# Percussive Sound Generation with Virtual Listeners and Modular Synthesizers

by

Amir Salimi

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

# Abstract

This work focuses on the virtual generation of short percussive samples which can be used by electronic music artists in their compositions. Although recent advancements in digital synthesis, heuristic search, and neural networks have been utilized for the generation of a variety of sounds, the lack of access to large audio datasets, the problem of open set recognition, and high computational costs persist as barriers towards the expansion of digital sound libraries using these techniques. We present our approach towards the automatic generation of synthesizer programs which mimic one-shot percussive sounds. This work documents the implementation of an automatic system for generation of virtual percussive synthesizer programs using classical signal processing and machine learning. This system relies on virtual ears to find synthesizer programs which mimic percussive sounds, and to further categorize these programs into a number of common drum types. We demonstrate promising results in both detection and categorization of percussive sounds by representation of digital audio through Fourier transformations and autoencoder embeddings. Manual listening tests of the generated sounds indicate that the system can successfully generate drum synthesizers and categorize drum sounds. To facilitate future research, we share our curated datasets of free percussive sounds. These datasets can also be used for the replication of our work.

# Preface

A summary of this thesis has been published in “Proceedings of the 2020 AI Music Creativity Conference, 2020” under the title “Make your own audience: virtual listeners can filter generated drum programs” [67]. The publication includes adapted portions from all chapters of this thesis. Dr. Abram Hindle has made contributions to the composition of the manuscript, assisted with summarizing the work, and conducted manual hearing tests.

# Acknowledgements

I would like to thank Dr. Abram Hindle for his patience and curiosity. This work could not have been completed without his guidance and support. I would also like to thank the residents of the Software Engineering Research Lab for their insights and company.

I am indebted and grateful to my partner Erin for her unwavering support throughout the years. I would also like to thank my parents for their endless sacrifices.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Our Goals

Digital recordings of novel, one-shot[1] drum sounds are not easy or cheap to find. New drum sounds can be obtained via live recordings, layering existing drum sounds, and meticulous combining of sound engineering techniques. The goal of this thesis is to program virtual synthesizers to generate sounds that are suitable substitutes for conventional drum sounds. In addition, given that the generated sound is percussive, it should be grouped with the drum sounds that it resembles.

## 1.2 Why Does the World Need More Drums?

A common approach to the creation of drum tracks for digital music is to combine short recordings of drums and other percussive elements in order to create a virtual drum-kit. This approach frees artists from the need to obtain and store real life instruments while enabling endless combinatorial possibilities. However, by relying on recordings of "real life" drum sounds, we are limited by what instruments exist in the real world and whether or not we have access to clean, one-shot recordings. We believe that virtual sound generation can alleviate these material limitations.

[1] A single hit on the drum that captures its capabilities. In this project, "one-shot" does not refer to learning using 1 example.

## 1.3 What Do We Mean By Drums?

Percussive sounds such as kick drums and snare drums are often created by striking percussive instruments in various ways [7]. These sounds are commonly used to create rhythm in musical compositions [53]. In this work, we are not concerned with the nuances of drums versus percussive sounds, as a result, we often use the terms “drum” and “percussion” interchangeably.

What we refer to in this work as “conventional” drum sounds are not necessarily recordings of real instruments, but any sound that can effectively take their place in a composition. For example, some of the most popular drum-kits used by artists today are digital recordings of analogue synthesizers such as the “Roland TR-808” drum machines [47, 31].

Any sound can be used as a one-shot recording of a drum so long as it adheres to the characteristics of the drum type it’s imitating. For example, since most physical drums (e.g., kicks, snares, hats) create sounds when struck, imitations of these drums are loudest at their onset, followed by a quick and linear drop in loudness; an exception to this general rule are shakers, which are *shaken* rather than *struck*. Another important distinguishing characteristic of drums is their timbre, or the frequencies they generate. For example, we expect kick drums to be dominated by lower audible frequencies, hats and shakers to occupy higher frequencies, and snares to occupy the mid-range frequencies.

## 1.4 What Solution Do We Propose?

We want to virtually create novel drum sounds. We need a generative source that produces audio based on some instructions. Additionally, we need a method of evaluation to help us determine which sounds resemble drums and are worth keeping. We also would like to know what instructions, or programs, caused our audio source to make the sounds we liked, so we can modify and experiment with these programs. In short, our approach requires virtual *generation* and *evaluation* of digital sounds.

## 1.5 What Is Our Methodology?

This is our plan for a generative system which can imitate sounds: a virtual synthesizer continually receives random programs and creates the corresponding sounds, while a virtual ear evaluates and assigns a score to each sound, this score is then used for the separation of undesired outputs from desired ones. This approach assumes that a fraction of the randomly generated sounds can be substituted for percussion and that the virtual ear will assign higher evaluation scores to this subset. While only random search was used in this work, our implementation allows for the future integration of heuristic search algorithms such that the parameters of the synthesizer can be selected based on the previously observed evaluations. Considering these requirements, we found the proper implementation of 2 major components to be crucial:

- *Virtual Synthesizer*: A flexible, deterministic, and tractable synthesizer that can create audio.
- *Virtual Ear*: A classifier that returns an evaluation of an audio sample; estimating an audio sample's fulfillment of a musician's requirements. The virtual ear's evaluation guides the generation process towards a desired path, making it a crucial component of our pipeline.

We approached this implementation with modularity and parallelizability in mind. This allows each component to be debugged, modified, and improved without requiring modifications in other components while increasing scalability and speed of experiments.

While the main focus of this project is the generation of novel percussive sounds, our methodology indicates promising results in regard to the creation of new presets for any virtual synth without a-priori knowledge of its parameters. We also demonstrate the viability of virtual synthesizers based on Digital Signal Processing (DSP) methods for fast, unsupervised creation of novel audio.

## 1.6 Thesis Statement and Contributions

The focus of this work is the creation of new audio samples for use by music producers. We propose a virtual approach to creation of digital drum-kits, by random and rapid programming of virtual synthesizers and use of virtual listeners for extraction and organization of synthesizer sounds resembling drums. We seek to implement a system of virtual sound generation where the majority of sounds generated can be used in digital music compositions as a replacement for one-shot sounds of at least one type of percussive instrument.

The principal contributions of this dissertation are:

- We introduce a framework for rapid, high quality sound generation with virtual synthesizers using simple digital signal processing techniques.
- For training of virtual listener models, we curated a dataset of free percussive sounds which is made available to future researchers [66].
- We train several machine learning models that can distinguish drum sounds from non-drum sounds and categorize the type of drum sounds. We measure and verify the accuracy of these models using conventional drum sounds.
- We combine our methods of synthesis and sound categorization to create systems capable of automatic generation of drum sounds of various categories and create synthesizer programs which can be further modified by sound engineers.
- We demonstrate the viability of our approach for the creation of digital percussive sounds by conducting manual hearing tests. Based on our blinded hearing tests, most of the outputs of our generative system of sounds are suitable replacements for drums of various categories.

# Chapter 2

# Background and Related Work

This chapter aims to provide a background for the subsequent chapters by providing a quick overview of four important topics:

(i) Digital sound, its features, and concepts that have been fundamental to our work.

(ii) Common digital synthesis techniques.

(iii) The applications of artificial neural networks (ANNs) for feature extraction and sound production. Although similar results can be yielded from either approach, we distinguish ANN based techniques from tradition digital signal processing (DSP).

(iv) Related works and their relative similarities and distinctions.

## 2.1 Digital Audio: Sound from Numbers

Sound is the result of a series of physical events. Most of what we hear is the product of physical disturbances, causing vibrations in our mutually shared, immersive mediums. Sound waves are vibrations traveling through air as part of an expanding, spherical wave front, exponentially losing intensity as they travel away from the source [59].

A sound wave can be viewed as the result of a function which governs amplitude through time, where time and amplitude exist in continuous dimensions. Waves can be approximated via a series of samples, associating time steps to a discrete range of amplitude values. Given a wave generation method, computers can make sound by sending a series of discrete values to a digital to analogue converter (DAC), which in turn can create vibrations within a speaker. *Digital synthesis* of audio is the process of creating these discrete values.

### 2.1.1 Sampling Rates and Quality of Digital Audio

In 1963, Mathews wrote on the potential and utility of computers as digital instruments [46]. He presented a snapshot of digital audio technology of his time and made predictions on what would be possible in the future [46]. Many of the techniques described by Mathews have not only remained popular and relatively unchanged, but also benefited from the increase of computational power throughout the decades [42, 69]. For instance, Mathews described the "discrete sampling continuous pressure waves" as a general method for computers to capture and internally represent audio [46].

Sound can take on the physical characteristics of a waveform [59]. Imagine that the curve shown in Figure 2.1 is representative of a sound wave we would like to digitally capture. A microphone enables a computer equipped with an analogue-to-digital converter chip (ADC) to record the vibrations caused by sonic pressure waves at fixed intervals. This means that the original, analogue waveform is now recorded as a discrete, digital signal. Each recorded sample would represent the amplitude of the wave at a time-step. The more packets of information we get, the better our digital recreation of the original sound.

In this context, *sampling rate* is an important feature of digital sound, referring to the number of samples per second of audio, typically measured in hertz (Hz). Sampling rate is not the only important factor when recording audio as it is important to record with not just speed, but also precision. Assuming perfect sensors, precision is the range of possible values we can assign to each sample. It is determined by bit depth: the number of bits we have to represent the values of each sample. Today, standard quality audio often refers to sampling rates of 44.1 kHz and 48 kHz and bit depth of 16 (that is, $2^{16}$ discrete values), while "high quality" audio indicates an increase in bit rate or bit depth [63]. Although subject to diminishing returns, high quality audio (e.g., 96 kHz/24-bit) may be preferable to most musicians and audio-engineers. In a meta-analysis of digital sound perception, Reiss found a small but statistically significant portion of people are able to discriminate the effect of standard and high quality audio with no prior training, and a dramatically higher detection rate after extensive training [63].

### 2.1.2 Loudness, Amplitudes and Envelopes

Loudness is a subjective description of a sound's intensity or energy level. It varies based on the complexity of sounds, the frequencies present, and hearing ability of the listener [23, 45]. It can only be measured relatively, by establishing a benchmark sound and surveying populations on the relative intensities [45]. Since loudness and intensity of sound correlate with the amplitude of digital waveforms (the values assigned to the samples), an imperfect but convenient alternative method for inferring the loudness of digital sounds is to compare relative amplitudes. A common function for inferring the loudness of digital signals is to apply the Root Mean Square (RMS) function to its samples [75].

Sounds typically vary in intensity as they unfold. This change in intensity is often described by the *envelope* of the sound, particularly in shorter samples. Envelopes can be mathematically described and used to shape signals. A common approach in digital sound synthesis is to output all samples at a consistent amplitude and apply an envelope later down the synthesis chain. Digital and analog synthesizers often have built-in ADSR modules to shape the volume of the output and other parameters. For digital sounds, Mitchell describes the envelope as either of the borders (since samples typically take the range of -1 to 1) that are created by graphing a signal and connecting the local absolute peak values [48]. In electronic music production, envelope

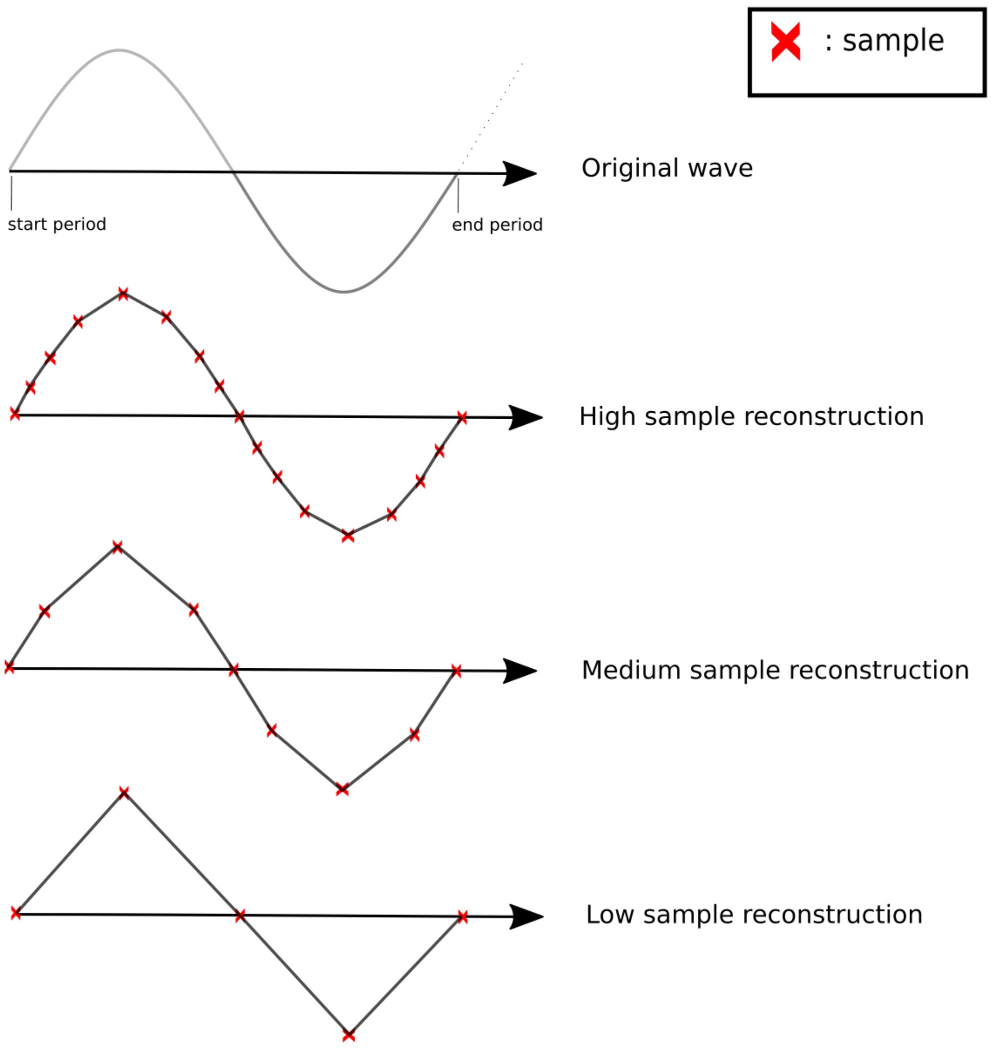

: sample
Original wave
start period
end period
High sample reconstruction
Medium sample reconstruction
Low sample reconstruction

Figure 2.1: Inadequate sampling rates can make reconstruction ambiguous. While reconstruction is possible in this case, consider the case when multiple signals with varying frequencies are overlapped, or the case when samples are further apart than 1 wave length.

is often characterized using 4 features: Attack, Decay, Sustain, and Release (ADSR). Attack describes how quickly the peak loudness is reached. Decay for how quickly the sound drops to sustain level. Sustain is the duration of sustaining intensity (for example, how long a finger is kept on a piano key). Release describes the speed of fading to silence (how fast the sound decays once the piano key is released). We will describe our method of approximating the envelope of sounds in Section 5.3.1.

### 2.1.3 Frequency, Pitch and Spectrograms

*Frequency* is used to describe number of repetitions within a time-frame, or how frequently a cycle is repeated. As discussed before, frequency of an audio signal is often measured in unit of hertz. Most sounds, particularly those from non-virtual sources, are a combination of multiple different pressure waves with different frequencies and amplitudes. *Pitch*, is a perceptual property that is tied to the frequencies present in a sound. How we perceive and describe the pitch of a sound is heavily dependent on the characteristics—such as frequency, amplitude, duration, etc. —of the waveforms it contains. Some sounds, such as piano keys or pure tones have a discernible pitch. Others, such as "pink noise" or the sound of rain, do not. Yet another factor to consider is the hearing ability of the subject, which varies between people based on factors such as age, environment, and musical training [63, 2, 54].

Spectrograms are graphs used to depict the duration and amplitude of frequencies present in a sound. To create spectrograms, sound must be decomposed into a set of simpler functions. A common method for the breakdown of complex, time-variant functions is the Fourier transformation and its many variations [15]. One such method is the discrete Fourier transformation (DFT) and the inverse DFT [15, 27, 52]. DFT and inverse DFT can convert digital sound from its time domain representation (sequence of samples) to its frequency domain representation (sets of frequency ranges and their amplitude) and vice versa. We share some examples along with our methodology of creating spectrograms in Section 5.3.1.

## 2.2 Digital Audio Synthesis

The phenomena of sound at intensities we commonly encounter can be described as the output of a *linear system* of functions [59]. A linear system is a system where the transformation of combined inputs is equal to the sum of the separately transformed inputs [41, 59]. In a linear system $\mathcal{S}$ with valid inputs and outputs $x$ and $y$, if we have:

$$\mathcal{S}(x_1) \xrightarrow{generates} y_1 \tag{2.1}$$

and

$$\mathcal{S}(x_2) \xrightarrow{generates} y_2 \tag{2.2}$$

The output of the system given both inputs is the sum of the individual outputs, or:

$$\mathcal{S}(x_1 + x_2) \xrightarrow{generates} y_1 + y_2 \tag{2.3}$$

This concept has important implications digital audio creation and analysis. Simple tones can be combined to create complex sounds, and complex sounds can be broken down for easier analysis [41]. It also allows experiments with simple sine waves to remain relevant in complex sound domains [59].

Various sound synthesis techniques have been developed by treating sound as a sequence of values. Linear systems are commonly used in creation of musical tones, while non-linear systems are used for introduction of distortion and noise where needed. In their taxonomy of digital synthesis techniques, Smith defines four families of algorithms: algorithms that process and modulate existing sounds (e.g., granular synthesis, wavelets), spectral models that aim to create a particular spectrum of sound (e.g., additive, subtractive), physical models which emulate the physics of real instruments, and abstract models (e.g., wave shaping, Karplus-Strong), often used for adding harmonics or distortion to simple sound signals [69].

Synthesizers are engines of synthesis that make use of one or more of these techniques for sound generation. Selection of the appropriate synthesis method depends not only on the expectations for the sound, but also the features of the synthesizer itself. Whether in goal oriented tasks such as text-to-speech or in creative endeavors such as ambient-noise generation, it is often desirable to work with systems that are quick, adaptable and tractable. For example, one might desire a text-to-speech system where slight changes to input parameters can introduce slight changes to the speech patterns, utterances, voices, etc. This ability to quickly modify and audition sounds becomes a necessity when the synthesizer is being used as a creative instrument in of itself, rather than an emulator for existing instruments and sounds.

Often used methods of digital sound generation are "additive" and "subtractive" synthesis, umbrella terms for some of the most simple and common methods of digital synthesis [49]. In additive synthesis, sounds are built as a sum of signals, where signals are outputs of oscillators (periodic wave generators). In subtractive synthesis, segments of a complex signal are removed until a desired sound is reached. A chain of one or more *digital filters*, which can subtract or reduce frequency ranges, are often used in subtractive synthesis. Digital *low-pass filters* lower the amplitude of signals with frequencies higher than a given *cutoff*, while *high-pass* filters remove lower than threshold frequencies. It is not uncommon for percussive sounds to have noisy, chaotic high frequency content during their short attack period, followed by harmonic low/medium frequencies [37].

