Automatic Estimation of FM Synthesis Parameters by Convolutional Neural Network

Sergio Rocha da Silva
sergio.silva16@unifesp.br
Universidade Federal de São Paulo
São José dos Campos, São Paulo, Brasil

Abstract

A clear and well-documented LaTeX document is presented as an article formatted for publication by ACM in a conference proceedings or journal publication. Based on the "acmart" document class, this article presents and explains many of the common variations, as well as many of the formatting elements an author may use in the preparation of the documentation of their work.

CCS Concepts

• **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.

Keywords

Do, Not, Us, This, Code, Put, the, Correct, Terms, for, Your, Paper

ACM Reference Format:
Sergio Rocha da Silva. 2026. Automatic Estimation of FM Synthesis Parameters by Convolutional Neural Network. In *Proceedings of The 41st ACM/SIGAPP Symposium on Applied Computing (SAC'26).* ACM, New York, NY, USA, 10 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

2 Theoretical foundations

2.1 Musical Computation

Musical computation is a broad and interdisciplinary field that encompasses all aspects of musical processing through a computer. It is beyond the scope of this article to provide a comprehensive overview of this field; however, for the purposes of this study, it is important to cover the basic elements of sound and the fundamentals of audio digitization.

2.1.1 Sound elements. Audio is a mechanical compression wave that travels through the air. This type of wave can be converted into an electrical signal through microphones, which consist of an arrangement of coils and magnets. As a wave, its main elements are:

- Pitch: The fundamental frequency of a sound, which determines whether it is low or high. The higher the basis frequency, the higher the pitch.
- Duration: The length of time a sound is sustained and can be heard.
- Loudness (or amplitude): The intensity or energy level of a sound. In terms of a wave, it corresponds to the amplitude of the sound wave.

There are other relevant concepts that could be discussed; however, for the sake of simplicity (and because this work is focuses on Machine Learning), only these will be mentioned.

However, it is important to highlight that although sound can be described in terms of the aspects mentioned above, hearing is mainly a cognitive process. Therefore, sound comparison cannot be fully addressed by mathematical tools, leaving a significant space for perceptual comparison and classification (not to mention aspects related to personal preferences).

2.1.2 Sound timbre. Another important aspect of acoustic sounds is timbre, which can be understood as the "color of sound."

From a mathematical point of view, timbre varies according to the harmonics present in a sound sample, or, in other words, according to the set of oscillations that compose the sound.

In the case of acoustic instruments, what happens is that the air vibrations produced by the instrument are not simple pure sinusoids. In fact, when a string vibrates, or when air resonates inside a tube, many parts of the instrument vibrate simultaneously, and the frequency with which each part vibrates differs according to its composition, position, and other factors.

The result is that each instrument has a unique sound, especially in the case of handcrafted wooden instruments such as violins or cellos.

To obtain a clear visualization of this aspect, it is common to convert a sound sample from the time domain to the frequency domain using a Fourier Transform technique.

For example, consider the two images below, which compare the same violin audio sample in both the time and frequency domains:

s the figures show, there is a main frequency that defines the pitch of the sound, along with many other frequencies that allow the ear to identify the instrument and distinguish between different kinds of sounds.

2.1.3 Audio digitalization. A microphone converts a pressure wave into an electrical waveform. However, in computational music, the audio signal must be discretized to be represented as a digital audio file. The primary process for digital audio representation is sampling.

SAC'26, Thessaloniki, Greece

ACM ISBN 979-X-XXXX-XXXX-X/26/03
https://doi.org/XXXXXXX.XXXXXXX

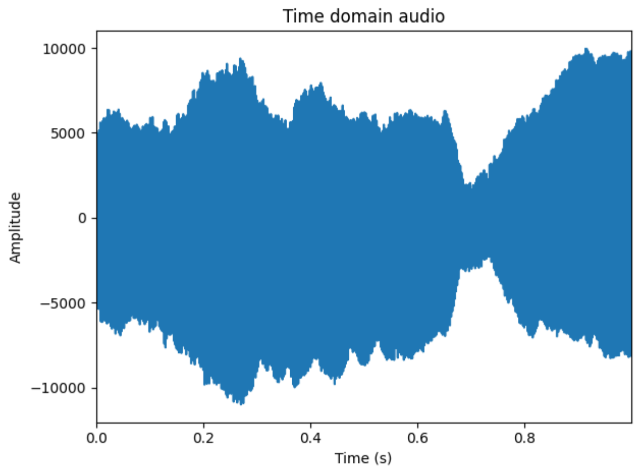

Figure 1: One second time domain violin sample.

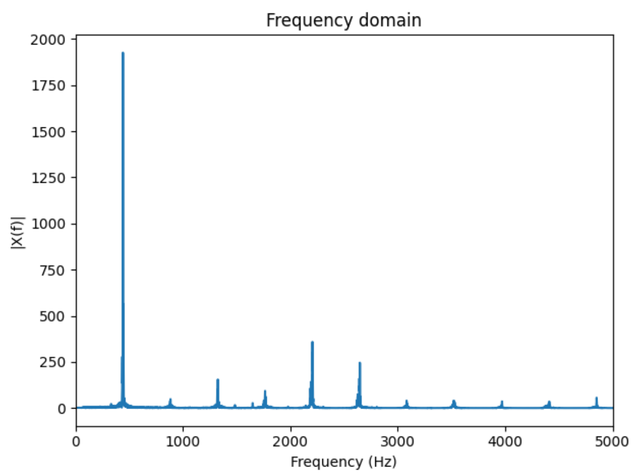

Figure 2: TThe same sample in frequency domain.

