You need a lot more prior work.
You need to explain a lot more.
You need to consider what people don't know.
THe motivation needs to be more clear and verbose

Percussive Sound Generation with Virtual Listeners and Modular Synthesizers

Probably we have to evaluate the currated collection.

by

You should talk about instruments you can make with this.

Amir Salimi

Can you cluster the results so we can get a soundfont of an instrument?

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

Master of Science

Department of Computing Science

University of Alberta

© Amir Salimi, 2020

Abstract

Digital sound artists often require a variety of percussive samples for their music. We observed a lack of copyright-free one-shot percussive sample-packs for purposes of research and music creation. Yet for more than two decades research involving digital synthesis, genetic search, and neural networks has been used to invent and approximate novel sounds. Our goal is to generate one-shot percussive samples by leveraging modern AI technologies alongside scalable signal generation methods. Could we generate novel drum sounds through machine learning, heuristic search and modular synthesizers? We centered our approach around the combination of two central components: (i) a "virtual ear" capable of learning key features of various sound groups and evaluating the proximity of unheard sounds to desired sets and (ii) a dynamic virtual synthesizer with a rich set of tractable parameters. We present a generative pipeline that utilizes robust digital signal processing methods and is guided by supervised learning and genetic search towards generation of novel, high-quality, one-shot percussive sounds. We present our findings and measurements of the various approaches taken towards the implementation of the virtual ear, virtual synthesizer, and generative pipeline; hoping to expose the advantages and shortcomings of the employed methodologies and their practicality in future works via a clear demonstration of their capabilities relative to our stated goal. We also discuss and share our curated dataset of sounds along with our codebase [1] which can be used for its further expansion.

[1] `https://github.com/imilas/Synths_Stacks_Search`

Contents

Chapter 1

Introduction

Maybe provided a more long winded but more evidence based argument that this is what people do. Imagine Davood is on your committee, does he know this?

The rise of Digital Audio Workstations (DAW) [11] and Virtual Studio Technology (VST) based plug-ins [19] have rapidly transformed the sonic and material landscape of music production in the recent years. Coupled with this rise in popularity is a vast array of commercial products and services dedicated to satiating the need of amateur and professional music producers for unique sounds; most commonly via audio samples: one-shot drum samples, long sustained notes (commonly referred to as pads or textures), and loops (percussive or melodic) are common deliverables. Two notable examples of these commercial services are *loopmasters*[1] and *splice.com*[2]. Furthermore, VST plug-ins can emulate complex audio synthesizers and effects which some producers may find daunting or time consuming 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 producers to further modify these presets until their desired sound is reached.

We want to leverage AI technologies, such as heuristic search and random search, to produce new drum samples, by finding synthesizers and their appropriate configurations that can produce various kinds of drum samples. Effectively we will combine machine listening and code generation of synthesizers to find drum samples.

Our work is motivated by the idea of finding new, convenient methods

[1] loopmasters.com
[2] splice.com

what do you achieve

for the expansion of a music producer's library of sounds. Primarily with generation of novel, one-shot audio samples but also with automated search and creation of presets for virtual synthesizers. Using the generation of short, percussive audio samples as a starting point, this project is a proof of concept for a promising avenue towards our motivational goal.

1.1 Methodology

Our attempt is centered around a generative pipeline of audio. We want random sounds to be created by a tractable source, we also want to evaluate these sounds so we can guide the random sounds generator towards desired sounds. Instead of learning weights and parameters in an audio-generating neural network, we wish to generate, search, and tune synthesizers to produce percussion sounds. Following this idea, we found the proper implementation of 2 major components to be crucial:

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

Our components are designed with modularity and parallelizability in mind. This allows each component to be debugged, modified, and improved without requiring modifications in other components while additionally increasing the scalability and speed of experiments. Section 2 contains further discussion of the components as well as the code that glues the project together.

While the main focus of this project is the generation of novel percussive sounds, our methodology indicates promising results with regards to creation of new presets for any virtual synth without the need for a-priori knowledge of the functions or its parameters (i.e the effect of parameter modulation on the sonic output). We also demonstrate the viability of a virtual synthesizers based

on Digital Signal Processing (DSP) methods for fast, unsupervised creation of novel audio, discussed in more detail in section 1.2.

