Every research has its constraints, the following are the key limitations of this study.

Dataset Constraints The primary limitation is the restricted nature of the datasets used. Training data was sourced from a limited selection of electric guitars. While this specificity might enable better performance for those particular inputs, it may limit the generalization capability of the models to handle other instruments or audio sources.

Computational Boundaries The computational constraints during the research phase meant that the models, especially the deeper architectures, might not have been trained to their full potential. Extended training could have possibly yielded better results.

Depth vs. Efficiency Trade-off The challenge of balancing between the depth (and consequently the potential accuracy) of a model and its computational efficiency has been highlighted and documented. Although deeper models appear to offer better performance, they often do so at the cost of higher computational demands, which affects real-time usability.

Evaluation Metrics The metrics utilized predominantly provided a quantitative perspective. Even though they offer objective insights, they clearly do not capture the subjective nature of audio perception completely.

5.4 Future Work

The findings and conclusions drawn from this thesis provide a foundation for future research in the field of neural audio effect modelling. Given the rapidly evolving nature of deep learning and audio processing techniques, numerous strategies might be adopted for future research.

Model Optimization Leveraging methods such as knowledge distillation, where a smaller model is trained to mimic the behaviour of a larger model, might allow for more efficient models without compromising on performance. Ensemble techniques or other forms of optimization are certainly worth further experiments.

Human Perception Assessment The metrics utilized in this study provide a quantitative assessment of model performance. However, audio perception is inherently subjective, and these metrics might not always correlate with human perception. Future studies can involve more extensive perceptual evaluations, where human listeners are involved to rate the quality of the generated audio effects. Such assessments can offer invaluable insights that purely quantitative metrics might miss. The Multiple Stimuli with Hidden Reference and Anchor (MUSHRA) methodology is widely recognized as an intermediate audio quality assessing tool. Furthermore, the Perceptual Evaluation of Audio Quality (PEAQ) as introduced by Imort et al. in [52] can also be applied for a perceptual evaluation based on the degree of impairment perceived by the listener.

Incorporating User Controllable Parameters Introducing feedback mechanisms within the models can emulate more traditional artificial reverberation effect units and might lead to richer and more dynamic audio effects. This and other traditional DSP techniques to control the pre- and post-stages of the neural audio effect can significantly improve its real-world usability and respond better to the original device behaviour.

This thesis aimed to shed some light on the challenges and possibilities inherent in emulating the spring reverb, using deep learning techniques. The results have provided insights into the trade-offs and decisions necessary in the design and deployment of such systems, with emphasis on the balance between model performance and computational efficiency. While the journey to a perfect emulation continues, this research serves as background work that seeks to identify and trace paths to be explored. Neural audio effects are evolving rapidly, and the synergy between traditional signal processing and deep learning techniques offers opportunities for future innovations towards expanded creativity.

List of Figures

List of Tables

Bibliography

[1] Zolzer, U. *DAFX: Digital Audio Effects* (Wiley, Chichester, West Sussex, England, 2011), 2nd ed edn.

[2] Levitin, D. J. *This Is Your Brain on Music: The Science of a Human Obsession* (Plume, New York, 2006).

[3] Valimaki, V., Parker, J. D., Savioja, L., Smith, J. O. & Abel, J. S. Fifty Years of Artificial Reverberation. *IEEE Transactions on Audio, Speech, and Language Processing* **20**, 1421–1448 (2012).

[4] van den Oord, A. *et al.* WaveNet: A Generative Model for Raw Audio (2016). `1609.03499`.

[5] Stoller, D., Ewert, S. & Dixon, S. Wave-U-Net: A Multi-Scale Neural Network for End-to-End Audio Source Separation (2018). `1806.03185`.

[6] Kalchbrenner, N. *et al.* Efficient Neural Audio Synthesis (2018). `1802.08435`.

[7] Vanhatalo, T. *et al.* A Review of Neural Network-Based Emulation of Guitar Amplifiers. *Applied Sciences* **12**, 5894 (2022).

