

Variational Parametric Models for Audio Synthesis

Thesis

submitted in partial fulfillment of the requirements
for the degree of

Bachelor & Master of Technology

by

Krishna Subramani

Roll No: 150020008


under the guidance of

Prof. Preeti Rao



Department of Electrical Engineering
Indian Institute of Technology Bombay

2020

Copyright © 2020 by Krishna Subramani
Attribution - NonCommercial 4.0 International (CC BY-NC 4.0) 

Declaration of Authorship

I, **Krishna Subramani** declare that this written submission represents my ideas in my own words and where others ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Krishna Subramani

2020

Abstract

With the advent of data-driven statistical modeling and abundant computing power, researchers are turning increasingly to deep learning for audio synthesis. These methods try to model audio signals directly in the time or frequency domain. In the interest of more flexible control over the generated sound, it could be more useful to work with a parametric representation of the signal which corresponds more directly to the musical attributes such as pitch, dynamics and timbre. These parametric representations also facilitate better musical control of the synthesized output. We present **VaPar Synth** - a Variational Parametric Synthesizer which utilizes a conditional variational autoencoder trained on a suitable parametric representation. We demonstrate our proposed model's capabilities via the reconstruction and generation of instrumental tones with flexible control over their pitch. We also investigate a parametric model for violin tones, in particular the generative modeling of the residual bow noise to make for more natural tone quality. To aid in our analysis, we introduce a dataset of Carnatic Violin Recordings where bow noise is an integral part of the playing style of higher pitched notes in specific gestural contexts. We obtain insights about each of the harmonic and residual components of the signal, as well as their interdependence, via observations on the latent space derived in the course of variational encoding of the spectral envelopes of the sustained sounds.

Index Terms - Generative Models, Conditional VAE, Source-Filter Model, Spectral Modeling Synthesis

Contents

Abstract	i
Contents	ii
List of Figures	iv
1 Introduction	1
1.1 Motivation	1
1.2 Thesis Organization	2
2 Audio Synthesis	3
2.1 Physical Modeling Synthesis	3
2.2 Spectral Modeling Synthesis	4
2.2.1 Sinusoidal Modeling	5
2.2.2 Deterministic plus Residual Modeling	6
2.2.3 Harmonic plus Stochastic Modeling	7
2.3 Generative Audio Synthesis	8
2.3.1 Sequential/Autoregressive Generation	10
2.3.2 Adversarial Generation	11
2.3.3 Framewise Autoencoding	12
2.4 Why Parametric?	13
3 Datasets Employed	14
3.1 NSynth	15
3.2 Good-Sounds	15
3.3 Carnatic Violin Dataset	16
3.3.1 Recording Room	16
3.3.2 Recording Devices	16
3.3.3 Note Recordings	17
3.3.4 Gamaka Recordings	18
4 VaPar Synth	20
4.1 Parametric Model	20
4.1.1 The Source-Filter Model	21
4.2 Generative Model	25
4.2.1 Autoencoders	25
4.2.2 Variational Autoencoders	26
4.2.3 Conditional VAE	27
4.2.4 Hyper-parameter Tuning	27

5	Experiments and Results	29
5.1	Source-Filter Model for Violin Audio	29
5.2	Pitch Conditioned Generative Models	31
5.2.1	Reconstruction	34
5.3	Interdependence of Harmonic, Residual	38
5.4	Generation/Synthesis	42
6	Continuous Pitch Contour Analysis	44
6.1	Pitch Extraction for Carnatic Music	45
6.2	Continuous f_0 Analysis of Harmonic Component	47
6.2.1	Reconstruction	47
6.2.2	Generation/Synthesis	47
6.3	Listening Tests	48
6.3.1	Network Reconstruction	49
6.3.2	Network Generation	50
6.4	Graphical User Interface for Experiments	51
7	Conclusion	52
	List of Publications	54
	Bibliography	55

List of Figures

1.1	Moog Modular Synthesizer	1
1.2	‘Generative Synth’ with controllable parameters	2
2.1	Flow graph for the Karplus Strong algorithm	3
2.2	Analysis - Transformation - Synthesis pipeline	4
2.3	Spectral Peak Picking	5
2.4	Connecting the spectral peaks across time frames	6
2.5	Violin Spectrogram	7
2.6	Sub-sampling the residual spectrum	8
2.7	Generative Models	8
2.8	NSynth Temporal Encoder	11
2.9	Framewise Autoencoder	12
3.1	Recordings per Note	16
3.2	Room Front	16
3.3	Room Rear	16
3.4	Recording Position of Main Mic	17
3.5	Shankarabharanam	19
3.6	Bhairavi	19
4.1	Harmonic plus Residual model	20
4.2	HpR model on a frame	21
4.3	Harmonic Spectral Envelopes	22
4.4	TAE over HpR	24
4.5	Parametric Model for a single frame, Overlap-Add to obtain waveform	24
4.6	General architecture of an autoencoder	25
4.7	Graphical Model showing dependence of X on z	26
4.8	Architecture of a VAE	27
4.9	CVAE, varying β	28
4.10	CVAE ($\beta = 0.1$) vs AE	28
5.1	Harmonic Spectral Envelopes	30
5.2	Harmonic Spectral Envelope from Carnatic Violin Dataset	30
5.3	Residual Spectral Envelope from Carnatic Violin Dataset	31
5.4	Independent Modeling (INet)	32
5.5	Harmonic VAE Latent Space without Pitch conditioning	33
5.6	Harmonic VAE Latent Space with Pitch conditioning	33
5.7	Residual VAE Latent Spaces without and with f_0 conditioning	34
5.8	Harmonic spectral envelope MSE plots	35
5.9	‘Conditional’ AE (CAE)	35

5.10	Flowchart of proposed vs state of art	36
5.11	Input Magnitude (dB) Spectrogram	36
5.12	Reconstructed magnitude (dB) spectrogram with MIDI 63 skipped	37
5.13	Reconstructed magnitude (dB) spectrogram with MIDI 63 included	37
5.14	CVAE MSE with and without Pitch Conditioning	37
5.15	Noise Levels vs Frequency	38
5.16	Harmonic Spectral Envelope for Different Loudness	40
5.17	Residual Spectral Envelope for Different Loudness	40
5.18	Concatenative Modeling (ConcatNet)	41
5.19	Modeling sum and differences (JNet)	41
5.20	Independent Modeling (INet)	41
5.21	Reconstruction MSE	42
5.22	Sampling from a CVAE	43
6.1	Pitch Contour and Discrete Notes	44
6.2	Pitch Contour extracted with the TwM algorithm	45
6.3	Comparison of pYIN and TwM	46
6.4	Comparison of pYIN and P_1	46
6.5	Spectrograms of Input and Reconstructed Audio	47
6.6	Spectrogram of a Generated Vibrato	48
6.7	GUI Snapshot	51
7.1	Accompaniment Generation	52

Chapter 1

Introduction

1.1 Motivation

When you hear about Audio or Music Synthesis, one of the classic synthesizers by Yamaha or Casio, or even Moog like the one shown in [Figure 1.1](#) must be coming to mind. Indeed, audio synthesis is ‘synthesizing’ music by controlling parameters like the pitch (the notes being played), the loudness and the timbre (the instrument being played). In ‘120 Years of Electronic Music’¹, the author defines an ‘Electronic Musical Instrument’ as “instruments that generate sounds from a purely electronic source as opposed to generating them electro-mechanically or electro-acoustically”.



Figure 1.1: Moog Modular Synthesizer²

With the advent of digital computing in the 70s, there was motivation to use the digital computer as a musical instrument, stemming from the realization that all of these operations could be performed on digital computers [1]. Quite a bit of the work in digital music synthesis has been motivated by its earlier analog counterparts. At its heart however, they primarily relied on signal processing methods to model the instruments, namely Physical Modeling and Spectral Modeling Synthesis.

Recently, with the advent of data-driven statistical modeling, and the availability of abundant computing power with GPUs, people have begun using Deep Learning

¹<https://120years.net/>

²https://en.wikipedia.org/wiki/Moog_synthesizer

for audio synthesis, which has aptly been labeled as “Neural Audio Synthesis”. These models primarily rely on the ability of neural networks to extract musically relevant information from tons of available recordings. As opposed to modeling the complicated instrument physics, neural networks can implicitly learn the complex factors underlying the sound. Thus, natural sounds can be generated by the model when trained on a dataset consisting of isolated musical notes being played with different styles and loudness.

With this in mind, our idea of a ‘generative synth’ could be an algorithm which can take as input control parameters and give us as output the audio signal corresponding to those parameters. [Figure 1.2](#) shows such a model.

