SOUND2SYNTH: Interpreting Sound via FM Synthesizer Parameters Estimation

Zui Chen[1*], **Yansen Jing**[1*], **Shengcheng Yuan**[2*], **Yifei Xu**[2], **Jian Wu**[2] and **Hang Zhao**[1]

[1]Institute for Interdisciplinary Information Sciences, Tsinghua University
[2]Beijing DeepMusic Technology Co., Ltd

{chenzui19, jingys19}@mails.tsinghua.edu.cn, {yuansc,xuyf}@lazycomposer.com,
wujian7752@163.com, Zhaohang0124@gmail.com

arXiv:2205.03043v1 [cs.SD] 6 May 2022

Abstract

Synthesizer is a type of electronic musical instrument that is now widely used in modern music production and sound design. Each parameter configuration of a synthesizer produces a unique timbre and can be viewed as a unique instrument. The problem of estimating a set of parameters configuration that best restore a sound timbre is an important yet complicated problem, i.e.: the synthesizer parameters estimation problem. We proposed a multi-modal deep-learning-based pipeline SOUND2SYNTH, together with a network structure Prime-Dilated Convolution (PDC) specially designed to solve this problem. Our method achieved not only SOTA but also the first real-world applicable results on the Dexed synthesizer, a popular FM synthesizer.[1]

1 Introduction

1.1 Background and Motivation

The rapid development of computer hardware and digital audio processing technologies has greatly influenced the music industry in recent years. With the help of Digital Audio Workstation (DAW) and Virtual Studio Technology (VST), musicians are now able to finish the entire composition, orchestration, mixing, and mastering process on their computer or hardware for special use, which could provide real-time sound feedback to the musician, and are much cheaper than an actual band or professional orchestra.

As the virtual instruments become more delicate, they are also becoming larger in size – professional virtual instrument libraries are reaching gigabyte level memory and terabyte level of disk storage for the audio samples – and more expensive to record and use. Also, many sounds in musicians' imagination are not easily recordable or even producible in nature. Both problems inspired the idea of generating sound by only manipulating digital waveform signals, without the need of using a large size or a large number of samples.

A synthesizer is an electronic musical instrument that generates audio signals from given MIDI inputs, relying mainly on computation instead of audio samples. The main types of synthesizers include: additive/subtractive synthesizers, which construct waveforms by adding simple waves to silence or filtering out simple waves from white noises; FM synthesizers, which use cascaded oscillators whose parameters are controlled by predecessors to obtain complex waveforms at the output; wavetable-based synthesizers, which manipulate a short sample (i.e.: a wavetable) or combine multiple samples to create complex sounds; and analog synthesizers, which simulate physical circuits or devices to generate sounds.

Most synthesizers have a large number of parameters, which are necessary for their expressiveness. However, many of the parameters are non-intuitive for humans, which means that non-experts can not imagine a sound from a set of parameters configuration (i.e.: a *preset*), or conversely, determine a synthesizer preset from a given sound. This is the hard problem of Digital Sound Design, which is now evolving into a complex subject and major that requires years of learning and practicing to expertise in.

Our research addresses the specific problem of synthesizer parameters estimation in digital sound design: finding a close synthesizer preset given a sound. This is a problem that is not well-studied while having a huge potential benefit in terms of both industrial and artistic perspectives. For example:

- **Sound Design**: In digital sound design, musicians want to compose with natural/imaginal sounds as synthesizer presets (so that the sound can smoothly transfer to different pitches and durations). Synthesizer parameters estimation is able to remarkably speed up the process of making new presets, helping musicians to focus on the creative job only instead of the technical job.
- **Sound Compression**: A synthesizer preset usually takes thousands of times or lesser space than the timbre sound wave it produces, thus synthesizer parameters estimation can be used in sound compression for efficient storage or transmission under circumstances that could tolerate minor trade-offs in quality for speed or memory efficiency (e.g.: online composing, preview rendering).
- **Synthesizer Expressiveness Test**: A good synthesizer parameters estimator can also serve as a quantitative benchmark for comparing the expressiveness of synthesizers, by providing a set of random/naturally-distributed/domain-specific audio samples and evaluating

[*]These authors contributed equally to this paper.

[1]Code, plug-in, demo clips, demo video, and demo songs are all available at the link: https://github.com/Sound2Synth/.

the average distance between the original samples and the closest generated samples by each synthesizer.

In a nutshell, synthesizer parameters estimation is a problem of great importance to music creativity and has a great market value in the modern digital music industry.

1.2 Problem Definition

Formally, a Synthesizer can be viewed as a function mapping $f : \Theta_f \times \mathcal{M} \to \mathcal{A}$, where $\mathcal{A}$ is the audio space, Θ_f is the configuration space of this particular synthesizer f, each $\theta \in \Theta_f$ is a preset of the synthesizer, $\mathcal{M}$ is the MIDI configuration space of a single note and each $\eta \in \mathcal{M}$ is the MIDI setting of a note, specified by note pitch, note velocity, and note duration, etc. Although this process is influenced by various other settings including sample rate, bit depth, etc, these settings are usually fixed and can be easily converted if necessary.