This is an entire chapter and you have to explain prior work. This is a big part

1.2 Related Work and Contradistinctions

Numerous 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 [3, 20, 14]. Even more recently Variational AutoEncoders (VAE's) have been utilized for generation of short percussive samples [1, 15]. In this work however, we opt to use digital signal processing methods to create a virtual synthesizer for the generation of audio signals as it provides several unique advantages:

i Fast, offline rendering of audio with no reliance on GPU: Currently not possible with state of the art models such as parallel WaveGan [20] and parallel WaveNet [14].

ii Rendering at high sampling rates: Performance speed being a common issue, the standard sampling rate in most audio generation work utilizing neural networks appears to be under 24 khz [20, 14, 1, 15]. However, a significant number of untrained human ears can detect a change in quality of audio between sampling rates of 192 khz and the industry standard of 44.1 khz [16] with a dramatic increase in quality detection after training. Therefore we can safely assume that most producers would prefer their audio samples to have sampling rates of 44.1 khz or higher. In this work, we fix our sampling rate to the 48 khz standard.

iii Neural networks are often viewed as unexplainable black box solutions. Some models such as VAE's can learn an underlying latent space of parameters and capture the "essence" of the different labels in a dataset. However, these spaces are learned in an unsupervised manner and must be manually analysed, perhaps extensively, before they can be understood [4]. The use of a virtual synthesizer for audio generation makes

our parameters readily understandable and easily modifiable.

Automatic programming of virtual synthesizers has also been a topic of interest. In early 2000s, Interactive Genetic Algorithms (IGA's) were utilized for the generation of new sounds with various sound-engines [10, 2]. More recent work by Yee-King et al. [21] 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. Since this work appears more focused on pads and textures rather than drums, 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, impressive 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 audio [5]. As stated before, our work explores the rapid approximation of percussion sounds with no previous knowledge about the sonic capabilities of our virtual synthesizer, exploring the actual parameter space rather than its latent representation.

1.3 Data And Project Replication

Our data is a large set of drum samples aggregated from personal libraries, free drum kits from the sample-swap project [3] which we further processed to suit our categories, and a large set of drum sounds aggregated from royalty free sources such as musicradar [4]. We will make our dataset of free-drum sounds available for download. The scripts used to download and process royalty free samples will also be made available. Further information about downloading our dataset can be found on this project's github page.

Our drum categories are claps, hats, kicks, rims/other, shakers, snares,

[3] https://sampleswap.org/
[4] https://www.musicradar.com/

mid-toms and high-toms. Other categories are chopped guitars, chopped pianos and n-stack-synths (random noise generated by the virtual synth with stack size of n, see 2.1), utilized for learning percussive vs non-percussive sounds. Stack sizes refers to how many synth functions are connected together in a synthesizer. Some notes about our dataset:

- Of the 6000 drum-sounds utilized in our work, the kick, snare and hat categories have the largest share at around 20% each, while the shaker and rim (other) categories have the smallest at 5% combined. Due to this we only focused on learning from kicks, snares, toms, claps and hats for Phase 1 of training (along with non-percussive groups of sound) 2.2.

- For Phase 2 of training we only focused on categorizing snares, claps, kicks, hats and other (percussive sounds such as shakers, rims and unusual percussions that we couldn't categorize were grouped into this category). Non-percussive datasets were not used for this phase.

- In order to offset bias from data imbalance during training of our models, the categorical cross entropy loss was weighted by the group sizes.

- For any given model, 80% of our data is used for training and 20% is used for testing.

- We limit the size of the n-stack-synths category to 50% of the total size of our drum dataset. This is done in order to measure whether the features extracted can address the "Open Set Recognition" problem, which will be discussed further in Section 2.2.

Diagram?