### 2.2.1 Virtual Synthesizers

Nearly 5 decades ago, Mathews claimed that any sound can be recreated via a computer by high frequency sampling of pressure waves [46]. He noted that since "a very high sampling rate is required...if this process is to be useful musically, programs for generating samples from the parameters of notes must be written" [46]. The methods of synthesis discussed in this chapter are a major component of such programs, and modern computers are more than capable of simultaneously running many instances of these algorithms. To further assist with their musical utility, the majority of digital synthesis systems work in tandem with notation protocols such as Musical Instrument Digital Interface (MIDI), which can modulate the parameters of these synthesis methods by modulating information pertaining to ADSR and other note characteristics, often in real time [50].

The rise Digital Audio Workstations (DAWs) [39] and Virtual Studio Technology (VST) plug-ins [71] which provide cheaper and virtual alternatives to recording studios and musical equipment have rapidly transformed the sonic and material landscape of music production in the recent years. DAWs such as FL-Studio[1] and Ableton[2] provide virtual alternatives to expensive hardware typically found in recording studios. These DAWs typically come with a large set of VSTs which can accurately imitate nearly all experimental and traditional physical instruments (e.g., moog synthesizers, pianos, choir voices) and audio effects (e.g., reverb, chorus, delay). Furthermore, communities such as KVRAudio[3] provide an ever-growing list of free and commercial VST instrument and effects which can be added to DAWs or run stand-alone.

Coupled with this rise in popularity is a vast array of commercial products and services which cater to the needs of amateur and professional music producers for unique sounds, often by provision of audio samples; one-shot drum samples, long sustained notes (referred to as pads or textures), and loops (percussive or melodic) are common deliverables. Two notable examples of commercial services which provide audio recordings for electronic music artists are *loopmasters*[4] and *splice.com*[5]. While VST plug-ins can emulate analogue synthesizers and effects, however, due to their (often) complex interface, some producers may find VST plugins daunting to work with from scratch. In many cases, VST plug-in vendors or unaffiliated enthusiasts sell additional presets for these plugins, targeted towards producers who do not have the time or interest in creating their own. The flexibility of the VST technology allows

[1]https://www.image-line.com/
[2]https://www.ableton.com/
[3]https://www.kvraudio.com/
[4]https://www.loopmasters.com/
[5]https://www.splice.com/

producers to modify these presets until their desired sound is reached.

## 2.3 Neural Networks And Sound

In Section 2.1, we defined digital synthesis as the “process of generating discrete values which approximate sound waves”. Virtual generation of sounds has motivated a wide variety of synthesis techniques which aim to create signals within a linear (or mostly linear) system. The recent exponential increase in computing power has been coupled with a wide range of research in probabilistic sound generation, mainly via generative neural networks.

Artificial Neural networks (ANNs) map inputs to outputs via a large network of parameters and activation functions. Given the right network shape and parameter weights, they can approximate a large set of functions [17, 13]. ANNs are often deployed when we do not have access to the system of functions which guide a process, but a mapped set of inputs and the corresponding outputs are available. Given this set, the parameters of a neural network can be tuned for approximating the effect of the system on any valid input. The architecture of the neural network (e.g., number of layers, connections, activation functions) is often selected via trial and error [10, 11, 6]. By definition, these approximations will never be more accurate than the system that is being approximated.

Since their emergence in the 1950’s, research on ANNs has gone through several eras of stunted growth [8, 4]. In the last decade, the increase in the affordability of high performance graphic cards has been coupled with a major resurgence of interest for ANNs and the emergence of a number of domain specific variations of the traditional ANN architectures (see Section 2.3.1).

Generative neural networks (GNNs) are utilized for the completion of sequences of values; often by taking an incomplete sequence as input and outputting the most likely value for the next step. The WaveNet architecture introduced in 2016 is considered a seminal breakthrough in the usage ANNs for sound synthesis [56] by surpassing state of the art speech synthesis techniques, which create outputs with the the combination of previously recorded

| work | feature extraction | synthesis | specilization |
|---|---|---|---|
| Oord et al. [56] | CNN | CNN | Speech |
| Yamamoto et a.l [72] | GAN | GAN | Speech |
| Aouameur et al. [5] | Autoencoder | Decoding of Latent Layers | Percussion |
| Ramires et al. [62] | CNN | FeedForward Network | Percussion |
| Yee-King et al. [73] | LSTM | DSP | Synth Pads |

Table 2.1: Quick reference for related works

audio snippets [68]. When trained on a large corpus of audio samples, GNNs such WaveNet can learn the "predictive distribution for each audio sample conditioned on all previous ones" [56]. Once this distribution is learned, it can be used to create sounds 1 sample at a time, a slow process, as Mathews predicted [46].

## 2.3.1 Related Works

ANN or DSP approaches can be taken towards the implementation of a virtual ear and a virtual synthesizer. The recent development of ANN frameworks has led to works which have utilized ANNs for both components [56, 72, 62]. Also common are works which have leveraged a mixture of both approaches, often by utilization of ANNs for the virtual ear and DSP methods for synthesis [5, 73].

Many deep neural network models have been proposed and utilized for the purpose of signal generation in recent years. WaveGans and WaveNet have been subject to significant improvements and experiments since their proposal [19, 72, 57]. Specifically for the generation of percussive sounds, a recent work by Aaouameur et al. [5] utilizes variational AutoEncoders (VAE's) for generation of drum sound spectrograms, which are then converted to sound using a Multi-head CNN model [5]. Another recent work by Ramires et al. [62] also uses neural networks for this purpose, where a feedforward neural network capable of creating sounds is guided by a small number of parameters which represent the producer's desired characteristics for a drum sound.

Automatic programming of virtual synthesizers has also been a topic of interest. Genetic Algorithms have long been utilized for the generation of new sounds with various sound-engines [35, 18, 32, 43]. More recent work by Yee-King et al. [73] used Long Short-Term Memory (LSTM) models and genetic algorithms to find the exact parameters used to create a group of sounds. The sounds approximated were made by the same virtual synthesizer, not an external source; making the eventual replication certain even with random search. In addition, the work by Yee-King et al. [73] is generally more focused on pads and textures rather than drums, and feature matching appears to not be concerned with the envelope of the sounds but rather the frequency content within arbitrary time windows. Yet another recent work by Esling et al. used a large dataset of over 10,000 presets for a commercial VST synthesizer to learn a latent parameter space which can be sampled for creation of new programs for audio synthesizers [21]. This bespoke latent space requires large amounts of synthesizer programs for the initial training, and cannot be used for other virtual synthesizers.

In the work presented here, we take a different approach to automatic programming of synthesizers by aiming for rapid approximation of percussion sounds with no previous knowledge about the sonic capabilities of our virtual synthesizer and directly exploring the synthesizer's parameter space rather than its latent representation through a neural network. Unlike previous related works, no prior examples of audio made by the synthesizer nor examples of manually generated programs are needed for our approach. Any synthesizer can be integrated into our system so long as *a*) its parameters are known, *b*) the system can randomly modify its parameters, and *c*) The system can render and extract the sound output for virtual listening tests. These requirements are not strict as VST synthesizers meet these expectations by design, giving our project a large scope of applicability for future work.

# Chapter 3

# Datasets

## 3.1 Why is Data Needed?

This project requires a *virtual synthesizer* capable of producing sounds with a variety of characteristics (as long as a fraction of these sounds can be suitable replacements for percussion). On the other-hand, the *virtual ear* has a more concrete task: the separation of drum-like sounds from other synthesizer sounds. Rather than manually defining the characteristics which distinguish percussive sounds from all other types, we use supervised machine learning to train the virtual ear by example. To this end, we gathered 3 databases of percussive sounds, and use supervised machine learning methods to create our virtual ear models. Chapter 5.1 will cover how these datasets are transformed and used for training.

## 3.2 Datasets

We curated 3 different datasets of drums, as well as a dataset of randomly generated synthesizer sounds. These drums are meant to represent conventional drum sounds, which can be recordings of physical drums or designed by sound engineers using analogue synthesizers such as the “Roland TR-808”. The breakdown of samples in each dataset is given in Table 3.1. Below is a summary of how each dataset was curated, and the steps to reproduce it, if possible.

### 3.2.1 FreeDB

FreeDB is our curated dataset of free drum-kits extracted from the "SampleSwap" project[1]. The SampleSwap database contains a variety musical and non-musical sounds. We manually selected the sub-directories from SampleSwap which contained drum sounds, and grouped drum sounds which did not belong to "Kick", "Snare", "Clap", "Hat" into the group "Other". FreeDB is copyright free and is available at: `https://zenodo.org/record/3994999`.

### 3.2.2 RadarDB

RadarDB is a set of drum sounds aggregated from royalty free sources such as music radar[2]. We cannot directly share this dataset as it is not copyright free, however, the script for its automated creation can be found under the "getting_data" directory of the project: `https://github.com/imilas/Synths_Stacks_Search`. Be aware that nearly 50GBs of compressed audio files will be downloaded, extracted, and filtered to create RadarDB.

### 3.2.3 MixedDB

MixedDB is a large set of 2 second or shorter drum samples aggregated from personal libraries. We cannot share this dataset as it contains copyrighted material.

### 3.2.4 NoiseDB

NoiseDB is our database of synthetic noise from 1, 3, and 5 stacked virtual synthesizers (synthesizer stacks will be discussed in the upcoming Section 4.3). 2000 sounds of each stack size were selected for a total of 6000 sounds. This dataset is used as a source of "negative examples", i.e, sounds which we generally want to reject, unless they are very similar to drum sounds.

### 3.2.5 FreeRadarDB

A database put together by combination of radarDB, FreeDB and NoiseDB.

[1] `https://sampleswap.org/`

[2] https://www.musicradar.com/

| DB Name | Categories |
|---|---|
| FreeDB | Kicks:533 - Snares:372 - Claps:230 - Hats:105 - Other:281 |
| RadarDB | Kicks:1054 - Snares:842 - Claps:353<br>Toms:349 - Hats:1561 - Rims:131 - Shakers:121 |
| MixedDB | Kicks:648 - Snares:732 - Claps:179 - Hats:105 - Toms:416 - Others:281 |
| NoiseDB | 1 Stack:2000 - 3 Stacks:2000 - 5 Stacks:2000 |
| FreeRadarDB | kick:1334 - snare:1035 - clap:401 - hat:1275 - Synthetic:1000 |

Table 3.1: Overview of our curated datasets.

## 3.3 How The Datasets Will Be Used

The drum and non-drum data here can be used as examples to learn the characteristics of drums and non-drums. In Section 5.1, we discuss how supervised machine learning algorithms are trained to categorize drums from non-drums using these datasets.

# Chapter 4

# Virtual Synthesizer

## 4.1 Why Do We Need a Virtual Synthesizer?

Earlier in Section 1.4, the solution we proposed towards the virtual creation of novel drum sounds required the implementation of a virtual, programmable sound synthesizer. We conceptualized a pipeline where a programmable virtual synthesizer rapidly creates audio based on random parameters, and a virtual ear accepts or rejects the sounds based on its training. This plan is visualized in Figure 4.1. In this chapter, we describe the implementation of a programmable synthesizer of sounds, its parameters, and the reasoning behind our engineering choices. In the subsequent chapters, we discuss the implementation of the virtual ear and the approach taken to combining and testing the two virtual components.

## 4.2 Why DSP over ANN Synthesizers?

As discussed in Section 2.3.1, many of the state of the art works tackling the problem of supervised audio generation utilize neural networks for synthesis of audio. Here, rather than using neural networks for sound synthesis, we generate programs for a virtual synthesizer which mainly utilizes additive and subtractive techniques. Our decision is based on the following factors:

(i) *Novelty and Creativity*: The goal here is to work within the limitations of any tractable sound source to create its approximations of a given sound category. We seek to create novel sounds via artificial, exploratory cre-

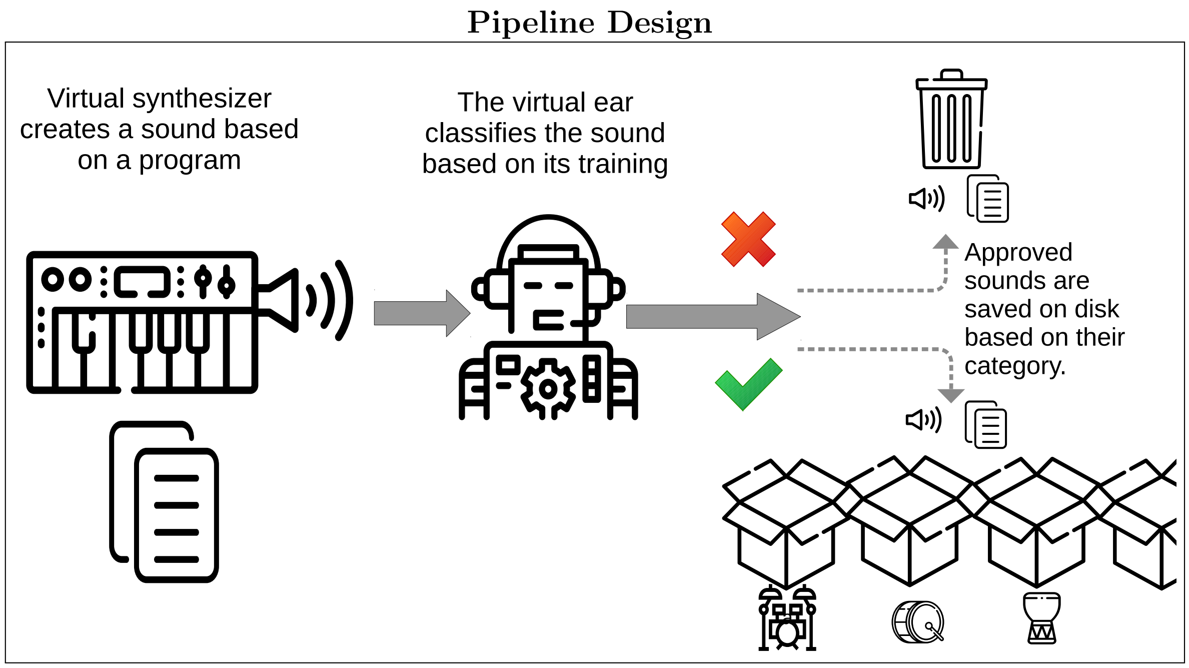

Pipeline Design
Virtual synthesizer creates a sound based on a program
The virtual ear classifies the sound based on its training
Approved sounds are saved on disk based on their category.

Figure 4.1: An implementation which allows for easy parallelization when needed. Virtual synthesizer rapidly generates random programs and the corresponding sounds, while the virtual ear will listen to the sounds and determine if they should be categorized as drums and if so, which category of drum do they belong to.

ativity. Boden defines this concept as an emergent property of generative work within confined rule sets [12]. An example is the perpetual popularity of 8-bit aesthetics in genres such as "chiptunes" [16].

(ii) *Interpretability*: Neural networks are often described as black boxes with uninterpretable weights [8]. Their highly recursive structure makes modern explanation methods such as saliency maps unreliable [65].

(iii) *Speed of Rendering*: Neural network synthesis is costly. Sub 24 kHz sample rates are common in most relevant works [72, 57, 5, 62]. This is far below CD quality sampling rates [63]. At our fixed sampling rate of 48 kHz, synthesizers with 8 submodules can create and save 1 second sounds to hard-disk with an average rendering time of 50 milliseconds[1].

(iv) *Flexibility and Scaling*: Probabilistic audio generation is often done se-

[1]Using a single process on a Macbook Air 2012 and Ubuntu 18.04

quentially. State of the art, parallel wave generation with GANs requires a fixed amount of rendering time for each time-step [72]. With our virtual synthesizer, the added footprint of increasing the length of rendered sounds or higher sampling rates is relatively minuscule.

## 4.3 Virtual Synthesizer Implementation

To create sounds, we make use of digital synthesizers capable of rapidly receiving or creating programs, as well as rendering the corresponding sounds offline. Classical DSP allows for quick, offline, and parallel generation of audio signals without the usage of GPUs. Pippi[2] and SciPy [36] libraries were extensively used for their DSP functionalities.

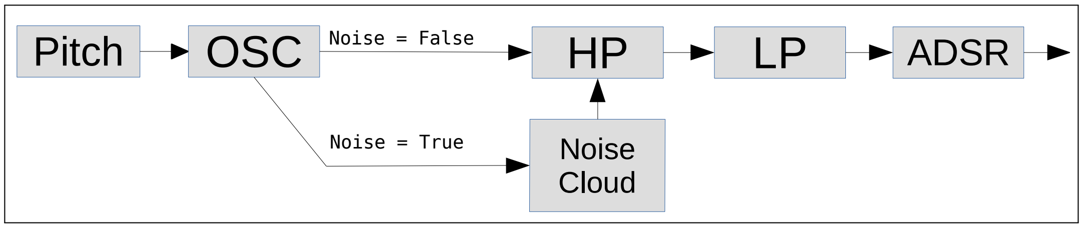


Figure 4.2: High level representation of a submodule. Each Synthesizer contains 1 or more submodules. Synthesizer programs set the number of these submodules and their parameters. A list of notes (Pitch) are sent to the oscillator (OSC), the resulting waveform may or may not be passed through a clouding effect. Next, a high-pass (HP) and a low-pass (LP) filter are applied. Finally, the attack, delay, sustain, and release periods of the sound are defined by the ADSR envelope. The full list of synthesizer block parameters are shown in Table 4.1.