[8] Wright, A., Damskägg, E.-P., Juvela, L. & Välimäki, V. Real-Time Guitar Amplifier Emulation with Deep Learning. *Applied Sciences* **10**, 766 (2020).

[9] Steinmetz, C. J. & Reiss, J. D. Steerable discovery of neural audio effects. In *NeurIPS 2021 Workshop on Machine Learning for Creativity and Design* (arXiv, 2021). `2112.02926`.

[10] Comunità, M., Steinmetz, C. J., Phan, H. & Reiss, J. D. Modelling black-box audio effects with time-varying feature modulation (2022). `2211.00497`.

[11] Martinez Ramirez, M. A., Benetos, E. & Reiss, J. D. Modeling Plate and Spring Reverberation Using A DSP-Informed Deep Neural Network. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 241–245 (IEEE, Barcelona, Spain, 2020).

[12] Pedroza, H., Meza, G. & Roman, I. R. EGFxSet: Electric guitar tones processed through real effects of distortion, modulation, delay and reverb (2022).

[13] Wright, A., Damskägg, E.-P. & Välimäki, V. Real-Time Black-Box Modelling With Recurrent Neural Networks. In *DAFx-19* (2019).

[14] Steinmetz, C. J. & Reiss, J. D. Efficient neural networks for real-time modeling of analog dynamic range compression (2022). `2102.06200`.

[15] Pakarinen, J. & Yeh, D. T. A Review of Digital Techniques for Modeling Vacuum-Tube Guitar Amplifiers. *Computer Music Journal* **33**, 85–100 (2009).

[16] Fettweis, A. Wave digital filters: Theory and practice. *Proceedings of the IEEE* **74**, 270–327 (1986).

[17] Esqueda, F., Kuznetsov, B. & Parker, J. D. Differentiable White-Box Virtual Analog Modeling. In *2021 24th International Conference on Digital Audio Effects (DAFx)*, 41–48 (IEEE, Vienna, Austria, 2021).

[18] Parker, J. D., Esqueda, F. & Bergner, A. Modelling of nonlinear state-space systems using a deep neural network. In *Proceedings of the 22nd International Conference on Digital Audio Effects (DAFx-19)* (2019).

[19] Martınez Ramírez, M. A., Benetos, E. & Reiss, J. D. A general-purpose deep learning approach to model time-varying audio effects (2019). `1905.06148`.

[20] Stein, M. Automatic detection of multiple, cascaded audio effects in guitar recordings. In *Proceedings of the 13th International Conference on Digital Audio Effects (DAFx-10)* (Graz, 2010).

[21] Abeßer, J. IDMT-SMT-Bass Dataset (2023).

[22] Kehling, C., Männchen, A. & Eppler, A. IDMT-SMT-Guitar Dataset (2023).

[23] Hawley, S. H., Colburn, B. & Mimilakis, S. I. SignalTrain: Profiling Audio Compressors with Deep Neural Networks (2019). `1905.11928`.

[24] Martínez Ramírez, M. A., Benetos, E. & Reiss, J. D. Deep Learning for Black-Box Modeling of Audio Effects. *Applied Sciences* **10**, 638 (2020).

[25] Martinez Ramirez, M. A., Benetos, E. & Reiss, J. D. Modeling plate and spring reverberation using a DSP-informed deep neural network (2019).

[26] Briot, J.-P. & Pachet, F. Deep learning for music generation: Challenges and directions. *Neural Computing and Applications* **32**, 981–993 (2020).

[27] Covert, J. & Livingston, D. L. A vacuum-tube guitar amplifier model using a recurrent neural network. In *2013 Proceedings of IEEE Southeastcon*, 1–5 (2013).

[28] Zhang, Z., Olbrych, E., Bruchalski, J., McCormick, T. J. & Livingston, D. L. A Vacuum-Tube Guitar Amplifier Model Using Long/Short-Term Memory Networks. In *SoutheastCon 2018*, 1–5 (2018).

[29] Damskägg, E.-P., Juvela, L. & Välimäki, V. Real-Time Modeling of Audio Distortion Circuits with Deep Learning (2019).