Chapter 2

Implementation

Give me an overview

2.1 Virtual Synthesizer

We used the python based Pippi library for sound generation[1]. This library uses a C back-end and focuses on fast offline generation of audio signals. In our implementation a synthesizer can have any number of sub-modules. The parameters that deterministically dictate the output signal of the sub-module as well as the range of values each parameter can take are shown in table 2.1. We call this number the *stack size*. For the range of parameters each sub-module can take we call the sets of parameters that characterize a synth's sub-modules a *program* (analogous to a preset). Each sub-module can make an audio signal with the length of 0.1-1 second. If the synthesizer has a stack size of more than 1 these audio signals are overlapped and the total amplitude is normalized.

2.2 The Ear

2.2.1 Feature Extraction

What we refer to as an "ear" is any method (such as machine listening) of evaluating a piece of audio, capable of "listening" to a piece of audio and giving it a score (or a list of scores) based on how well it satisfies certain criteria. Since in this work we are mainly focused on percussive generation, what we require from the ear is to give us probabilities of an audio sample belonging to various

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

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 to generate noise
Sample 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 Cuttoff	0-20000	-
LP filter Cuttoff	20000-HP	never lower than HP cutoff
Filter Order	4,8,16	butterworth filter order

Table 2.1: Synthesizer Sub-module Parameters. Despite the simplicity of the parameters and our efforts at constraining the ranges, the number of parameters that can be randomly chosen for each sub-module is in the order of 10^{15}

categories. As our synthesizer outputs are deterministic for all programs the ear's evaluations would allow us to associate the categorical probabilities of each sound with the program that generated it. In section 3 we discuss how these scores were used for navigation of our synthesis towards parameters which give us the desired sonic output.

With our dataset 1.3 of labeled drum sounds we can confidently categorized unlabeled drum sounds given that we know they are drum sounds. However, a major hurdle towards the implementation of a "drum from non-drum" recognizing ear is that the set of sounds that are not percussive is infinite. In our case, drum groups are an example of closed sets, since sufficiently large sample pack can effectively describe drum categories. However, effective representation of non-drum data is not practical. We hypothesize that the implementation of our virtual ear falls within the domain of "Open Set Recognition"[17]. The problem arises when a set that is being categorized has practically infinite diversity,making learning-by-example difficult without further domain knowledge. methodologies[7, 12].

[b]

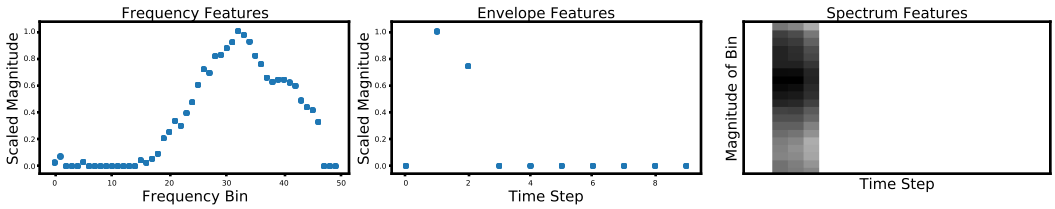

Figure 2.1: Recorded hat sample

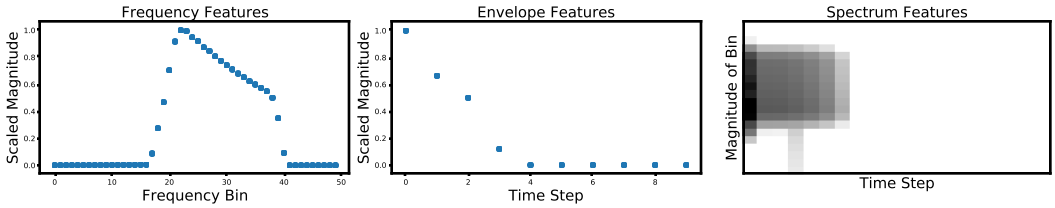

Figure 2.2: Randomly generated audio with percussive qualities, resembling a tight snare

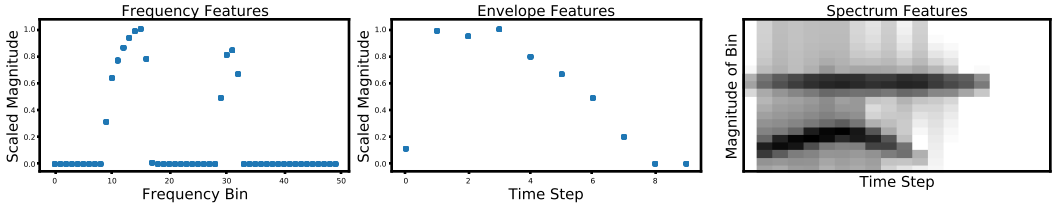

Figure 2.3: A randomly generated noise with a percussive envelop but non-percussive frequency features (modulated pitch)

Figure 2.4: 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.

