Deep Learning for the Synthesis of Sound Effects

Adrián Barahona-Ríos

PhD

University of York
Department of Computer Science
August 2023

ABSTRACT

In media production, the sound design process often involves the use of pre-recorded sound samples as the source of the audio assets. However, the increasing size and complexity of interactive media such as video games, may render this process very time-consuming and memory-demanding.

In contrast, the use of sound synthesis for sound effects can improve the sound palette of media, tackling the challenges derived from current workflows. These synthesised sound effects are usually generated using digital signal processing (DSP) methods. Nonetheless, creating sound effects using DSP methods may be challenging, and can produce unsatisfactory results, which hampers their adoption among audio professionals.

Recent data-driven approaches propose an alternative to these DSP methods for the synthesis of audio, surpassing them and establishing the state of the art in sound generation. This thesis explores the suitability of DSP systems, generative deep learning architectures, and a combination of both for the synthesis of sound effects, with an especial focus on game audio.

The results show that some DSP methods, with constraints, can be perceptually effective for this task. Furthermore, it is shown: how generative deep learning methods, not necessarily bound by those constraints, are not far from achieving a plausibility comparable to pre-recorded samples; how they can also be trained in data-scarce scenarios outperforming DSP approaches in plausibility and variation of the synthesised sounds; and how a combination of deep learning and DSP processes can be used to build expressive models, linking human-interpretable controls to the output audio.

The implications of the proposed work suggest that both generative deep learning methods and a combination of them alongside DSP approaches contribute to addressing the challenges hampering the adoption of synthesised sound effects. This work could lead to the establishment of novel data-driven workflows tailored to the preferences of audio professionals, in line with current industry demands.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

ADSR Attack Decay Sustain Release

AE Autoencoder

AI Artificial Intelligence

ANN Artificial Neural Network

BFA Bayes Factor Analysis

CNN Convolutional Neural Network

DAW Digital Audio Workstation

DDSP Differentiable Digital Signal Processing

DL Deep Learning

DSP Digital Signal Processing

FAD Fréchet Audio Distance

FFT Fast Fourier Transform

FIR Finite Impulse Response

Fs Sampling Rate

GAN Generative Adversarial Networks

GRU Gated Recurrent Unit

IFFT Inverse Fast-Fourier Transform

IIR Infinite Impulse Response

ISTFT Inverse Short-Time Fourier Transform

LSTM Long-Short Term Memory

MLOps Machine Learning Operations

MRSTFT Multi-Resolution STFT

NAS Neural Audio Synthesis

ReLU Rectified Linear Unit

RNN Recurrent Neural Network

RS Real Or Synthetic

SMS Spectral Modelling Synthesis

STFT Short-Time Fourier Transform

Tanh Hyperbolic Tangent

VAE Variational Autoencoder

VR Virtual Reality

WAV Waveform Audio File Format

ACKNOWLEDGEMENTS

This thesis was supported by the EPSRC Centre for Doctoral Training in Intelligent Games & Game Intelligence (IGGI) [EP/L015846/1] and Sony Interactive Entertainment Europe. Part of this thesis work was undertaken on the Viking Cluster, which is a high performance computing facility provided by the University of York. I am grateful for computational support from the University of York High Performance Computing service, Viking and the Research Computing team.

I would like to especially thank Sandra Pauletto, Jez Wells and Tom Collins for their guidance and support, and the current and past thesis advisory panel members James Walker, Rebecca Fiebrink, Danjeli Schembri and Oliver Hume for their feedback during the course of this work.

Also to my friends, colleagues and staff at the IGGI programme who made every step of this journey more enjoyable, and to the University of York Music Computing and Psychology Lab, the KTH Division of Media Technology and Interaction Design and SIEE, especially the London Audio Team, for being excellent peers.

Finally, to my family for their constant support throughout the years, and to Laura, who has been a source of light and joy during all these adventures, academic or otherwise.

Adrián Barahona-Ríos,
York,
2023.

UKRI Engineering and Physical Sciences Research Council

Sony Interactive Entertainment

Music Computing and Psychology Lab

DECLARATION

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References. Publications that contribute to this thesis are listed below.

Chapter 4 is based on the following publications:
Conference paper:

> Barahona, Adrián, and Sandra Pauletto. "Perceptual Evaluation of Modal Synthesis for Impact-Based Sounds," *16th Sound and Music Computing Conference*, Málaga, Spain. 2019.

Demo:

> Barahona-Ríos, Adrián. "Procedural Audio Models for Video Games with Chunity," *Audio Developer Conference (ADC)*, London, UK. 2019.

Chapter 5 is based on the following publication:
Conference paper:

> Barahona-Ríos, Adrián, and Sandra Pauletto. "Synthesising Knocking Sound Effects Using Conditional WaveGAN," *17th Sound and Music Computing Conference*, Torino, Italy. 2020.

Chapter 6 is based on the following publication:
Conference paper:

> Barahona-Ríos, Adrián, and Tom Collins. "SpecSinGAN: Sound Effect Variation Synthesis Using Single-Image GANs," *19th Sound and Music Computing Conference*, Saint-Étienne, France. 2022.

Chapter 7 is based on the following publication:
Journal paper:

> Barahona-Ríos, Adrián, and Tom Collins. "NoiseBandNet: Controllable Time-Varying Neural Audio Synthesis of Sound Effects Using Filterbanks," *IEEE/ACM Transactions on Audio, Speech, and Language Processing 32: 1573-1585.* 2024.

Other related work undertaken during this thesis is listed below.

Conference papers (co-author):

> Houel, Malcolm, Abhilash Arun, Alfred Berg, Alessandro Iop, Adrián Barahona-Ríos and Sandra Pauletto. "Perception of Emotions in Knocking Sounds: An Evaluation Study," *17th Sound and Music Computing Conference*, Torino, Italy. 2020.

> Sandra Pauletto, Adrián Barahona-Ríos, Vincenzo Madaghiele and Yann Seznec. "Sonifying Energy Consumption Using SpecSinGAN," *20th Sound and Music Computing Conference*, Stockholm, Sweden. 2023.

Patents:

> Barahona-Ríos, Adrián. "Audio Generation Methods and Systems," *European Patent number EP4120259A1.* 2023.

> Barahona-Ríos, Adrián. "Audio Generation Methods and Systems," *European Patent number EP4120262A1.* 2023.

> Barahona-Ríos, Adrián. "Audio Generation Methods and Systems," *European Patent number EP4120258A1.* 2023.

CHAPTER 1

INTRODUCTION

Sound effects can be generally defined as those sound elements other than music or speech [1], and their origin can be traced back to the early days of theatre in ancient Greece, where they were performed live. Between the late 1920s and early 1930s, recorded sound effects proliferated as a solution to the increasing technical sophistication required by radio dramas, which made live performance of sound effects impractical [2]. Since then, radio, film, and television have contributed to developing the medium to its current standard. Video game audio, however, had a parallel development due to its younger nature and its particular technical challenges. It was 1972 when "Pong", the first commercial video game with sound, was released [3]. "Pong", which is a top-down-view table tennis simulator, produced its very simple – by current standards – sound effects by synthesising a square-wave at different frequencies every time the ball touched either the court walls, the player's paddles, or in the event of a score. Advances in technology, such as frequency modulation (FM) synthesis and especially sampling, which allowed for the use of pre-recorded audio in games, increased the complexity of the video game audio palette, eventually elevating it to the same level as the film industry [4].

1.1 Motivation

Currently, the sound effects used in game audio or film are typically pre-recorded and created by either Foley artists or sound designers. To source these sounds effects, designers either: record the sounds on demand (e.g., in order to sound design footsteps in grass, they may record the source material directly); take them from pre-recorded sound libraries (e.g., a pre-existing sound library may contain those footsteps in grass); or, more commonly, take a mixed approach where sounds are recorded directly *and* obtained from libraries, then layered and processed together to create the final sound effect. In the context of game audio, once these sound effects are created, they need to be *implemented*, which consists of creating the logic, audio hooks and behaviours to determine when and how a sound effect is triggered during gameplay [5]. This implementation is usually done by employing middlewares such as Audiokinetic Wwise[1] or FMOD,[2] which link the audio logic with the game engine and gameplay mechanics. This is a consequence of video games being typically nonlinear where, unlike film or television, actions are not pre-determined, and players may potentially interact with the virtual environment freely at any moment.

Thus, since games are interactive and actions may be repeated an arbitrary number of times, it is common to use multiple sound files to sound

[1] `https://www.audiokinetic.com/en/products/wwise/`

[2] `https://www.fmod.com/`

design different audio variations of a single in-game interaction. Usually, this is done to prevent listener fatigue and avoid repetition [5], as well as to mimic reality (increase verisimilitude), where two sounds are hardly identical. To provide further variation to the sonic interactions, regularly, the sound design process involves also the use of different sound layers for a single sound effect [5]. For instance, a footstep sound effect can be broken down into the tip, the heel, and the shoe fabric layers. Considering the size of some of the current games [6], the number of audio interactions may make the current workflow involved in producing the game audio very time-consuming.

Apart from repetition – and its associated challenges – pre-recorded samples have several limitations in highly interactive scenarios such as in virtual reality (VR), where haptic controllers are commonly used. Unlike most games, where the interactions with the environment are discrete (e.g., pushing a button, clicking a mouse), VR interactions may be continuous (e.g., handling an object, shaking it, scrapping it). Some of these interactions also occur to a degree in non-VR situations (e.g., a car accelerator pedal linked to a game controller shoulder button), but in general virtual reality presents its own challenges and demands VR-specific solutions [7].

To illustrate the above point, let us consider what it is, in principle, a simple experience in VR, discussed in [8]: the sound design of a ping-pong ball. Because users are allowed to have $1:1$ interactions with the environment using haptic controllers, sound designers need to create all possible sounds and interactions of handling and releasing the ball itself. Moreover, as the virtual experience happens in a room with a set of different interactable objects, they also need to create the sound of a ping-pong ball being struck by other objects, which increases the complexity of the sound design task further. Considering that the previous example is somewhat simple, more complex scenarios such as bigger open-world games may increase the complexity and audio storage burdens considerably when using current workflows.

An alternative to using pre-recorded sound samples is the use of sound synthesis to generate the sound assets. This is often called generative or procedural audio [9]. Procedural audio usually refers to the use of real-time digital signal processing (DSP) systems such as sound synthesisers, while generative (audio) can be defined as "algorithms to produce an output that is not explicitly defined" [10, p.1]. Apart from sound synthesisers, other DSP methods involve the manipulation of audio files in order to obtain a desired effect, transforming the source asset [11, 12]. Thus, generative and procedural audio allow the dynamic creation of sound assets on demand. The dynamism of generative and procedural audio systems make them well suited to use in video games. "Grand Theft Audio V" by Rockstar Games, which became the most profitable entertainment product of all time in 2018 [13], uses procedural audio extensively [14]. Another example is "No Man's Sky" by Hello Games, which uses sound synthesis for the creature vocals and background fauna [15].

However, as outlined before, nowadays most video games are sound designed by using pre-recorded samples – as happens in the film industry or, in general, entertainment media. So if the use of procedural audio has multiple benefits over the use of pre-recorded samples, why is it not yet broadly adopted? Farnell [9] identifies a series of challenges that hampers the proliferation and adoption of procedural audio. Among others, lack of: plausibility (i.e., the synthesised sounds suffer from low perceived quality when compared to pre-recorded samples); appropriate tools (i.e., there is a scarcity of procedural audio software tools catered to the needs of audio professionals); procedural audio-specific skills and industry inertia (i.e., related – and arguably a consequence of the previous challenges, audio professionals may not consider procedural audio as a viable option) [9].

In other words, creating or using procedural or generative audio models may prove challenging for sound designers, and the results may be perceived as unsatisfactory. While some of these problems are currently being addressed (see Section 2.3 for mention of new tools), synthesis of sound effects remains an open research problem [9, 16, 17, 18, 8].

However, recent *data-driven* techniques, such as those presented in [19], have attained the state of the art in sound synthesis, surpassing “classic” DSP methods and offering alternative approaches for sound generation. Consequently, and with a specific focus on game audio, this thesis will investigate the applicability of DSP techniques and novel deep learning methods for sound effects synthesis, addressing the research questions outlined in Section 1.3.

1.2 Scope

The work carried out throughout this thesis is focused on the evaluation and development of algorithms for the synthesis of sound effects, and it is particularly centered around interactive environments such as video games. The main aims of this thesis are the evaluation of classic digital signal processing algorithms devoted to this task, and the exploration, development and evaluation of novel generative deep learning architectures capable of producing audio both unconditionally and with intuitive control schemes derived from game audio workflows.

Specifically, the work carried out during this thesis primarily engages with filter-based modal synthesis, generative adversarial networks, and a combination of digital signal processing and deep learning for the synthesis of sound effects, having an emphasis on the usage of limited training data. The algorithms and approaches proposed in this work are evaluated either subjectively by performing a listening study, objectively, or both.

1.3 Research Questions

1. To what extent can DSP-based procedural audio techniques be applied to video games in terms of perceived plausibility?
2. How can novel deep learning techniques be applied directly to the synthesis of sound effects?
3. To what extent can a combination of DSP-based techniques and machine learning methods be applied to the synthesis of sound effects?

1.4 Contributions

The main contributions of this thesis are listed below.

- A subjective evaluation of the plausibility of filter-based modal synthesis for the generation of impact-based sound effects, plus an open-source demo comprised of the algorithm evaluated, alongside other interactable real-time DSP procedural audio models implemented in a virtual environment.

- An exploration of the suitability of generative adversarial networks for the class-conditional synthesis of sound effects, including its open-source implementation and the dataset created for this study.
- The development and evaluation of a generative adversarial network architecture capable of synthesising variations of one-shot sounds effects by training on a single audio example.
- The development and evaluation of an architecture combining deep learning and DSP capable of synthesising controllable time-varying sound effects training on limited data, including its open-source implementation.

1.5 Statement of Ethics

This thesis carries out experiments (i.e., listening studies) that involve the participation of human subjects. The participants were informed of the nature of the studies, and the studies were approved by the University of York Physical Sciences Ethics Committee board, in compliance with the University's Code of practice and principles for good ethical governance.[3] The consent forms and the demographic questions that participants were asked in the listening studies can be found in the Questionnaires appendix.

[3] https://www.york.ac.uk/staff/research/governance/research-policies/ethics-code/

It is also acknowledged that the field of knowledge engaged by this thesis could further automate human creative processes, potentially negatively impacting those who work in such areas. However, the primary goal of the algorithms explored in this thesis is to assist, not replace, individuals in these roles.

1.6 Thesis Structure

The thesis is organised as follows.

Chapter 1 is the introduction to this thesis, outlining the research motivations and the gap in knowledge that this work seeks to fill. Chapter 2 contains an introduction to digital signal processing methods employed in the synthesis of audio, and their applications in the context of sound effects. Chapter 3 contains an introduction to the field of deep learning and generative deep learning, including an overview of neural audio synthesis and differentiable digital signal processing, again with an emphasis on sound effects.

Chapter 4 evaluates filter-based modal synthesis against pre-recorded samples for the synthesis of percussive sound effects. It also showcases a real-time demo implementation of this and other DSP techniques within a video game engine. This chapter addresses the first research question (RQ 1). Chapter 5 assesses the use of generative adversarial networks for class-conditional[4] sound effects synthesis. This chapter partially addresses the second research question (RQ 2). Chapter 6 addresses the problems derived from potential data scarcity when using deep learning methods for

[4] Training on a dataset comprised of multiple sound classes such as emotions in knocking sound effects, being able to select which class – or emotion, in this case – the synthesiser outputs.

the synthesis of sound effects. This chapter completes addressing the second research question (RQ 2). Chapter 7 investigates the use of DSP elements alongside deep learning for the synthesis of controllable sound effects. This chapter addresses the third research question (RQ 3).

Finally, Chapter 8 summarises the findings and outlines the future directions of this work.

CHAPTER 2

DSP-BASED SOUND SYNTHESIS

2.1 Fundamentals of DSP

Signals can be defined as *something* that conveys information, and they are represented mathematically as functions with one or more independent variables [20]. In audio, signals are usually functions of time, split onto continuous-time signals (i.e., *analog* signals, represented by a continuous independent variable), and discrete-time signals (i.e., signals where the independent variable is discretised); with their amplitude (i.e., their values with respect to time) being also either continuous or discrete [20]. In *digital signals*, both the time and the amplitude values are discrete [20], with digital signal *processing* (DSP) being the techniques and algorithms devoted to manipulate those signals [21]. This section will glance over important DSP concepts related to this work.

In Chapter 1, it is described how the source of most sound effects in media are pre-recorded samples. Commonly, the starting point of those sound effects are real-world sounds, which are recorded using a transducer (i.e., a microphone) and stored as digital audio files. The process of transforming a continuous signal (e.g., a sound happening in the real world) into a digitalised version of itself is done through sampling, where amplitude values of the signal are measured at a particular rate in time – conveniently called the *sampling rate* – and quantised to discrete values [22]. The sampling rate, Fs, has an important effect on the ability to capture the details of an analog signal. By the sampling theorem [23], it is stated that, if a signal is band-limited to a frequency half of the sampling rate (i.e., Fs/2, also known as the Nyquist frequency), this can be sampled (i.e., digitalised) without any loss of information [24]. Thus, the sampling rate determines the maximum frequency a system is able to capture. For instance, an audio signal sampled at 16 kHz will be able to contain frequencies up to 8 kHz.

Band-limiting the signal is done through *filtering*. Generally speaking, filtering consists of attenuating – and also boosting – certain frequencies, or frequency regions, by using a *filter*. Among others, popular filter *frequency responses* include highpass filters (they remove low frequencies, letting high frequencies pass above a cutoff), lowpass filters (they remove high frequencies, letting low frequencies pass below a cutoff, used for instance to band-limit a signal), or bandpass filters (they allow frequencies between a range, attenuating the frequencies outside them, or boosting the frequencies within the band edges) [21]. When multiple filters are configured in parallel, the arrangement is commonly referred to as a filterbank, which is a collection of – usually – bandpass filters that cover different, often contiguous, frequency regions of the input signal.

While analog filters exist (and in fact they were the first ones, using components such as resistors or capacitors), in the digital realm, digital filtering is done by using mathematical algorithms to process discrete-time (i.e., sampled) signals, handling delayed (i.e., time-shifted) versions of either the input signal, or both the input and output signals. Depending on this factor, digital filters can be categorised as finite impulse response (or FIR, if they process a finite delayed version of its inputs), or as infinite impulse response (or IIR, if they process both past inputs and past outputs to generate the current output) [22]. FIR and IIR filters have distinctive characteristics each: FIR filters are more suitable for applications requiring strict phase linearity (i.e., having uniform signal delay across all frequencies), while IIR filters are more efficient at achieving complex filter responses with fewer computational resources.

DSP also provides a wide range of tools to analyse signals, and one of the most relevant algorithms to this end is the Discrete Fourier Transform (DFT). At the core of the DFT is the Fourier analysis, which is the decomposition of a signal as a sum of sine and cosine functions [25]. Thus, the DFT is the application of the Fourier transform to digital signals, which when used, results in a *decomposed* signal consisting of a set of *sinusoids* [21]. This representation is called the *frequency domain*, and carries information about how much of each of the sinusoidal frequencies is present in the analysed signal, including both their magnitudes (i.e., their amplitudes) and phases (i.e., their time shifts) [21]. When computing the DFT is also common to multiply the signal by a smoothing curve (commonly referred to as a *window* function), such as the Hamming or the Hanning windows, to increase its accuracy [21]. The DFT is also invertible: it is possible to retrieve the original time-domain signal from its frequency-domain representation by computing an Inverse Discrete Fourier Transform (IDFT) [21]. The Fast Fourier Transform (FFT) [26] is an important algorithm that builds upon the DFT (more specifically, the complex DFT) to provide a more computationally efficient solution to calculate the transform.

However, the DFT (and therefore the FFT) computes a static snapshot of the spectrum. For instance, for a time-varying signal (e.g., a chirp, which is a signal whose frequency increases or decreases over time), the DFT will identify *all* the frequencies present on it, but not their evolution over time (e.g., ascending or descending frequencies in the chirp case). To overcome this limitation, the Short-Time Fourier Transform (STFT) breaks up the signal into – usually overlapping – segments and computes their FFT. This approach is characterized by two main parameters: the window length, which is the length in samples of the signal segment (this length can match the FFT size, especially when they are equal, or when the signal segment is zero-padded to this length), and the hop size, which is the number of samples by which the window is advanced at each step [22]. As with the DFT or FFT, it is possible to invert the STFT and retrieve the original time-domain signal from the time-varying spectrum by computing an Inverse Short-Time Fourier Transform (ISTFT). From a data representation point of view, it is common to discard the signal phase spectrum, since it does not carry as much information as the magnitude [21]. It is however possible to retrieve the phase from a magnitude-only STFT using DSP algorithms such as Griffin-Lim [27], which exploits redundancies in the STFT to iteratively estimate its phase – thus making it feasible to approximate a time-domain signal from only its magnitude spectrum.

2.2 DSP-Based Sound Synthesis Techniques

While Section 2.1 starts by describing how most sound effects are built from pre-recorded audio files, it is also possible to generate synthetic signals either directly or by analysing and processing such pre-recorded samples. Digital sound synthesis is defined as numerical algorithms that produce audio, often in real-time [28]. The field of digital sound synthesis started in the 1950s [29], usually involving methods that combined different *basic* elements together, such as oscillators (i.e., generators that produce periodic waveforms [9]) or filters. Using substantially more computational power today, combined with advancements in software tools and a body of creative and engineering work, sound synthesis now addresses a wide range of practical uses, such as speech synthesis, musical instrument simulation, sound effect synthesis or data sonification. This section will summarise some of these advancements and their applications to the synthesis of sound effects.

Sound synthesis techniques have historically been categorised into various groups, often within a musical context rather than sound effects. In 1991, Julius Smith developed a taxonomy dividing techniques into four categories: processed recordings, spectral models, physical models, and abstract algorithms [30]. The processed recordings category refers to sample-based techniques, where the sound source of the synthesiser is a pre-recorded sample. Spectral modelling techniques parameterise the sounds from a perceptual standpoint [31]. Physical modelling techniques aim to replicate the physical behaviour of a system by describing it through mathematical models. Abstract techniques refer to the use of fundamental DSP components – such as generators or filters – to shape a desired sound, often without considering the physical behaviour of the sound source. Smith also predicted that sound synthesis will be dominated by spectral and physical models in the future, with abstract techniques potentially fading away, and with sample-based techniques absorbed into spectral modelling techniques.

On the other hand, Stefan Bilbao divides sound synthesis techniques into two main categories: abstract techniques and physical models [29]. To Bilbao, abstract techniques lack realism ("plausibility") and control schemes compared to physical models, and overall the abstract category includes all techniques that are not based on a physical principle but on a perceptual one.

Alternatively, Moorer [32] splits sound synthesis approaches into three categories, basing them on how the computer is used to produce the sounds: direct synthesis, analysis-based synthesis and musique-concrète. Direct synthesis is similar to Smith's [30] "abstract synthesis", where the sounds are produced by directly combining different computational modules to achieve a desired sound. Analysis-based synthesis is rooted in the analysis of a pre-existing digitalised sound, somewhat akin to Smith's [30] "spectral models". Musique-concrète is also similar to Smith's [30] "sample-based" category, where a pre-recorded sound is processed by the computer in diverse ways to achieve a desired sound. Additionally, regardless of the synthesis approach, Moorer [32] classifies the processes used in sound creation as additive (sum of components), subtractive (removing spectral content) or modulation (anything not covered by the previous two). Finally, Moorer [32] categorises synthesis systems based on their turnaround time as offline (the computation is performed once, and the results are "baked"), interactive (the computation is carried out over a short segment of audio, with the results displayed to collect user feedback for refining the output), and online (the output and control parameters are listened and modified in real-time, respectively).

There are more general sound synthesis surveys and taxonomies, such as [33, 34, 35, 36]; and specific to sub-set of techniques, such as [37] for physics-based synthesis. However, regardless of the taxonomy used, there can often be a fine line between the different categories, and this line can often be subjective. For this thesis, with the aim of providing a granular context regarding the most relevant techniques for synthesizing sound effects, the DSP-based synthesis methods will be divided into five different categories: the four categories proposed by Smith [30] (abstract, sample-based, spectral and physical models), plus statistical techniques. The motivation to include statistical approaches is to bridge the gap between "classic" DSP methods and the deep learning domain (introduced in Chapter 3).

2.2.1 Abstract Techniques

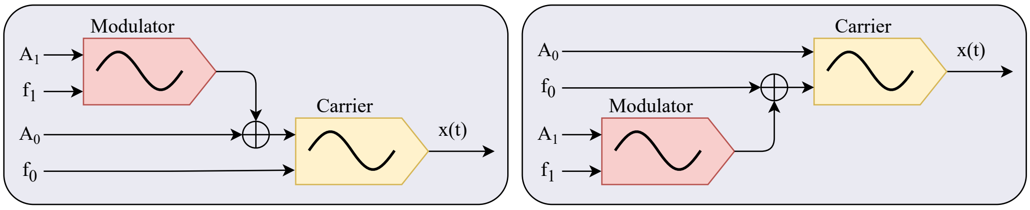
Abstract techniques were the first synthesis methods explored, dating back to the 1950's [29]. They are based on the combination of different basic components, such as oscillators or filters.

2.2.1.1 Modulation Techniques

Modulation techniques are based on the modification of part of a target signal, called carrier, by another signal, called modulator [38]. These signals, often oscillators, can be of different shapes (e.g., sinusoidal, square or sawtooth). It is possible to use more than two oscillators by adding modulators (e.g., a carrier modulated by two signals).

Amplitude modulation (AM) synthesis, depicted in Figure 2.1a, is based on the modulation of the amplitude of the carrier by the modulator. The modulator amplitude controls the amount of the modulation on the carrier amplitude, while the modulator frequency affects the rate of the carrier amplitude modulation. A variant of this technique is ring modulation, where the amplitude of the carrier is determined by the modulator alone [33].

Frequency modulation (FM) synthesis, depicted in Figure 2.1b, is based on the modulation of the frequency of the carrier by the modulator [39]. Again, the modulator amplitude (called *modulation index* in FM synthesis) controls the amount of modulation affecting the carrier. The modulator frequency controls the carrier frequency modulation.



(a) AM synthesis diagram

(b) FM synthesis diagram

Figure 2.1: Diagram of an AM (a) and a FM (b) circuit. A_0 and f_0 are the carrier amplitude and frequency. A_1 and f $_1$ are the modulator amplitude and modulator frequency.

While FM synthesis can produce a rich time-varying spectra, creating complex time-varying sounds can be challenging. Since FM synthesis behaves mostly like a black box with its parameters acting in a non-linear way, they may not be easily linked to the sound produced from a perceptual point of view. Thus, Chowning compiled in the 1970s a series of parameter configurations to obtain sounds such as brass-like tones, woodwind-like tones, or percussive sounds [39].

2.2.1.2 Piecewise or Brute Force

Given that abstract techniques are based on combining basic components, a very direct approach is to craft a sound by combining these components until the result is perceptually satisfactory. This approach is called “brute force” or “piecewise” [9].

While effective for certain sounds, brute force techniques are limited as their effectiveness will depend directly on the skill of the sound designer, the complexity of the sound to model, and the time available to craft the sound. Thus, brute force techniques may not be “time-effective” for complex sounds or interactions.

2.2.1.3 Digital Waveshaping

Digital Waveshaping synthesis consists of introducing distortion to a signal $x(t)$ by computing some function $f(t)$, called the “shaping function” [40]. If it is assumed that $x(t)$ is for instance a sinusoidal, processing the signal $x(t)$ with the shaping function $f(t)$ alters its timbre $x' = f\left[x(t)\right]$, producing rich spectra from a single pure tone.

2.2.2 Sample-Based Techniques

Sampling or sample-based techniques are those which use a – generally short – pre-recorded sample as the sound source [36].

2.2.2.1 Wavetable Synthesis

A wavetable is defined as a block of memory (i.e., a "table") where a discretised signal is stored [38]. Wavetable synthesis consists of reading and looping through the values of the table, which is computationally inexpensive. The stored signal may have any length, although it is generally short, and usually of one period of a waveform.

2.2.2.2 Concatenative Synthesis

Concatenative synthesis consists of dividing a dataset of source sounds into smaller segments called units, usually of length between 10ms to 1s. The resulting database of units is also called corpus. The units are selected (and played back) employing an unit selection algorithm which, by comparing their acoustic features or descriptors, chooses which units are more suitable to synthesise a target sound [41, 42]. The selected units are then *concatenated* to produce the target sound. In other words, concatenative synthesis aims to select the best sounds ("units") from a database ("corpus") in order to reproduce a signal ("target"), based on features or descriptors.

Concatenative synthesis can be seen, in a way, as an hybrid between sample-based techniques and statistical approaches (see Section 2.2.5). While the sound source (i.e., the unit) is a pre-recorded sample, the unit selection algorithm selects the units using statistical methods, such as the the Euclidean distance in the multi-dimensional feature-descriptor space [43].

2.2.2.3 Granular Synthesis

Granular synthesis is based on the concept of slicing a pre-recorded sound into small sonic events (called "grains"), and playing them back [44]. The length of the grains can vary from 200μs to 200ms, although it is typically in the range of $1-50$ms. When played back, the frequency of grain occurrence (i.e., the number of grains played over time) is referred to as grain density. Each of these grains has a distinct amplitude envelope, frequency content, and even a different length. The length of the grains lead to different effects on the synthesised sound, which Roads documented in [45], varying from loss of pitch from the original sound ($\approx$ 1ms grains), to stable pitch ($\approx$ 50ms grains), or periodic tremolo ($\approx$ 200ms grains). Grain length can be also constant, time-variant, randomised, or parameter-dependant.

2.2.3 Spectral Models

Spectral models are based on the perceptual modelling of sounds as they are perceived by the listener [28]. These techniques usually fall into the "analysis-synthesis" paradigm, where sounds are computationally analysed and reconstructed (i.e., synthesised) by means of DSP components. There are numerous techniques apart from the ones that will be described in this section, such as the phase vocoder [46] or the inverse fast-Fourier transform (IFFT) synthesis [47]; as well as techniques derived from them, like the sines plus transients plus noise method [48], which adds transient modelling to a sinusoidal plus noise model. However, the focus will be on the main building blocks that are most relevant to the current work.

2.2.3.1 Additive Synthesis

Additive synthesis consists of summing individual sinusoidal partials with different amplitudes and frequencies to create sounds [38]. It is rooted in the Fourier theorem, which states that any periodic function $f(x)$ can be expressed as a sum of sinusoids with different phases and amplitudes. For instance, a signal $x(t)$ comprised of N sinusoidal partials with ϕ_N initial phases and f_N and A_N frequencies and amplitudes respectively, can be defined as [29]:

$$x(t) = \sum_{n=1}^{N} A_n cos(2\pi f_n t + \phi_n) \tag{2.1}$$

Alternatively, a signal $x(t)$ comprised of N sinusoidal partials with $A_N(t)$ and $\phi_N(t)$ time-varying instantaneous amplitudes and phases respectively, can be created as [49]:

$$x(t) = \sum_{n=1}^{N} A_n(t) cos\left[\phi_n(t)\right] \tag{2.2}$$

with the formula to calculate the instantaneous frequency $f_i(t)$ from the instantaneous unwrapped phase $\phi(t)$ of a sinusoidal partial given by [50]:

$$f_i(t) = \frac{1}{2\pi}\frac{d\phi(t)}{dt} \tag{2.3}$$

A sound is described as harmonic when the partial frequencies are integer multiples of a fundamental frequency F_0, and the relationships between the amplitudes of these different harmonic partials contribute to the perceived timbre of the sound [38]. However, unlike musical sounds, most sound effects may not contain a harmonic spectrum.

In an analysis-synthesis framework, there are multiple “pure additive” approaches for extracting the time-varying sinusoidal partials of a sound. For instance, in [51] they estimate the sinusoidal partial parameters from the short-time Fourier transform (STFT) using a peak-picking algorithm; or in [52, 49] they use the complex continuous wavelet transform (CCWT) to perform a similar task.

2.2.3.2 Subtractive Synthesis

In subtractive synthesis, also known as source-filter modelling, particularly in the physical modelling literature [29], the sound is modeled by passing an excitation function through a filter [32]. For instance, a white noise signal, which has a spectral flatness close to 1 (i.e., all frequencies are equally represented in the spectrum), can be processed with a low-pass filter to remove its high frequencies from a certain cutoff frequency F_{cutoff}. Note that, despite the technique commonly being referred to as subtractive, the filters may also amplify frequencies from the source signal [37]. Hence, the term “source-filter” modelling is preferred when this amplification is prone to occur, especially in physical models.

2.2.3.3 Spectral Modelling Synthesis

Spectral modelling synthesis (SMS) combines additive and subtractive synthesis to generate sounds. SMS divides sounds into deterministic (modeled with sinusoidal partials) and stochastic (modeled with filtered white noise) components [31]. Thus, a signal $x(t)$ comprised of N sinusoidal partials with $A_N(t)$ and $\phi_N(t)$ time-varying instantaneous amplitudes and phases respectively can be represented as:

$$x(t) = \sum_{n=1}^{N} A_n(t) cos \left[\phi_n(t)\right] + e(t) \tag{2.4}$$

where $e(t)$ is the noise component, defined as [31]:

$$e(t) = \int_0^t h(t,\tau) u(\tau) d\tau \tag{2.5}$$

with $u(\tau)$ being a white noise signal and $h(t,\tau)$ a time-varying filter.

SMS has the advantage of being able to produce noise-like signals (such as the attack of instruments) that otherwise would be expensive to model using pure additive approaches. This is because, in theory, in order to produce a noise signal using sinusoids there would have to be a single sinusoid at every frequency of the spectrum [31].

2.2.4 Physical Modelling

Physically-Inspired, Physically-Derived, Physical Models, and Physics-Based Synthesis aim to either incorporate physical elements of the sound source to the model (e.g, a synthesiser) or to fully simulate the sound of a physical phenomenon using numerical methods. All these concepts can be broadly grouped into the physical modelling umbrella.

There are multiple pieces of work that explore the various techniques employed in this category. For instance, Bilbao [29] categorises them into lumped mass-spring networks, modal synthesis, digital waveguides, hybrid methods, and direct numerical simulation. In a somewhat similar manner, Välimäki et al. [37] classify discrete-time physical models into six main paradigms: finite difference models, mass-spring networks, modal decomposition, digital waveguides, wave digital filters and source-filter models. The categories in the latter will be adopted since they provide a comprehensive overview of this field.

2.2.4.1 Finite Difference Models

Finite difference models (grouped under direct numerical simulation in [29]) are based on solving partial differential equations (PDEs) using a finite difference approximation. They are also referred to as "finite difference time domain" (FDTD) methods in time-dependent systems [29]. They consist of discretising a continuous system described by PDEs (such as those of a vibrating string, or the membrane of a drum) onto a grid, where the numerical solution is calculated [29, 37].

2.2.4.2 Mass-Spring Networks

In mass-spring networks (categorised as lumped mass-spring networks in [29]), a physical system is modelled as a set of mass elements connected by springs and dampers [37]. The simplest example would be a mass suspended by a spring, which has a solution of [22]:

$$y(t) = y_0 e^{(-rt/2m)} cos\left(t\sqrt{k/m - (r - 2m^2)}\right) \tag{2.6}$$

with y_0 being the initial position, m the mass, k the spring constant and r the damping, as depicted in Figure 2.2. Building upon this idea, if a series of lumped mass-spring systems are interconnected to form a lumped mass-spring network, it is possible to model a string by placing them in a linear configuration, or a drum membrane by arranging them in a grid configuration [29].

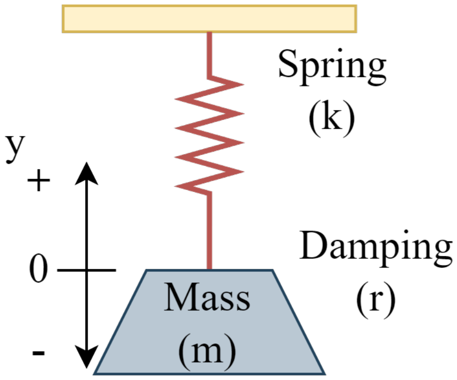


Figure 2.2: Mass-spring system diagram (reproduced from [22]).

2.2.4.3 Modal Synthesis

Modal synthesis is rooted in the idea of simulating the vibration of linear systems, decomposing them as a series of modes (also called the modal representation) that are derived from the physical (e.g., mechanical, geometrical) characteristics of the object to be modelled [53, 54]. These modes are represented by frequencies and amplitudes.

There are multiple forms of obtaining the modal data of a system. For instance, a method to extract these modes from an object is by performing a system eigendecomposition to simulate the behaviour of its surface at different points as it is deformed by being struck [55]. Another option is to extract these modes from real recordings with the objective of modelling the physical object. For instance, this could be accomplished by analysing various recordings of an object being struck using a frequency domain representation in an analysis-synthesis approach. Following this concept, Cook [56, 22] proposed the use of filter-based modal synthesis to build a "physically-informed" model, which can be used in real-time with intuitive controls (e.g., striking position, material or impact force), building it from the sound recordings themselves.

2.2.4.4 Digital Waveguides

Digital waveguides (DWG) are based on discretising and sampling the solution of the traveling wave equation across time and space [57]. For instance, for a one-dimensional system such as a string or a tube, the solution to the aforementioned equation can be implemented using a bi-directional delay line simulating two waves propagating independently in the left and right directions [57]. Two-dimensional systems such as membranes can be also modelled in a relatively similar way as in finite difference models (see Section 2.2.4.1), but easing the computational burdens [58]. Three-dimensional systems also exist, such as those applied to the simulation of room acoustics [59]. Digital waveguides have been broadly used, from simulating musical instruments, to room acoustics, or the human vocal tract [60].

2.2.4.5 Wave Digital Filters

Wave digital filters (WDFs) are a type of digital filters that are conceptually inspired by classic circuit theory [61]. WDFs are highly modular as the different circuit components and derived configurations can be seen as "building blocks" of the system. Thus, they are especially suited for lumped mass modelling approaches [37], or for finite-difference schemes when they are used in conjunction with digital waveguides [29] (see previous Section 2.2.4.4).

2.2.4.6 Source-Filter Models

As in their abstract counterpart (see Section 2.2.3.2), source-filter models consist of filtering an excitation signal through a filter to remove or accentuate certain frequencies. Välimäki et al. [37] argue that in order to consider a source-filter model part of the physical modelling category, the source signal has to convey some information of the physical system, and the filter has to be derived from the physical structure of the object being modelled.

2.2.5 Statistical Approaches

While the statistical modelling of audio signals only gained popularity in recent years due to the proliferation of novel deep learning techniques, there were attempts to model audio signals using statistical approaches before the popularity of neural networks.

Despite being considered part of the spectral models and a subtractive synthesis method by Smith [30] and Cook [22] respectively, Linear Predictive Coding (LPC) [62] could be seen as a statistical approach too. LPC, which has been extensively used in speech signal processing, involves modelling a signal as a linear combination of its past samples with the aim to predict the next one. Arguably leaning more towards the statistical category, McDermott et al. [63] approached sound texture synthesis by decomposing a target sound into sub-bands using a cochlear filterbank in order to extract a set of statistics from their sub-band amplitude envelopes. Another approaches treat signal processing problems, such as sub-band demodulation or time-frequency analysis, as inference problems to obtain alternative representations using Bayesian probability [64].

2.3 DSP-Based Synthesis of Sound Effects

Following from the previous Section 2.2, where an overview of popular sound synthesis techniques were presented (i.e., the theory), this section introduces how these techniques can be used to create sound effects (i.e., the tools and applications). There is an abundance of resources on the synthesis of sound effects, ranging from academic research and industry products, including books that approach the subject from a practical perspective [9, 22], to reviews and surveys of the field [65, 66, 67, 68, 69, 34].

Sound synthesis of sound effects is an open research problem, encompassing a wide range of approaches and methodologies from various schools of thought. This chapter compiles a representative set of these approaches. Starting with categories of sounds that are somewhat related to video games and media, in [70] they used an analysis-synthesis modal approach – similar to spectral modelling – to generate weapon sounds, resynthesising the deterministic part using sinusoidal modelling and the stochastic part using subtractive synthesis. Recently, in [71] they used subtractive synthesis to produce ocean waves, implementing the procedural audio model in the Web Audio API and outperforming, in a listening study, other methods such as spectral modelling synthesis [31] or the statistical approach described in [63]. Selfridge thoroughly studied the simulation of aeroacoustic sound effects using physically-inspired models, modelling sounds such as sword swings or propellers [72, 73, 74, 75]. Footsteps, which are ubiquitous in games and virtual reality scenarios, have been also addressed using techniques such as physical models [76, 77, 78]. Other sounds like hand clapping have been also studied, employing resonator filters and controlling high-level parameters in real-time, such as the number of people in the audience or the synchronisation of the claps [79, 80, 81, 82, 83]. In [84, 85, 86] they employed a series of synthesis techniques to propose an alternative way of thinking about sound design, calling their framework "TAPESTREA" (*Techniques and Paradigms for Expressive Synthesis, Transformation and Rendering of Environmental Audio*). They use an analysis-synthesis approach to produce an adaptive soundscape, splitting it into foreground events and background environmental cues. Other examples of environmental sound synthesis can be found in the works of [87, 88, 89], where they employ a method derived from the IFFT synthesis, addressing the sound spatialisation as well.

Regarding other work emerging from the computer animation and physic-based simulation field, in [90] they computed the propagation of the waves resulting from analysing the deformation in solid objects being struck. Alternatively, in [91] they begin with a 3D object to build a physics-based model and simulate user interactions. Another example is [92], where they aim to resolve the sound radiation resulting from computer animations, incorporating also near-field scattering and diffraction effects. However, approaches of this type are typically computationally expensive to run in real-time.

Concatenative synthesis has been used to produce sounds driven by virtual graphic animations, such as cloth [93] or paper [94]. In [95] they also used concatenative synthesis to perform controllable audio texture synthesis of sounds such as rain, modulating its intensity (e.g., light, heavy). Continuing with texture synthesis, in [96] they introduced the Cascade Time-Frequency Linear Prediction (CTFLP) algorithm for resynthesising sounds such as rain or footsteps; and in [97], they used a granular overlap-add synthesis approach to produce sound textures of shore or stream sounds.

There is also work done in the area of filterbanks, multi-rate systems and sub-band processing, such as in [98, 99] where they synthesised noisy environmental sounds such as rocks crumbling or a fireplace; or in [63] where they use the sub-band amplitude statistics to build a synthesis model. Wavelets have been also used in this context, with studies such as [100] where they use the discrete wavelet transform (DWT) and its inversion coefficients to produce stochastic-based sounds; or in [101, 102], where they adapted a texture synthesis method from the computer vision domain to the audio domain using the Dual-Tree Complex Wavelet Transform (DT-CWT), performing audio texture synthesis derived from short sound excerpts.

In terms of studies comparing different procedural audio and sound synthesis approaches, there is work exploring multiple sound categories and interaction types such as [66, 103, 104], which can serve as a guide to choose suitable models depending on the context. There is also extensive research on how to design interactions with synthesisers and procedural audio models. For instance, in [105] they studied mapping strategies for modelling a squeaking door; or in [106] they proposed a control framework based on audio features to synthesise sounds. Other approaches use latent force modelling [107, 108] to discover high-level and meaningful control parameters for synthesisers.

There are some readily available commercial procedural audio tools catered towards sound designers. Nemisindo[1] [16] is a sound design service that provides a variety of interactive procedural audio models – ranging from footsteps to impacts – in a web-based experience. They also offer procedural

[1] https://nemisindo.com/

audio models packs integrated into popular commercial game engines, including a nature sound pack for the Unreal Engine and a combustion engine sound pack for the Unity game engine. GameSynth by Tsugi[2] is another procedural audio tool that offers multiple models, all housed within a dedicated standalone program. Apart from model presets, their software offers a modular design (somewhat similar to visual audio programming languages such as Pure Data), allowing the users to not only adjust the model's parameters but also to change their signal flow and modules. The Wwise middleware by Audiokinetic also offers some plugins build upon sound synthesis, such as Impacter,[3] a tool that employs an analysis-synthesis approach to generate controllable variations of impact sounds. Yet another range of products are the ones offered by LeSound,[4] which develops digital audio workstation (DAW) plugins using procedural audio technologies for sounds such as fire or rain. Another popular product is REV by CrankcaseAudio,[5] which uses sampling-based methods such as granular synthesis to produce engine sounds.

There are also tools and resources coming out from the research community that can be used to synthesise sound effects or ease the development of algorithms using different techniques. The Synthesis ToolKit (STK) [109, 110] is an open source cross-platform C and C++ programming language library released in the 1990's, featuring a wide range of synthesis algorithms and convenient functions (e.g., audio callback, audio effects). Around the same date, SynthBuilder [111] was introduced, providing a graphical environment for creating signal-processing algorithms by combining building blocks. Fol-

[2] http://tsugi-studio.com/web/en/products-gamesynth.html

[3] https://www.audiokinetic.com/en/products/plugins/impacter/

[4] https://lesound.io/

[5] http://www.crankcaseaudio.com/

lowing the graphical environment paradigm, in the book “Designing Sound” Farnell [9] introduces multiple Pure Data patches of sound effects with parametric controls. There are also high-level spectral analysis tools that facilitate the adoption of analysis-synthesis methods, such as SPEAR [112]. Concerning frameworks tailored specifically for physical models, examples include Modalys [113] or MOSAIC [54].

More recently, there have been a series of tools and libraries released for the Faust functional programming language, such as the Faust-STK (inspired by the original Synthesis Toolkit) [114]; or the Faust Physical Modelling Library [115] which, as it is built upon Faust, it is able to export its code to other platforms such as Pure Data or the Unity game engine. Another compelling tool built upon Faust is Mesh2Faust [116], which takes a 3D mesh – such as a bell – and transforms it into a physical model, extracting its modal information by performing a finite element analysis. Additionally, there are other audio programming languages incorporating physical models libraries, such as ChucK [117], which can also be integrated in interactive platforms, such as in the Unity game engine via the Chunity plugin [118].

2.4 Chapter Summary

Modelling audio signals using digital signal processing components is a field of knowledge that has been active for more than 70 years. Research in digital sound synthesis has led to multiple techniques to this end, some of them catered towards the generation of sound effects – often called procedural audio in this context [9]. This chapter summarises those techniques and provides examples of their applications, as well as tools coming from both academia and industry products.

However, while some techniques may be effective (refer to Chapter 4), digital sound synthesis remains an open research problem [9, 16, 17, 18, 8], which causes – at least, partially – the limited adoption procedural audio has among audio professionals [9]. Nonetheless, novel data-driven approaches have shown promise in audio generation, outperforming "classic" DSP techniques for tasks such as speech synthesis [19]. Chapter 3 introduces those novel techniques and highlights their applications for sound synthesis.

CHAPTER 3

DEEP LEARNING

3.1 Fundamentals of Deep Learning

While some of the techniques illustrated in Chapter 2 can be used to effectively create procedural audio models – as it will be shown in Chapter 4, crafting such models usually relies on a time-consuming process of adjusting the algorithm until it produces the desired result. Additionally, more complex sounds or interactions may result in highly-engineered and hand-crafted algorithms to capture the granularity of their behaviour. Thus, automating – at least part of – this process can help building these systems. Conveniently, machine learning is the field of knowledge that studies programming computers so they learn from experience rather than being explicitly programmed [119]. In other words, machine learning models learn from data (i.e., examples), extracting patterns from it [120]. Deep learning (DL) is a form – or a subset – of machine learning that learns complex concepts from simpler concepts [120], typically stacking multiple layers (hence the *deep*) to hierarchically extract features from the data. Both machine learning and deep learning are under the broader artificial intelligence (AI) umbrella [120].