It is worth mentioning that almost all practical synthesizers are non-surjective functions, which means that there almost certainly exists audio $A \in \mathcal{A}$ that can not be generated by a specific synthesizer f.

The Synthesizer parameter estimation problem can be formulated as follows: given an audio $A \in \mathcal{A}$ and a fixed input note $\eta_0 \in \mathcal{M}$, the goal is to find a preset that can best restore the audio under a particular distance metric:

$$\min_{\hat{\theta} \in \Theta_f} \mathrm{dis}(f(\hat{\theta}, \eta_0), A) \tag{1}$$

Our experiments mainly concentrate on the Dexed synthesizer [Gauthier, 2013], since it is one of the most popular open-source FM synthesizers, and it has a sufficient amount of free presets easily accessible on the Internet. We will use mean squared Euclidean Distance between Mel-Frequency Cepstral Coefficients Distance (MFCCD) as our evaluation metric, since MFCCD is proven by experiment to be well-aligned with human perception [Terasawa *et al.*, 2005], and is widely used in various audio-related tasks, e.g.: speech recognition.

1.3 Related Work

The work in synthesizer parameters estimation can be traced back to the work of Horner et al. [Horner *et al.*, 1993], which used a genetic algorithm to find the settings of parameters for frequency modulation matching synthesis. Mitchell et al. utilized the advances in multi-modal evolutionary optimization to perform dynamic sound matching of FM synthesizer [Mitchell and Sullivan, 2005].

Roth [Roth and Yee-King, 2011] first introduced neural networks into this problem and systematically compared the performance of traditional methods and neural networks. Matthew Yee-King [Yee-King *et al.*, 2018] further developed the usage of neural networks by using LSTM++ (bidirectional LSTM with highway connections) networks. They conducted experiments to compare their method APVST with various traditional approaches on the Dexed synthesizer, achieving comparable performance, while being much more efficient. However, results comparable to traditional approaches are still far from sufficient for real-world application.

After Yee-King's work, researches are also carried out on different synthesizers, e.g.: InverSynth [Barkan and Tsiris, 2018] on JSyn [Burk, 2014], SerumRNN [Mitcheltree and Koike, 2021] on Serum [Records, 2014]. Both works are using only simple network architectures: InverSynth used vanilla CNN networks, and SerumRNN used vanilla RNN networks. Also, Jsyn was a small synthesizer whose expressiveness is not sufficient for wide application, and SerumRNN focused only on the effects applied to the sounds, instead of the full preset configuration of Serum.

An interesting method FlowSynthesizer[Esling *et al.*, 2019] using auto-encoder and normalizing flow, was tested on Diva [u-he Software Synthesizers and Effects, 2011] – an analog synthesizer. This method does not rely on a running synthesizer instance but only on the data points (i.e.: audio paired with ground truth preset), and it achieved amazing results at the time. However, it was only able to achieve a relatively satisfactory result on a mere subset of Diva parameters instead of the entire Diva synthesizer. A similar method was used in follow by PresetGen VAE [Le Vaillant *et al.*, 2021], which is the previous SOTA on the Dexed synthesizer.

Another direction is to build differentiable synthesizers [Engel *et al.*, 2020], which could then easily be integrated with neural network models. However, such a result could not be directly applied to existing commercial synthesizers, which are powerful but non-differentiable.

According to the works mentioned above, learning-based results comparable to traditional methods could be achieved, while there is no work so far producing human-tolerable results on a powerful synthesizer with full configuration space. Also, many of the works are limited to a single synthesizer or a synthesizer type.

Thus, we aim to develop a pipeline that could work on different types of synthesizers, achieving not only SOTA but also real-world applicable results on full configuration space.

1.4 Novelty

Our main contributions in this paper are listed as follows:

- We proposed **SOUND2SYNTH** – a deep-learning-based FM synthesizer parameter estimation pipeline, which does not require an online instance of a synthesizer during training. The proposed pipeline has achieved SOTA in quantitative comparison and is close to applicable in terms of human perception (Tab. 1). Our pipeline could also in principle generalize to additive, subtractive and simple analog synthesizers.
- We proposed **Prime-Dilated Convolution (PDC)** – a new convolution network layer structure specially designed for better utilization of Constant-Q Transform (CQT) chromagram information of an audio sample.
- We demonstrated the benefit of using **Multi-modal Features** in a combined network. The experiment showed that spectral (visual), waveform (sequential), and statistical (numerical) sound feature information altogether improved the generalizability of the model.
- We introduced various techniques for optimizing sound processing for neural networks and revealed many inspiring discoveries, which could as well be applied to other audio-related tasks.

2 Methodology

2.1 A Closer Look at the Dexed Synthesizer

Frequency Modulation synthesis (or FM synthesis) is a form of sound synthesis whereby the frequency of an oscillator is changed by modulating its frequency with a modulator. The frequency of an oscillator is altered in accordance with the amplitude of a modulating signal. The modulator can be another oscillator, whose frequency can be modulated again by the third oscillator, and so on.