Sampling consists of recording a sequence of wave amplitudes at fixed intervals. For example, if a sound is recorded using a sampling rate of 22,100 Hz, it means that the computer will capture 22,100 audio samples per second.

This approach is very efficient because human hearing can detect frequencies between 20 Hz and 20,000 Hz. However, for most musical sounds, the maximum frequency is typically below 10 kHz (in fact, even a violin — one of the highest-pitched instruments — reaches only around 3,500 Hz). And according to the Nyquist-Shannon Theorem, to properly sample a periodic signal, the sampling rate must be at least twice the frequency of the highest component of the signal.

Thus, 22,100 Hz can be a good sampling rate for recording music while saving disk space. However, if the goal is to cover the full range of human hearing, higher rates, such as 44,200 Hz, can also be used.

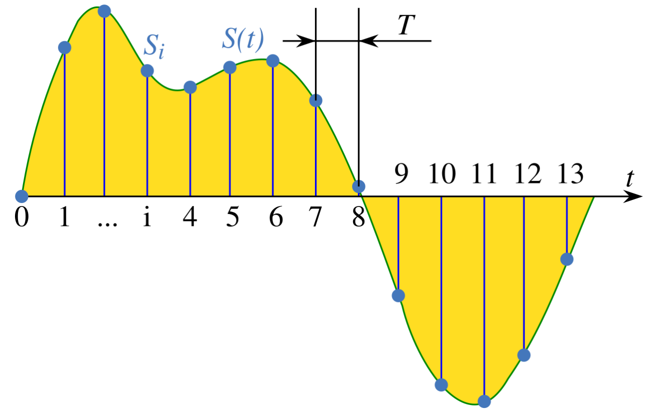

Figure 3: Signal sampling. Source: [1]

Note that with this process, all sound elements highlighted before, can be recorded and reprocedure later.

2.1.4 Sound synthesis. After the development of sound recording techniques, musical computation evolved toward the creation of artificial waveforms. Of course, before digital audio synthesis, there were several techniques that allowed analog synthesis. However, the underlying idea is essentially quite similar.

In summary, audio synthesis techniques can be divided into three basic types:

- Additive synthesis: Perhaps the most straightforward idea from a mathematical point of view. It consists of generating each frequency component of a desired timbre separately and then summing them. In fact, it is more like a weighted sum, since each component is weighted according to its contribution to the timbre. A normalization step is commonly applied after the process to avoid amplitude overload.
- Subtractive synthesis: The opposite of the previous technique, but historically more common due to its effectiveness. In short, it consists of taking a complex waveform, rich in harmonics, and applying various filters to obtain the desired sound. It is more empirical than additive synthesis (which is based on spectral analysis), but simpler to implement and less computationally intensive. Typically, the original waveforms are complex ones such as the sawtooth wave, square wave, or triangular wave.
- FM synthesis: Another empirical approach to sound synthesis, which will be addressed in detail in a later section. In short, it consists of altering a simple waveform by applying a high-frequency modulation, not to the generated waveform itself, but directly to the frequency parameter of the carrier oscillator. This kind of distortion of the base frequency produces mirrored harmonics that enrich the timbre of the sound. It is up to the musician to choose the appropriate parameters to produce the desired timbre.

There are many other techniques that could be mentioned; however, these three represent the most direct forms of synthesis, or the most purely mathematical ones, in the sense that they do not require manipulation of pre-recorded audio.

2.1.5 *ADSR Audio envelop.*

Another important technique for achieving good results is the so-called "Attack, Decay, Sustain, and Release (ADSR) envelope", which consists of applying a multiplicative function to the waveform, altering only its amplitude.

The basic idea is to control the sound intensity by simulating the natural behavior of acoustic sounds. Typically, an ADSR envelope is defined as a function with a domain between 0 and 1, which is applied to the generated waveform:

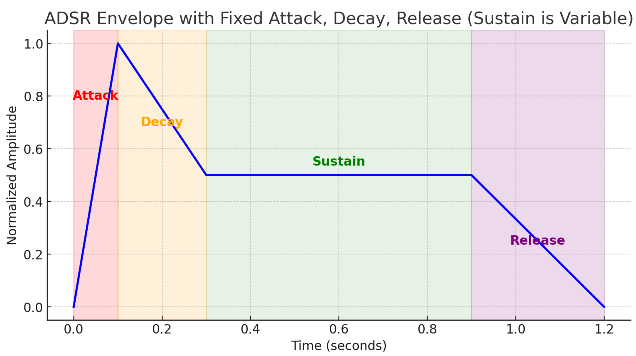

Figure 4: ADSR linear envelop.

The ADSR envelope can be understood as an amplitude function:

$$y(t) = e(t) \cdot s(t)$$

Where:

- $s(t)$: the raw generated sound wave.
- $e(t)$: the ADSR envelope function (normalized amplitude ranging from 0 to 1).
- $y(t)$: the resulting sound.

Although real audio samples can show a wide variation of envelope behaviors, it is common practice to use a linear function to achieve the goal, since the human ear is more sensitive to abrupt changes and to relative differences. However, for longer variations, the ear can still perceive the change, which is why some synthesizers allow the use of exponential or logarithmic functions as well.