The virtual synthesizer contains a set of one or more submodules. Each submodule is a self-contained noise making unit. Submodules have identical structures, but widely different outputs can be achieved depending on the values assigned to their parameters. The output of the virtual synthesizer is the normalized addition of the output of its submodules. As shown in Figure 4.3, each submodule creates a sound independently, and the results are added and normalized to create the final output. The synthesizer can have any

[2]https://github.com/luvsound/pippi

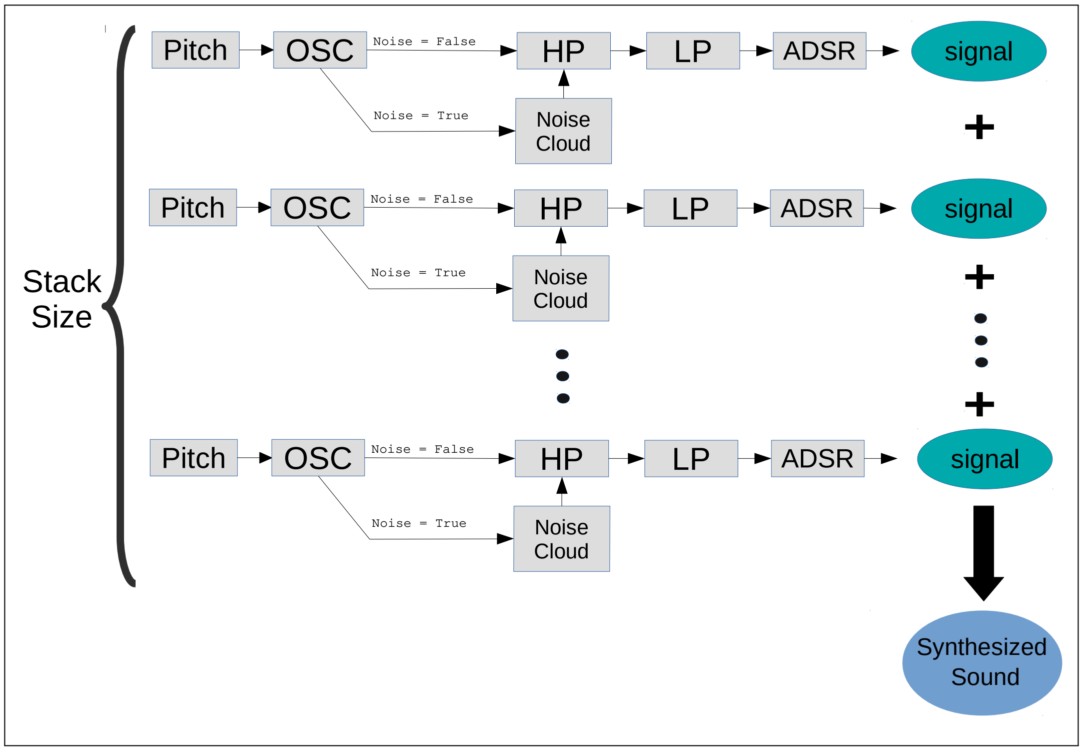


Figure 4.3: The output of the virtual synthesizer is the normalized addition of the output of its submodules. A synthesizer can have any number of submodules.

| Parameters | Value Range | notes and constraints |
|---|---|---|
| Attack | 0-3 | A-D-S-R values relative |
| Decay | 0-3 | relative to A-S-R |
| Sustain | 0-3 | relative to A-D-R |
| Release | 0-3 | relative to A-D-S |
| OSC type | sine,square,saw | tone type |
| IsNoise | boolean | whether touse OSC type to generate noise |
| Length | 0-1 second | - |
| StartTime | 0-1 second | Length+Start<1 |
| Amplitude | 0.1-1 | 1 = max amplitude |
| Pitches(notes) | list of pitches | range of C0(16.35hz) to B9 |
| HP filter Cutoff | 0-20000hz | - |
| LP filter Cutoff | 20000-HP | never lower than HP cutoff |
| Filter Order | 4,8,16 | butterworth filter order |

Table 4.1: Synthesizer submodule parameters. Despite the simplicity of the parameters and efforts at constraining the ranges, the combinations of parameters that can be randomly chosen for each submodule is in the order of $10^{15}$

.

number of submodules. The parameters that dictate the output signal of each submodule as well as the range of values each parameter can take are shown in Table 4.1. We call the number of submodules in each virtual synthesizer the *stack size*. We call the sets of parameter values that characterize a synthesizer's submodules a *program* (analogous to a preset for a VST).

Since we are interested in short, one-shot percussive sounds, each virtual synthesizer program will generate a 1 second piece of audio. This 1 second limit is over twice the length of the average one-shot drum sample in MixedDB (around 0.4 seconds). Each submodule can make an audio signal with the length of 0.1-1 second, and play it at any point within the 1 second rendering time[3].

In Section 2.1.2, we discussed ADSR envelopes, which are used in sound synthesis for modulating the loudness of sounds overtime. Here, each submodule creates its signal at full amplitude before shaping it according to its internal ADSR parameter. Prior to being applied to the signal, each of these parameters is assigned an integer value in the range of 0-3, and normalized

[3]The entire sound must fit within the second, for example, a 0.3 second sound cannot begin playing past 0.7 seconds into the rendering time frame

relative to the others such that

$$A_{norm} + D_{norm} + S_{norm} + R_{norm} = 1$$

Where each value $v_{norm}$ in the $\{A_{norm}, D_{norm}, S_{norm}, R_{norm}\}$ set is normalized such that:

$$\text{for each } v \,\epsilon\, \{\text{A,D,S,R}\}$$
$$v_{norm} = \frac{v}{A + D + S + R}$$

The OSC type will determine the wave-shape of the signal. This parameter is limited to three fundamental wave forms: sine waves, square waves and saw waves. We also allow the creation of noise signals, which can imitate timbral characteristics of higher pitched drum samples at a very low computation cost, compared to the addition of thousands of sine waves at various frequencies. If the "IsNoise" boolean is set to true, the OSC type parameter loses importance as the wave type will be routed through a noise-cloud.

Each submodule is a monophonic synthesizer. That is, each submodule can play one note (or frequency) at a time. However, quick changes in pitch can occur in drum sound. To mimic such sounds, synthesizer submodules may slide between 4 different pitches in the 1 second time frame. Each pitch value is a note with a frequency and length value. Each submodule accepts a list of 5 consecutive possible pitch values. The submodule will play each note in the list consecutively after normalizing the length values. The pitch notes are played in a portmanteau fashion such that there is no audible gap. This normalization of length values is similar to that of the ADSR values.

### 4.3.1 How Will The Synthesizer Be Used?

We use this synthesizer as our source of noise generation. A stack number can be picked randomly, and for each submodule in the stack, the parameter space can be randomly sampled. This gives us a randomly generated *program*, which is then given to the synthesizer for rendering. These randomly generated sounds can then be listened to in order to find interesting sounds. The goal is to automate the listening procedure for rapid generation of desirable sounds, which are drums in the scope of this project. In the next Chapter, we will discuss the virtual ear, which can automatically listen to the outputs of this synthesizer and separate percussive sounds from non-percussive sounds.

# Chapter 5

# The Virtual Ear

## 5.1 Why Do We Need A Virtual Ear?

In the previous chapters, we conceptualized a system that can generate new drum sounds using a virtual source of random sounds, and an automatic ear that can pick out sounds which resemble drums. The virtual synthesizer was introduced in the previous chapter. What is needed now is a tool that can automatically separate the sounds resembling percussion from those which do not. Here, what we refer to as an "ear" is any method of scoring and classifying audio (e.g machine listening) [44, 64]. The ideal virtual ear will be capable of receiving a piece of audio and scoring it based on how well it satisfies certain criteria. In the scope of this project, the virtual ear must receive the random sounds generated by the virtual synthesizer, and separate those resembling drums from non-drums, and categorize the type of drums.

## 5.2 Implementation Steps

To create a virtual ear, we use machine learning algorithms. In order for these algorithms to work as intended, they must be given examples of drum and non-drum data. Here, the datasets of drum and non-drum sounds discussed in Chapter 3 are utilized. In order to speed up training times, we do not train our models on raw audio data, rather, we define two different transformation functions which project audio data into smaller spaces. These transformation functions can also be thought of as feature extraction steps. These functions

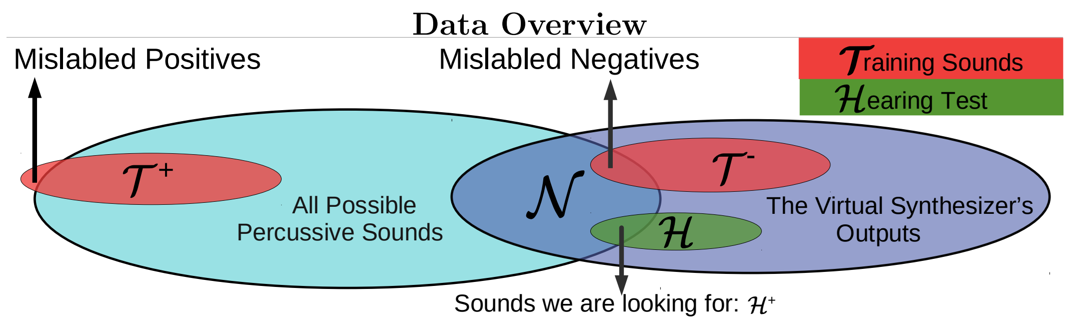


Figure 5.1: An illustration of the discrepancy between the sounds used to train the classifiers and the type of sounds the classifier is expected to classify. $\mathcal{N}$ is the hypothetical set of sounds our synthesizer is capable of making that could be used as percussion. The inclusion of sounds in $\mathcal{N}$ may vary from person to person. The positive samples, $\mathcal{T}^+$, represents the percussive sounds we have in our datasets, a small portion of which may be mislabeled. $\mathcal{T}^-$ is a set of sounds produced by our synthesizer that are used as negative examples; however, a small portion may be similar to percussive sounds and can be thought of as mislabled negatives. $\mathcal{H}$ is the set of sounds used for manual surveys (See Chapter 6).

are discussed in Section 5.3. In Section 5.4, the implementation and measurements of two different model types are discussed.

### 5.2.1 Learning Caveats

To train the models, percussive sounds are used as positive examples, while synthesizer sounds are used as negatives. Figure 5.1 highlights critical problems with the learning approach. The change in learning domains—particularly in the case of positive examples of percussive instruments—should not interfere with transformation of knowledge from the training of the ear to the hearing test.

Traditional classification tasks often assume that the data points used for training the model and future unlabeled data will emerge from the same system or processes [24, 51]. This assumption requires that sufficient positive examples of all possible classes exist and are trained on. Works which involve the implementation of GANs have documented scenarios in which networks will assign high categorization probabilities to nonsensical, out of context data which should be rejected rather than categorized [24, 51, 28]. This issue is reflective of the open set recognition (OSR) problem [24, 51], where learning by examples can prove difficult when the category that is being learned is infinitely large.

We consider drum sounds to be a closed set, since it is reasonable that a sufficiently large sample pack can effectively describe conventional drum categories, while effective representation of all possible non-drum sounds is not attainable via examples alone. A hurdle towards the implementation of a "drum from non-drum" recognizer is that the set of sounds that are not drums do not share any characteristics beyond "not being a drum", making the learning process using negative examples difficult. To ensure the quality of the results, particularly in the categorization of drums from non-drums, manual hearing tests need to be conducted to measure the quality of the results. The results of our blinded hearing tests are discussed in Chapter 6.1.

## 5.3 Representing Sound

We are looking to implement machine learning models capable of analyzing sounds. Here we establish how digital sounds will be transformed into smaller feature spaces before discussing the algorithms which learn from them. In this project two different approaches towards this goal are taken:

1. By using features extracted from Fourier Transformations of sounds
2. By using features extracted from autoencoders which condense the Fourier transformations.

These approaches will be discussed in detail within this section.

### 5.3.1 Fourier Transforms

In this work we rely on the application of the fast Fourier transform (FFT), and by extension, short-time Fourier transforms (STFT) for feature extraction. Various works have demonstrated effective reconstruction of signals given their short-time Fourier transforms [52, 27]. If the STFT of a signal can be used for its reconstruction, perhaps it can be utilized as a source of fundamental features [38, 34]. Using FFT, a signal can be represented by a vector with each index corresponding to a frequency-bin (a range of frequencies too close to be distinguishable) and the value at each index corresponding to the combined-magnitude of the frequencies within the bin. STFT can be employed when temporal changes in frequency bins are of interest; this can be done by the application of FFT[1] to a sliding time-window on the signal to create a vector for each time step. This matrix can effectively represent the frequencies present in the signal at each time step, given the right window-length and hop-size (how much the window is shifted at each time-step).

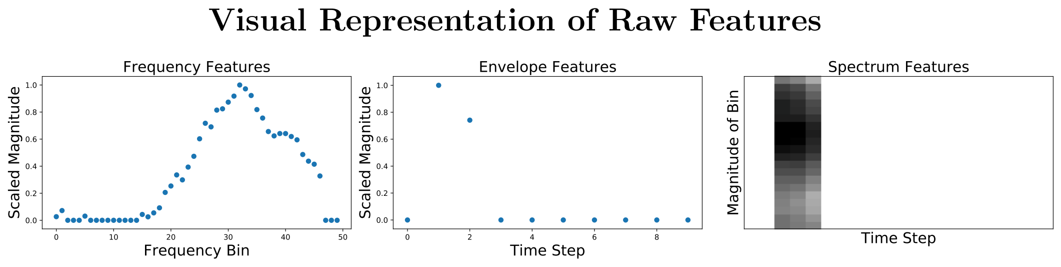
To extract the FFT feature sets from a signal, we defined the following 3 transformation functions. These functions are applied at training time to transform raw digital signals into simpler formats, which may be easier to learn from:

1. Envelope Transformation: The goal of this feature is to capture the changes in loudness for the duration of the signal. Using STFT we generate a matrix $M_{i \times j}$ with rows $i$ and columns $j$ corresponding to time steps and frequency bins respectively, and with values $v_{i \times j}$ indicating the magnitude of the frequency bin $j$ at each time-step $i$. Information about the envelope of the signal can be extracted by summing the values of $M$ for each time-step (or row $i$), giving us a feature vector $v_i$. For each sound, this vector is normalized to the range of 0 to 1. The information contained in this vector is similar to that of an RMS measurement.

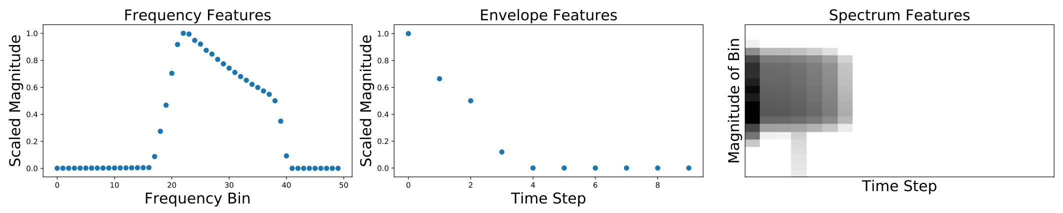
2. Frequency Transformation: A static, normalized snap-shot of the frequencies present within the audio. The calculation of this feature vector is similar to the envelope, but the summation is done along the frequency axis.

3. Spectrum Transformation: STFT with its values normalized from 0-1 per sample. Since this feature is a 2D matrix rather than a vector, it captures more information about our signal. This spectrogram is "mel-scaled"[70] to better represent human perception of audio frequencies.

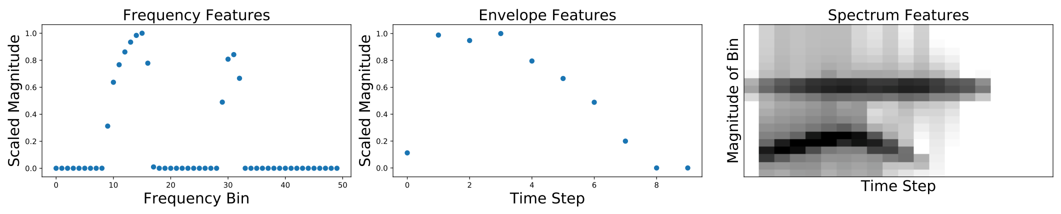
[1]more accurately, discrete Fourier transforms



(a) Recorded hat sample



(b) Randomly generated audio with percussive qualities, resembling a tight snare



(c) A randomly generated noise with a percussive envelop but non-percussive frequency features (modulated pitch)

Figure 5.2: Graphed representation of features extracted for 3 different samples. Sample *a* is a recorded hat from our database. sample *b* is an example of randomly generated noise with percussive qualities that we found suitably similar to a snare sound. Sample *c* is an example of a randomly generated noise where the spectrum features are necessary for proper classification.

Once the transformation functions are applied to a signal, we have 3 feature sets which can be learned from. The Frequency bin and envelope features are 1 dimensional arrays of values in the 0-1 range, and are given directly to the learning models. The spectrum features are a 2 dimensional array of values in the 0-1 range, which can be thought of as a greyscale image. Depending on the the learning algorithm, the spectrum features may be flattened into a single dimension (e.g fully connected neural net) or left unchanged (e.g convolutional neural net).

In Section 5.3.2, we discuss how the spectrum features can be further encoded, or condensed, using autoencoder neural networks. The procedure of learning from these feature sets is covered in Section 5.4. Section 5.4 discusses the training procedure for models which directly use the FFT features described above, and models which use encoded spectrum features are discussed in Section 5.4.2.

### 5.3.2 Embedded FFT Features

The advantage of the spectrum features discussed earlier is that they contain temporal information for the amplitude of each frequency bin. This means that it contains both envelope and frequency features, along with other possibly useful information. This comes at the cost of requiring a much larger representation space than either frequency or envelope features. To speed up and simplify training, we want to further reduce the dimensionality of the spectrum features, ideally without the loss of vital information needed for classification. For this purpose, we utilized autoencoder networks, which are an approach to dimensionality reduction using deep neural networks [30, 29].

Autoencoders are comprised of an encoder and a decoder. The encoder has the task of meaningfully projecting the spectrum data of each sound onto a small bottleneck layer, while the decoder aims to use this projection to replicate the input data as closely as possible. Successful training will lead to an encoder network capable of projecting audio data into a low dimensional vectors that can be reconstructed faithfully. Here, the original data is the spectrum features, and the encoding at the bottle neck layer is the simplified

**AutoEncoding Spectrograms**

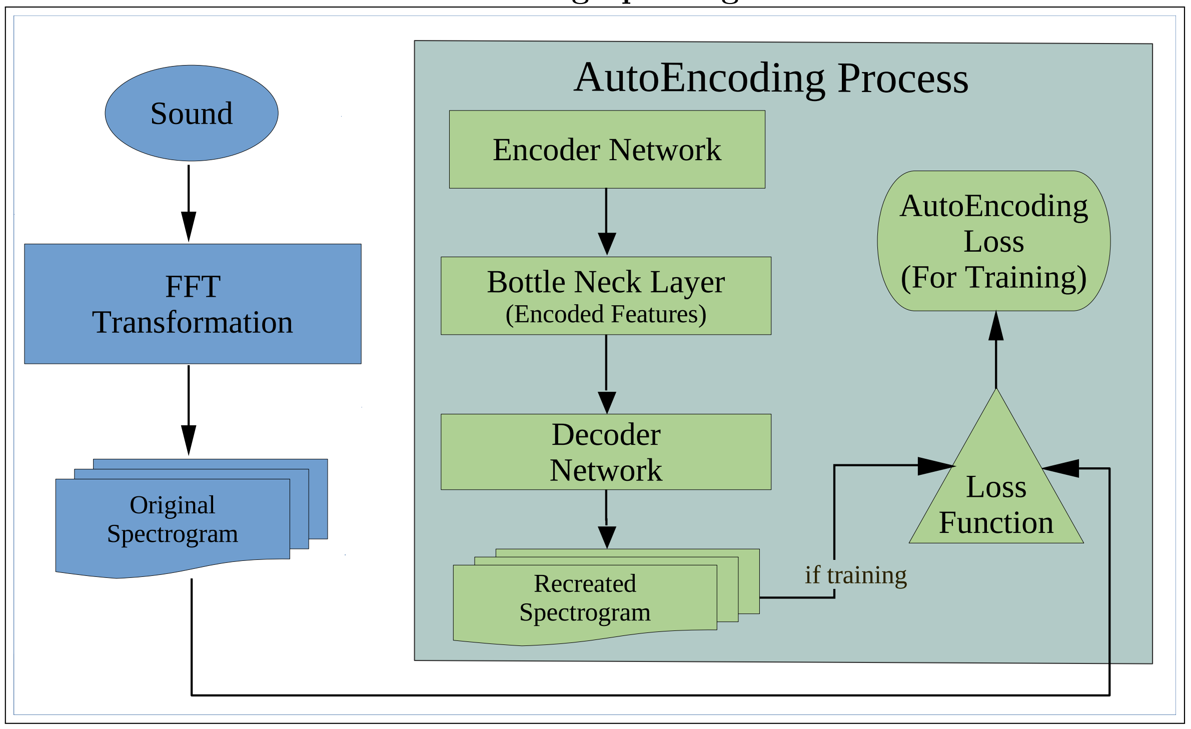


Figure 5.3: Overview of autoencoder training. Once an autoencoder is trained, the decoder and loss function are not needed, and the bottleneck layer values will be used as features.

feature sets which we would like to use for training. With some luck, this low dimensional vector will contain vital information which can be used for training models much faster than using the original data, with little loss in performance. The autoencoder training process is depicted in Figure 5.3.