The Dexed synthesizer [Gauthier, 2013] is an open-source software synthesizer with 6 oscillators, aimed to model the FM synthesis algorithm in the Yamaha DX7 hardware, which is one of the best known and most successful synthesizers.

Figure 1: Screenshot of Dexed. Left: The user interface of the full Dexed synthesizer. Right: One oscillator of the Dexed synthesizer.

2.2 Overall Pipeline

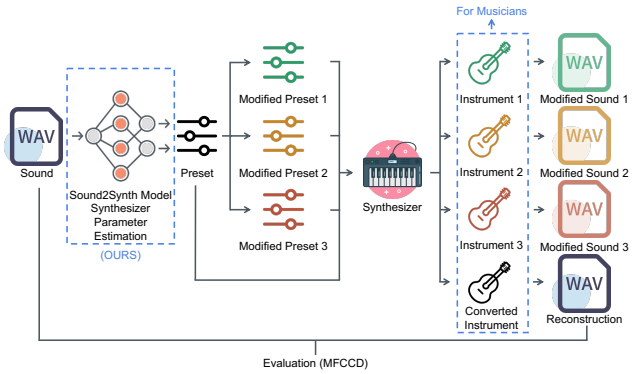

Figure 2: SOUND2SYNTH overall pipeline illustration.

For each sound, it is first converted to different forms to address different aspects, including: Short-Time Fourier Transform (STFT) spectrogram, Mel spectrogram, CQT chromagram, MFCC, and other statistical information.

Each form of the input is then fed to the corresponding backbone that best fits for preprocessing: spectrograms are handled by CNNs[2]; chromagram is handled by Prime-Dilated Convolutional Neural Network (PDCNN), which is an original structure and will be elaborated in Sec. 2.3; MFCC is handled by an LSTM network [Hochreiter and Schmidhuber, 1997]; other statistical information is handled by simple MLPs. The processed features of each input form are concatenated to obtain the global features, containing all the information extracted from the input audio.

[2]In fact, among vanilla CNN [LeCun *et al.*, 1998], VGG [Simonyan and Zisserman, 2015], ResNet [He *et al.*, 2015] and DenseNet [Huang *et al.*, 2016], VGG prevailed to be the best backbone for audio-related tasks by preliminary experiments.

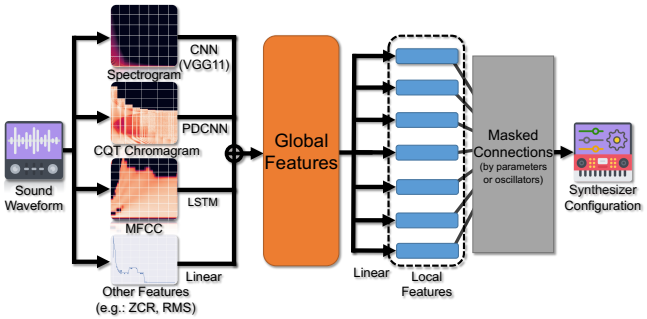

Figure 3: SOUND2SYNTH model architecture illustration.

For parameters estimation, we first use a single linear layer with non-linearity to process the global features. Then we split the processed global features into groups of local features divided by oscillators or specific parameters. Then the local features are dealt with by multiple layers of linear and non-linearity connections masked according to parameters.

Finally, each parameter prediction is derived from the oscillator's local feature group using additional MLPs. Since continuous parameters in synthesizers usually have minimal precision, we can convert all continuous parameters into discrete parameters, therefore converting all real-value estimation problems to classification problems.

It is worth highlighting that, our pipeline supports not only online training but also training via offline datasets, which means it does not require access to a synthesizer instance during training. Synthesizer rendering is itself a computation-heavy operation, which would become the bottleneck if involved in the training procedure. Thus our workflow that allows separating the model from a synthesizer instance is more efficient and sometimes necessary in a real-world scenario.

Thus, we evaluate our model on configuration space Θ_f during training. That is if we denote the network as $N_{f,\varphi} : \mathcal{A} \to \Theta_f$, parameterized by φ, instead of optimizing the following true objective over an offline dataset $\mathcal{D}$:

$$\begin{aligned} &\min_{\varphi} \mathcal{L}_{\text{MFCCD}}(\varphi) \\ &= \min_{\varphi} \mathbb{E}_{(A^i,\theta^i)\sim\mathcal{D}}[\text{MFCCD}(f(N_{f,\varphi}(A^i),\eta_0),A^i)] \end{aligned} \quad (2)$$

During training, we only aim to optimize the Mean Squared Error (MSE) between predicted parameters and the groundtruth parameters:

$$\min_{\varphi} \mathcal{L}_{\text{MSE}}(\varphi) = \min_{\varphi} \mathbb{E}_{(A^i,\theta^i)\sim\mathcal{D}}[\text{MSE}(N_{f,\varphi}(A^i),\theta^i)] \quad (3)$$

This may cause dis-alignment between the training objective and the desired goal since:

- Parameters have different importance, a wrongly predicted "coarse" parameter has a larger influence than a wrongly predicted "fine" parameter.