2.2 FM Synthesis

As shown in the previous sections, one of the most popular sound synthesis techniques is Frequency Modulation Synthesis (FM Synthesis), which was introduced by John M. Chowning in his famous paper "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation", published at the Stanford Artificial Intelligence Laboratory.

Chowning explored the effects of frequency modulation when applied with high frequencies, close to the audible range, instead of the traditional use of inaudible frequencies.

In fact, the concept of frequency modulation was already well known before this paper; however, it had been used mainly in radio transmission of music or to apply slight distortions through low-frequency oscillators (LFOs), producing effects such as vibrato.

As Chowning himself pointed out in his paper, when the technique is applied to vary the frequency of the carrier wave through a high-frequency modulator wave, it "results in a surprising control of audio spectra" (Chowning).

In summary, considering only two oscillators (the main one, called the carrier, and the modulator) the instantaneous frequency of the output audio can be expressed by the following function:

$$f(t) = f_c + I \cdot \sin(2\pi f_m t)$$

Where:

- $f(t)$: the instantaneous frequency of the generated sound wave.
- f_c: the original frequency of the carrier wave.
- I: the modulation index, which defines the intensity of the modulation (the greater the value, the more harmonics are generated).
- f_m: the frequency of the modulator wave (which determines the spacing of the harmonics).
- t: the time variable.

The most important point, however, is that this simple manipulation results in symmetric sidebands around the carrier frequency, spaced at multiples of f_m.

For a more intuitive understanding, if a carrier frequency f_m is modulated by a frequency f_m, the resulting spectrum will contain:

$$f_c \pm n f_m$$

For example, by choosing $f_c = 440$ and $f_m = 220$, the resulting wave will contain frequency components such as 220 Hz, 440 Hz, 660 Hz, 880 Hz, and so on, with decreasing amplitudes (the amplitudes follow the Bessel coefficients, but this discussion is beyond the scope of this article).

The last important characteristic of Frequency Modulation Synthesis concerns the ratio between the carrier frequency and the modulator frequency f_c/f_m, since this ratio directly influences the perceived audio result.

This ratio determines whether the generated harmonics align with the natural harmonic series (widely used by musicians) or not:

- If the ratio is an integer number, the harmonic sidebands (in the spectral view) will appear as integer multiples of f_c, generating harmonic sounds similar to those of real instruments (consistent with traditional music theory).
- If the ratio is not an integer value, the harmonic sidebands will not align perfectly with f_c, and the generated sound may appear metallic or dissonant (this effect is useful for creating sounds such as bells or for producing special audio effects).

2.3 Audio comparison

The task of audio comparison is not straightforward, especially when the goal is to achieve equality in terms of human perception. It can be complex, computationally demanding, and inherently subjective.

However, as described in the section on timbre, one intuitive and efficient way to approach this problem is by comparing the audio in its spectral representation, since timbre is precisely defined by this combination of frequencies.

That said, this approach alone is not sufficient, because the Fourier Transform considers the entire audio signal at once, mixing different parts of a sound sample. For example, if a sample contains three different pitches in sequence, its frequency-domain representation will show them simultaneously, which may give the impression that the timbre is a mixture of all these pitches (similar to a chord rather than a sequence).

In fact, according to Claesson (2019), FFT-based spectral comparison is valid but only a part of the solution. He highlights the following additional metrics:

- FFT Distance: This consists of computing the Fast Fourier Transform of the two samples being compared and then calculating the normalized Euclidean distance between them (the maximum distance is 1, and the minimum is 0).
- Short Time Fourier Transform (STFT) Distance: Equivalent to the FFT Distance, but calculated over short, overlapping slices of the sample, which provides time-localized spectral information.
- Log-Mel Spectogram Distance: Similar to the previous metrics, but based on the Log-Mel Spectrogram, which is essentially an adaptation of the FFT spectrum that incorporates human auditory perception. In summary, the Log-Mel Spectrogram is obtained by computing the STFT, mapping the frequencies onto the Mel scale (explained later in this document), and finally projecting the amplitudes onto a logarithmic scale.

In addition to these three metrics, Claesson also recommends the use of Euclidean Distance of the Envelope to compare audio envelopes. This approach can be very useful, but since this work is focused on the similarity between original and synthesized audio in terms of timbre, the envelope comparison, while important for real-world applications, falls outside the scope of the present study.

2.3.1 Mel-Scale Representation. The Mel scale was proposed by Stevens, Volkmann, and Newman in 1937 (Claesson, 2019), and its main purpose is to reflect the human perception of frequency.

The key difference from the linear scale is that, as the values on the scale increase, the perceptual difference between two adjacent points becomes smaller (and the corresponding difference in hertz is also reduced).

In fact, the Mel scale maps linear frequencies onto a logarithmic scale. A frequency f s converted to the Mel scale through the following formula:

$$\mathrm{Mel}(f) = 2595 \log_{10}\left(1 + \frac{f}{700}\right)$$

For a better understanding of this conversion, consider the following graph, which compares the linear hertz scale with the Mel scale:

It is a very useful metric for audio comparison, as it takes into account the human auditory perception of frequency.

2.4 Convolutional Neural Networks (CNN)

2.4.1 2D Convolutional Neural Networks. In summary, a convolutional neural network is based on the idea of filters.

Before the introduction of CNNs, was already known that many computer vision problems can be solved through the evaluation of image features, which become more evident depending on the filters applied.