Despite artificial neural networks (ANNs), which are loosely biologically-inspired and at the core of deep learning, were introduced back in 1943 [121], it is only in recent years that their adoption has become widespread. This is partly due to the – relative – abundance of data available to train the models, the increase in computing power over the years, and the improvement of the algorithms coupled to the research output available [122]. The proliferation of deep learning techniques has inspired multiple reviews (e.g., [123, 124, 125, 126, 127, 128, 129, 130, 131, 132]) and books (e.g., [120, 133, 134, 122]) about the field, both on general deep learning and on audio-oriented topics. From automatic speech recognition to text-to-image applications, a wide range of deep learning architectures have been extensively employed across multiple industries for different tasks, outperforming other "classic" approaches and achieving state-of-the-art results. Consequently, it is also an attractive method to explore in the context of the synthesis of sound effects.

The most archetypal deep learning architecture is the feed-forward multi-layer perceptron (MLP), depicted in Figure 3.1. Multi-layer perceptrons *just* map some inputs to some outputs by stacking many simple functions together [120]. For each of the neurons, their inputs are scaled by a series of weight w parameters, summed together, and the result is added with a single bias b parameter. Those two parameters (weights and biases) are in fact the learnable parameters of a *basic* MLP. Finally, the output of each of the neurons is typically processed by an activation function, which enables the network to introduce non-linearities. This process is depicted in Figure 3.2.

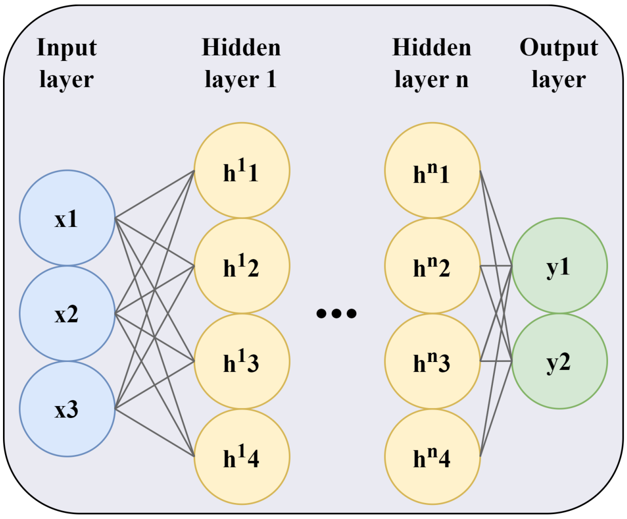


Figure 3.1: Diagram of a multi-layer perceptron. The network consists of 3 inputs that are fed into n hidden layers comprised of 4 hidden units (i.e., neurons) each, and finally to a last output layer, yielding two outputs. The number of inputs, hidden layers, hidden units, and outputs can be modified.

Different activation functions are suitable for different tasks or approaches, leading to a difference in performance depending on this choice [135]. Among others, two typical activation functions are the rectifier linear unit (ReLU), which sets negative values of x to 0, defined as [136]:

$$\mathrm{ReLU}(x) = \max(0, x), \tag{3.1}$$

or the hyperbolic tangent (Tanh), which effectively squashes x to a $[-1, 1]$ range, defined as [135]:

$$\mathrm{Tanh}(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \tag{3.2}$$

A MLP learns by adjusting its parameters (i.e., its weights and biases) to minimize the loss on its training data. This process involves backpropagation, which computes the gradient of the parameters with respect to a

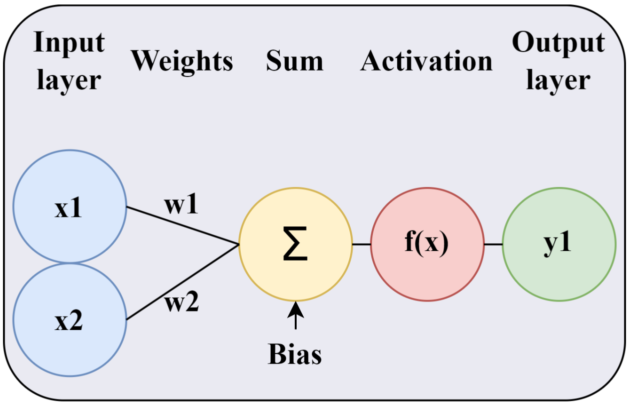


Figure 3.2: Diagram of a single neuron. Two inputs, x_1 and x_2 are multiplied by two weights w_1 and w_2 and summed together. A bias b parameter is summed to the result, which is then processed by an activation function, yielding the neuron output y_1.

loss (i.e., a fitness function or metric that measures the performance of the network for a given task), often using automatic differentiation libraries such as TensorFlow[1] or PyTorch.[2] The computed gradients are then used by a gradient descent optimisation algorithm to update the network parameters in a direction that reduces the loss (thus improving its performance) [120]. The size of the step that the gradient descent algorithm takes in the direction of the negative gradients is determined by the learning rate, a *hyperparameter* (i.e., a variable that is not inherently part of the parameters of a network) which constitutes an important factor to consider to ensure a stable and optimised training. The process of updating the network parameters is repeated, step by step, until it converges to the point where further iterations do not improve the network's performance.

During the training process, it is common to use techniques to aid the network to learn in a stable manner. For instance, initialising the network

[1] `https://www.tensorflow.org/`

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

parameters with a specific strategy, such as the Xavier [137] or He [138] initialisations, may help it to converge faster [120]. Likewise, normalisation techniques, such as batch [139] or layer [140] normalisations can significantly improve the training process [141], ensuring that the activations throughout the network maintain a consistent scale and distribution. Another important factor is the network's ability to perform well on inputs apart from those seen in training data, increasing its *generalisation*. The group of strategies aimed at increasing a network's generalisation is called *regularisation*, defined as modifications made to a learning algorithm aimed to reduce its generalisation error but not its training error [120]. Common regularisation techniques are dropout [142], which randomly deactivates a subset of neurons in a network layer, or data augmentation, which applies transformations to the training data (e.g., rotating an image or pitch-shifting an audio signal) in order to increase the amount examples in the dataset without actually collecting new data [120].

Apart from the MLP example illustrated above, deep learning offers a large array of networks and layers, each one tailored to different types of data. Namely, convolutional neural networks (CNNs) [143] employ convolutional layers, which use convolutional operations instead of matrix multiplications as in traditional neural networks [120] to extract *feature maps* from the data, usually in the form of multi-dimensional arrays (i.e., *tensors*). Convolutional layers usually involve pooling, where the output feature maps are downsampled by replacing the outputs of a certain region with a statistic of those values [120], effectively reducing their dimensionality and helping to capture complex features from the data as the networks grows deeper.

3.2 Generative Deep Learning

Generative models learn to represent an estimate of a probability distribution, p_{model}, learning it from a training dataset p_{data} [124]. Generative models traditionally encounter challenges such as the interdependency among features (e.g., the relationship between two pixels in an image) or the fact that only a small set of solutions provide a satisfactory sample (e.g., the chances of producing a plausible image by just shuffling pixel values are remote) [133]. Deep learning addresses these challenges by using *representation learning*, where a model automatically learns hierarchical representations from the data, capturing complex patterns from it. Therefore, when using deep learning for *generating* data, this is usually referred to as *generative deep learning* or *deep generative modelling*. When high-level features are not directly modelled, they are described as lower-level *latent* representations, learning the mapping between the training examples and their representation in the *latent space* [133]. Although not all generative deep learning models necessarily produce a latent space or use latent variables explicitly (i.e., learn compressed or intermediate representations from the data) such as – in general – autoregressive models (see Section 3.2.3), latent variable models are very extended in generative deep learning.

While there are multiple methods for estimating the probability distribution p_{model}, most deep learning approaches employ the principle of maximum likelihood, defined as [120]:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{m} \log p_{\text{model}} \left(x^{(i)}; \theta\right) \tag{3.3}$$

where θ are the model parameters and $x^{(i)}$ is the i instance of a dataset comprised of m examples. This principle seeks to tune the model parameters θ so they maximise the likelihood of the training data [124]. In simpler terms, once trained, generative models aim to produce data in the style of the data they were trained on. Also, depending how they compute or represent the likelihood, deep generative models can be seen as explicit density models when they explicitly define a density function $p_{\text{model}}(x;\theta)$, such as autoencoders, autoregressive models, flow-based models, or diffusion models; or implicit density models when they learn the density function $p_{\text{model}}(x;\theta)$ indirectly, such as generative adversarial networks [124].

Generative deep learning offers a wide set of creative possibilities and applications in the audio domain. For instance, it can be used to: synthesise new data from a dataset (e.g., generating novel footstep sounds [18]); manipulate generated data by traversing the latent space (e.g., traversing a region in the latent space to increase or decrease a desired feature [144]); learn control mappings (e.g., generating audio by specifying an instrument pitch and loudness [145]); perform inpainting (e.g., reconstructing a section of a data instance that has been masked [146]).

This section will introduce multiple generative deep learning techniques that are currently being employed in state-of-the-art approaches within the field of sound generation. While there are more generative modelling techniques prior to deep learning, and other deep learning approaches and hybrid models that combine multiple of these methods together, they are left out of this section due to the scope of this thesis, such as energy based models [147].

3.2.1 Autoencoders

Autoencoders (AEs), which were formally proposed back in the 1980s [148], are a class of models that aim to reconstruct an output from its input [123]. They are usually comprised of two networks: an encoder that compresses high-dimensional data X to a lower-dimensional representation z; and a decoder that reconstructs the original data $\hat{X}$ back from this lower-dimensional representation z [133]. They can be seen as unsupervised compression algorithms, where the – unless conditioned, unlabelled – data is passed through a bottleneck network to end up mapped as a set of latent variables $X \rightarrow z$, forming the latent space. The decoder learns the inverse mapping $z \rightarrow \hat{X}$ which results, once trained, in a deterministic set of latent variables for each data instance (i.e., each data instance is "projected" to a deterministic point in the latent space). This deterministic nature makes traditional autoencoders not very good at either structuring the latent space or compressing (i.e., reconstructing) the data [134], as the latent space may not be continuous (i.e., neighboring points do not necessarily contain similar features) or complete (i.e., regions of the latent space may not produce relevant content).

Variational autoencoders (VAEs) [149, 150] aim to solve these issues by encoding the data not as a deterministic point in the latent space, but as a probability distribution $p(z)$ of μ mean and σ^2 variance instead. Thus, the encoder computes $q_\phi(z|x)$ and the decoder $p_\phi(x|z)$. To prevent $p(z)$ from being skewed and impose a prior, a regularisation Kullback-Leibler (KL) divergence loss is used to penalise $p(z)$ from deviating from a normal Gaussian distribution $\mathcal{N}(0, 1)$. β-VAEs [151] introduce an additional constraint by scaling the KL regularisation term in the loss function, which aids in disentangling the latent space. The objective function can be then written as follows [151]:

$$\mathcal{L}(\theta, \phi; \mathbf{x}, \mathbf{z}, \beta) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \beta \ \mathcal{D}_{\mathrm{KL}}[q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})] \tag{3.4}$$

where the first term is the reconstruction loss, the second term is the KL divergence loss, and β is the parameter that controls how much of the regularisation affects the expression. Higher (*i.e.*, > 1) values of β increase the latent space disentanglement, but reduce the reconstruction capabilities in return.

There are many other relevant VAE-based architectures. For instance, the Vector Quantised Variational Autoencoder (VQ-VAE) [152, 153], which instead of learning a continuous latent representation as in a "regular" VAE, it generates a discretised latent space, contributing to a more efficient latent space representation and – in general – to a higher quality output and a more stable training process; or the Nouveau VAE (NVAE) [154], which implements a deep hierarchical structure alongside a carefully selected set of normalization and regularization techniques, making it capable of producing large images and yielding state-of-the-art results.

3.2.2 Generative Adversarial Networks

Generative adversarial networks (GANs) [155] are a generative deep learning modelling technique where – at least – two networks compete against each other. These networks are the discriminator, which tries to differentiate between real and fake data; and the generator, which learns to produce plausible data by attempting to fool the discriminator. During training both networks are trained jointly. On the one hand, the discriminator D is trained in a supervised learning fashion [124], predicting whether the input data is real or fake. On the other hand, the generator G tries to learn a probability distribution p_g over the real data x by sampling from a – usually Gaussian – prior distribution $p_z(z)$ to generate data that is indistinguishable from the real data x to the discriminator [155]. Thus, the two networks engage in a min-max game, the solution of which is a Nash equilibrium [124], and the value function $V(D, G)$ is defined by [155]:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \quad (3.5)$$

In other words, D tries to both maximise and minimise the values of $D(x)$ and $D(G(z))$ respectively, and G tries to maximise $D(G(z))$ [124]. The process is depicted in Figure 3.3. Once trained, sampling from $p_z(z)$ results in new data in the style of the training data x (i.e., $G(z) \to x'$).

GANs are, however, hard to train. Among other issues, their loss is not informative during training (i.e., the generator loss is only compared against the discriminator), they suffer from mode collapse (i.e., the generator can find small subsets of generated data that constantly fool the discriminator), and their hyperparameters can also be large and difficult to adjust [133]. Wasserstein GAN (WGAN) [156] and Wasserstein GAN + Gradient Penalty (WGAN-GP) [157] address these problems by introducing a loss that correlates with the generator performance, achieving also a better training stability.

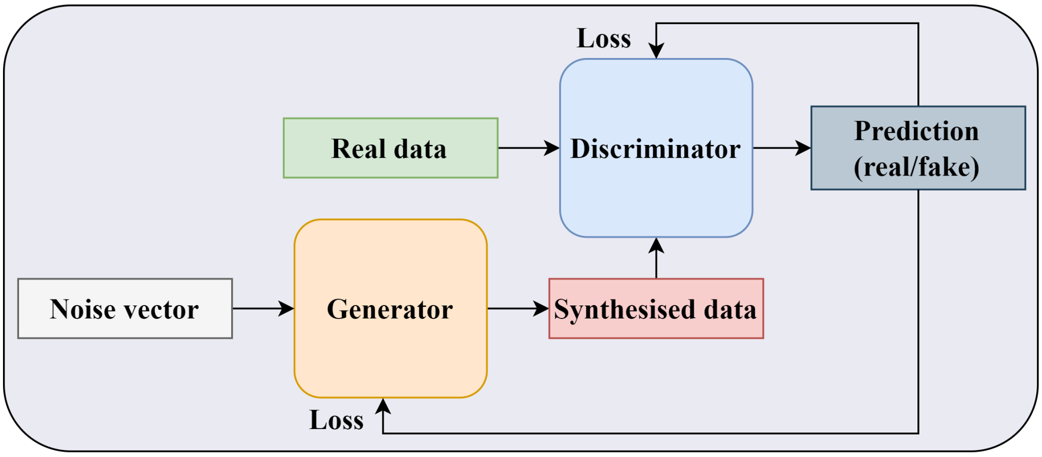


Figure 3.3: Diagram of a generative adversarial network. The generator aims to fool the discriminator by synthesising outputs that resemble the real data, while the discriminator aims to distinguish between real and synthetic data.

GANs have been very prolific in the generative deep learning literature (especially before the recent proliferation of diffusion models, see Section 3.2.5), obtaining the state-of-the-art results at multiple points in time with popular architectures such as Deep Convolutional GAN (DCGAN) [158], BigGAN [159], Progressive Growing GAN (PGAN) [160] or StyleGAN [161, 162, 163].

3.2.3 Autoregressive Models

Autoregressive models [164] are especially suited to model sequential data such as text or audio, where each new sample in a sequence depends on the past ones [131]. They are based in the chain rule of probability, where the probability of a variable that can be decomposed as $x = x_1, ..., x_n$ can be represented as a product of conditional probabilities [128]:

$$p(x) = p(x_1, ..., x_n) = \prod_{i=1}^{n} p(x_i|x_1, ..., x_{i-1}) = \prod_{i=1}^{n} p(x_i|x_{<i}) \qquad (3.6)$$

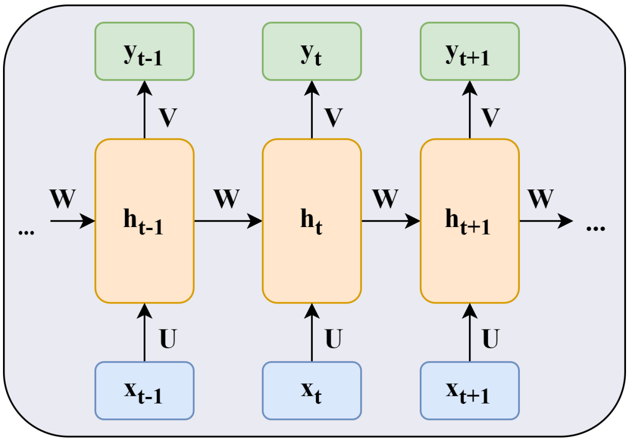


Figure 3.4: Diagram of a typical recurrent neural network. At each timestep t the hidden states h_t are computed, parametrised by an input-to-hidden weight matrix U, a hidden-to-hidden weight matrix W, and a hidden-to-output weight matrix V.

Over the years, there have been multiple architectures and models that fit into the autoregressive paradigm. In this context, particularly relevant are recurrent neural networks (RNNs), which are neural networks containing an internal loop (hence the *recurrent*) that iterate over sequences, keeping track of an internal state based on the elements seen at each time step [134], as depicted in Figure 3.4. However, RNNs suffer from a vanishing gradient problem in longer sequences [133], where the gradients become increasingly small over it and effectively prevents the network from learning, as the update to its weights becomes proportionally small. To solve this problem, Long-Short Term Memory (LSTM) networks [165] were proposed, which expand RNNs by introducing a memory cell that allows the network to selectively remember (or forget) information, allowing it to model longer sequences without the vanishing gradient problem. Another popular RNN architecture are Gated Recurrent Units (GRUs) [166], which are somewhat similar to LSTMs in terms of design and performance [167], but simpler conceptually.

Apart from RNNs, other autoregressive methods use temporal (i.e., *causal*) convolutions to model sequential data, such as in [168, 169] where they employ this method applied to the computer vision domain. A similar approach is used in the audio field in the WaveNet architecture [19]. Yet another very popular autoregressive architecture, different from the aforementioned ones, is the transformer architecture [170]. Transformers use self-attention (a mechanism to determine which previous time steps to consider at the current time step [128], or in other words, select which ones to pay *attention* to), granting them the ability to connect distant dependencies and therefore model longer sequences. Transformers are currently very prolific, being widely employed in the context of large language models (LLMs), such as GPT-4 [171]. Beside such approaches, there are also models that mix multiple methods, such as in [172], where they combine causal convolutions with self-attention.

3.2.4 Normalising Flows

Normalising flows are rooted in the idea of directly modelling a complex data distribution by transforming a simpler data distribution (e.g., a standard normal $\mathcal{N}(0, 1)$) through a series of invertible and differentiable mappings [173]. This series of invertible mappings transform the input data into latent representations [131]. Thus, a vector x can be represented as the result of applying a transformation T to a vector z sampled from $p_z(z)$ [174]:

$$x = T(z), \text{where } z \sim p_z(z) \tag{3.7}$$

Since it is common to link k transformations together, these can be expressed as follows [131]:

$$x = T_0 \circ T_1 \circ ... \circ T_k(z) \tag{3.8}$$

$$z = T_k^{-1} \circ T_{k-1}^{-1} \circ ... \circ T_0^{-1}(x) \tag{3.9}$$

with those mappings parameterised by the deep learning model. The term “normalising flows” is given by these transformations: a collection of samples from $p_x(x)$ are “normalised” to $p_z(z)$, and the trajectory that maps the samples from $p_z(z)$ is known as the “flow” [174].

3.2.5 Diffusion Probabilistic Models

Diffusion probabilistic models [175, 176], more commonly referred to as simply “diffusion models”, are based on the concept of gradually introducing noise to a data sample $x_0 \sim q(x_0)$ using a noise schedule $\beta_{1:T}$ so the resulting data x_T at the end of the process (i.e., after T iterations of adding noise) is normally distributed [128] (i.e., $p(x_T) \approx \mathcal{N}(0, I)$ for large values of T [176]). More formally, diffusion models typically consist of two processes: the forward (or *diffusion*) process and the reverse process. First, the diffusion process, where the data $x_0 \sim q(x_0)$ is gradually corrupted by noise over T steps, is defined by a Markov chain as follows [176]:

$$q(x_1, ..., x_T|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1}), \tag{3.10}$$

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I) \tag{3.11}$$

where the resulting $x_1, ..., x_T$ at each time step are latents of the same dimensionality of the original data $x_0 \sim q(x_0)$ [176], T are the diffusion steps (the length of the Markov chain) and $\beta_1, ..., \beta_T$ are typically small positive constants that relate to the noise variance schedule.

The reverse process (the process that transforms x_T back to x_0) is also defined by a Markov chain parametrised by θ as follows [177, 176]:

$$p_\theta(x_0, ..., x_{T-1}|x_T) = \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \tag{3.12}$$

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \tag{3.13}$$

with $p(x_T) = \mathcal{N}(0, I)$. Thus, the aim of $p_\theta(x_{t-1}|x_t)$ is to remove the noise added in the forward process [177]. By doing so, new data can be generated by drawing a sample $x_T \sim \mathcal{N}(0, I)$ and iteratively transforming it through the reverse process (i.e., $x_{t-1} \sim p_\theta(x_{t-1}|x_t)$ for $t = T, T-1, ..., 1$) until x_0 – the *denoised* data point – is produced [177].

Diffusion models have recently emerged as one of the most promising paradigms in generative deep learning, achieving state-of-the-art performance in tasks such as large-scale text-to-image using a combination of diffusion models and CLIP embeddings [178],[3] with models like the popular DALL-E

[3] "Contrastive Language-Image Pretraining", used to capture relations between images

2 [179] among many others.

3.3 Neural Audio Synthesis

Neural audio synthesis refers to the use of deep learning techniques for the synthesis of sounds [180]. Similar to Chapter 2 on DSP, this section will introduce the applications of the theory that has been discussed immediately prior (see Section 3.2).

It is common that advancements in the deep learning domain generally appear first in the computer vision field (such as image synthesis), and they are then adapted to the audio domain. Audio however has its own challenges, such as the high dimensionality of the data [181] where, for instance, a second of audio sampled at 44.1 kHz contains 44 100 data points, all of them related to each other at different scales. Thus, the audio representation (i.e., how the audio is fed to the network) is an important factor to consider. Deep learning models usually work in either the time domain (i.e., raw audio samples) or in the frequency domain (i.e., spectrograms) [131]. In [182], they compared different representations for an audio PGAN [160] architecture trained on musical notes, ranging from raw audio to different frequency domain representations, such as complex values taken directly from the STFT or the constant-Q transform [183]. In their context, they found that complex values and magnitude and instantaneous frequency spectrograms performed the best as audio representations for the task. Alternatively to time or frequency domain representations, another approach may consist of using DSP components such as filters or oscillators in conjunction with deep learning

and text descriptions.

(see Section 3.3.1).

Another fact to consider when modelling audio is the suitability of the loss (i.e., fitness) function used during training; and the objective and subjective metrics used to evaluate the performance of a model on a particular task.

Apart from methods that do not "see" the data directly and model it by proxy (e.g., adversarial training in GANs), it is important to be able to compute the similarity among two signals for tasks such as reconstruction in VAEs. While this could be done by performing a point-wise comparison between two waveforms, identical waveforms may sound different, and perceptually identical audio samples may have different waveforms [145]. Another approach could be comparing the STFT of two audio samples, but the STFT suffers from a time and frequency resolution trade-off, where higher FFT values lead to a better frequency resolution but poorer time resolution, and vice-versa. As an alternative, a multi-scale short-time Fourier transform can be used, where instead of a single FFT, a series of STFTs with different FFT values are computed to compensate for their time and frequency trade-off [184, 185, 145]. There are other alternative representations, such as the wavelet-based joint time-frequency scattering (JTFS) [186], used recently in the context of deep learning and audio classification with promising results [187, 188]. Incorporating perceptual considerations have been also explored, such as in [189, 190], where they build a differentiable perceptually-driven similarity metric by asking participants to discriminate between perceptually similar samples. In terms of usability, there are packages such as auraloss [191] which implements multiple audio-focused loss functions in Pytorch.

Regarding the evaluation (i.e., assessing how well a model accomplishes a task), this is usually achieved through subjective evaluations, objective evaluations, or both. Subjective evaluations typically consist of conducting a listening study in which participants are asked to rate the data generated by the model. This could be done to assess the quality (e.g., how plausible a sound is for a certain category), the diversity (e.g., how different multiple samples generated by a model are) or the similarity (e.g., how close the generated and ground truth samples are, useful in reconstruction tasks), to name a few; assessing either a particular model alone (e.g., "Are the sounds produced by the model plausible?") or several models against each other (e.g., "Is this model more plausible than the other?"). On the other hand, objective metrics rely on computational methods to assess a model's performance. While, for purely reconstruction tasks, the loss functions outlined above can be used to evaluate a model's reconstruction capabilities, there are also other metrics that can be used to objectively evaluate deep learning models. For instance, the Fréchet Audio Distance (FAD) [192] is a popular reference-free evaluation metric that correlates well with human judgment. The FAD is calculated by extracting the embeddings from both the generated sounds and the ground truth audio using a pre-trained VGGish [193] classification model and computing multivariate Gaussians on them to calculate their Fréchet distance [194]. There are other objective metrics, such as the Inception Score (IS), [195] which is a metric created to assess class-conditioned GANs on quality and diversity, the Number of Statistically-Different Bins (NDB) [196] to measure the generated output diversity, or the Kernel Inception Distance (KID) [197] to compare distributions of the generated and ground truth data, to name a few.

With regards of the methods themselves, one of the first major leaps in audio synthesis using deep learning was WaveNet [19], introduced in 2016. WaveNet, which is an autoregressive method based on the PixelCNN architecture [168, 198], employs dilated convolutions to increase the network's receptive field and capturing long-term dependencies in the waveform domain. This approach granted WaveNet the ability to generate raw audio at a 16 kHz sampling rate, establishing the state of the art in text-to-speech applications at its time, reducing the gap with human performance by 50%. WaveNet-like architectures have been proposed to perform other tasks, such as musical note generation [180], among many others. While successful, WaveNet suffers from a very slow inference time, as the audio samples are synthesised sequentially, one after the other. Parallel WaveNet [199] addresses this problem by proposing a flow-based approach, speeding up the architecture by a factor of 1000 with no perceptual differences. Another autoregressive approach that generates raw audio, released very close to WaveNet in time, is SampleRNN [181], which introduces a model comprised of multiple RNNs using a hierarchy of different scales to overcome the challenge of modelling high dimensional data.

Neural audio synthesis has made significant progress since its early stages and, while it is arguably still in an early phase of development, it has been a very active area of research, employing a wide range of techniques. For instance, regarding GANs, a notable architecture is WaveGAN [200], which was the first attempt at generating unconditional raw audio with an adversarial training approach, proposed in 2018. WaveGAN is based on the DCGAN architecture [158], but adapting it to model raw audio by replacing DCGAN's two-dimensional convolutions (designed for processing two-dimensional data like images) by one-dimensional convolutions, effectively "flattening" the DCGAN architecture, allowing the processing of time-series data. In the WaveGAN paper, Donahue et al. also proposed a frequency domain variant of the architecture: SpecGAN [200]. SpecGAN produces log-magnitude spectrograms that are then transformed back to audio using the Griffin-Lim algorithm [27], mimicking the DCGAN architecture in this case, but using spectrograms as the data representation (instead of images). Its non-autoregressive nature granted WaveGAN an inference speed orders of magnitude faster than, for instance, WaveNet. Very close in appearance to WaveGAN, another popular audio GAN architecture was introduced: GANSynth [201]. GANSynth is built upon the PGAN architecture [160] but adapting it to generate audio spectra instead of images, incorporating also a pitch conditioning vector and an auxiliary loss to aid the discriminator in the pitch label classification [202]. They experimented with multiple audio representations, such as a model using magnitude and phase spectrograms, a model using instantaneous frequency as a replacement for the phase, or a model using instantaneous frequency along with – invertible – mel-scale magnitude spectrograms. In their experiments, they outperformed WaveGAN in the task of producing pitched musical notes. Like WaveGAN, GANSynth produces audio orders of magnitude faster than WaveNet (54K times faster, specifically). Regarding other generative methods, flow-based modelling is used in models like FloWaveNet [203] or WaveFlow [204]; or diffusion models, which are used in architectures such as DiffWave [177] or WaveGrad [205] for waveform generation. Lately, there has been an emphasis on achieving real-time performance, desirable for live adaptability and audio manipulation. RAVE ("Realtime Audio Variational autoEncoder") [206] is an example of this, capable of producing waveforms at a 48 kHz sampling rate with an inference speed 20 times faster than real-time on a consumer laptop CPU.

Apart from the choice of different generative methods, yet another relevant factor to consider is how a particular model produces the data, and whether this can be controlled or not. Unconditional models (such as the original WaveGAN [200]), generate samples from the learned data distribution without any conditioning information or control. Depending on the model, it is possible however to affect the output data by, for instance, steering the latent space in a particular manner, such as in [207], where they use interactive machine learning to learn mappings between a RAVE [206] model's latent space and human performances. RAVE is also capable of timbre transfer, which involves using a model trained on a specific category of sounds (e.g., sax improvisations) to apply the acoustic characteristics of an out-of-domain sound (e.g., singing voice; or a different sax sound) to the timbre of the former. This effectively transforms one audio into the other and drives the synthesis in this manner. Other models take conditional inputs, such as in the aforementioned GANSynth [201], where the synthesis is conditioned by the pitch of a musical note. Apart from pitch, the – discrete – conditioning labels could represent other characteristics of the sound, such as high-level timbral features, demonstrated in [208]. It is also possible to provide continuous conditioning, such as in [145], where pitch and loudness vectors are used. Another approach is to generate music for a particular genre, as in Jukebox [209]; or to describe the sound through natural text prompts, as recently introduced in MusicLM [210] or MusicGen [211]. Apart from explicitly specifying desired controls, another option could involve masking (i.e., removing) a section of an audio file or spectrogram, and subsequently reconstructing (i.e., generating) a new version of the masked section, such as in [146].

While most of the focus on neural audio synthesis has been on either the speech or music synthesis domains, there has been work addressing sound effects specifically. One of the challenges to bring data-driven methods to this category of sounds has been the scarcity of high-quality data, as training audio deep learning architectures typically demands a large amount of samples. However, there has been an effort to collect and bring annotated datasets to the public. These datasets are mostly catered to the sound event recognition (SER) task, but may be compatible with sound generation nonetheless, despite the quality expectations from one task to the other may differ (i.e., for SER applications, sounds may be noisy and mixed with other background sounds as the aim is to recognise events in a real-world scenario, while in the generation task one of the aims is to generate high-quality sounds comparable to those found in a professional sound library). The UrbanSound8K dataset [212] is an example of this, containing almost 9000 $\approx$4-second audio clips from 10 categories of urban sounds. Likewise, the ESC-50 dataset [213] brings a collection of 2000 short clips of 50 sound classes. The Audio Set [214] is a human-annotated dataset comprised of over 1 700 000 10-second segments of 632 sound classes collected from YouTube videos, distributed as pre-computed features rather than raw audio. The FSD50K dataset [215] brings more than 50K manually-annotated clips of 200 different sound classes, sourcing the sounds from the Freesound [216] website and distributing them as raw audio with a permissive license. Recently, the Epic-Sounds dataset [217] was released, comprising of more than 70K categorised segments across 44 classes, containing the sound events alongside a first-person video of the actions that produced them. Additionally, there are datasets focused on the vocal imitation of sound effects, such as [218, 219]; or datasets of specific sounds, such as gunshots originating from different weapons and perspectives [220]. Apart from curated datasets, there are platforms such as Freesound [216], where users can upload sounds and distribute them with a permissive license at no cost.

Regarding research on sound generation for sound effects using deep learning, [221] proposes the use of a conditional SampleRNN [222] to generate acoustic scenes (e.g., the sound of an office, a beach or a park). In [18] footsteps are synthesised conditioned on surface materials using a hybrid network comprised of a conditional WaveGAN generator [223] and a HiFi-GAN discriminator [224]. In [225] they use a VQ-VAE [152] to generate sounds of the 10 categories of the UrbanSound8K dataset [212], such as siren or street music. [226] uses a WaveFlow [204] architecture to synthesise explosion sound effects, successfully achieving timbre-transfer as well. There has also been research on the use of vocal imitations [227] or onomatopoeic words [228, 229] to condition the sound effect generation. Regarding situations where the data is either scarce or difficult to obtain, in [230] they synthesised longer audio streams from just $\approx 20s$ of data. Especially relevant for VR scenarios, in [231] they propose using a Y-Autoencoder [232] to disentangle gestures and material interactions within the context of sound effect synthesis.

With the recent increase of popularity of diffusion models and text-conditioned architectures, there has been a proliferation of studies combining these approaches with sound effects. For instance, in [233] they generated sound effects conditioned on several classes, such as air conditioner or children playing; or in [234, 235, 236, 237, 238] they conditioned the generation with natural text prompts (e.g., "a dog barking in a park"). There has also been research to automatically incorporate sound to silent videos using a diverse set of methods, synchronising the generated sound effects to the actions in the video such as, for instance, in [239, 240, 241, 242, 243].

Due to its relatively early stage, the adoption of neural audio synthesis among users has not yet reached the same level of widespread usage as its equivalent DSP counterparts, particularly within the field of sound design. Computing requirements such as the need – in some cases – of high-end GPUs to run the models, the lack of established workflows and a higher bar to access user interfaces (such as command-line applications) may have hampered their adoption, keeping the use of these models to the research community. However, there has been an increasing interest in bringing neural audio synthesisers to the general public using tools that prospective users are already familiar with. A middle-ground between command-line applications and high-level tools are web services such as Hugging Face spaces,[4] where they offer a front-end platform backed by computational resources to run models directly from the web. For instance, the AudioLDM [236] text-to-audio model is accessible trough it.[5] There have been recent model-specific solutions to deploy neural audio synthesisers to existing tools, such as DrumGAN VST [244], integrated in the Steinbergs's Backbone drum VST,[6] or

[4] https://huggingface.co/spaces/

[5] https://huggingface.co/spaces/haoheliu/audioldm-text-to-audio-generation

[6] https://www.steinberg.net/vst-instruments/backbone/

VST versions of the DDSP architecture [145, 245].[7] Another recent example is Neutone,[8] a platform that aims to bring pre-trained models (including neural audio synthesisers) to digital audio workstations in the form of a VST plugin, offering a simple interface for tasks such as timbre transfer, but with a "model-sceptic" approach. Audio-centric GPU acceleration is also being actively developed in projects such as GPU Audio.[9] Regarding specific frameworks, recently ChAI[10] (ChucK for AI) was released with the aim of providing a real-time interactive platform for musicians, incorporating machine learning algorithms to the ChucK programming language. There is also an audio-centric neural network library, RTNeural,[11] which is built with a performance and real-time usage in mind.

Outside of the synthesis context, there are other existing AI-powered tools within the audio and music industry, such as the LANDR online mastering service[12] or the NeuralDSP[13] AI-powered digital guitar amplifier simulation suite, to mention two relevant examples.

3.3.1 Differentiable Digital Signal Processing

Differentiable digital signal processing (DDSP) [145] is based on the idea of employing DSP techniques and components like those introduced in Chapter 2 – such as oscillators or filters, and optimise their parameters through gradient descent, often integrating them to a deep learning architecture. Pre-

[7] `https://magenta.tensorflow.org/ddsp-vst`
[8] `https://neutone.space/`
[9] `https://www.gpu.audio/`
[10] `https://chuck.stanford.edu/chai/`
[11] `https://ccrma.stanford.edu/~jatin/rtneural/`
[12] `https://www.landr.com/online-audio-mastering/`
[13] `https://neuraldsp.com/`

cisely, in the original DDSP paper [145], they address the challenge of building expressive sound models that can be trained using gradient-based optimisation methods. Traditional DSP algorithms are typically non-differentiable (i.e., they do not allow gradients to propagate through them), making them difficult to incorporate into deep learning frameworks. DDSP provides a solution by introducing a set of differentiable DSP modules that can be used as building blocks for constructing complex sound models. To illustrate their approach, they build a differentiable spectral modelling synthesiser [31] (see Section 2.2.3.3) comprised of a harmonic (i.e, harmonic sinusoids) and a stochastic (i.e., subtractive synthesiser) components, conditioning the model on pitch and loudness. They also compute the room acoustics of the sound source during training by modelling an impulse response common to the dataset. An overview of their proposed architecture is depicted in Figure 3.5. As a result, they were able to build a synthesiser with human-interpretable controls, capable of producing plausible musical notes. Since the control parameters (i.e., pitch and loudness) are extracted from the data itself, they can also achieve timbre transfer. This is done by extracting the desired features from a target audio to be transformed (e.g., a voice), feeding them to the synthesiser and transforming that audio into musical notes with the timbre of the instrument the model is trained on (e.g., a violin). The effectiveness of DDSP (i.e., high-fidelity synthesis capabilities alongside human-interpretable controls) is due to exploiting the inherent biases of the DSP components which, in contraposition of the undesired potential biases introduced by waveform or Fourier-based models, are in fact beneficial to the synthesis task [145].

While there was some work combining DSP component with neural net-

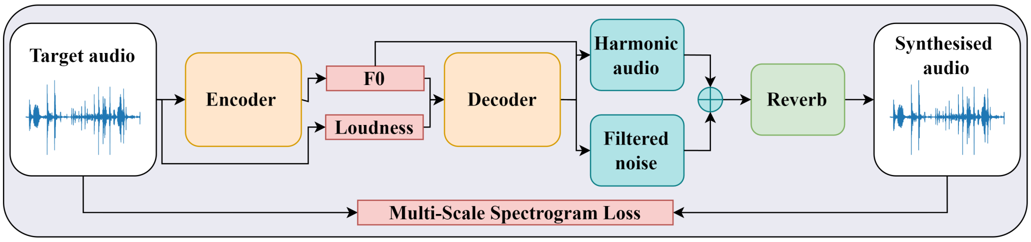


Figure 3.5: Simplified diagram of the DDSP architecture [145]. The fundamental frequency $F0$ is extracted using a pre-trained CREPE [246] encoder, while the loudness is extracted directly from the target audio signal. The decoder predicts N harmonic sinusoidal amplitudes (using the extracted $F0$ as their pitch) and a time-varying FIR filter, comprising the harmonic and noise synthesisers respectively. The output from both synthesisers is summed, and convolved with a dataset-wide learned impulse response (i.e., its reverb). The final output is compared against the ground truth using a multi-scale spectrogram loss.

works prior to the work of [145], such as in [247, 248] where they use neural networks to estimate the parameters of physical models; or in [184], where the use a differentiable waveshaping method for text-to-speech synthesis, DDSP [145] established the current basis – and arguably more relevant to this chapter, the naming convention – of the technique. Since then, multiple approaches have been proposed using DDSP-inspired architectures. For instance, within a harmonic musical context, there have been architectures using waveshaping [249], wavetable [250], frequency modulation [251, 252], or subtractive [253] synthesis methods. There has also been research on experimenting with network-bending techniques on top of the DDSP architecture [254].

Other approaches focus on synthesizing rigid-body impacts by estimating the parameters of differentiable modal resonators based on the object's shape [255], which they recently implemented in an interactive application [256]. In a somewhat related manner, and closer to physics-based synthesis methods (see Section 2.2.4), in [257] they synthesise modal impacts from 3D shapes; and in [258] they take a similar approach, addressing the acoustic transfer as well. Regarding procedural audio-oriented work specifically, in [259] they build a highly-engineered system to synthesise harmonic engine sounds using a variant of the DDSP architecture and recently, in [260], they synthesised footsteps and pouring water sounds using the DDSP time-varying FIR noise synthesiser.

Another relevant approach related to differentiable digital signal processing is sound matching and automatic synthesiser programming, where a model aims to reconstruct a sound using parametric audio synthesis [261]. In other words, the model takes a target sound as its input, and outputs the synthesiser parameters that reconstruct it. There has been research on this direction such as, for instance, in [262] where they use normalising flows and VAEs for this task; or in [263] where they used a novel convolutional neural network to approximate sounds using a FM synthesiser. To support these efforts, there have been open datasets pairing sounds with synthesiser controls, such as [264], where they released a billion 4-second long synthesised sound corpus along with the associated parameters used in their creation.

It is worth mentioning that, beyond neural audio synthesis, DDSP methods have been also used to model audio effects and audio production processes. For instance, they have been used to model dynamic compressors [265], automatic Disc Jockey (DJ) transitions [266], retrieve audio effect parameters from recordings [267], or distortion modelling in the context of data augmentation for automatic speech recognition [268], to enumerate some examples.

3.4 Chapter Summary

Deep learning techniques have recently achieved state-of-the-art performance in diverse disciplines for multiple tasks, ranging – for instance – from speech recognition [269] to image generation [175, 176] or natural language understanding [171]. Generative deep learning refers specifically to a series of deep learning architectures designed to generate data that resembles the training data of a dataset.

This chapter summarises those techniques, with particular emphasis on deep learning architectures relevant to audio generation – also called neural audio synthesis [180]. The chapter also addresses differentiable digital signal processing, which employs gradient descent to optimise DSP components (such as the ones discussed in Chapter 2), often embedding them within neural networks. Chapters 5, 6 and 7 of this thesis will use the methods introduced in this chapter, but bringing them to the sound effect synthesis domain, and assessing their performance for this task.

CHAPTER 4

EFFECTIVE DSP TECHNIQUES: MODAL SYNTHESIS

4.1 Introduction

Going back to Chapter 1 – the introduction of this thesis – it was outlined that one of the possible causes hampering a wider adoption and spread of procedural audio models according to [9] is the perceived plausibility (i.e., the "realism") of the synthesised sounds. This chapter investigates the suitability of *classic* (i.e., non-neural network based) DSP techniques, such as the ones described in Chapter 2, evaluating them from a perceptual standpoint in order to assess whether or not they can be perceived as "synthetic".

More specifically, this chapter addresses filter-based modal synthesis [56, 22], and evaluates its perceptual performance against pre-recorded samples for a series of impact sounds in a listening study. Filter-based modal synthesis is a technique derived from physical models which is especially indicated for the synthesis of percussive sounds [56, 22], and can be seen as a special case of subtractive synthesis. From a high-level perspective, filter-based modal synthesis uses M narrow parallel filters with their center frequencies defined by, for instance, the frequency peaks extracted from the spectral analysis of a target sound. The resulting sound obtained by filtering white noise through those M filters is subtracted from the original sound source, obtaining a noisy sound envelope from it. The final sound is obtained by combining those two sounds (the result from the filters, also know as the deterministic part of the signal, and the extracted noise envelope, also known as the stochastic part or the residue of the signal) together, summing them in the time-domain. Considering this method uses time-invariant filters and a pre-recorded extracted residue, its computational burden is compatible with real-time scenarios such as video games, making it an attractive method in this context.

Hit or impact-based sounds are the acoustic consequence of physical collisions. Changes in an object material or size will produce changes to the resulting impact sound. For games or interactive applications with hundreds or thousands of interactable assets, such as open world games or VR experiences, combinatorial explosion awaits the sound designer hoping to use pre-recorded samples to design any particular scenario where assets collide [8]. Thus it is desirable, in the context of game audio, to synthesise such sounds in order to overcome those potential time (i.e., designing all the sounds) and storage (i.e., disk space needed to store them) burdens. This chapter addresses the first research question of this thesis (RQ 1), aiming to assess whether or not it is possible for listeners to detect synthesized impact sound effects using filter-based modal synthesis from pre-recorded samples.

4.2 Method

As outlined in Section 2.2.4.3, there are multiple methods to extract the modal information of a system. Modes are the individual sinusoidal frequencies to which an object vibrates. Impact (or percussive) sounds are characterised by a series of decaying modes resulting from striking the object that produces the sound [56]. As an example, Figure 4.1 depicts the waveform (top) and the spectrogram (bottom) of a recorded sound resulting from striking a coffee mug, where the horizontal straight lines in the spectrogram are the modes (or partials) of the sound.

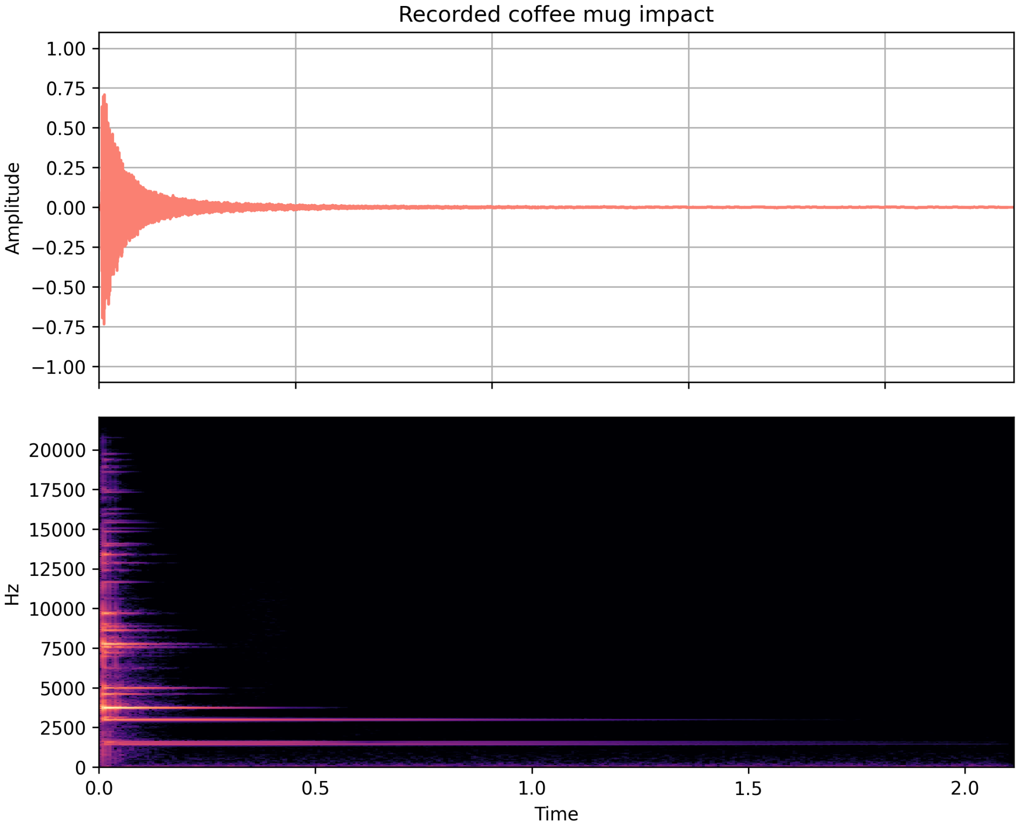


Figure 4.1: Waveform (top) and magnitude spectrogram (bottom) of a recorded coffee mug impact sound.

To extract the modes, it is possible, for instance, to simulate the deformation of objects being struck by performing a system eigendecomposition [55]. However, this chapter adopts an approach derived from physically informed sonic modelling (PhISM) [56], where the modes are extracted from an impact recording by using spectral analysis. This approach allows for the relatively simple extraction of the modal information of any impact recording, granting the source sound exhibits clear modes when struck.

More specifically, to extract the modes from a recorded sound, the code provided by [270], written in the ChucK programming language [117], is used. The process consists of computing a STFT with an FFT of size $L_{\text{FFT}} = 16384$, a hop size of $L_{\text{hop}} = L_{\text{FFT}}/8$ and a Blackman-Harris window of size $L_{\text{window}} = L_{\text{FFT}}/4$ to extract an user-defined M modes. The modes are computed by detecting the largest peaks in the the frame-averaged magnitude spectrum. Once extracted, these peaks are sorted in pairs (mode frequency in hertz, amplitude) and their amplitudes normalised to a $[0, 1]$ range, where 1 corresponds to the most prominent mode in the analysed sound. Those frequency and amplitude pairs represent the deterministic portion of the signal. The extracted modes are then subtracted from the original sound in the frequency domain, obtaining the residue by computing an IFFT. The residue, which represents the stochastic component of the signal, is stored in an audio file and it is triggered alongside the deterministic component in order to synthesise the final output signal.