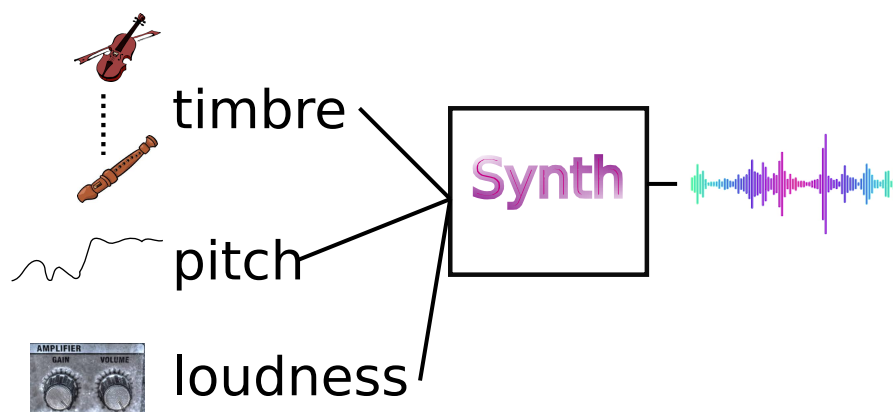


Figure 1.2: ‘Generative Synth’ with controllable parameters

1.2 Thesis Organization

From [Chapter 2](#) of this thesis, we begin by discussing classical methods of audio synthesis, followed by the more modern generative models for audio synthesis. [Chapter 3](#) extensively discusses the datasets we employ for our tasks. It also introduces and describes in detail the Carnatic Violin Dataset we record. [Chapter 4](#) explains the parametric models we employ for audio, and the generative networks we use. [Chapter 5](#) and [Chapter 6](#) elaborate upon the various experiments we perform and the accompanying analysis of the results we obtain.

Parts of this thesis include research work that has either been accepted or submitted to various Conferences. Refer to [List of Publications](#) for more details on the publications.

Chapter 2

Audio Synthesis

2.1 Physical Modeling Synthesis

Physical models try to emulate the process of sound generation by modeling the underlying physics and dynamics of the process. The motivation behind this is the fact that the “source of life in most acoustic instruments is Resonance” [2].

At the heart of simulating any stringed instrument is the wave equation,

$$\mathbf{K} \frac{\partial^2 y}{\partial x^2} = \epsilon \frac{\partial^2 y}{\partial t^2}. \quad (2.1)$$

Where \mathbf{K} is the string tension and ϵ is the string mass density. There are different ways to model the instrument physically [3], two of which are,

1. State Space models.
2. Digital Waveguide models.

Both the methods try to solve Equation 2.1 by converting it into a discrete-time system, and modeling it as a linear system [3].

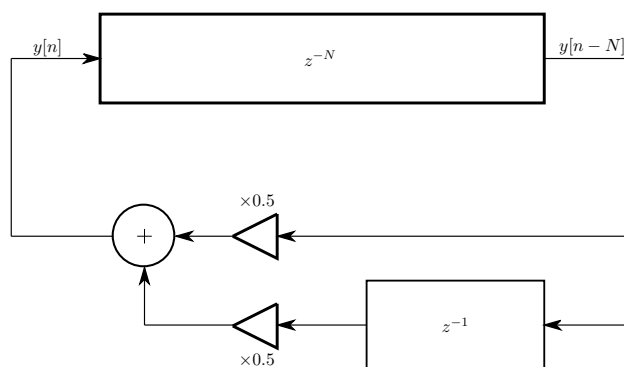


Figure 2.1: Flow graph for the Karplus Strong algorithm

The Karplus Strong algorithm is a precursor to the digital waveguide model which attempts to mimic the sound of plucked string. It does so by filtering a waveform in a feedback fashion as shown in Figure 2.1. The z^{-N} delay also functions as a buffer, and it is initially filled with white noise samples. The transfer function of the model is akin to a feedback IIR filter (or a comb filter) and if the parameters are chosen appropriately, it can model the sound of a plucked string, like a guitar.

The major advantage of the physical modeling technique is the freedom it offers to control any musically relevant aspect of the synthesized audio by modeling the underlying physics into the dynamical system. However, at the same time, you are limited by the accuracy of the model i.e. if the physics is too complicated to model, you might have to simplify it, thus affecting the sound synthesized. Another issue occurs when you have to synthesize in real-time, as the process to simulate the dynamical system is often expensive and needs fast computation.

2.2 Spectral Modeling Synthesis

From the previous section, we saw that physical models are useful to describe sounds whose generation dynamics are known a priori. Moreover, physical models mimic the dynamics of synthesis by behaving like the ‘sound source’, but, from psycho-physics studies¹ of the ear, we see that the ear behaves like a harmonic analyzer. Thus, it makes more sense to construct a model which models the synthesis at the ‘receiver’. This the major motivation to move on to another synthesis technique, namely Spectral Modeling Synthesis (SMS) by Serra et al. [4, 5].

SMS aims to model the spectral content of audio with the aim to obtain “musically useful representations” which make it possible to ‘Transform’ and ‘Reconstruct’ the sound easily, as shown in Figure 2.2,



Figure 2.2: Analysis - Transformation - Synthesis pipeline

where $x(t)$ is the input audio, and $y(t)$ is the transformed audio.

One of the two major underlying assumptions in the Spectral models to be discussed ahead is that the signal can be broken down into a ‘Sinusoidal’ component and a ‘Stochastic’ component as follows,

$$\mathbf{x} = \mathbf{x}_{sine} + \mathbf{x}_{stochastic}, \quad (2.2)$$

where \mathbf{x}_{sine} is the ‘deterministic component of the signal, and $\mathbf{x}_{stochastic}$ is the ‘random’ one. The deterministic part is one that can be written as a sum of sinusoidal signals, and the random part is the one that effectively cannot.

Since we are working with harmonic signals, the Fourier Transform is the first analysis tool that comes to mind. A simple extension to analyze non-stationary signals is the Short Time Fourier Transform (STFT), which assumes that the signal can be broken down into smaller segments where it can be assumed to be stationary. The transformation is,

$$X_l(k) := \sum_{n=0}^{N-1} w(n)x(n + lH)e^{-j\omega_k n}, \quad (2.3)$$

where $x(n)$ is the signal, $w(n)$ is an appropriate windowing function, N is the window size and H is the hop size. N governs the trade-off between the frequency and time resolution and $w(n)$ controls the side-band leakage.

Serra et al. [4, 5] present the following 3 methods of spectral modeling in their work, each of which shall be explored in detail.

¹http://artsites.ucsc.edu/ems/music/tech_background/te-03/teces_03.html

1. Sinusoidal Modeling.
2. Deterministic + Residual Modeling.
3. Deterministic + Stochastic Modeling.

2.2.1 Sinusoidal Modeling

As the name suggests, the signal is modelled as a sum of time varying sinusoidal components,

$$s(t) = \sum_{r=1}^R A_r(t) \cos(\theta_r(t)), \quad (2.4)$$

$$\theta_r(t) = \int_0^t \omega_r(\tau) d\tau + \theta_r(0) + \phi_r, \quad (2.5)$$

where R is the number of sinusoidal components, $A_r(t)$ is the instantaneous amplitude and $\theta_r(t)$ is the instantaneous phase. The analysis is performed as follows,

1. Peak picking - At each time frame, the local maxima in the short-time spectrum of the frame have to be found. To obtain a more accurate estimate for the peak frequency, three point parabolic interpolation is performed [6]. An example is shown below in Figure 2.3.

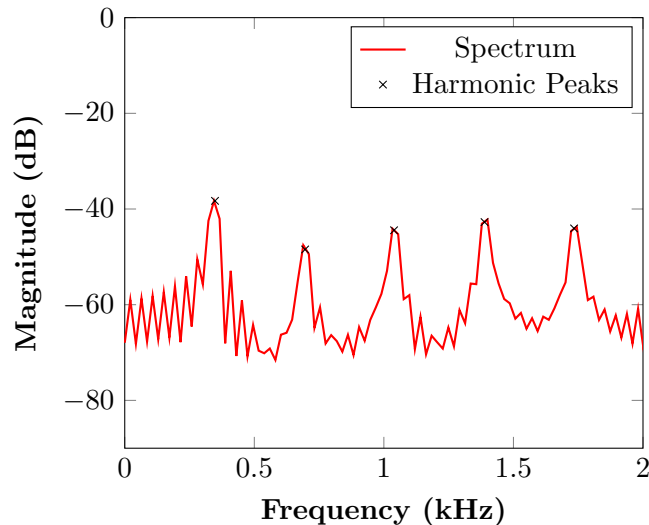


Figure 2.3: Spectral Peak Picking

2. Peak continuation - Once the peaks are obtained at each time frame, they have to be continuously connected to each other. A heuristic approach to connect the peaks is detailed in [4], shown below in Figure 2.4. The intuition is to connect sinusoids that last for at least a minimum duration.