- Different configurations (in terms of numerical values of parameters) may produce similar sound timbres.

We proposed *gradient-inspired weighting* technique to handle this problem, which will be explained in Sec. 2.5.

2.3 Prime Dilated Convolution

Intuition

The general CNN treats image and spectrogram as the same data structure and ignores significant distinctions in their contents. One of the most essential distinctions comes from the harmonic characteristics of sound. Due to the physical properties of resonance frequency and mechanical waves, whenever a sound vibrates at fundamental frequency F_0, a series of frequencies at its integer multiples ($2F_0$, $3F_0$, etc.), called harmonics, are also likely to vibrate. This fact results in a common observation of multi-stripes shape vertically stacking in the spectrogram, as depicted in Fig. 4. In this paper, we refer to this phenomenon as *harmonic features*.

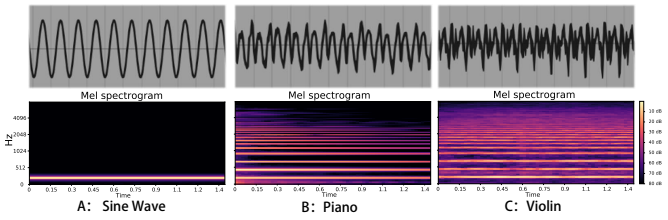

Figure 4: Illustration of different sound timbres having different harmonic features. From left to right, audios are obtained from sine wave, piano, and violin, separately. The audios are transformed into Mel-spectrogram. The phenomenon of multi-stripes shape due to the harmonics is very clear in these acoustic scenes.

Considering relationships among harmonics as different timbres, downstream networks (in this case, synthesizer parameters classifier) can utilize and leverage these harmonic features to improve performance. PDC constructs a sparse filter by expanding the concept of dilated convolution to accurately reach all the integer harmonics in a log-scale spectrogram. Unlike the regular dilated convolution which has a fixed dilated step, PDC's dilated location is not evenly distributed, since the distance between two harmonics in a spectrogram is not constant. PDC reaches all integer harmonics not at once, but through stacking itself. We apply the mathematical rule of prime factorization, decomposing an integer into the product of a few prime numbers. These primes are further decomposed into the product of some integer ratios between 1 and 2. PDC's dilated location is then built according to these ratios. In this way, PDC has a fixed receptive field, and it only requires a few primes to complete the filter construction, rather than traversing every integer.

Prerequisite

Let $\mathbf{X} \in \mathbb{R}^{C\times K\times T}$ denote the spectrogram, where C is the number of channels, K is the number of frequency bins, and T is the number of temporal segments. Let $f(k)$ denote the frequency at the k-th bin, then $\mathbf{X}(c,k,t)$ refers to the energies of sound in c-th channel, at the t-th time frame, around frequency $f(k)$. The inverse function of f is denoted as $f^{-1}(\cdot)$, which gives the index of the frequency bin according to the real frequency.

The only prerequisite of PDC is for $\mathbf{X}$ to be a log-scale spectrogram, which means that $f(k)$ is an exponential function. In this paper, we use standard Constant-Q Transform (CQT) to generate spectrograms, where $f_{CQT}(k) = f_{\min}2^{k/B}$, $f_{\min}$ is the lowest frequency that CQT covers (by default, $f_{\min} \approx 32.70$Hz), and B denotes *bins per octave*, i.e., the number of bins between any frequency and its double frequency, which refers to the resolution of the spectrogram. The goal of prime-dilated convolution is to reach any integer harmonics in the spectrogram by setting the dilated location. Thus, there is a need to measure the distance between two harmonics. Let $d(n,m,F) = |f^{-1}(mF) - f^{-1}(nF)|$ denote the distance of bins between n-times and m-times harmonics based on their common fundamental frequency F. In CQT, the distance $d(n,m,F)$ does not change with F.

$$\forall n,m \in \mathbb{N}, \quad \forall F_1,F_2 \in \mathbb{R}^+,$$
$$d(n,m,F_1) = d(n,m,F_2) = \left|B\log_2\frac{m}{n}\right| \tag{4}$$

We will use $d(n,m)$ instead of $d(n,m,F)$ in the remaining part of the paper for simplicity.

Although $d(n,m)$ does not change with F, it is affected by n and m, according to Eq. (4). Therefore, the dilated step cannot be constant. Because n and m may vary within a wide range, solving Eq. (4) to get every possible distance between two harmonics would create a huge amount of dilated locations, i.e., creating a tall and dense filter with a lot of parameters. Instead, we introduce the prime-ratio function to represent all the distances using a small list of prime numbers and create a sparse filter.

Prime-Ratio Function

For any prime number p, the prime-ratio function $r(p)$ is defined as follows.

$$r(p) = 2^{-s}p, \quad s = \max\{s \in \mathbb{N} | 2^s < p\} \tag{5}$$

Given the mathematical rule of prime factorization, i.e., for every $n \in \mathbb{N}$ and $n \geq 2$, there exists only one way to decompose n into the product of prime powers. Considering that $r(2) = 2$, therefore, $p = r(2)^s r(p)$, and the following theorem holds true for every integer.