4.3 Experiments

To evaluate the synthesis method through a listening study, nine sounds will be recorded from materials that exhibit clear modes when struck. The choice of the materials is constrained to those that are a) likely to be suitable for modelling with modal synthesis (i.e., they exhibit clear frequency-invariant modes when struck); b) relevant to game audio (i.e., they are common materials that could appear in interactive environments).

These materials are:

- Ceramic (2 instances): a plate and a coffee mug.
- Glass (3 instances): an empty bottle, a water glass and a pint glass.
- Metal (3 instances): a flask and two metal lids of different sizes.
- Wood (1 instance): a short wooden rod.

The nine objects are depicted in Figure 4.2. The objects are struck using a metal spoon, and the resulting audio is recorded using a Zoom H6 handheld recorder at ≈15cm from the sound source, using the built-in XY capsules. All sounds are recorded at 44.1 kHz/24 bit and downmixed to mono. The materials are struck multiple times and one representative impact is selected in order to analyse it. 100 modes are then extracted from each of the objects, which are subtracted from the original recording to generate the residue, as described above.

As with the analysis code, the modal synthesizer is also programmed in ChucK [117], using a modified version of a modal synthesizer found in [270]. The sound is synthesised by shaping white noise with an ADSR envelope and

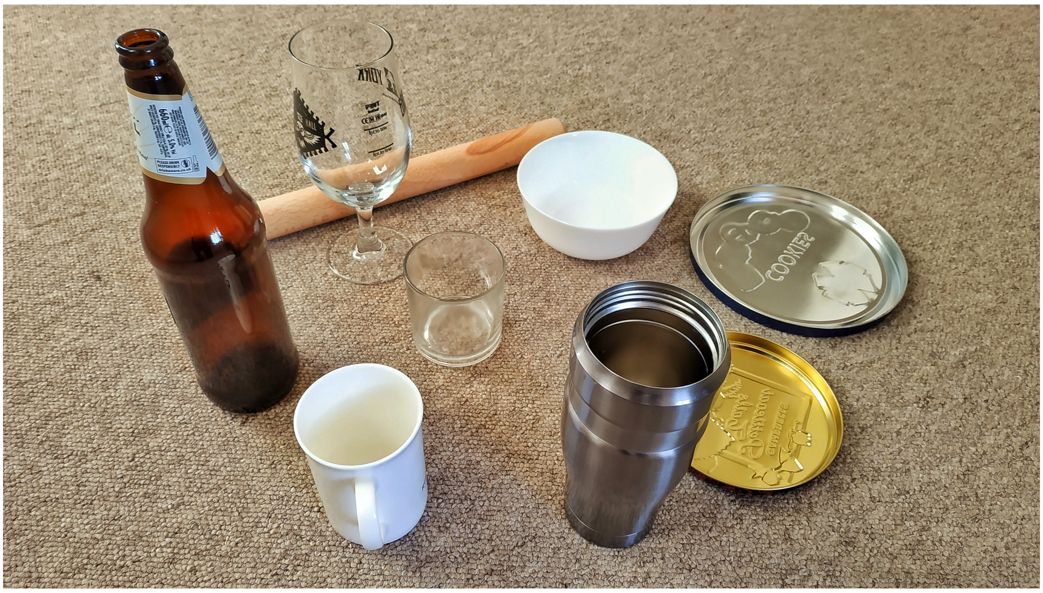


Figure 4.2: Overview of the objects used to record the target sounds, including ceramic, glass metal and wood materials.

filtering it through a series of IIR resonant biquad bandpass filters using the built-in ChucK “ResonZ” class,[1] setting their center frequencies and amplitudes to those resulting from the modal analysis. The result of the filtering operation is summed, in the time-domain, with the extracted residue – which is stored in an audio buffer – producing the final output. The synthesiser responds to a user’s “on-click” events with a new impact sound for each click.

Since the DSP components used to synthesise the sounds are interpretable (i.e., IIR filters and audio buffers), it is possible to randomise the signal each time an output is generated. The aim of this randomization is to create a natural variation when comparing any two hits. This is desirable in the context of game audio, where, as outlined above, it is common to use several audio clips to sound design the same in-game interaction in order to avoid repetition [8, 5]. For each mode, F, the randomisation scheme includes:

[1] `https://chuck.stanford.edu/doc/program/ugen_full.html#ResonZ`

the individual mode frequency F_{freq}, the individual mode amplitude F_{amp}, the filter Q factor F_{Q}; and also the residue playback speed S_{rate} (hence, its pitch), and the gain that scales the deterministic D_{gain} and stochastic S_{gain} components. The level of randomization is different for each of the materials. Thus, for each impact sound, the output pulse code modulation signal y is defined as:

$$y = \left(D_{\mathrm{gain}} \cdot \sum_{m=1}^{M} \tau_m(F_{\mathrm{freq}}, F_{\mathrm{amp}}, F_{\mathrm{Q}}) \right) \; + \; S_{\mathrm{gain}} \cdot \varphi(S_{\mathrm{rate}}) \tag{4.1}$$

where τ_m is the signal resulting from processing white noise with the m-filter, and φ is the residue. This process is depicted in Figure 4.3.

In order to produce the synthesised sounds, a live performance of the modal synthesiser is recorded. Specifically, an audio file, ranging from 3 to 6 seconds in length and comprising a series of consecutive hits for each object, is produced. Since the synthesized versions do not have any reverberation, as they were not recorded in any physical environment (i.e., they are synthesised, and therefore "dry"), and to avoid any potential biases in the perceptual evaluation, an impulse response of the room where the original source sounds were recorded is created. The impulse response is used by convolving it with the synthesised sounds, in order to simulate the acoustic environment of the recorded sounds. An example of the synthesised sounds for each of the different materials is depicted in Figure 4.4. It is worth mentioning that, while the sounds for each of the materials are synthesised using the same modal data, it can be seen that each of the individual impacts in the waveform and spectrogram pairs are slightly different from each other, thanks to the aforementioned randomisation scheme.

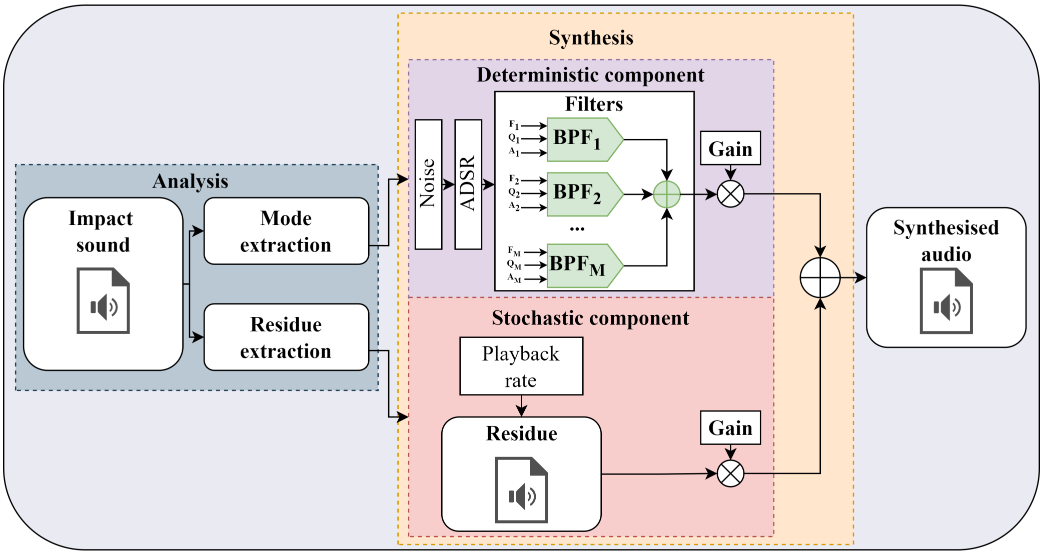


Figure 4.3: Diagram of the modal analysis/synthesis process. An impact sound is analysed to extract its deterministic and stochastic components. During the synthesis stage, the deterministic component is synthesised by shaping a white noise signal with an ADSR envelope, filtering it through M bandpass filters, summing them together, and scaling the output by a gain factor. The stochastic component is built by playing the residue back, setting the playback rate to a variable range, and also scaling the output by a gain factor. Both outputs from the deterministic and stochastic components are summed together to render the final synthesised audio.

The recorded and synthesised sounds as well as the code for the modal synthesiser can be found at the project repository.[2]

4.4 Evaluation

4.4.1 Metrics

In order to evaluate whether the synthesised sounds can be perceived as “synthetic” (i.e., not plausible), a listening study to assess their performance

[2] `https://github.com/adrianbarahona/SMC-Conference-2019_Perceptual-Evaluation-of-Modal-Synthesis-for-Impact-Based-Sounds/`

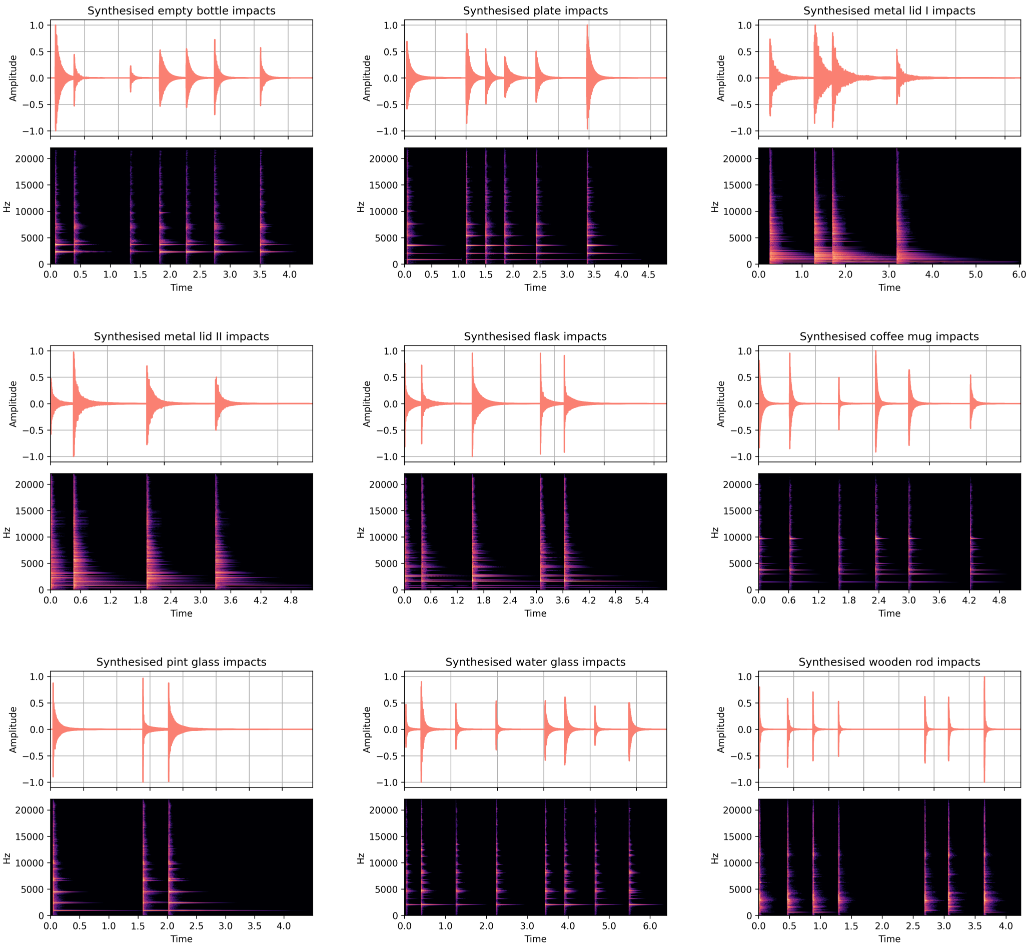


Figure 4.4: Waveform (top) and magnitude spectrogram (bottom) pairs of the synthesised impact sounds for each of the objects (i.e., materials) considered. The individual impacts are generated by triggering the synthesiser and randomising multiple properties of the deterministic and stochastic components of the sound, effectively creating natural variations between the impacts.

is designed. To this end, the RS (real or synthetic) listening test guidelines [271] are employed. The RS listening test describes a series of guidelines to perform subjective evaluation when comparing real recordings of musical instruments and algorithms aimed at reproducing the real instrument behaviour. In other words, it establishes a test and guidelines to inquire whether or not synthesised sounds can be discriminated from recording samples. While the test was originally proposed in the context of the subjective evaluation of musical instruments, there is nothing that prohibits its use in the context of sound effects.

To analyse the results and evaluate the statistical significance of the listening study outcomes, a one-sample parametric t-test on the participants' classification accuracy is conducted. The the participants' accuracy is defined as the percentage of correctly labeled sounds (i.e., recorded sounds labelled as recorded and synthesised sounds labelled as synthetic). The null hypothesis is that there would be no difference between the two systems (i.e., the recorded and synthesised sounds are perceived similarly), thus the participants will perform as a classifier with 50% accuracy, akin to random guessing. The alternative hypothesis is that participants can reliably distinguish between recorded and synthesised sounds. The t-value of the participants' accuracy is computed as follows [272]:

$$t = \frac{\overline{X} - \mu}{\frac{S}{\sqrt{n}}} \tag{4.2}$$

where $\overline{X}$ and S are the mean and standard deviation of the participants' accuracy respectively, μ is the hypothetical mean (50% in this case) and n is the sample size. The calculated t-value is used as a foundation for making inferences through the application of Bayes Factor Analysis (BFA). Bayesian hypothesis testing is widely regarded as superior to the frequentist variety, with the former allowing for finding evidence in favour of the null hypothesis ("no difference between systems") if the data suggest as much [273]. The Bayes factor BF_{10} resulting from the BFA can be interpreted using Table 4.1

Table 4.1: Bayes factor BF_{10} interpretation for hypotheses H_1 and H_0.

Bayes factor BF_{10}	**Interpretation**
>100	Extreme evidence for H_1
30 – 100	Very strong evidence for H_1
10 – 30	Strong evidence for H_1
3 – 10	Moderate evidence for H_1
1 – 3	Anecdotal evidence for H_1
1	No evidence
0.33 – 1	Anecdotal evidence for H_0
0.1 – 0.33	Moderate evidence for H_0
0.03 – 0.1	Strong evidence for H_0
0.01 – 0.03	Very strong evidence for H_0
$<$ 0.01	Extreme evidence for H_0

[274, 275], which summarises how two hypotheses H_1 (e.g., difference between systems) and H_0 (e.g., no difference between systems) compare.

Apart from the BFA, the the RS listening guidelines are also followed, evaluating the listening study results employing two more metrics: the discrimination factor d and the F-measure. In a binary classification problem (e.g,, discriminating between real and synthesised audio), the discrimination factor d is defined as [276]:

$$d = \frac{P_{\text{CS}} - P_{\text{FP}} + 1}{2} \tag{4.3}$$

where $\mathrm{P_{CS}}$ is the ratio in a $0 - 1$ range of correctly detected synthesized sounds, and $\mathrm{P_{FP}}$ the ratio of false positives (recorded samples perceived as synthetic). For d values below 0.75, the two sound groups compared are considered indistinguishable from each other. Values of d around 0.5 are not different from random guessing.

The F-measure, which is favoured over the discrimination factor d by

[271], especially when the number of real and synthetic samples is not equal, is defined as:

$$\text{F-measure} = \frac{(\beta^2 + 1) \cdot \text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}} \tag{4.4}$$

where precision is defined as

$$\text{Precision} = \frac{P_{\text{CS}}}{P_{\text{CS}} + P_{\text{FP}}} \tag{4.5}$$

and recall defined as

$$\text{Recall} = \frac{P_{\text{CS}}}{P_{\text{CS}} + P_{\text{FN}}} \tag{4.6}$$

where again P_{CS} is the number of correctly detected synthesized sounds, P_{FP} the number of false positives (recorded samples perceived as synthetic), P_{FN} the number of false negatives (synthetic sounds labeled as recorded) and $\beta = 1$ is a standard choice. The interpretation of the F-measure values is similar to the d values.

Apart from the evaluation metrics, the RS listening test also proposes a series of guidelines to carry out a successful listening study, called the RS procedure [271]. These guidelines recommend recruiting 20 participants for the listening study with a diverse range of expertise in the instrument being modelled, including experts. Since the present study is not aimed at modelling musical instruments but sound effects instead, the level of expertise in sound design is considered to be equivalent to this criterion. The guidelines also suggest employing a clear user interface to label the sounds, where the user should label each sound as either “recorded” or “synthesised”. To test the attention and reliability of the provided answers, they also recommend to include a clearly synthesised sample (called the "acid test"), in anticipation that any user who is maintaining attention during the test would label it as synthetic. Ideally, the number of real and synthesised stimuli should be equal (i.e., the same number of synthesised and recorded sounds are presented to the users). Finally, the test should not last longer than 15 minutes, where each sound is only heard once before labelling, and the test should be carried out in a controlled environment using headphones.

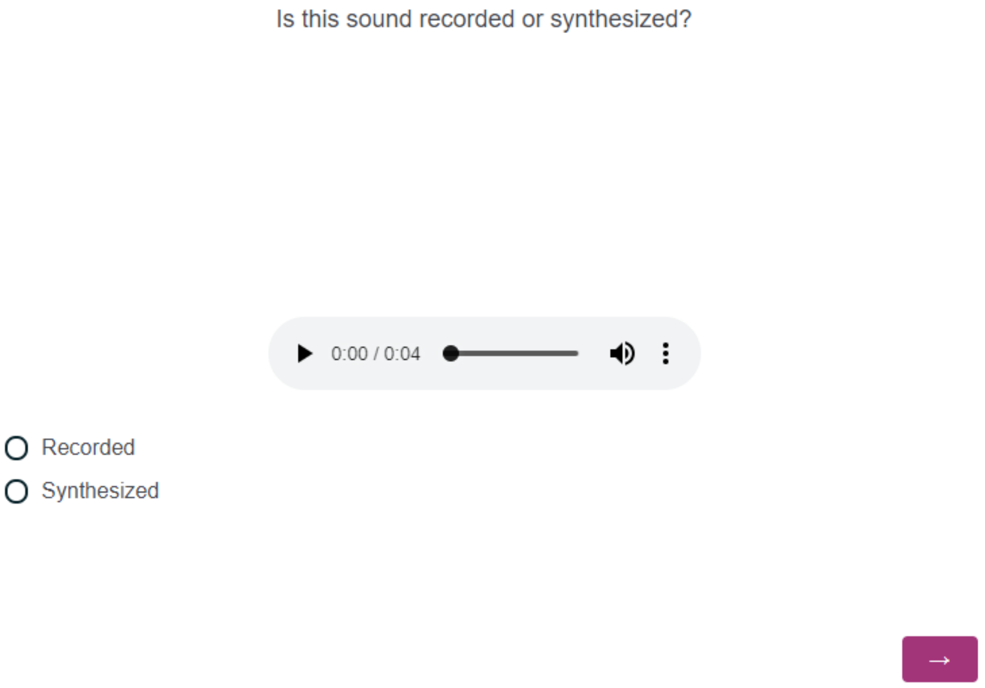
4.4.2 Listening Study

In preparation for the listening study, the original non-synthetic recordings (the sounds used to extract their modal data from) are sliced to generate one audio file for each object comparable in length to the performed synthesised sounds, such as the ones depicted in Figure 4.4. All audio files (recorded an synthesised) were normalised to 0 LU loudness and exported to Ogg Vorbis. The choice of Ogg Vorbis instead of uncompressed WAV was determined by the technical limitations of Qualtrics, the questionnaire platform used.[3] However, this should not drastically affect the outcome of the listening study [277]. No further processing was applied to the audio files.

Contrary to the RS guidelines, in this study the test was not carried out in a controlled listening environment. This is due to the listening study being conducted online with no information of the playback device used by the participants, although they were encouraged to use headphones. This however may help to replicate more closely the playing environment in the context of video games, where the participants are likely to use their own

[3] https://www.qualtrics.com

equipment to play the game or interactive application. The participants are asked to identify, stimuli by stimuli, whether the sound played is recorded or synthesized. The user interface from the study is depicted in Figure 4.5. As suggested by the RS guidelines, a minimal interface that clearly conveys the purpose of the test is designed.



Is this sound recorded or synthesized?

○ Recorded

○ Synthesized

Figure 4.5: Screenshot of the online test interface presented to the participants in the listening study. Participants are asked to listen to the stimuli once and select whether they think the sound is either recorded or synthesised.

Participants classify the samples without being asked to specify the material to which a sample belongs. The participants are presented with the same 18 sounds (9 recorded and 9 synthesized), with their order shuffled, and they are asked to listen to each sound only once. As suggested in the RS test guidelines, an acid test (i.e., a clearly synthesized sound, which in this study is a burst of white noise) is included to act as an attention control. The participants are also asked for their demographic background and their level of expertise in sound design, ranked from 1 (no expertise) to 5 (professional). The online test can be accessed at the online platform.[4] The consent form and the demographic questions that participants were asked can be found in the Questionnaires appendix.

4.4.3 Participants

The participants were recruited both at the University of York (e.g., peers) and online (e.g., audio professionals). A total of 19 participants, 12 males and 6 females (1 participant did not disclose this information) with ages between 18 and 58 and a diverse level of self-reported expertise in sound design participated in the listening study. A breakdown of the participants level of expertise in sound design is depicted in Figure 4.6.

4.4.4 Results

All participants correctly labelled the acid test as synthetic, so no participant's data were discarded at this stage, and the acid test was removed from the analysis of the results. The numerical results of the RS listening test can be found in Table 4.2. A visualisation of these results is depicted in Figure 4.7. It can be clearly seen that both descriptive statistics (the discrimination factor d and the F-measure) are below the 0.75 threshold (no difference between systems) and their means closer to the 0.5 mark, which is equivalent to random guessing. Thus, the listening study results suggest

[4] `https://york.qualtrics.com/jfe/form/SV_0Ipmi7JUFd3FaVT`

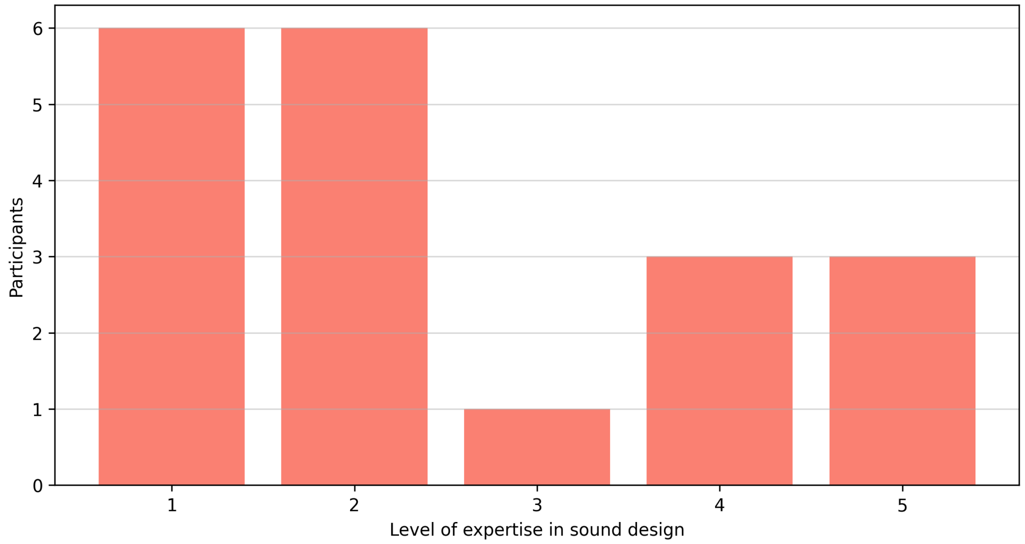


Figure 4.6: Self-reported level of expertise in sound design of the 19 participants in the listening study.

Table 4.2: RS listening test discrimination factor d and F-measure (mean±sd and maximum value) results for all participants.

$\boldsymbol{d}$	Max d	**F-measure**	Max F-measure
0.5 ± 0.12	0.72	0.43 ± 0.16	0.73

that recorded and synthesised sounds are indistinguishable from each other. There is also a relatively low variance among the participants' answers, as can be seen in Figure 4.7.

Regarding the t-test and BFA results, a t-value of $t = 1.911e^{-08}$ is computed using the participants' accuracy scores, $\mu = 50$ and $n = 19$. The BFA found moderate evidence in favour of the null hypothesis ($BF_{10} = 0.237$), supporting the RS test outcome which indicates no difference between systems.

The listening study raw results broken down into the different materials

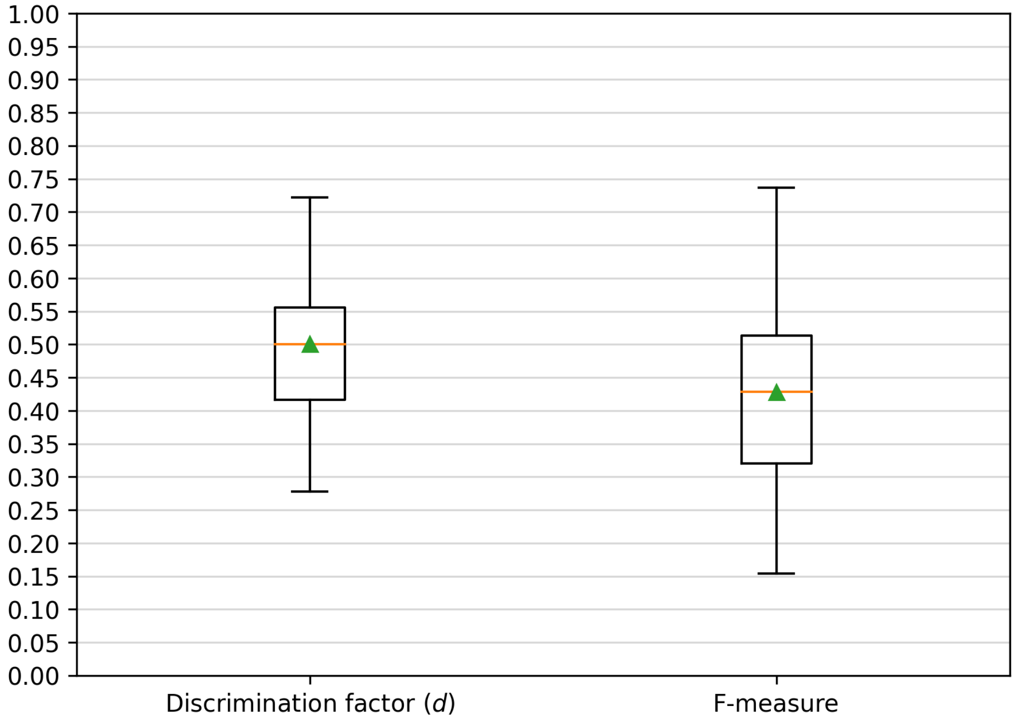


Figure 4.7: RS listening test results for all participants. The discrimination factor d is depicted on the left, and F-measure on the right. For each plot, the mean is annotated by a green triangle, while the median is annotated by a horizontal orange line.

are also provided, represented in Table 4.3. The data suggest that some material categories are harder to distinguish than others. For instance, the synthetic version of the wood rod was successfully identified by 13 of the 19 participants. On the other hand, the synthetic version of the mug was only identified by 4 of 19.

The data also suggest that the level of expertise in sound design was not a decisive factor for identifying synthetic sounds. Participants with an expertise in sound design ranked between 1 and 2 out of 5 obtain $d = 0.48 \pm 0.10$ and $F = 0.42 \pm 0.14$. Participants with the highest level of expertise in sound design, 4 and 5 out of 5, score $d = 0.55 \pm 0.11$ and $F = 0.46 \pm 0.17$.

Table 4.3: RS listening test raw results broken down into the different materials. The left column represents the correctly labelled recorded sounds, while the right column represents the correctly labelled synthesised sounds.

	Material	**Correctly labelled (as recorded)**		**Material**	**Correctly labelled (as synthesised)**
Recorded	*Ceramic*	66%	**Synth**	*Ceramic*	29%
	Glass	54%		*Glass*	37%
	Metal	60%		*Metal*	40%
	Wood	68%		*Wood*	68%

In fact, the participant who obtained the highest d and F-measure (0.72 and 0.73, respectively), has no expertise (i.e., they report 1 out of 5) in sound design.

4.5 Use Case: Interactive Real-Time Procedural Audio Models in a Virtual Environment

Since, as described at the introduction (see Section 4.1), the computational burden of the evaluated system is compatible with real-time scenarios thanks to the DSP components used, and given that the perceptual evaluation is satisfactory for the system, a real-time demo of it is built to highlight its usability in the context of game audio. To this end, the Unity game engine[5] is used in order to design a virtual environment to deploy some of the modal synthesiser configurations (i.e., some materials) employed in this chapter. There are some other procedural audio models included, inspired by the

[5] `https://unity.com/`

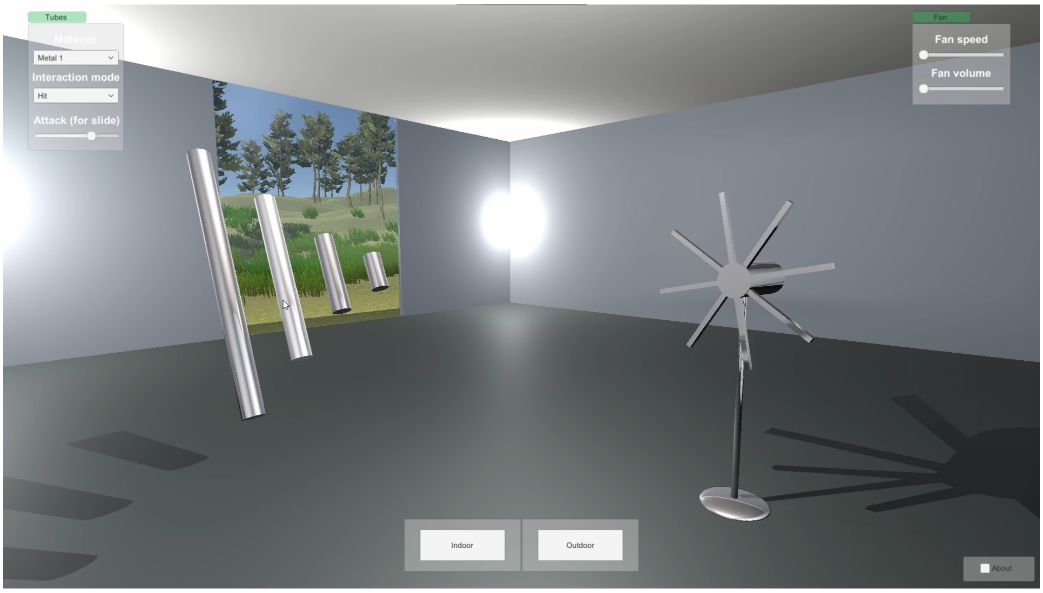


Figure 4.8: Procedural audio demo indoor scene. To the left there are multiple bars whose sound is produced using filter-based modal synthesis. To the right there is a fan whose sound is designed using subtractive and modulation techniques.

design principles proposed by [9].

The procedural audio models are implemented in the Unity engine using the Chunity plugin, which allows for the use of the ChucK programming language within the Unity game engine. The built-in Unity spatialiser is used to render the sound within the game environment. The code for the standalone ChucK code alongside the Chunity integration can be found at the online project repository,[6] and a video of the demo can be found online.[7]

The demo comprises two scenes: an indoor scene and an outdoor scene. The indoor scene, depicted in Figure 4.8, consists of two procedural audio models:

- **Modal bars:** The modal bars, depicted on the left, are built upon the

[6] `https://github.com/adrianbarahona/Procedural_Audio_Chunity`

[7] `https://www.youtube.com/watch?v=1q5gbQq7CUw&`

synthesiser and some materials evaluated in this chapter. Users are allowed to select from 3 different materials in a drop-down menu, allowing them to choose from two types of metal and glass. To take into account the different bar lengths, principles from physically-inspired synthesis are employed, where some physical characteristics of the sound source are embedded into the synthesiser [22, 9]: the frequency of the modes and the playback rate of the residue are adjusted, setting them higher (faster for the playback rate) for smaller bars, and lower (slower for the playback rate) for longer bars, by applying a global frequency modifier to the modal center frequencies and playback rate.

To interact with the bars, users can click on top of them, triggering an ADSR envelope and residue file, and therefore an impact each time the system registers an on-click event. A "slide" interaction is also implemented, where the residue is removed and the attack is slowed to a user-defined range of the ADSR envelope that controls the noise. Finally, to further reduce the computational burden of the system, via the code users can the set the minimum and maximum of number of resonators M_{filters} (i.e., narrow IIR filters) to use, in the range $M_{\text{filters}} = [20, 100]$, in case the demo needs to run on very limited hardware.

- **Fan blade model:** The fan model, depicted on the right, is built by processing a continuous white noise stream through three parallel bandpass filters. The amplitude of the synthesiser is modulated with a sinusoidal oscillator to produce a distinct, "choppy" fan-blade sound. A schematic of the model signal flow is depicted in Figure 4.9. Users can interact with the model by increasing or decreasing the fan speed

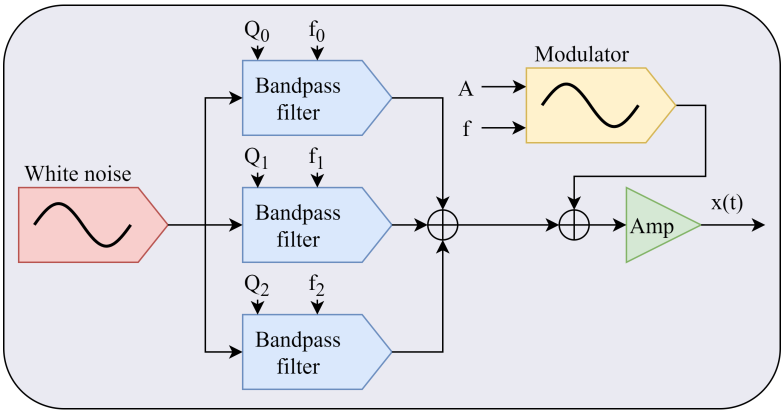


Figure 4.9: Schematic of the fan blade model signal flow. A white noise stream is passed through a series of bandpass filters and their amplitude is modulated by a sinusoidal oscillator.

via a slider. The slider is connected both to the filters and sinusoidal oscillator frequencies, thus increasing their overall pitch and amplitude modulation rate controlled by the user-defined fan speed parameter.

Likewise, the outdoor scene, depicted in Figure 4.10, comprises two different procedural audio models, and creates a weather audio system:

- **Wind model:** For the wind model, a continuous white noise signal is processed through two parallel bandpass filters. One of the filters covers the lower end of the frequency spectrum, synthesising the wind rumble. The other filter covers the higher end of the frequency spectrum, synthesising the sound of the wind against the tree branches. The output amplitudes from both of the filters are gently modulated by a sinusoidal oscillator to create a natural variation to the wind amplitude envelope. Users can interact with the model by using a “wind intensity” slider. The wind intensity affects the frequency and quality Q of the filters and the sinusoidal oscillator frequency.

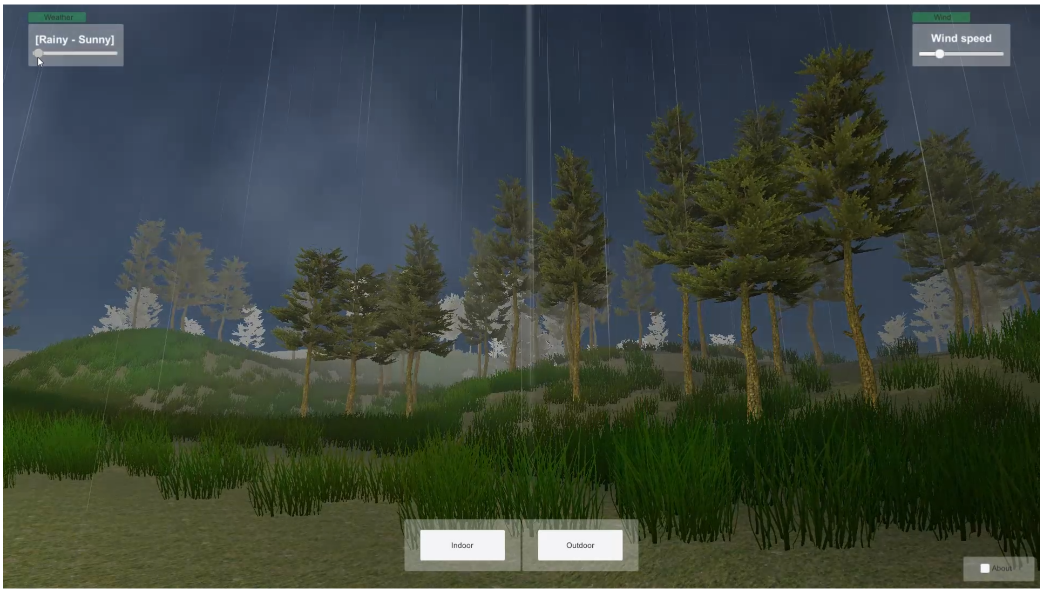


Figure 4.10: Procedural audio demo outdoor scene. The weather system is comprised of two procedural audio models: a modulated subtractive synthesiser for the wind and a subtractive plus sample-based model for the rain sounds.

- **Rain model:** The rain model uses lowpass and highpass filters to shape a continuous white noise instance to create a frequency spectrum comparable to rain sounds. The individual rain drops are generated by triggering a short percussive finger snap recording. Users can control the rain intensity using a slider, which affects the amplitudes of both the output from the filters and the finger snap recording, and the cutoff frequency of the lowpass filter. The intensity slider also controls the trigger rate of the recording, increasing its frequency of occurrence the more intense the rain is (i.e., the density of drops increases with the rain intensity, and vice-versa). Somewhat similar to the wind model, the rain intensity value is modulated by a sinusoidal oscillator to create a non-static amplitude envelope.

4.6 Chapter Summary

The perception of synthesised sound effects, more specifically their perceived plausibility, is an issue that may hamper the adoption of procedural audio [9], and creating plausible procedural audio models may be challenging and time-consuming for sound designers.

This chapter evaluates the perception of filter-based modal synthesis for impact sound effects using a subjective evaluation method inspired by the RS listening test [271], as well as supplementing this with an inferential-statistical test called Bayes factor analysis (BFA). The RS listening test suggests a series of guidelines and metrics aimed at carrying out a listening study where participants are asked to discriminate between real and synthesised sounds. Results show that, for the analysed materials, recorded and synthetic samples are indistinguishable from each other. The BFA also found moderate evidence supporting this outcome. Therefore, this chapter addresses the first research question (RQ 1) of this thesis.

A demo of the modal synthesiser evaluated in this chapter is also presented, which runs in real-time within a video game engine, highlighting its suitability in the context of game audio. The demo also introduces other procedural audio models (a fan blade and a weather system) running also in real-time and inspired by the design principles of [9].

The process to build a procedural audio model using filter-based modal synthesis can be further streamlined. This process consists of three steps:

1. The election of the target sound to be analysed, which can be either recorded on demand or sourced from pre-recorded sound libraries;

2. The analysis of the target sound, extracting its modal information and residue;

3. The synthesiser model, where the end user selects the number of resonators M_{filters} to employ, their randomisation range, and their control scheme.

Those three steps may be unified to build a tool sound designers can use and understand end-to-end, directly transforming a target sound into a deployed procedural audio model. There are, however, some improvements to the modal synthesiser in terms of efficiency. In this chapter, for the listening study, a fixed number of filters of $M_{\text{filters}} = 100$ is set. It is desirable nonetheless to establish an amplitude threshold value where filters whose amplitude fall under this value are discarded, leading to a more compact model (i.e., a model that employs less filters and therefore less computational resources). Another consideration could be the effect of frequency masking, where sounds that are either played in a very busy soundscape or far away from the user could potentially reduce their number of filters, which can be somewhat equivalent of the concept of "level of detail" in computer graphics, where the polygon count or texture quality of 3D models increases as a user approaches them [9]. These two improvements (the optimal number of filters per material and the optimal number of filters as a variable of distance or frequency masking) are left as future work.

Yet another improvement to the modal synthesiser could be the extraction of individual modal decay times. As it can be clearly appreciated in Figure 4.1, each of the modes have a distinct decay time (i.e., the individual frequencies "fade out" at a different rate). However, the evaluated modal synthesiser does not take this factor into account. This effect is compensated for somewhat by the mode amplitudes (i.e., modes with a higher amplitude will decay slower, as can be seen in Figure 4.4). While the perceptual evaluation suggests the synthesised sounds are indistinguishable from recorded samples in isolation, the individual mode decay time may be desirable in situations where the aim is to resynthesise the target sound with precision. This may be especially relevant to materials that performed comparatively worse, such as wood (see Table 4.3).

Regarding the evaluation, a different approach could have been to compare recorded and synthesised sounds side by side, ranking them in terms of perceived plausibility. However, in this chapter it is opted to compare them independently – the experimentees are shown a single sound at a time, and it is evaluated whether or not that sound, in isolation, is perceived as synthetic. The motivation of taking this approach relies in the idea that if pre-recorded and synthesised impact sound effects are indistinguishable from each other in isolation, the synthesised sounds can be used without perceptual loss of authenticity and obtaining all the benefits procedural audio presents. While the synthesised version derived from analysing a sound may or may not be an exact reconstruction of the original pre-recorded sample, it is remarkable that it still could be employed to sound design plausible sonic interactions, assuming the perceptual evaluation is favorable for the synthesised sounds.

Finally, while the listening study outcome suggests that the use of filter-based modal synthesis is a perceptually effective technique in this context, this is evaluated only for a subset of sounds (i.e., impacts) with certain characteristics (i.e., objects that exhibit clear modes when they are struck), and for time-invariant filters (i.e., the filters do not change their frequency bands dynamically).

Despite this technique having the potential of being relatively easy to adopt by sound designers for creating bespoke procedural audio models, other sounds or types of interactions (e.g., 1 : 1 interactions using haptic controllers) may need different techniques, which could be time-consuming to craft, or their result unsatisfactory from a perceptual standpoint. Moreover, the rest of the procedural audio models described in Section 4.5 are built using the design principles in [9], falling under the "piecewise" or "brute force" synthesis category (see Section 2.2.1.2), where multiple DSP components are combined until the resulting sound is perceptually satisfactory, thus relying on the skill of the sound designer. Additionally, it has been shown that, in most cases and depending on the technique used, sound effects synthesised using DSP may still lack plausibility compared to pre-recorded samples [104, 17, 18].

Alternatively, machine learning (which, as described in Chapter 3, is rooted in the concept of training computational models via examples [119]) may be an attractive solution to the problem of sound synthesis. As outlined before, recent breakthroughs in deep learning such as the emergence of WaveNet [19] (see Chapters 1 and 3) closed the gap with human performance by 50%, outperforming the previous DSP-based state of the art for English text-to-speech. Thus, to address the above issues and motivated by these recent advancements in generative deep learning, Chapters 5, 6, and 7 of this thesis explore data-driven methods capable of potentially synthesising *arbitrary* sounds, and opening up the possibility of novel interactions and sound transformations.

CHAPTER 5

CLASS-CONDITIONAL NEURAL AUDIO SYNTHESIS OF SOUND EFFECTS

5.1 Introduction

As described in Chapter 3, from a high-level perspective, generative deep learning techniques are capable of creating new data in the style of the data on which they are trained. Generative adversarial networks (GANs) [155] have performed well on tasks in the field of computer vision (e.g., [160, 161, 162, 163]), and they have been also applied to audio, with architectures such as WaveGAN [200], which models waveforms directly, or GANSynth [201], which uses an intermediate spectral representation. While GANSynth [201] could be potentially used to generate sound effects, the architecture focuses on modelling harmonic musical notes, which differs from the nature of most sound effects as they do not necessarily exhibit an harmonic spectrum. On the other hand, in [278] they use a WaveGAN architecture [200] to synthesise drum sounds within a virtual environment, a case that is closely related to the synthesis of sound effects for video games or interactive applications. Thus, taking inspiration from [278], this chapter employs a WaveGAN architecture for the class-conditional synthesis of sound effects, as defined on p. 8.

More specifically, the potential user controllability over the synthesised audio is refined by using a conditional WaveGAN architecture [223]. Unlike the original WaveGAN architecture [200], which produces signals without any control from users, conditional architectures – such as conditional GANs [279] – incorporate a label (or a series of labels) that allow for the global latent conditioning of the model. For instance, if an unconditional model is trained on a corpus of knocking sound effects with different emotional intentions, it will synthesise a sound with a random emotional intention each time the generator is called. However, if an emotional intention label is added to the model, the model can be prompted to synthesise a knocking sound effect using any of the dataset's emotional intentions (i.e., any of the conditioning labels), with the user specifying it at the time of generation. This allows the user to achieve finer control over the synthesis without the need to rely on multiple parallel models – each trained on a different class of the dataset.

In this chapter, the focus is set on the synthesis of knocking sound effects with emotional intention. Knocking sound effects, which are are usually composed from one or more impact-sounds against a surface, are a key element in storytelling as they are often used as a transition element. For example, in films or video games, a knocking action can express the emotions of the person at the door, as well as create expectations in the audience or player about the possible reactions of a player or character hearing the knock. Knocking sound effects are an interesting subject of study for sound synthesis since they have a distinctive frequency-domain component (i.e., the individual knock synthesis derived from the material of the door, the force of the impact, etc.) as well as a highly articulated time-domain component (i.e., the arrangement and relationship of the individual knocks in time, forming a knocking action). While impact sounds such as knocking sound effects could be synthesised using modal synthesis [22] as shown in the previous Chapter 4, novel architectures in machine learning, and more specifically deep learning, offer the possibility of synthesising sounds using alternative methods, introducing control schemes such as label conditioning (e.g., emotional intention) as previously outlined. Moreover, unlike Chapter 4, which is focused on one-shot impacts, here complete knocking actions are modelled, learning the emotional intention from the data itself and also not necessarily limiting the synthesiser to a particular subset of sounds (i.e., sounds that exhibit clear modes as in Chapter 4). Thus, the modeling here aims not only to generate plausible knocking sound effects but also to create knocking actions that resemble the emotional intention behind the person knocking on the door, going further than Chapter 4.

5.2 The Knocking Sound Effects Dataset

Generative adversarial networks, such as WaveGAN [200], need large amounts of data to train, and can potentially increase their performance the more data they are trained on [159]. Since knocking sound effects performed with emotional intention are not a common category of sounds, and in order to synthesise high-quality sound effects that convey the intended emotions, a Foley artist is commissioned to create a dataset. Specifically, Ulf Olausson is asked to record a dataset of knocking sound effects with emotional intention at the FoleyWorks studios in Stockholm.[1] Inspired by previous work on knocking sounds [280], five basic emotions [281] are chosen to be portrayed in the dataset: anger, fear, happiness, neutral and sadness.

In order to portray a plausible rendition of the emotions, the Foley artist is given the following scenarios to perform the knocking actions:

- **Anger**: telling a flatmate for the 4th time to turn down the very loud music.
- **Fear**: alerting a neighbour of a possible risk.
- **Happiness**: telling a flatmate that they won a prize.
- **Neutral**: a parcel delivery.
- **Sadness**: telling a friend that someone passed away.

The Foley artist is also asked to perform diverse interpretations of the provided scenarios in order to produce a wider variety of sounds. The dataset was recorded with a Rode NT1 microphone, performing the knocks to a closed wooden door inside the Foley studio facilities, as depicted in Figure 5.1.