Thus, with the sinusoidal peaks across frames, you can add the constituent sinusoidal signals to obtain the sinusoidal reconstruction of the audio. Issues arise when the signal you try to model cannot be inherently modelled by sums of sinusoids. Consider an extreme example of white noise, which has a flat spectrum. In that case, to capture

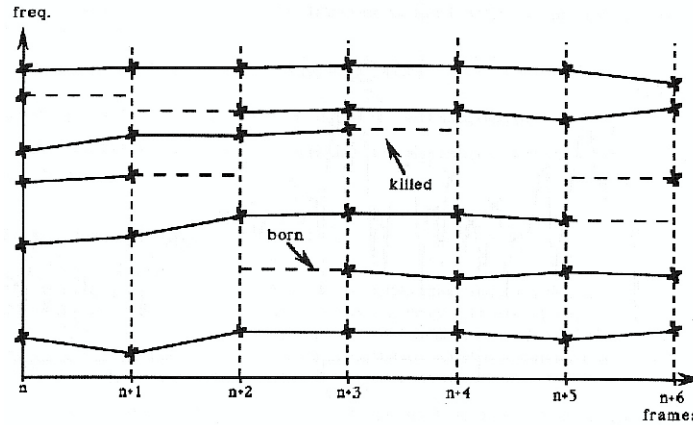


Figure 2.4: Connecting the spectral peaks across time frames (figure taken from [4])

the signal variability, a very large number of sinusoids will be used. In spite of that, the reconstructed signal will sound synthetic and artificial (you cannot expect to represent white noise accurately only with sinusoidal functions). This motivates the introduction of the next spectral modeling scheme, the Deterministic plus Residual Modeling.

2.2.2 Deterministic plus Residual Modeling

As discussed, if the sound is inherently ‘noisy’ (breathy part of speech, ocean sound etc.), then, the sinusoidal model fails to model the sound. Thus the need to explicitly model this ‘noisy’ component. Figure 2.5 shows the example of a violin spectrogram. We can clearly see the harmonics, and can also observe the residual component (as the ‘noisy’ portion of the spectrogram).

In the Deterministic + Residual model (DpR), an additional constraint is enforced on the sinusoidal component. Previously, the sinusoids were allowed to model all the components of the audio signal, but now, they are restricted to model the partials of sound i.e. the harmonics of the sound. This is motivated by looking at the generation of sound by a musical instrument. Most of the energy goes to the modes of vibrations i.e. the harmonics/partial of the sound, and the ‘noisiness’ comes from the excitation mechanism (hammer striking the piano string, bow in the violin etc.) as opposed to the vibration. Since the sinusoids are harmonically related, the method is also called the Harmonic plus Residual (HpR) model, a name we will be using often ahead. The HpR model represents a sound in the following way,

$$s(t) = \sum_{r=1}^R A_r(t) \cos(\theta_r(t)) + e(t), \quad (2.6)$$

where $\theta_r(t)$ is defined as in Equation 2.4 and $e(t)$ is the residual, which is obtained by subtracting the deterministic component from the original signal. The major differences from the Sinusoidal model is the restriction on the sinusoids to model only the partials, which manifests itself in the Peak continuation process where unlike the sinusoidal model, you only want to track partials as opposed to all the components. Serra in [4] presents an algorithm to do so. The output of the algorithm is the clear and stable partials of the sound.

The above approach tries to ensure that only the ‘non-noisy’ (or deterministic) part of the signal is modelled in the reconstructed signal, thus we obtain the ‘residual’ or

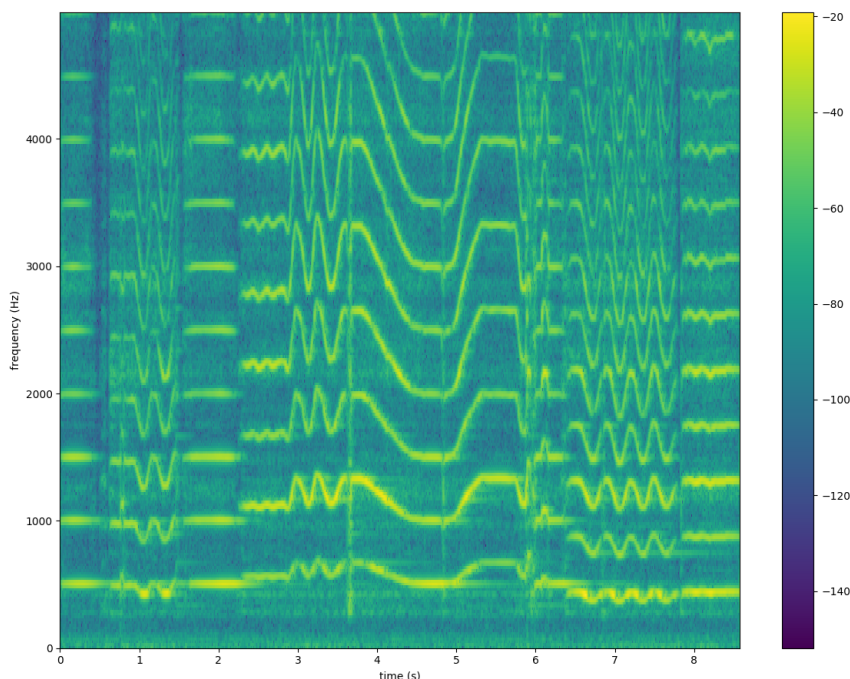


Figure 2.5: Violin Spectrogram

the ‘noisy’ part of the signal by subtracting this reconstructed signal from the original. The HpR model gives you the flexibility to modify individually the deterministic and residual components. The addition of the residual makes the sound seem more ‘natural’ as opposed to explicitly modeling it with sinusoids. However, the residual is still not a ‘flexible’ representation for transformation. This is motivation for the next model, where an additional constraint is placed on the residual - that of being a stochastic signal, which allows it to be more compactly represented.

2.2.3 Harmonic plus Stochastic Modeling

By assuming that the residual is a stochastic signal, we can write it as the output of a linear time-variant system on white noise,

$$\hat{e}(t) = \int_0^t h(t, t - \tau)u(\tau)d\tau, \quad (2.7)$$

where $u(t)$ is white noise and $h(t, \sigma)$ is the impulse response of a slow time-varying filter (impulse response at time t is $h(t, \cdot)$). Instead of preserving the whole residue as it is, you assume it is the output of a filter. Thus, all that needs to be done is determine the coefficients of the filter. The filter representation used by Serra in [4] is simply the sub-sampled version of the residual spectra. To sub-sample is to downsample the residual spectrum, as shown below in Figure 2.6, which acts as the stochastic approximation of the residual spectra. In Chapter 4, we present different method of modeling the residual spectra, which builds on top of this one.

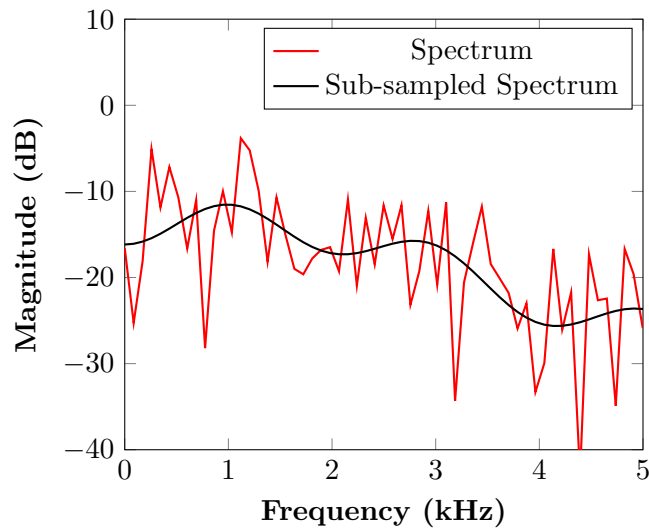


Figure 2.6: Sub-sampling the residual spectrum

2.3 Generative Audio Synthesis

Physical and Spectral Modeling Synthesis are model driven procedures for audio synthesis. “Neural Audio Synthesis” changes the game to that of using data-driven based learning approaches to audio synthesis. How does this work then? These class of learning based approaches try to model the underlying distribution on which the data points lie. Once that is learnt, these models can then ‘sample’ new data points from this distribution. In the context of audio, you can think of these as models from which we can sample and thus ‘generate’ new audio. This is a very difficult thing to do without any prior knowledge of the data distribution. This is where ‘Generative Models’ come in, where neural networks are used to try and approximate the underlying data distribution.