Theorem 1. *For all $n \geq 2$, n can be represented in exactly one way as a product of the prime-ratio powers, i.e.,*

$$n = \prod_{i=1}^{l} r(p_i)^{\alpha_i} \tag{6}$$

where $p_1 < p_2 < \cdots < p_l$ are prime numbers and α_i's are positive integers.

Calculating $d(1,n)$ based on Eq. (6) results in the following equation:

$$d(1,n) = \sum_{i=1}^{l} \alpha_i d(1, r(p_i)) \tag{7}$$

Eq. (7) states that the distance between any integer harmonic and its fundamental frequency can be represented as a finite linear summation of the prime-ratio's distance $d(1,r(p_i))$, where $d(1,r(p_i))$ has two characteristics:

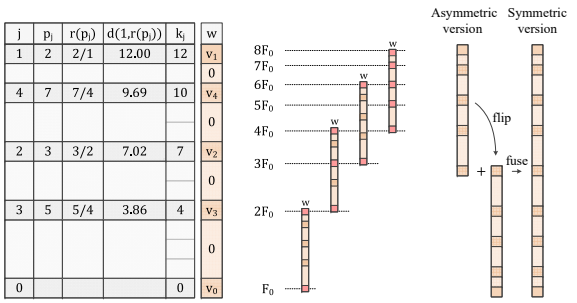

Figure 5: Main concept of the prime-dilated convolution, illustrated with the hyper-parameters $B = 12$ and $l = 4$. (a) shows the dilated structure of asymmetric version, where j is index, p_j is the j-th smallest prime, $r(p_j)$ is the prime-ratio function, $d(1, r(p_j))$ is the distance of bins to the bottom line, $\vec{v}$ is the trainable parameters, $\vec{w}$ is the filter after dilation, and k_j is the dilated location. (b) illustrates the way the filter shifts and stacks to capture all the harmonics in a log-scale spectrogram. (c) introduces the symmetric version of PDC which is constructed from the first version's flip and fusion.

1. It always represents the distance between two integer harmonics, because according to Eq. (4) and (5),

$$d(1, r(p_i)) = d(2^{s_i}, p_i) \tag{8}$$

where 2^{s_i} and p_i are both integer numbers.

2. It always lies between 0 and B, since $r(p) \in (1, 2]$ clearly is true for every prime, and $0 = d(1, 1) < d(1, r(p_i)) \leq d(1, 2) = B$.

Formulation

The main concept of PDC is inspired by the above theorem and their inferences. Because: 1) any integer can be represented as the product of $r(p)$; 2) the product of $r(p)$ appears as the summation of $d(1, r(p))$; and 3) if the dilated location is set as $d(1, r(p))$, the summation refers to the shift and stack of convolution operation. In other words, if a series of distance $d(1, r(p_i))$ is taken as dilated locations as shown in Fig. 5, then any integer harmonics will be captured by the shift and stack of the filter. For example, $6 = r(2)^2 r(3)$, so the six-times harmonics can be reached by the stack of dilation $d(1, r(2))$, $d(1, r(2))$ and $d(1, r(3))$. In our experiments, we obtain such a stack by inserting a single PDC filter after every convolutional layer. All the distances $d(1, r(p))$ are no greater than B, leaving PDC with a fixed receptive field.

In practice, the dilated location in a filter has to be an integer, but $d(1, r(p_i))$ is often an irrational. Therefore, PDC constructs the dilation according to the integer approximation of $d(1, r(p_i))$ sequence, which is formally defined as follows:

Let $p_1 < p_2 < \cdots < p_l$ denote the smallest l prime numbers selected. Let k_j denote the integer which is the closest to $d(1, r(p_j))$, we then have:

$$\begin{aligned} k_0 &= 0, \\ k_j &= \arg\min_k |k - B \log_2 r(p_j)|, \quad j = 1, \ldots, l \end{aligned} \tag{9}$$

We introduce two versions of the PDC filter in terms of how to construct the dilated locations.

Asymmetric version. Let $K = \{k_j\}_{j=0}^{l}$ denote the set of dilated locations. Let vector $\vec{v} = (v_0, v_1, \ldots, v_l)$ denote the trainable parameters in PDC. Let $\vec{w} = (w_0, w_1, \ldots, w_B)$ denote the vector after dilation, where the receptive field is $(B + 1) \times 1$. The values of w_k are defined as follows.

$$\begin{aligned} w_{k_j} &= v_j, \quad j = 0, 1, \ldots, l \\ w_k &= 0, \quad k \notin K \end{aligned} \tag{10}$$

Eq. (11) creates an asymmetric structure $\vec{w}$ which covers only higher harmonics, as shown in Fig. 5 (a) and (b).