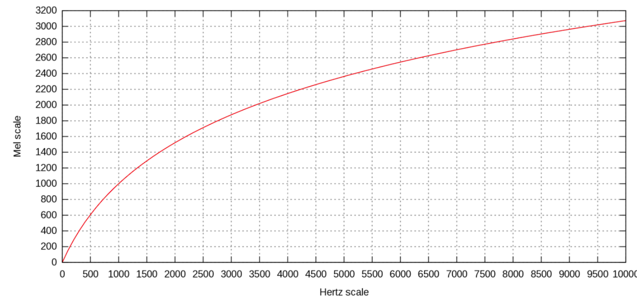

Figure 5: Comparision between linear hertz scale and Mel scale (Claesson, 2019).

For exampe, to classify differe animals, just the silhouettes can be enough. Of course, the fewer features considered, the higher the chance of mistakes, but, as popular saying goes: "if it walks like a duck, quacks like a duck, flies like a duck… it's a duck!". That's the basic idea.

So, the CNNs was concept in such a way that several layers of filters of fixed size (chosen by the network designer) are defined, in addition to some pooling layers (which will be discussed later).

These filter layers are also organized according to the designer's choices (usually based on many experiments) and can be applied in sequence, in parallel, using different combinations, and so on.

However, it is important to note that the filters serve as the link between the different representations of the input image built within the network.

That is, although the network still takes an image as input, and although this image is processed to extract the various features needed for the problem, there are no dense interconnections between these image representations.

In fact, what interposes between one image representation and another is precisely a CNN filter. And even this filter is not densely connected to the image representations.

In practice, the filter is not connected to any specific part of the image. Instead, it is "slid" across the image, performing a Hadamard multiplication (element-wise product) between the filter and each frame of the image (of the same size), producing, as output, a "new filtered image" (multiplication, element by element, of two matrices of the same size, with a final sum of all elements — and, typically, with the application of an activation function to the result).

Visually, the work can be represented as follows:

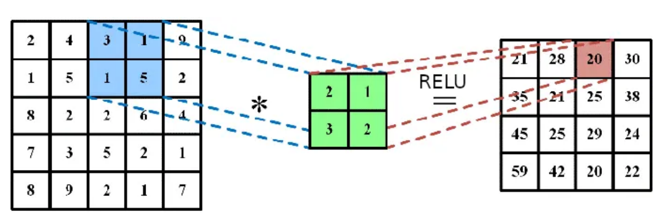

Figure 6: Hadamard product between a filter and an input layer slice (Adapted from ResearchGate, 2024)).

And it is precisely from this sliding operation that the network architecture takes its name, since this action corresponds to the mathematical operation of convolution:

To an inout image I, a filter (or Kernel) K, the convolution output O, in a point (i, j) is given by:

$$O(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} K(m,n) \times I(i+m, j+n)$$

Where:

- $K(m,n)$: the filter element on position (m,n).
- $I(i+m, j+n)$: the element of the input on position of current window.
- M, N: the height and width of the filter.
- $O(i,j)$: the output element on position (i,j).

Thus, CNNs can perform image filtering while at the same time reducing the amount of memory required compared to traditional fully connected feed-forward networks (because memory is allocated only for the filters and their results, rather than for each individual interconnection weight).

After extracting the necessary features from the data, a small traditional dense (multi-layer) network is added after the convolution (and pooling) layers. This dense network is responsible for identifying the patterns required to predict the target variable (whether for classification or regression).

A CNN is typically divided into two stages. The first stage consists of the application of filters (along with pooling layers), while the second stage applies a fully connected network. In this second stage, the output of the first part undergoes a flattening process so that it can be used as input to the fully connected layers, which handle the usual tasks of classification or regression.

3 Related work

4 Proposed approach

The overall idea of the project can be summarized as follows:

Use AI to guide an FM synthesizer in simulating any instrument.

Or, in a more detailed explanation:

Use a Convolutional Neural Network (CNN) to analyze an instrument's sound sample and predict the correct parameters for an FM synthesizer to re-synthesize the "same" sound, enabling the synthesizer to simulate that instrument playing any song.

However, this idea is still quite abstract and broad. Therefore, consider the following steps to achieve the project's goal:

(1) Implement a simple FM synthesizer, which can later be used both to generate the dataset and to test the ability to re-synthesize a sound.
(2) Generate an audio dataset with random audio samples, each created by randomly selecting FM synthesizer parameters, essentially recording tuples of (parameter values; audio sample).
(3) Train a CNN to predict the correct parameters for the same FM synthesizer, given only an audio sample.
(4) Evaluate the model's results using the NSynth dataset to test its generalization capabilities.

Frequency Modulation Synthesis was chosen because it is one of the most common and effective methods for generating audio samples, owing to its simplicity and low computational cost.

This technique became extremely popular in the 1980s, when Yamaha released the Yamaha DX7 synthesizer, and in the 1990s it became a standard for many commercial synthesizers, and it is still used today in the production of electronic music (Claesson, 2019).

However, the effective use of this technique depends on advanced knowledge to select the right parameters for generating good results, which can involve thousands of parameters.

Therefore, the ability to simplify parameter selection constitutes a strong justification for this study, and AI models appear as an affordable way to learn the right patterns for controlling an FM synthesizer.

The basic idea is that a model can be produced to guide an FM synthesizer in mimicking any sound, thereby optimizing the effort to compose electronic music.