### 5.3.3 Architecture and Hyper-Parameter Optimization

We make the assumption that if an autoencoder can accurately encode and decode a spectrogram, then the values at the bottleneck layer form a viable alternative to the spectrogram features, while being much smaller in size. Based on this assumption, the optimization goal when training the autoencoder networks is to minimize the difference between the original and decoded spectrogram. The autoencoder designs used in this project were kept relatively simple, all with approximately 150,000 parameters (although a significant reduction occurs when the decoders are discarded during feature extraction).

| Hyper-Param. | Description | Values | Distribution |
|---|---|---|---|
| Model Type | Affects encoder's first hidden layer | CNN,FC | Categorical |
| Optimizer | Updates network's weights based on loss | Adam,SGD | Categorical |
| Hidden Layers | Extra hidden layer for the Encoder | True,False | Categorical |
| Time Steps | Temporal granularity of the spectrogram. Affects FFT windowing. | 10,20 | Categorical |
| Learning Rate | Optimizer's learning rate | $1^{-4}$ ... $1^{-1}$ | Uniform |
| Frequency Bins | Number of spectrogram frequency bins | 10, 30,60 | Categorical |
| Regularization | L2 regularization parameter. Penalizes large weights to prevent overfitting | $1^{-6}$ ... $1^{-1}$ | Uniform |
| Latent Size | Size of bottle neck layer or number encoded features | 8,16,64 | Categorical |
| Dropout Rate | Random zeroing of activations between layers to prevent over-fitting | 0,0.5,0.1 | Categorical |

Table 5.1: The Hyper-Paramter space in which the optimization was conducted.

Important to mention is that reported loss values of the models do not necessarily reflect whether the encoder will capture data that is useful for any purpose other than being utilized by the decoder (such as classification and categorization of drums).

These small networks train quickly without the need for much data. Here, we did not give any examples of synthetic noise to the encoders, and only train on our datasets of conventional drums. We hypothesize that if a synthetic noise has drum-like characteristics, its compressed, latent representation given by the encoder will be similar to the drum sounds in our databases and vice versa. In Section 5.11, we use these latent expressions to train several machine learning models.

Within the context of machine learning, a model's *hyper-parameters* are fixed parameters which are set before the training begins (e.g., number of layers, size of layers, loss function) and are not learned during the minimization procedure [9]. Also within this context, hyper-parameter optimization is the task of searching for a set of hyper-parameters which would maximize the model's performance, often done by a series of automatic test trials [9, 11, 10]. As a wide variety of viable autoencoder architectures have been proposed [5, 20, 25, 74, 61], we are faced with a number of choices for autoencoder design. To assist with the construction of the model, a number of possible choices for the architecture and the transformation function were defined (see Table 5.1). These choices were then used in hyper-parameter optimization to extract promising sets of values.

The list of possible choices for the hyper-parameters can be found in Table 5.1. The batchsize $\mathcal{B}$, was set to 64 for training and 4 for testing. We included not only model parameters but also spectrogram transformation parameters within this search space, as GPU accelerated FFT calculations allows ad hoc audio transformations to take place parallel to the training process. We implemented 3 base models which are affected by these hyper-parameters. The "Model Type" parameter dictates whether CNN or fully connected models are selected. If a "fully connected" model is selected, the "hidden layers" parameter selects between the two implementations. The specifications for these models can be found in Tables 5.3, 5.4 and 5.2.

Using the optuna optimization tools [1], we conducted 500 search trials, where each trial consisted of up to 20 epochs of training using a fixed set of hyper-parameters, and each epoch consisted of 1 training round using the entire dataset. The trials' success is measured in their final loss value, calculated by applying the model to test data-set. Here, the test data-set is all sounds from MixedDB. An additional 100 trials with 40 epochs of training were conducted following the initial 500 to test the effect of longer epochs. Each trial's intermediate results (loss at every n epochs where $0 < n < 20$) were reported to a multi-armed bandit based pruner for early stoppage of unpromising trials[40]. We employed a tree-structured parzen estimator for better navigation of the search space [11, 1] but found short reversions to a random sampling coupled with a decrease in the frequency of pruning helpful in exiting local minima.

| Layer-# | Output Shape | Param Num | Details |
|---|---|---|---|
| Conv2d-1 | [$\mathcal{B}$, 8, 30, 20] | 208 | Encoder's input<br>Num. Channels:8<br>kernel:5x5<br>stride:1<br>padding:2 |
| ReLU-2 | [$\mathcal{B}$, 8, 30, 20] | 0 | |
| MaxPool2d-3 | [$\mathcal{B}$, 8, 15, 10] | 0 | kernel:5x5<br>stride:2 |
| Dropout-4 | [$\mathcal{B}$, 8, 15, 10] | 0 | |
| Linear-5 | [$\mathcal{B}$, 8] | 9,608 | Encoder's output |
| Linear-6 | [$\mathcal{B}$, 256] | 2,304 | Decoder's Input |
| Dropout-7 | [$\mathcal{B}$, 256] | 0 | |
| Linear-8 | [$\mathcal{B}$, 600 ] | 154,200 | Decoder's output |

Table 5.2: CNN model design with latent size of 8. 30 and 20 are the assumed frequency bins and step size. Total number of parameters is 166,320.

| Layer-# | Output Shape | Param Num | Details |
|---|---|---|---|
| Linear-1 | [$\mathcal{B}$, 128] | 76,928 | Encoder's input |
| Dropout-2 | [$\mathcal{B}$, 128] | 0 | |
| Linear-3 | [$\mathcal{B}$, 8] | 9,608 | Encoder's output |
| Linear-4 | [$\mathcal{B}$, 128] | 2,304 | Decoder's Input |
| Dropout-5 | [$\mathcal{B}$, 128] | 0 | |
| Linear-6 | [$\mathcal{B}$, 600 ] | 77,400 | Decoder's output |

Table 5.3: Fully connected model with only 1 hidden dimension for encoder and decoder. Design assumes latent size of 8. 30 and 20 are the assumed frequency-bins and step-size values. Total number of parameters is 156,512.

| Layer-# | Output Shape | Param Num | Details |
|---|---|---|---|
| Linear-1 | [$\mathcal{B}$, 128] | 76,928 | Encoder's input |
| Dropout-2 | [$\mathcal{B}$, 128] | 0 | |
| Linear-3 | [$\mathcal{B}$, 32] | 4,128 | |
| Dropout-4 | [$\mathcal{B}$, 128] | 0 | |
| Linear-5 | [$\mathcal{B}$, 8] | 9,608 | Encoder's output |
| Linear-4 | [$\mathcal{B}$, 32] | 2,304 | Decoder's Input |
| Dropout-5 | [$\mathcal{B}$, 32] | 0 | |
| Linear-4 | [$\mathcal{B}$, 128] | 2,304 | |
| Dropout-5 | [$\mathcal{B}$, 128] | 0 | |
| Linear-6 | [$\mathcal{B}$, 600 ] | 77,400 | Decoder's output |

Table 5.4: Fully connected model with 2 hidden dimensions for encoder and decoder. Design assumes latent size of 8. 30 and 20 are the assumed frequency-bins and step-size values. Total number of parameters is 163,232.

We depict the correlation of parameter values with trial results in Figures 5.6. We extract a more concrete measurement of hyper-parameter influence using the fANOVA importance evaluator [33]. We limit this analysis to 500 trials with 20 epochs. The results of this estimation are depicted in Figure 5.7. Contrasting the results of the fANOVA evaluator with Figure 5.6, we notice several issues. Notably, no importance is attributed to "Model Type" and "Optimizer" parameters, despite a visible difference depicted in the slice-graph (Figure 5.6). This may be due to the imbalanced sampling of the hyper-parameter space, prompted by our greedy search in place of random sampling. Another possible cause is the "averaging" of loss results regardless of variance: For example, Figure 5.6 depicts CNN models as having the best and worst results while FC models are reliably average; making the fANOVA's 0 importance attribution logical, but not optimal when we are strictly looking for the best models.

### 5.3.4 Visualizing The Encodings

To better understand the delineation potential of the encoded features, we trained an autoencoder with our top performing hyper-parameter set, shown in

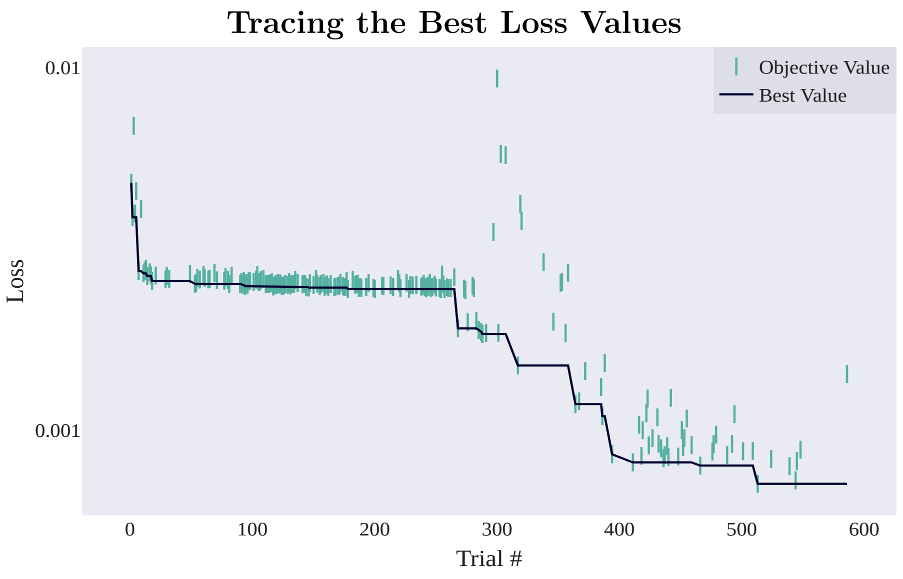


Figure 5.4: Smaller objective values imply better hyper-paramter sets. As more trials are executed, how quickly does the best found objective value decrease? The goal here is to run trials that return the smallest objective values possible. The effect of a switch to random sampling and an increase of the pruning threshold can be observed during trials 270 and 310.

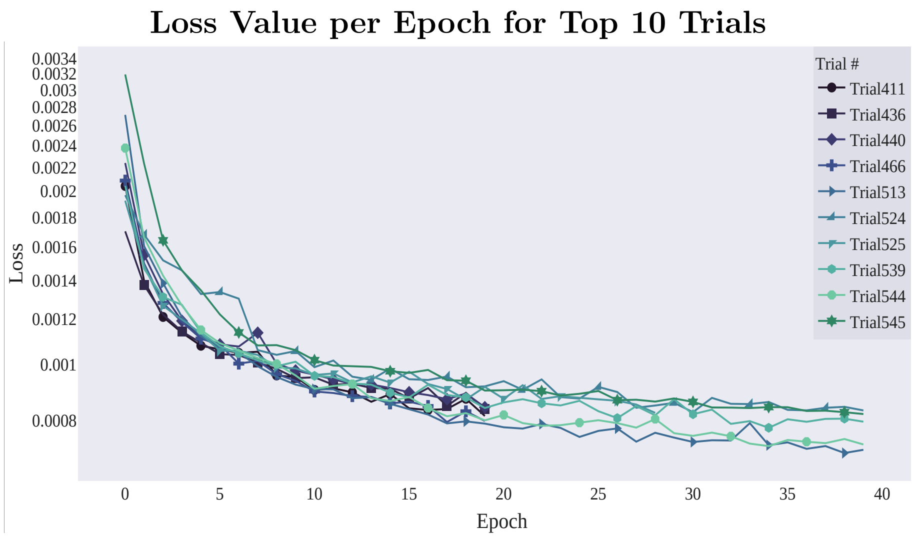


Figure 5.5: Loss value per epoch for top 10 trials. Some of the initial 500 trials appear in this list, despite having half the number of epochs. The learning curves tend to flatten quickly. Therefore, 20 epochs may be a reasonable number for measuring hyper-parameter viability.

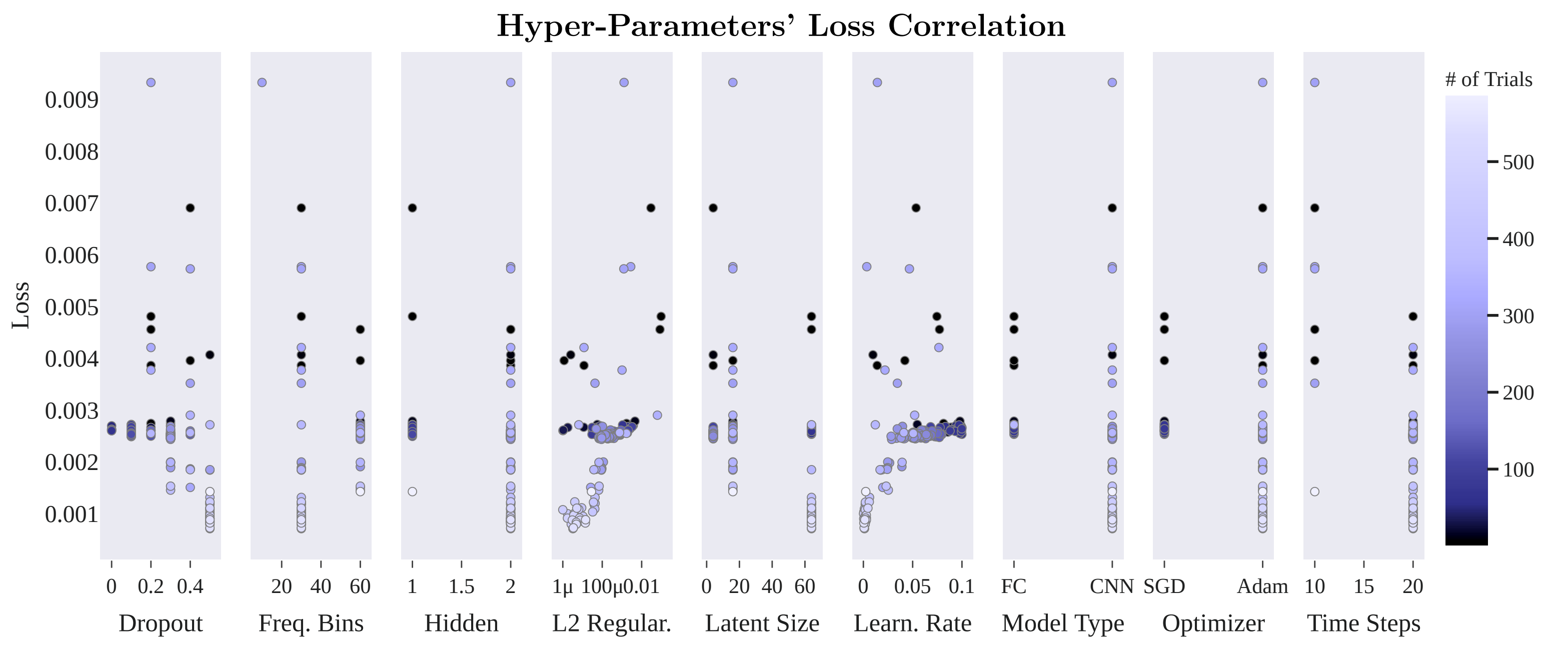


Figure 5.6: Sliced plot depicting the correlation between hyper-parameters and loss values. The color-scale shows the number of times each parameter has been used in a trial. Our sampling algorithm aims to utilize spaces with higher potential more often. The effect of higher resolution in the frequency and time dimensions of the spectrogram are notable.

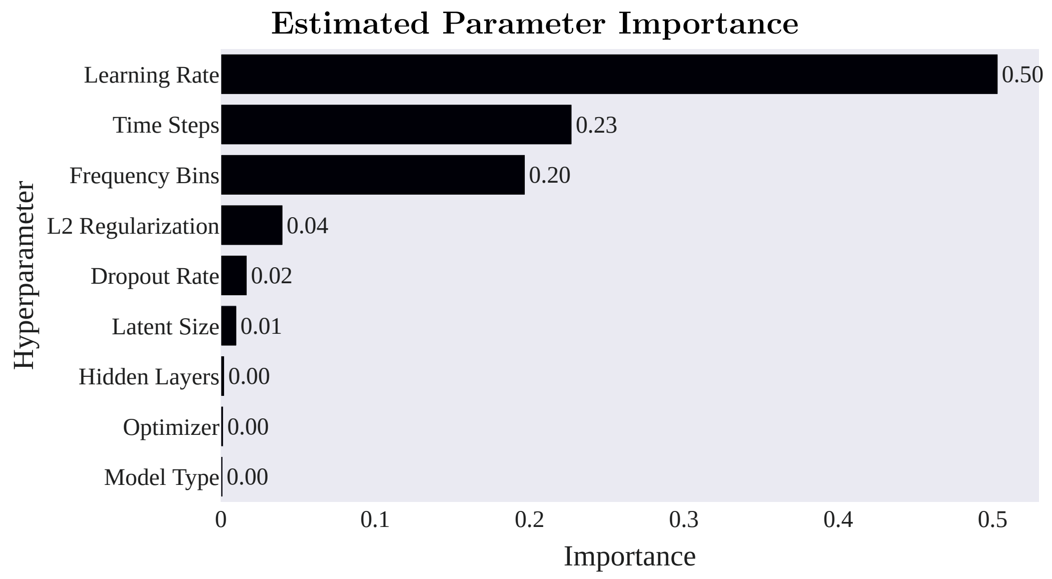


Figure 5.7: The parameters' estimated importance in determining the outcome of trials. Specifications of the spectrogram seem to affect the outcome more so than the model's configuration. We attribute the contrast between the results here and those in Figure 5.6 to the irregular rate of sampling from the hyper-parameter space.

| Hyper-Param. | Value |
|---|---|
| Model Type | CNN |
| Optimizer | Adam |
| Hidden Layers | 1 |
| Learning Rate | 0.001145 |
| Frequency Bins | 30 |
| Time Steps | 20 |
| Latent Size | 64 |
| Regularization | $3.25^{-6}$ |
| Dropout Rate | 0.5 |

Table 5.5: Top performing hyper-parameter set

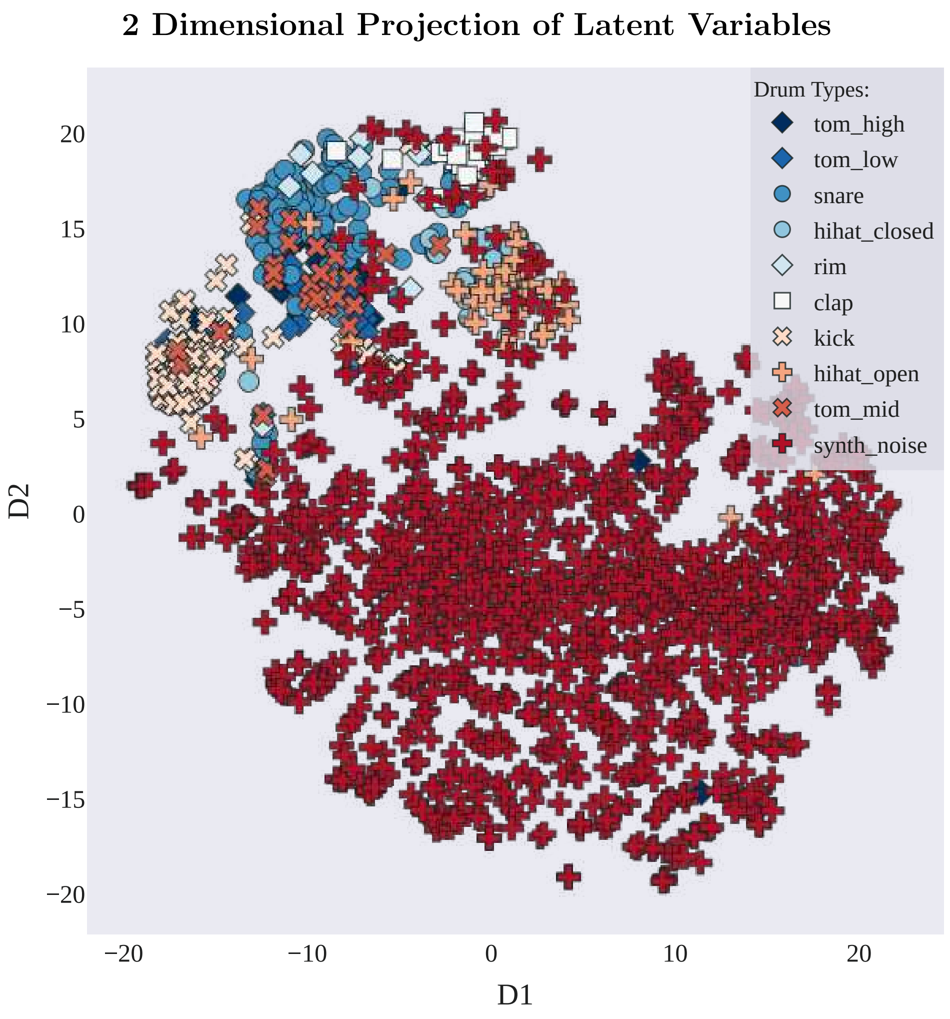


Figure 5.8: Projection of an embedding model's low dimensional encoding on to a 2D plane. D1 and D2 are the two dimensions output by T-SNE. We implemented interactions for these graphs for manual inspection of samples.