A total of 600 knocking actions (120 actions per emotion) are recorded. Here, “knocking action” refers to a sequence of individual knocks that convey a certain emotion. 20 actions per emotion were discarded to filter out undesirable noise or artifacts. The final 500 audio files are trimmed so each action starts on the first knock onset and finish on the last knock decay. The dataset can be accessed at the online repository [282].[2]

[1] `http://www.foleyworks.se/`

[2] `https://doi.org/10.5281/zenodo.3668503`

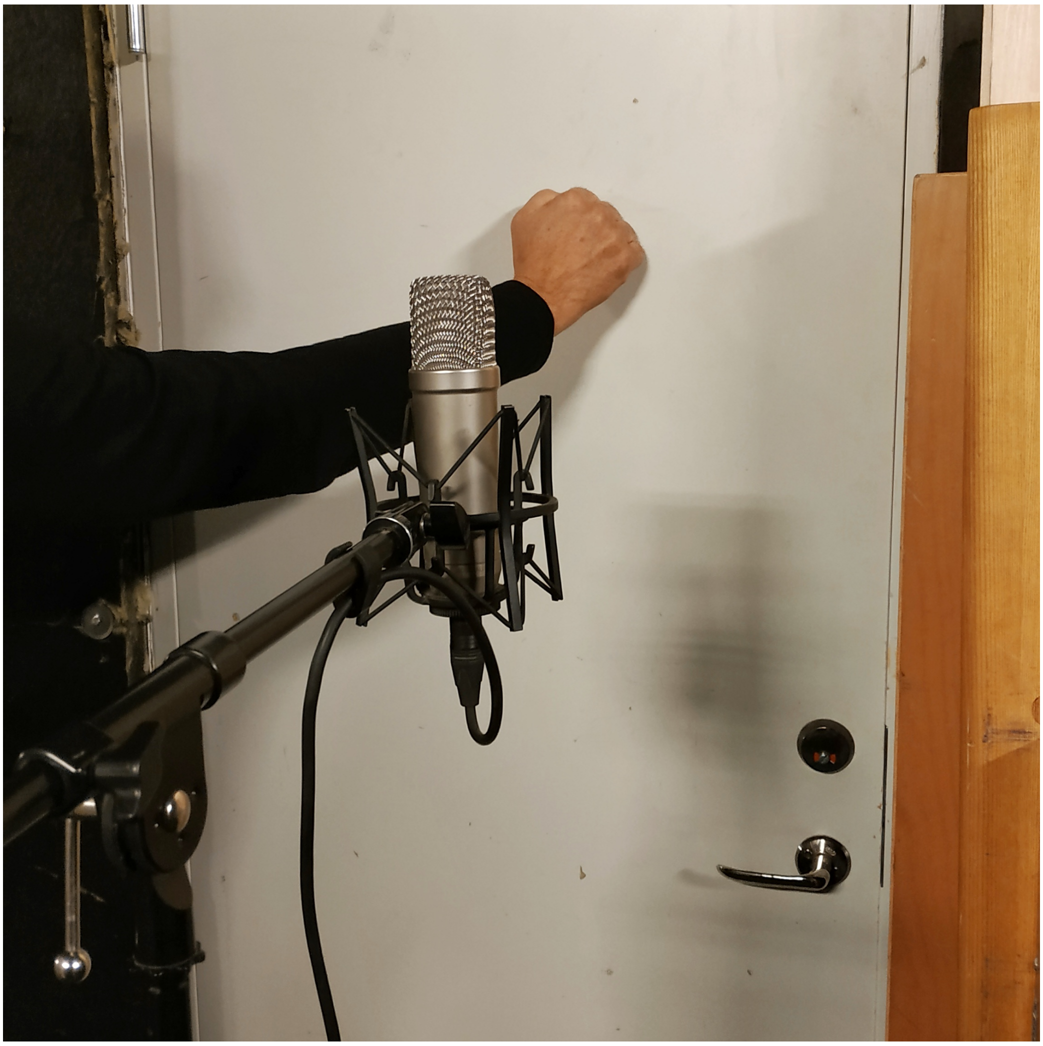


Figure 5.1: Microphone placement during the knocking sound effect dataset recording. The knocks are performed to a closed wooden door.

5.3 Method

The proposed method consists of using a conditional WaveGAN architecture [200, 223] conditioned on the emotions of knocking sound effects to produce raw audio (i.e., the architecture outputs audio samples directly, without using any intermediate representation such as spectrograms), learning to synthesise those sound effects alongside their intended emotion from the training

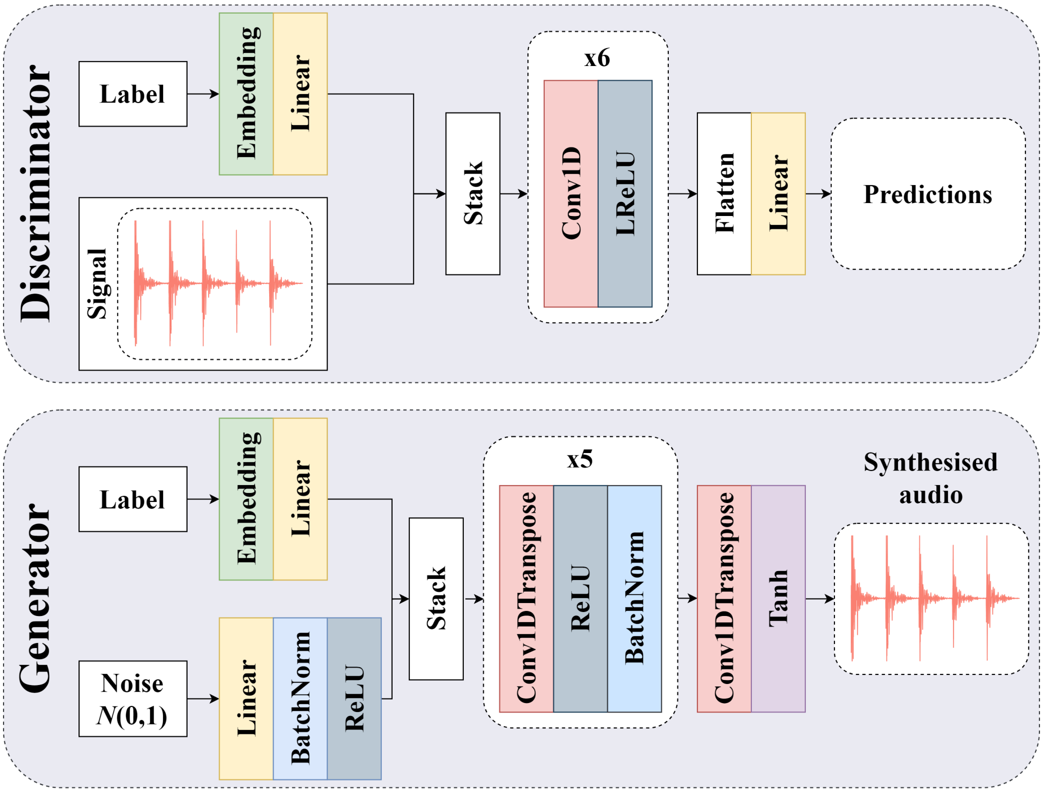


Figure 5.2: Conditional WaveGAN architecture. The discriminator – or critic in this particular case –, depicted on top, takes a signal and a conditioning label and processes them through a series of strided convolutions to make predictions of the “realness” or “fakeness” of the input data. The generator, depicted at the bottom, takes a noise vector from a normal Gaussian distribution $\mathcal{N}(0,1)$ alongside a label, and processes them through a series of strided transposed convolutions to produce a signal.

dataset described in Section 5.2. Once trained, the model is able to synthesise knocking sound effects with emotional intention by simply sampling from the latent space and specifying the target emotion at the time of generation.

5.3.1 Architecture

The conditional WaveGAN architecture used, depicted in Figure 5.2 is comprised of two neural networks: the discriminator and the generator.

The discriminator takes two inputs: a signal of length L_{signal}, which in this case it is a real or synthesised knocking sound effect; and a conditioning label, which in the particular case of this study is an integer number in a $[0, 4]$ range, representing each one of the possible values one of the emotions in the dataset. The label is processed through an embedding layer with a hidden size of $20 \cdot N_{\text{label}}$, where N_{label} is the number of labels in the dataset (5 in this case, thus $20 \cdot N_{\text{label}} = 100$). Its output is processed through a linear layer with a hidden size of L_{signal} immediately after. The processed label is then concatenated along its channel axis with the unaltered input signal. The concatenated tensor (i.e., the embedding layer concatenated with the input signal) is processed through 6 blocks of one-dimensional convolutions, followed by Leaky ReLU activations. Each one-dimensional convolutional layer has a kernel size of 25, stride 4 and $64, 128, 256, 512, 1024$ and 2048 filters respectively; and a $\alpha = 0.2$ for all Leaky ReLU activations. The output from the last convolutional layer is flattened and processed through a linear layer comprised of a single neuron to make the discriminator predictions. through

Similarly, the generator also receives two inputs: a 100-dimensional latent vector z comprised of values drawn from a normal Gaussian distribution $z \sim \mathcal{N}(0, 1)$; and a global conditioning label similar to the conditioning label in the discriminator. The latent vector z is processed through a linear layer with a hidden size of 2^{15}, followed by a batch normalization operation and a ReLU activation. The label is processed exactly as in the discriminator. The output from processing the latent vector z and the label are concatenated along their channel axis, and processed through 5 blocks of transposed one-dimensional convolutional layers, followed by ReLU activations and batch normalisation operations. The transposed one-dimensional convolutional layers have a kernel size of 25, stride 4 and $1024, 512, 256, 128$ and 64 filters respectively. The result is processed by a last transposed one-dimensional convolutional layer with kernel size 25, stride 4 and a single filter. Finally, the output from this layer is scaled to a $[-1, 1]$ range with a hyperbolic tangent (Tanh) activation in order to produce the audio signal.

While most of the architectural considerations proposed in the WaveGAN paper [200] are followed, it was observed in pilot experiments that the proposed phase shuffle operation worsened the output of the model with this particular dataset, thus it was discarded. The Keras implementation of the conditional WaveGAN architecture used in this study is available at the online repository.[3]

5.3.2 Training and Inference

During training, for every training iteration, the initial step involves optimising the discriminator. To this end, a batch of knocking sound effects is synthesised using the generator and passed to the discriminator alongside the labels used to generate the audio. The logits (i.e., the predictions) of the discriminator are computed for this fake data. Then, real recordings from the dataset are passed alongside their corresponding emotion label to the discriminator and its predictions are registered as well. To assess the discriminator performance a WGAN-GP loss function [157] is used. Thus, the discriminator (which is also called *critic* when using Wasserstein distance

[3] https://www.github.com/adrianbarahona/conditional_wavegan_knocking_sounds

[156], as in this case) is incentivised to correctly label the "realness" and "fakeness" of real and synthetic data respectively (i.e., to output values that separate their distributions as far apart as possible) by its objective function. As suggested in [155], the discriminator is trained for D_{steps} steps per training step.

The generator is trained after the discriminator, during the same training step. To do so, a batch of synthesised sound effects is generated and passed – alongside the labels used to generate them – to the discriminator in order to obtain its predictions. The generator is incentivised to "fool" the discriminator by its objective function, aiming to obtain predictions from the discriminator that are as close as possible to the real data (i.e., reducing the Wasserstein distance between the two distributions). This process (i.e., training the discriminator and the generator) is repeated for N_{epochs} epochs.

Both the generator and the discriminator process data in chunks of size L_{signal}. Thus, the input of the discriminator and the signal produced by the generator are fixed and cannot be extended in time or altered. In practice, this constraint implies that the length of the generated audio must fall within the range of $[0, L_{\text{signal}}]$. For shorter sounds (i.e., sounds with a length of $L < L_{\text{signal}}$), the generator will ideally produce silence from the end of the sound to the end of the generated signal of length L_{signal}.

During inference, the discriminator can be discarded as it fulfilled its purpose of training the generator, which is ready to be used as a neural audio synthesiser. To synthesise a knocking sound effect of length L_{signal}, a 100-dimensional Gaussian noise latent vector $z \in \mathbb{R}^{100}$ is passed to the generator, where $z \sim \mathcal{N}(0, 1)$, alongside an integer in $[0, 4]$, representing each of the 5 emotions, to obtain the signal.

5.4 Experiments

To later assess the performance of the proposed conditional WaveGAN architecture, it is trained using the dataset introduced in Section 5.2. Considering the longer knocking action in the dataset is of 125 685 samples at 48 kHz, accommodating such a duration would require a very large architecture configuration. Since the proposed configuration (see Section 5.3.1) is able to process up to 65 536 samples, the sampling rate is halved to fit all the possible training examples in the architecture, thus processing audio at 22 050 Hz. At this sampling rate, the longest sound in the dataset is of 62 843 samples. Thus, all sounds can fit in the architecture.

As described in Section 5.3.1, most WaveGAN architectural considerations (e.g., kernel size or stride in convolutional layers) and hyperparameters are used. However, during pilot experiments, it was found that using a learning rate of 0.0002 (instead of 0.0001) for both the discriminator and generator, and a batch size of 128 (instead of 64) helped the network converge faster. The network is trained for $N_{\text{epochs}} = 600\text{K}$ epochs, employing an Adam optimiser, and using a WGAN-GP loss as described above. The discriminator training steps are set to $D_{\text{steps}} = 5$, as in the original WaveGAN paper [200]. On a NVIDIA Tesla V100, the training process takes ≈72 hours. Once trained, the generator model has a total of 73M parameters and a size of ≈286 MB on disk.

During inference, the time required to synthesise a single batch signal with an output length of 65 536 samples (around 3 seconds of audio at a

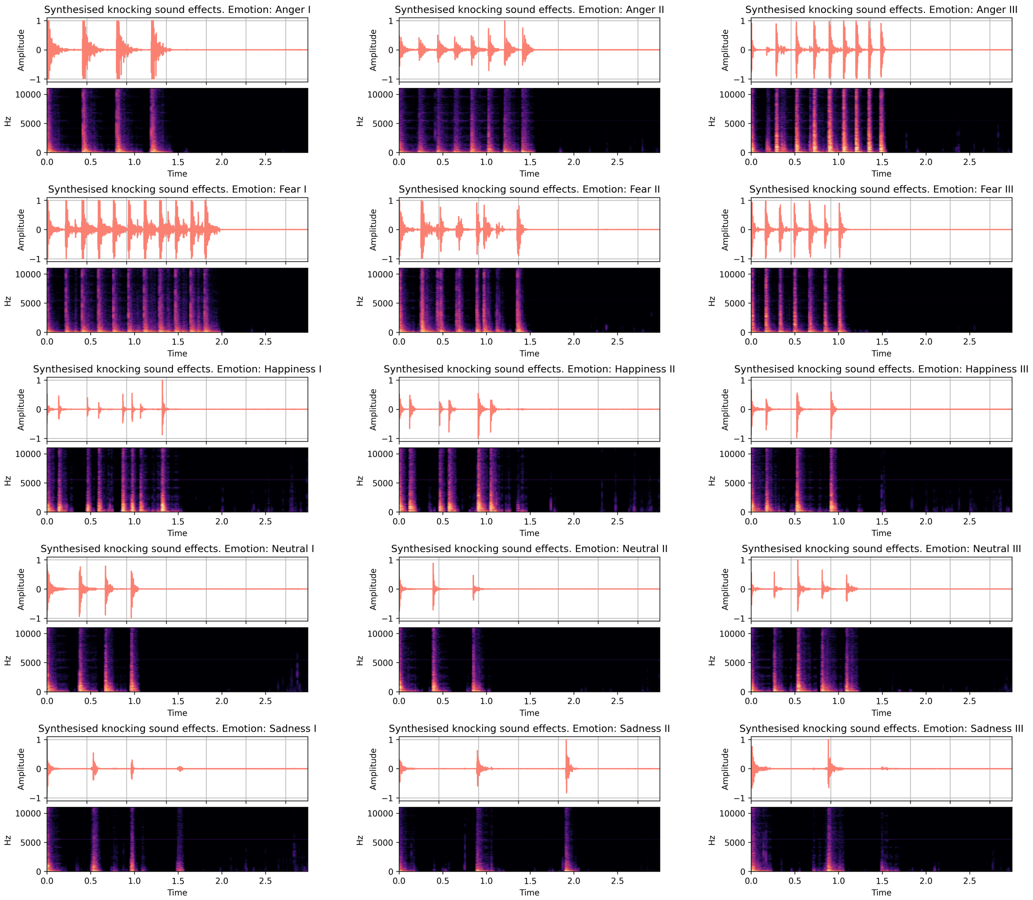


Figure 5.3: Waveform (top) and magnitude spectrogram (bottom) pairs of synthesised knocking sound effects with different emotions. Each column represents a different instance of a synthesised sound effect with a particular emotion. Each row contains contains the sound synthesised per emotion: anger, fear, happiness, neutral and sadness respectively.

22 050 Hz sampling rate) is 66.9 ± 5.9 (mean ± sd) milliseconds on a consumer GPU (NVIDIA GTX 1060), measured on a 100-run test. Alternatively, the time required to synthesise a 100-batch signal with an output length of 65 536 samples each (around 3 seconds of audio each at a 22 050 Hz sampling rate, or ≈5 minutes of audio if combined) is of 726.5 ± 12.5 (mean ± sd) milliseconds on a consumer GPU (NVIDIA GTX 1060), measured also on a

100-run test.

Examples of the synthesised sounds per emotion are depicted in Figure 5.3. Sound examples of both the recorded dataset and the synthesised sounds can also be found at project website.[4]

5.5 Evaluation

To evaluate the performance of the proposed method we: 1) design a listening study aimed to assess both whether or not the synthesised sounds are distinguishable from recorded samples and to understand how the intended emotions are perceived; 2) extract and compare a series of acoustic features from the knocking sound effects introduced in Section 5.2 and the synthesised knocking sound effects produced by the model.

To this end, to compare the recorded and synthesised sounds in equal number, a total of 500 knocking sound effects (100 sounds effects per emotion) are synthesised.

5.5.1 Subjective Evaluation

A listening study is designed to assess the plausibility of the sounds synthesised by the model, and to understand how listeners interpret the intended and perceived emotions from both the recorded and synthesized sounds.

As in Chapter 4, to learn how the synthesised sounds are perceptually perceived compared to the recorded dataset in terms of plausibility, the listening study is designed using the RS (real and synthetic) guidelines [283].

[4] https://www.adrianbarahonarios.com/conditional_wavegan_knocking_sounds/

The listening test results are evaluated using the two metrics suggested by the RS listening guidelines: the discrimination factor d and the F-measure. Regarding the interpretation of both metrics, considering a $[0, 1]$ scale, values up to 0.5 are no different from random guessing, values between 0.5 and the 0.75 mark show no clear distinction between the recorded and synthesised sounds, and values closer to 1 show the recorded and synthetic sounds are clearly distinguishable from each other. Likewise, as in Chapter 4, the statistical significance of the listening study outcome is evaluated by performing a one-sample parametric t-test on the participants' classification accuracy (i.e., the percentage of recorded and synthesised sounds labelled correctly). The computed t-value is used to perform a Bayes factor analysis (BFA), where the null hypothesis is that participants would not be able to differentiate between the recorded and the synthesised sounds (i.e., there would be no difference between systems), thus they would have a classification accuracy of 50%, similar to random guessing. The alternative hypothesis is the opposite (i.e., participants can distinguish between recorded and synthesised sounds). The Bayes factor BF_{10} resulting from the BFA can be interpreted using Table 4.1 on p. 75.

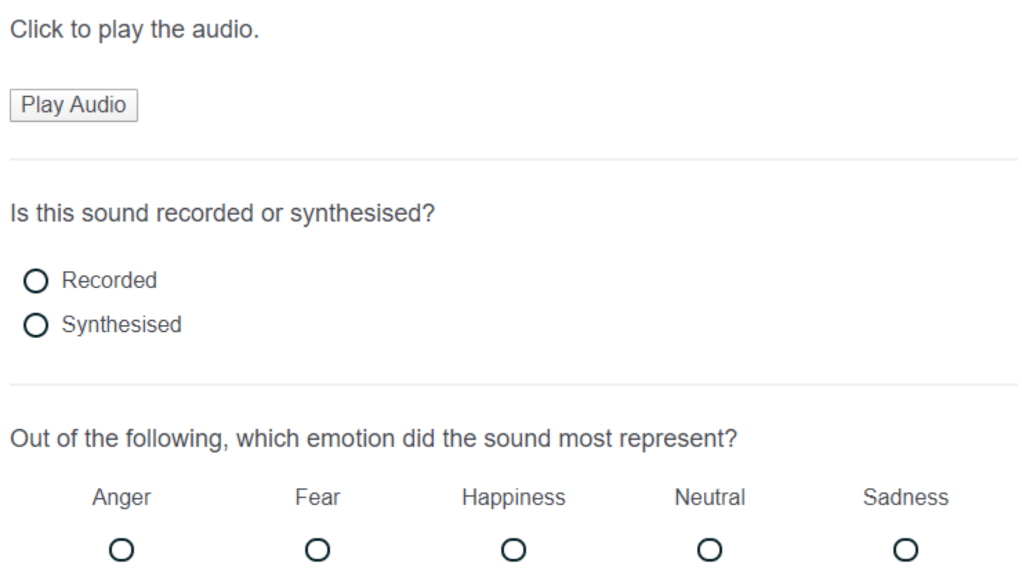
Following the RS test guidelines [283], people with a diverse set of expertise in the topic at hand (sound design, in this case) are recruited. A clear interface is designed to ask the participants whether they consider a sound is either recorded or synthesised, and they are asked to listen to each sound only once. A clearly synthesised sound (called "acid test") is also included to ensure that participants are paying attention. The listening test is carried out online. The participants are encouraged to use headphones, which could be comparable to the scenario of a person playing video games with their own equipment.

Additionally, to understand how the emotions are perceived in both the recorded and synthesised audio, the participants are asked to label the emotion each of the stimuli represent the most, giving them the 5 considered emotions (anger, fear, happiness, neutral and sadness) to choose from. The participants are also asked to introduce their demographic information, and to report their level of expertise in sound design ranked from 1 (no expertise) to 5 (professional). A screenshot of the listening test interface used in the listening study can be seen in Figure 5.4. The online listening study, hosted in the Qualtrics platform, can be accessed online,[5] and the consent form and the demographic questions that participants were asked can be found in the Questionnaires appendix.

Each participant is presented with a total of 51 sounds: 25 sounds are recorded (5 per emotion), 25 sounds are synthesised (5 per emotion) and 1 sound is the “acid test”. The selection of sounds (i.e., the 50 sounds out of the 1000 from both the recorded and synthesised corpus displayed to each participant), and the order in which they are presented, are randomised for each participant. The sounds are presented in their original sampling rate: 48 000 Hz WAV for the recorded sounds, and 22 050 Hz WAV for the synthesised sounds. While this discrepancy in sampling rates between the recorded and synthesised sounds may affect the perceptual evaluation, the original sampling rates are maintained since they can better represent the performance of the proposed model against pre-recorded sounds.

A total of 22 participants (17 males, 5 females) with ages from 18 to

[5] https://york.qualtrics.com/jfe/form/SV_9HTTm9ggUIvt3Ct



Click to play the audio.

Is this sound recorded or synthesised?

○ Recorded
○ Synthesised

Out of the following, which emotion did the sound most represent?

Anger	Fear	Happiness	Neutral	Sadness
○	○	○	○	○

Figure 5.4: Screenshot of the online test interface presented to the participants in the listening study. Participants are asked to listen to the stimuli once and select whether they think the sound is either recorded or synthesised, and to choose which of the 5 considered emotions the stimuli represent the most.

58 participate in the listening study. A breakdown of the participants self-reported level of expertise in sound design is depicted in Figure 5.5. One participant failed to identify the "acid test" correctly, and therefore was removed from the evaluation completely. Another participant did not label the emotion of one stimuli, and therefore was removed from the evaluation of the emotion labeling.

The results from the RS listening test are displayed in Table 5.1. The raw percentage of participants who correctly labeled the recorded and synthesised sounds can be seen in Table 5.2. A visualisation of the RS test results with the individual participant scores broken down by the level of the participants level of expertise in sound design is shown in Figure 5.6. While the RS listening

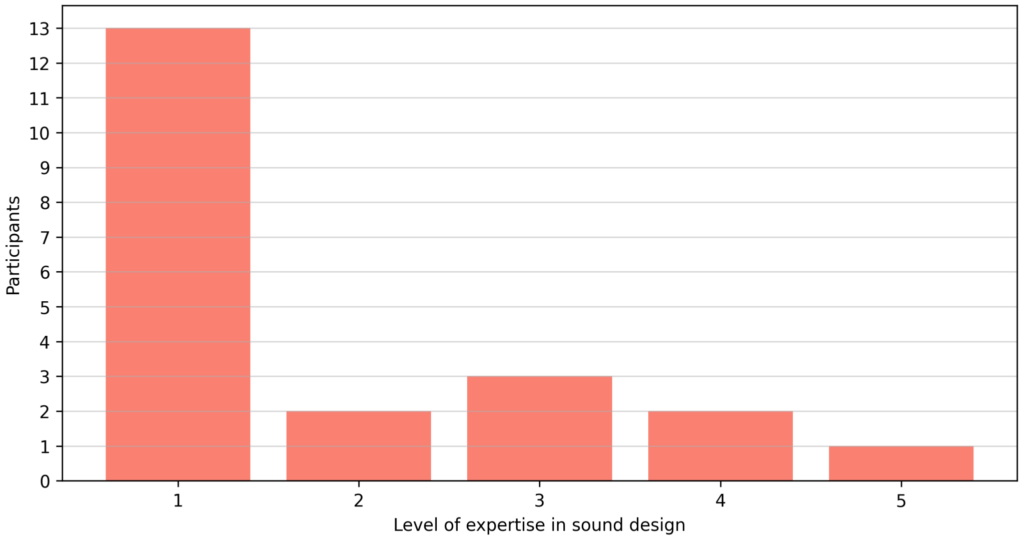


Figure 5.5: Self-reported level of expertise in sound design of the 21 participants in the listening study.

test results are on average below the 0.75 threshold, synthesised samples can be identified almost consistently by participants with medium to high expertise in sound design, as depicted in Figure 5.6 in blue. However, most of the participants with low expertise in sound design fail to discriminate between the recorded and synthesised sound effects.

Regarding the t-test and the BFA, a t-value of $t = 3.207$ is computed using all participants' accuracy scores, $\mu = 50$ and $n = 21$. The BFA found strong evidence in favour of the alternative hypothesis ($BF_{10} = 10.05$), suggesting that, in general, participants can discriminate between the recorded and the synthesised sounds. Breaking down the listening study results based on the participants' expertise in sound design, similar to the depiction in Figure 5.6, it is found that this factor has an impact to the perceived plausibility of the sounds. Specifically, participants with medium to high expertise in sound design (i.e., ≥ 3 out of 5) obtain a t-value of $t = 10.025$ using their accuracy

Table 5.1: RS listening test discrimination factor d and F-measure (mean ± sd) results for all participants.

d	F-measure
0.63 ± 0.31	0.67 ± 0.24

Table 5.2: RS listening test raw results showing the percentage of participants correctly labelling both the recorded sounds (as recorded) and the synthesised sounds (as synthesised).

	Emotion	Correctly labelled (as recorded)		Emotion	Correctly labelled (as synthesised)
Recorded	*Anger*	73.3%	Synthesised	*Anger*	60.9%
	Fear	72.4%		*Fear*	60.0%
	Happiness	80.9%		*Happiness*	63.4%
	Neutral	68.6%		*Neutral*	65.7%
	Sadness	60.9%		*Sadness*	65.7%

scores, $\mu = 50$ and $n = 6$. The BFA found extreme evidence ($BF_{10} > 100$) in favour of the alternative hypothesis, reinforcing the RS test outcome, which indicates that participants with medium to high expertise in sound design are able to discriminate between recorded and synthesized sounds. On the other hand, participants with low expertise in sound design (i.e., ≤ 2 out of 5) obtain a t-value of $t = -1.023$ using their accuracy scores, $\mu = 50$ and $n = 15$. The BFA found anecdotal evidence ($BF_{10} = 0.41$) in favour of the null hypothesis (i.e., the model is no far from "fooling" non-experts).

The raw percentage of the participant emotion labeling for the recorded and the synthesised stimuli are shown in Table 5.3 and Table 5.4 respectively.

To assess the statistical significance of how the emotions are perceived in both the recorded dataset and in the synthesised sounds, a Chi-squared test on the emotion labeling by the participants is performed.

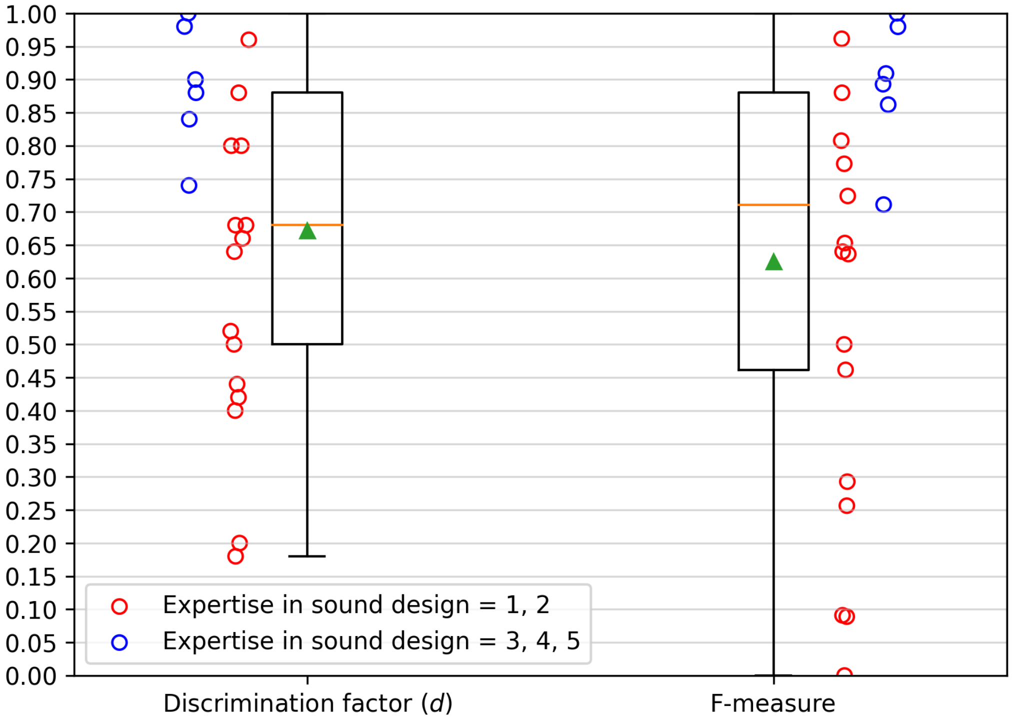


Figure 5.6: RS listening test discrimination factor d and F-measure values for all participants. The left box plot represents the discrimination factor d, with the individual score per participant represented by a circle, where the red and blue colours indicate low and medium to high expertise in sound design respectively. The right plot represents the F-measure, with the individual participant scores annotated to its right. For each box plot the mean is annotated by a green triangle, whereas the median is annotated by a horizontal orange line.

For the recorded stimuli the Chi-squared is performed with $\chi^2(16, N = 500) = 564.341, p = .000 < .05$ (columns compared with a z-test and p values adjusted with Bonferroni method). Results show that anger, happiness, neutral and sadness are statistically different from others. Anger and fear are not statistically different from each other, but they are statistically different from the other emotions. This indicates that fear is confused with anger.

For the synthesised stimuli the Chi-squared test is performed with $\chi^2(16, N =$

Table 5.3: Recorded stimuli percentage of intended and perceived emotion labeling by the participants. The most perceived emotion per intended emotion is highlighted in bold.

				Perceived		
		Anger	***Fear***	***Happiness***	***Neutral***	***Sadness***
Intended	***Anger***	**68.0%**	12.0%	10.0%	10.0%	0.0%
	Fear	**60.0%**	28.0%	5.0%	5.0%	2.0%
	Happiness	4.0%	2.0%	**64.0%**	27.0%	3.0%
	Neutral	5.0%	6.0%	15.0%	**63.0%**	11.0%
	Sadness	2.0%	10.0%	2.0%	29.0%	**57.0%**

Table 5.4: Synthesised stimuli percentage of intended and perceived emotion labeling by the participants. The most perceived emotion per intended emotion is highlighted in bold.

				Perceived		
		Anger	***Fear***	***Happiness***	***Neutral***	***Sadness***
Intended	***Anger***	**47.0%**	19.0%	8.0%	18.0%	8.0%
	Fear	40.0%	**44.0%**	9.0%	7.0%	0.0%
	Happiness	1.0%	12.0%	**44.0%**	37.0%	6.0%
	Neutral	6.0%	6.0%	17.0%	**62.0%**	9.0%
	Sadness	0.0%	16.0%	3.0%	34.0%	**47.0%**

500) $= 376.371, p = .000 < .05$ (columns compared with a z-test and p values adjusted with Bonferroni method). Identically to the recorded stimuli emotion labeling results, the Chi-squared test shows that anger, happiness, neutral and sadness are statistically different from others and that anger and fear are not statistically different from each other, but they are statistically different from the other emotions.

5.5.2 Feature Analysis

Research in music performance and speech has demonstrated that sound performed with different intended emotions present emotion-specific acous-

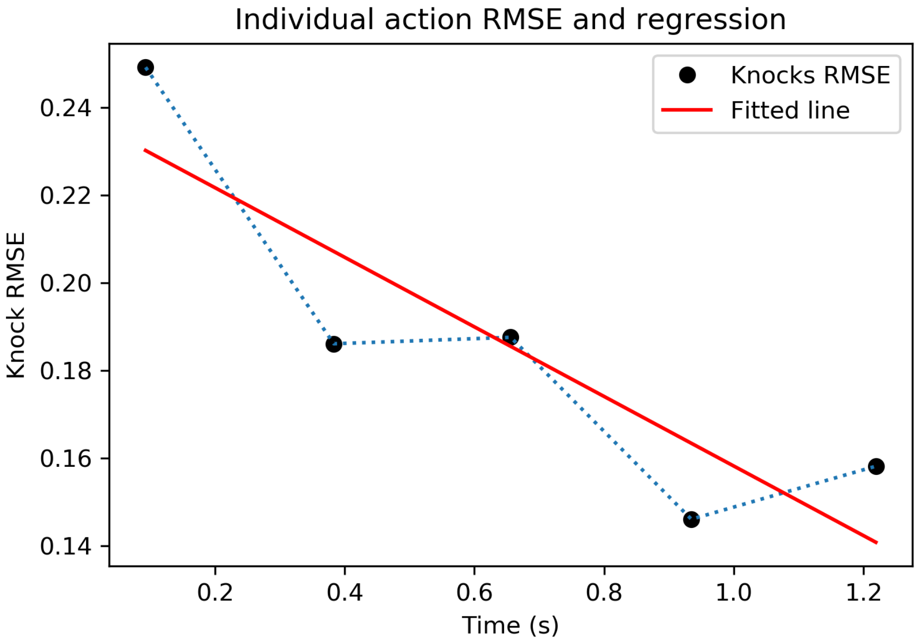


Figure 5.7: Example of the root-mean-square energy (RMSE) slope feature. The x-axis represents knocking action duration in seconds. The y-axis represents the RMSE of the individual knocks. The individual knock positions and computed RMSE values in the action are represented by the black dots. The fitted line indicates the result of regressing Knock RMSE against time. The slope of the fitted line, depicted in red, is negative, therefore the action has a decrescendo energy pattern.

tic patterns [284]. Thus, in order to understand the variations of the knocking patterns for different emotions, and to compare the original dataset to the synthesised sounds, a series of acoustic features are extracted from the dataset. The different features used for the analysis are the following:

- **Action duration**: Length of the knocking action. The knocking action length is the time passed from the first knock onset to the last knock decay.
- **Number of knocks per action**: The number of knocks are retrieved

by counting the number of onsets detected in each audio file.

- **Knocking rate**: The knocking rate is retrieved by dividing the number of knocks in an action by the total time of the action. This feature is computed only in actions with 2 or more knocks.
- **Knocking regularity**: The knocking regularity measures how regular in time (i.e., how steady) are the knocks in an action. Actions where the knocks are performed on a steady pace will have a higher regularity. To extract this feature, the inter-onset interval (IOI) of each action is calculated, followed by its coefficient of variation (the IOI standard deviation divided by the mean). Irregular actions will have higher coefficient of variation. This feature is computed only in actions with more than 2 knocks.
- **Root-mean-square energy (RMSE) slope**: This feature retrieves the crescendo or decrescendo energy pattern of an action. The root-mean-square energy of each individual knock is computed and, using these, a linear regression is calculated for each action. The slope of the fitted line determines whether an action has a crescendo (positive slope values) or decrescendo (negative slope values) energy pattern. An example of this feature computed to an individual action is depicted in Figure 5.7. This feature is computed only in actions with 2 or more knocks.

These features are also used in a parallel study [285], where the emotions of knocking sound effects performed by non-Foley artists are analysed to understand how they relate to the ones in this chapter.

The computed action duration, number of knocks, knocking rate and knock regularity are depicted Figures 5.8, 5.9, 5.10, 5.11 and 5.12 respectively. It can be seen that the synthesised sounds retain the features from the recorded dataset and, while their distributions are not identical, synthesised sounds follow the recorded dataset trend in all the computed features.

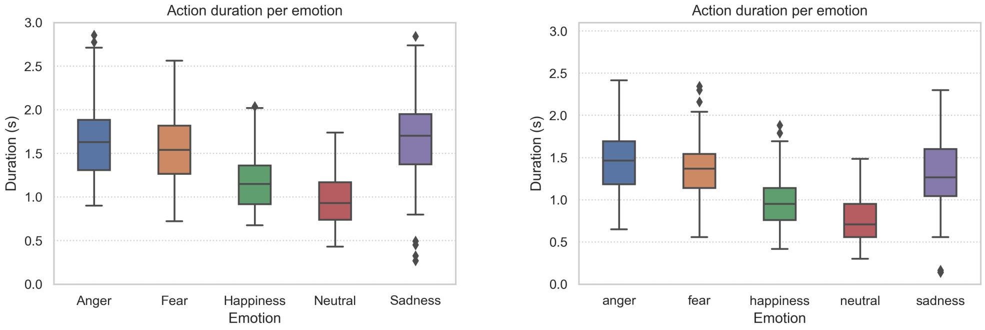


Figure 5.8: Recorded (left) and synthesised (right) knocking action duration per emotion.

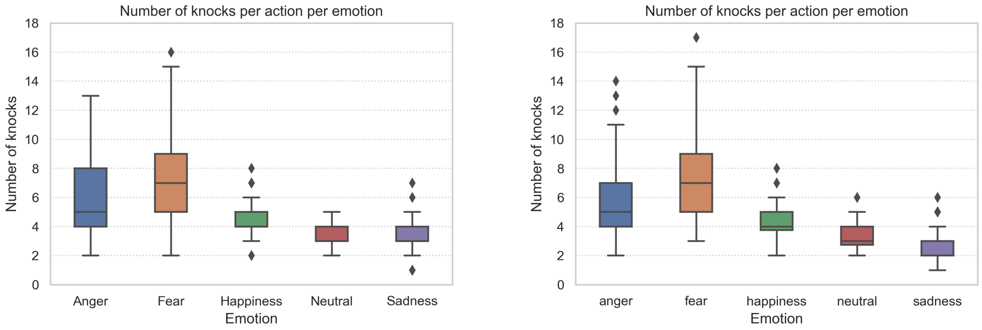


Figure 5.9: Recorded (left) and synthesised (right) number of knocks in each action per emotion.

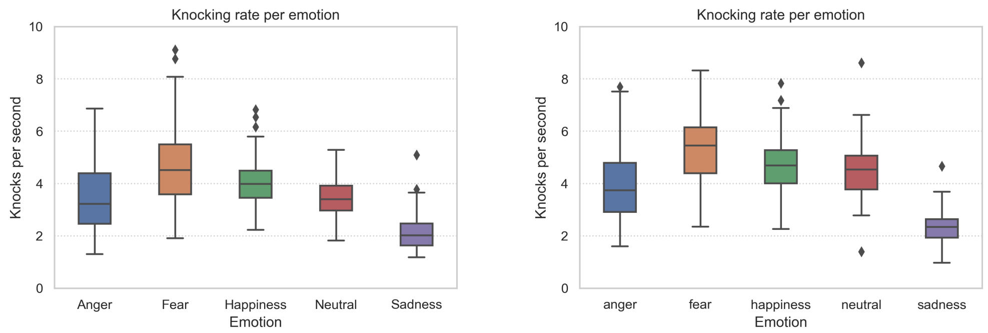


Figure 5.10: Recorded (left) and synthesised (right) knocking rate per action per emotion.

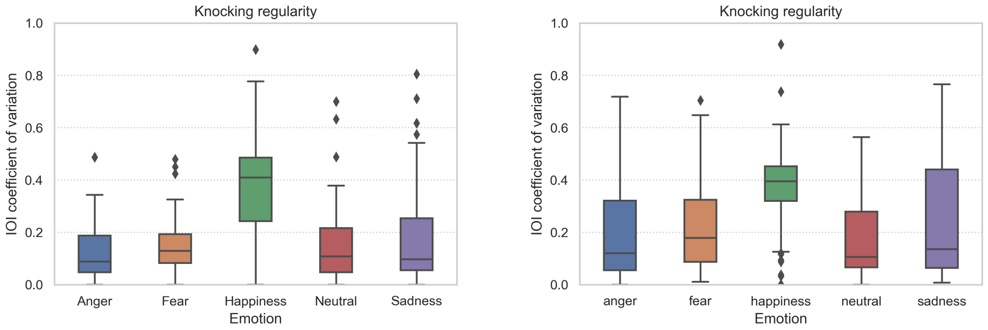


Figure 5.11: Recorded (left) and synthesised (right) knocking regularity per action per emotion. Higher values indicate less regularity.

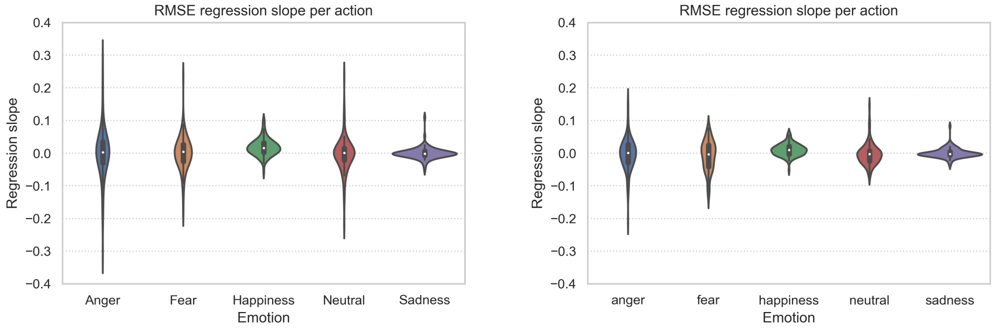


Figure 5.12: Recorded (left) and synthesised (right) action RMS energy slope per emotion.

5.6 Chapter Summary

While effective for certain sounds and interactions, creating plausible procedural audio models for *arbitrary* sounds using DSP methods may be a challenging and time consuming task for sound designers. This chapter proposes the use of conditional WaveGAN, a deep learning architecture tailored towards the modelling of audio in the waveform domain, as an alternative to “traditional” DSP methods for the synthesis of sound effects. More specifically, this chapter focuses on the modelling of knocking sound effects with emotional intention. The proposed system is evaluated both subjectively and by analysing a series of audio features from the recorded and synthesised sounds. Thus, this chapter addresses the second research question (RQ 2) of this thesis.

Regarding the subjective evaluation of the plausibility of the synthesized sounds, results from the RS listening test show that the model is close to be able to synthesise sounds that would be indistinguishable from pre-recorded samples to persons without expertise in sound design. This is supported by the BFA, which found anecdotal evidence in favour of the null hypothesis (i.e., no difference in perception) among this group of participants. However, RS listening test results show that participants with medium to high expertise in sound design can consistently discriminate between recorded and synthesized sounds, a fact for which the BFA found extreme evidence. This could be due to participants with medium to high expertise in sound design being familiar with the quality expectations of sound effects, thus being more sensitive to artifacts and sampling rate discrepancies between the recorded and synthesized sounds. Another factor could be the playback device used, where participants proficient in sound design may have professional equipment that aids in detecting possible artifacts in the synthesized sounds.

In terms of the intended and perceived emotions, results show that, on average, most emotions are correctly labelled in both the recorded and the synthesised sound groups, with the exception of fear and anger, which are confused with each other for both the recorded and synthesised sounds. Results suggest that, even though the synthesised sounds are identified by people with expertise in sound design, the perceived emotion ratings are similar to the ones in the pre-recorded samples. Since the acoustic features of the synthesised dataset are very similar to those of the recorded dataset, it is hypothesised that they strongly contribute to the perception of emotions, and that they are sufficiently well modelled in the synthesised dataset to allow listeners to perceive the intended emotions, to a similar degree to the recorded dataset. Thus, the generator is able to capture the emotional intentions from the data. Additionally, the acoustic features distribution is not exactly the same in the two datasets. This is desirable considering that the aim is to generate new data and not to resynthesise data that was already in the training dataset.

In terms of applications, this approach could be used in video games and post-production. While sound synthesis using GANs is not done in real-time (GANs synthesise one or several sounds at a time, not a continuous stream of sound, sample by sample), it has been shown in a comparable scenario that the sound generation may be fast enough to be used in real-time without perceived latency [278]. In the experiments reported in [278], they achieved a latency of $\approx$ 25ms using a smaller model than the one used in this chapter, generating 4096 samples at a time, a lower sampling rate than the one used in this work (16 000 Hz), and a GPU with more capacity (NVIDIA GTX 1080). In contrast, a latency of 66.9 ± 5.9 (mean ± sd) ms is achieved to generate 65 536 samples at 22 050 Hz using a less capable GPU (NVIDIA RTX 1060). While the threshold of ≈60 milliseconds is larger than that required to avoid the perception of latency between an action and the resulting sound (≈30 milliseconds, [286]), by reducing both the architecture size and the sampling rate, and increasing the hardware specifications, it is possible to synthesise audio without perceived latency, as reported in [278].

Alternatively, instead of generating a new audio asset in reaction to a player performing an action, a batch of sounds may be created and stored in a buffer to play them sequentially. Once the buffer is close to its exhaustion, another newly synthesised batch of sounds could replace it. In the experiments, it has been shown that generating a batch of 100 sounds (equivalent of generating ≈5 minutes of audio) takes 726.5 ± 12.5 milliseconds with the proposed configuration.

Regarding the control over the synthesised samples, the model is capable of synthesise knocking sound effects with emotional conditioning, passing the desired emotion at the time of generation. It may be useful, however, to include other acoustic features, such as the features used for the analysis in Section 5.5.2, to further refine the control possibilities. This control scheme would allow, for instance, to specify the number of knocks to synthesise or their energy pattern within the knocking action.

While the sounds synthesised by the model can be identified as "synthetic" by people with expertise in sound design, it is not far from consistently “fooling” non-experts. Nonetheless, to explore how to improve the perceptual performance of the model it is planned, in future work, to use the GANSynth [201] architecture, to compare it against the proposed conditional WaveGAN architecture on the synthesis of knocking sound effects. While GANSynth is designed for the synthesis of musical notes with pitch conditioning – a domain somewhat separate from sound effects, especially from percussive sounds – it received positive perceptual results in their evaluation, and may yield favourable perceptual outputs in the context of knocking sound effects as well.

Yet another consideration to potentially improve the models performance is the use of sub-band decomposition as in [206], where raw audio is modelled by predicting sub-band signals using a pseudo-quadrature mirror filterbank (PQMF) [287]. It is also possible to employ a different discriminator, such as in [18], where they replace the WaveGAN discriminator for a HiFi-GAN discriminator [224]. Alternatively, a different audio representation could be considered also, such as those proposed in [182].

However, regardless of the architecture used, a problem when training data-intensive deep learning models – such as most GAN architectures – is precisely the data itself, or lack thereof. For this chapter, a professional Foley artist was commissioned to record a dataset of knocking sound effects with emotional intentions, since such sounds are challenging to find in the necessary quantity and quality to train a model like the one proposed above. However, it is not practical to record or commission a dataset for every new model trained on a different sound category. Moreover, lack of training data may hamper the quality and diversity of the synthesised data [159], which is a challenging issue when using generative methods, especially for audio [288]. Chapter 6 of this thesis will address these issues, proposing an architecture that relies on a *single* sound effect only in order to be trained.