Furthermore, in terms of computational cost, using an AI model to predict parameters and guiding an FM synthesizer is much more effective than applying a Generative Neural Network to generate the audio. Additionally, a synthesizer can be best applied in real-time composition and electronic musician performance.

4.1 Training the CNN

As described in the "Theoretical Background" section, the main characteristic of the sound that should be reproduced to achieve the main work goal is the "timbre", which is defined as the combination of frequencies present in the sound.

It is clear that techniques such as filtering, Fourier Transform, and etc, could be used to extract the timbre components from an audio signal, but it is not enough to control an FM Synthesizer. In fact, the FM Synthesis controlling is, normally, an experimental process involving trial and error.

Additionally, as described before, each FM Synthesis operator produces additional frequency components in the signal, and there is no practical method to predict the correct parameter settings, but, as a mathematical function, it is easy to see that this synthesis pattern can be easily approximated using AI models.

Thus, the current work aims to use a CNN model, since it is a very effective method for learning patterns, respecting the internal data structure, and without requiring a lot of preprocessing work. In other words, a CNN can simultaneously learn the time-dependent aspects of an audio signal and can act as a pattern extractor, by applying a sequence of filtering operations, allowing the use of the raw audio format as input.

And so, to train the CNN model, a generated random dataset will be used, which main advantage is that it contains both the audio signal and the expected parameters for generating each sample sound. Therefore, the training process is straightforward and should yield good performance.

The figure below presents the overall scheme for this training process.

4.1.1 Evaluating Results. As introduced in the "Theoretical Background" section, the following audio similarity metrics will be used in this work:

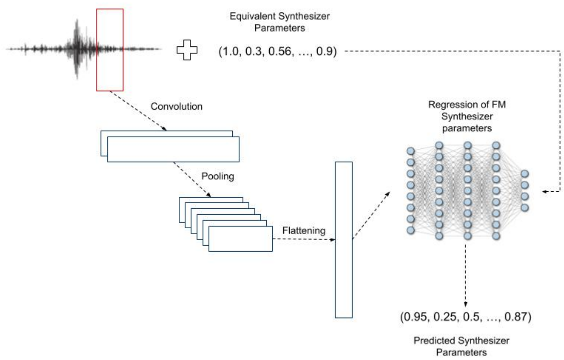

Figure 7: The CNN training scheme.

- FFT distance: Used to compare the frequency components present in an audio signal.
- STFT distance: Used to compare the variation of frequency components over time.
- Log-mel-spectrogram distance: Used for a similar purpose as FFT distance, but typically models human cognitive pitch perception more accurately.

These metrics will be applied to perform a final test of the model's generalization capability.

The proposed approach is, after the model training, to use another external dataset, the Google NSynth dataset, which is better described below, and will be used in order to determine whether the model is able to re-synthesize real instruments, following the steps outlined below:

(1) Train the model with the random generated dataset
(2) Test the model with the NSynth dataset
(3) Re-synthesize the NSynth dataset using the predicted parameters
(4) Calculate the audio similarity metrics
(5) Evaluate the results

5 Experiments

5.1 Setup

5.1.1 Implemented FM Synthesizer. There are several well-known FM synthesizers on the market, but most are not open-source. Additionally, commercial synthesizers often have a vast number of parameters that can be used to control the audio production process. For example, Native Instruments' FM8 can use around 1,000 parameters (Claesson, 2019), and Teenage Engineering's OP-1 synthesizer can be configured with up to 10^{76} parameters (Claesson, 2019).

This incredibly large number of parameters is intended to give users fine-grained control over the generated audio, allowing for a wide variety of pitches (simulating a broad range of instruments and even non-instrument sounds, such as bird songs or car horns, among others).

However, since the scope of this project is not focused on a commercial solution but rather on research purposes, a simple FM synthesizer will suffice.

Therefore, this work will implement a basic FM synthesizer with only 6 operators, inspired by the classic Yamaha DX7, which was one of the most disruptive milestones in the history of electronic music composition. In fact, this simple setup is enough to cover all FM synthesis theories and achieve good performance during AI model evaluation.

In summary, the implemented synthesizer will consist of:

- 5 modulator operators
- 1 carrier operator
- 1 ADSR envelope
- 2 synthesis algorithms (one completely serial, and another with 3 serial operators followed by 2 parallel operators)

The synthesizer will be implemented in Python, using the NumPy library for numerical processing and the SciPy library for audio file generation (in addition to the python-soundfile library).

The figure below presents the FM Synthesizer algorithms:

TODO

To summarize, the implemented synthesizer renders the audio signal according to the following steps:

(1) Generate each operator signal, applying it to vary the phase of the next operator in the sequence (a completely FM synthesis also varies the frequency, but, as the Yamaha DX7, it only varies the phase, for simplicity, and to avoid complex noise treatment). All signals are rendered using a 48 kHz sample rate.
(2) Combine the output of each operator according to the synthesis algorithm (when there are parallel operators, they are summed).
(3) Apply the ADSR envelope to the signal.
(4) Downsample the signal to 16 kHz (applying a low-pass filter to avoid aliasing).

And it is important to highlight that the high sample rate of 48 kHz was chosen to allow the FM synthesis to work well enough, since at high frequencies there is a risk of obtaining aliasing and noise (due to the extremely high frequency components generated by the normal spacing of the FM operators). Furthermore, the low-pass filtering, used to downsample, is also useful to prevent the aliasing problem.