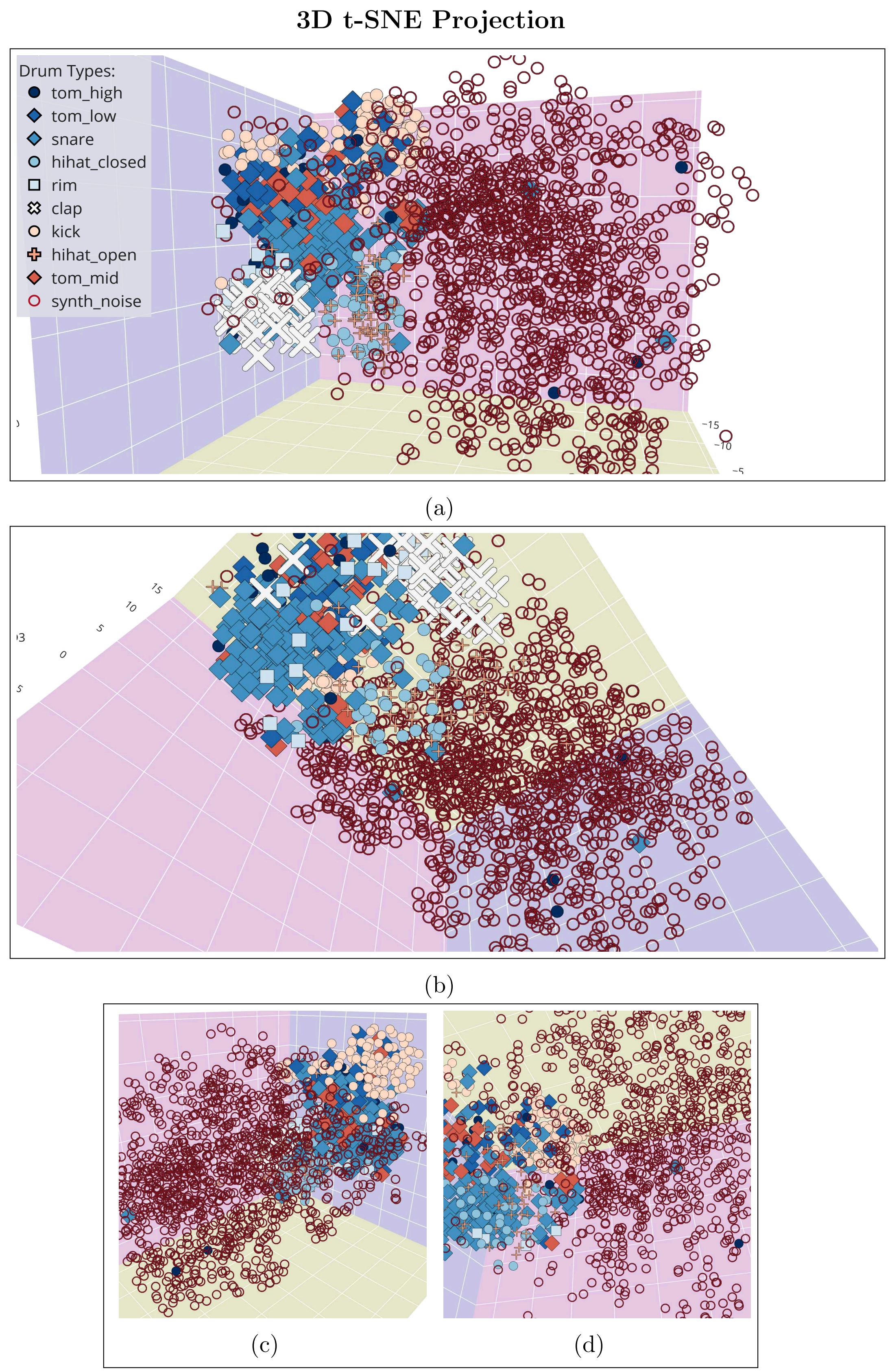


Figure 5.9: Our feature projections and interactive graphs can also be done in 3D

Table 5.5. We used 80% of MixedDB to train the model for 200 epochs. Using this model, we encoded the remaining 20% and approximately 1000 randomly selected sounds from NoiseDB into 64 dimensions. These 64-dimensions were then projected onto a 2 or 3-dimensional plane using t-SNE.

In Section 5.4, we will thoroughly discuss the expectations of an effective virtual ear. In summary, if the embeddings are useful in helping the virtual ear with separation of drums from not-drums as well as categorization of drum types, we expect the projection of drum sounds to be clustered together, with the majority of the synthetic noise sounds appearing away from these clusters. Our projections confirm that this is the case. We are most interested in synthetic noise sounds which appear within or near these clusters. Do these synthetic noise sounds have drum like characteristics?

To test this, we implemented interactive graphs identical to those depicted in Figure 5.8 and 5.9. These graphs allow for interactions such as playing the samples (by hovering the cursor on-top) and movement of the scene camera for a closer look at clusters. Manual inspections detailed in Chapter 6 reveal a noticeable, positive correlation between distance and similarity of a synthetic sound to a drum cluster. Divergences from the envelope features expected from drums are a common point of failure. A noticeable case was a synthetic noise resembling a kick drum played in reverse within the kick cluster. While encouraging, we hypothesize that more specialized forms of encoding or addition of other features are needed for reducing the frequency of errors and strengthening this correlation.

### 5.3.5 What To Do With Extracted Features

So far, we discussed two methods of feature extraction for audio. The FFT features can represent various aspects of sound which may be helpful to a virtual ear. Using autoencoders, FFT features can be further condensed into small embedded spaces. These embedded spaces are harder to understand and explain, but the 2-dimensional projections shown in Figure 5.8 indicate that they contain vital information about the different drum and non-drum sound types in our databases.

The features discussed here can be used to train classifiers which can separate desirable sounds from undesirable sounds. Section 5.4 covers the implementation of classifiers which directly use the FFT features and Section 5.4.2 covers the use of embeddings to train virtual ear models.

## 5.4 Ear Decisions

Having considered the caveats and requirements, once a sound is generated and passed onto the ear, we expect the virtual ear to make decisions in response to two important questions:

*Decision.1 Could the sound be used as a drum?*

*Decision.2 If it does sound like a drum, what type of drum should it be?*

*Decision.1* requires knowledge of what drums **do not** sound like, or knowledge of an infinitely large set, which cannot be fully represented via examples. An important consideration is that the source of sounds used for training the model (conventional drum sounds) will be fundamentally different from the source of unlabeled sounds we wish to categorize (i.e., noise from a synthesizer.) This issue is reflective of the OSR problem [24, 51]. Figure 5.1 highlights a number of caveats with our training approach.

Assuming that the result of *Decision.1* is positive, the sound is deemed percussive and the virtual ear makes *Decision.2* by finding the best drum category for the sound. The number of categories available is dependent on the database of drums used for training. Since drum categories are not open sets (i.e., they can be described via examples), the OSR problem should not play a role in the decision making process.

The goal is to create a pipeline of sound generation where the synthesizer is used for the rapid generation of sounds and the virtual ear is used for the acceptance of inputs which satisfy some fundamental characteristics of percussive instruments. The representation of sounds is critical in allowing the acceptance of novel sounds as part of the drum group despite their anomalies.

TPE Design

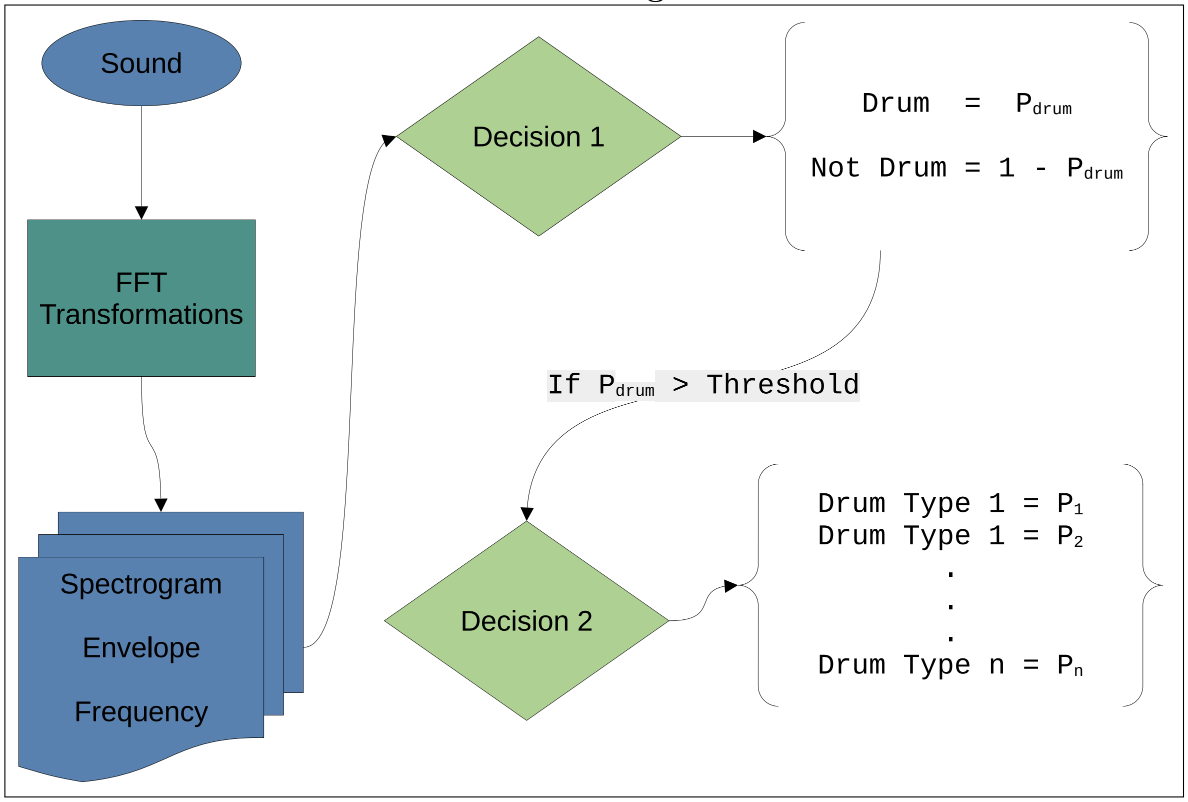


Figure 5.10: TPE's receive a sound and make decisions sequentially.

In Section 5.3, we discussed 2 different approaches to feature extraction. This leads to two different implementations of a virtual ear: *two phased ears* (TPEs) and *mixed ear models* (MEMs). TPEs are a combination different models for each of *Decision.1* and *Decision.2*. The features utilized by these models are manually defined. MEMs use a highly compressed, automatically encoded representation of sound to give simultaneous answers to both questions.

### 5.4.1 Two Phased Ears

Using the features and transformation described in the previous section as input, we trained several neural network models with the pytorch library. The tasks at hand with regards to *Decision.1* is to separate drums from not-drums (DrumVsNotDrum, or DVN). In *Decision.2* the aim is to categorize the type of drums and percussion (DrumVsDrum, or DVD).

Multiple neural network architectures using different subsets of the FFT features to specialize in *Decision.1* or *Decision.2* are combined to make decisions sequentially. For *Decision.1*, all drums in RadarDB and FreeDB and 6000 examples of virtual synthesizer noise are used. For *Decision.2*, we combined the two databases and merge toms into kicks and rims/shakers into "other". To train the models, 80% of the sounds in the dataset were used, and the remaining 20% of sounds were used for validation. The loss function and optimizer are Categorical Cross-Entropy and Adam respectively. Training continued until no reduction in loss and accuracy is observed for 10 epochs. These accuracy numbers are weak as we did not account for category sizes or cross validate. Further details about the design decisions, performance, and the architectures for the DVN models can be found in Section 5.4.1.1. Likewise, the architectures for the DVD models can be found in Section 5.4.1.2.

#### 5.4.1.1 DVN Models

In the TPE approach, DVN models are tasked with the separation of drums from non-drums, or *Decision.1*. Here, we use 2 neural network architectures to train for this task. These architectures are trained on the spectrogram representations of each sound as described in Section 5.3.1, and aim to predict whether a sound is percussive or not. For convenience, we also prefer models which train quickly (under an hour) without parallelization[2].

Since training is done only on small spectrogram features sets, there are benefits and downsides in the utilization of different neural network layers. CNN layers are capable of identifying patterns regardless of their location and orientation within a 2D space [3]. However, here were working with spectrogram data, where the 2D position of each feature is critical to its meaning. CNN layers also have the benefit of being less computationally expensive than fully-connected layers [3]; but models which only utilize fully-connected layers are a viable option here due to the small feature space. Considering these possible benefits and costs as well as our preference for simplicity and quick training, we train two different models for *Decision.1*:

[2]Using NVIDIA GeForce GTX 1070

| Layer-# | Output Shape | Details |
|---|---|---|
| Linear-1 | [$\mathcal{B}$, 600] | Input Layer |
| PReLU | [$\mathcal{B}$, 600] | |
| Linear-2 | [$\mathcal{B}$, 20] | |
| PReLU | [$\mathcal{B}$, 20] | |
| Linear-3 | [$\mathcal{B}$, 10] | |
| Linear-3 | [$\mathcal{B}$, 4] | |
| Softmax | [$\mathcal{B}$, size(DvN)] | DvN Probabilities |

Table 5.6: Sequential Layers for FC-DVN

| Layer-# | Output Shape | Details |
|---|---|---|
| Conv2d-1 | [$\mathcal{B}$, 1, 30, 20] | kernel:7x3<br>stride:1<br>padding:3x1 |
| ReLU-2 | [$\mathcal{B}$, 1, 30, 20] | |
| Dropout-4 | [$\mathcal{B}$, 1, 15, 10] | |
| LSTMCell | [$\mathcal{B}$, 30] | hidden size:800 |
| Linear-1 | [$\mathcal{B}$, 2] | |
| Softmax | [$\mathcal{B}$, 2] | DVN probabilities |

Table 5.7: Sequential Layers for CNNLSTM-DVN

1. **FC-DVN**: A fully connected network trained on spectrogram features which reached 97% accuracy on the test data for *Decision.1*. Architecture details are shown in Table 5.6.

2. **CNNLSTM-DVN**: A combination of CNN and LSTM models, where the CNN model extracts higher level features that are fed temporally to an LSTM cell. Here, LSTM layers are utilized in order to extract temporal features from the CNN layers. This model is trained on spectrum data and reaches 98% accuracy for *Decision.1*. Architecture details are shown in Table 5.7.

#### 5.4.1.2 DVD Models

As FC-DVN and CNNLSTM-DVN appeared effective for *Decision.1*, we trained identical models for the categorization of drum sounds, or *Decision.2*. The only modification being the change in the final output layers depending on the number of drums being categorized. In addition, as we hypothesize that *Decision.2* may be a relatively easier task (a large number of sounds are pruned after *Decision.1*), we trained another fully connected model using only the envelope and frequency features of each sound (i.e, a vector of size 60). These models and their performances are as follows:

1. **FC-DVD**: Fully connected 3 layer neural net with 78% accuracy for 6-way drum categorization in Phase 2. Architecture details are shown in Table 5.8.

2. **CNNLSTM-DVD**: A CNN model trained on Spectrum features. Reaching 82% accuracy in a 6-way drum categorization in Phase 2. Architecture details are shown in Table 5.9.

3. **E+F-DVD**: A fully connected model trained on a concatenation of envelope and frequency features. Reaching 80% accuracy for 6-way drum categorization in Phase 2. Architecture details are shown in Table 5.10.

Based on the results on the testing data, these models show promise in separation of drums from non-drums and categorization of drums, however, we cannot know their effectiveness as part of a generative system of drum sounds until manual hearing tests are done. That is, while the ears may be effective at rejecting non-percussive sounds, we cannot know what percentage of the sounds accepted by these virtual ears resemble drums without manual hearing tests. In Chapter 6, we discuss how the TPE models were used in tandem with the virtual synthesizer along with the results of our hearing tests.