CHAPTER 6

ADDRESSING DATA SCARCITY: SINGLE-EXAMPLE AUDIO GENERATION

6.1 Introduction

In the previous chapter, the synthesis of class-conditional sound effects using GANs was investigated. To that end, a professional Foley artist was commissioned to record a dataset of knocking sounds with emotional intentions. However, recording or commissioning datasets on demand each time a new generative architecture is trained is time-consuming, especially with the amount of data some of these architectures – such most GANs – may need. To contextualise this, in [208] they use a dataset comprised of ≈300K samples of kick, snare and cymbal sounds to train a GAN and synthesise drum sounds conditioned on timbral features. In Chapter 5, the focus was on an arguably more scarce category of sounds: knocking sound effects performed with emotional intentions. Hence, creating a dataset was necessary to train the network. Other more common categories of sound effects, such as footsteps, may be more suitable to model using data-intensive approaches given their wider availability. However, depending on the granularity of the desired control scheme, they may be also challenging to model. For instance, while the floor surface can be a natural choice to condition a footstep sound effect model, it may also be desirable to specify the type and material of the shoe (e.g., boots or trainers; leather or fabric), the perspective and distance of the recording (e.g., first or third person; close-up or far away), the articulation (e.g., walking, running), to name a few. This potential granularity becomes even more troublesome when the category of sounds is rare or scarce. Moreover, while audio augmentation techniques such as adding noise to a signal, shifting its pitch, performing time stretching, or masking section of its spectrogram are commonly used in deep learning [289], those will not necessarily produce a novel sound effect. Instead, they modify the existing sound effect, which – especially in situations where the sound category is particularly scarce – will not result in a suitable dataset for training a model.

In this chapter these issues are addressed by proposing the use of single-image generative adversarial networks (single-image GANs) to generate variations of a single training example, removing the need of a large dataset. Thus, instead of training on a curated – and potentially, large and difficult to obtain – dataset aimed towards the synthesises of a particular category of sounds, it is left to a prospective user (e.g., a sound designer) to decide which sound is the best suited to, for instance, design an action. That sound will be used to train the network and generate infinite variations from it, as if they where different takes from the same recording session.

Single-image GANs exploit the internal statistics of a single training example to generate novel variations from it. An example of these architectures is SinGAN [290]. SinGAN is an unconditional generative model that uses a progressive growing multi-scale approach, training a fully convolutional GAN on a different resolution at each stage. The model starts producing small-sized images, which are upsampled and fed to the next stage alongside a random noise map. SinGAN uses patch-GANs [291], training on overlapping patches of the training image at the different stages. Given the GAN receptive field is fixed with respect of the image size, the model learns to capture finer details as the training progresses. This fully convolutional design also enables image generation of arbitrary size just by changing the dimensions of the input noise maps. ConSinGAN [292] is another single-image GAN architecture. Built upon SinGAN, the authors proposed some improvements to it, such as concurrent training of the different stages or the resizing approach when building the image "pyramid" for the different resolutions, reducing the number of parameters and the training time. These architectures are also capable of performing other tasks such as retargeting, animation or super-resolution. Regarding the audio domain, Catch-A-Waveform (CAW) [230] is a recent audio time-domain architecture inspired by single-image GANs that is capable of producing novel audio samples of arbitrary length with just 20 seconds of training data. They demonstrate the architecture's performance on music, speech and environmental sounds (such as applause or thunderstorm), yielding promising results. CAW is also capable of performing different tasks directly on the audio domain, such as bandwidth extension, denoising or audio inpainting. However, this chapter is focused on the modelling at the individual sound effect level, with the aim of producing novel one-shots instead of streams of audio such as music excerpts or speech. As in Chapter 5, this chapter addresses the second research question (RQ 2) of this thesis.

6.2 Method

From a high-level perspective, the method consists on using a single-image GAN architecture and train it using a single sound effect to synthesise infinite variations from it. To potentially increase the level of variation, game audio workflows are adopted, where it is common to use different sound layers for a single sound effect in the sound design process in order to increase its variability [5]. Thus, the use of multi-channel spectrograms to train the model on the various layers that comprise a single sound effect (such as, for instance, the heel, the tip and the shoe Foley in a footstep), is explored. Once trained, the model is able to produce infinite unconditional variations of the sound – or the layers, for models trained on multi-channel spectrograms.

6.2.1 Audio Representation

This section starts by describing the audio representation used by the model, both for training and inference. Since the approach is based on computer vision single-image GAN architectures, such as SinGAN [290] or ConSinGAN [292], spectrograms are employed as the audio representation of the model. Despite that, from a data representation point of view, spectrograms are not equal to images, they can be trained in a somewhat similar manner.

While a 2-channel frequency-domain representation consisting of a magnitude spectrogram and instantaneous frequency (IF) has been used to achieve state-of-the-art results on GAN audio synthesis of pitched musical notes [201, 182], [293] recently studied the use of 1-channel phaseless log-magnitude spectrograms as an alternative for synthesising non-harmonic sounds (such as chirps or pops), achieving better perceptual results in this context. To invert the phaseless log-magnitude spectrogram back to audio, they reconstruct the phase using the Phase Gradient Heap Integration (PGHI) algorithm [294]. Another popular phase reconstruction method is the Griffin-Lim [27] algorithm. A pilot experiment is conducted with a phaseless log-magnitude spectrogram representation, reconstructing its phase with both the PGHI the Griffin-Lim algorithms and, in these preliminary tests and context, Griffin-Lim produced better perceptual results using a 75% frame overlap. The FFT size choice also has a significant impact on the results, being 512 the size that produced the best consistent results in the pilot experiments. It is opted then to use a phaseless log-magnitude spectrograms with a FFT size of 512, 75% overlap, a Hanning window of the same size of the FFT and reconstructing the phase with Griffin-Lim [27].

Similar to [295], multi-channel spectrograms are used as the input to the model. The use of multi-channel spectrograms instead of single-channel spectrograms in the context of sound effects synthesis presents some benefits such as, for instance, allowing for some parametrisation of the sound synthesis, as the layers could be synced to an animation, triggered asynchronously, have different amplitude from each other, etc. While in [295] the authors use the multi-channel spectrograms to represent the pitch and intensity of musical notes, here they are used to represent the different layers of the sound effect. To be precise, multi-channel spectrograms are built by stacking multiple sound layers along the channel axis, and these layers are provided by the user to the network directly (they are not extracted programmatically from layered sounds). The multi-channel spectrogram term is used (instead of multi-layer) for consistency with the literature.

To build the multi-channel spectrograms the different sound layers are loaded and their amplitudes normalised in a range of $[-1, 1]$. Next, the longest sound effect layer (in the time-domain) is measured, and the remaining layers are zero-padded to this length. Then, the multiple audio layers are transformed into log-magnitude spectrograms by taking the STFT with, as described above, FFT size of 512, 75% overlap and a Hanning window of the same size of the FFT. The phase information is discarded immediately after, and the different log-magnitude spectrograms are stacked along the tensor channel axis, as if they were different channels of an image. Finally, the multi-channel spectrograms are standardised (mean 0, standard deviation 1). During inference this standardization is reverted, the logarithm inverted, and the channel and batch dimensions are permuted (so the layers appear as different sounds in a batch of sounds instead as if they where different channels), reconstructing the phase using the Griffin-Lim algorithm [27] and transforming the spectrograms back to audio by taking the ISTFT. If the training sound effect has only one layer, a single-layer log-magnitude spectrogram is used. Unless stated otherwise, all sound layers are mono, with a sampling rate of 44.1 kHz.

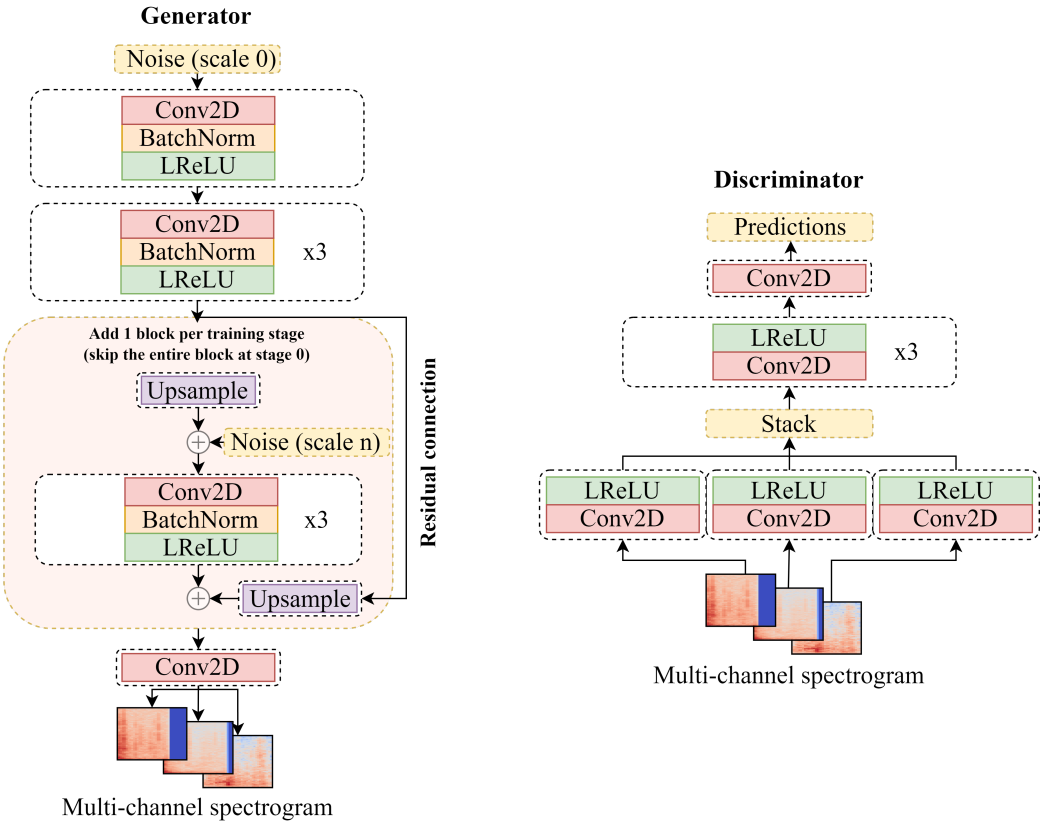


Figure 6.1: SpecSinGAN architecture. The generator is depicted on the left, while the discriminator is depicted on the right. For reading simplicity, the internal upsampling and padding operations in the generator are not represented in the image.

6.2.2 Architecture

SpecSinGAN, depicted in Figure 6.1 is built upon the ConSinGAN [292] architecture. ConSinGAN [292] retains the benefits of SinGAN [290], but proposing multiple improvements to it, achieving state-of-the-art results, hence its use as a foundation to develop SpecSinGAN.

The SpecSinGAN discriminator, depicted to the right in Figure 6.1, takes – depending on the model configuration – either single or multi-channel log-magnitude spectrograms as its input. Thus, the SpecSinGAN discriminator will have as many input blocks in parallel as the number of channels (i.e., sound effect layers) of the training spectrogram, each one of them consisting of a convolutional layer followed by a Leaky ReLU activation. For instance, for a 3-channel spectrogram, the discriminator will have 3 input blocks, and each channel of the spectrogram will be processed individually by just one of them in parallel. The resulting feature maps are stacked along the batch axis, in a somewhat similar manner to [295]. Then, those feature maps go onto 3 groups of convolutional layers followed by Leaky ReLU activations, and finally onto a last convolutional layer. All the convolutional layers have a kernel size of 3, stride 1, 1 dilation rate and a N_{filters} filters, except for the last layer where $N_{\text{filters}} = 1$. For the Leaky ReLU activation an $\alpha = 0.05$ is used. As suggested by the ConSinGAN authors [292], a second discriminator – used during the final training stages – is also implemented, as slight improvements on the results were found. The second discriminator is identical to the first one, but with dilation on all its convolutional layers to increase its receptive field.

The SpecSinGAN generator, depicted to the left in Figure 6.1, is similar to the growing generator in ConSinGAN. Thus, the generator trains on N_{stage} stages, increasing the spectrogram scale from an user-defined minimum scale at stage 0, to the full-band spectrogram at the last scale. For the first training stage, the generator has 4 convolutional layers, each one of them followed by batch normalisation and a Leaky ReLU activation, and a final output convolutional layer. As the output range of the feature maps (i.e., the different scale spectrograms) is not constrained, a hyperbolic tangent activation is not used in the output of the generator. At each subsequent training stage, 3 more convolutional layers with batch normalisation and Leaky ReLU activations are added just before the output convolutional layer, increasing the generator capacity (i.e., producing "larger" spectrograms) as the training progresses. All layers have the same hyperparameters as the intermediate layers in the first discriminator.

6.2.3 Training and Inference

The training stages in the training process are set to $N_{\text{stage}} = 10$. At the start, an "image pyramid" of training spectrograms is built, going from the first stage where the spectrograms are downsampled to a coarser scale, to the last stage, where the spectrograms are at the original resolution. The maximum size of the image is set to its original resolution, and the minimum size depending on the sound. The training starts at the coarsest scale, where a noise map (i.e., a Gaussian noise $\mathcal{N}(0, 1)$ "image") with the shape of the user-defined minimum size at scale 0 is created. The generator receives those noise maps and produces a spectrograms at that scale. As can be seen in Figure 6.1, the features maps (i.e., "coarse" spectrograms) from previous stages are upsampled and passed directly to the current stage, mixing them with the corresponding noise at that scale to increase diversity. The default noise amplification (i.e., a multiplier to weight the noise against the feature maps) is set to 0.1. The feature maps from the previous stage are also upsampled and summed after the convolutional block at each stage. Overall, the training process is identical to ConSinGAN, with the exception of adding a second discriminator halfway through training.

As with [292], 3 stages are trained concurrently, using a learning rate of

0.0005 for the current stage and a 0.1 scaling for the other 2 stages below. WGAN-GP [157] is used for the adversarial loss, in combination with a reconstruction loss with a weight of 10 to increase the training stability, where the generator learns to resynthesise the training spectrograms. Because the receptive field of the discriminator is fixed, at stage 0, due to the spectrograms being smaller, there are less overlapping patches to evaluate than in the final stages, thus the discriminator is able to assess the global structure of the spectrogram. As the produced spectrograms "grow" in resolution (i.e, they increase their size) during the training process but the receptive field of the discriminator remains unchanged, the discriminator focuses more on the details of the spectrograms as the training progresses. The second discriminator – with an increased receptive field – introduced halfway through training prevents the generator from deviating from a coherent global structure at the final training stages. The ConSinGAN upsamplig strategy for the generator is also used, where the feature maps after the first convolutional block are slightly upsampled to increase the diversity at the edge of the spectrograms.

As depicted in Figure 6.2, during inference, the x-axis dimension of the noise maps that go into the generator is randomised. As a result, each multichannel spectrogram of a generated batch has a different length. This is thanks to the fully-convolutional nature of the network, where the input noise maps are not constrained to be of a specific shape. Note the x-axis dimension of the noise maps is not randomised during training, but only during inference once the model is trained. The spectrograms are transformed into audio using the process described in Section 6.2.1, and the layers of the batch are

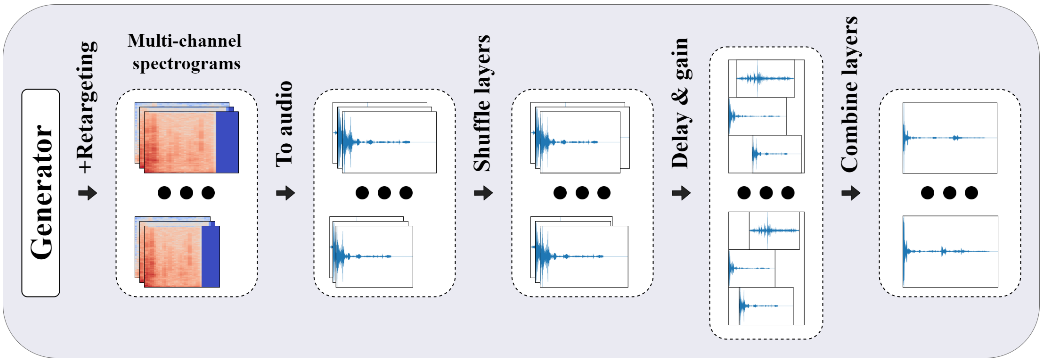


Figure 6.2: Multi-channel synthesis during inference: the generator produces multi-channel log-magnitude spectrograms that are slightly different in length on their x-axis. They are transformed back to audio by reconstructing their phases using the Griffin-Lim algorithm and taking the ISTFT of the different layers, shuffling the different length layers afterwards. Finally, a randomised delay and gain is applied to the individual layers that comprise the sound effect, combining them to render the final audio files.

shuffled, combining the different-length layers. Finally, the delay and gain of the layers are randomised with respect to each other, and the layers are summed into a single audio file.

6.3 Experiments

Four one-shot sound effect categories that are commonly found in video games and media are selected, using three sound layers for each of them: footsteps on concrete (heel, tip and shoe fabric), footsteps on metal (heel, tip and metal rattle), gunshots (noise/body, mechanic component and tail) and character jump (human efforts, Foley of the character clothes and metal clinks of character equipment). The training sound effects are collected from the Freesound website [216]. To assess the effect of the multi-channel spectrogram approach, two SpecSinGAN models are trained per sound effect:

one with single-channel spectrograms and another with multi-channel spectrograms. The models trained with single-channel spectrograms use a single training audio file with all the different layers combined.

SpecSinGAN allows the input of both arbitrary length audio files and arbitrary numbers of layers. However, it was found that different sound effects require different training hyperparameters, depending on the number of layers, the shape (i.e., length) of the training sound, the frequency content, and the desired degree of variation. For this study purposes, three layers of relatively short one-shots (≈200-750 ms) are a good compromise. In general, more layers or reverberant sounds will require a higher number of iterations per training stage. For sounds with very sharp transients at the beginning, adding a small zero-padding at the start of the audio file before training can prevent artifacts. Other parameters could be changed to accommodate longer or more challenging sounds, such as the the number of training stages or even the sampling rate or the FFT size.

In the experiments, both the single and multi-channel spectrograms of the footsteps on concrete and metal are trained for 2000 iterations per stage, using $N_{\text{filters}} = 64$ filters in the convolutional layers, a dilation rate of 3 in the second discriminator, and setting the minimum tensor size of the training pyramid (on any axis) to 50. Gunshots are trained for 8000 iterations per stage using $N_{\text{filters}} = 128$ filters, a dilation rate of 2 in the second discriminator, and setting the minimum size to 11. Finally, the character jumps are trained for 8000 iterations per stage using $N_{\text{filters}} = 128$ filters, a dilation rate of 3 in the second discriminator, and setting the minimum size to 25. As an indication, training on an NVIDIA Tesla V100 takes approximately 50 minutes for single and multi-layer footsteps on concrete and metal, 200 minutes for single-layer gunshots, 600 minutes for multi-layer gunshots, and 240 and 500 minutes for single and multi-layer character jumps respectively. Once trained, the saved models have a size on disk of ≈4.5 MB and ≈330K parameters for all the footsteps models and of ≈17.5 MB and ≈1.3M parameters for the gunshot and character jump models.

During inference the delay and gain of the different layers are randomised so they are coherent with the aesthetics of the final sound effect. In terms of retargeting, no more than a randomised ±15% range multiplier is applied to the x-axis (corresponding with the spectrogram time axis) of the input noise maps. Multiplying the input noise maps by a larger number may result in audible artifacts. For reference, synthesising 1000 sound effects on a NVIDIA Tesla V100 took approximately 60 and 100 seconds for single and multi-channel footsteps in concrete respectively. An example of some of the recorded sounds alongside synthesised sounds using single and multi-channel configurations of SpecSinGAN is depicted in Figure 6.3. Sound examples can be also find at the project website.[1]

6.4 Evaluation

A listening study is designed to evaluate the sounds resulting from the experiments in Section 6.3, for the 4 categories of footsteps on concrete, metal, gunshot, and character jump. 4 different systems are compared on plausibility and variation: real recordings (hereafter, Real), SpecSinGAN with single and multi-channel spectrograms (denoted by the subscript "single" or

[1] www.adrianbarahonarios.com/specsingan

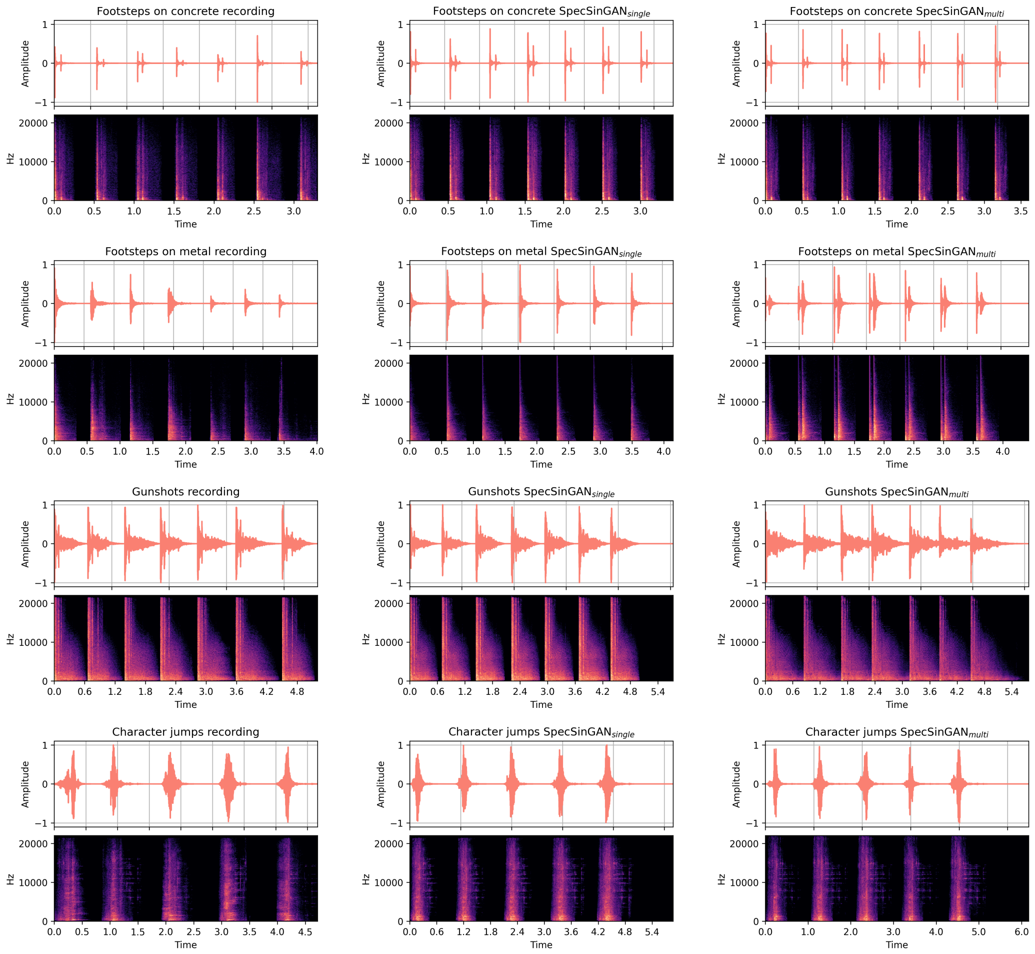
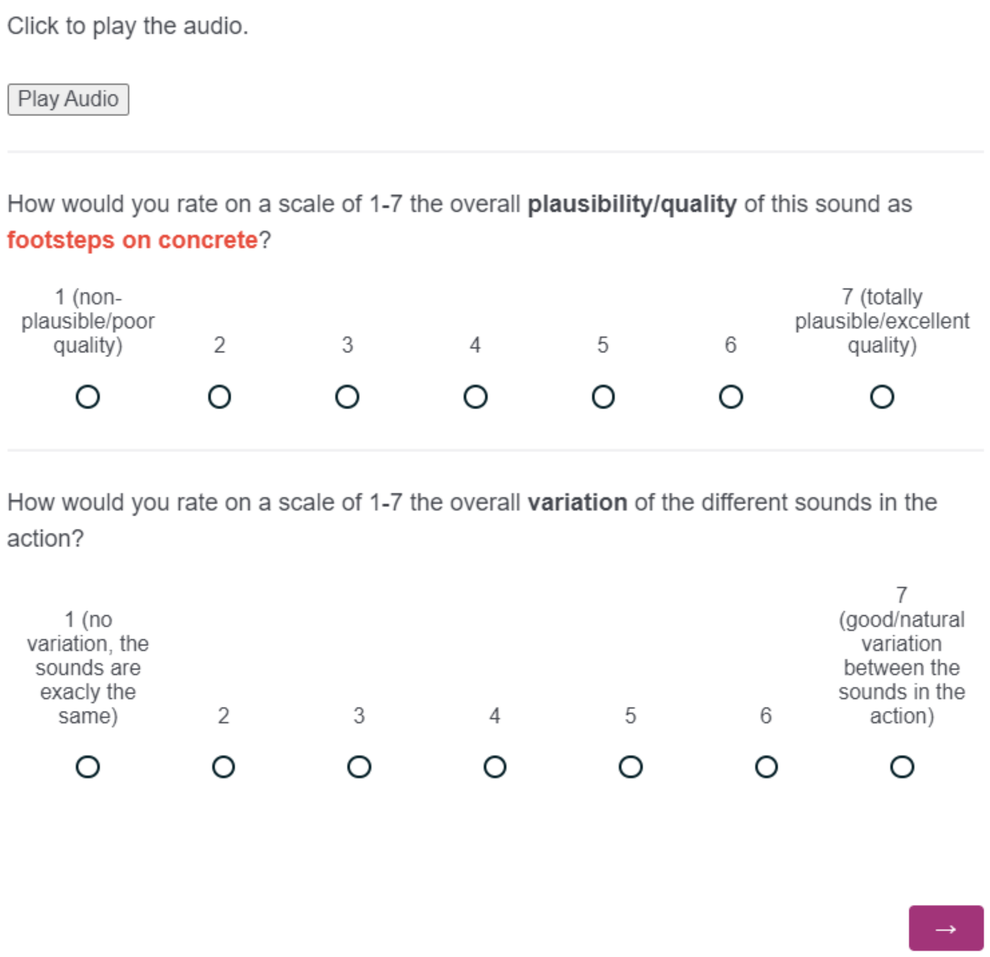


Figure 6.3: Waveform (top) and magnitude spectrogram (bottom) pairs of recorded and synthesised sound variations produced by SpecSinGAN$_{\text{single}}$ and SpecSinGAN$_{\text{multi}}$. The first column depicts an example of the recorded sounds, the second and third columns depict SpecSinGAN$_{\text{single}}$ and SpecSinGAN$_{\text{multi}}$ respectively. Each of the rows depicts one of the sound effects considered: footsteps on concrete, footsteps on metal, gunshots and character jumps respectively. Note the network outputs a single sound effect at a time, and they are concatenated along their x-axis for this example.

“multi” respectively), and Nemisindo [16, hereafter, NM]. NM is a web-based procedural audio service that enables the creation of synthesised sound effects using DSP methods. For the character jump sounds, only Real and SpecSinGAN variants were compared, as NM does not offer this type of sound at the time the study was carried out. Sounds for the Real category are sourced from the same Freesound [216] training examples used in Section 6.3, taking sound variations from the same file or designing them (e.g., cutting, equalising, fading) when needed. The SpecSinGAN sounds are those of Section 6.3. NM sounds are taken directly from their presets (without searching the parameter space), finding the closest ones to the target category of sounds.



Click to play the audio.

How would you rate on a scale of 1-7 the overall **plausibility/quality** of this sound as **footsteps on concrete**?

1 (non-plausible/poor quality)	2	3	4	5	6	7 (totally plausible/excellent quality)
○	○	○	○	○	○	○

How would you rate on a scale of 1-7 the overall **variation** of the different sounds in the action?

1 (no variation, the sounds are exacly the same)	2	3	4	5	6	7 (good/natural variation between the sounds in the action)
○	○	○	○	○	○	○

Figure 6.4: Screenshot of the online test interface presented to the participants in the listening study. Participants are asked to listen to the stimuli once and select on a [1..7] scale their perceived level of plausibility an variation. The image depicts the rating of a footstep in concrete sound effect.

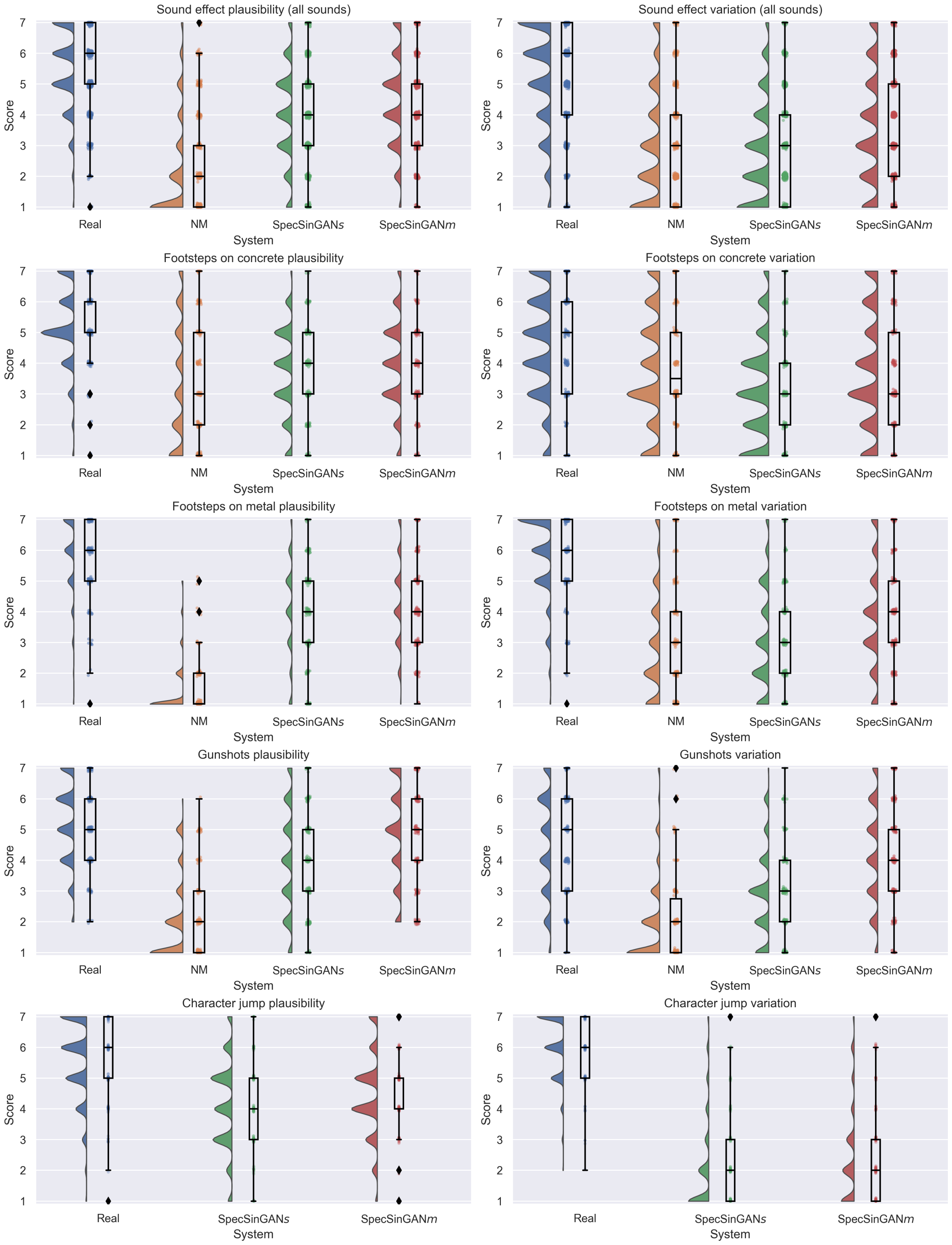


Figure 6.5: Listening study results for the different sound effects and systems considered. Real recordings are denoted by ‘Real’, Nemisindo by ‘NM’ and SpecSinGAN using either single or multi-channel spectrograms by ‘SpecSinGAN$_{\text{single}}$’ and ‘SpecSinGAN$_{\text{multi}}$’ respectively. Participants were asked to rate each stimuli on both plausibility and variation on a [1..7] scale. Note the scatter plot represents the individual ratings, with jitter added to prevent overlapping as only natural numbers were given as rating options.

Table 6.1: Listening study raw results (mean ± sd) for real recordings (Real), Nemisindo (NM), single-channel SpecSinGAN ($\text{SpecSinGAN}_{\text{single}}$) and multi-channel SpecSinGAN ($\text{SpecSinGAN}_{\text{multi}}$). The first half of the table shows the results for sound effect plausibility, and the bottom half the results for sound effect variation. The sounds are evaluated on a [1..7] scale where higher values are better (best performers highlighted in bold).

			Sound effect plausibility		
	Footsteps on concrete	*Footsteps on metal*	*Gunshots*	*Character jump*	All sounds
Real	**5.16±1.32**	**5.74±1.32**	**5.21±1.33**	**5.59±1.28**	**5.42±1.33**
NM	3.33±1.95	1.44±0.89	2.05±1.34	-	2.27 ±1.65
$\text{SpecSinGAN}_{\text{single}}$	3.68±1.45	3.96±1.60	3.97 ±1.71	4.13±1.45	3.93±1.56
$\text{SpecSinGAN}_{\text{multi}}$	3.95±1.45	4.29±1.46	4.87±1.56	4.44±1.39	4.38±1.50
			Sound effect variation		
	Footsteps on concrete	*Footsteps on metal*	*Gunshots*	*Character jump*	All sounds
Real	**4.55±1.69**	**5.81±1.35**	**4.51±1.66**	**6.06±1.04**	**5.23±1.62**
NM	3.76±1.83	3.00±1.76	2.04±1.40	-	2.93±1.82
$\text{SpecSinGAN}_{\text{single}}$	3.13±1.66	3.09±1.67	2.87±1.46	2.13±1.55	2.80±1.63
$\text{SpecSinGAN}_{\text{multi}}$	3.73±1.85	3.81±1.73	4.16±1.66	2.53±1.60	3.55±1.82

The sounds are presented concatenated to make sound actions (e.g., footsteps turned into walking), resulting in audio clips of ≈5-seconds long. Participants rated 5 sounds per category per system in terms of plausibility and variation on a scale of 1 (not at all plausible/varied) to 7 (completely plausible/appropriately varied). As in Chapters 4 and 5, a clearly synthesised sound (a burst of noise) is included to test the attention and reliability of the participants. The Prolific platform is used to conduct the listening study, pre-screening participants such that only those over 18 years old who play video games for at least 6 hours a week are selected, encouraging them to use headphones during the evaluation. A total of 30 participants are recruited, with ages ranging from 18 to 40 years old, and a diverse set of expertise in sound design (from non-experts to professionals), compensating them £9/h. The online listening study interface presented to the participants is depicted in Figure 6.4, and the consent form and the demographic questions that participants were asked can be found in the Questionnaires appendix. The raw answers from the participants are visualised in Figure 6.5 and numerically presented in Table 6.1.

While in Chapters 4 and 5 the RS listening test [271] and a parametric Bayes factor analysis (BFA) are used to assess the performance of the proposed system compared to pre-recorded samples in terms of plausibility, here non-parametric BFA is used instead, because the ratings are collected on a Likert scale. Multiple systems are also compared (i.e., pre-recorded samples, two variants of SpecSinGAN and DSP procedural audio models), and not just the plausibility is evaluated, but the plausibility *and* variation of the recorded and synthesised sounds. Thus, the BFA may provide a more comprehensive interpretation of the results based on the hypotheses of how the systems compare to each other. To conduct the BFA the method proposed in [273] is used, with the interpretation of how two hypotheses H_0 and H_1 compare from [274, 275] summarised in Table 4.1 on p. 75.

It was hypothesised that Real would have better plausibility and variation than any other system, and the BFA found extreme evidence for this ($BF_{10} > 100$). It was also hypothesised SpecSinGAN would have slightly more plausibility than NM for footsteps on concrete, finding anecdotal and strong evidence for this for SpecSinGAN$_{\text{single}}$ ($BF_{10} = 1.09$) and SpecSinGAN$_{\text{multi}}$ ($BF_{10} = 12.5$) respectively. In addition, it was hypothesised that SpecSinGAN would have higher plausibility ratings than NM for footsteps on metal, and the BFA found extreme evidence for this ($BF_{10} > 100$). Finally, it was hypothesised SpecSinGAN and NM would have similar plausibility for gunshots, and the BFA rejected this ($BF_{10} > 100$). The data suggested SpecSinGAN having more plausibility than NM for gunshots. Regarding variation, it was hypothesised SpecSinGAN and NM would have similar values, confirmed for SpecSinGAN$_{\text{single}}$ according to the BFA ($BF_{10} = 0.09$) and rejected for SpecSinGAN$_{\text{multi}}$ ($BF_{10} > 100$). In the latter case, the data suggest SpecSinGAN$_{\text{multi}}$ had higher variation values than NM.

6.5 Chapter Summary

Audio asset creation can be a time-consuming process. In this chapter SpecSinGAN was introduced, an unconditional generative architecture capable of synthesising novel variations of one-shot sound effects training on a single training example.

SpecSinGAN performed statistically significantly better in the listening study, compared to the procedural audio models considered. NM presets available at the time of the listening study were used instead of searching the parameter space, so, while this is a reasonable choice, it is possible that NM (or any other DSP tool) would produce better results following a more exhaustive search of its parameter space.

In future work, it is planned to introduce user control over the synthesis, allowing users to define certain high-level properties of the synthesised sounds. For instance, for a footstep model user could specify the articulation (e.g., walking, running), the type of shoe, or the surface by either incorporating that information onto the model in the form of additional sound layers; or by using an ensemble system, where each of the models is devoted to one property. Additionally, to both increase the plausibility and the control of the sounds produced by SpecSinGAN, it may be worth considering novel single-image diffusion models such as [296], which have outperformed other GAN approaches such as SinGAN [290], as well as introduced text conditioning to the architecture. Another improvement could be the implementation of automatic hyperparameter tuning, given that, as discussed in Section 6.3, different sounds required different hyperparameters and, despite there being some intuitions on how to tune them, this still involves a manual process.

While this chapter is focused on the synthesis of one-shot sound effects, in [297] this system was also used to produce continuous streams of sounds in the context of data sonification. Specifically, in [297] SpecSinGAN is used as the sound source to sound design the energy consumption drawn from a smart plug by generating a fire-like sound that increases or decreases in intensity depending on the energy used. The fire sound is comprised of two elements, each one of them synthesised by a different SpecSinGAN model: the fire crackles and the fire base layer. The fire crackles are generated by training a model on three different crackle sounds, and generating variations of them as one-shots – similar to the procedure introduced in this chapter. The fire base layer is generated by training a model on three fire sound intensities (low rumble, middle rumble and hiss), and overlapping the variations resulting from the model in the time-domain, achieving a continuous time-varying layer that never repeats itself. In this particular case, all sounds are generated offline and used as the source for the sonification.

Finally, it is also acknowledged that, while the focus has been on arbitrary sound effects, further listening studies need to be carried out to understand how SpecSinGAN compares to DSP methods such as [12] for generating variations of target percussive sounds and to [230], adapting it to work with shorter sounds at 44.1 kHz. It is also observed that, despite showing SpecSinGAN is a viable alternative to synthesise arbitrary one-shot sound effects, DSP-based systems are capable of not only producing controllable continuous streams of audio, but also of running in real-time with direct input from either human-interpretable controls or in-game parameters, granting them great adaptability. Chapter 7 will address these challenges, focusing on the controllability and introducing an architecture that combines DSP components with deep learning for this task. However, SpecSinGAN can be useful in two main contexts. First, in a context where sound designers need to produce novel variations of a specific pre-recorded sound, avoiding hand-crafting multiple versions of it. Second, as a data augmentation tool where, unlike common audio augmentation techniques that do not necessarily lead to a novel sound effect but to the same sound effect with some transformations applied to it, the resulting variations are akin to different takes from the same recording session.

CHAPTER 7

SOUND EFFECTS SYNTHESIS USING DIFFERENTIABLE DSP

7.1 Introduction

Back in Chapter 4 of this thesis it is demonstrated that, with constraints, DSP systems can be an effective method for the synthesis of sound effects. Chapters 5 and 6 focus on the synthesis of sound effect using deep learning methods directly, disregarding part of the vast wealth of knowledge described in Chapter 2. In this chapter, a combination of DSP methods alongside deep learning (i.e., DDSP, as described in Section 3.3.1) are used to achieve controllable sound effect synthesis.

In the original DDSP paper [145], they synthesise harmonic musical notes controlled by pitch and loudness using a harmonic plus noise synthesiser [31]. Once trained, the resulting synthesiser is able to produce sounds with human-interpretable controls (e.g., pitch and loudness). In the context of the synthesis of sound effects, and particularly in game audio, human-interpretable continuous controls are desirable, as the synthesiser could adapt its output to, for instance, in-game events (in the case of running in real-time) or to animations (running offline). DDSP-based models also benefit from requiring comparatively less data to train than other data-driven approaches [145]. Additionally, DDSP synthesisers have been demonstrated to be able to run in real-time [245], offering the potential of being integrated into live scenarios such as video games.

The end goal here is to build a general-purpose DDSP synthesiser capable of producing 1) sounds with acceptable time and frequency resolution, and 2) audio of arbitrary length, just by providing conditioning vectors containing the desired parametric controls. The original DDSP synthesiser [145] relies on the premise that the audio it aims to model is harmonic, which is not the case for most sound effects. Very recently, [298] proposed a method to estimate sinusoidal components using gradient descent, which has been a challenging task when using Fourier-based loss functions [299], opening the possibility of modelling inharmonic sinusoids using DDSP synthesisers. Sound effects, however, may contain noisy or very narrow-band elements that are difficult to model using sinusoidal partials [98], plus the method of [298] has yet to be applied to the context of sound effects. Another option could be to use a time-varying finite impulse response (FIR) filter as in the original DDSP subtractive noise synthesiser, but it suffers from a time and frequency trade-off, where, in order to obtain good frequency resolution, the number of taps in the FIR filter need to be relatively high, which in return smears the transients, and vice-versa. This phenomenon is depicted in the middle columns of Figure 7.1, where multiple configurations of the original

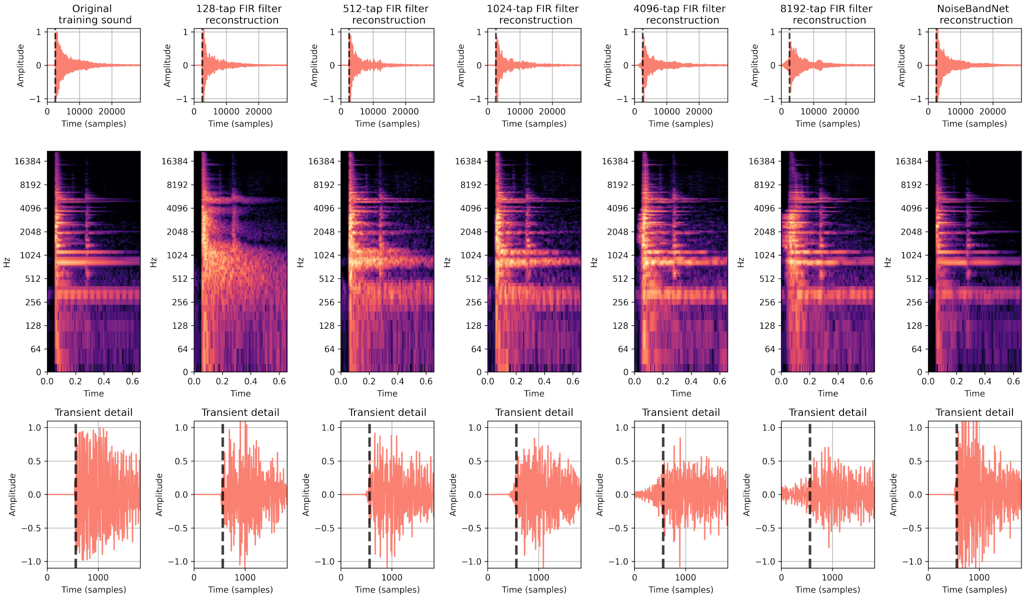


Figure 7.1: Reconstruction task comparison between the DDSP time-varying FIR noise synthesiser and NoiseBandNet. The top row shows the waveform of the entire sound, the middle row its log-magnitude spectrogram and at the bottom a detail of the transient. The transient spot is annotated with a vertical dashed line in the first and third rows. The left column shows the original training sample: a short metal impact. The middle columns show the resulting audio reconstructions of five different configurations of the DDSP time-varying FIR noise synthesiser with 128, 512, 1024, 4096 and 8192 taps respectively, chosen with the aim of covering a range going from sharp transient reconstruction to clear transient smearing, all of them with a hop size of 32 samples. Observe its time and frequency trade-off: the frequency resolution increases with the number of taps at the same time the time resolution decreases, and vice-versa. Using the same hop size for all configurations highlights that a short hop size does not necessarily enhance the transient reconstruction capabilities in filters with more taps, and it simultaneously ensures that the shortest filter configuration (i.e., 128 taps) has at least a 75% overlap (i.e., 32 samples). The right column shows the NoiseBandNet reconstruction using 2048 filters and a synthesis window size of 32 samples, maintaining both good time and frequency resolution.

DDSP [145] time-variant FIR filter synthesiser are used to reconstruct an impact sound effect, which contains a sharp transient. As an alternative, and inspired by the work of [98] where they use multi-rate filterbanks and sub-band processing to overcome the time and frequency trade-off of the inverse FFT method (very closely related to the DDSP FIR-noise synthesiser case), this chapter explores the use of filterbanks in this context, leading to a definition of a new architecture called NoiseBandNet. While sub-band processing is not used in the present work, some of the ideas from [98] are incorporated into a differentiable pipeline, linking human-interpretable control parameters to the output audio. NoiseBandNet is compared to the original DDSP noise synthesiser, establishing a more suitable method to generate time-varying inharmonic sound effects of arbitrary length using DDSP synthesisers, with both good time and frequency resolution. Code and audio examples can be found at the project website.[1]

7.2 Method

The proposed method consists of using a deep learning model similar to the original DDSP architecture, conditioned on high-level audio controls to output M-channel time-varying amplitudes, and multiply them by the M bands resulting from processing white noise through a filterbank. From a high-level perspective, first, a filterbank is built comprised of adjacent FIR filters with narrow frequency responses that jointly cover an arbitrarily wide-ranging frequency spectrum. Then, the filtering operation of a white noise instance with all the different filters of the filterbank is precomputed, “baking” those noise bands in order to ease the computational burden of this approach. Lastly, an architecture similar to the original DDSP paper is used to predict the

[1] https://www.adrianbarahonarios.com/noisebandnet/

time-varying amplitudes of each of the bands for a target dataset, conditioning it on high-level controls, and effectively linking control parameters to the output of the synthesiser. The final output is generated by summing all the bands together in the time-domain.

7.2.1 Filterbank Design

Since the method does not make assumptions about the frequency content of the sound to be modelled, and in order to allow for the synthesis of both narrow and broad frequency components, narrow bandpass filters need to be designed, with the union of their combined frequency responses covering the whole frequency spectrum $[0, \mathrm{Fs}/2]$, where Fs is the sampling rate, which is set to $\mathrm{Fs} = 44.1\,\mathrm{kHz}$. Thus, two adjacent bandpass filters will share one of their two band edges with each other to cover the totality of the frequency spectrum.