5.1.2 Generating the dataset. This project will use two datasets, each for a specific purpose: the first is the generated dataset, used for training, and the second is the NSynth dataset, used to evaluate the model's generalization capabilities.

One of the most challenging steps in the KDD process is obtaining and processing data. However, for the purposes of this work, an effective approach can be used. Similar to Claesson's approach (Claesson, 2019), since the goal is to achieve good control over the synthesizer parameters, it is sufficient to generate many samples of synthesizer parameters and their respective audio outputs.

Therefore, the main dataset will consist of 5,000 samples generated by the implemented FM synthesizer, with the following characteristics:

- 5,000 samples
- 4 seconds in duration
- Monophonic samples (1-channel audio)

Table 1: Random Uniform ADSR Time Intervals

Parameter	Interval
Attack	[0.005, 0.5]
Decay	[0.02, 0.6]
Sustain	[0.1, 0.9]
Release	[0.05, 0.8]

- 16 kHz audio sample rate (the synthesizer will generate samples at 48 kHz, but they will be downsampled to 16 kHz to be compatible with the NSynth dataset)
- Pitches varying between 20 Hz and 6,000 Hz
- Random ADSR envelopes

The expectation is to achieve a level of learning where the model can accurately predict the parameters for a wide range of sounds that can be generated by the implemented FM synthesizer.

However, as the main purpose is to re-synthesize real instruments, the randomness of this dataset should be controlled to achieve reasonable music property samples, or, in other words, the dataset was constructed respecting the well-known limits and characteristics of real instruments:

- The minimum frequency for the carrier is 20 Hz, and the maximum is 6 kHz (for example, the maximum frequency for a violin is approximately 3 kHz; another important aspect for this maximum choice is that we need to stay under 8 kHz due to the output sample rate, because of the Nyquist frequency). The frequency is chosen by a uniform random distribution.
- The minimum ratio for each modulator is 1/8, and the maximum is 8 (to avoid aliasing in the final signal). Two methods are used to select the ratio: a) with 90% probability, the ratio will be a random uniform choice between discretized values (for example: 1/8, 1/6, ..., 2, 3, 54, ..., 8; the idea is to simulate a real instrument); and b) with 10% probability, the ratio will be a uniform log distribution between 1/8 and 8 (allowing random metallic sounds, like bells, but respecting human auditory perception, where the difference between low frequencies is more perceived than between high frequencies).
- The minimum beta is 0, and the maximum is 8 (also to avoid aliasing in the final signal). The beta is chosen by a uniform random distribution, controlled such that 20% of the values are between 0 and 0.5, 60% are between 0.5 and 3, and 20% are between 3 and 8 (prioritizing rich timbres and avoiding pure sinusoidal sounds, as well as excessive metallic sounds).
- The amplitude is fixed at 1.0, to avoid that different combinations of beta and amplitude generate the same frequency spectrum and the same timbre (prioritizing the CNN learning).

About the ADSR envelop, the following adjustments were made:

With these additional adjustments, the generated dataset will be more representative of simulating real instruments.

5.1.3 External dataset. In addition to the generated dataset, an external dataset called NSynth, from Google's Magenta project, will be used.

This dataset has a similar configuration to the generated one, and its main purpose is to evaluate the generalization capabilities of the model. This is because it contains many real instrument samples, including both acoustic and electronic instruments (as well as some synthetic sounds generated by professional synthesizers), which the model has never seen before.

Only the test set of the NSynth dataset will be used, with the following characteristics:

- 4,096 samples
- 4 seconds in duration
- Monophonic samples (1-channel audio)
- 16 kHz audio sample rate
- 11 instrument families
- Pitches ranging from 27.5 Hz to 4,186.01 Hz
- The envelope includes a sustain of up to 3 seconds and a decay of 1 second

5.2 Training Protocol

5.2.1 Environment Settings. To train the model, the following environment settings were used:

- Python Version: 3.11.2
- Tensorflow Version: 2.19.0
- Keras Version: 3.11.1
- Nvidia-cuda Version: 12.5.82
- Nvidia-cudnn Version: 3.0.75
- Nvidia Driver Version: 535.247.01
- CUDA Version: 12.2
- GPU: NVIDIA GeForce GTX 1650 (4GB memory)

5.2.2 Train/Test Split. As each sample in the generate dataset is a complete sound of 4 secound long (and 64,000 audio samples, wich is 16 kHz multiplied by 4 seconds), a simple train/test split of 75/25 was used, resulting in 3,750 samples for training and 1,250 samples for testing.

But, from these 3,750 samples, 20% of them will be used for validation, resulting in a total of 3,000 samples for training and 750 samples for testing.

5.2.3 CNN Architecture and Hiperparameters. The model architecture used is as following images, where the first cover the features extraction, and the second cover the regression (a common MLP):

The complete model is simple the sum of the two parts. But is important highlight the hyperparameters of each part:

Other important highlights:

- The target variable is normalized to have a mean of 0 and a standard deviation of 1, prior to training.
- The Adamax optimizer was used.
- The Mean Squared Error (MSE) was used as the loss function.
- The recorded metrics were the Mean Absolute Error (MAE), and the Mean Squared Error (MSE).
- The model was trained for 200 epochs, with early stopping enabled using 10 epochs of patience and best weights restoring.
- The model was trained with a batch size of 32.