In the next section, we discuss the implementation of MEMs, a separate class of virtual ears which make *Decision.1* and *Decision.2* simultaneously without the need for two different models.

| Layer-# | Output Shape | Details |
|---|---|---|
| Linear-1 | [$\mathcal{B}$, 600] | Input Layer |
| PReLU | [$\mathcal{B}$, 600] | |
| Linear-2 | [$\mathcal{B}$, 20] | |
| PReLU | [$\mathcal{B}$, 20] | |
| Linear-3 | [$\mathcal{B}$, 10] | |
| Linear-3 | [$\mathcal{B}$, Drum Types] | |
| Softmax | [$\mathcal{B}$, Drum Types] | DvD Probabilities |

Table 5.8: Sequential Layers for FC-DVD

| Layer-# | Output Shape | Details |
|---|---|---|
| Conv2d-1 | [$\mathcal{B}$, 1, 30, 20] | kernel:(7 , 3)<br>stride:(1 , 1)<br>padding:(3 , 1) |
| ReLU-2 | [$\mathcal{B}$, 1, 30, 20] | |
| Dropout-4 | [$\mathcal{B}$, 1, 15, 10] | |
| LSTMCell | [$\mathcal{B}$, 30] | hidden size:800 |
| Linear-1 | [$\mathcal{B}$, Drum Types] | |
| Softmax | [$\mathcal{B}$, Drum Types)] | DvD probabilities |

Table 5.9: Sequential Layers for CNNLSTM-DVD

| Layer-# | Output Shape | Details |
|---|---|---|
| Linear-1 | [$\mathcal{B}$, 60] | 10 Envelope + 50 Encoded Features |
| PReLU | [$\mathcal{B}$, 60] | |
| Linear-2 | [$\mathcal{B}$, 30] | |
| PReLU | [$\mathcal{B}$, 30] | |
| Linear-3 | [$\mathcal{B}$, 10] | |
| Linear-3 | [$\mathcal{B}$, 10] | |
| Softmax | [$\mathcal{B}$, Drum Types] | DvD Probabilities |

Table 5.10: Sequential Layers for E+F-DVD

**MEM Design**

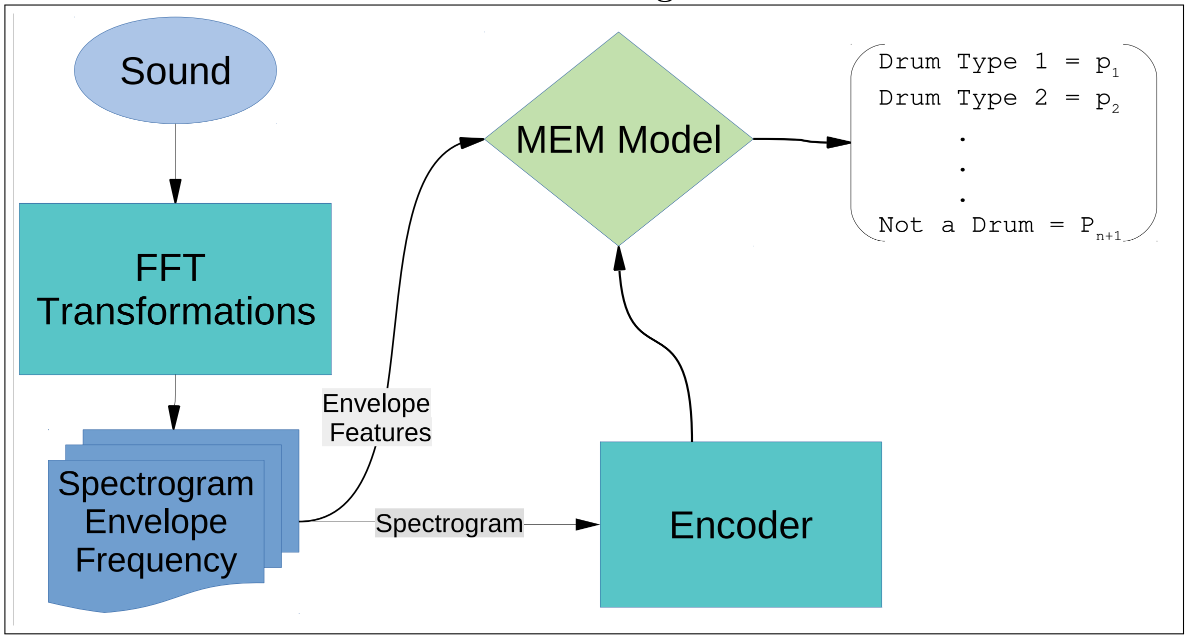


Figure 5.11: MEMs use both FFT features and embedding features to make both decisions simultaneously.

### 5.4.2 Mixed Ear Models

Unlike TPEs which make their decisions in two steps, mixed ear models (MEMs) simultaneously categorize a new sound's drum type or put it in the "synthetic noise" category. We call this task *drum vs drum vs not-drum*, or "**DvDvN**".

Manual t-SNE inspections discussed in Section 5.3.4 highlighted a disregard for envelope shapes as a major source of failure. Because of this, the feature set used to train MEMs contains envelope and embedded features. The performance of the models before and after the addition of envelope features (a vector of size 10) to the feature space is shown in Figures 5.12 and 5.13.

The autoencoder model trained on MixedDB is used as an embedding feature extractor. RadarDB, FreeDB and NoiseDB are combined for training/test data. To prevent class overlaps as much as possible, only claps, hats, kicks, snares and synthetic noise groups are used for measuring model effectiveness. Samples longer than 1 second are excluded in order to reduce potentially mislabeled data. This combined dataset was described in Section 3.2.5

| Model name | Modified Parameters[†] | Used Weights? [‡] |
|---|---|---|
| Support Vector Classifier (SVC) | Gamma:0.001, C:100, kernel:rbf | Yes |
| LinearSVC | C:10 | Yes |
| K Nearest Neighbors | Number of Neighbors:31 | No |
| Random Forest Classifier | Number of Estimators:500 | Yes |
| Extra Trees Classifier | Number of Estimators:1100 | Yes |

Table 5.11: Models implemented for comparison using envelope and embedded features.

[†] Parameters not mentioned have neither been tuned nor changed from scikit-learn's default values (as of version 0.23)

[‡] Class weights are used unless not applicable to classifier.

#### 5.4.2.1 Model Selection

Using embeddings and envelope features to represent audio, five classification models were trained for the task of categorizing the five different sound groups. We did not use neural networks for classification as traditional classification models have been shown to match, if not exceed deep-learning models with less training time, given that the feature set is a simplified representation extracted using neural networks [55].

We arbitrarily select 5 traditional classification models from the scikit-learn library (version 23) [58]. The 5 classification models used and their hyperparameters are presented in Table 5.11, and any unmentioned hyperparameters are set to the library's defaults. To compare the models, we conducted a 10-fold cross validation and plot the F-scores in Figure 5.12. Each of the 10 F-scores reported is the average F-score when classifying individual drum groups. Class weights were used where possible to mitigate the effects of an imbalanced dataset [60, 14]. When utilized, the weight for each class $c$ is calculated as:

$$c_{weight} = 1 - \frac{\text{Number of samples in group } c}{\text{Total number of samples}}$$

To see how these models perform on the binary DvN task (i.e *Decision.1*), the performance of the models is measured in the same manner after all drums are grouped together. The results of these tests are shown in Figures 5.12 and 5.13. Considering the performance of these models, we selected the extra-trees model as the most promising for the MEM virtual ear. Extra-trees is an

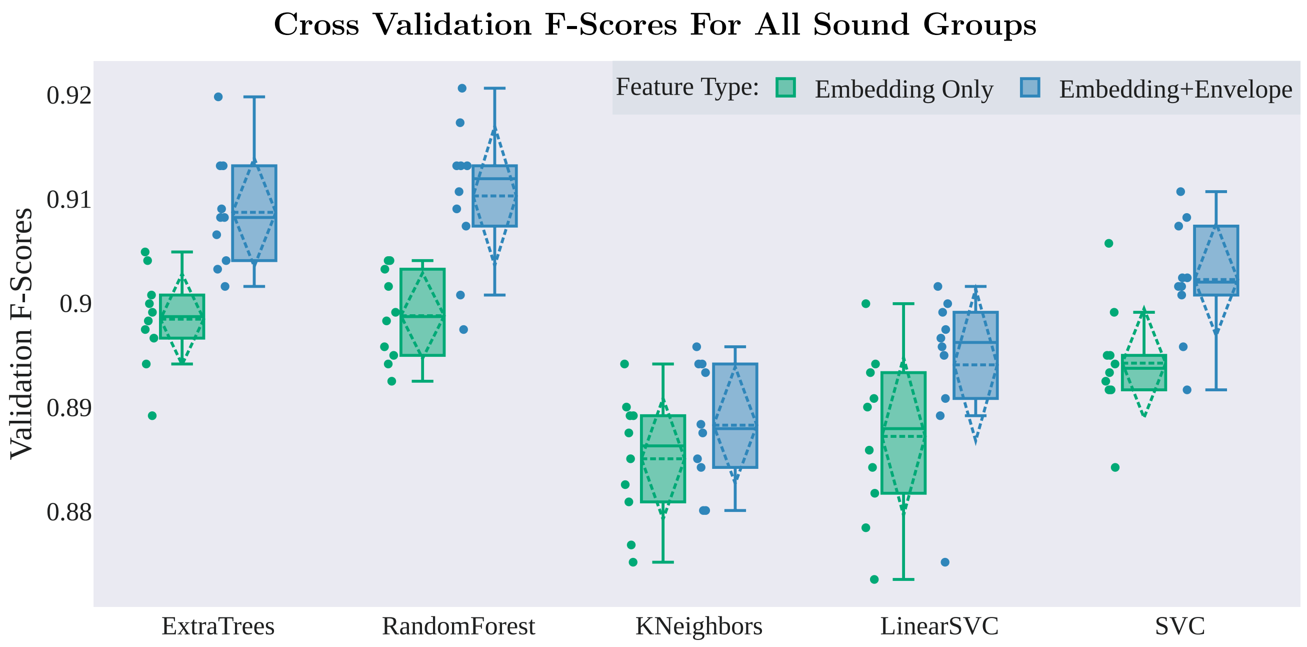


Figure 5.12: Boxplots visualizing the F-Score results for each cross-validation. The individual scores, means, medians, standard-deviation and outliers are depicted. The differences are noticeable, yet means lie within the 88-92% range. Envelope features improve classification accuracy for all models.

ensemble-model which uses the results of multiple random forests trained on different feature subsets [26, 58]. We used this extra-trees model as part of our drum generation system, the implementation process for this system and the results of our manual hearing tests are discussed in Chapter 6.

To create the confusion matrices and the F-Scores shown in Figure 5.14, the extra-trees model was trained on 80% of the database, and tested with the remaining 20%. Based on these reports, having multiple options for drum categorization does not noticeably influence the models accuracy in *Decision.1*. The DvDvN model's slightly smaller false negative rate for synthetic noise (11 vs 13 false negatives) is countered by a slightly higher rate of categorizing drums as synthetic noise (12 vs 9). Going forward, the DvDvN implementation is used as it makes the DvN model redundant. The finalized MEM used in the subsequent chapters is created by combining the autoencoder model with the ExtraTrees DvDvN classifier.

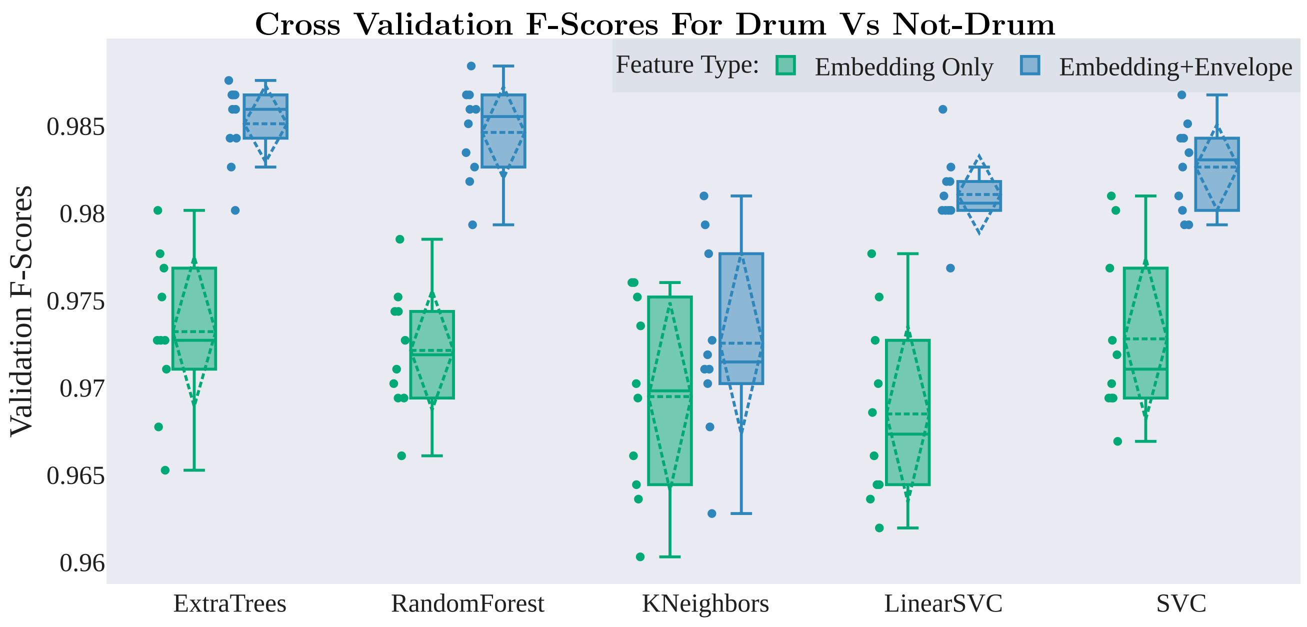
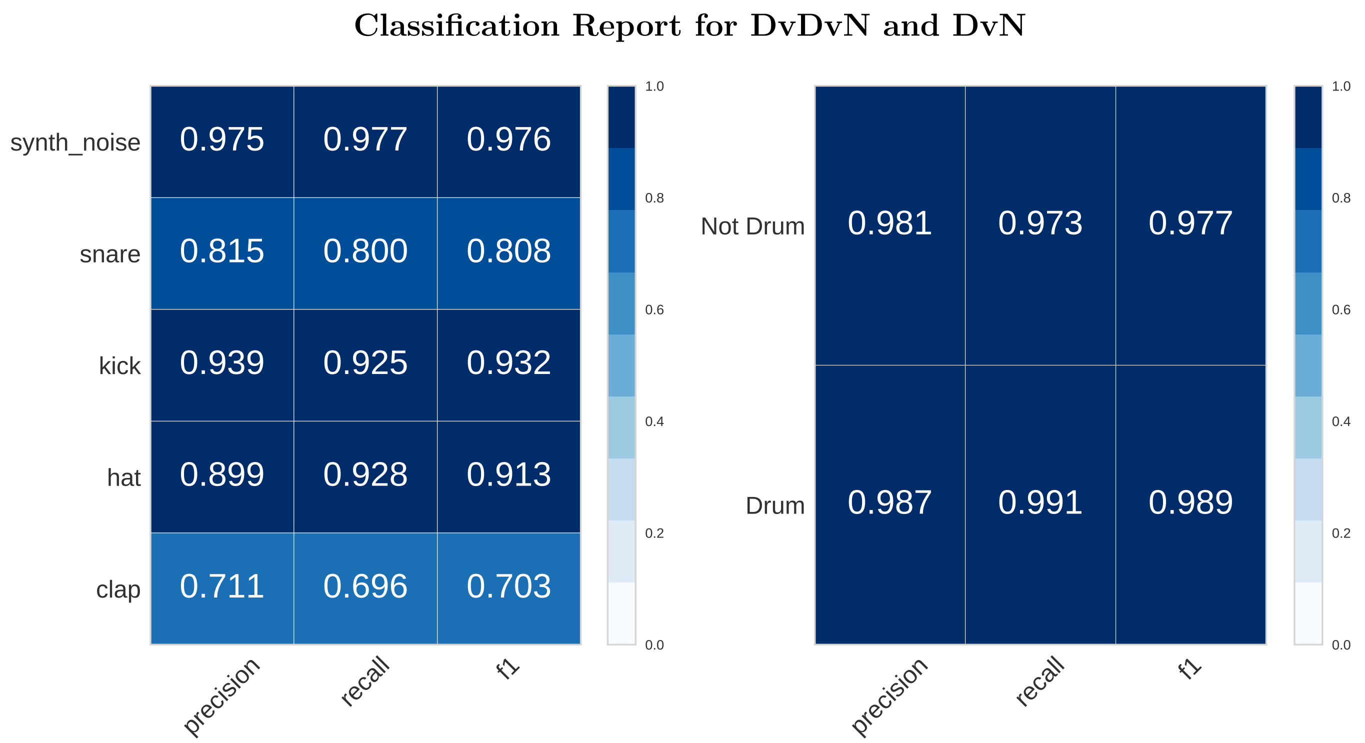


Figure 5.13: F-Score results for each cross-validation. Models perform better as there are less categorization groups. Envelope features increase accuracy for all models. Random Forest and Extra Trees remain the top two models.

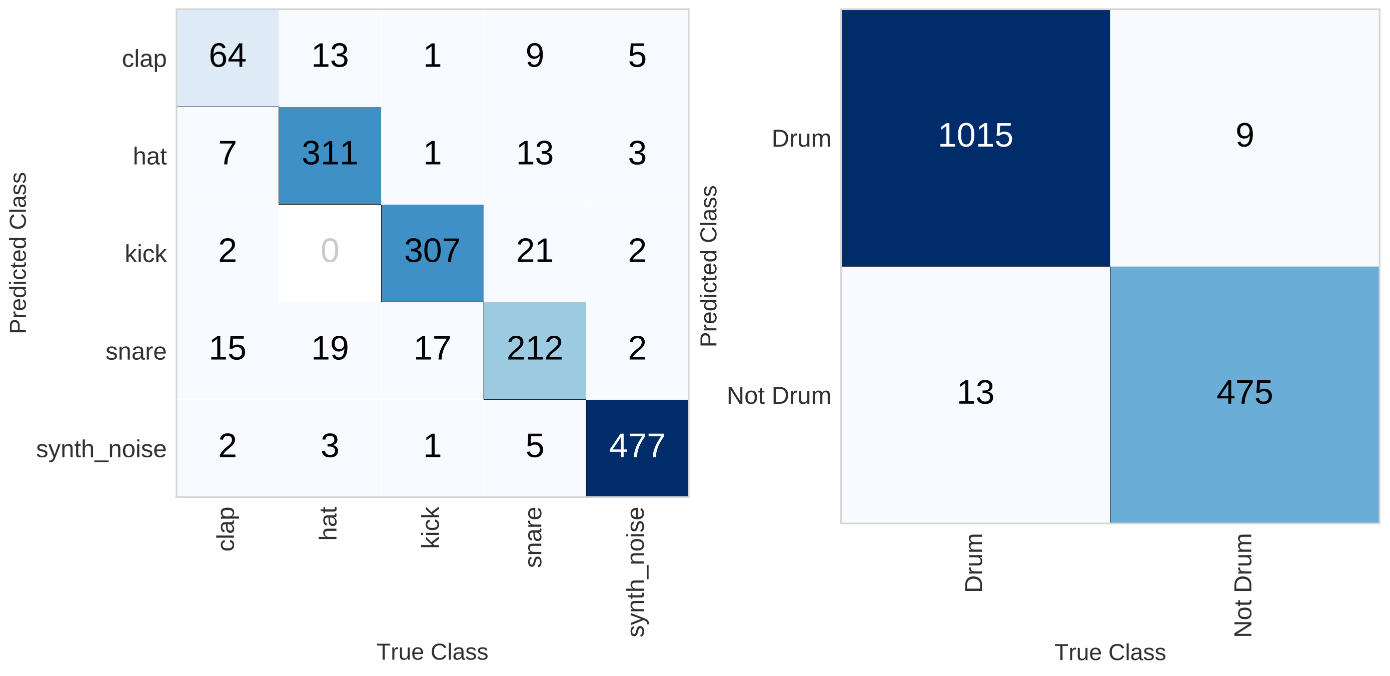
## 5.5 How Will TPEs and MEMs Be Used?