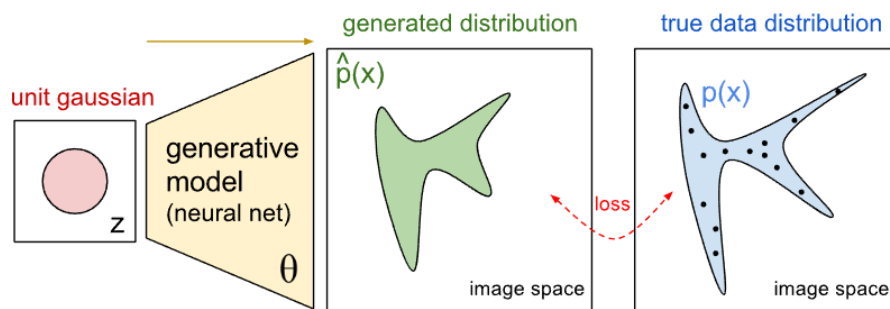
Figure 2.7: Generative Models²

Figure 2.7 tries to explain how generative models work. They assume that the data can be characterized by a unit Gaussian. They then use a neural network to estimate the generated distribution, which they compare with the actual data through a loss function which is minimized via a suitable optimizer (gradient descent, ADAM etc.). The reason for choosing a Gaussian is because points sampled from a Gaussian distribution can be arbitrarily transformed to any distribution with a suitable function [7]. The neural network ‘learns’ that transforming function from the data. The generative model,

²<https://openai.com/blog/generative-models/>

once trained can sample points from the learnt distribution, effectively ‘generating’ new samples.

These models can work on different representations of audio (time/frequency). The commonly used representations are,

1. Time domain waveform : The simplest representation of audio. It is essentially the waveform samples. The advantage is its simplicity and the fact that you don’t have to worry about phase to reconstruct the audio after transformation. However, for coherent waveform synthesis, the model has to take into account information from multiple timescales, which complicates the model architecture required. These complicated architectures require large amounts of data and compute power to train effectively.
2. Time-Frequency representation : As opposed to simply using the time domain samples, one can use a Time-Frequency (TF) representation of the audio signal. It is a 2-D input where one of the axes is time, and the other is frequency. There are various TF representations,
 - (a) Short Time Fourier Transform : The simplest TF representation. Simply involves stacking together the Fourier transform of windowed segments of the audio at increasing time steps together. It was also discussed above in SMS. Its main advantage is its simplicity. However, a major disadvantage is that to perfectly invert the transformation, you need the phase information, which is difficult to estimate after transforming the audio.
 - (b) Mel Frequency Spectrogram : Similar to the Mel Frequency Cepstral Coefficients in speech processing, this involves post processing the Short Time Fourier Transform by filtering it with a melodic filterbank, followed by a non-linear (log) transformation. This is motivated by our perception of different frequencies, and by the logarithmic nature of our hearing. Like the above, the major issue is with the need of the phase spectrum for perfect inversion.
 - (c) Constant-Q Transform : The CQ transform is a filter bank with logarithmically spaced filters and increasing bandwidth. The major advantage of this is that it simultaneously offers better frequency resolution at low frequencies, and better temporal resolution at higher frequency. However, a huge disadvantage is the non-invertibility of the transformation. This has been recently improved by introducing the invertible CQ transform using non-stationary Gabor frames [8].
 - (d) Wavelets : Tzanetakis et al. [9] explored the discrete wavelet features for classification. However, these have not been explored in the context of audio synthesis with deep learning models.
3. Parametric representation - We can use a parametric model for the audio like Physical models or Spectral models. An example is the Sinusoidal model as discussed before in SMS. The advantages of this approach are manifold,
 - A huge reduction in the number of control variables.
 - Better control over audio synthesis if a suitable parametric model is chosen.
 - No issue of phase invertibility like the TF representations. Given the parameters, the audio is easily synthesized.

Keeping the above representations in mind, Generative Models for audio synthesis broadly follow three different methodologies,

1. Sequential/Autoregressive - Generating the time domain waveform samples using either sequential [10] or autoregressive networks [11].
2. Adversarial - Using Generative Adversarial Networks [12] to model either a time-frequency representation of the audio, or the audio waveform directly.
3. Autoencoding - Framewise reconstruction of short-time magnitude spectra using autoencoders [13].

2.3.1 Sequential/Autoregressive Generation

Sequential generation methods choose to work with audio directly in the waveform domain. What they do is try to predict the next sample given the previous samples [14], or try to predict the next frame given the previous sequence of frames [15]. Sequential models use RNNs or LSTMs [10] to predict the samples/frames. Autoregressive networks model the joint probability distribution in an autoregressive fashion i.e. each sample depends on all the samples previous to it,

$$P(\bar{x}) = \prod_{t=1}^T P(x_t | x_{t-1}, x_{t-2} \dots x_1). \quad (2.8)$$

They then use dilated convolutions³ [11] to implement this in an efficient and parallelizable manner. WaveNet [11], which was used to model speech was heavily inspired by previous work by the same authors on image generation, the PixelCNN [16].

The major issue with music is that it has longer temporal range as compared to speech and hence it became necessary to use more information from previous samples to capture long term structure, which increase the network complexity and size, and slows down training. Engel et al. [17] modified the architecture of WaveNet by adding an additional unit, a temporal encoder to take into account more temporal information from past samples, as shown in Figure 2.8

A few disadvantages of autoregressive modeling are,

1. These are big networks, and thus require large amounts of data and resources (GPUs) to train. As mentioned in their own repository⁴,

“Training for both these models is very expensive, and likely difficult for many practical setups. Nevertheless, We’ve included training code for completeness and transparency. The WaveNet model takes around 10 days on 32 K40 gpus (synchronous) to converge at 200k iterations. The baseline model takes about 5 days on 6 K40 gpus (asynchronous).”

One K40 GPU costs $\approx 1000\$$, so that’s 6000\$ just for the baseline (forget the main model, only Google can train it!). This presents a huge roadblock to researchers who do not have access to such computation and resources.

³<https://storage.googleapis.com/deepmind-live-cms/documents/BlogPost-Fig2-Anim-160908-r01.gif>

⁴<https://github.com/magenta/magenta/tree/master/magenta/models/nsynth>

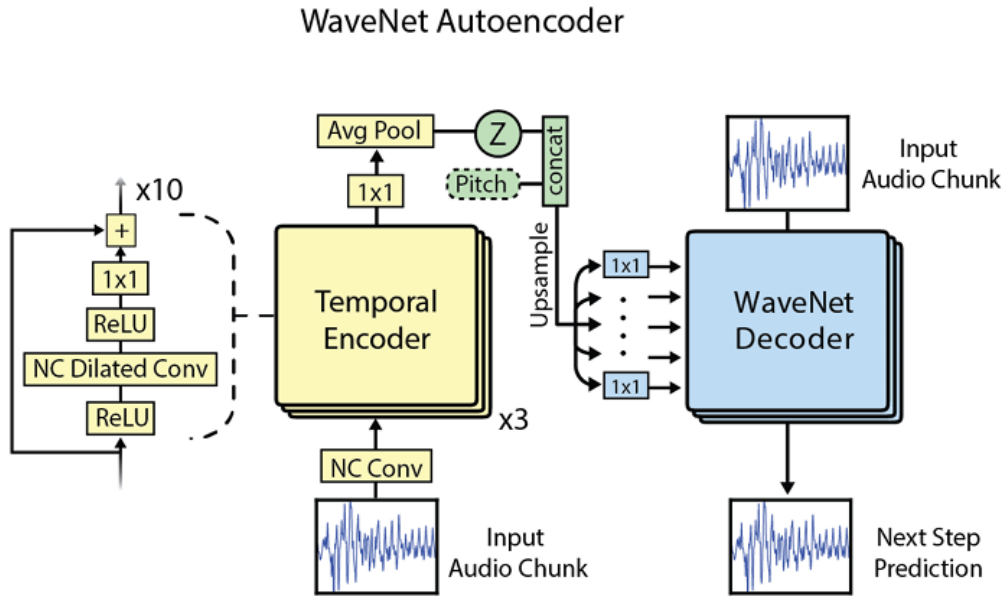


Figure 2.8: NSynth Temporal Encoder, [17]

2. It is not only the training that is expensive, generating waveforms from the above network is also complicated. Generating 1 second of audio requires running the network for almost an hour! Researchers are trying to overcome this difficulty though through network level optimizations, like in Parallel-WaveNet [18].

2.3.2 Adversarial Generation

At the heart of adversarial generation lies the Generative Adversarial Network, more commonly known in research as a GAN [12]. A GAN has two networks, a Generator and a Discriminator. The generator generates samples, and the discriminator tries to classify the sample as generated or actual. The authors in [12] argue that this is a minimax game, and on convergence, the generator should ideally generate samples from the data distribution.