[30] Damskägg, E.-P., Juvela, L., Thuillier, E. & Välimäki, V. Deep Learning for Tube Amplifier Emulation (2019). `1811.00334`.

[31] Martinez Ramirez, M. A. & Reiss, J. D. Modeling of nonlinear audio effects with end-to-end deep neural networks (2019). `1810.06603`.

[32] Martínez Ramírez, M. A., Wang, O., Smaragdis, P. & Bryan, N. J. Differentiable Signal Processing With Black-Box Audio Effects (2021). `2105.04752`.

[33] Wright, A. & Välimäki, V. Neural Modeling of Phaser and Flanging Effects. *Journal of the Audio Engineering Society* **69**, 517–529 (2021).

[34] Simionato, R. & Fasciani, S. Deep Learning Conditioned Modeling of Optical Compression. In *Proceedings of the 25th International Conference on Digital Audio Effects (DAFx20in22)* (2022).

[35] Hochreiter, S. & Schmidhuber, J. LSTM can solve hard long time lag problems. In *Proceedings of the 9th International Conference on Neural Information Processing Systems*, NIPS'96, 473–479 (MIT Press, Cambridge, MA, USA, 1996).

[36] Hochreiter, S. & Schmidhuber, J. Long Short-term Memory. *Neural computation* **9**, 1735–80 (1997).

[37] Schmitz, T. & Embrechts, J.-J. Real Time Emulation of Parametric Guitar Tube Amplifier with Long Short Term Memory Neural Network. In *Computer Science & Information Technology*, 149–157 (Academy & Industry Research Collaboration Center (AIRCC), 2018).

[38] Paszke, A. *et al.* PyTorch: An Imperative Style, High-Performance Deep Learning Library (2019). `1912.01703`.

[39] Lea, C., Flynn, M. D., Vidal, R., Reiter, A. & Hager, G. D. Temporal Convolutional Networks for Action Segmentation and Detection (2016). `1611.05267`.

[40] Wright, A. & Valimaki, V. Perceptual loss function for neural modeling of audio systems. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 251–255 (IEEE, Barcelona, Spain, 2020).

[41] Steinmetz, C. J., Pons, J., Pascual, S. & Serrà, J. Automatic multitrack mixing with a differentiable mixing console of neural audio effects (2020). `2010.10291`.

[42] Steinmetz, C. J. & Reiss, J. D. Randomized Overdrive Neural Networks (2021). `2010.04237`.

[43] Birnbaum, S., Kuleshov, V., Enam, Z., Koh, P. W. W. & Ermon, S. Temporal FiLM: Capturing Long-Range Sequence Dependencies with Feature-Wise Modulations. In *Advances in Neural Information Processing Systems*, vol. 32 (Curran Associates, Inc., 2019).

[44] He, K., Zhang, X., Ren, S. & Sun, J. Deep Residual Learning for Image Recognition (2015). `1512.03385`.

[45] Shannon, C. E. A Mathematical Theory of Communication. *The Bell Systems Technical Journal* **27**, 55 (1948).

[46] Smith, J. O. *Mathematics Of The Discrete Fourier Transform* (W3K Publishing, 2007), second edition edn.

[47] Bogdanov, D. *et al.* Essentia: An audio analysis library for music information retrieval. In *International Society for Music Information Retrieval Conference (ISMIR'13)* (International Society for Music Information Retrieval (ISMIR), 2013).

[48] Abadi, M. *et al.* TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems (2016). `1603.04467`.

[49] Sutskever, I., Martens, J., Dahl, G. & Hinton, G. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning* (2013).

[50] Farina, A. Simultaneous measurement of impulse response and distortion with a swept-sine technique (2000).

[51] Schroeder, M. R. New Method of Measuring Reverberation Time. *The Journal of the Acoustical Society of America* **37**, 1187–1188 (1965).

[52] Imort, J. *et al.* Distortion Audio Effects: Learning How to Recover the Clean Signal (2022). `2202.01664`.