With the virtual synthesizer in place, we required a virtual ear: a tool capable of automatic separation of drums from non-drums and categorization of drum sounds. We took two different approaches to implementing such a tool, which gave us two types of virtual ears: TPE Ears, which make the two decisions sequentially, and MEM ears which makes these decisions simultaneously. The distinguishing characteristic of TPEs and MEMs is the type of feature set that they use for learning. TPEs use envelope and spectrogram features derived using Fourier transformations, while MEMs use encoded spectrogram features, which is a much smaller set of features, yet achieve similar classification performance.

Based on results shown in this chapter, TPEs and MEMs appear to have high categorization accuracy in our testing data. But without manual hearing tests, we cannot know whether the synthesizer sounds that are deemed percussive by the virtual ears resemble drum sounds or simply have characteristics that the virtual ears were not familiar with, leading to mistakes. This is due to



(a) Precision, recall, F1-Score, and number of supporting examples. The sounds in the “synth noise” category are identical to the sounds in the “not drum” category.



(b) Confusion matrices

Figure 5.14: F-Scores and confusion matrix of ExtraTrees model for both DvDvN and DvN categorization.

the infinite variety of sounds that are not percussive (i.e, the OSR problem), which makes such training tasks difficult.

In the next chapter we first discuss how TPEs and MEMs are used to implement generative systems capable of outputting percussive sounds and how the outputs of these systems were manually tested in order to measure their success in creation of drum sounds. We also discuss how this success hinges on the virtual ear's ability in separation drums from non-drums as well as accurate categorization of drums.

# Chapter 6

# Results

The proposed system for virtual drum generation proposed in Section 1.6 required the implementation of a virtual synthesizer and a virtual ear. The implementation and characteristics of these components were discussed in Chapters 4 and 5.1 respectively. In this chapter, we discuss how the components are used to create generative systems of drum sounds. The labeled sounds created by the system are recorded on disk, allowing us to measure the success of this system by blinded, manual categorization of the outputs and calculating the level of agreement between categorical labels assigned by us and the labels assigned by the various ear models.

## 6.1 Putting The System Together

The virtual synthesizer generates a second of audio based on randomly generated programs, and the virtual ear can analyze the generated sounds and categorizes them as percussive or non-percussive. These two components work together to execute the plan laid out for the final, generative system of drum sounds, shown in Figure 4.1. A code example of how the system is put together is shown in Listing 6.1.

This loop runs in a single CPU process, therefore generation can scale-up easily. Here, sounds are rapidly generated and sent to the virtual ear, which then separates drums from non-drums. The sounds labeled as drums are further analyzed and assigned a drum type. Finally, the accepted sounds are saved to hard-disk for use in compositions. In this chapter, we cover the

Listing 6.1: Example Program For The Generative System

```
import ear
import synthesizer
# randomly programming the synthesizer 1000 times
while(i<1000):
    i+=1
    # stacksize is randomly selected from a list
    stack_size = random([1,2,4])
    # make synthesizer, randomly program it, generate sound
    sound = synthesizer.random_sound(stack = stack_size)
    # Check if ear determines sound to be percussive
    is_drum = ear.is_drum(sound)
    if is_drum:
        # give sound a drum category
        drum_category = ear.drum_type(sound)
        # save sound to hard_disk
        save_sound("path/to/drum_category/",sound)
```

results of our blinded, manual hearing tests conducted on the sounds generated by these two systems.

## 6.2 How and Why Surveys are Conducted

The main function of the manual hearing tests is to measure the precision of the virtual ear when the model is performing open-set recognition, i.e, separating drums sounds from none-drum sounds. If a small subset of synthetic noise can stand in for percussive sounds, then the ideal synthetic ear will be able to separate the desired sounds from noise with high recall and precision (see Figure 5.1). Low recall will cause a slow-down of the pipeline, as a larger number of random sounds need to be generated and evaluated. Low recall is also likely to increase the loss of novel sounds. Low precision will make manual clean-up of generated samples necessary and arduous.

To assess the performance of each pipeline, we randomly create drum programs and generate the corresponding audio signal. This sound is then fed through the ear model to determine if the sound is percussive. If so, a category is assigned to the sound. This sound and the corresponding synthesizer program are saved to hard-disk. Here we assess the success rate of two different pipelines by manual inspection and categorization of a randomly selected subset of its results. We cross reference the manual categorizations with the categories assigned by the virtual ear and seek to answer the following two research questions for each system:

- **RQ.1**: Do we agree that the sounds are percussive?
- **RQ.2**: Do we agree with the drum category assigned to the percussive sounds?

## 6.3 Survey of Two-Phased Ear Performance

### 6.3.1 Methodology and Dataset

To answer **RQ.1** and **RQ.2** for the TPE approach, we put a generative system together using the TPEs discussed in Section 5.4, and ran the system until a large number of samples in each category were found and saved to disk. Next, we randomly picked around 50 samples in the following categories: "snare", "kick", "hat", "clap" and "other" (combination of rims, shakers and unusual percussive sounds). This gave us a total of 257 samples. These samples were determined to be percussive and then categorized by 3 different drum vs drum models (FC, CNNLSTM, E+F). We ensured a balanced division between samples of stack size 1, 2 and 4 (each stack is responsible for a third of the samples under each category).

### 6.3.2 Survey Application and Results

Amir Salimi and Abram Hindle categorized these samples without knowledge of each other's results, or drum categorizations assigned by the virtual ears. It's important to note that each responder had an additional category of "Bad" for samples that they deemed not percussive. The "Bad" category indicates that the sample should have not been accepted as percussive.

We assess the reliability of agreement between persons and categorization models via the Fleiss' kappa coefficient [22]. The value of 0 or less for this coefficient indicates no agreement beyond random chance, and the value of 1 indicates perfect agreement. Our kappa measurements shown in Table 6.1 lie

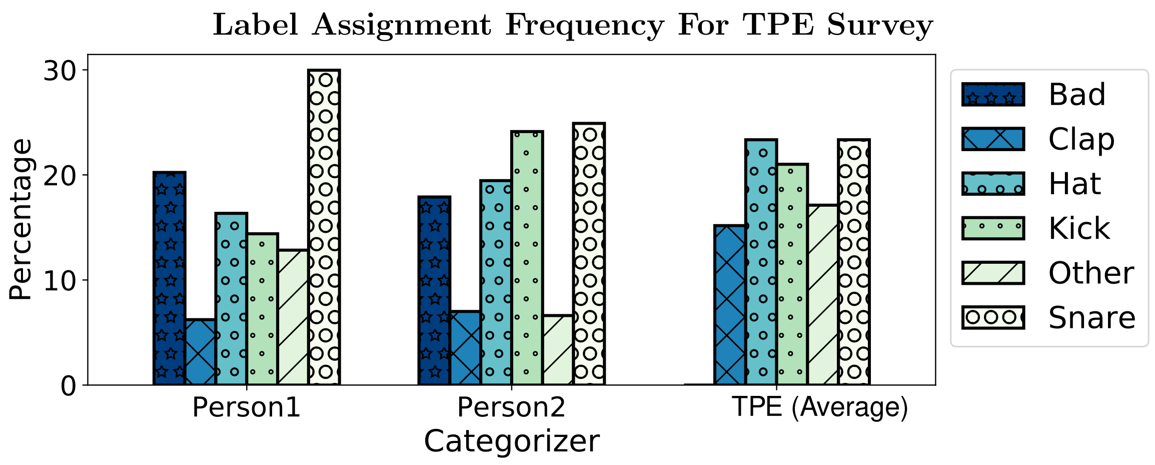


Figure 6.1: Frequency of assigned labels by persons versus the true number of labels (for TPEs)

| Drop Rule | Size | H+H | H+FC | H+CNN | H+E/F | 3 models |
|---|---|---|---|---|---|---|
| No Drops | 257 | 0.37 | 0.35 | 0.36 | 0.36 | 0.28 |
| Assigned "Bad" By Both | 236 | 0.31 | 0.37 | 0.37 | 0.38 | 0.30 |
| Assigned "Bad" By Either | 180 | 0.47 | 0.50 | 0.48 | 0.48 | 0.34 |
| Assigned "Bad" or "Other" By Either | 154 | 0.47 | 0.59 | 0.54 | 0.50 | 0.35 |

Table 6.1: Table of Fleiss' kappa coefficients to measure the degree of agreement between persons (HvH) and various TPEs: persons with FC model (H+FC), persons with CNNLSTM model, persons with all models (H+E/F), and between the 3 models. "Drop Rule" column indicates if any samples were dropped. We show the measurements after dropping samples if they are deemed bad by either or both responders. We also show measurements after dropping the "other" category along with samples deemed bad by either responder.

within the 0.35-0.45 range, indicating mild to moderate agreement between persons and machines. We again measure this coefficient after dropping samples that were categorized as "Bad" by the authors, as samples that persons deem to be "Bad" should not have been categorized by the models at all. Dropping of samples that both authors deemed "Bad" causes an 8% reduction of our data (21 samples) and a small increase in kappa score. Dropping samples deemed "Bad" by either reviewer resulted in a 30% reduction of samples and relatively large increase in kappa scores.

### 6.3.3 Takeaways Of The TPE Pipeline Survey

Our main takeaways from the TPE survey are as follows:

- For **RQ.1**, the survey brings into question the reliability of our DVN models, as 30% of the generated samples were deemed not percussive by at least 1 reviewer and 8% by both reviewers.
- For **RQ.2**, the task of categorizing synthetic drums appears difficult. Survey shows that even after removal of "Bad" samples, the scale of agreement between just 2 persons (H+H) as well as between 2 persons and any of the models (H+FC, H+CNN, H+E/F) is moderate at best; while the same models can achieve over 98 percent accuracy when tested on recorded drum samples. This may be a manifestation of the "open set recognition" problem.
- While there is much room for improvement, our pipeline can generate and categorize drums and percussive sounds with a promising degree of success.

## 6.4 Survey of Mixed Ear Model Performance

### 6.4.1 Methodology and Dataset

To answer **RQ.1** and **RQ.2** for the MEM approach, we kept the other components of the pipeline the same, while using the MEM as the virtual ear. With TPEs, the "other" category is assigned when the sound is determined to be percussive in the first phase, but the second phase determines that the sound does not belong to any of the drum categories it is familiar with. MEMs either assign the sound to a drum category or determine that it is not a drum, therefore the MEM survey differs from the TPE survey in that the pipeline only creates the four drum categories, yet the "other" category can still be assigned by survey responders. This means that two out of six possible categories are only available to responders, which can cause an inherent bias towards worse agreement scores compared to the last survey.

| Drop Rule | Size | HvH | H+MEM |
|---|---|---|---|
| No Drop | 300 | 0.336 | 0.250 |
| Assigned Bad By Both | 249 | 0.200 | 0.260 |
| Assigned Bad By Either | 151 | 0.460 | 0.473 |
| Assigned “Bad” or “Other” By Either | 120 | 0.620 | 0.587 |

Table 6.2: Table of Fleiss’ kappa coefficient to measure the degree of agreement between persons (HvH) and persons and MEM. We measure the agreeability scores after dropping bad samples if both or either persons assigned the sample as such. We also measure agreeability when all samples deemed “Bad” or “other” by either person are removed.

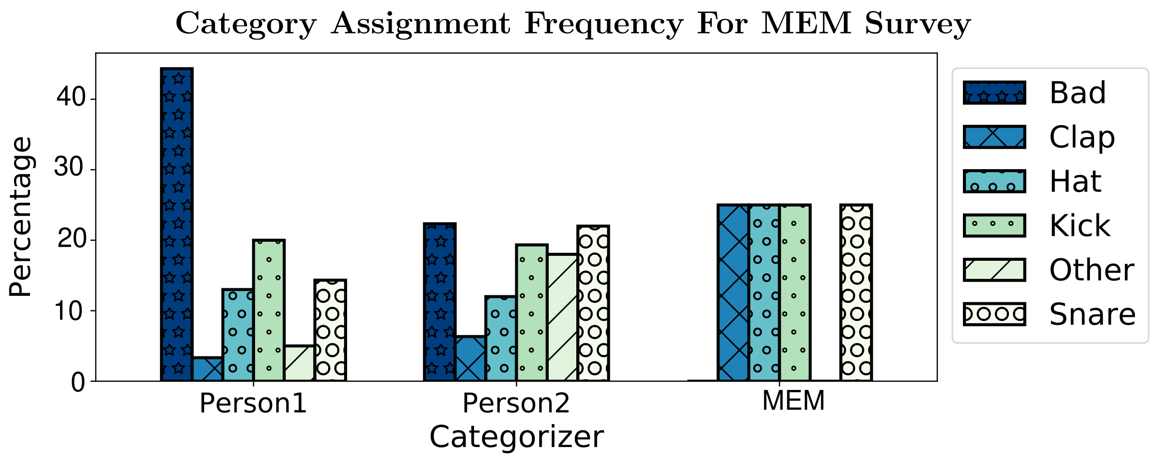


Figure 6.2: Frequency of assigned labels by persons versus the true number of labels (for MEMs)

### 6.4.2 Survey Application and Results

As shown in Table 6.2, before dropping the “bad” outputs, there is mild to moderate agreeability. By dropping “bad” samples (therefore accounting for failures in *Decision.1*), a dramatic increase in agreeability score is seen in the results. This highlights a the MEM’s weakness in the separation of drums from non-drums.

### 6.4.3 Takeaways Of The MEM Pipeline Survey

Our main takeaways from the MEM survey are as follows:

- For **RQ.1**, 50% of sounds were deemed “bad” by at least one survey taker, and 17% by both.

- For **RQ.2**, the improvement in kappa scores after reduction of “bad” outputs show that similar to TPEs, mistakes in separation of drums in non-drums is a major point of failure.

- Overall, these results show success in usage of spectrogram autoencoders as feature extractors for sounds, however we suspect that open set recognition is a major weakness.

## 6.5 Survey Conclusion

The results highlighted here are encouraging, as 50% or more of the outputs from either pipeline is deemed percussive by both survey takers, and moderate to high agreement between human and virtual listeners is observed for both *Decision.1* and *Decision.2*. A comparison of the survey results helps identify common weaknesses and avenues for improvement in the future. By using agreeability scores as the performance measure, we compare the two models with regards to separation of drums from non-drums and categorization of drum types:

- **Decision 1:** The sounds generated were deemed percussive by both subjects in 70% of the cases with the TPE system, and 50% with the MEM. This discrepancy is highlighted in the agreement scores as well; The agreement scores in Tables 6.1 and 6.2 show the scores before and after failures in *Decision.1* were removed according to various rules. Here, we see a sharper increase in the agreeability scores for the MEM as more failures in drum vs non-drum separation are removed. This further highlights that *Decision.1* is a more pronounced weakness for the MEM.

- **Decision 2:** When all “bad” and “other” sounds are removed, the models can be fairly compared for their performance in *Decision.2*. Here, the MEM matches the performance of FC, the best drum categorizing TPE. This suggests that drum classification using spectrogram embeddings can match or exceed the performance of models which learned from the entire spectrogram. This is interesting as the autoencoder model used for feature extraction was trained for encoding spectrograms into a small, 8-dimensional feature space and recreating the original using said features, not feature extraction for drum classification. Yet this feature space proves useful in drum-classification, although not in drum vs non-drum classification.

Overall, the OSR problem (as described in Section 5.2.1) is a major weakness in this project, particularly in the MEM approach. When removing all failures in *Decision.1*, the agreeability scores of the MEM and the best TPE are comparable and within a satisfactory ”moderate to strong agreement” kappa range for *Decision.2*. However, despite these failures, the approach taken in this work has enabled the generation of virtually synthesized drum sounds in an unsupervised manner, achieving the original goal of the project outlined in the thesis statement in Section 1.6.

# Chapter 7

# Drum-Kit Mutation

## 7.1 Suitability for Live Performance

We originally set out to generate new drum samples for the purpose of creating new drum-kits which musicians could use to create drum tracks for their compositions. One use case for the drum generation approach pursued in this work is to create and evolve drum-kits on the fly. In order to test the suitability of the generative system for both drum-kit generation and live performances, we designed a framework which utilizes the generative system to create a "live performance" framework as outlined in Figure 7.1. This framework continually evolves the sounds within a drum-pattern (or rhythm) by creating and mutating a drum-kit using the outputs of the best TPE system. This framework could be particularly useful in a setting where drum rhythms are repeated for long durations. In order to test this approach, we created a basic music notation program where each drum track is defined by the following attributes:

- Beats Per Minute (BPM): This defines the "speed" of the pattern.
- Number of Beats: How many beats are in the pattern.
- Drum Pattern: An action must be taken at each beat, either a drum is played or no action is taken and nothing is played for that beat. The drum pattern dictates what action is taken at each beat. This makes the drum patterns monophonic (at most one sound can be played at a time).