Inspired by the success of GANs in generating highly realistic images [12], Donahue et al. [19] were motivated to use GANs to generate time domain audio by modifying the architecture to generate samples (WaveGAN). They also try to generate audio from TF representations, namely the spectrogram. One of the major issues with the TF representations is the issue of invertibility to obtain the time domain waveform. As you do not preserve the phase information, the generated waveform is noisy. To deal with these, Engel et al. [20] introduce GANSynth, which proposes modifications to the TF representations and the network architecture to synthesize more coherent and less noisy audio.

The previous issue of timescales does not arise here (at least with the TF representations). This is because the whole representation is modelled at once. Thus, at least in principle, all the timescales involved in audio synthesis are taken care off. However, this is at the cost of a lot of complications that arise during training the network. GANs are very unstable to train. Modifications have been proposed to the architecture , inspite

of which they take extremely long to train. The invertibility of the TF representation is an additional challenge to consider.

2.3.3 Framewise Autoencoding

An autoencoder is an artificial neural network, with the added condition that the output is same as the input. By modifying the architecture appropriately such that the network is bottlenecked, the autoencoder obtains a lower dimensional representation of the input, this effectively ‘compressing’ it.

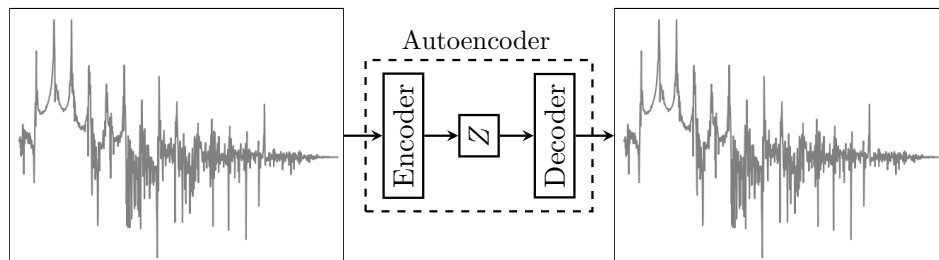


Figure 2.9: Framewise Autoencoder

Figure 2.9 shows the autoencoder for a single spectral frame as the input. The process is repeated across all input spectral frames. During reconstruction, the same issue of phase estimation arises as it did during the adversarial methods. To overcome it, the obtained spectrogram is inverted using Griffin-Lim [21] to estimate the phase spectrogram. Griffin-Lim iteratively estimates the phase spectrogram by beginning with an initial random estimate, and then performing a series of STFTs and inverse STFTs with the phase being updated at each step. With each iteration, the target is to converge to the optimal phase spectrogram.

Saroff et al. [22] were among the first to use autoencoders to perform frame-wise reconstruction of short-time magnitude spectra. They were inspired to model an autoencoder using neural networks because, given ‘enough data’, these networks could learn mappings from higher dimensional spaces to lower dimensional spaces, which could be perceptually relevant to audio synthesis. Their main motivation to use the spectral (FFT) based representation was that it could be inverted back into the time domain (assuming the phase information is preserved). Their investigations revealed a ‘graininess’ in the reconstructed sound, which could be attributed to using the Griffin-Lim algorithm for phase estimation. They also claim that the use of a deep model by simply stacking more layers in the architecture does not necessarily improve the quality of the reconstructed audio, which additionally implies that minimizing the spectral mean squared error might not be optimal for audio perception.

Roche et al. [23] extended this analysis. With the release of NSynth [17], they were able to harness the large number of instrument recordings to experiment with different autoencoder architectures, namely variational and recurrent autoencoders. They experimented with the network parameters for optimal reconstruction of magnitude spectrograms, and also analyzed the so called ‘latent space’ which is essentially a low dimensional representation of the input magnitude spectrogram. With inputs as 512 dimensional magnitude FFT vectors, the latent space dimensions explored were from 4-100. We also explore similar latent space dimensions in our models, which we shall discuss in detail in Chapter 4. The authors also analyze the usability of this latent space in the interpolation of sounds.

One limitation acknowledged by the above authors was the lack of meaningful control over the latent space for use in synthesis. Esling et al. [24] incorporated a regularization term in the VAE latent space in order to effect some control over the perceptual timbre of synthesized instruments. Through their experiments, they were able to ‘generate’ audio by sampling points from this latent space, and could also perform interpolation in this space.

A common drawback of the previous methods is the lack of phase during the reconstruction process, leading to an inherently lossy reconstruction of the audio. Another issue is the frame-wise analysis-synthesis based reconstruction procedure, which does not take into account the temporal evolution of the signal.

2.4 Why Parametric?

One commonality among all the methods discussed so far is that they choose to work with the raw audio, either in the time domain or in the frequency domain. Consider synthesis of a given instrument sound with flexible control over the pitch. Pitch shifting without timbre modification, at least for speech requires the use of a source-filter model with the filter (spectral envelope) being kept constant [25]. The advantage of parametric models are that they do not require us to model the audio waveform or spectrum directly, rather we can work in the reduced parametric space. Combine this with the generative modeling capabilities of a neural network, and you can obtain a powerful audio synthesizer, one that can rely on small, simple network architectures, can be trained with lesser data, and that can potentially generate high quality audio with musically relevant control over it. Recognizing this in the context of speech synthesis, Blaauw et al. [26] used a vocoder representation for speech, and then trained a VAE to model the frame-wise spectral envelope. Engel et al. [27] realized this with their Differential Digital Signal Processing pipeline, which used an autoencoder coupled with the HpR model. However, they do not explicitly considers the modeling of the residual signal.

Julius Smith⁵ compares Physical Modeling to Spectral Modeling. With our survey of generative models, we were inspired to extend the comparison by including generative models.

Physical Modeling	Spectral Modeling
Model only restricted sounds	Model any general sound
Sound is expressive and natural	Sound is not that expressive
Several Equations to solve	Several operations to perform
Represents sound source	Represents sound receiver

Generative Modeling
Depending on available data can model anything
Data-driven
Computationally Intensive
Represents either source or receiver depending on model and data

⁵https://ccrma.stanford.edu/~jos/kna/Projections_Future.html

Chapter 3

Datasets Employed

At the heart of any machine learning based problem lies the dataset. Just like the proverb “You are what you eat”, the equivalent phrase for machine learning would be “Your model will only be as good the data you use in it”¹. Of course, this is only partially true, the other major component of the learning model is the optimizer to consider, that is however a discussion for another thesis, not this one. Collecting and curating data is a challenging task. People in the Computer Vision community have been doing so since a long time, and hence there are many good quality annotated datasets available for using deep learning models. The same cannot be said for audio however.

Considering our task of generation, we would ideally want a dataset that contains recordings of notes being played. However, there are multiple things to take in consideration, such as the quality of the note being played, the loudness of the note, the playing style, the instrument being used etc. Keeping these in mind, we describe three datasets. The first, NSynth [17] is a large repository of instrument note recordings for multiple pitch and loudness values. The second, Good-sounds [28] is also an instrument recording dataset, but with a fewer recordings and instruments taken into consideration.

The third dataset is a custom dataset that we record under a controlled setting. We felt the need to record our own dataset because the previous two datasets were not sufficient keeping in mind our target for controlled synthesis of expressive instrument sounds. We shall now describe each dataset in detail.

But before describing the datasets used, we would first like to talk a bit about the instrument we will be modeling in our work - the *Violin*. The Violin is a popular instrument in Indian music, adopted from the West, due to its human voice-like timbre and ability to produce continuous pitch movements [29]. What makes it a popular choice in Carnatic music (classical music from Southern India) is its ability to produce a continuous pitch variation. This is an important component of the melodic motifs of raga music, that involve changing pitch and dynamics throughout the playing gesture. Consider the task of synthesizing a violin solo for a Carnatic music concert. Let us assume we have with us a dataset with a number of isolated notes at different pitches corresponding to different Carnatic Ragas and played at a set of volume levels. Given this, can we train a system for the synthesis of “natural sounding music” in the same artist’s style given any ‘musical score’ containing the typical continuous gesture motifs?

¹<https://www.capgemini.com/2017/10/quality-data-a-must-have-for-ai/>

3.1 NSynth

Introduced by Engel et al. [17] in 2017, NSynth² (Neural Synth after the paper) is, as of today, possibly the largest dataset for single instrument tones. The authors were motivated to prepare such a large dataset, because, unlike in the case of images, such a curated dataset did not exist for audio tasks.