The process begins by deciding the number of filters M that will comprise the filterbank to obtain a good frequency resolution, which, based on pilot experiments, is set to $M = 2048$ filters. Second, it is decided how those filters will be distributed across the frequency spectrum. As in [98], the lower end of the frequency spectrum is emphasised by covering those frequencies with more filters than at the higher end. Specifically, half of the filters (1024 in this case, $[1, ..., 1024]$) are employed to cover the first quarter of the frequency spectrum $[0, \mathrm{Fs}/8]$, distributing their center frequencies linearly in the interval. The other half of the filters (the other 1024 filters, $[1025, ..., M]$) cover the remaining interval of the frequency spectrum, $[\mathrm{Fs}/8, \mathrm{Fs}/2]$, with their center frequencies spaced evenly on a logarithmic scale, thus increasing their bandwidth along it (i.e., filters at the higher end of the spectrum have a greater bandwidth than filters at the lower end).

To implement the filterbank, real FIR filters are used, designing them with the Kaiser window method with a transition width Δ_ω of 20% of the filter bandwidth:

$$\Delta_\omega = \frac{|\omega_1 - \omega_2|}{\mathrm{Fs}} \cdot 0.2 \tag{7.1}$$

where Fs is the sampling rate, and ω_1 and ω_2 are the left and right band edges respectively. A stopband attenuation of $A_s = 50\,\mathrm{dB}$ is used. All the filters are bandpass except for the first one, which is a lowpass filter that covers the $[0, f_{\mathrm{min}}]$ interval, with $f_{\mathrm{min}} = 20\,\mathrm{Hz}$; and the last one, which is a highpass filter that covers the $[\omega_1, \mathrm{Fs}/2]$ interval, were ω_1 is the right band edge of the penultimate bandpass filter.

An example of the frequency response of some of the filters is depicted in Figure 7.2. For reference, using the configuration described above, the longer FIR filter in the filterbank has a total of 120 287 taps. The bandwidth of the linearly-distributed (bandpass) filters in the low end is $B_{(1,\ldots,1024)}$ $\approx$5.4 Hz, and the bandwidth of the last (highpass) filter is B_M $\approx$30 Hz. Once the filterbank is built, all the filters' impulse responses are zero-padded to the next power of 2 of the length of the filter with more taps, so they all have the same length. Using the proposed configuration, this results in filters with 131 072 taps. The large length of these filters is due to their very narrow nature.

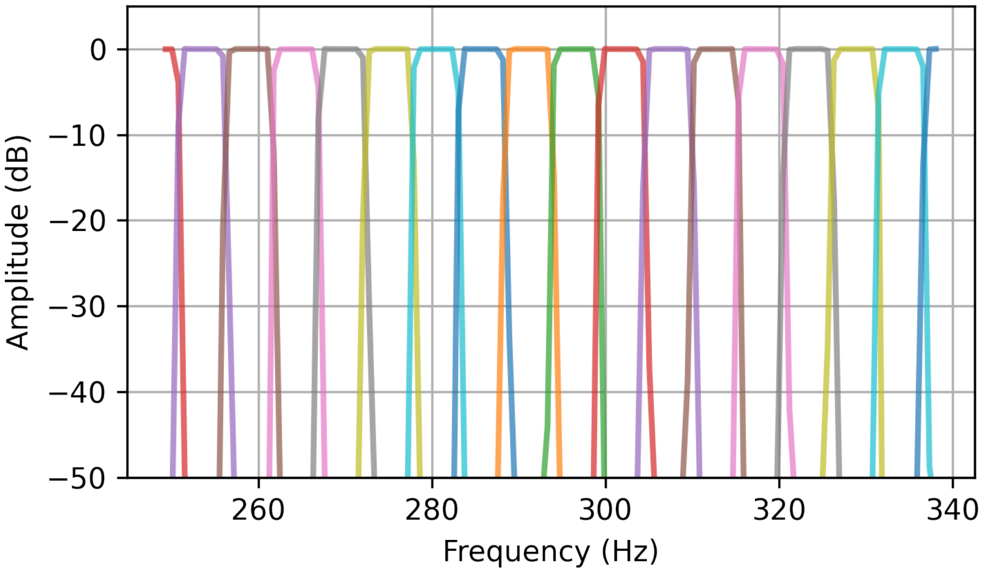


Figure 7.2: Detail of the frequency response of some of the filters employed in a 2048-filter filterbank. Each of the filters is represented by a different colour.

7.2.2 Deterministic Loopable Noise Bands

Considering the system uses many ($M = 2048$) and long ($131\,072-$tap) FIR filters, generating the noise bands themselves (i.e., a convolving a noise instance with all the filters) is a computationally expensive operation, which can bottleneck both the training and inference of the model. This is especially true during training where, at each training step, the noise bands may need to be recalculated; or in longer sequences during inference (e.g., generating 120-seconds' worth of audio).

To ease the computational burden of the system, the technique described in [300] is followed, where they propose a method to extend stationary sounds such as airplane cabin noise, but applying it to the noise bands resulting from filtering a white noise instance with all the filters of the filterbank. The aim is to compute these noise bands only once and store ("bake") them, removing the need of recomputing the operation each time the synthesiser generates an output.

More specifically, their proposed FFT convolution approach is used, leading to sounds that can be concatenated along their x-axis thanks to circular convolution. By the convolution theorem, it is known that convolution in the time-domain is equivalent to point-wise frequency-domain multiplication, which can be written as follows for the filtering of a white noise signal with an FIR filter [300]:

$$Y = R_{\text{noise}} R_{\text{filter}} e^{j(\theta_{\text{noise}} + \theta_{\text{filter}})}, \tag{7.2}$$

with R and θ representing the magnitude and phase respectively resulting from the FFT. Since white noise ideally has a flat magnitude response, they set it to unity $R_{\text{noise}} = 1$, and since the phase of the noise signal, θ_{noise}, already randomises the phase of the operation $(\theta_{\text{noise}} + \theta_{\text{filter}})$, they replace it with a random phase, obtaining the final expression [300]:

$$Y = R_{\text{filter}} e^{j(\theta_{\text{random}})}, \tag{7.3}$$

where θ_{random} is formed by uniformly distributed random values drawn from a $[-\pi, \pi]$ interval and having its first and last values (DC and Nyquist frequencies, respectively) set to 0 [300]. Since the FFT exhibits Hermitian symmetry for real-valued data, the values beyond the Nyquist frequency (the negative frequency values) are just the complex conjugate of the positive ones mirrored from the Nyquist frequency, excluding the Nyquist and DC elements.

Finally, by taking the inverse FFT of Equation 7.3, the “loopable” noise band is created due to the resulting circular convolution operation described above.

Using the proposed configuration, each of the noise bands have a length of 131 072 samples, corresponding to ≈ 3 seconds of audio at 44.1 kHz. A deterministic behaviour is also enforced by setting the same random seed each time a noise band is generated. This is done to 1) maintain coherence each time noise bands are built (i.e., the noise bands used during training and inference will be identical) and 2) being able to share the same noise band instances across multiple models, granting they have the same filterbank configuration. Also, since the amplitude of each of the resulting noise bands may be very small, due to the narrow portion of the frequency spectrum they focus on, the maximum amplitude value $A_{\max}$ across all the noise bands that comprise the filterbank is found, and all the bands are divided by this $A_{\max}$ value, effectively scaling their amplitudes up to what it was found to be a reasonable level. While this leads to neither homogeneous amplitude distribution across bands, nor a normalised amplitude (i.e., in the range $[-1, 1]$), when all the bands are summed together without further intervention, the scale of the individual noise bands will be handled by the time-varying amplitude predicted by the model (see Section 7.2.3).

Thus, by using the method proposed in [300], deterministic and loopable noise bands that only need to be computed once and can be extended arbitrarily in time by just concatenating them along their x-axis, are generated. An example of this is depicted in Figure 7.3, where two instances of the same noise band are concatenated, one after the other. Conceptually, each of those noise bands could be somewhat seen as a wavetable. A wavetable is defined as a block of memory (i.e., a "table") where a discretised signal is stored [38] and, while they are usually employed to store a single period of a waveform, a loopable noise band may be regarded as a period – as it can be looped – of the portion of the frequency spectrum it captures.

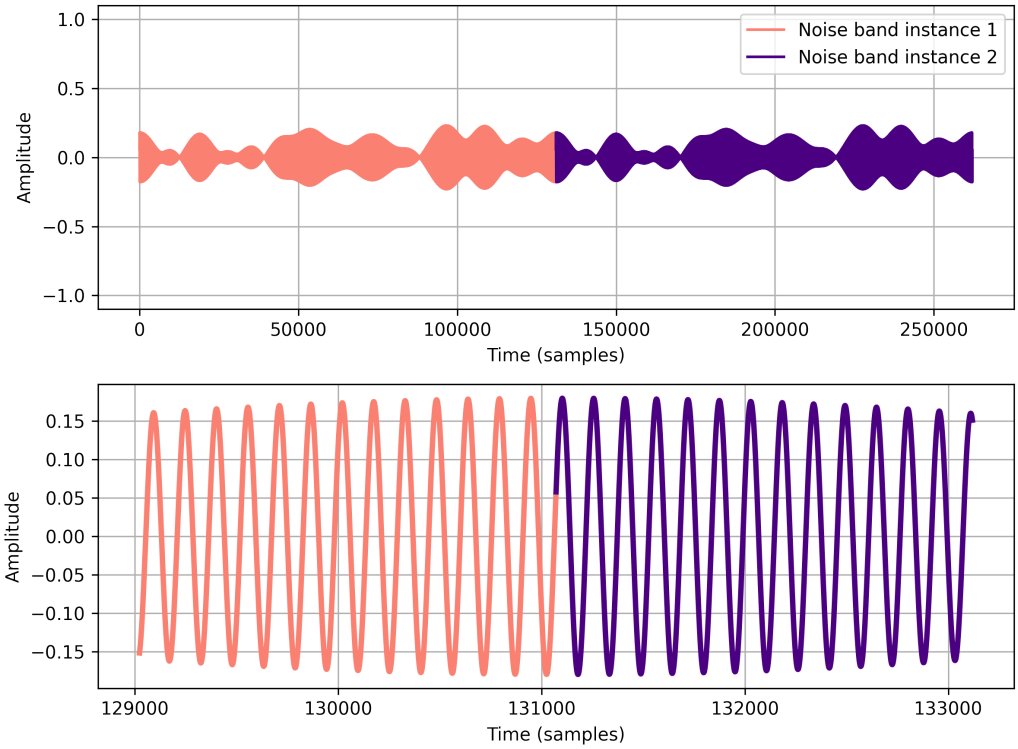


Figure 7.3: Loopable noise bands. Two instances of the same noise band are concatenated along their x-axis. The top figure shows the waveform of both noise bands, one after the other, each one with a distinctive colour. The bottom figure shows the detail of the point where the end of the first noise band instance meets the start of the second one. Notice how, thanks to circular convolution, the start and the end of the segments are "joined up".

7.2.3 Architecture

NoiseBandNet, depicted in Figure 7.4, is built upon the original DDSP architecture [145], but replacing their harmonic-plus-noise synthesiser with a filterbank structure. As in DDSP, the internal sampling rate of the model is a fraction of the target dataset sampling rate Fs. To obtain a good time resolution, and as in [98], a synthesis window size W of 32 samples is selected, granting the model an internal sampling rate of Fs$/W$, thus producing an amplitude value every W samples. Greater W values will lead to poorer time resolution but less computational burden, and vice-versa.

The inputs to the neural network component of NoiseBandNet are the control parameters, which in the Figure 7.4 are loudness and spectral centroid extracted from the training data. These control parameters may be different depending on the control scheme, such as only loudness, or other user-defined controls. Independently of the control scheme, and to synchronise the control parameters to the training data (i.e., to have a $1 : 1$ mapping between the control parameters and the samples in the target audio), originally the control parameters will have the same length as the dataset waveforms, interpolating them to this length if needed. Before passing the control parameter vectors to the network, they are resampled to Fs$/W$, the internal sampling rate of the model.

Similar to [145], the control vectors are passed through a time-distributed multi-layer perceptron (MLP) block (one per control parameter vector and in parallel, as depicted in Figure 7.4) and a gated recurrent unit (GRU) [166]. The output of the GRU is passed through a series of time-distributed MLP blocks leading to final time-distributed dense layer which outputs the

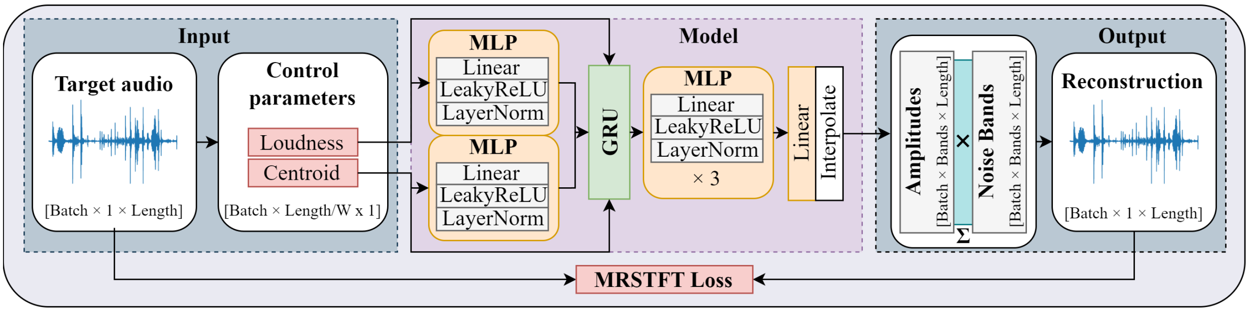


Figure 7.4: Overview of the NoiseBandNet architecture and training process. In this case, loudness and spectral centroid features are extracted from the training audio and passed to the network, which predicts an M-band matrix of time-varying amplitudes at a Fs sampling rate divided by a synthesis window size W. Depending on the control scheme, these features or control parameters may be different (e.g., only loudness or user-provided control parameters). The predicted amplitudes are upsampled using linear interpolation by a factor of W to match the audio length, and multiplied by the M noise bands. The output audio is generated by summing all the bands together. Finally, the model is optimised by comparing the resulting sound against the target audio using a multi-resolution STFT (MRSTFT) loss.

the M-channel time-varying amplitudes (each one of them corresponding to each of the noise bands), with a sampling rate of Fs/W. As in [145], to avoid negative amplitude values the resulting amplitudes are scaled using a modified sigmoid activation function, in this case:

$$y = 2.0 \cdot \text{sigmoid}(x)^{log10} + 10^{-18} \tag{7.4}$$

Then, to bring them to audio rate Fs, these amplitudes are upsampled by a factor of W using linear interpolation. The amplitudes are multiplied by the noise bands in the time-domain, and summed together to produce the final output audio. Unless stated otherwise, mono audio is modeled, with a sampling rate Fs of 44.1 kHz.

7.2.4 Training and Inference

The network is trained on batches of audio chunks of length L_{chunk}. All the training waveforms are concatenated along the time dimension, and random chunks of length L_{chunk} are selected from them. This avoids the network memorising predicted amplitude values to the position of the training examples with respect of time, and so increases its generalisation capabilities when generating longer sequences (especially important when training with small datasets or one-shots). If the training dataset is comprised of a very short ($L_{\text{dataset}} < L_{\text{chunk}}$) training example, it is simply repeated along the x-axis until $L_{\text{dataset}} \geq L_{\text{chunk}}$. As the control parameters have the same length as the audio, the same chunk is selected and resampled to Fs/W before passing them to the network.

Likewise, it is possible that the length of the training chunks L_{chunk} may be smaller than the length of the noise bands L_{bands}. To prevent the network being exposed to portions of the noise bands that were never seen during training, the noise bands are "rolled" along their x-axis at each training step, to a randomised integer shift drawn from a uniformly distributed random value in $[0, L_{\text{bands}}]$, achieving the use of a different, randomised portion of the noise bands at each training step. During training, the output audio is compared against the target audio using a multi-resolution STFT (MRSTFT) loss [185], with the aim of reconstructing the input audio for the given control parameters.

Once trained, the model needs only control parameter vectors of arbitrary length L_{control} to produce an output of $L_{\text{control}} \cdot W$ length in samples. Due to the nature of the architecture used, this output is deterministic (i.e., the model produces the same output amplitudes for the same control parameter input). However, in practice, the output audio resulting from multiplying the noise bands by the same predicted amplitudes may be slightly different since, as described above, the start of the noise bands is randomised by a $[0, L_{\text{bands}}]$-shift, and their energy is not constant over their length (refer to Figure 7.3, where the amplitude of the band fluctuates over time).

7.3 Reconstruction

First, NoiseBandNet is evaluated by comparing its reconstruction capabilities to different configurations of the original DDSP time-varying FIR noise synthesiser [145]. Their synthesiser produces an output by convolving white noise frame-by-frame with an FIR filter predicted by the network and then overlap-adding the frames. As in [145], the FIR filters' impulse responses are not modelled directly, but their magnitudes are instead.

7.3.1 Experiments

To evaluate the reconstruction capabilities of the systems, five sound effect categories relevant to video games which exhibit both broad and narrow spectral components and a wide range of amplitude envelopes are selected [9, 301]:

- Footsteps (≈4.4 seconds): Footsteps on a metallic staircase.

- Thunderstorm (≈14.0 seconds): Rain and close thunder sounds.

- Pottery (≈95.0 seconds): Breaking and scrapping pottery shards.

- Knocking (≈11.0 seconds): Knocking sound effects with different intensities and emotional intentions.
- Metal (≈19.0 seconds): Hitting and scrapping metal bars.

All the training sound effects are sourced from the Freesound website [216], except for the knocking sound effects, where an excerpt of the dataset provided by [301] is used.

Loudness and spectral centroid are used to evaluate the reconstruction capabilities of the systems, as they are related to the original DDSP loudness and pitch control vectors, but without the constraint of being harmonically-oriented. To extract the loudness and the spectral centroid, FFT sizes of 128 and 512 are used, respectively, both with 75% overlap. For each sound category, each of the features is normalised to a $[0, 1]$ range. Note this normalisation is dataset-dependent: the control parameters are not normalised using their their full range (e.g., $[0, \mathrm{Fs}/2]$ in the spectral centroid case), but using the maximum and minimum values computed for the feature across a particular dataset. This prevents feature values being localised to a small portion of the $[0, 1]$ interval for certain sounds (e.g., quieter sound categories would have most of their loudness values close to 0).

A NoiseBandNet model is trained for each of the sound categories using a hidden size of 128 for all layers, a $M = 2048$ band filterbank, and a synthesis window W of 32 samples, following the same design described in Section 7.2. All models are trained for 10 000 epochs using a learning rate of 0.001, batch size of 16, an audio chunk size of 65 536 samples, Adam optimiser, and an MRSTFT loss [185] (with FFT sizes for the MRSTFT of $8192, 4096, 2048, 1024, 512, 128, 32$, 75% overlap, and window lengths of the same size as the FFTs), employing the auraloss implementation [191].

Using an NVIDIA Tesla V100, the training process takes ≈45 min for all models, except for the pottery model, which takes ≈180 min. Once trained, the saved model weights have a size of ≈1.8 MB, with each model having a total of 464K parameters. During inference, the time required to synthesise a single batch signal with an output length of 1 322 976 samples (around 30 seconds of audio at 44.1 kHz) is of 529.5 ± 2.4 (mean ± sd) milliseconds on a consumer GPU (NVIDIA GTX 1060), and 13.4 ± 0.3 (mean ± sd) seconds on a consumer CPU (AMD Ryzen 5 1600), measured on a 100-run test.

NoiseBandNet resynthesis capabilities are evaluated against four variants of the original DDSP model filtered noise synthesiser [145], with a configuration of FIR filter taps of 256 ($\text{DDSP}_{256\text{ taps}}$), 512 ($\text{DDSP}_{512\text{ taps}}$), 1024 ($\text{DDSP}_{1024\text{ taps}}$) and 4096 ($\text{DDSP}_{4096\text{ taps}}$). A hop size of 32 samples is used for all of the models. While such hop size is small for some models compared to a more standard 75% overlap, this value is used to 1) compare NoiseBandNet and DDSP using a configuration that is as close as possible for all systems, and 2) demonstrate that a smaller hop size does not necessarily improve the time resolution for the DDSP time-varying FIR noise synthesiser (refer to Figure 7.1). A hidden size of 128 is used for all of the models, with and a single input MLP per input feature, as in NoiseBandNet (see Figure 7.4). The DDSP noise synthesiser Pytorch implementation found in [253][2] is used, reverb is not modelled, and the same training configuration and loss function as in the NoiseBandNet models are employed.

[2] https://github.com/YatingMusic/ddsp-singing-vocoders

7.3.2 Results

Two objective metrics are used to assess all five models' reconstruction fidelities. First, the MRSTFT loss described above, measured from 19 different values at training time as models converged near their final epochs, to compensate for small possible fluctuations occurring during training. Second, the Fréchet Audio Distance (FAD) [192], a quality metric that correlates to human listeners better than spectral distances, using the implementation found in.[3] MRSTFT loss and FAD results are reported in Table 7.1.

A two-way ANOVA on the MRSTFT loss data with factors for model (five levels, of NoiseBandNet and the four variants of the original DDSP model) and sound effect (five levels of footsteps, thunderstorm, pottery sounds, knocking sound effects and metal sounds) reveals significant main effects of model ($F(4, 450) = 2128, p < .001$) and sound effect ($F(4, 450) = 1217, p < .001$) and a significant interaction ($F(16, 450) = 193, p < .001$), suggesting that the type of model drives differences in loss, so does the type of sound effect, and that certain combinations of model and sound effect lead to either particularly low or high loss values.

[3] `https://github.com/gudgud96/frechet-audio-distance`

Table 7.1: MRSTFT loss (mean ± sd) and FAD results for the reconstruction task. Lower values of MRSTFT loss and FAD are better (best performers highlighted in bold).

	Footsteps		*Thunderstorm*		*Pottery*		*Knocking*		*Metal*		Average values	
	MRSTFT	***FAD***	***MRSTFT***	***FAD***	***MRSTFT***	***FAD***	***MRSTFT***	***FAD***	***MRSTFT***	***FAD***	***MRSTFT***	***FAD***
NoiseBandNet	**1.14±0.01**	**5.41**	**1.24±0.01**	**9.06**	**1.22±0.04**	1.33	**1.10±0.01**	**2.44**	**1.15±0.01**	**4.65**	**1.17**	**4.578**
$\text{DDSP}_{256\text{ taps}}$	1.29±0.01	8.45	1.44±0.01	10.08	1.38±0.02	2.17	1.32±0.01	8.68	1.60±0.01	34.64	1.40	12.804
$\text{DDSP}_{512\text{ taps}}$	1.26±0.01	9.22	1.41±0.02	10.10	1.37±0.02	**1.22**	1.30±0.01	5.17	1.46±0.01	28.75	1.36	10.89
$\text{DDSP}_{1024\text{ taps}}$	1.24±0.01	9.89	1.42±0.01	10.33	1.38±0.03	1.55	1.29±0.01	5.35	1.35±0.01	25.92	1.34	10.61
$\text{DDSP}_{4096\text{ taps}}$	1.22±0.02	7.02	1.42±0.02	9.58	1.39±0.04	2.06	1.32±0.02	4.23	1.27±0.01	15.53	1.32	7.69

An analysis of multiple pairwise comparisons (Tukey's Honest Significant Difference method) was conducted to investigate which pairings of groups differ. It was found that NoiseBandNet significantly outperforms $\text{DDSP}_{256\text{ taps}}$ (mean diff = 0.234, $p < .001$), $\text{DDSP}_{512\text{ taps}}$ (mean diff = 0.189, $p < .001$), $\text{DDSP}_{1024\text{ taps}}$ (mean diff = 0.164, $p < .001$) and $\text{DDSP}_{4096\text{ taps}}$ (mean diff = 0.150, $p < .001$). Thus, NoiseBandNet obtains significantly better MRSTFT reconstruction values compared to the variants of the original DDSP noise synthesiser. In terms of the DDSP model variants' performance across the different sound effect categories, they were most effective for footsteps, followed by knocking, pottery, and thunder, with relatively small differences in performance between variants. This was in contrast to the metal category, where DDSP variants displayed relatively large differences in performance.

The FAD results follow a very similar pattern to those for MRSTFT, with the exception of $\text{DDSP}_{512\text{ taps}}$, which performs better than NoiseBandNet in terms of FAD for the pottery sound effect category. This discrepancy may be caused by the small size of the datasets, which negatively affects the accuracy of the metric [192].

7.4 Creative Uses

As a second evaluatory perspective on NoiseBandNet, this section highlights some potential creative uses to which NoiseBandNet can be put.

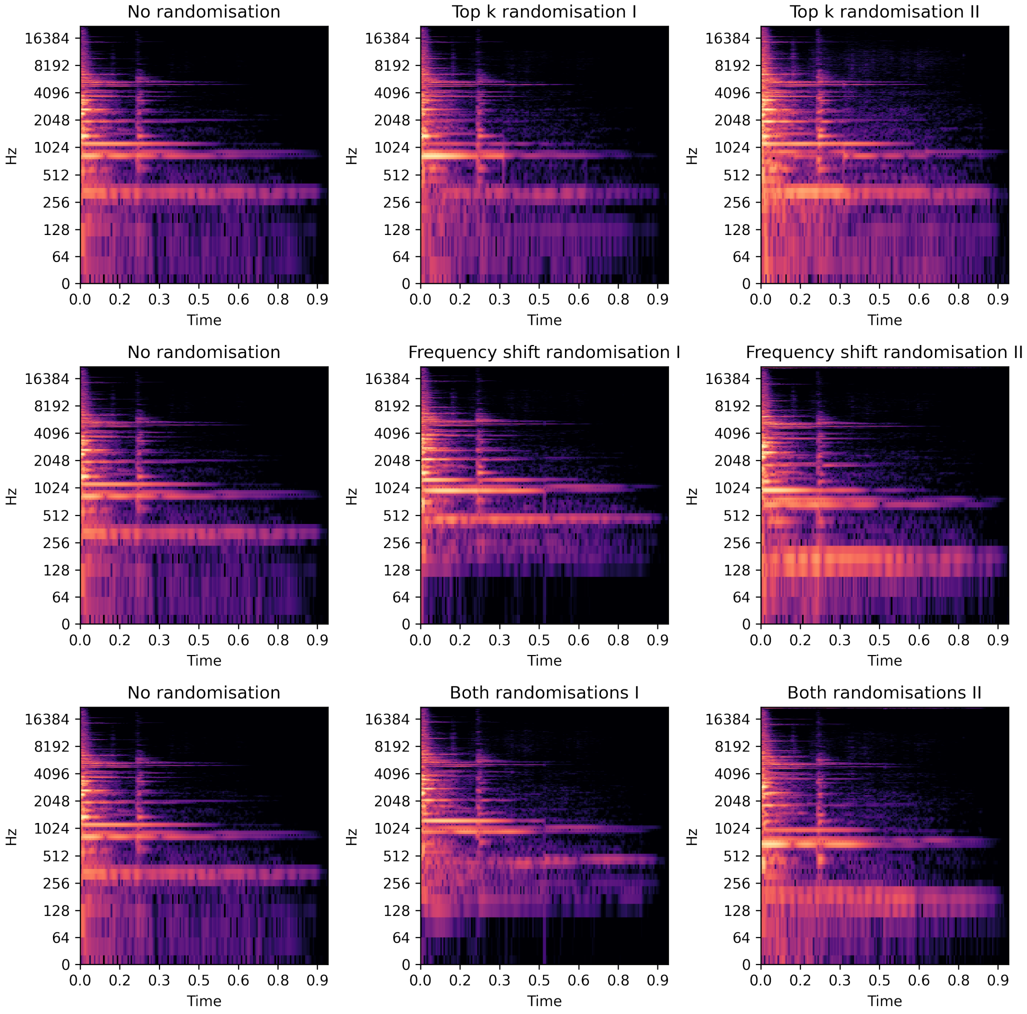


Figure 7.5: Log-magnitude spectrograms from the result of the different randomisation schemes. The left column represents a non-randomised (just reconstructed) sound: a metal impact. The second and third columns show two examples of the resulting randomised sound. In the first row the top k randomisation scheme is employed, using $L_{\text{frame}} = 430$ (3 frames), $k = 100$ and a randomised multiplier in a $[0.0, 1.0]$ range. The second row depicts the frequency shift randomisation scheme with $L_{\text{frame}} = 645$ (2 frames), $f_{\text{init}} = 30$ and $f_{\text{shift}} = 3$. The third row shows both randomisation combined, using $L_{\text{frame}} = 645$ (2 frames), $k = 100$, a $[0.0, 1.0]$ multiplier, $f_{\text{init}} = 30$ and $f_{\text{shift}} = 3$.

7.4.1 Amplitude Randomisation

Given that NoiseBandNet uses DSP components at its core to produce audio, their inherent biases can be exploited to further alter the output signal. As outlined in Section 7.2.4, the output amplitudes of the model using the proposed architecture is deterministic. Here, as an example, two strategies to generate variations from the predicted time-varying amplitudes are presented. This may be especially relevant to game audio, where it is common to use multiple audio clips to sound design the same in-game interaction in order to avoid repetition [5]. The proposed randomisation scheme is somewhat conceptually similar to the randomisation scheme used in Chapter 4 with the filter-based modal synthesiser, where the properties of the filters were modified for each sound in order to produce different variations between impacts.

First, it is proposed to randomise the top k amplitudes k_{amp} within a desired frame length L_{frame} (i.e., the output amplitudes are randomised each L_{frame} amplitude values). To achieve this, first the frame length L_{frame} is selected and the output amplitudes are split to non-overlapping frames of that length. Note these frames are split before performing the linear interpolation operation that upsample the amplitude values to audio rate. Then, the desired top amplitudes k_{amp} on each frame are found by summing the amplitude values for each of the bands on that frame across the time-axis, selecting the greater k values. After that, a randomised amplitude modifier in a user-defined range of $[mult_{\text{min}}, mult_{\text{max}}]$ is applied by multiplying the amplitude values on that frame by it, scaling them up or down. Since all amplitudes still need to be interpolated to audio rate, the transition between their values is smoothed. It was also found that, if a different amplitude randomisation for the same amplitude output is computed, stereo sequences can be generated by panning them left and right, as the resulting signal, in combination to the variation introduced by the band shift explained in Section 7.2.4, will be slightly different for relatively small $[mult_{\mathrm{min}}, mult_{\mathrm{max}}]$ values.

Second, another strategy is proposed to perform frame-wise pitch-shift on the output amplitudes. Again, a desired frame length L_{frame} is selected and the output amplitudes are split to non-overlapping frames of it before the linear interpolation operation. Within that frame, all the amplitude values are "rolled" to a randomised integer value $[-f_{\mathrm{shift}}, f_{\mathrm{shift}}]$ on their band-axis, effectively transposing the bands from one amplitude to another. The previous f_{shift} values are taken into account to compute the current shift, implementing a process somewhat similar to a random walk. An initial frequency shift f_{init} that rolls all the amplitude values in a randomised $[-f_{\mathrm{init}}, f_{\mathrm{init}}]$ range is also allowed, which effectively transposes the entire sound. Likewise, the subsequent linear interpolation operation to audio rate will provide a relative smooth transition between shifts.

An example of both schemes is depicted in Figure 7.5, showing the top k randomisation in the first row, the frequency shift randomisation in the second row and both randomisation schemes applied together in the third row.

7.4.2 Loudness Transfer

In [145] they were capable of performing timbre transfer (i.e., transferring the pitch and loudness of an incoming audio to the instrument the model is trained on) using just 13 minutes of expressive solo violin performances. However, unlike harmonic sounds that are constrained by a discretised and well-defined pitch, in Section 7.3.1 spectral centroid is used as an alternative control vector for inharmonic sounds (such as in most sound effects), thus introducing a higher degree of freedom to the control parameters. Considering the deterministic nature of the output amplitudes from the model, and since obtaining enough expressive data to represent all possible loudness and spectral centroid and interactions may be challenging in the context of sound effects, a control scheme is employed here such that it only relies on one of the features: loudness. The aim is to transfer the relative loudness envelope of one sound to another, training the network in the latter and using the extracted loudness envelope of the former during inference. This is possible due to loudness being mathematically defined (i.e., it can be extracted programmatically) and normalised to a $[0, 1]$ range relative to the training and inference data, as described in Section 7.3.1, thus covering the full loudness range regardless of the training data.

To demonstrate the loudness transfer capabilities of the model, the same training procedure than in Section 7.3.1 is followed, but using loudness as the only control parameter. Three different models are trained on the following short sounds: a metal impact (≈1.0 second), the Wilhelm scream (≈1.2 seconds), and an electric drill sound (≈3.4 seconds). Due to having a single control parameter and therefore a single input MLP, the trained models are slightly smaller than the ones trained on two control parameters. More specifically, they have a total of ≈414K parameters (as opposed to 464K) and their weighs a size of ≈1.6 MB (as opposed to ≈1.8 MB). For reference,

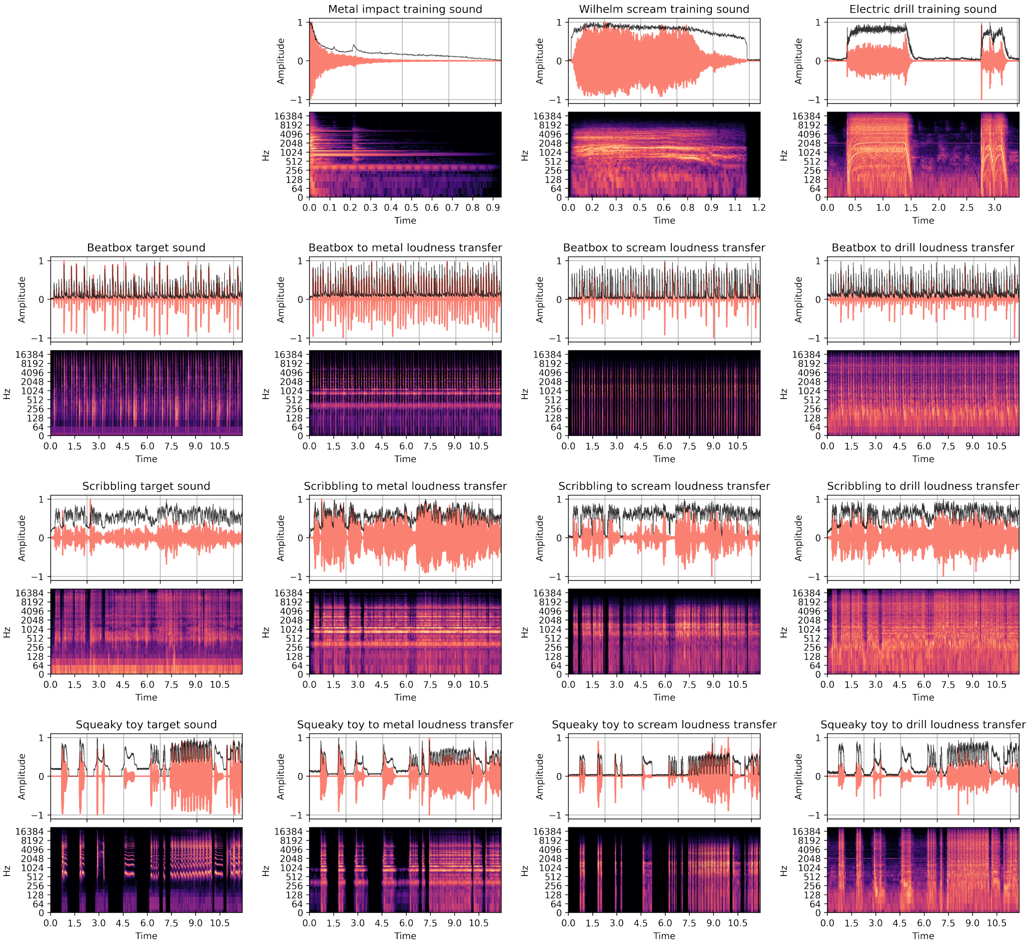


Figure 7.6: Waveforms (top) and log-magnitude spectrograms (bottom) pairs resulting from the loudness transfer experiments. The extracted loudness of each sound is represented in black, superimposed on the waveforms in a $[0, 1]$ range. The first row depicts the three sounds used to train each of the models: a metal impact, the Wilhelm scream. and an electrical drill sound effect. The first column contains the sounds used for transferring their loudness envelopes. Starting from the second row, the second, third, and fourth columns contain the loudness transfer results for the different sound combinations.

the MRSTFT reconstruction loss for the different models is of 1.04 ± 0.01 for the metal impact model, 1.09 ± 0.01 for the Wilhelm scream model and 1.17 ± 0.01 for the drill model using the same objective as in Section 7.3.1.

Another three sounds are then chosen to transfer their loudness envelopes to all the trained models: a rhythmic beatbox sound effect, scribbling using a pencil onto paper sounds, and a squeaky toy sound effect. All the training and inference sound effects are collected from the Freesound website [216]. The loudness transfer is performed by simply extracting the loudness from the target sounds (beatbox, scribbling and drill in the examples) computing it using the same procedure described in Section 7.3.1 (including normalising it to a $[0, 1]$ range), and using the resulting vector (interpolated accordingly to the internal Fs/W sampling rate) as the input to the trained models (metal impact, Wilhelm scream and electric drill). A 2^{19} sample length excerpt from the target sounds ($\approx$12 s at 44.1 kHz) is chosen, and the operation described above is applied.

While both the loudness of the target and trained models sounds are normalised in a $[0, 1]$ range, it may occur that most (or the most perceptually relevant) of their values are contained on an specific interval, and outliers distort it. To solve this potential issue, and to allow for a finer control over the output of the model, a user-defined scale modifier is applied to the loudness values. In the experiments, the following modifiers were applied to the input loudness values of the sounds depicted in Figure 7.6: metal impact ($+0.1$ on beatbox, -0.1 on scribbling, -0.1 on squeaky toy), Wilhelm scream ($+0.3$ on beatbox, $+0.15$ on scribbling, no modification on squeaky toy), electric drill ($+0.25$ on beatbox, no modification in scribbling, -0.1 on squeaky toy).

The result from the experiments is depicted in Figure 7.6. It can be discerned how the target loudness envelope, depicted in the first column is successfully transferred to the trained models, depicted in the second, third and fourth columns, starting from the second row. It is also noticeable how the frequency content of the resulting transferred sounds is time-varying, changing over time depending on the input control parameter.

7.4.3 Training and Synthesis using User-Defined Control Parameters

Since loudness (or spectral centroid) curves may be challenging to control and interpret from a user-perspective, or may not be adequate for the potential use-cases of a particular model, here the training on user-provided control parameters is explored, taking inspiration from the Wwise audio middleware Real-Time Parameter Controls (RTPCs).[4] RTPCs can be used to attach in-game parameters (e.g., the speed of a car) to sound properties (e.g., the pitch of the engine), linking game events to sound control curves. In this scenario, by anticipating the creative use of the synthesiser, a user (e.g., a sound designer) may directly draw the desired control curves used during training and, once trained, use them to control the output of the model.

To this end, a graphical user interface is designed to manually label the data based on potential control curves. The tool is depicted in Figure 7.7, containing a waveform of the sound to be modelled on top and its spectrogram at the bottom. By clicking on top of the spectrogram, a user can manually draw the control curve. Once drawn, this curve is normalised to

[4] https://www.audiokinetic.com/en/library/edge/?source=SDK&id=concept_rtpc.html

[0, 1], in preparation for use with the model. The same three sounds used in Section 7.4.2 are chosen, envisioning a user who wishes to control the following aspects with their control curves: for metal impact, the curve might control impact force; for the Wilhelm scream, the curve might control scream intensity; for the electric drill, the curve might control drill power. The three models are then trained with those hand-drawn control curves as their single input control parameter. For reference, in this case the MRSTFT reconstruction loss for the different models is of 1.05 ± 0.01 for the metal impact model, 1.01 ± 0.01 for the Wilhelm scream model and 1.23 ± 0.01 for the drill model, using the same objective and configuration as in Section 7.3.1.

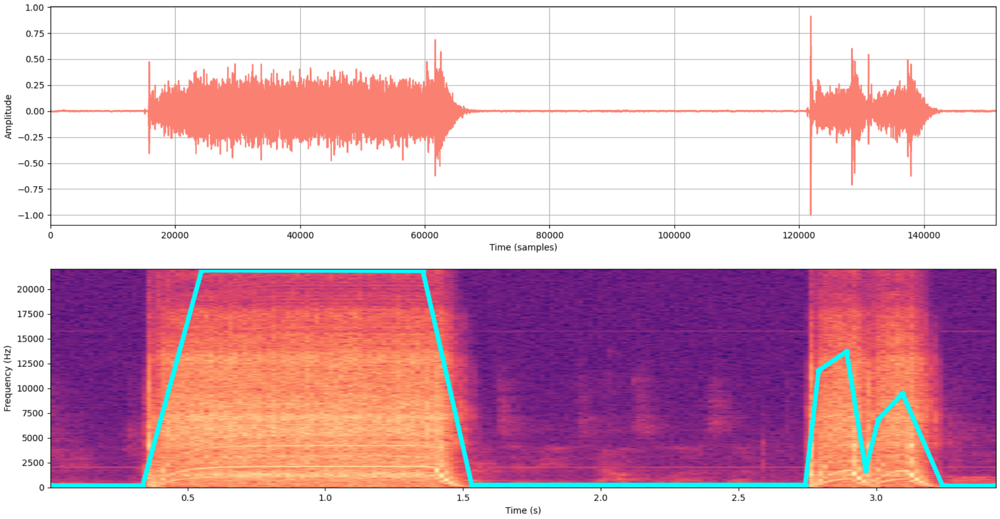


Figure 7.7: Graphical user interface used to manually label the data. The image depicts the waveform (top) and the magnitude spectrogram (bottom) of an electric drill sound. The cyan line on top of the spectrogram is the hand-annotated control curve.

To control the synthesiser, a corresponding UI tool for drawing the inference control parameters is provided. The control tool functions exactly as the tool depicted in Figure 7.7, but now the user has a blank canvas to draw their desired control curve for driving the synthesiser. Three hand-crafted curves per model, with a length of 2^{14} each, are drawn (second, third, and fourth columns of Figure 7.8). Since the output amplitudes are upsampled to audio rate interpolating them by a factor defined by the synthesis window size $W = 32$, the output signal length is of $2^{14} \cdot 32 = 524\,288$ samples or ≈12 seconds at 44.1 kHz. The results of the experiments are depicted in Figure 7.8. The first column contains the original sounds along their user-defined control curves used during training, represented in black on top of the waveforms. The subsequent columns are the synthesised sounds resulting from using the new user-defined inference curves, depicted also in black on top of their waveforms. For each sound category, it can be seen that the resulting audio is broadly consistent with the intended control curve.

7.5 Chapter Summary

The combination of DSP components (such as in Chapter 4) alongside deep learning techniques (such as in Chapters 5 and 6) for the synthesis of sound effects is a method that has shown promise in recent years, at least where assumptions hold regarding the harmonicity of the sounds being modeled [145]. Harmonic sounds represent only a portion of sound effects, however, and so it remained an open problem how to model and then synthesise *arbitrary sounds* with acceptable time and frequency resolution, and of arbitrary length.

The contribution of this chapter is to address the modelling and synthesis of arbitrary sounds using DDSP synthesisers, tackling both the problem of reconstruction fidelity (see Section 7.3) and exploring some of the creative uses

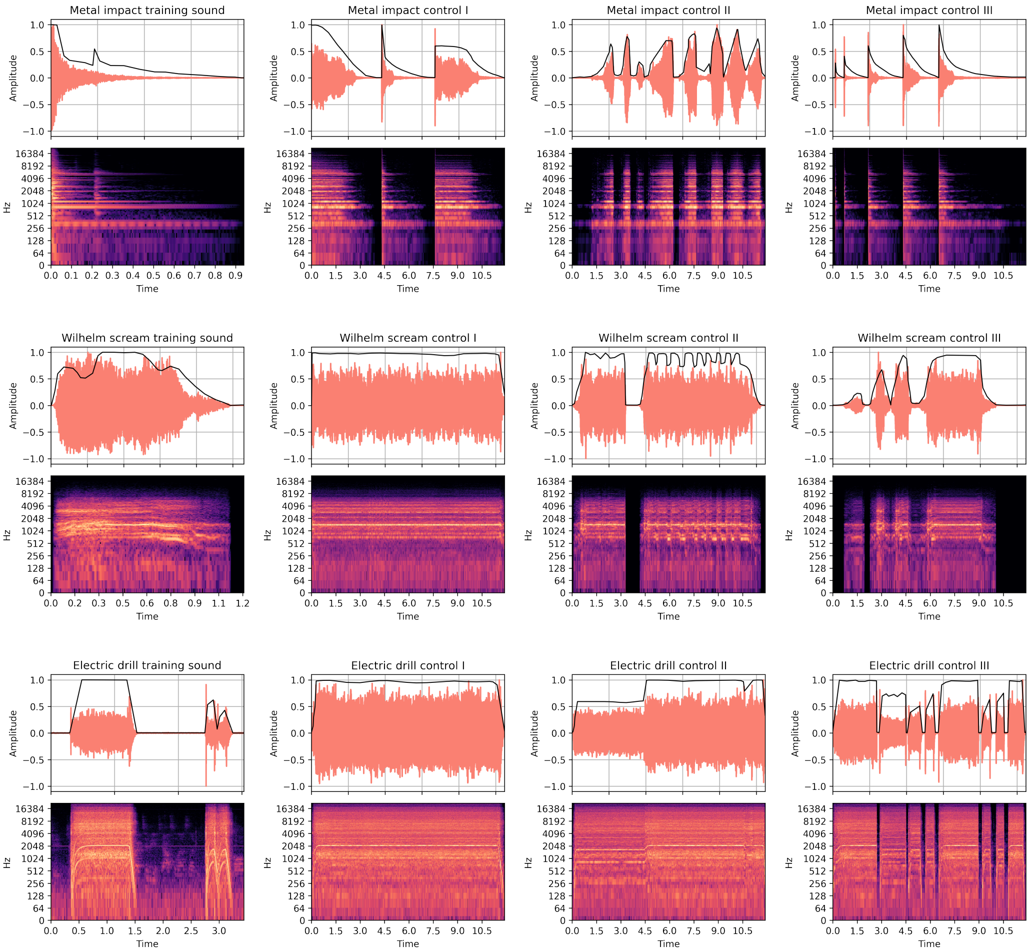


Figure 7.8: Waveforms (top) and log-magnitude spectrograms (bottom) pairs resulting from the training on user-defined control experiments. The training sounds are depicted in the first column, with their user-defined training control parameters represented in black superimposed on the waveforms in a $[0, 1]$ range. The second, third, and fourth columns contain the sounds generated by using user-defined inference curves for the three models, each one in a different row. The user-defined inference control curves are also represented in black on top of the waveforms in a $[0, 1]$ range.

(see Section 7.4). The proposed solution is encapsulated in a model called NoiseBandNet, an architecture capable of synthesising continuous sound effects conditioned on high-level parametric controls with consistently good time and frequency resolution. The use of filterbanks to shape white noise is proposed, establishing a suitable approach towards modelling non-musical or inharmonic sound effects using DDSP synthesisers. NoiseBandNet is also lightweight and can be trained on very limited data (≈1 second of audio), as shown in the experiments. The potential creative uses of the architecture are also highlighted by generating sound variations, performing loudness transfer, and training and synthesising audio with user-defined control parameters.

NoiseBandNet is evaluated against four configurations of the original DDSP filtered noise synthesiser [145], and it was found that NoiseBandNet significantly outperforms all DDSP variants on the task of resynthesising sounds from different categories, for nine out of ten (sound category, evaluation metric) combinations – the exception being for the metric of FAD on pottery sounds. In addition to overall better reconstruction capabilities compared to the original DDSP noise synthesiser, the proposed filterbank is not constrained by having its frequency response distributed linearly, such as in the case of a time-varying FIR filter. Thus, both the number of filters and their distribution across the frequency spectrum is flexible and can be altered to accommodate other use cases.