Symmetric version. Symmetric PDC is created by flipping and fusing the asymmetric version, as shown in Fig. 5 (c). Let $K = \{k_j\}_{j=-l}^{l}$ denote the set of dilated locations, where the negative index of k_{-j} is defined as k_j's opposite:

$$k_{-j} = -k_j, \quad j = 1, \ldots, l \tag{11}$$

Let vector $\vec{v} = (v_{-l}, \ldots, v_l)$ denote the trainable parameters in PDC, and $\vec{w} = (w_{-B}, \ldots, w_B)$ denote the vector after dilation, where the receptive field is $(2B + 1) \times 1$. The values of w_k are defined as follows:

$$\begin{aligned} w_{k_j} &= v_j, \quad j = -l, \ldots, l \\ w_k &= 0, \quad k \notin K \end{aligned} \tag{12}$$

The convolution operation $pdc(\cdot)$ can be parameterized with the dilated filter constructed by $\vec{w}$.

2.4 Multi-modal Feature Engineering

Besides spectrograms, chromagrams, and MFCC, we also utilized certain statistical features in the network, which are closely related to sound timbre. For example, the following information is widely used in audio processing tasks:

- **Amplitude Envelop**: The changes in the amplitude of a sound over time.
- **RMS Energy**: The root mean square energy of audio.
- **Zero Crossing Rate**: The rate at which a signal changes between positive value and negative value.
- **Wiener Entropy**: Also known as *Spectral Flatness*, a metric to measure whether a sound is tonal or noisy.

Notice that the information is scalars per time step, given that a fixed input note has a fixed duration, we can directly use an MLP mapping from time steps to feature dimensions to process each statistical information.

2.5 Techniques

Label Smoothing

As mentioned above, by discretizing continuous parameters, all parameter estimation could be treated as classification problems. In practice, all continuous parameters are in $[0, 1]$ range and are divided into K segments, as in a K-way classification task.

Unlike normal classification tasks, discretized numerical classes are not symmetric – wrongly classifying a class as an adjacent class has a smaller influence than classifying it as an arbitrary other class. Thus, we can split part of the probability mass of the ground truth label into neighboring classes. Technically, the ground-truth label of length K would first be 1d-convoluted with a Gaussian kernel of $\sigma = \sigma_0/K$, normalized to have a total probability mass sum to 1, and then be used as the target for cross-entropy loss computation.

Gradient-Inspired Weighting

As mentioned in Eq. (2.2), only considering MSE loss on parameter space could result in overfitting the configuration space, while performing badly in the audio space.

Observation 1. *Most parameters in most presets are local continuous: a small change in the preset would also indicate a small change in the rendered audio and MFCC.*

This implies that we can approximate local audio space loss using a linear loss term.

Observation 2. *Based on preliminary experiments, our model would be able to generate predictions relatively close to the ground truth.*

This implies that we can use the gradient field around ground truth θ^* to substitute the one around prediction $\hat{\theta}$.

Combining the observations, we can approximately state:

$$\begin{aligned} & \left.\frac{\partial \mathcal{L}_{\text{MFCCD}}(\theta)}{\partial \theta_i}\right|_{\theta=\hat{\theta}} \\ = & \left.\frac{\partial \mathcal{L}_{\text{MFCCD}}(\theta)}{\partial \mathcal{L}_{\text{MSE}}(\theta_i)}\right|_{\theta=\hat{\theta}} \cdot \left.\frac{\partial \mathcal{L}_{\text{MSE}}(\theta_i)}{\partial \theta_i}\right|_{\theta=\hat{\theta}} \\ \approx & \left.\frac{\Delta \mathcal{L}_{\text{MFCCD}}(\theta)}{\Delta \mathcal{L}_{\text{MSE}}(\theta_i)}\right|_{\theta=\theta^*} \cdot \left.\frac{\partial \mathcal{L}_{\text{MSE}}(\theta_i)}{\partial \theta_i}\right|_{\theta=\hat{\theta}} \end{aligned} \tag{13}$$

By preprocessing $\left.\frac{\Delta \mathcal{L}_{\text{MFCCD}}(\theta)}{\Delta \mathcal{L}_{\text{MSE}}(\theta_i)}\right|_{\theta=\theta^*}$ for each parameter θ_i of each preset θ^* in dataset $\mathcal{D}$, we can estimate an importance weight of prediction $\hat{\theta}_i$, to be used in training.

This technique could be applied only if the number of training samples in dataset $\mathcal{D}$ is small, since rendering audio using a synthesizer and computing audio space loss for every parameter of every data point is very time-consuming. Optimizing this method will be left as future work.

3 Dataset

There are 155 parameters for the Dexed synthesizer in total: 94 continuous parameters, 59 discrete parameters, and 2 fixed parameters (*Algorithm* and *Output*).

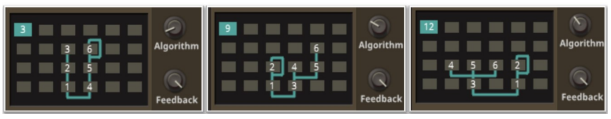

Figure 6: Different *Algorithm*s. There are 32 *Algorithm*s in total.