In our case, as discussed in Section 3.2, the majority of sounds generated by the synth do not resemble percussion. Even if we only consider 1 second long sounds made with a single stacked synth, the set of synthesizer parameters (and we assume, the sound generated by the synth given these parameters) is extremely large. In our case, we determined that there are 2 steps necessary to effectively extract percussive sounds from the randomly generated sounds produced by virtual synthesizers:

1. Phase 1: Binary separation of sounds with percussive features from non-percussive sounds.

2. Phase 2: Given confidence that the sound being categorized is percussive, categorizing its type.

We are interested in extracting generalizable, domain agnostic features. In this work we rely entirely on Fast Fourier Transform (FFT) and by extension Short-time Fourier Transforms (STFT) for feature extraction. 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 a more accurate representation is desired; via the application of the FFT to a sliding time-window on the signal to create a matrix (a list of vectors). 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).

Various works have demonstrated effective reconstruction of signals given their STFT [13, 9]. For our purpose, if the original signal can be faithfully reconstructed from its STFT, analysis of the STFT may be relied on as source of fundamental features necessary for audible signal categorization. In the interest of keeping the feature space small and fundamental, other methods of feature extraction such as Spectral-Centroids [18] and Zero-Crossing Rates [8] are not utilized.

Using only the STFT of signals as source of feature extraction we defined 3 transformation functions which we believe to capture important, unique attributes of percussive sounds. These functions were applied at training time to 1 second audio samples before being sent to a classifier; transforming a signal into a set of features which we hope can capture the necessary information for our categorization task.

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. This vector is then normalized to the range of 0 to 1. The information contained in this vector is similar to that of a Root-Mean-Square measurement.

2. Frequency Transformation: A static, normalized snap-shot of the 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. Another important distinction is that since capturing an adequate frequency resolution is important for this transformation, we utilized shorter hop-sizes and wider windows. A Mel Scale transformation was also applied in hopes that the captured features better represent human perception of frequencies.

3. Spectrum Transformation: This function is simply a Mel Scaled STFT with its values normalized from 0-1. Since this features is a 2D matrix rather than a vector it captures more information about our signal but requires heavier, more complex computational methods to be utilized.

2.2.2 The Models

Using the described features, we trained several neural network models for Phase 1 and 2 in the Pytorch environment. The task of Phase 1 is to separate drums from not-drums (DrumVsNotDrum, or DVN). The task of Phase 2 is to categorize drums and percussion (DrumVsDrum, or DVD). We kept our feature space small, making it viable for feature selection and model design to be done on a trial and error basis. For all models, accuracy is calculated by prediction of all test dataset labels and the loss function and optimizer are Categorical-CrossEntropy and Adam respectively. Training continues until no reduction in loss and accuracy is observed in 10 epochs. All activation functions are PReLU:

1. FC-DVN: Fully connected network trained on Envelope features, reaching 97% accuracy on our test data for Phase 1. With size of 10x5x10.

Given these little subsections with good explanations

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. This model is trained on spectrum data and reaches 98% accuracy on our test set. Its structure is the combination of a CNN with 2 output channels and kernel size $(7, 3)$; Followed by an LSTM model of hidden size 800 and a fully connected layer of size 20x2.

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. Size of 50x10x2x6.

4. CNN-DVD: A CNN model trained on Spectrum features. Reaching 82% accuracy in a 6-way drum categorization in Phase 2. A combination of a CNN model with output channel size of 4, kernel of size of 5, another CNN model with output channel size of 8 and kernel of size 3. Followed by a fully connected network of shape 100x20x6.