Taking inspiration from current game audio workflows, the creative possibilities of NoiseBandNet are also outlined through a series of experiments, providing examples of amplitude randomisation, automatic loudness transfer and training models using user-defined controls. The code employed to generate those sounds alongside the audio examples described throughout the chapter can be found in the accompanying material at the project website.[5]

While a filterbank configuration with a higher frequency resolution on the low end (broadly inspired by [98]) which provided satisfactory results on pilot experiments was used, the design could be further improved by considering auditory perception, for instance increasing the emphasis between 500 and 4000 Hz, where the sensitivity of frequency changes to pure tones is higher [302]. Apart from the effect of the number of filters and their distribution on synthesis quality, it is also planned to explore the use of alternative loss functions, such as the differentiable joint time-frequency scattering (JTFS), used recently in the context of audio classification with promising results [187].

Since the proposed noise band structure is not tied to the network architecture itself, for future work it is aimed to use noise bands with other approaches. For instance, the architecture could be replaced with a more lightweight temporal convolutional network (TCN) [303], which has been successfully employed to model audio effects [265] and in differentiable FM synthesisers [251]. Another option may be using adversarial training [155] or a variational autoencoder (VAE) [149, 150], which opens up the possibility of non-deterministic behaviour. Additionally, NoiseBandNet could be applied to harmonic and musical signals, replacing the original DDSP noise synthesiser, or potentially in combination with it when sounds contain inharmonic partials (e.g., training using an approach derived from [298]).

Despite the saved model weights being small in size ($\approx$1.8 MB and $\approx$1.6 MB for two and one control parameters, respectively), the size of the noise bands

[5] `https://www.adrianbarahonarios.com/noisebandnet/`

is large (≈1 GB on disk for the proposed configuration), due to their the number and length. However, as described in Section 7.2.2, thanks to the deterministic nature of the noise bands when using the same filterbank configuration, a single instance can be used across multiple models, thus only needing to create a single set of them. Nonetheless, to further optimise the model size and alleviate the computational burden involved in multiplying the output amplitudes from the model with the noise bands in the time-domain, it is planned to investigate the use of neural audio codecs such as Encodec [304], and multi-rate filterbanks and sub-band processing as in [98]. As it was reported in Section 7.3.1, the offline generation on a consumer GPU is fast (≈529.5 milliseconds to generate 30 seconds of audio), but it is much slower on CPU (≈13.4 seconds to generate the same length). While more research needs to be conducted to address these points, it is hypothesised that a combination of architectural changes (such as the use of TCNs), a more efficient auditory-informed filterbank configuration, and the use of neural audio codecs and sub-band processing as described above may result in faster generation, which is especially relevant for real-time synthesis in the context of game audio.

Regarding the automatic extraction of control parameters from the audio, above (Section 7.3.1) loudness and spectral centroid are used, computed using DSP methods. Other approaches, such as [259], develop highly engineered solutions to extract control parameters from the target audio, such as engine RPM in their case. It is desirable, however, to accommodate a wider range of sounds and use cases. While a first approach could be the use of sound event detection to extract similar sounding clips from longer signals in data-scarce scenarios, such as in [305], achieving the potential granularity required to successfully label continuous data (e.g., drill power in the examples) may be challenging. Another direction, inspired by the recent proliferation of text-to-audio models such as [235, 236], could be the generation short audio clips catered towards being controlled by a model such as NoiseBandNet. For instance, a text-to-audio model could be prompted to generate a drill sound effect with a linearly increasing drill power, and the output audio could be used as the input to NoiseBandNet alongside a linearly increasing control parameter vector going from $[0, 1]$ (minimum and maximum drill power values), granting the text-to-audio model successfully renders a sound with those properties.

It is acknowledged that while three different experiments exploring the creative uses of the architecture are presented, these could be expanded and evaluated in a user study. Future work will comprise carrying out a study with audio experts to evaluate the creative possibilities of the model, and the plausibility of the synthesised sounds. The study will also inform the amount and type of data needed to satisfactorily accomplish a control task, and the feasibility of training with multiple user-defined control parameters (e.g., a weather audio model with both "rain and thunder intensity" control curves). Since NoiseBandNet uses DSP components (time-varying amplitudes applied to filters) that audio experts are familiar with, it is also planned to evaluate and expand the randomisation schemes outlined in Section 7.4.1. Ultimately, the aim is to understand how the model introduced in this chapter could affect the workflows of sound designers and, more generally, audio experts in years to come when using controllable neural audio synthesisers in the context of game audio.

CHAPTER 8

CONCLUSIONS

8.1 Discussion

How high-fidelity sound effects can be generated 1) automatically or 2) with dynamic or creative control where necessary or desired – all without compromising the quality or plausibility of the output audio – is a topic of interest to the fields of sound design and game audio [9, 5], psychoacoustics [63], and extended reality [8]. However, the efficient creation of plausible and/or controllable sound synthesis models for the generation of sound effects remains an open research problem [9, 16, 17, 18, 8]. This thesis explores methods to solve these problems by surveying the suitability of DSP approaches for certain sounds, by employing novel deep learning architectures for general sound effect synthesis, and by using a combination of DSP and deep learning methods for the controllable synthesis of arbitrary sound effects.

First, Chapter 4 addresses the first research question (RQ 1): **"To what extent can DSP-based procedural audio techniques be applied to video games in terms of perceived plausibility?"**

To this end, Chapter 4 assesses the perceived plausibility of filter-based modal synthesis [56, 22] for the synthesis of sound effects. Filter-based modal synthesis is a DSP-based subtractive synthesis approach derived from physical-inspired synthesis, especially suited for the synthesis of percussive sounds. A diverse set of objects from four different types of materials (ceramic, glass, metal and wood) that may be relevant to video games and that are suitable for the considered synthesis method (i.e., they exhibit clear modes when struck) is selected. Aiming to synthesise impact sounds, procedural audio models are constructed for each object, followed by an evaluation of their perceived plausibility in a listening study. Results show that for the sounds considered, presented in isolation, recorded and synthesised sounds are indistinguishable from each other. In other words, all participants of the listening study were unable to detect the synthesised sounds, and this fact did not depend on the expertise in sound design of the listener. To emphasise the suitability of the synthesis method in the context of game audio, an intractable real-time implementation of the evaluated procedural audio models was also presented within a virtual environment. Thus, it is demonstrated that, from a perceptual point of view and for a certain subset of sounds and interactions, filter-based modal synthesis is a suitable method to create procedural audio models in the context of video games, hence addressing the first research question (RQ 1) of this thesis. As outlined in Section 4.6, this chapter focuses on a particular category of sounds (time-invariant impact sounds of objects that exhibit clear modes when struck), which only represent a small percentage of sound effects. Hence, driven by recent advancements in the field of deep learning that have surpassed “classic” DSP methods (see

Section 3.3), the next chapters of the thesis explore data-driven techniques. These methods are not necessarily bound by specific sound characteristics, and offer opportunities for novel interactions and control schemes.

Second, Chapters 5 and 6 address the second research question (RQ 2): **"How can novel deep learning techniques be applied directly to the synthesis of sound effects?"** They do so by describing how one can both: a) synthesise sound effects with class conditioning by training on a dataset, and b) generate variations from single sound effects using deep learning methods.

Chapter 5 focuses on the class-conditional synthesis of sound effects. That is, by learning from a dataset of sounds belonging to different classes (e.g., emotional intentions in knocking sound effects), a generative deep learning model is capable of synthesising novel sounds, controlling which of those classes the synthesised sound belong to. To achieve this task, a conditional WaveGAN architecture [200, 223] is used, training it on a dataset of knocking sound effects with emotional intentions commissioned to a professional Foley artist [282]. The plausibility and the intended and perceived emotion of the synthesised sound effects are evaluated in a listening study. Results show that, while synthesised sounds can be identified by participants with expertise in sound design, they are not far from being indistinguishable from the pre-recorded samples by non-experts. The intended and perceived emotion labeling by participants in both the recorded and synthesised sounds is similar, being all of them correctly labelled, except for anger and fear, which are confused with each other. A series of acoustic features extracted from the recorded and synthesised sounds are also compared. These features are also similar in both groups of sounds, suggesting that they may have an effect on the perception of emotions in knocking sound effects.

To address the challenges derived from likely data-scarce scenarios in the context of sound effects modelling, Chapter 6 focuses on training a generative deep learning model using just a single sound. To achieve this single-image GANs [290, 292] are used, bringing them to the audio domain and encapsulating the approach in an architecture called SpecSinGAN. Inspired by game audio workflows, multi-channel log-magnitude spectrograms are used to train the network on the different layers that comprise a sound effect, synthesising variations from them. The approach is evaluated in a listening study against pre-recorded samples and DSP-based procedural audio model variants [16], assessing their plausibility and variation. Results demonstrate that SpecSinGAN significantly outperforms the procedural audio model variants, for both evaluation metrics. SpecSinGAN is not as plausible or varied as pre-recorded samples, however.

Third, Chapter 7 addresses the third research question (RQ 3): **"To what extent can a combination of DSP-based techniques and machine learning methods be applied to the synthesis of sound effects?"**

Chapter 7 focuses on the controllable time-varying synthesis of sound effects. To achieve this, a network is trained to predict the time-varying amplitudes of a series of narrow noise bands produced by filtering white noise through a filterbank of adjacent FIR filters, conditioning it on time-varying control parameters. The method, encapsulated in an architecture called NoiseBandNet, effectively links those input feature vectors – representing control parameters – to the synthesised audio. The reconstruction (i.e., resynthesis) capabilities of the approach are evaluated against a baseline state-of-the-art differentiable digital signal processing architecture [145], for five categories of sound effects. Results show that this method outperforms the baseline in objective evaluations, achieving both good time and frequency resolution.

To emphasise the usability of the approach, a series of creative experiments were performed. These experiments allow prospective users to generate variations, perform loudness transfer or, inspired by game audio workflows, train and synthesise audio according to user-defined control curves. Thus, Chapter 7 addresses the third research question (RQ 3), by using DSP components (FIR filters, time-varying amplitudes) alongside deep learning to synthesise sound effects.

8.2 Assumptions and Limitations

It is assumed that the third-party software tools (e.g., automatic differentiation libraries or DSP components) utilised throughout the development at various stages (e.g., analysis, training, or inference) of the algorithms discussed in this thesis are reliable and stable, yielding the anticipated outputs. It is also assumed that the participants responded honestly to the listening studies conducted as part of this work.

In addition to the specific limitations outlined in each chapter, the scope of the studies conducted in this thesis adheres to what is described in Section 1.2, thus there are certain areas that are not prioritised, such as machine learning operations (MLOps), inference time optimisation (including real-time generation), or user experience design. Similarly, within the scope of this thesis, each chapter comprising this work considers a single solution to its problem statement. However, while multiple approaches (e.g., different architectures) could lead to similar solutions, they are not explored due to time constraints.

8.3 Future work

Throughout this thesis, sound has been modelled with deep learning methods using multiple audio representations. Specifically, sound has been modelled in the waveform domain (Chapter 5), using phaseless log-magnitude spectrograms (Chapter 6), and DSP components (Chapter 7). While, as outlined in Section 3.3, those are arguably the three most common representations (i.e., raw audio, spectral, DDSP-based), other forms of audio representation may also be desirable. For instance, especially relevant for the methods considered in Chapters 5 and 6 that address the second research question (RQ 2) may be, respectively: the use of alternative waveform representations, such as the discrete representations derived from neural audio compressors like [306], which have been used already in the context of text-to-audio generation in [235]; or alternative time-frequency representations such as the wavelet-based joint time-frequency scattering [186].

Regarding the use of alternative DSP representations, those will depend on the differentiable DSP systems considered. An option could be to employ bespoke procedural audio models, such as proposed in [9, 16], and to use a gradient descent optimisation process in order to match a target sound (e.g., a physically-informed footsteps model trained on real footsteps sound effects, retaining the physical information of the system).

Considering the quality – or plausibility – of the generated audio, it was found that the methods considered in Chapters 5 and 6 still lack plausibility when compared to pre-recorded samples. However, while in those chapters GANs are employed, more recent diffusion models (see Section 3.2.5) can be considered for future work, as they have shown promise in the field of computer vision, outperforming GANs in terms of quality [179]. Nonetheless, in Chapters 4 and 7 it was shown that DSP components offer exceptional synthesis capabilities from a plausibility point of view. Thus, the exploration of further DDSP-based neural audio synthesisers is left as a desirable direction for future work.

In respect of the control schemes, this thesis has explored the synthesis of variations from user-provided sound assets (Chapters 4, 6 and 7), class-conditional synthesis (Chapter 5), and user-defined time-varying parametric controls (Chapter 7). In a recent survey conducted with film audio professionals, when questioned about the desired control scheme when using generative deep learning methods, most of the participants were interested in generating samples from a user-provided reference audio file [307]. This supports the ideas behind Chapters 6 and 7, where users are able to train models with very limited user-provided data (e.g., a single sound) to synthesise variations or build controllable synthesisers, thus establishing a clear user-centric direction to take into account for future work.

Another point to consider, especially relevant to game audio, is the time required to produce an output (i.e., the synthesis turnaround time – either real-time or not). Chapter 4 demonstrates the implementation of real-time DSP-based procedural audio models in a virtual environment. However, in

Chapters 5, 6 and 7), the sound is generated offline. Nonetheless, similar neural audio synthesis approaches have been demonstrated to run in real-time [278, 245, 206]. Thus, a desirable future direction would be to adapt and implement the methods introduced in Chapters 5, 6 and 7 so they can run in real-time, which, in the context of game audio, could be done natively either in a game engine or as an audio middleware plugin; or as a DAW VST as in [244].

Another factor to consider is the location of the device used to generate the sounds, which can either be remote (on the server-side) or local (on the client-side). While remote generation may be compatible with offline systems since there are no latency burdens or immediate output expectations, online generation (i.e., real-time sound synthesis) may be challenging on a server-side configuration, due to latency constraints. However, with the continuous enhancement of bandwidth speeds [308] and the development of high-quality neural audio compressors (e.g., [306, 304]) capable of significantly reducing the required data packages for sound transmission, real-time sound synthesis on the server-side without latency burdens becomes feasible. This concept is already being used in cloud gaming solutions, which run the entire game on the server-side and stream its content at a rate of 44 Mbit/s to end users [309]. Client-side generation relies on both the available hardware resources and the computational demands of the model. As mentioned earlier, while certain neural audio synthesisers are capable of real-time execution on a CPU (e.g., [278, 245, 206]), other models may be too computationally intensive to run locally, especially if the resources, in the context of game audio, are allocated to other processes (e.g., graphics rendering or physics simulation).

With regard to the field of knowledge itself, as outlined in Chapter 3, most previous research in neural audio synthesis has focused on either the speech or music domains. Thus, while some of the ideas and concepts can be carried over to the topic of study of this thesis, the synthesis of sound effects lags behind its other audio counterparts. There is, however, a growing interest in this topic. For instance, the emergence of LLMs and text-to-audio architectures such as [234, 235, 236, 237, 238] may put the topic of the synthesis of sound effects in the spotlight. Another example is the DCASE "Foley Sound Synthesis" challenge [310], where, alongside DCASE's usual tasks on sound event detection and recognition, the 2023 edition included a generative task aimed at the class-conditional synthesis of sound effects for the first time.

8.4 Concluding remarks

To conclude, with a higher research output that addresses the aforementioned challenges and caters to the preferences of audio professionals, coupled with an already increasing industry demand and interest in this technology, the arrival and adoption of data-driven sound effects synthesis for video games, extended reality, interactive applications, or film, is poised to become a matter of *when*, rather than *if*.

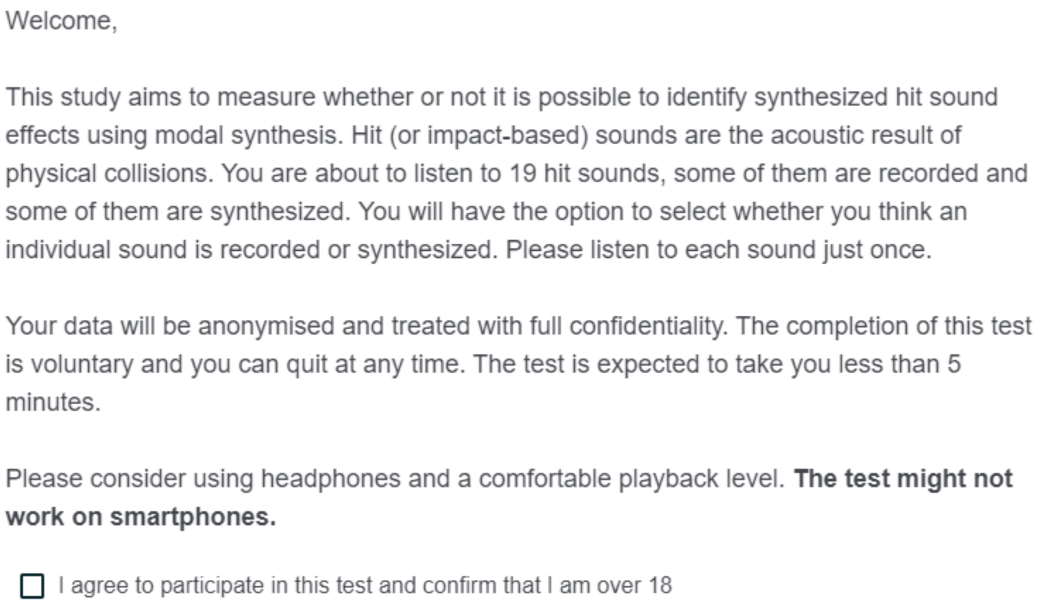
Appendices

APPENDIX A

QUESTIONNAIRES

The figures depicting the consent forms and demographic information sheets showed to the participants are listed below.

- The consent form and demographic information collected in Chapter 4 can be seen in Figure A.1 and Figure A.2 respectively.
- The consent form and demographic information collected in Chapter 5 can be seen in Figure A.3 and Figure A.4 respectively.
- The consent form and demographic information collected in Chapter 6 can be seen in Figure A.5 and Figure A.6 respectively.



Welcome,

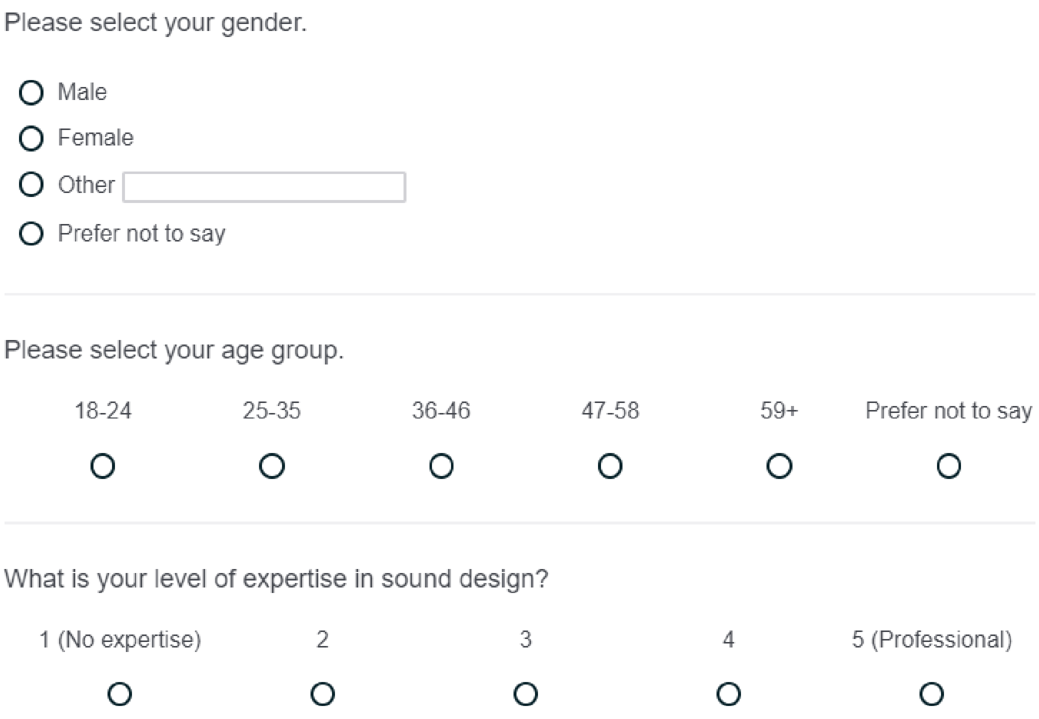
This study aims to measure whether or not it is possible to identify synthesized hit sound effects using modal synthesis. Hit (or impact-based) sounds are the acoustic result of physical collisions. You are about to listen to 19 hit sounds, some of them are recorded and some of them are synthesized. You will have the option to select whether you think an individual sound is recorded or synthesized. Please listen to each sound just once.

Your data will be anonymised and treated with full confidentiality. The completion of this test is voluntary and you can quit at any time. The test is expected to take you less than 5 minutes.

Please consider using headphones and a comfortable playback level. **The test might not work on smartphones.**

□ I agree to participate in this test and confirm that I am over 18

Figure A.1: Participant information form in the listening study that contributes to Chapter 4.



Please select your gender.

○ Male
○ Female
○ Other □
○ Prefer not to say

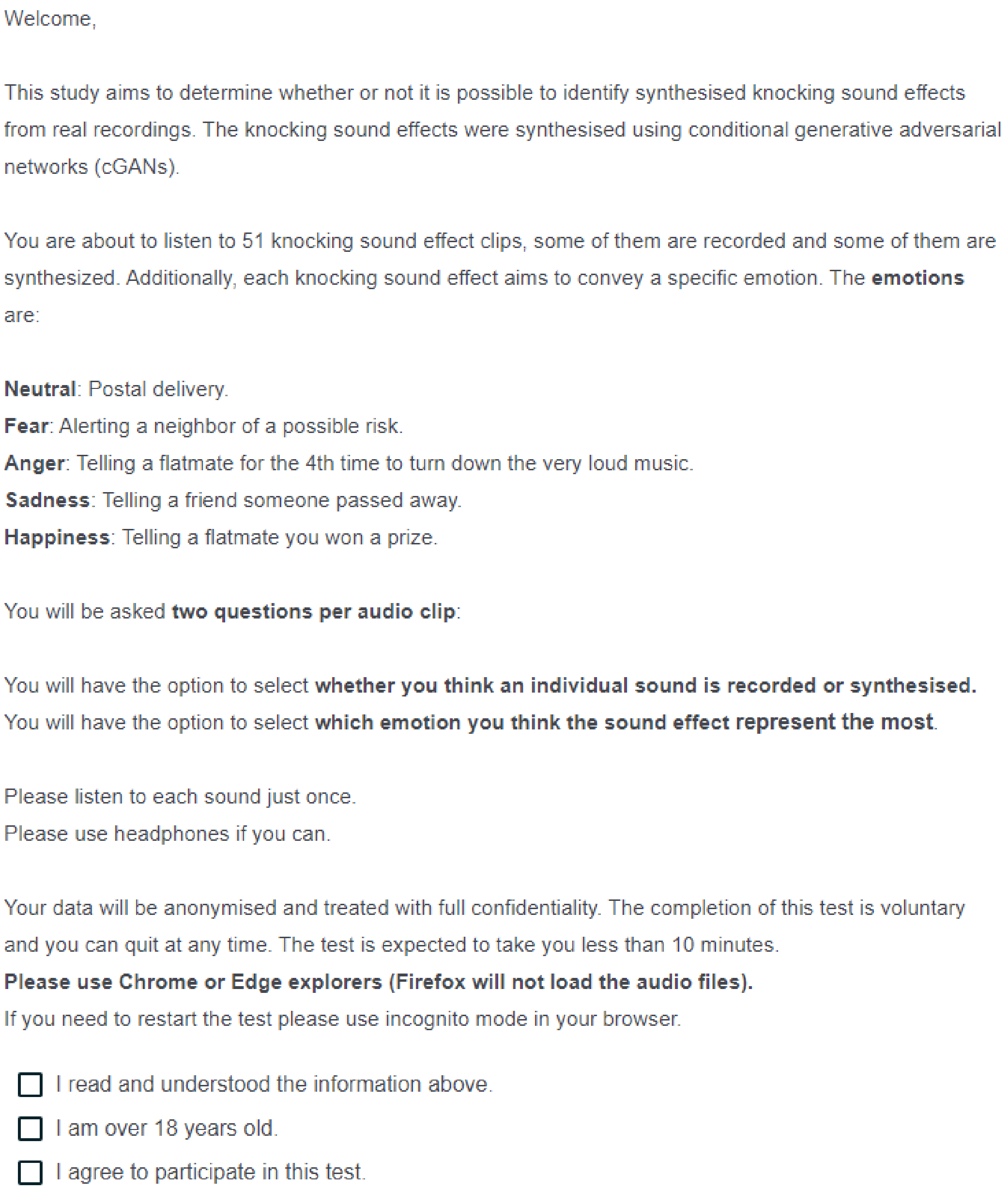
Please select your age group.

18-24	25-35	36-46	47-58	59+	Prefer not to say
○	○	○	○	○	○

What is your level of expertise in sound design?

1 (No expertise)	2	3	4	5 (Professional)
○	○	○	○	○

Figure A.2: Demographic information collected in the listening study that contributes to Chapter 4.



Welcome,

This study aims to determine whether or not it is possible to identify synthesised knocking sound effects from real recordings. The knocking sound effects were synthesised using conditional generative adversarial networks (cGANs).

You are about to listen to 51 knocking sound effect clips, some of them are recorded and some of them are synthesized. Additionally, each knocking sound effect aims to convey a specific emotion. The **emotions** are:

Neutral: Postal delivery.
Fear: Alerting a neighbor of a possible risk.
Anger: Telling a flatmate for the 4th time to turn down the very loud music.
Sadness: Telling a friend someone passed away.
Happiness: Telling a flatmate you won a prize.

You will be asked **two questions per audio clip**:

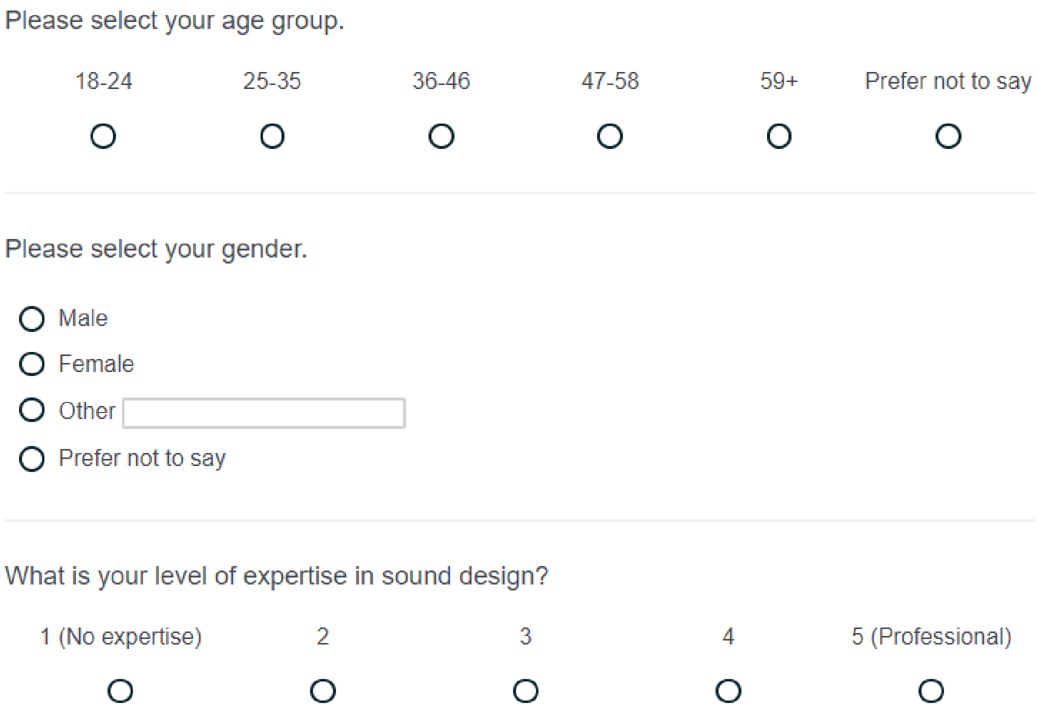
You will have the option to select **whether you think an individual sound is recorded or synthesised.**
You will have the option to select **which emotion you think the sound effect represent the most**.

Please listen to each sound just once.
Please use headphones if you can.

Your data will be anonymised and treated with full confidentiality. The completion of this test is voluntary and you can quit at any time. The test is expected to take you less than 10 minutes.
Please use Chrome or Edge explorers (Firefox will not load the audio files).
If you need to restart the test please use incognito mode in your browser.

□ I read and understood the information above.
□ I am over 18 years old.
□ I agree to participate in this test.

Figure A.3: Participant information form in the listening study that contributes to Chapter 5.



Please select your age group.

18-24	25-35	36-46	47-58	59+	Prefer not to say
○	○	○	○	○	○

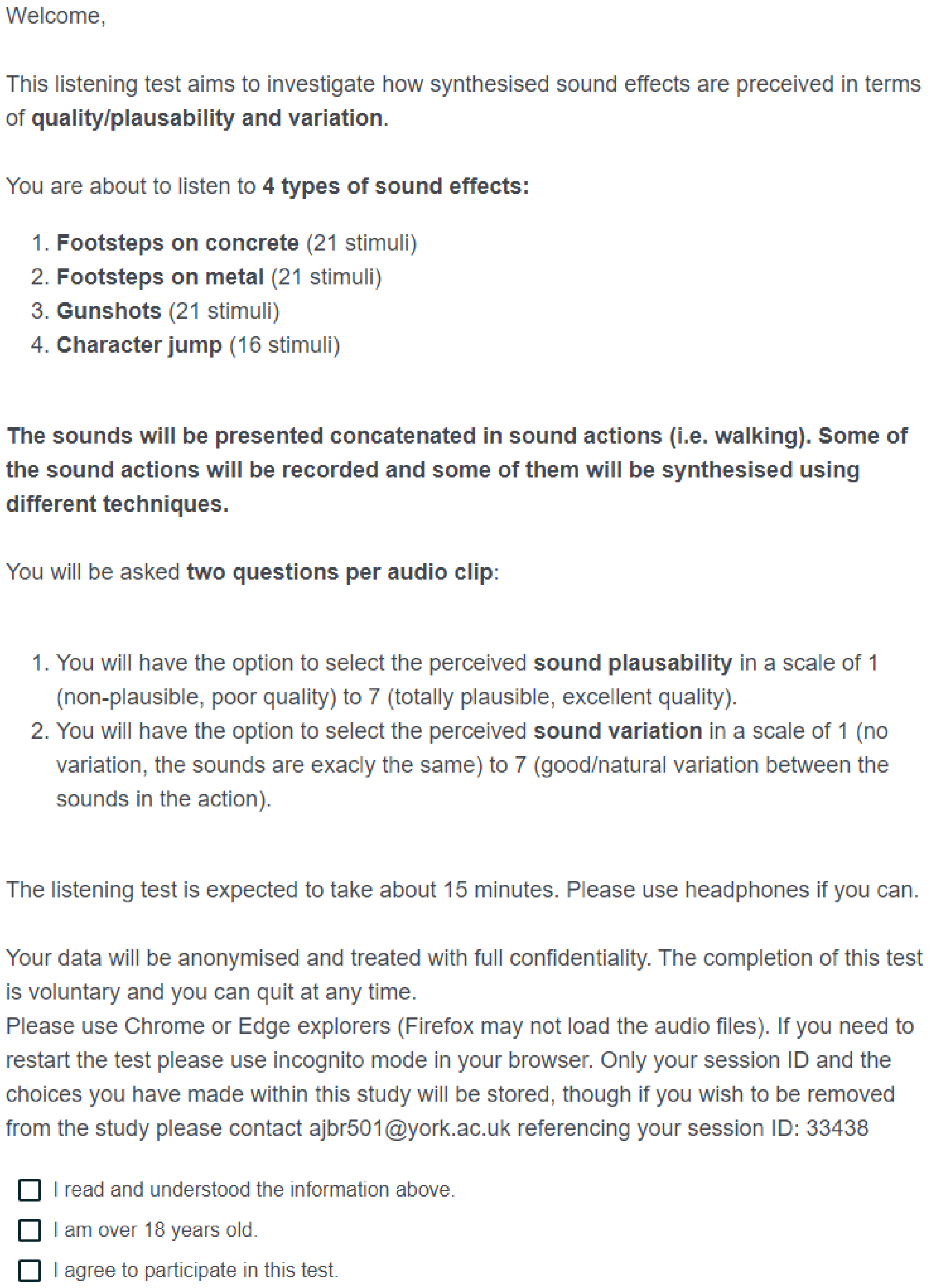
Please select your gender.

○ Male
○ Female
○ Other
○ Prefer not to say

What is your level of expertise in sound design?

1 (No expertise)	2	3	4	5 (Professional)
○	○	○	○	○

Figure A.4: Demographic information collected in the listening study that contributes to Chapter 5.



Welcome,

This listening test aims to investigate how synthesised sound effects are preceived in terms of **quality/plausability and variation**.

You are about to listen to **4 types of sound effects:**

1. **Footsteps on concrete** (21 stimuli)
2. **Footsteps on metal** (21 stimuli)
3. **Gunshots** (21 stimuli)
4. **Character jump** (16 stimuli)

The sounds will be presented concatenated in sound actions (i.e. walking). Some of the sound actions will be recorded and some of them will be synthesised using different techniques.

You will be asked **two questions per audio clip**:

1. You will have the option to select the perceived **sound plausability** in a scale of 1 (non-plausible, poor quality) to 7 (totally plausible, excellent quality).
2. You will have the option to select the perceived **sound variation** in a scale of 1 (no variation, the sounds are exacly the same) to 7 (good/natural variation between the sounds in the action).

The listening test is expected to take about 15 minutes. Please use headphones if you can.

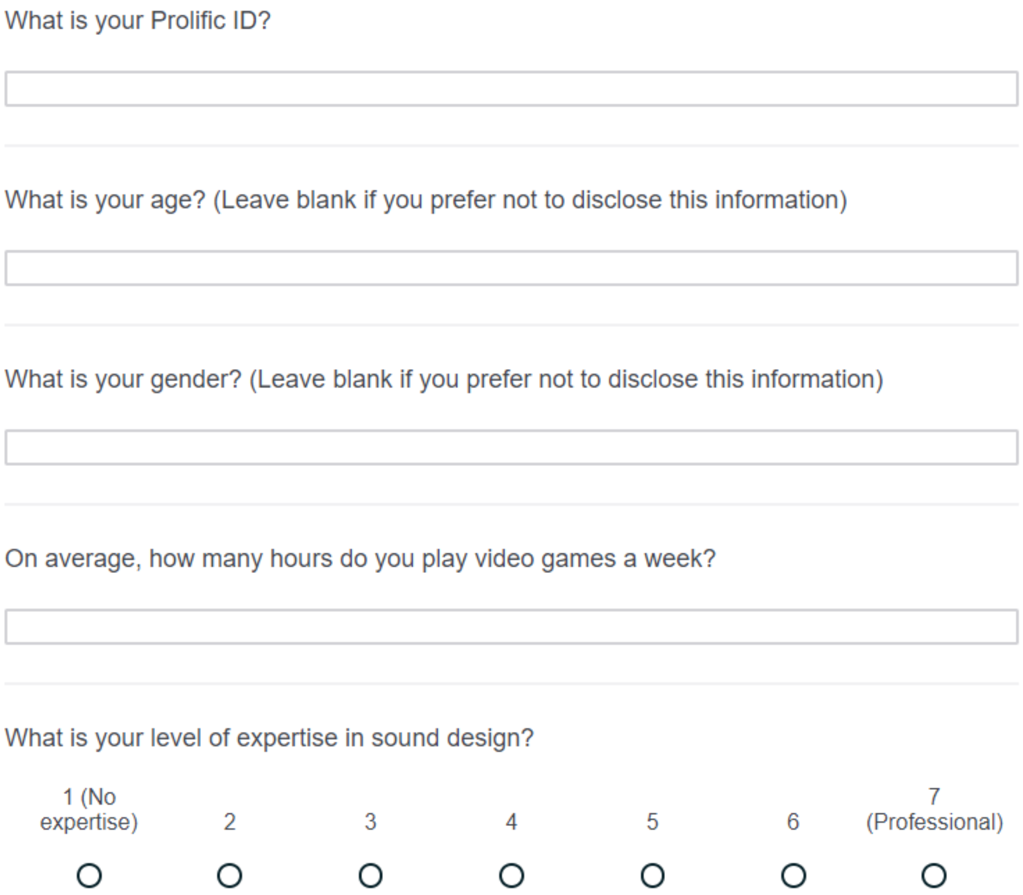
Your data will be anonymised and treated with full confidentiality. The completion of this test is voluntary and you can quit at any time.
Please use Chrome or Edge explorers (Firefox may not load the audio files). If you need to restart the test please use incognito mode in your browser. Only your session ID and the choices you have made within this study will be stored, though if you wish to be removed from the study please contact ajbr501@york.ac.uk referencing your session ID: 33438

□ I read and understood the information above.

□ I am over 18 years old.

□ I agree to participate in this test.

Figure A.5: Participant information form in the listening study that contributes to Chapter 6. The session ID is randomly generated for each participant.



What is your Prolific ID?

What is your age? (Leave blank if you prefer not to disclose this information)

What is your gender? (Leave blank if you prefer not to disclose this information)

On average, how many hours do you play video games a week?

What is your level of expertise in sound design?

1 (No expertise)	2	3	4	5	6	7 (Professional)
○	○	○	○	○	○	○

Figure A.6: Demographic information collected in the listening study that contributes to Chapter 6.

REFERENCES

[1] C. Hausman, F. Messere, and P. Benoit, *Modern Radio and Audio Production: Programming and Performance.* Cengage Learning, 2015.

[2] R. L. Mott, *Sound Effects: Radio, Television and Film.* McFarland, 2014.

[3] C. Hopkins, *Video Game Audio: A History, 1972–2020.* McFarland, 2022.

[4] K. S. Chang, G. B. Kim, and T. Y. Kim, "Video Game Console Audio: Evolution and Future Trends," in *Computer Graphics, Imaging and Visualisation (CGIV 2007)*, pp. 97–102, IEEE, 2007.

[5] G. Zdanowicz and S. Bambrick, *The Game Audio Strategy Guide: A Practical Course.* Focal Press, 2019.

[6] "The Biggest Maps in Video Games." `https://eloutput.com/en/videojuegos/listas/mapas-mas-grandes/`. Accessed: 2023-08-06.

[7] L. P. Berg and J. M. Vance, "Industry Use of Virtual Reality in Product Design and Manufacturing: A Survey," *Virtual reality*, vol. 21, pp. 1–17, 2017.

[8] "Post-Keynote Panel: Procedural Sound Synthesis for AR/VR." `https://www.youtube.com/live/9ngwhfF0FhA?feature=share&t=6368/`. Accessed: 2023-02-09.

[9] A. Farnell, *Designing Sound.* MIT Press, 2010.

[10] M. Yee-King and I. Dall'Avanzi, "Procedural Audio in Video Games," 2018.

[11] D. B. Lloyd, N. Raghuvanshi, and N. K. Govindaraju, "Sound Synthesis for Impact Sounds in Video Games," in *Symposium on Interactive 3D Graphics and Games*, pp. 55–62, 2011.

[12] J. Fagerström, S. J. Schlecht, V. Välimäki, *et al.*, "One-to-Many Conversion for Percussive Samples," in *International Conference on Digital Audio Effects*, pp. 129–135, 2021.

[13] "9 Years Ago, One Action Game Changed Entertainment Forever." `https://www.inverse.com/gaming/gta-v-9th-anniversary`. Accessed: 2023-02-09.

[14] A. MacGregor, "The Sound of Grand Theft Auto V," 2014.

[15] "Behind the Sound of "No Man's Sky": A Q&A With Paul Weir on Procedural Audio." `https://www.asoundeffect.com/no-mans-sky-sound-procedural-audio/`. Accessed: 2023-02-09.

[16] P. Bahadoran, A. Benito, T. Vassallo, and J. D. Reiss, "Fxive: A Web Platform for Procedural Sound Synthesis," in *AES Convention 144*, Audio Engineering Society, 2018.

[17] A. Barahona-Ríos and T. Collins, "SpecSinGAN: Sound Effect Variation Synthesis Using Single–Image GANs," in *19th Sound and Music Computing Conference, Saint-Étienne, France*, 2022.

[18] M. Comunità, H. Phan, and J. D. Reiss, "Neural Synthesis of Footsteps Sound Effects with Generative Adversarial Networks," in *Audio Engineering Society Convention 152*, Audio Engineering Society, 2022.

[19] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A Generative Model for Raw Audio," *arXiv preprint arXiv:1609.03499*, 2016.

[20] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Dignal Processing.* Prentice-Hall, 1999.

[21] S. W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing.* California Technical Publishing, 1999.

[22] P. R. Cook, *Real Sound Synthesis for Interactive Applications.* CRC Press, 2002.

[23] C. E. Shannon, “A Mathematical Theory of Communication,” *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[24] W. Pirkle, *Designing Audio Effect Plug–Ins in C++: With Digital Audio Signal Processing Theory.* Routledge, 2012.

[25] P. Bloomfield, *Fourier Analysis of Time Series: An Introduction.* John Wiley & Sons, 2004.

[26] J. W. Cooley and J. W. Tukey, “An Algorithm for the Machine Calculation of Complex Fourier Series,” *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[27] D. Griffin and J. Lim, “Signal Estimation From Modified Short-Time Fourier Transform,” *IEEE Transactions on acoustics, speech, and signal processing*, vol. 32, no. 2, pp. 236–243, 1984.

[28] T. Tolonen, V. Välimäki, and M. Karjalainen, “Evaluation of Modern Sound Synthesis Methods,” 1998.

[29] S. D. Bilbao, *Numerical Sound Synthesis.* Wiley Online Library, 2009.

[30] J. O. Smith, “Viewpoints on the History of Digital Synthesis,” in *Proceedings of the International Computer Music Conference,*, pp. 1–10, 1991.

[31] X. Serra and J. Smith, “Spectral Modeling Synthesis: A Sound Analysis/Synthesis System Based on a Deterministic Plus Stochastic Decomposition,” *Computer Music Journal*, vol. 14, no. 4, pp. 12–24, 1990.

[32] J. A. Moorer, “Signal Processing Aspects of Computer Music: A Survey,” *Proceedings of the IEEE*, vol. 65, no. 8, pp. 1108–1137, 1977.

[33] E. Miranda, *Computer Sound Design: Synthesis Techniques and Programming.* Routledge, 2012.

[34] S. Liu and D. Manocha, “Sound synthesis, propagation, and rendering: A survey,” *arXiv preprint arXiv:2011.05538*, 2020.

[35] H. G. Alles, “Music Synthesis Using Real Time Digital Techniques,” *Proceedings of the IEEE*, vol. 68, no. 4, pp. 436–449, 1980.

[36] C. Roads, *The Computer Music Tutorial.* MIT press, 1996.

[37] V. Välimäki, J. Pakarinen, C. Erkut, and M. Karjalainen, "Discrete–time modelling of musical instruments," *Reports on progress in physics*, vol. 69, no. 1, p. 1, 2005.

[38] D. Creasey, *Audio Processes: Musical Analysis, Modification, Synthesis, and Control.* Routledge, 2016.

[39] J. M. Chowning, "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation," *Journal of the audio engineering society*, vol. 21, no. 7, pp. 526–534, 1973.

[40] M. Le Brun, "Digital Waveshaping Synthesis," *Journal of the Audio Engineering Society*, vol. 27, no. 4, pp. 250–266, 1979.

[41] D. Schwarz, G. Beller, B. Verbrugghe, and S. Britton, "Real–Time Corpus–Based Concatenative Synthesis With Catart," in *9th International Conference on Digital Audio Effects (DAFx)*, 2006.

[42] D. Schwarz, "Current Research in Concatenative Sound Synthesis," in *International Computer Music Conference (ICMC)*, pp. 1–1, 2005.

[43] D. Schwarz, "Distance Mapping for Corpus–Based Concatenative Synthesis," in *Sound and Music Computing Conference*, 2011.

[44] C. Roads, "Introduction to Granular Synthesis," *Computer Music Journal*, vol. 12, no. 2, pp. 11–13, 1988.

[45] C. Roads, *Microsound.* MIT press, 2004.

[46] J. L. Flanagan and R. M. Golden, "The Phase Vocoder," *Bell System Technical Journal*, vol. 45, no. 9, pp. 1493–1509, 1966.

[47] X. Rodet and P. Depalle, "Spectral Envelopes and Inverse FFT Synthesis," in *Audio Engineering Society Convention 93*, Audio Engineering Society, 1992.

[48] T. S. Verma and T. H.-Y. Meng, "An Analysis/Synthesis Tool for Transient Signals That Allows a Flexible Sines + Transients + Noise Model for Audio," in *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181)*, vol. 6, pp. 3573–3576, IEEE, 1998.

[49] J. Ponce de León, J. R. Beltrán, and F. Beltrán, "Instantaneous Frequency Estimation and Representation of the Audio Signal

Through Complex Wavelet Additive Synthesis," *International Journal of Wavelets, Multiresolution and Information Processing*, vol. 12, no. 03, p. 1450030, 2014.

[50] B. Boashash, "Estimating and interpreting the instantaneous frequency of a signal. I. Fundamentals," *Proceedings of the IEEE*, vol. 80, no. 4, pp. 520–538, 1992.

[51] R. McAulay and T. Quatieri, "Speech Analysis/Synthesis Based on a Sinusoidal Representation," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 34, no. 4, pp. 744–754, 1986.

[52] J. P. de León Vázquez, *Análisis y Síntesis de Señales de Audio a Través de la Transformada Wavelet Continua y Compleja: el Algoritmo CWAS*. PhD thesis, Universidad de Zaragoza, 2012.

[53] J.-M. Adrien, "The Missing Link: Modal Synthesis," in *Representations of Musical Signals*, pp. 269–298, 1991.

[54] J. D. Morrison and J.-M. Adrien, "MOSAIC: A Framework for Modal Synthesis," *Computer Music Journal*, vol. 17, no. 1, pp. 45–56, 1993.

[55] C. Bruyns, "Modal Synthesis for Arbitrarily Shaped Objects," *Computer Music Journal*, pp. 22–37, 2006.

[56] P. R. Cook, "Physically Informed Sonic Modeling (PHISM): Synthesis of Percussive Sounds," *Computer Music Journal*, vol. 21, no. 3, pp. 38–49, 1997.

[57] J. O. Smith, "Physical Modeling Using Digital Waveguides," *Computer music journal*, vol. 16, no. 4, pp. 74–91, 1992.

[58] S. Van Duyne and J. O. Smith, "The 2–D digital waveguide mesh," in *Proceedings of IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pp. 177–180, IEEE, 1993.

[59] L. Savioja, T. Rinne, and T. Takala, "Simulation of Room Acoustics With a 3–D Finite Difference Mesh," in *The 1994 International Computer Music Conference, Aarhus, September 12-17, 1994*, pp. 463–466, Int. Computer Music Ass. and Danish Inst. of Electroa. Music, 1994.

[60] J. Mullen, D. M. Howard, and D. T. Murphy, "Digital Waveguide Mesh Modeling of the Vocal Tract Acoustics," in *2003 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (IEEE Cat. No. 03TH8684)*, pp. 119–122, IEEE, 2003.

[61] A. Fettweis, "Wave digital filters: Theory and practice," *Proceedings of the IEEE*, vol. 74, no. 2, pp. 270–327, 1986.

[62] D. O'Shaughnessy, "Linear Predictive Coding," *IEEE potentials*, vol. 7, no. 1, pp. 29–32, 1988.

[63] J. H. McDermott and E. P. Simoncelli, "Sound Texture Perception via Statistics of the Auditory Periphery: Evidence From Sound Synthesis," *Neuron*, vol. 71, no. 5, pp. 926–940, 2011.

[64] R. E. Turner, *Statistical Models for Natural Sounds*. PhD thesis, UCL (University College London), 2010.

[65] G. Widmer, D. Rocchesso, V. Välimäki, C. Erkut, F. Gouyon, D. Pressnitzer, H. Penttinen, P. Polotti, and G. Volpe, "Sound and Music Computing: Research Trends and Some Key Issues," *Journal of New Music Research*, vol. 36, no. 3, pp. 169–184, 2007.