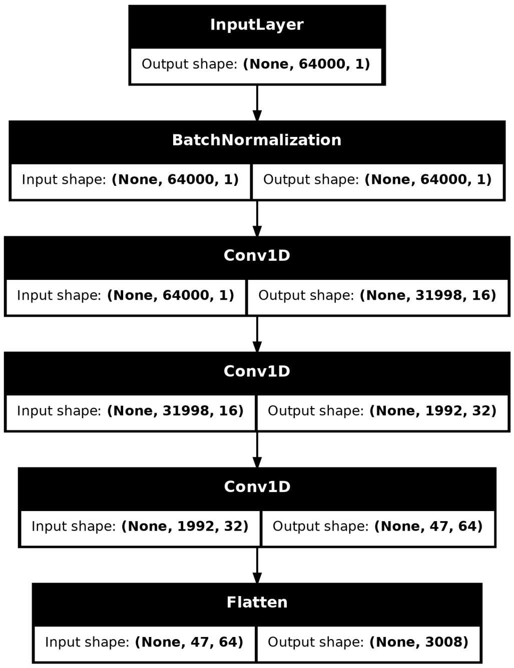

Figure 8: The features extraction part (CNN).

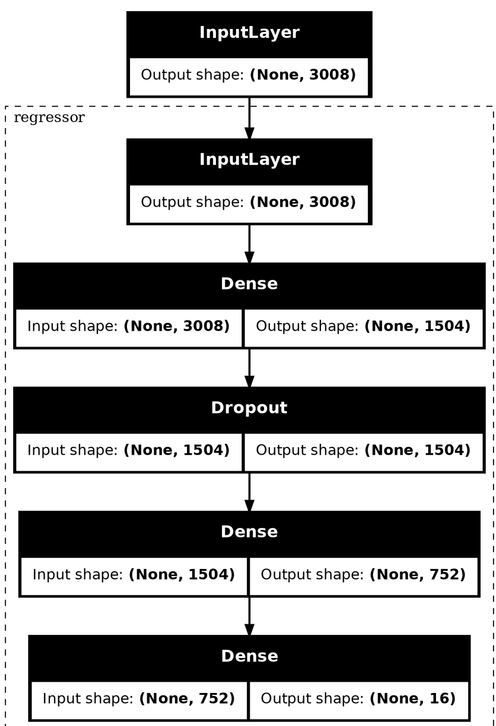

Figure 9: The regressor part (common MLP).

Table 2: Hyperparameters of Features Extraction (CNN part)

Component	Hyperparameter	Value
Input Layer	Input shape	$(input\_len, input\_dims)$
BatchNormalization	–	Applied to input layer
Conv1D (extrator1)	Filters	16
	Kernel size	5
	Strides	2
	Activation	relu
	Regularization	l2
	Use bias	True
Conv1D (extrator1_2)	Filters	32
	Kernel size	128
	Strides	16
	Activation	relu
	Regularization	l2
	Use bias	True
Conv1D (extrator1_3)	Filters	64
	Kernel size	512
	Strides	32
	Activation	relu
	Regularization	l2
	Use bias	True
Flatten	–	Applied after Conv1D layers

Table 3: Hyperparameters of the Regressor (MLP part)

Component	Hyperparameter	Value
Dense (regressor, layer 1)	Units	$input\_dims/2$
	Activation	variable
	Regularization	l2
	Use bias	variable
Dropout	Rate	0.2
Dense (regressor, layer 2)	Units	$input\_dims/4$
	Activation	variable
	Use bias	variable
Dense (regressor output)	Units	$output\_dims$
	Activation	None
	Use bias	variable

5.3 Results

5.3.1 Training/Validation Results. The training and validation results can be viewed in the following:

	loss	mae	mse	val_loss	val_mae	val_mse	epoch
41	0.598830	0.526464	0.545482	0.654662	0.541040	0.601261	41
42	0.592598	0.521328	0.538901	0.657815	0.552085	0.603486	42
43	0.589606	0.518936	0.535034	0.655543	0.534783	0.600631	43
44	0.585538	0.516488	0.529611	0.679045	0.558076	0.621769	44
45	0.581472	0.512927	0.524058	0.674356	0.551178	0.616817	45

Figure 10: Evolution of loss functions over the last five epochs.

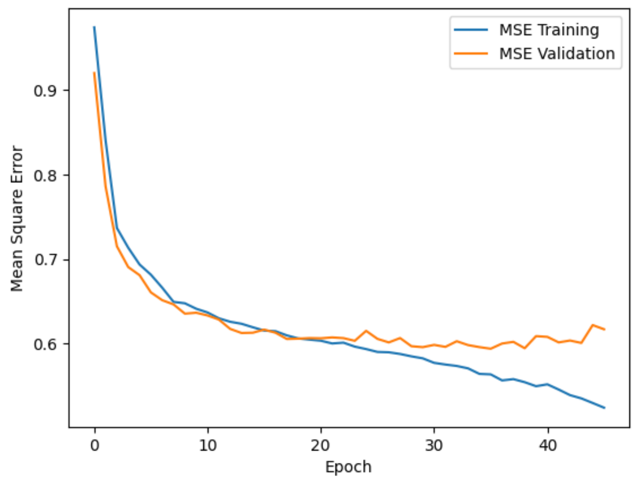

Figure 11: Evolution of MSE during training.