Algorithm is a special parameter in Dexed, which determines the modulation relationship between the six oscillators. The physical meaning of all parameters depends on the choice of *Algorithm*. WLOG, we restrict our experiment on a fixed *Algorithm* setting. In practical application, a model should be trained for different algorithms respectively. And then we select the output of the model with the smallest audio space loss as the final output of the system.

There is another special parameter called *Output*, which controls the volume of the generated sound. In our datasets, we fixed the value of *Output* to the maximum value, otherwise, the generated audio loudness of many presets is so low that it affects the perception of the human ear. In the model training and inferencing process, we also fixed the *Output* parameter, which is consistent with the data set. Fixing this parameter is also a reasonable decision in practice since users can easily adjust the overall volume of the synthesizer output afterward, and it is often necessary to do so.

We combined three different methods to generate datasets:

- **Preset Based**: We collected 100k+ presets on the Internet, which are widely used in real-life music composition. We input each preset θ^i into the Dexed synthesizer to render the corresponding audio files A^i. We split the preset-audio pairs into 32 sub-datasets according to different values of *Algorithm*.
- **Preset Augmentation**: We applied simple data augmentation to enlarge the dataset. Given a preset, we can fix the value of most of its parameters and then uniformly sample the values of the remaining parameters.[3] Note that randomly generated presets may not be audible, we set a minimum threshold of audible volume to sieve those presets out.
- **Random Walk Based**: To improve and test generalizability, besides collecting presets online, we also randomly sampled presets from configuration space to construct a random dataset. Similarly, we only preserved presets that can generate audible sounds.

4 Results

Quantitative Results on Dexed

Method	MFCCD
*Hill Climbing	21.96
*Genetic Algorithm	31.32
APVST MLP	31.38
APVST LSTM	32.76
APVST LSTM++	22.59
PresetGen VAE	14.70
*Similarity Threshold for Human Perception	$10 \sim 15$
SOUND2SYNTH (OURS)	6.32
SOUND2SYNTH multi-modal (OURS)	**5.36**

Table 1: Experiment results. MFCCD is the lower the better. All MFCCDs are measured under T6 setting: 6 oscillators on Dexed.

The detailed experiment settings are elaborated in Appendix. A.

From a quantitative perspective (Tab. 1), our model largely outperforms previous SOTA: PresetGen VAE [Le Vaillant *et*

[3] Dexed presets are grouped into different *themes*, which are split and augmented separately so that there is no data leakage during augmentation.

*These figures obtained from APVST [Yee-King *et al.*, 2018]. Our subjective listening test also agrees with this similarity threshold.

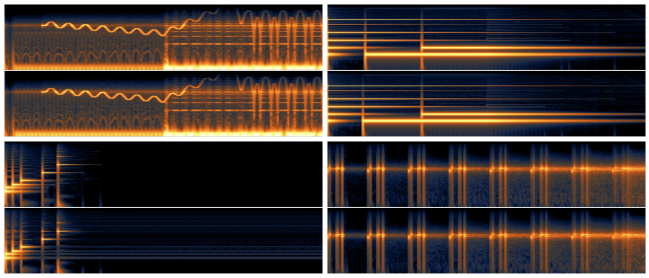

Figure 7: Four sampled spectrogram cases. Ground truth audios are on the top and audios generated using predicted parameters are on the bottom inside each group.

al., 2021].[4] From a visual perspective (Fig. 7), the spectrograms of our predictions are very similar to that of the ground truths. From an auditory perspective, audios generated using predicted preset and the ground truth audio are very alike.

5 Conclusion

We proposed a novel multi-modal pipeline, along with a prime-dilated convolution structure and many other useful techniques in audio processing, to tackle the synthesizer parameters estimation problem. The result of our pipeline, SOUND2SYNTH, is not only significantly better than previous SOTA on the Dexed synthesizer but also able to reach human auditory perception precision. We have released code, plug-in, audio demos, and example use cases in which our plug-in is boosting musicians' creativity and simplifying the process of creation substantially. This could have an impact on the development of AI for art in the field of music and sound design, and it could be beneficial for other audio processing tasks in the future.

A Experiment Settings

WLOG, we fixed the input note η_0 to be at the middle C pitch (C4). The note is always pressed with maximum velocity, sustained for 4 beats, and recorded 8 beats in total, under tempo 120 bpm. All audios are converted to 48kHz sample rate and 32 bit depth. All 6 oscillators of Dexed are used, including 155 parameters in total: 94 continuous parameters, 59 classification parameters, and 2 fixed parameters. All continuous parameters are discretized into 64 classes.

Our experiments are carried out on a pre-generated dataset containing 30106 training/validation data points and 1679 test data on Dexed. Among the training/validation data points, 6191 are directly sampled from existing presets, 22237 are augmented from those presets, and 1678 are generated purely at random. In practice, 80% of the data points are used for training and 20% are held out for validation. Notice that the test dataset is generated from independent held-out themes of presets and random-walk is not used, preventing data leakage.