The dataset consists of 305,979 musical note recordings sampled at $F_s = 16$ kHz, each having a specific pitch, loudness and timbre. The pitch and loudness are specified by their MIDI values. The MIDI pitch is the standard notation used in Western Music. The relation between MIDI pitch p and pitch in Hz f_0 is given by,

$$f_0 = f_{\text{ref}} 2^{(p-69)/12}, \quad (3.1)$$

where f_{ref} is the reference or tuning frequency. From Equation 3.1, MIDI 69 corresponds to f_{ref} . The loudness is specified by the MIDI velocity, which is the force with which the piano key is struck. Both MIDI pitch and velocities are discrete integers which can take values from 0 - 127.

For the NSynth dataset, the reference frequency f_{ref} is 440 Hz (or A4 for the piano aficionados). The MIDI numbers range from 21-108, the range for a standard piano. The MIDI velocities are five discrete values, [25,50,75,100,127]. Each note played is produced by one of three ways - Acoustically (played on the acoustic instrument), Electronically (played on an electronic instrument, like the electronic guitar) or Synthetically (synthesized audio using a MIDI synth). There are 11 instrument classes in total - bass, brass, flute, guitar, keyboard, mallet, organ, reed, string, synth_lead and vocal. Each note is further characterized by a note quality, which describes the note and how it was recorded.

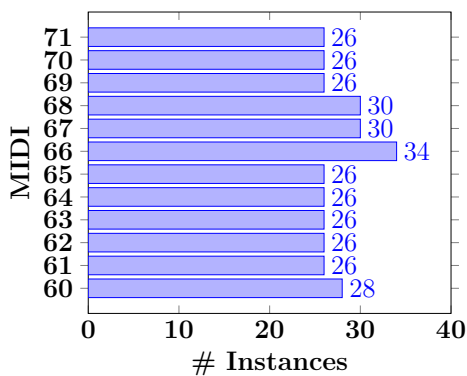
There is a ‘String’ class of instruments. However, the authors have not detailed whether the recordings are solely from the violin, or whether they are from the viola (or possibly cello) as well. Thus, this dataset is insufficient for our final goal of controlled and expressive violin audio synthesis.

3.2 Good-Sounds

As the name suggest, Good-Sounds [28] is literally a recording of notes being played in a ‘good’ manner, with the annotation being done by music teachers. Part of a bigger corpus (which includes bad notes as well). It consists of two kinds of recordings (individual notes and scales) for 12 different instruments sampled at $F_s = 48$ kHz. We select the violin subset of the data. The recordings have been labeled for played as good and bad. We use only ‘good’ recordings, all of which are played in a fixed mezzo-forte (medium) loudness on a single violin. We choose to work with the 4th octave (MIDI 60-71) representing mid-pitch range. The number of notes per MIDI number, and the average duration per note are summarized in Figure 3.1 and Table 3.1 respectively.

The data was collected by recording professional musicians (with a degree in music) playing the notes in isolation. The violin recordings are good, but they are limited in expressivity (fixed loudness and playing style). We perform the initial set of experiments with this dataset, but the lack of expressivity motivated us to record and create our own dataset.

²<https://magenta.tensorflow.org/datasets/nsynth>



MIDI	60	61	62	63	64	65
<i>Duration</i>	4.4	4.7	5.2	4.8	4.9	4.7
MIDI	66	67	68	69	70	71
<i>Duration</i>	4.3	4.5	4.3	5.0	4.5	4.6

Table 3.1: Average note duration (s)

Figure 3.1: Recordings per Note

3.3 Carnatic Violin Dataset

There does not exist a publicly available dataset suitable for synthesis of Carnatic Music, especially for the violin. NSynth [17] is a large musical note recording dataset. Good-sounds [28] is also a similar dataset consisting of musical notes and scales recorded for different instruments. However, both of these dataset work with the MIDI notes and are not that expressive. Keeping in mind our task of expressive synthesis, we would ideally like a dataset which is recorded keeping in mind the Carnatic playing style. We recorded an experienced Carnatic violinist playing a set of Carnatic scale notes at various loudness, playing styles and Ragas.

3.3.1 Recording Room

The dataset was recorded in an anechoic recording chamber as shown in Figure 3.2 (showing the main microphone mounted on the stand) and Figure 3.3.



Figure 3.2: Room Front



Figure 3.3: Room Rear

The blanket was wrapped around the table to minimize any reverberations due to the table itself (as it could not be removed from the room, turns out it was assembled (permanently) inside the room!). Figure 3.4 shows the recording position of the Behringer mic near the bridge of the violin. The Roland recorder was placed on the floor near the violinist.

3.3.2 Recording Devices

Regarding the recording devices, we used the following two -



Figure 3.4: Recording Position of Main Mic

1. Behringer ECM8000³ Ultra Linear Condenser mic - This has a relatively flat frequency response from 5 Hz to 20 kHz, hence useful when recording instruments.
2. Roland R-09HR⁴ High Resolution Recorder - This device affected our recordings by attenuating the frequencies in certain bands, somewhat like a low pass filter with cutoff around 3 kHz. (We do not know why exactly this attenuation occurred, maybe the internal recorder low passed the incoming audio). However, we see opportunity in this! This data can be used to test if models can improve the quality of the recording (some kind of inverse filtering).

To obtain the recordings from the Behringer, we interfaced it with a Laptop using the Focusrite Scarlett USB audio interface. We used Ableton Lite (provided with the Scarlett Device) to obtain the recordings and save them. We saved the recordings as 16 bit WAV files sampled at $F_S = 44.1$ kHz.

The Roland Portable recorder also saved the recordings as 16 bit WAV files at the same sampling rate $F_S = 44.1$ kHz to the on-board SD card, which we transferred to a Laptop later.

3.3.3 Note Recordings

All the recordings are for a fixed tonic E4 (≈ 330 Hz). With the tonic fixed, we record the 12 Carnatic notes Table 3.2 with different recording parameters Table 3.3. An immediate question is whether the note positions are fixed for Carnatic Ragas once the tonic is fixed, or whether there is variability in the note position for different Ragas. Koduri et al. [30] discusses this in detail. There are variations in note intonations across Ragas. However, for a fixed tonic, these variations are not so drastic so as to cause a Ri2 in one raga to be labeled as Ga2 in another one.

Carnatic Note	Sa	Ri ₁	Ri ₂	Ga ₂	Ga ₃	Ma ₁
Notation	Sa	Ri1	Ri2	Ga2	Ga3	Ma1
Carnatic Note	Ma ₂	Pa	Dha ₁	Dha ₂	Ni ₂	Ni ₃
Notation	Ma2	Pa	Dha1	Dha2	Ni2	Ni3

Table 3.2: Carnatic Notes

	Description	Notation
Octave	Lower, Middle, Upper	L, M, U
Loudness	Soft, Loud	So, Lo
Style	Smooth, Attack	Sm, At

Table 3.3: Recording Parameters

1. Octave: We record 3 octaves, around the tonic, one below and one above the tonic octave. The pitch range of the recorded notes is from 115 Hz to ≈ 1100 Hz.

³<https://www.behringer.com/Categories/Behringer/Microphones/Condenser/ECM8000/p/P0118>

⁴<https://www.roland.com/global/products/r-09hr/>

However, it was difficult for the violinist to play the higher frequency notes after Pa in the upper octave (because of the ill defined fingering positions for the notes), hence there are limited recordings of the upper octave notes. The notes have been maintained at the constant pitch by the violinist except in two cases:

- (a) When playing loudly (bowing near the bridge), the notes went off tune sometimes.
- (b) When playing the upper octave, to sustain the notes, the violinist unintentionally had to add some vibrato

The above are rare instances, and have been taken care off during the annotation process by removing the affected pieces.

2. Loudness: To capture dynamics, we have captured two loudness styles, soft and loud. The violinist has tried his best to ‘objectively’ maintain the loudness levels, however during recording it so happens that sometimes he plays certain sections louder than the neighbouring loudness. The violinist has tried to avoid these, but in the event that they occur, they have been labelled as a single loudness only.
3. Style: There are two playing styles the violinist considers. A Smooth one without any attack directly playing the note, and a style where he starts each note with an explicit attack.

The above recordings have been segmented and annotated into individual instances, each containing a single note being played in a single style. The labelling convention followed is: (#ID)-(Note)-(Octave)-(Loudness)-(Style). An example of one such instance is 130-Dha1_M_So_At. The total duration of the note recordings is **1143** seconds across 363 instances.