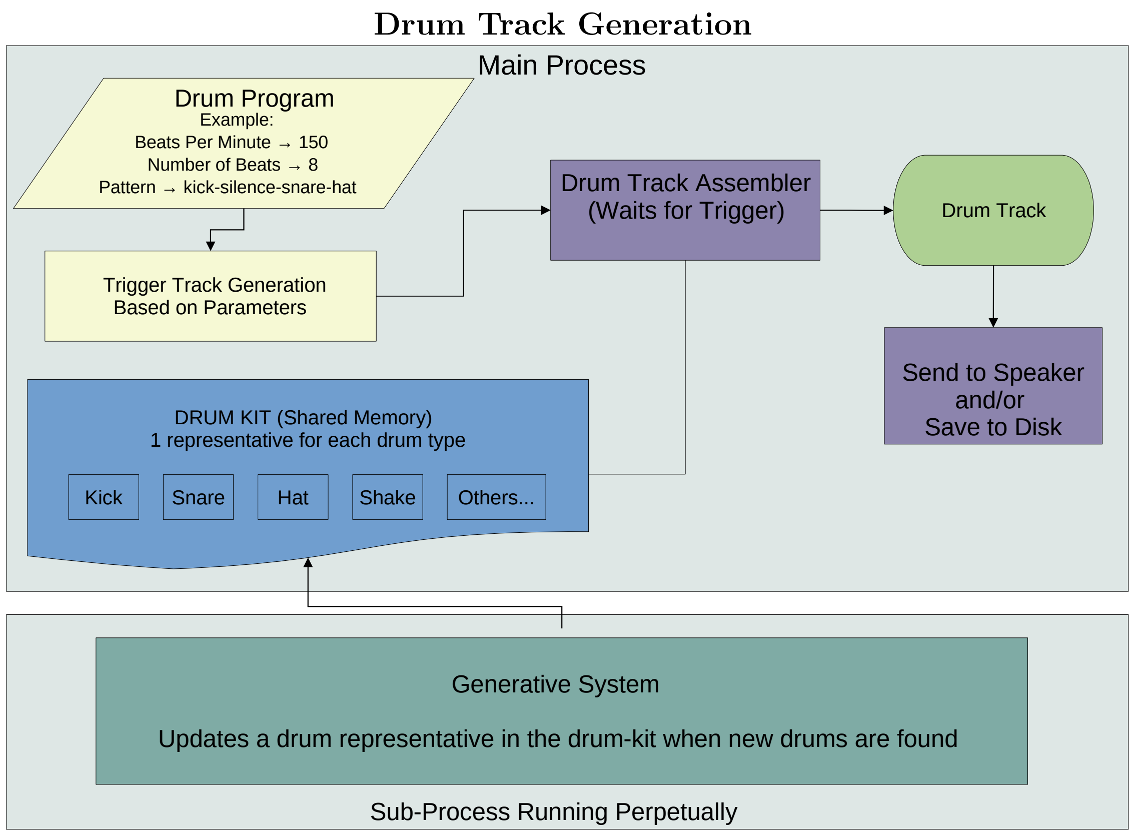


Figure 7.1: Live drum programming framework. Given a pre-defined drum program, the main process creates drum tracks using a drum-kit. This drum-kit is continually modified by a generative system running in a background process. In a live setting, the drum track assembler can be triggered at set times depending on the program, in offline generation, it can be triggered whenever there is a modification to the drum-kit.

| Program | BPM | Beats | Action at Each Beat |
|---|---|---|---|
| Loop 1 | 180 | 8 | Kick-Snare-Hat-Kick-Snare-Hat-Kick-Shake |
| Loop 2 | 125 | 16 | Kick-.-Kick-.-Snare-.-Hat-Shake-Hat-Shake-Kick-.-Snare-.-. |

Table 7.1: Parameters for each drum program. A period (".") indicates no action

This experiment starts with a drum program and a drum-kit which contains 1 representative for the hat, kick, snare and shaker percussion categories. Next, a process begins to perpetually listen for new outputs from the TPE system. Each time a new drum sound is created, the process replaces a sound in the drum-kit depending on the drum category; next, the drum track is regenerated using the new drum-kit. With this approach, if the outputs are listened to sequentially or in a live setting, the same rhythm is repeated, but the drums within the rhythm mutate over time. To analyse this approach, we defined two simple drum programs (shown in Table 7.1), and let the experiment run until 1000 drum tracks are generated, that is, 1000 new drum sounds are found and used for the creation of a drum track. The results for the rendition of the programs are available for download on Zenodo[1].

## 7.2 Analyzing the Outputs

We conduct a listening test in order to quantify the usefulness of this framework. For this test, Amir Salimi and Abram Hindle listened to 100 drum tracks for each drum program and recorded the number of drum tracks they personally found appropriate for their live performance, that is, the rendered track accurately represented the underlying drum program. The result of this survey is shown in Table 7.2. Due to the subjective nature of this analysis, readers are encouraged to download and analyze the results as well. We consider the majority of the drum tracks output by this system to be successful renditions. As discussed previously in Section 6.5, as many as 30% of the drum sounds output by the TPE system did not resemble drums when played in isolation. However, within a larger rhythmic pattern, the effect of these

[1]https://zenodo.org/record/5202776

| Program | Listener 1 Likes/100 | Listener 2 Likes/100 | Average Percentage |
|---|---|---|---|
| Loop 1 | 73/100 | 54/100 | 68% |
| Loop 1 | 55/100 | 55/100 | 55% |

Table 7.2: Measuring the quality of generated drum tracks by calculating the percentage of liked outputs for each listener.

failures is diminished and can at times introduce an interesting dimension to the tracks. We also note that due the the source of drum sounds that are learned from and imitated in this work, generation of drum tracks may be more suitable for experimental music performances.

We monitored the drum discovery rate while rendering the drum programs, and found that the relative ratio of drum to non-drum generations was approximately 1/100. This means that 100 sounds need to be created before 1 modification is made to the drum-kit. Using 1 CPU thread[2], these 100 iterations take approximately 7 seconds. This discovery rate can be increased or decreased in a number of ways. For example, by increasing the number of discovery threads at the cost of heavier load to the CPU. It is also possible to make trade-offs in precision and sensitivity, that is, accepting more sounds that may not fit their role well within the composition and decreasing discovery time, or rejecting more unsuitable sounds and increasing discovery time.

## 7.2.1 Noticeable Limitations

These drum tracks highlight other issues and limitations of this work. Other than improving the virtual ears, some of the issues and possible solutions are as follows:

- Discovery time: Stricter, more powerful drum versus non-drum classifiers may reject nearly all synthesized outputs, making the creation of new drum-kits time consuming if not impossible. This means that the introduction of an algorithm which intelligently modifies parameters in order to successfully make drums will be a necessary step in the future

[2]AMD Ryzen 7 2700X Eight-Core Processor

in order to create desired sounds more efficiently.

- Limited variety: The virtual synthesizer used in this work has a limited set of parameters. This causes a lack of variety in the sounds output by the system. Expansion of the virtual synthesizer parameters or an interface which allows the use of VST synthesizers could help with this problem.

## 7.3 Conclusion

The goal here was to create a tool which can create and mutate drums used within drum-tracks for live performances. In an offline setting, this tool should be able to quickly generate new drum-kits for musicians. To this end, we implemented a simple framework which creates drum tracks given a drum pattern.

This framework uses the generative system of drums described in the previous chapters to gradually replace the drum sounds within the drum pattern (i.e., the underlying drum-kit). We analyzed the outputs of this framework and believe that many of the outputs could be useful for musicians.

This experiment shows that the approach taken in this work towards a generative system of drum sounds can have many creative applications. While some of the weaknesses and failures of the classification models may be less pronounced in a live performance, this experiment also highlights other possible avenues of improvement and expansion which we hope to address in future works.

# Chapter 8

# Conclusions, Validity, and Future Work

## 8.1 Conclusions

As proposed in our thesis statement in Section 1.6, the goal of this project was to create an automatic system in which a virtual ear discovers new drum sounds from randomly programmed virtual synthesizers. We built and tested two generative systems for the creation of multiple types of drum sounds using automatic programming of virtual synthesizers. We verified its results with human listeners, and found that of the sounds generated were deemed percussive by both subjects in 70% of the cases with the TPE system, and 50% with the MEM. While many avenues of improvement are available, the implementation outlined here satisfies our stated goals. Our work enables not only the creation of new libraries of percussion sounds, but new synthesizer programs which can be modified and studied. Manual listening tests revealed much room for improvement, particularly with accurate separation of percussive sounds from the infinitely large set of non-percussive sounds. We demonstrated some success in our utilization of latent representations of autoencoder networks as exponentially smaller representations of audio spectrograms, yet these results did not yield any noticeable advantage over direct usage of non-encoded features.

## 8.2 Threats to Validity

### 8.2.1 Construct Validity

We did not measure, nor can we guarantee, the novelty of the generated drum sounds. Let's consider a system which outputs the exact same percussive sound in every iteration. This system would have achieved a perfect kappa coefficient score with 0 percent non-percussive outputs, despite having no utility in music production.

### 8.2.2 Internal Validity

The lack of consistency in training and accuracy measurements makes comparisons between TPEs and MEMs difficult, therefore we cannot confidently say which model performed better overall. The models have been trained on different subsets of the 3 different drum datasets, and cross validation was utilized in measuring the MEMs but not the TPEs. The design of the virtual synthesizer—its parameters and parameter values—is loosely based on commonly found parameters in VST synthesizers, therefore, we cannot guarantee that these results will translate to other virtual synthesizers. We also did not measure what percentage of the randomly generated sounds are percussive before being filtered by the virtual ear.

### 8.2.3 External Validity

In addition, since both survey subjects have a preference for "experimental" electronic music, which may influence their perception of what is and is not an acceptable drum sound. Furthermore, both survey subjects played a role in the implementation of the project, which may influence their leniency in output sounds as percussive or non-percussive.

## 8.3 Future Work

Effective implementation of models that can learn with a small number of examples is a priority as it will allow for a larger variety of sounds to be generated, and perhaps address the OSR problem. Program generation can be improved by reinforcement learning or other heuristics, however, we need to ensure that there is a stronger agreement between the synthetic ear's scores and human listeners. We will also explore more specialized autoencoder architectures and training methods in order to improve feature extraction.

# References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631, 2019.

[2] Claude Alain and Kelly L McDonald. Age-related differences in neuromagnetic brain activity underlying concurrent sound perception. *Journal of Neuroscience*, 27(6):1308–1314, 2007.

[3] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. Ieee, 2017.

[4] James A Anderson, Edward Rosenfeld, and Andras Pellionisz. *Neurocomputing*, volume 2. MIT press, 1988.

[5] Cyran Aouameur, Philippe Esling, and Gaëtan Hadjeres. Neural drum machine: An interactive system for real-time synthesis of drum sounds. *arXiv preprint arXiv:1907.02637*, 2019.

[6] Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. In *Advances in neural information processing systems*, pages 3084–3092, 2013.

[7] Dan Barry, Derry Fitzgerald, Eugene Coyle, and Bob Lawlor. Drum source separation using percussive feature detection and spectral modulation. 2005.

[8] Imad A Basheer and Maha Hajmeer. Artificial neural networks: fundamentals, computing, design, and application. *Journal of microbiological methods*, 43(1):3–31, 2000.

[9] Yoshua Bengio. Gradient-based optimization of hyperparameters. *Neural computation*, 12(8):1889–1900, 2000.

[10] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(Feb):281–305, 2012.

[11] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

[12] Margaret A Boden. Computer models of creativity. *AI Magazine*, 30(3):23–23, 2009.

[13] Pierre Cardaliaguet and Guillaume Euvrard. Approximation of a function and its derivative with a neural network. *Neural Networks*, 5(2):207–220, 1992.

[14] Nitesh V Chawla, Nathalie Japkowicz, and Aleksander Kotcz. Special issue on learning from imbalanced data sets. *ACM SIGKDD explorations newsletter*, 6(1):1–6, 2004.

[15] William T Cochran, James W Cooley, David L Favin, Howard D Helms, Reginald A Kaenel, William W Lang, George C Maling, David E Nelson, Charles M Rader, and Peter D Welch. What is the fast fourier transform? *Proceedings of the IEEE*, 55(10):1664–1674, 1967.

[16] Karen Collins. In the loop: Creativity and constraint in 8-bit video game audio. *Twentieth-century music*, 4(2):209, 2007.

[17] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[18] Palle Dahlstedt. Creating and exploring huge parameter spaces: Interactive evolution as a tool for sound generation. In *ICMC*, 2001.

[19] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. Neural audio synthesis of musical notes with wavenet autoencoders, 2017.

[20] Philippe Esling, Axel Chemla-Romeu-Santos, and Adrien Bitton. Generative timbre spaces with variational audio synthesis. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, 2018.

[21] Philippe Esling, Naotake Masuda, Adrien Bardet, Romeo Despres, et al. Universal audio synthesizer control with normalizing flows. *arXiv preprint arXiv:1907.00971*, 2019.

[22] Joseph L Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378, 1971.

[23] Harvey Fletcher and Wilden A Munson. Loudness, its definition, measurement and calculation. *Bell System Technical Journal*, 12(4):377–430, 1933.

[24] Chuanxing Geng, Sheng-jun Huang, and Songcan Chen. Recent advances in open set recognition: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[25] André Gensler, Janosch Henze, Bernhard Sick, and Nils Raabe. Deep learning for solar power forecasting—an approach using autoencoder and lstm neural networks. In *2016 IEEE international conference on systems, man, and cybernetics (SMC)*, pages 002858–002865. IEEE, 2016.

[26] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[27] Daniel Griffin and Jae Lim. Signal estimation from modified short-time fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(2):236–243, 1984.

[28] Mehadi Hassen and Philip K Chan. Learning a neural-network-based representation for open set recognition. In *Proceedings of the 2020 SIAM International Conference on Data Mining*, pages 154–162. SIAM, 2020.

[29] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[30] Geoffrey E Hinton and Richard S Zemel. Autoencoders, minimum description length and helmholtz free energy. In *Advances in neural information processing systems*, pages 3–10, 1994.

[31] Frank Hoffmann. Drum machine. In *Encyclopedia of Recorded Sound*, pages 678–678. Routledge, 2004.

[32] Andrew Horner, James Beauchamp, and Lippold Haken. Machine Tongues XVI: Genetic algorithms and their application to FM matching synthesis. *Computer Music Journal*, 17(4):17–29, 1993.

[33] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. An efficient approach for assessing hyperparameter importance. In *International conference on machine learning*, pages 754–762, 2014.

[34] Muhammad Huzaifah. Comparison of time-frequency representations for environmental sound classification using convolutional neural networks. *arXiv preprint arXiv:1706.07156*, 2017.

[35] Colin G Johnson. Exploring the sound-space of synthesis algorithms using interactive genetic algorithms. In *Proceedings of the AISB'99 Symposium on Musical Creativity*, pages 20–27. Society for the Study of Artificial Intelligence and Simulation of Behaviour, 1999.

[36] Eric Jones, Travis Oliphant, Pearu Peterson, et al. Scipy: Open source scientific tools for python. 2001.

[37] Stephen Lakatos. A common perceptual space for harmonic and percussive timbres. *Perception & psychophysics*, 62(7):1426–1439, 2000.

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

[39] Colby N Leider. *Digital audio workstation.* McGraw-Hill, Inc., 2004.

[40] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[41] Richard G Lyons. *Understanding digital signal processing, 3/E.* Pearson Education India, 2004.

[42] Chris A Mack. Fifty years of moore's law. *IEEE Transactions on semiconductor manufacturing*, 24(2):202–207, 2011.

[43] Matthieu Macret, Philippe Pasquier, and Tamara Smyth. Automatic calibration of modified fm synthesis to harmonic sounds using genetic algorithms. In *Proceedings of the 9th Sound and Music Computing Conference*, Copenhagen, Denmark, 2012.

[44] Robert G Malkin. Machine listening for context-aware computing. *PHD thesis, CMU*, 2006.

[45] Max Mathews. Music, cognition, and computerized sound: An introduction to psychoacoustics. chapter 6: What is Loudness?, page 71. The MIT press, 1999.

[46] Max V Mathews. The digital computer as a musical instrument. *Science*, 142(3592):553–557, 1963.

[47] David Mellor. Hands on: Roland tr808 drum machine (sos feb 1993). *Sound On Sound*, (Feb 1993):42–46, 1993.

[48] Daniel Mitchell. Basicsynth. chapter 6: Envelope Generators. Lulu. com, 2009.

[49] Daniel Mitchell. Basicsynth. chapter 1: Synthesis Overview. Lulu. com, 2009.

[50] Robert A Moog. Midi: musical instrument digital interface. *Journal of the Audio Engineering Society*, 34(5):394–404, 1986.

[51] Martin Mundt, Iuliia Pliushch, Sagnik Majumder, and Visvanathan Ramesh. Open set recognition through deep neural network uncertainty: Does out-of-distribution detection require generative classifiers? In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[52] S Nawab, T Quatieri, and Jae Lim. Signal reconstruction from short-time fourier transform magnitude. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 31(4):986–998, 1983.

[53] Rodney Needham. Percussion and transition. *Man*, 2(4):606–614, 1967.

[54] Dina L Newman, Laurel M Fisher, Jeffrey Ohmen, Robert Parody, Chin-To Fong, Susan T Frisina, Frances Mapes, David A Eddins, D Robert Frisina, Robert D Frisina, et al. Grm7 variants associated with age-related hearing loss based on auditory perception. *Hearing research*, 294(1-2):125–132, 2012.

[55] Stephen Notley and Malik Magdon-Ismail. Examining the use of neural networks for feature extraction: A comparative analysis using deep learning, support vector machines, and k-nearest neighbor classifiers. *arXiv preprint arXiv:1805.02294*, 2018.

[56] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[57] Aaron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George van den Driessche, Edward Lockhart, Luis C Cobo, Florian Stimberg, et al. Parallel wavenet: Fast high-fidelity speech synthesis. *arXiv preprint arXiv:1711.10433*, 2017.

[58] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[59] John Pierce. Music, cognition, and computerized sound: An introduction to psychoacoustics. chapter 4: Sound Waves and Sine Waves, page 38. The MIT press, 1999.

[60] Foster Provost. Machine learning from imbalanced data sets 101. In *Proceedings of the AAAI'2000 workshop on imbalanced data sets*, volume 68, pages 1–3. AAAI Press, 2000.

[61] Yunchen Pu, Zhe Gan, Ricardo Henao, Xin Yuan, Chunyuan Li, Andrew Stevens, and Lawrence Carin. Variational autoencoder for deep learning of images, labels and captions. In *Advances in neural information processing systems*, pages 2352–2360, 2016.

[62] António Ramires, Pritish Chandna, Xavier Favory, Emilia Gómez, and Xavier Serra. Neural percussive synthesis parameterised by high-level timbral features. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 786–790. IEEE, 2020.

[63] Joshua D Reiss. A meta-analysis of high resolution audio perceptual evaluation. *Journal of the Audio Engineering Society*, 2016.

[64] Robert Rowe. *Interactive music systems: machine listening and composing.* MIT press, 1992.

[65] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

[66] Amir Salimi and Abram Hindle. Drum and percussion kits, 10.5281/zenodo.3994999, Aug 2020.

[67] Amir Salimi and Abram Hindle. Make your own audience: virtual listeners can filter generated drum programs. In *Proceedings of the 2020 AI Music Creativity Conference, 2020*, pages 1–8, 2020.

[68] Diemo Schwarz. Corpus-based concatenative synthesis. *IEEE signal processing magazine*, 24(2):92–104, 2007.

[69] Julius O Smith. Viewpoints on the history of digital synthesis. In *Proceedings of the International Computer Music Conference*, pages 1–1. International Computer Music Association, 1991.

[70] Stanley S Stevens and John Volkmann. The relation of pitch to frequency: A revised scale. *The American Journal of Psychology*, 53(3):329–353, 1940.

[71] George Tanev and Adrijan Božinovski. Virtual studio technology inside music production. In *International Conference on ICT Innovations*, pages 231–241. Springer, 2013.

[72] Ryuichi Yamamoto, Eunwoo Song, and Jae-Min Kim. Parallel wavegan: A fast waveform generation model based on generative adversarial networks with multi-resolution spectrogram. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6199–6203. IEEE, 2020.

[73] Matthew John Yee-King, Leon Fedden, and Mark d'Inverno. Automatic programming of vst sound synthesizers using deep networks and other techniques. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(2):150–159, 2018.

[74] Zixing Zhang, Fabien Ringeval, Jing Han, Jun Deng, Erik Marchi, and Björn Schuller. Facing realism in spontaneous emotion recognition from speech: Feature enhancement by autoencoder with lstm neural networks. 2016.

[75] Eberhard Zwicker. Procedure for calculating loudness of temporally variable sounds. *The Journal of the Acoustical Society of America*, 62(3):675–682, 1977.