5. FC-DVD: Fully connected 3 layer neural net with 78% accuracy for 6-way drum categorization in Phase 2. Size of 400x200x50.

The parameters are hand-picked and un-tuned. As discussed in section 3.2, higher accuracy rates in these models do not translate to higher agreeableness with humans. leading us to believe that model accuracy on test data alone cannot be relied upon when the domain of sounds being categorized is switched from original percussion samples to virtual synth sounds.

With our models showing high accuracy on testing data, we combine models in order to increase the efficacy of each phase and address the "open set problem" for our task. For Phase 1 we only determine sounds as percussive if both FC-DVN and CNN-LSTM have determined it as such with over 90% confidence. For the majority of our random generations that is not the case, but if a randomly generated sound has passed this phase, our three categorizers assign their categorizations to this sound. These categorizers have a moderate degree of agree-ability as seen in 3.2, but often the decision is not unanimous. The fourth method of categorization, "averaged-cat", is imple-

mented by taking the sum of the softmax outputs of all three categorizers, using it to determine the category.

These models can be combined and weighted in various ways and the confidence thresholds can be modified in order to implement "virtual ears" with different properties. A glaring issue in the current implementation is the treatment of softmax outputs as a reasonable measure of a model's confidence. Ignoring that some models may have unwarranted higher confidence in their scores, skewing attempts at finding a consensus.

And what do you do afterwards?
I need more detail here.

Chapter 3

Novel Generations

3.1 The Pipeline

With our *synth generator* and *virtual ear* in place we can quickly generate random virtual synthesizer programs. Once we render the virtual synthesizers with these programs and create the corresponding audio sample, we can categorize the audio sample. Simply put, we randomly create drum programs and generate and the corresponding signal. This signal is then fed through $Phase1$ and $Phase2$ of our ear model to determine if the signal is percussive. If so, a category is assigned to our sound. This sound can then be saved as an audio file, and its virtual synthesizer can also be saved.

With our current model, a single iteration of this pipeline will take approximately 50 milliseconds for Synthesizers with stack sizes of 8 or less. This means around 20 iterations can be done per second using a single process. This pipeline scales up efficiently with multiprocessing. We have not measured the iteration latency for this pipeline when a GPU is leveraged. For reference, our CPU measurements are collected on a 2012 Macbook Air running Ubuntu 18.04. In future work we will attempt guided generation via genetic and other heuristic search methods. Here we attempt a deep analysis of the randomly generated samples and the corresponding scores given by the ear. This analysis can expose weaknesses in our methodology which can be improved for future works where various search methods will be evaluated.

I need results. How many things does it try. How many programs or samples are made before it finds one?

3.2 Survey of Randomly Generated Drums

To measure the quality of the samples produced by our pipeline and the power of our models, we randomly generated around 50-60 samples in the following categories: "snare", "kick", "hat", "clap" and "other" (combination of rims, shakers and unusual percussive sounds"). These samples were determined to be percussive and then categorized by 4 different models (FC, CNNLSTM, E+F and AVG). 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). Both reviewers then categorized these samples without knowledge of other categorizations (human or computation models). It's important to note:

- Humans had an additional category of "bad" for samples that they deemed not percussive. The "bad" category indicates that the sample should have been skipped in Phase 1.
- With 6 categorization groups, humans had the same categorization in 47% of cases.
- The agreement between the humans and AVG was 44% and 47%, not significantly lower than agreement with each other.
- Of 257 samples humans agreed with FC, CNNLSTM, AVG and E+F respectively in 78, 76, 76 and 46 of cases.