[66] A. Misra and P. R. Cook, "Toward Synthesized Environments: A Survey of Analysis and Synthesis Methods for Sound Designers and Composers," in *ICMC*, 2009.

[67] D. Schwarz, "State of the Art in Sound Texture Synthesis," in *Digital audio effects (DAFx)*, pp. 221–232, 2011.

[68] S. Serafin, M. Geronazzo, C. Erkut, N. C. Nilsson, and R. Nordahl, "Sonic Interactions in Virtual Reality: State of the Art, Current Challenges, and Future Directions," *IEEE computer graphics and applications*, vol. 38, no. 2, pp. 31–43, 2018.

[69] D. Moffat, R. Selfridge, and J. D. Reiss, "Sound Effect Synthesis," in *Foundations in Sound Design for Interactive Media*, pp. 274–299, Routledge, 2019.

[70] L. Mengual, D. Moffat, and J. D. Reiss, "Modal Synthesis of Weapon Sounds," in *Audio Engineering Society Conference: 61st International Conference: Audio for Games*, Audio Engineering Society, 2016.

[71] H. E. Tez, R. Selfridge, and J. Reiss, "Ocean Wave Sound Synthesis and Perceptual Evaluation," in *Audio Engineering Society Convention 151*, Audio Engineering Society, 2022.

[72] R. Selfridge, J. D. Reiss, E. J. Avital, and X. Tang, "Physically Derived Synthesis Model of an Aeolian Tone," in *Audio Engineering Society Convention 141*, Audio Engineering Society, 2016.

[73] R. Selfridge, D. Moffat, and J. D. Reiss, “Physically Derived Sound Synthesis Model of a Propeller,” in *Proceedings of the 12th International Audio Mostly Conference on Augmented and Participatory Sound and Music Experiences*, pp. 1–8, 2017.

[74] R. Selfridge, D. Moffat, E. J. Avital, and J. D. Reiss, “Creating Real-Time Aeroacoustic Sound Effects Using Physically Informed Models,” *Journal of the Audio Engineering Society*, 2018.

[75] R. Selfridge, *Real-Time Sound Synthesis of Aeroacoustic Sounds Using Physically Derived Models.* PhD thesis, Queen Mary University of London, 2019.

[76] R. Nordahl, S. Serafin, and L. Turchet, “Sound Synthesis and Evaluation of Interactive Footsteps for Virtual Reality Applications,” in *2010 IEEE Virtual Reality Conference (VR)*, pp. 147–153, IEEE, 2010.

[77] L. Turchet, S. Serafin, S. Dimitrov, and R. Nordahl, “Physically Based Sound Synthesis and Control of Footsteps Sounds,” in *Proceedings of the 13th International Conference on Digital Audio Effects (DAFx-10)*, pp. 161–168, 2010.

[78] L. Turchet, “Footstep Sounds Synthesis: Design, Implementation, and Evaluation of Foot–Floor Interactions, Surface Materials, Shoe Types, and Walkers’ Features,” *Applied Acoustics*, vol. 107, pp. 46–68, 2016.

[79] L. Peltola *et al.*, “Analysis, Parametric Synthesis, and Control of Hand Clapping Sounds,” *Master of Science Thesis, Helsinki University of Technology*, 2004.

[80] C. Erkut, “Towards Physics-Based Control and Sound Synthesis of Multi-Agent Systems: Application to Synthetic Hand Clapping,” in *Proc. Nordic Music Technology Conf*, 2006.

[81] C. Erkut and K. Tahiroğlu, “ClaPD: A Testbed for Control of Multiple Sound Sources in Interactive and Participatory Contexts,” in *Proceedings of the PureData Convention*, vol. 7, 2007.

[82] L. Peltola, C. Erkut, P. R. Cook, and V. Valimaki, “Synthesis of Hand Clapping Sounds,” *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, no. 3, pp. 1021–1029, 2007.

[83] A. Jylha, C. Erkut, I. Ekman, and K. Tahiroglu, “iPalmas - An Interactive Flamenco Rhythm Machine,” in *4th Conference on Interaction with Sound: Audio Mostly 2009*, 2009.

[84] A. Misra, P. R. Cook, and G. Wang, "A New Paradigm for Sound Design," in *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, 2006.

[85] A. Misra, P. R. Cook, and G. Wang, "TAPESTREA: Sound Scene Modeling by Example," in *ACM SIGGRAPH 2006 Sketches*, pp. 177–es, 2006.

[86] A. Misra, G. Wang, and P. R. Cook, "TAPESTREA: A New Way to Design Sound," in *Proceedings of the 17th ACM international conference on Multimedia*, pp. 1033–1036, 2009.

[87] C. Verron, M. Aramaki, R. Kronland-Martinet, and G. Pallone, "Spatialized Additive Synthesis of Environmental Sounds," in *Audio Engineering Society Convention 125*, Audio Engineering Society, 2008.

[88] C. Verron, M. Aramaki, R. Kronland-Martinet, and G. Pallone, "A 3-D Immersive Synthesizer for Environmental Sounds," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 18, no. 6, pp. 1550–1561, 2009.

[89] C. Verron, M. Aramaki, R. Kronland-Martinet, and G. Pallone, "Spatialized Synthesis of Noisy Environmental Sounds," *Auditory Display. Springer-Verlag*, pp. 392–407, 2010.

[90] J. F. O'Brien, P. R. Cook, and G. Essl, "Synthesizing Sounds From Physically Based Motion," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 529–536, 2001.

[91] K. Van Den Doel, P. G. Kry, and D. K. Pai, "FoleyAutomatic: Physically-Based Sound Effects for Interactive Simulation and Animation," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 537–544, 2001.

[92] J.-H. Wang, A. Qu, T. R. Langlois, and D. L. James, "Toward Wave-Based Sound Synthesis for Computer Animation," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 109–1, 2018.

[93] S. S. An, D. L. James, and S. Marschner, "Motion-Driven Concatenative Synthesis of Cloth Sounds," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, pp. 1–10, 2012.

[94] C. Schreck, D. Rohmer, D. L. James, S. Hahmann, and M.-P. Cani, "Real-Time Sound Synthesis for Paper Material Based on Geometric

Analysis," in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'16)*, pp. 211–220, Eurographics Association, 2016.

[95] D. Schwarz and N. Schnell, "Descriptor-Based Sound Texture Sampling," in *Sound and music computing (SMC)*, pp. 510–515, 2010.

[96] M. Athineos and D. P. Ellis, "Sound Texture Modelling With Linear Prediction in Both Time and Frequency Domains," in *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03).*, vol. 5, pp. V–648, IEEE, 2003.

[97] M. Fröjd and A. Horner, "Sound Texture Synthesis Using an Overlap–Add/Granular Synthesis Approach," *Journal of the Audio Engineering Society*, vol. 57, no. 1/2, pp. 29–37, 2009.

[98] D. Marelli, M. Aramaki, R. Kronland-Martinet, and C. Verron, "Time-Frequency Synthesis of Noisy Sounds With Narrow Spectral Components," *IEEE transactions on audio, speech, and language processing*, vol. 18, no. 8, pp. 1929–1940, 2010.

[99] D. Marelli, M. Aramaki, R. Kronland-Martinet, and C. Verron, "An Efficient Time-Frequency Method for Synthesizing Noisy Sounds With Short Transients and Narrow Spectral Components," *IEEE transactions on audio, speech, and language processing*, vol. 20, no. 4, pp. 1400–1408, 2011.

[100] N. E. Miner and T. P. Caudell, "Using Wavelets to Synthesize Stochastic-Based Sounds for Immersive Virtual Environments," Georgia Institute of Technology, 1997.

[101] D. O'Regan and A. Kokaram, "Multi-Resolution Sound Texture Synthesis Using the Dual-Tree Complex Wavelet Transform," in *2007 15th European Signal Processing Conference*, pp. 350–354, IEEE, 2007.

[102] A. Kokaram and D. O'Regan, "Wavelet Based High Resolution Sound Texture Synthesis," in *Audio Engineering Society Conference: 31st International Conference: New Directions in High Resolution Audio*, Audio Engineering Society, 2007.

[103] N. Böttcher and S. Serafin, "Design and Evaluation of Physically Inspired Models of Sound Effects in Computer Games," in *Audio Engineering Society Conference: 35th International Conference: Audio for Games*, Audio Engineering Society, 2009.

[104] D. Moffat and J. D. Reiss, "Perceptual Evaluation of Synthesized Sound Effects," *ACM Transactions on Applied Perception (TAP)*, vol. 15, no. 2, pp. 1–19, 2018.

[105] C. Heinrichs and A. McPherson, "Mapping and Interaction Strategies for Performing Environmental Sound," in *2014 IEEE VR Workshop: Sonic Interaction in Virtual Environments (SIVE)*, pp. 25–30, IEEE, 2014.

[106] M. Hoffman and P. R. Cook, "Feature-Based Synthesis: Mapping Acoustic and Perceptual Features Onto Synthesis Parameters," in *ICMC*, Citeseer, 2006.

[107] W. J. Wilkinson, J. D. Reiss, D. Stowell, *et al.*, "Latent Force Models for Sound: Learning Modal Synthesis Parameters and Excitation Functions From Audio Recordings," 2017.

[108] W. J. Wilkinson, J. D. Reiss, and D. Stowell, "A Generative Model for Natural Sounds Based on Latent Force Modelling," in *Latent Variable Analysis and Signal Separation: 14th International Conference, LVA/ICA 2018, Guildford, UK, July 2–5, 2018, Proceedings 14*, pp. 259–269, Springer, 2018.

[109] P. R. Cook, "Synthesis ToolKit in C++, Version 1.0," in *SIGGRAPH 1996, Course #17 &18, Creating and Manipulating Sound to Enhance Computer Graphics*, 1996.

[110] P. R. Cook and G. P. Scavone, "The Synthesis Toolkit (STK)," in *ICMC*, 1999.

[111] N. Porcaro, D. Jaffe, P. Scandalis, J. Smith, T. Stilson, and S. Van Duyne, "SynthBuilder: A Graphical Rapid-Prototyping Tool for the Development of Music Synthesis and Effects Patches on Multiple Platforms," *Computer Music Journal*, vol. 22, no. 2, pp. 35–43, 1998.

[112] M. Klingbeil, "Software for Spectral Analysis, Editing, and Synthesis," in *ICMC*, 2005.

[113] R. E. Causse, J. Bensoam, and N. Ellis, "Modalys, A Physical Modeling Synthesizer: More Than Twenty Years of Researches, Developments, and Musical Uses," *The Journal of the Acoustical Society of America*, vol. 130, no. 4, pp. 2365–2365, 2011.

[114] R. Michon and J. O. Smith, "Faust-stk: A set of linear and nonlinear physical models for the faust programming language," in *Proceedings of the 14th International Conference on Digital Audio Effects (DAFx-11), Paris, France*, pp. 19–23, 2011.

[115] R. Michon, J. Smith, C. Chafe, G. Wang, and M. Wright, "The Faust Physical Modeling Library: A Modular Playground for the Digital Luthier," in *International Faust Conference*, 2018.

[116] R. Michon, S. Martin, and J. Smith, "MESH2FAUST: a Modal Physical Model Generator for the Faust Programming Language-Application to Bell Modeling," in *Proceedings of the 2017 International Computer Music Conference, ICMC.*, 2017.

[117] G. Wang, P. R. Cook, *et al.*, "ChucK: A Concurrent, On-The-Fly, Audio Programming Language," in *ICMC*, 2003.

[118] J. Atherton and G. Wang, "Chunity: Integrated Audiovisual Programming in Unity," in *NIME*, pp. 102–107, 2018.

[119] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.

[120] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT press, 2016.

[121] W. S. McCulloch and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.

[122] A. Géron, *Hands-on Machine Learning With Scikit-Learn, Keras, and Tensorflow.* " O'Reilly Media, Inc.", 2022.

[123] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[124] I. Goodfellow, "NIPS 2016 Tutorial: Generative Adversarial Networks," *arXiv preprint arXiv:1701.00160*, 2016.

[125] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, and S. S. Iyengar, "A Survey on Deep Learning: Algorithms, Techniques, and Applications," *ACM Computing Surveys (CSUR)*, vol. 51, no. 5, pp. 1–36, 2018.

[126] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, M. Hasan, B. C. Van Essen, A. A. Awwal, and V. K. Asari, "A State-Of-The-Art Survey on Deep Learning Theory and Architectures," *electronics*, vol. 8, no. 3, p. 292, 2019.

[127] S. Dargan, M. Kumar, M. R. Ayyagari, and G. Kumar, "A Survey of Deep Learning and Its Applications: A New Paradigm to Machine Learning," *Archives of Computational Methods in Engineering*, vol. 27, pp. 1071–1092, 2020.

[128] S. Bond-Taylor, A. Leach, Y. Long, and C. G. Willcocks, "Deep Generative Modelling: A Comparative Review of VAEs, GANs, Normalizing Flows, Energy-Based and Autoregressive Models," *IEEE transactions on pattern analysis and machine intelligence*, 2021.

[129] I. H. Sarker, "Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions," *SN Computer Science*, vol. 2, no. 6, p. 420, 2021.

[130] S. Dong, P. Wang, and K. Abbas, "A Survey on Deep Learning and Its Applications," *Computer Science Review*, vol. 40, p. 100379, 2021.

[131] A. Natsiou and S. O'Leary, "Audio Representations for Deep Learning in Sound Synthesis: A Review," in *2021 IEEE/ACS 18th International Conference on Computer Systems and Applications (AICCSA)*, pp. 1–8, IEEE, 2021.

[132] Y. Cao, S. Li, Y. Liu, Z. Yan, Y. Dai, P. S. Yu, and L. Sun, "A Comprehensive Survey of AI-Generated Content (AIGC): A History of Generative AI from GAN to ChatGPT," *arXiv preprint arXiv:2303.04226*, 2023.

[133] D. Foster, *Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play.* O'Reilly Media, 2019.

[134] F. Chollet, *Deep Learning With Python.* Simon and Schuster, 2021.

[135] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark," *Neurocomputing*, vol. 503, pp. 92–108, 2022.

[136] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

[137] X. Glorot, A. Bordes, and Y. Bengio, “Deep Sparse Rectifier Neural Networks,” in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, JMLR Workshop and Conference Proceedings, 2011.

[138] K. He, X. Zhang, S. Ren, and J. Sun, “Delving Deep Into Rectifiers: Surpassing Human-Level Performance on Imagenet Classification,” in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[139] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” in *International conference on machine learning*, pp. 448–456, pmlr, 2015.

[140] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer Normalization,” *arXiv preprint arXiv:1607.06450*, 2016.

[141] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, “How Does Batch Normalization Help Optimization?,” *Advances in neural information processing systems*, vol. 31, 2018.

[142] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving Neural Networks by Preventing Co-adaptation of Feature Detectors,” *arXiv preprint arXiv:1207.0580*, 2012.

[143] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, “Handwritten Digit Recognition With a Back-Propagation Network,” *Advances in neural information processing systems*, vol. 2, 1989.

[144] K. Tahiroğlu, M. Kastemaa, and O. Koli, “GANSpaceSynth: A Hybrid Generative Adversarial Network Architecture for Organising the Latent Space using a Dimensionality Reduction for Real-Time Audio Synthesis,” in *Proceedings of the 2nd Joint Conference on AI Music Creativity*, (Online), p. 10, AIMC, July 2021.

[145] J. Engel, C. Gu, A. Roberts, *et al.*, “DDSP: Differentiable Digital Signal Processing,” in *International Conference on Learning Representations*, 2020.

[146] T. Bazin, G. Hadjeres, P. Esling, and M. Malt, “Spectrogram Inpainting for Interactive Generation of Instrument Sounds,” in *Proceedings of*

the 2020 Joint Conference on AI Music Creativity, (Stockholm, Sweden), KTH Royal Institute of Technology, July 2020.

[147] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang, "A Tutorial on Energy-Based Learning," *Predicting structured data*, vol. 1, no. 0, 2006.

[148] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[149] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," in *Proc. 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[150] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic Backpropagation and Approximate Inference in Deep Generative Models," in *International conference on machine learning*, pp. 1278–1286, PMLR, 2014.

[151] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "β-VAE: Learning Basic Visual Concepts With a Constrained Variational Framework," in *International conference on learning representations*, 2017.

[152] A. Van Den Oord, O. Vinyals, *et al.*, "Neural Discrete Representation Learning," *Advances in neural information processing systems*, vol. 30, 2017.

[153] A. Razavi, A. Van den Oord, and O. Vinyals, "Generating Diverse High-Fidelity Images With VQ-VAE-2," *Advances in neural information processing systems*, vol. 32, 2019.

[154] A. Vahdat and J. Kautz, "NVAE: A Deep Hierarchical Variational Autoencoder," *Advances in neural information processing systems*, vol. 33, pp. 19667–19679, 2020.

[155] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[156] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein Generative Adversarial Networks," in *International conference on machine learning*, pp. 214–223, PMLR, 2017.

[157] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved Training of Wasserstein GANs," *Advances in neural information processing systems*, vol. 30, 2017.

[158] A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks," in *4th International Conference on Learning Representations, ICLR, San Juan, Puerto Rico, May 2–4*, 2016.

[159] A. Brock, J. Donahue, and K. Simonyan, "Large Scale GAN Training for High Fidelity Natural Image Synthesis," in *International Conference on Learning Representations*, 2018.

[160] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive Growing of GANs for Improved Quality, Stability, and Variation," in *International Conference on Learning Representations*, 2018.

[161] T. Karras, S. Laine, and T. Aila, "A Style-Based Generator Architecture for Generative Adversarial Networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4401–4410, 2019.

[162] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and Improving the Image Quality of Stylegan," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 8110–8119, 2020.

[163] T. Karras, M. Aittala, S. Laine, E. Härkönen, J. Hellsten, J. Lehtinen, and T. Aila, "Alias–Free Generative Adversarial Networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 852–863, 2021.

[164] Y. Bengio, R. Ducharme, and P. Vincent, "A Neural Probabilistic Language Model," *Advances in neural information processing systems*, vol. 13, 2000.

[165] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[166] D. Bahdanau, K. H. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," *3rd International Conference on Learning Representations, ICLR*, 2015.

[167] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," 2014.

[168] A. Van Den Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel Recurrent Neural Networks," in *International conference on machine learning*, pp. 1747–1756, PMLR, 2016.

[169] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, "PixelCNN++: Improving the PixelCNN With Discretized Logistic Mixture Likelihood and Other Modifications," 2016.

[170] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention Is All You Need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[171] OpenAI, "GPT-4 Technical Report," 2023.

[172] X. Chen, N. Mishra, M. Rohaninejad, and P. Abbeel, "PixelSNAIL: An Improved Autoregressive Generative Model," in *International Conference on Machine Learning*, pp. 864–872, PMLR, 2018.

[173] I. Kobyzev, S. J. Prince, and M. A. Brubaker, "Normalizing Flows: An Introduction and Review of Current Methods," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 11, pp. 3964–3979, 2020.

[174] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, "Normalizing Flows for Probabilistic Modeling and Inference," *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 2617–2680, 2021.

[175] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep Unsupervised Learning Using Nonequilibrium Thermodynamics," in *International Conference on Machine Learning*, pp. 2256–2265, PMLR, 2015.

[176] J. Ho, A. Jain, and P. Abbeel, "Denoising Diffusion Probabilistic Models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6840–6851, 2020.

[177] Z. Kong, W. Ping, J. Huang, K. Zhao, and B. Catanzaro, "Diffwave: A Versatile Diffusion Model for Audio Synthesis," 2020.

[178] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, *et al.*, "Learning Transferable Visual Models From Natural Language Supervision," in *International conference on machine learning*, pp. 8748–8763, PMLR, 2021.

[179] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical Text-Conditional Image Generation With Clip Latents," *arXiv preprint arXiv:2204.06125*, 2022.

[180] J. Engel, C. Resnick, A. Roberts, S. Dieleman, M. Norouzi, D. Eck, and K. Simonyan, "Neural Audio Synthesis of Musical Notes With Wavenet Autoencoders," in *International Conference on Machine Learning*, pp. 1068–1077, PMLR, 2017.

[181] S. Mehri, K. Kumar, I. Gulrajani, R. Kumar, S. Jain, J. Sotelo, A. Courville, and Y. Bengio, "SampleRNN: An Unconditional End-To-End Neural Audio Generation Model," in *International Conference on Learning Representations*, 2017.

[182] J. Nistal, S. Lattner, and G. Richard, "Comparing Representations for Audio Synthesis Using Generative Adversarial Networks," in *2020 28th European Signal Processing Conference (EUSIPCO)*, pp. 161–165, IEEE, 2021.

[183] J. C. Brown, "Calculation of a Constant Q Spectral Transform," *The Journal of the Acoustical Society of America*, vol. 89, no. 1, pp. 425–434, 1991.

[184] X. Wang, S. Takaki, and J. Yamagishi, "Neural Source-Filter Waveform Models for Statistical Parametric Speech Synthesis," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 28, pp. 402–415, 2019.

[185] R. Yamamoto, E. Song, and J.-M. Kim, "Parallel WaveGAN: A Fast Waveform Generation Model Based on Generative Adversarial Networks With Multi-Resolution Spectrogram," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6199–6203, IEEE, 2020.

[186] J. Andén, V. Lostanlen, and S. Mallat, "Joint Time-Frequency Scattering," *IEEE Transactions on Signal Processing*, vol. 67, no. 14, pp. 3704–3718, 2019.

[187] J. Muradeli, C. Vahidi, C. Wang, H. Han, V. Lostanlen, M. Lagrange, and G. Fazekas, "Differentiable Time-Frequency Scattering On GPU," in *Digital Audio Effects Conference (DAFx)*, 2022.

[188] C. Vahidi, H. Han, C. Wang, M. Lagrange, G. Fazekas, and V. Lostanlen, "Mesostructures: Beyond Spectrogram Loss in Differentiable Time-Frequency Analysis," *Journal of the Audio Engineering Society*, vol. 71, no. 9, pp. 577–585, 2023.

[189] P. Manocha, A. Finkelstein, R. Zhang, N. J. Bryan, G. J. Mysore, and Z. Jin, "A Differentiable Perceptual Audio Metric Learned From Just Noticeable Differences," in *Proceedings of the Annual Conference of the International Speech Communication Association, Interspeech*, vol. 2020, pp. 2852–2856, 2020.

[190] P. Manocha, Z. Jin, R. Zhang, and A. Finkelstein, "CDPAM: Contrastive Learning for Perceptual Audio Similarity," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 196–200, IEEE, 2021.

[191] C. J. Steinmetz and J. D. Reiss, "Auraloss: Audio Focused Loss Functions in PyTorch," in *Digital music research network one-day workshop (DMRN+ 15)*, 2020.

[192] K. Kilgour, M. Zuluaga, D. Roblek, and M. Sharifi, "Fréchet Audio Distance: A Metric for Evaluating Music Enhancement Algorithms," in *Proc. Interspeech 2019*, pp. 2350–2354, 2019.

[193] S. Hershey, S. Chaudhuri, D. P. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, *et al.*, "CNN Architectures for Large-Scale Audio Classification," in *2017 ieee international conference on acoustics, speech and signal processing (icassp)*, pp. 131–135, IEEE, 2017.

[194] D. Dowson and B. Landau, "The Fréchet Distance Between Multivariate Normal Distributions," *Journal of multivariate analysis*, vol. 12, no. 3, pp. 450–455, 1982.

[195] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved Techniques for Training GANs," *Advances in neural information processing systems*, vol. 29, 2016.

[196] E. Richardson and Y. Weiss, "On GANs and GMMs," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[197] M. Bińkowski, D. J. Sutherland, M. Arbel, and A. Gretton, "Demystifying MMD GANs," in *International Conference on Learning Representations*, 2018.

[198] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, *et al.*, "Conditional Image Generation With PixelCNN Decoders," *Advances in neural information processing systems*, vol. 29, 2016.

[199] A. Oord, Y. Li, I. Babuschkin, K. Simonyan, O. Vinyals, K. Kavukcuoglu, G. Driessche, E. Lockhart, L. Cobo, F. Stimberg, *et al.*, "Parallel WaveNet: Fast High-Fidelity Speech Synthesis," in *International conference on machine learning*, pp. 3918–3926, PMLR, 2018.

[200] C. Donahue, J. McAuley, and M. Puckette, "Adversarial Audio Synthesis," in *International Conference on Learning Representations*, 2018.

[201] J. Engel, K. K. Agrawal, S. Chen, I. Gulrajani, C. Donahue, and A. Roberts, "GANSynth: Adversarial Neural Audio Synthesis," 2018.

[202] A. Odena, C. Olah, and J. Shlens, "Conditional Image Synthesis With Auxiliary Classifier GANs," in *International conference on machine learning*, pp. 2642–2651, PMLR, 2017.

[203] S. Kim, S.-g. Lee, J. Song, J. Kim, and S. Yoon, "FloWaveNet: A Generative Flow for Raw Audio," pp. 3370–3378, 2019.

[204] W. Ping, K. Peng, K. Zhao, and Z. Song, "WaveFlow: A Compact Flow-based Model for Raw Audio," in *International Conference on Machine Learning*, pp. 7706–7716, PMLR, 2020.

[205] N. Chen, Y. Zhang, H. Zen, R. J. Weiss, M. Norouzi, and W. Chan, "WaveGrad: Estimating Gradients for Waveform Generation," 2020.

[206] A. Caillon and P. Esling, "RAVE: A Variational Autoencoder for Fast and High-Quality Neural Audio Synthesis," *arXiv preprint arXiv:2111.05011*, 2021.

[207] G. Vigliensoni and R. Fiebrink, "Steering Latent Audio Models Through Interactive Machine Learning," in *ICCC'23: 14th International Conference on Computational Creativity, Waterloo, Canada*, 2023.

[208] J. Nistal, S. Lattner, and G. Richard, "DrumGAN: Synthesis of drum sounds with timbral feature conditioning using generative adversarial networks," in *Proc. 21st International Society for Music Information Retrieval Conference*, pp. 590–597, ISMIR, Nov. 2020.

[209] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, "Jukebox: A Generative Model For Music," *arXiv preprint arXiv:2005.00341*, 2020.

[210] A. Agostinelli, T. I. Denk, Z. Borsos, J. Engel, M. Verzetti, A. Caillon, Q. Huang, A. Jansen, A. Roberts, M. Tagliasacchi, *et al.*, "MusicLM: Generating Music From Text," *arXiv preprint arXiv:2301.11325*, 2023.

[211] J. Copet, F. Kreuk, I. Gat, T. Remez, D. Kant, G. Synnaeve, Y. Adi, and A. Défossez, "Simple and Controllable Music Generation," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[212] J. Salamon, C. Jacoby, and J. P. Bello, "A Dataset and Taxonomy for Urban Sound Research," in *Proceedings of the 22nd ACM international conference on Multimedia*, pp. 1041–1044, 2014.

[213] K. J. Piczak, "ESC: Dataset for Environmental Sound Classification," in *Proceedings of the 23rd ACM international conference on Multimedia*, pp. 1015–1018, 2015.

[214] J. F. Gemmeke, D. P. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter, "Audio Set: An Ontology and Human-Labeled Dataset for Audio Events," in *2017 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 776–780, IEEE, 2017.

[215] E. Fonseca, X. Favory, J. Pons, F. Font, and X. Serra, "FSD50K: An Open Dataset of Human-Labeled Sound Events," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 30, pp. 829–852, 2021.

[216] F. Font, G. Roma, and X. Serra, "Freesound Technical Demo," in *Proceedings of the 21st ACM international conference on Multimedia*, pp. 411–412, 2013.

[217] J. Huh, J. Chalk, E. Kazakos, D. Damen, and A. Zisserman, "Epic-Sounds: A Large-Scale Dataset of Actions that Sound," in *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5, IEEE, 2023.

[218] M. Cartwright and B. Pardo, "Vocalsketch: Vocally Imitating Audio Concepts," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pp. 43–46, 2015.

[219] B. Kim, M. Ghei, B. Pardo, and Z. Duan, "Vocal Imitation Set: A Dataset of Vocally Imitated Sound Events Using the Audioset Ontology," in *DCASE*, pp. 148–152, 2018.

[220] R. Kabealo, S. Wyatt, A. Aravamudan, X. Zhang, D. N. Acaron, M. P. Dao, D. Elliott, A. O. Smith, C. E. Otero, L. D. Otero, *et al.*, "A Multi-Firearm, Multi-Orientation Audio Dataset of Gunshots," *Data in brief*, vol. 48, p. 109091, 2023.

[221] Q. Kong, Y. Xu, T. Iqbal, Y. Cao, W. Wang, and M. D. Plumbley, "Acoustic Scene Generation With Conditional SampleRNN," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 925–929, IEEE, 2019.

[222] C. Zhou, M. Horgan, V. Kumar, C. Vasco, and D. Darcy, "Voice Conversion with Conditional SampleRNN," in *Proc. Interspeech 2018*, pp. 1973–1977, 2018.

[223] C. Y. Lee, A. Toffy, G. J. Jung, and W.-J. Han, "Conditional WaveGAN," *arXiv preprint arXiv:1809.10636*, 2018.

[224] J. Kong, J. Kim, and J. Bae, "HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis," *Advances in Neural Information Processing Systems*, vol. 33, pp. 17022–17033, 2020.

[225] X. Liu, T. Iqbal, J. Zhao, Q. Huang, M. D. Plumbley, and W. Wang, "Conditional Sound Generation Using Neural Discrete Time-Frequency Representation Learning," in *2021 IEEE 31st International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6, IEEE, 2021.

[226] S. Andreu and M. V. Aylagas, "Neural Synthesis of Sound Effects Using Flow-Based Deep Generative Models," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 18, pp. 2–9, 2022.

[227] Y. Okamoto, K. Imoto, S. Takamichi, R. Nagase, T. Fukumori, and Y. Yamashita, "Environmental Sound Conversion From Vocal Imitations and Sound Event Labels," *arXiv preprint arXiv:2305.00302*, 2023.

[228] Y. Okamoto, K. Imoto, S. Takamichi, R. Yamanishi, T. Fukumori, and Y. Yamashita, "Onoma-to-wave: Environmental Sound Synthesis from Onomatopoeic Words," *APSIPA Transactions on Signal and Information Processing*, vol. 11, no. 1, 2022.

[229] H. Ohnaka, S. Takamichi, K. Imoto, Y. Okamoto, K. Fujii, and H. Saruwatari, "Visual Onoma-To-Wave: Environmental Sound Synthesis From Visual Onomatopoeias and Sound-Source Images," in *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5, IEEE, 2023.

[230] G. Greshler, T. Shaham, and T. Michaeli, "Catch-A-Waveform: Learning to Generate Audio from a Single Short Example," *Advances in Neural Information Processing Systems*, vol. 34, pp. 20916–20928, 2021.

[231] S. Schwär, M. Müller, and S. J. Schlecht, "A Variational Y-Autoencoder for Disentangling Gesture and Material of Interaction Sounds," in *Audio Engineering Society Conference: AES 2022 International Audio for Virtual and Augmented Reality Conference*, Audio Engineering Society, 2022.

[232] M. Patacchiola, P. Fox-Roberts, and E. Rosten, "Y-Autoencoders: Disentangling Latent Representations via Sequential Encoding," *Pattern Recognition Letters*, vol. 140, pp. 59–65, 2020.

[233] S. Pascual, G. Bhattacharya, C. Yeh, J. Pons, and J. Serrà, "Full-Band General Audio Synthesis With Score-Based Diffusion," in *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5, IEEE, 2023.

[234] D. Yang, J. Yu, H. Wang, W. Wang, C. Weng, Y. Zou, and D. Yu, "Diffsound: Discrete Diffusion Model for Text-To-Sound Generation," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2023.

[235] F. Kreuk, G. Synnaeve, A. Polyak, U. Singer, A. Défossez, J. Copet, D. Parikh, Y. Taigman, and Y. Adi, "AudioGen: Textually Guided Audio Generation," in *Proc. The Eleventh International Conference on Learning Representations*, 2023.

[236] H. Liu, Z. Chen, Y. Yuan, X. Mei, X. Liu, D. Mandic, W. Wang, and M. D. Plumbley, "AudioLDM: Text-to-Audio Generation with Latent Diffusion Models," in *Proc. 40th International Conference on Machine Learning* (A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, eds.), vol. 202 of *Proceedings of Machine Learning Research*, pp. 21450–21474, PMLR, 23–29 Jul 2023.

[237] R. Huang, J. Huang, D. Yang, Y. Ren, L. liu, M. Li, Z. Ye, J. Liu, X. Yin, and Z. Zhao, "Make-An-Audio: Text-To-Audio Generation

with Prompt-Enhanced Diffusion Models," in *Proceedings of the 40th International Conference on Machine Learning*, 2023.

[238] D. Ghosal, N. Majumder, A. Mehrish, and S. Poria, "Text-to-Audio Generation using Instruction-Tuned LLM and Latent Diffusion Model," *arXiv preprint arXiv:2304.13731*, 2023.

[239] A. Owens, P. Isola, J. McDermott, A. Torralba, E. H. Adelson, and W. T. Freeman, "Visually Indicated Sounds," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2405–2413, 2016.

[240] S. Ghose and J. J. Prevost, "Autofoley: Artificial Synthesis of Synchronized Sound Tracks for Silent Videos With Deep Learning," *IEEE Transactions on Multimedia*, vol. 23, pp. 1895–1907, 2020.

[241] S. Ghose and J. J. Prevost, "FoleyGAN: Visually Guided Generative Adversarial Network-Based Synchronous Sound Generation in Silent Videos," *IEEE Transactions on Multimedia*, 2022.

[242] S. Li, L. Zhang, C. Dong, H. Xue, Z. Wu, L. Sun, K. Li, and H. Meng, "FastFoley: Non-autoregressive Foley Sound Generation Based on Visual Semantics," in *National Conference on Man-Machine Speech Communication*, pp. 252–263, Springer, 2023.

[243] Y. Du, Z. Chen, J. Salamon, B. Russell, and A. Owens, "Conditional Generation of Audio from Video via Foley Analogies," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2426–2436, 2023.

[244] J. Nistal, C. Aouameur, I. Velarde, and S. Lattner, "DrumGAN VST: A Plugin for Drum Sound Analysis/Synthesis with Autoencoding Generative Adversarial Networks," in *International Conference on Machine Learning (ICML), Workshop on Machine Learning for Audio Synthesis*, 2022.

[245] F. Ganis, E. F. Knudsen, S. V. Lyster, R. Otterbein, D. Südholt, and C. Erkut, "Real-Time Timbre Transfer and Sound Synthesis Using DDSP," in *Proc. 18th Sound and Music Computing Conference, SMC 2021*, pp. 175–182, Sound and Music Computing Network, 2021.

[246] J. W. Kim, J. Salamon, P. Li, and J. P. Bello, "CREPE: A convolutional representation for pitch estimation," in *2018 IEEE International*

Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 161–165, IEEE, 2018.

[247] A. T. Cemgil and C. Erkut, "Calibration of Physical Models Using Artificial Neural Networks With Application to Plucked String Instruments," *Institute of Acoustics*, vol. 19, pp. 213–218, 1997.

[248] L. Gabrielli, S. Tomassetti, S. Squartini, and C. Zinato, "Introducing Deep Machine Learning for Parameter Estimation in Physical Modelling," in *Proceedings of the 20th International Conference on Digital Audio Effects*, 2017.

[249] B. Hayes, C. Saitis, and G. Fazekas, "Neural Waveshaping Synthesis," in *Proc. 22nd International Society for Music Information Retrieval Conference*, pp. 254–261, ISMIR, Oct. 2021.

[250] S. Shan, L. Hantrakul, J. Chen, M. Avent, and D. Trevelyan, "Differentiable Wavetable Synthesis," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022.

[251] F. Caspe, A. McPherson, and M. Sandler, "DDX7: Differentiable FM Synthesis of Musical Instrument Sounds," in *Proc. 23rd International Society for Music Information Retrieval Conference*, pp. 608–616, ISMIR, 2022.

[252] Z. Ye, W. Xue, X. Tan, Q. Liu, and Y. Guo, "NAS-FM: Neural Architecture Search for Tunable and Interpretable Sound Synthesis based on Frequency Modulation," pp. 5869–5877, 8 2023. AI and Arts.

[253] D.-Y. Wu, W.-Y. Hsiao, F.-R. Yang, O. D. Friedman, W. Jackson, Y.-W. Liu, Y.-H. Yang, *et al.*, "DDSP-Based Singing Vocoders: A New Subtractive-Based Synthesizer and a Comprehensive Evaluation," in *Ismir 2022 Hybrid Conference*, 2022.

[254] M. Yee-King and L. McCallum, "Studio Report: Sound Synthesis With DDSP and Network Bending Techniques," 2021.

[255] R. Diaz, B. Hayes, C. Saitis, G. Fazekas, and M. Sandler, "Rigid-Body Sound Synthesis with Differentiable Modal Resonators," in *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2023.

[256] R. Diaz, C. Saitis, and M. Sandler, "Interactive Neural Resonators," 2023.

[257] X. Jin, S. Li, T. Qu, D. Manocha, and G. Wang, "Deep-Modal: Real-Time Impact Sound Synthesis for Arbitrary Shapes," in *Proceedings of the 28th ACM International Conference on Multimedia*, pp. 1171–1179, 2020.

[258] X. Jin, S. Li, G. Wang, and D. Manocha, "NeuralSound: Learning-Based Modal Sound Synthesis With Acoustic Transfer," *ACM Transactions on Graphics (TOG)*, vol. 41, no. 4, pp. 1–15, 2022.

[259] A. Lundberg, "Data-Driven Procedural Audio: Procedural Engine Sounds Using Neural Audio Synthesis," Master's thesis, KTH School of Electrical Engineering and Computer Science (EECS), 2020.

[260] D. Serrano, "A Neural Analysis–Synthesis Approach to Learning Procedural Audio Models," Master's thesis, New Jersey Institute of Technology, Department of Computer Science, 2022.

[261] H. Han, V. Lostanlen, and M. Lagrange, "Perceptual-Neural-Physical Sound Matching," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 2023.

[262] P. Esling, N. Masuda, A. Bardet, R. Despres, and A. Chemla-Romeu-Santos, "Flow Synthesizer: Universal Audio Synthesizer Control With Normalizing Flows," *Applied Sciences*, vol. 10, no. 1, p. 302, 2019.

[263] Z. Chen, Y. Jing, S. Yuan, Y. Xu, J. Wu, and H. Zhao, "Sound2Synth: Interpreting Sound via FM Synthesizer Parameters Estimation," in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22* (L. D. Raedt, ed.), pp. 4921–4928, International Joint Conferences on Artificial Intelligence Organization, 7 2022. AI and Arts.

[264] J. Turian, J. Shier, G. Tzanetakis, K. McNally, and M. Henry, "One Billion Audio Sounds From Gpu-Enabled Modular Synthesis," in *2021 24th International Conference on Digital Audio Effects (DAFx)*, pp. 222–229, IEEE, 2021.

[265] C. J. Steinmetz and J. D. Reiss, "Efficient Neural Networks for Real–Time Modeling of Analog Dynamic Range Compression," in *Audio Engineering Society Convention 152*, Audio Engineering Society, 2022.

[266] B.-Y. Chen, W.-H. Hsu, W.-H. Liao, M. A. M. Ramírez, Y. Mitsufuji, and Y.-H. Yang, "Automatic dj transitions with differentiable audio effects and generative adversarial networks," in *ICASSP 2022-2022 IEEE*

International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 466–470, IEEE, 2022.

[267] C. J. Steinmetz, N. J. Bryan, and J. D. Reiss, "Style Transfer of Audio Effects With Differentiable Signal Processing," *Journal of the Audio Engineering Society*, vol. 70, no. 9, pp. 708–721, 2022.

[268] Z. Guo, C. Chen, and E. S. Chng, "DENT-DDSP: Data-Efficient Noisy Speech Generator Using Differentiable Digital Signal Processors for Explicit Distortion Modelling and Noise-Robust Speech Recognition," 2022.

[269] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," in *International Conference on Machine Learning*, pp. 28492–28518, PMLR, 2023.

[270] P. Cook and J. O. Smith, "Physics-Based Sound Synthesis for Games and Interactive Systems," 2023.

[271] L. Gabrielli, S. Squartini, and V. Välimäki, "A Subjective Validation Method for Musical Instrument Emulation," in *Audio Engineering Society Convention 131*, Audio Engineering Society, 2011.

[272] A. Ross and V. L. Willson, *Basic and Advanced Statistical Tests: Writing Results Sections and Creating Tables and Figures.* Springer, 2018.

[273] J. van Doorn, A. Ly, M. Marsman, and E.-J. Wagenmakers, "Bayesian Rank-Based Hypothesis Testing for the Rank Sum Test, the Signed Rank Test, and Spearman's ρ," *Journal of Applied Statistics*, vol. 47, no. 16, pp. 2984–3006, 2020.

[274] H. Jeffreys, *The Theory of Probability.* Oxford University Press, 1961.

[275] M. D. Lee and E.-J. Wagenmakers, *Bayesian Cognitive Modeling: A Practical Course.* Cambridge university press, 2014.

[276] C.-W. Wun and A. Horner, "Perceptual Wavetable Matching for Synthesis of Musical Instrument Tones," *Journal of the Audio Engineering Society*, vol. 49, no. 4, pp. 250–262, 2001.

[277] H. P. S. Selasky, "Evaluation of Perceptual Sound Compression with Regard to Perceived Quality and Compression Methods," Master's thesis, Høgskolen i Agder ; Agder University College, 2006.

[278] M. Chang, Y. R. Kim, and G. J. Kim, "A Perceptual Evaluation of Generative Adversarial Network Real-Time Synthesized Drum Sounds in a Virtual Environment," in *2018 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, pp. 144–148, IEEE, 2018.

[279] M. Mirza and S. Osindero, "Conditional Generative Adversarial Nets," *arXiv preprint arXiv:1411.1784*, 2014.

[280] R. Vitale and R. Bresin, "Emotional Cues in Knocking Sounds," in *10th International Conference on Music Perception and Cognition, Sapporo, Japan, August 25-29, 2008*, p. 276, 2008.

[281] P. Ekman, "Basic emotions," *Handbook of cognition and emotion*, vol. 98, no. 45-60, p. 16, 1999.

[282] A. Barahona-Ríos and S. Pauletto, "Knocking Sound Effects With Emotional Intentions." `https://doi.org/10.5281/zenodo.3668503`, Feb. 2020.

[283] L. Gabrielli, S. Squartini, and V. Välimäki, "A Subjective Validation Method for Musical Instrument Emulation," in *AES 131st Convention*, (New York, USA), 2011.

[284] P. N. Juslin and P. Laukka, "Communication of emotions in vocal expression and music performance: Different channels, same code?," *Psychological bulletin*, vol. 129, no. 5, p. 770, 2003.

[285] M. Houel, A. Arun, A. Berg, A. Iop, A. Barahona-Ríos, and S. Pauletto, "Perception of Emotions in Knocking Sounds: An Evaluation Study," in *17th Sound and Music Computing Conference, Torino, Italy*, 2020.

[286] T. Mäki-Patola and P. Hämäläinen, "Latency Tolerance for Gesture Controlled Continuous Sound Instrument Without Tactile Feedback," in *ICMC*, Citeseer, 2004.

[287] T. Q. Nguyen, "Near-Perfect-Reconstruction Pseudo-QMF Banks," *IEEE Transactions on Signal Processing*, vol. 42, no. 1, pp. 65–76, 1994.

[288] J. M. Antognini, M. Hoffman, and R. J. Weiss, "Audio Texture Synthesis With Random Neural Networks: Improving Diversity and Quality," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3587–3591, IEEE, 2019.

[289] S. Wei, S. Zou, F. Liao, *et al.*, "A Comparison on Data Augmentation Methods Based on Deep Learning for Audio Classification," in *Journal of physics: Conference series*, vol. 1453, p. 012085, IOP Publishing, 2020.

[290] T. R. Shaham, T. Dekel, and T. Michaeli, "SinGAN: Learning a Generative Model From a Single Natural Image," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4570–4580, 2019.

[291] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-To-Image Translation With Conditional Adversarial Networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017.

[292] T. Hinz, M. Fisher, O. Wang, and S. Wermter, "Improved Techniques for Training Single-Image GANs," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1300–1309, 2021.

[293] C. Gupta, P. Kamath, and L. Wyse, "Signal Representations for Synthesizing Audio Textures with Generative Adversarial Networks," pp. 159–166, 2021.

[294] Z. Průša, P. Balazs, and P. L. Søndergaard, "A Noniterative Method for Reconstruction of Phase From STFT Magnitude," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 5, pp. 1154–1164, 2017.

[295] G. Le Vaillant, T. Dutoit, and S. Dekeyser, "Improving Synthesizer Programming From Variational Autoencoders Latent Space," in *2021 24th International Conference on Digital Audio Effects (DAFx)*, pp. 276–283, IEEE, 2021.

[296] V. Kulikov, S. Yadin, M. Kleiner, and T. Michaeli, "SinDDM: A Single Image Denoising Diffusion Model," in *International Conference on Machine Learning*, pp. 17920–17930, PMLR, 2023.

[297] S. Pauletto, A. Barahona-Ríos, V. Madaghiele, and Y. Seznec, "Sonifying Energy Consumption Using SpecSinGAN," in *20th Sound and Music Computing Conference, Stockholm, Sweden*, 2023.

[298] B. Hayes, C. Saitis, and G. Fazekas, “Sinusoidal Frequency Estimation by Gradient Descent,” in *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2023.

[299] J. Turian and M. Henry, “I'm Sorry for Your Loss: Spectrally-Based Audio Distances Are Bad at Pitch,” in *"I Can't Believe It's Not Better!" NeurIPS 2020 workshop*, 2020.

[300] V. Välimäki, J. Rämö, and F. Esqueda, “Creating Endless Sounds,” in *Proc. 21st Int. Conf. Digital Audio Effects (DAFx-18), Aveiro, Portugal*, pp. 32–39, 2018.

[301] A. Barahona-Ríos and S. Pauletto, “Synthesising Knocking Sound Effects Using Conditional WaveGAN,” in *17th Sound and Music Computing Conference, Torino, Italy*, 2020.

[302] A. J. Oxenham, “How We Hear: The Perception and Neural Coding of Sound,” *Annual review of psychology*, vol. 69, pp. 27–50, 2018.

[303] S. Bai, J. Z. Kolter, and V. Koltun, “An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling,” *arXiv preprint arXiv:1803.01271*, 2018.

[304] A. Défossez, J. Copet, G. Synnaeve, and Y. Adi, “High Fidelity Neural Audio Compression,” *Transactions on Machine Learning Research*, 2023.

[305] Y. Wang, J. Salamon, N. J. Bryan, and J. P. Bello, “Few-Shot Sound Event Detection,” in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 81–85, IEEE, 2020.

[306] N. Zeghidour, A. Luebs, A. Omran, J. Skoglund, and M. Tagliasacchi, “Soundstream: An End-To-End Neural Audio Codec,” *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 30, pp. 495–507, 2021.

[307] S. Oh, M. Kang, H. Moon, K. Choi, and B. S. Chon, “A Demand-Driven Perspective on Generative Audio AI,” 2023.

[308] U. Cisco, “Cisco Annual Internet Report (2018-2023) White Paper,” *Cisco: San Jose, CA, USA*, vol. 10, no. 1, pp. 1–35, 2020.

[309] A. Di Domenico, G. Perna, M. Trevisan, L. Vassio, and D. Giordano, "A Network Analysis on Cloud Gaming: Stadia, GeForce Now and PSNow," *Network*, vol. 1, no. 3, pp. 247–260, 2021.

[310] K. Choi, J. Im, L. Heller, B. McFee, K. Imoto, Y. Okamoto, M. Lagrange, and S. Takamichi, "Foley Sound Synthesis at the DCASE 2023 Challenge," 2023.