3.3.4 Gamaka Recordings

Besides the above controlled recording of notes, we have also obtained the recordings of 4 ragas which contain Gamakas -

1. Mayamalavagowla (**MMG**)
2. Bhairavi (**BHA**)
3. Shankarabharanam (**SHB**)
4. Shanmukhapriya (**SHP**)

For each of the Ragas, the Arohanam (**Ar**), Avarohanam (**Av**) and a short snippet (**SS**) of a song in that Raga have been recorded. The short snippets have been decided by the violinist to ‘maximize’ the gamaka recordings. Each of the short snippet is approximately 3.5 minutes in duration. The Snippets contain different Gamakas used in Carnatic Music. They are not of uniform loudness (i.e. they have some variability). Also, the playing style is not consistent (like the previous sustain note recordings). The naming convention is similar to the previous one: (#ID)-(Raga)-(Ar/Av/SS). The total duration of the Raga recordings is **1075** seconds.

From these Raga Recordings, shorter snippets of a few seconds each which contain either one or multiple Gamakas have been extracted. These snippets are useful to test the networks abilities, which will be discussed in detail ahead. The total duration of these short Gamaka snippets is **113** seconds.

Gamakas

Gordon Swift defines Gamakas [31] as “the subtle shadings of a tone, delicate nuances and inflections around a note that please and inspire the listener”. Very loosely, they can be defined as the ‘ornamentation’ or deflections of notes (either around a single note or when transitioning across notes). Gamakas are characteristic in Carnatic Music, and are difficult to describe directly because the large number of Gamaka variations possible. Srikumar [32] describes Gamaka ontologies in terms of their pitch contours (shown in Table 3.4). He has made a “raga-agnostic” description of gamaka contours, and we have tried to find these contours in our own recordings.


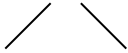



Gamaka	Pitch Contour	Description
Kampitam		Variations around a note (like frequency modulation)
Jaru		Glides from one note to another (like glissandi)
Odukkal		Sharp increase in pitch from a note
Nokku		Sharp decrease in pitch from a note
Kandippu		Descent across 3 notes with a stop at the second note

Table 3.4: Gamaka (with approximate contours and their descriptions)

Here is an example of pitch contours extracted from Raga short snippets with gamaka segments overlaid on top. They have been extracted using the pYIN algorithm (as implemented in Essentia [33]).

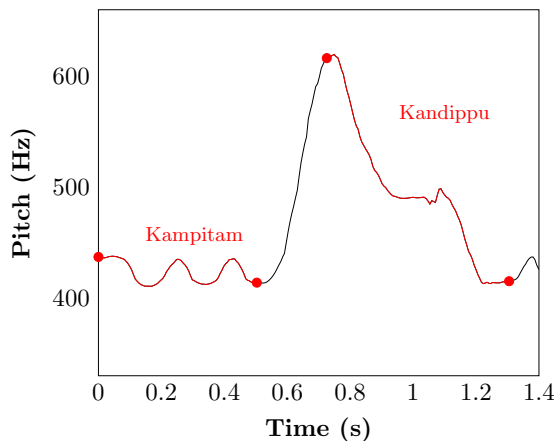


Figure 3.5: Shankarabharanam

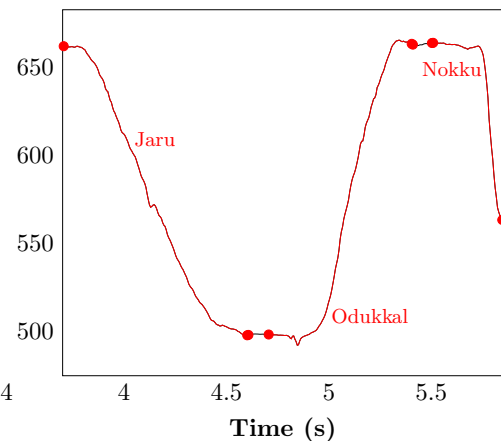


Figure 3.6: Bhairavi

Chapter 4

VaPar Synth

Having discussed parametric models for audio, generative audio synthesis and appropriate datasets to test the system, we now proceed to introduce our parametric synthesizer, the **Variational Parametric Synthesizer** or, **VaPar Synth**. The two main components are the parametric model, and the generative model.

4.1 Parametric Model

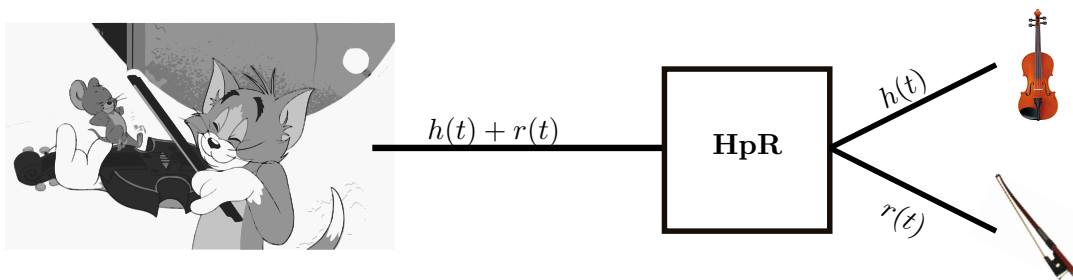


Figure 4.1: Harmonic plus Residual model

Audio can be modelled parametrically in a manner that perceptually relevant parameters become available for musical control over the synthesized sound. A good demonstration of this is the Harmonic plus Residual (HpR) model which we discussed in [Chapter 2](#). Summarized in [Figure 4.1](#), the main idea is to decompose a signal into a sum of sinusoids whose frequencies are integer multiples of a fundamental frequency, and a residual. Consider the audio signal as $s(t)$,

$$s(t) = \sum_{r=1}^R A_r(t) \cos(\theta_r(t)) + r(t) = h(t) + r(t),$$

where the first term $h(t)$ is the harmonic component, and the second term $r(t)$ is the residual. The residual is in essence that part of the audio signal that cannot be represented by a sum of harmonic partials with $R =$ number of partials used. Examples in musical instruments involve the breathy sound when playing the flute and the scratchy sound the bow makes when it moves against the violin string during note sustain regions.

The HpR model acts on a single spectral frame as input. The harmonic component is represented by the harmonic magnitudes, and the residual component is represented

stochastically by the sub-sampled residual spectrum. Figure 4.2 summarizes the HpR model for the harmonic component.

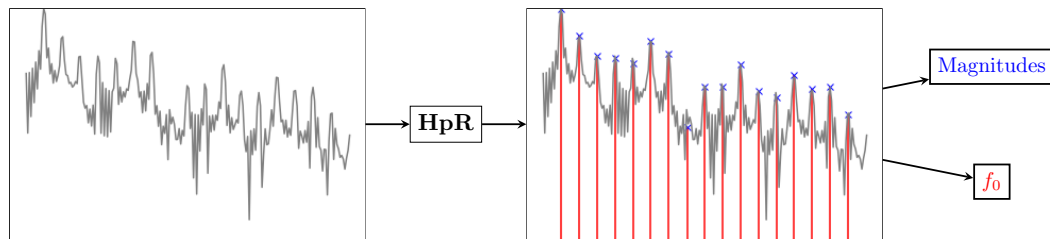


Figure 4.2: HpR model on a frame

We want to build on this by employing a source-filter inspired representation on top of the HpR model. Consider for example the spectrum of a harmonic instrument. Does one really need the information at all the frequencies to represent the audio? Not really. Since the instrument is harmonic, you only need the values of the spectrum at multiples of the fundamental frequency, or the harmonic partials. If we are further able to obtain a relation between the amplitude and frequency (for example in a square wave, the harmonics decay as $\frac{1}{n}$). Thus, for representing the instrument spectrum, the two major things we need are its fundamental frequency and this so called 'relation' between the amplitudes and harmonics. This has already commonly been done in speech processing through what is called the source-filter model.

4.1.1 The Source-Filter Model

Caetano and Rodet [34] suggested applying another model on top of the Harmonic plus Residual model, namely the source-filter (SF) model. This model is usually used for speech analysis as it perfectly captures the speech generation process. In that model, speech is considered to be produced only by two elements :

- **Source:** The device producing the excitation of the whole system, creating a signal with an overall flat and large spectrum. For speech, the source would be the vocal chords. In our model, the source represents the excitation signal, which is generally produced by vibrating air columns (flute) or strings (guitar, violin).
- **Filter:** The element that will shape the signal produced by the source to only keep certain resonances and modify the magnitude of the different spectral peaks. For speech, the filter would be the magnitude responses corresponding to the vocal tract, the mouth, the nasal cavity. In our model, the filter is generally the magnitude response imposed on the excitation signal by the instrument cavity/body.

An important assumption in the Source Filter representation is the independence between the source and filter. This assumption holds quite well for speech signals. The same cannot be said for instruments. There could be coupling/feedback between the source and the filter which causes the filter to drive the source to oscillate at its resonant frequency [35]. However, it can be extended to musical instruments as well under certain assumptions, as discussed in [36]. This leads to a compact representation for the harmonic and residual components of the audio,

- **Harmonic:**
 - Source: Harmonic frequency components

- Filter: Harmonic spectral envelope for each frame
- Residual:
 - Source: White noise
 - Filter: Residual spectral envelope for each frame

One of the crucial tasks in the SF model is the estimation of the spectral envelopes. We shall discuss the algorithm in detail.