[4]MFCCD metric is influenced by the number of filter banks, however, we computed MFCCD under both 13-band ([Yee-King *et al.*, 2018]) and 40-band ([Le Vaillant *et al.*, 2021]) settings and observed no remarkable difference in the evaluation results. Thus we reported the 13-band MFCCD in Tab. 1.

On model architecture, the extracted global features have the same dimension of 2048 for all model structures. In the case of the multi-modal structure, each backbone is assigned a small portion of features. Specifically, convolutional backbones, which are used to extract features from spectrogram and CQT chromagram, each have an output dimension of 512, while other backbones, which are used to extract features from waveform, MFCC, or statistical information, each have an output dimension of 128. The masked classifier has 64 hidden neurons for each group (a parameter or an oscillator).

We trained our models using the AdamW [Loshchilov and Hutter, 2019] optimizer with a universal weight decay 10^{-4} and a linear warm-up cosine annealing scheduler with 4 fixed warm-up epochs and a peak learning rate 2×10^{-4} over at most 30 epochs. We used a virtual batch size of 64 data points per batch by using gradient accumulation. We adopted training tricks including gradient clipping, snapshot, early stopping, stochastic weight averaging, etc. It is worth noticing that small Gaussian noise is added to training data points to improve the robustness of the model.

We trained each of our models on a Linux server using a single NVIDIA GeForce GTX 1080Ti GPU. The maximum GPU RAM usage is no more than 9GB for a properly chosen physical batch size.

B SOUND2SYNTH Plug-In

Figure 8: Screenshot of SOUND2SYNTH plug-in built on Dexed. The part highlighted by the rectangle is the SOUND2SYNTH interface.

Using our SOUND2SYNTH model, we developed and released a plugin based on the Dexed synthesizer. The plug-in first "Ping" the server running the neural network to establish a connection. Then by "Match"ing an input audio file, our SOUND2SYNTH model will automatically calculate the corresponding parameters and assign them back to the synthesizer. The plug-in also supports "Download" to serialize and save preset in human-readable JSON format.

Ethical Statement

There are no ethical issues.

Acknowledgments

We would like to thank Xiaoguang Liu from Beijing DeepMusic Technology Co., Ltd for providing computing resources and managing the project. We would also like to thank Yang Yuan and Jiaye Teng from IIIS, Tsinghua University for supporting this research.

References

[Barkan and Tsiris, 2018] Oren Barkan and David Tsiris. Deep synthesizer parameter estimation. *CoRR*, abs/1812.06349, 2018.

[Burk, 2014] Phil Burk. jsyn. https://github.com/philburk/jsyn, 2014.

[Engel *et al.*, 2020] Jesse H. Engel, Lamtharn Hantrakul, Chenjie Gu, and Adam Roberts. DDSP: differentiable digital signal processing. In *ICLR*, volume abs/2001.04643, 2020.

[Esling *et al.*, 2019] Philippe Esling, Naotake Masuda, Adrien Bardet, Romeo Despres, and Axel Chemla-Romeu-Santos. Universal audio synthesizer control with normalizing flows. *CoRR*, abs/1907.00971, 2019.

[Gauthier, 2013] Pascal Gauthier. dexed. https://github.com/asb2m10/dexed, 2013.

[He *et al.*, 2015] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[Hochreiter and Schmidhuber, 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[Horner *et al.*, 1993] 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.

[Huang *et al.*, 2016] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks, 2016.

[Le Vaillant *et al.*, 2021] Gwendal Le Vaillant, Thierry Dutoit, and Sébastien Dekeyser. Improving synthesizer programming from variational autoencoders latent space. In *DAFx20in21*, 2021.

[LeCun *et al.*, 1998] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

[Loshchilov and Hutter, 2019] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

[Mitchell and Sullivan, 2005] Thomas Mitchell and Charlie Sullivan. Frequency modulation tone matching using a fuzzy clustering evolution strategy. In *AES*, 2005.

[Mitcheltree and Koike, 2021] Christopher Mitcheltree and Hideki Koike. Serumrnn: Step by step audio VST effect programming. *CoRR*, abs/2104.03876, 2021.

[Records, 2014] Xfer Records. Serum: Advanced wavetable synthesizer - xfer records. https://xferrecords.com/products/serum, 2014.

[Roth and Yee-King, 2011] Martin Roth and Matthew Yee-King. A comparison of parametric optimization techniques for musical instrument tone matching. In *AES*, 2011.

[Simonyan and Zisserman, 2015] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *ICLR*, 2015.

[Terasawa *et al.*, 2005] Hiroko Terasawa, Malcolm Slaney, and Jonathan Berger. Perceptual distance in timbre space. In *ICAD*, 01 2005.

[u-he Software Synthesizers and Effects, 2011] u-he Software Synthesizers and Effects. Diva: The spirit of analogue — u-he. https://u-he.com/products/diva, 2011.

[Yee-King *et al.*, 2018] Matthew John Yee-King, Leon Fedden, and Mark d'Inverno. Automatic programming of VST sound synthesizers using deep networks and other techniques. *IEEE Trans. Emerg. Top. Comput. Intell.*, 2(2):150–159, 2018.