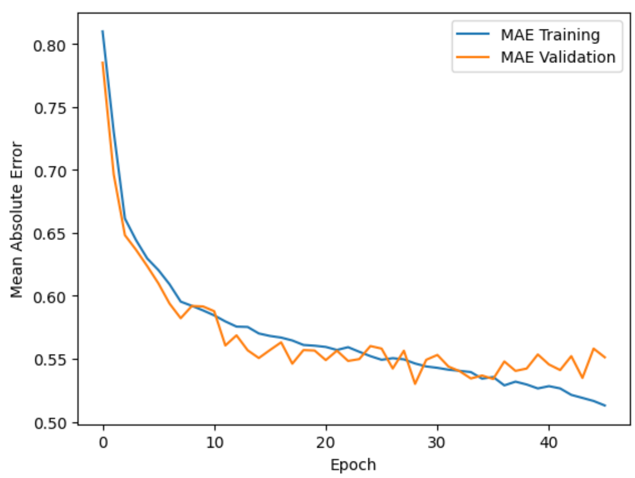

Figure 12: Evolution of MAE during training.

Table 4: Evaluation Metrics on Test Set

Metric	Value
RMSE	38.5391
MSE	1485.2655
MAE	7.6973

5.3.2 Test Results. To compute the test results, the predicted values, from the spplited test portions, was transformed back to the original space, and the following metrics were calculated:

5.3.3 Evaluation Results. As explained in the section about the proposed approach, after the CNN training, to evaluate the model's ability to re-synthesize real instruments, the NSynth dataset was used, in order to calculate specific metrics focused on the capacity of the FM synthesizer to recreate the same timbres.

To achieve this, the model was applied to the NSynth test dataset, and the predicted parameters were used with the implemented FM synthesizer. Then, each signal output was compared with the respective original signal, calculating the proposed metrics:

- FFT distance
- STFT distance
- Log-mel-spectrogram distance

However, to achieve a good evaluation, the proposed metrics were also used to compare each NSynth audio with a baseline, which consists of a pure sinusoidal signal, with a frequency of 440 Hz (i.e., the A4 note, which is used as a reference to tune the majority of instruments).

Then, the results can be seen in the following:

Note that the Log Mel metric was compared, including a normalized version, which is obtained by dividing the raw Log Mel value by the product of the number of Mel bands and the time window quantity.

The normalized Log Mel is proposed because it is better suited for comparing results with other works. For example, authors such as [TODO] achieved good results with values between 0.005 and 0.02, which indicates that the current work result is within a satisfactory range.

Furthermore, to facilitate a better understanding, the metrics were also compared by grouping the dataset by instrument family. This allows for the evaluation of the model's performance on different classes of sounds:

And as can be seen, on average, the top three models performed best for the mallet, keyboard, and guitar families.

Table 5: Comparison of Means (Resynthesis vs Baseline, with Improvement %)

Metric	Resynthesis	Baseline	Improvement (%)
FFT	15640.799	33508.780	53.34
STFT	2772.460	7290.362	61.95
Log Mel (raw)	144.194	184.872	22.00
Log Mel (normalized)	0.00894	0.01146	21.99

Table 6: Comparison of Standard Deviations (Resynthesis vs Baseline)

Metric	Resynthesis	Baseline
FFT	6824.366	1887.364
STFT	1080.760	434.867
Log Mel (raw)	47.726	30.136
Log Mel (normalized)	0.00296	0.00187

Table 7: Comparison of Means by Instrument Group

Group	FFT	STFT	Log Mel (raw)	Log Mel (norm)
bass	13781.79	2513.21	142.53	0.00884
brass	17602.07	3057.35	158.85	0.00985
flute	18812.31	3473.10	166.27	0.01031
guitar	13431.71	2502.01	131.60	0.00816
keyboard	12192.70	2190.55	123.30	0.00765
mallet	9906.05	1969.04	110.33	0.00684
organ	22169.99	3789.86	178.80	0.01109
reed	16668.45	2716.18	135.03	0.00837
string	16223.92	2769.57	155.10	0.00962
vocal	29903.85	4925.21	186.67	0.01157

Table 8: Comparison of Standard Deviations by Instrument Group

Group	FFT	STFT	Log Mel (raw)	Log Mel (norm)
bass	6414.21	846.99	45.92	0.00285
brass	3850.62	726.66	37.93	0.00235
flute	4602.86	965.61	36.67	0.00227
guitar	4892.00	774.12	40.54	0.00251
keyboard	4559.03	643.60	43.21	0.00268
mallet	2914.28	455.63	41.31	0.00256
organ	4588.24	946.80	42.91	0.00266
reed	6217.81	954.98	28.91	0.00179
string	6973.87	1153.48	56.12	0.00348
vocal	6303.27	1402.80	44.51	0.00276

6 Conclusion

7 Future Work

Acknowledgments

This work is supported by *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES), *Conselho Nacional de Desenvolvimento Científico e Tecnológico* (CNPq), grants 406417/2022-9, 102475/2024-5, and 312209/2022-3, and *Fundação de Amparo à Pesquisa do Estado de São Paulo* (FAPESP), grant 2020/09835-1 (CPA IARA).

References

[1] Myriam KRASILCHIK, Celso de Barros Gomes, Francisco Inácio Homem Melo, Heliodoro Teixeira Bastos Filho, Julio Roberto Katinsky, and Tupã Gomes Corrêa. 1996. A USP e sua identidade visual. Tech. rep. São Paulo. http://www.scs.usp.br/identidadevisual/wp-content/uploads/myriamkrasilchik_1996.pdf.