We assess the reliability of agreement between humans and categorization models via the Fleiss' kappa coefficient [6]. The value of 0 or less for this coefficient indicates no agreement beyond random chance, and the value of 1 indicating perfect agreement. Our kappa measurements 3.1 lie within the 3.5-4.5 range, indicating mild to moderate agreement between humans and machines. We again measure this coefficient after dropping samples that were categorized as "bad" by the authors, as samples that humans deem to be "bad" indicate a failure in Phase 1 and arguably should not have been categorized by the models at all. Dropping of samples that both authors deemed "bad"

Drop Bad?	Size	HvH	H+FC	H+CNN	H+ALL	3 models
No	257	0.37	0.35	0.35	0.36	33
if both	236	0.31	0.37	0.37	0.38	33
if either	180	0.47	0.50	0.48	0.46	0.37

Table 3.1: Table of Fleiss' kappa coefficient to measure the degree of agreement between humans (HvH), humans with FC model (H+FC), humans with CNNLSTM model, humans with all models (H+ALL), and the 3 models

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. Possible takeaways from this survey:

- The survey brings into question the reliability of our phase 1 models, as 30% of the generated samples were deemed not percussive by at least 1 reviewer and 8% by both reviewers
- The task of categorizing synthetic drums is difficult. Survey shows that the scale of agreement within humans as well as between humans and various model combinations is moderate at best, even after removal of "bad" samples. While the same models can easily achieve 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.

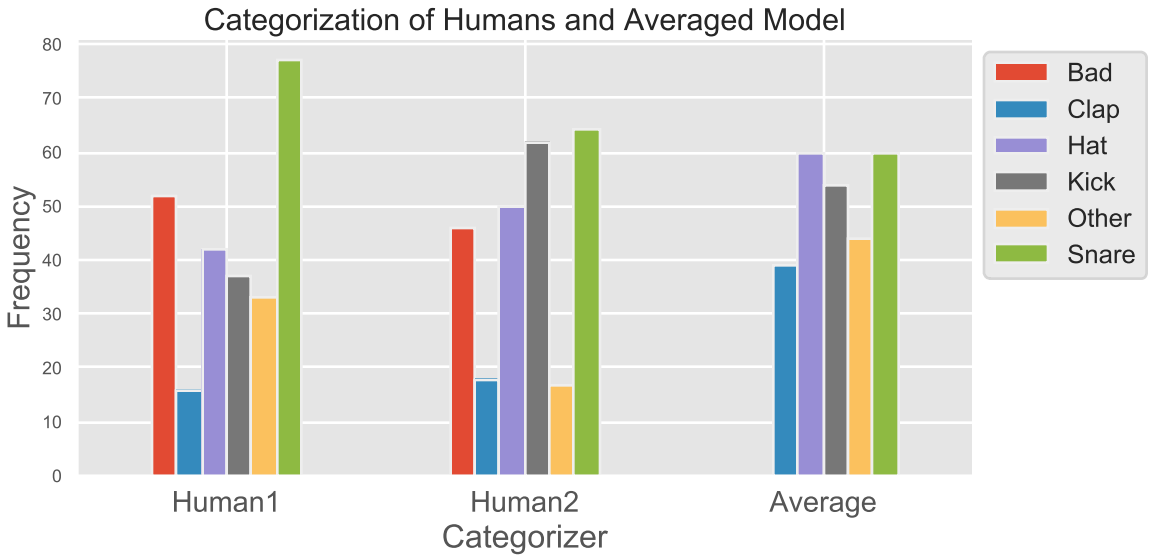

Figure 3.1: Frequency of assigned labels for the human categorizers and the average categorizer

Correctly???

I think you need to consider the problem of currating a collection as well.

Chapter 4

Conclusion

4.1 Summary

In summary we were capable of generating many new and novel drum samples that were agreed upon by human-raters and machines. Lessons learned included:

- Our methodology was successful in generating novel examples of some percussion categories.
- Generating virtual synthesizers allows for the fast generation and evaluation of audio created from a complex synth. Our implementation enables classification of the virtual synthesizer and sounds into percussive categories, enabling not only the creation of new libraries of percussion sounds, but new libraries of percussion synthesizers.
- Despite our careful feature extraction and training, our generative pipeline is not bullet-proof. Based on our human survey many of the samples generated in our pipeline are non-percussive.
- Implementation of an virtual ear that can distinguish sounds from closed sets (kick drums, snare drums etc) from an infinitely large set (non-percussive sounds) is difficult.

Based on our observations, the biggest hurdle towards generation of novel percussive sounds is an ear that can distinguish non-percussive sounds from percussive sounds.

4.2 Future Work

In future work, we will continue our attempt at effective, domain agnostic separation of noise from drums. Our focus will be on extraction of generalizable features which can help us with classification problems regardless of the audio domain i.e source of audio.

We will seek solutions from the state of the art methods in regards to open set recognition. We plan to utilize genetic search and other heuristic methods for guided generation of sound rather than random search applied to a heuristic. We will also introduce new features into our virtual synthesizer to add diversity to generated samples.

Caution: For cross-references to work, when files are compiled separately, the referenced file must be compiled at least once before the referring file is compiled.

References

[1] 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.

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

[3] 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.

[4] 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.

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

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

[7] Chuanxing Geng, Sheng-jun Huang, and Songcan Chen. Recent advances in open set recognition: A survey. *arXiv preprint arXiv:1811.08581*, 2018.

[8] Fabien Gouyon, François Pachet, Olivier Delerue, et al. On the use of zero-crossing rate for an application of classification of percussive sounds. In *Proceedings of the COST G-6 conference on Digital Audio Effects (DAFX-00), Verona, Italy*, page 26, 2000.

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] António Ramires, Pritish Chandna, Xavier Favory, Emilia Gómez, and Xavier Serra. Neural percussive synthesis parameterised by high-level timbral features. *arXiv preprint arXiv:1911.11853*, 2019.

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

[17] Walter J Scheirer, Anderson de Rezende Rocha, Archana Sapkota, and Terrance E Boult. Toward open set recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(7):1757–1772, 2012.

[18] Emery Schubert, Joe Wolfe, and Alex Tarnopolsky. Spectral centroid and timbre in complex, multiple instrumental textures. In *Proceedings of the international conference on music perception and cognition, North Western University, Illinois*, pages 112–116. sn, 2004.

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

[20] Ryuichi Yamamoto, Eunwoo Song, and Jae-Min Kim. Parallel wavegan: A fast waveform generation model based on generative adversarial networks with multi-resolution spectrogram. *arXiv preprint arXiv:1910.11480*, 2019.

[21] 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.

Appendix A

Background Material

Material in an appendix.

We plot an equation in figure A.1.

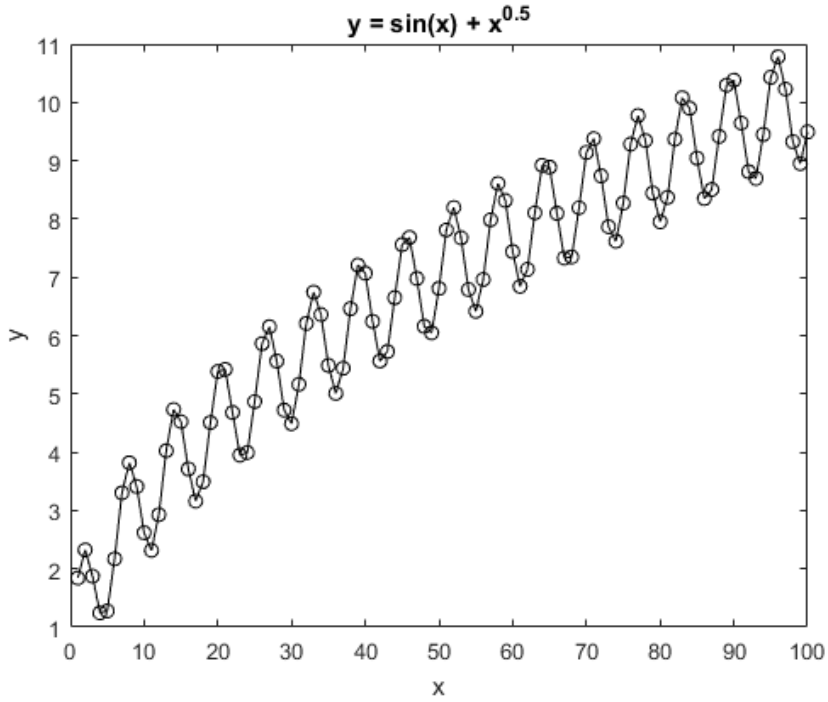

Figure A.1: A graph of $y = \sin(x) + \sqrt{x}$