True Amplitude Envelope (TAE) Algorithm

Roebel et al. [25] outline a procedure to extract the harmonic envelope using the True Amplitude Envelope (TAE) algorithm. The original paper by Imai in [37] had been published in Japanese only (and no online version available of the original paper), thus we follow the procedure highlighted in [25, 36] to extract the TAE.

Before discussing the TAE, we introduce the idea behind the cepstrum and cepstral liftering. The real valued cepstrum is a homomorphic representation of the audio spectrum [38] defined as,

$$C_p = \text{real}[\text{IFFT}[\log[\text{FFT}(\bar{x})]]] \quad (4.1)$$

where \bar{x} is the audio frame input. It is used to identify pitch periodicity in the audio spectrum. The spectrum is taken into the cepstral domain and only a given number of cepstral coefficients is kept (also known as liftering) as defined in Equation 4.2 with F_s as the sampling rate and f_0 as the pitch.

$$K_{cc} < \frac{F_s}{f_0}. \quad (4.2)$$

As explained in [34], the cepstral liftering should only keep the coefficients allowing spectral variations corresponding to the envelope of the spectrum. With C'_p as the liftered cepstrum (keeping only the first K_{cc} cepstral coefficients and the rest to zero), the spectral envelope is simply obtained as,

$$SE = \text{FFT}[C'_p]. \quad (4.3)$$

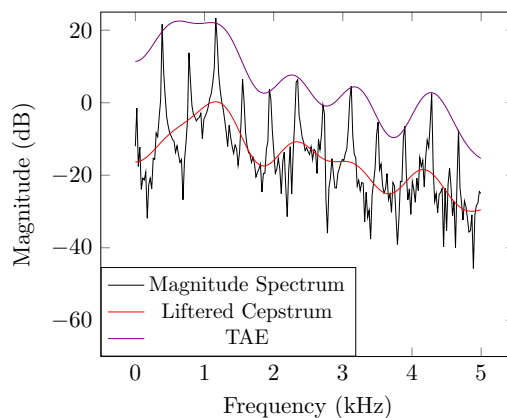


Figure 4.3: Harmonic Spectral Envelopes

Figure 4.3 shows the liftered cepstral envelope. One issue with this method is that the envelope obtained tends to follow the mean energy. However, we are interested in the values of the spectrum at the harmonic peaks. The TAE algorithm addresses this issue by iterative cepstral liftering.

Since the TAE works on top of the HpR model, the input to the TAE model is the sub-sampled harmonic spectrum. The sub-sampling rate corresponds to the fundamental frequency of the frame. The TAE algorithm is presented in [algorithm 1](#).

Algorithm 1: TAE algorithm

```

input :  $X$  - Subsampled magnitude spectrum
output:  $V_f$  - TAE
1 Initialization;
2  $A_0 = \log(X)$ ,  $V_0 = -\infty$ 
3 while  $A_i(k) - A_0(k) < \Delta \forall k \leq N$  do
4   for  $k \leq N$  do
5      $A_i(k) = \max(A_{i-1}(k), V_{i-1}(k))$ ;
6   end
7   Liftering the cepstrum;
8    $C_p = \text{real}[\text{IFFT}(A_i)]$ ;
9   for  $k \leq N$  do
10    if  $k \leq K_{cc}$  then
11       $C'_p(k) = C_p(k)$ ;
12    end
13    else
14       $C'_p(k) = 0$ ;
15    end
16  end
17   $V_i = \text{real}[\text{FFT}(C'_p)]$ ;
18 end
19  $V_f \leftarrow V_i$ ;

```

The algorithm principle is rather straightforward. Let A_i and V_i be respectively the envelope and the liftered cepstrum represented in the spectral domain at iteration i , with A_0 being the log magnitude spectrum of the original sound and $V_0(k) = -\infty, \forall k$. Then, at each iteration, A_i is taken to the cepstral domain, liftered, and then taken back to the spectral domain. If the algorithm were to stop here after a single iteration, it would be a simple cepstral liftering. However, the drawback of the cepstrum liftering is that energy is lost when removing the higher components, which results in an envelope following the mean energy instead of the peaks. This is taken care of by the TAE algorithm by applying the following condition at each iteration:

$$A_i(k) = \max(A_{i-1}(k), V_{i-1}(k)) \quad (4.4)$$

The algorithm then stops when the original spectrum is nowhere larger than the envelope by a threshold Δ . The iterative process pushes the envelope towards the peaks by filling the valleys of the spectrum. Figure 4.3 shows the TAE overlaid on the harmonic spectrum. On comparison with the liftered cepstrum in the same plot, the differences can clearly be seen. The liftered cepstrum follows the mean energy, whereas the TAE

passes through the peaks of the magnitude spectrum. Figure 4.4 summarizes the TAE algorithm as applied over the HpR algorithm.

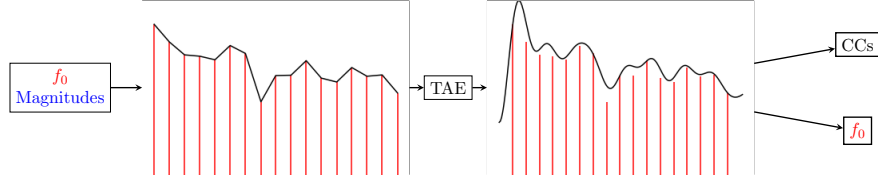


Figure 4.4: TAE over HpR

Figure 4.5 summarizes the parametric representation of violin audio that we employ. It is a source-filter inspired representation that builds on top of the HpR model [36, 34]. All the blocks mentioned are performed on spectral frames extracted from the sustain portions of single note recordings by applying energy thresholds.

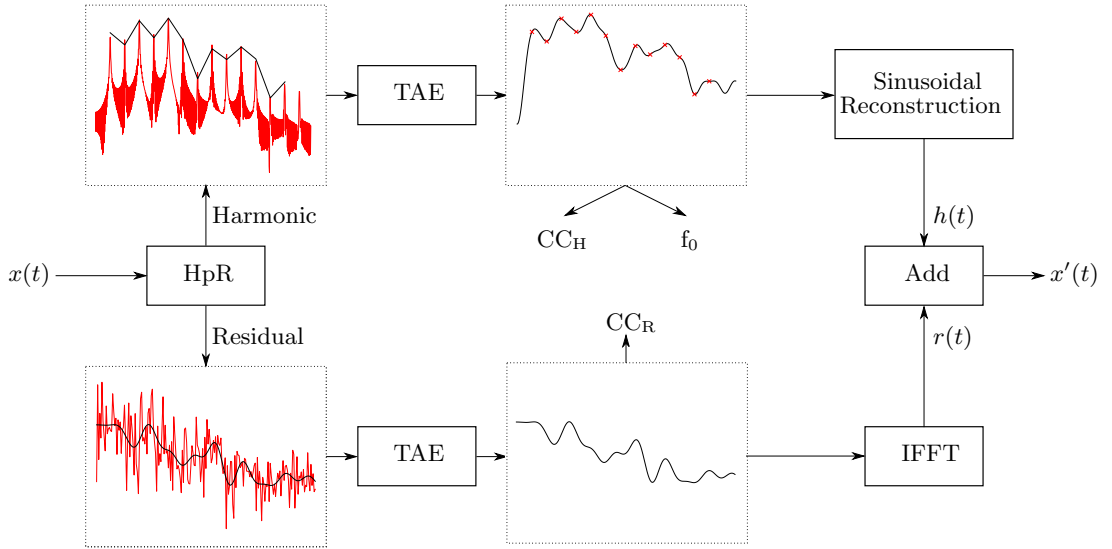


Figure 4.5: Parametric Model for a single frame, Overlap-Add to obtain waveform

1. We run the HpR model [5] on each spectral frame.
2. We sub-sample the obtained Harmonic and Residual Spectra. For the Harmonic, we only keep the amplitude peaks corresponding to the harmonic locations (equivalent to sampling the harmonic spectrum at its fundamental frequency). For the residual, we simply downsample the original spectra to a chosen fixed frequency interval. A residual sub-sampling rate of 100 Hz is mentioned for a sinusoidal representation of speech in [39]. We use a higher sub-sampling rate of 430 Hz, which is one of the values used by Serra in SMS-Tools [5, 4].
3. With the sub-sampled spectra, we use the TAE Algorithm to obtain a smooth spectral envelope for each of the harmonic and residual components. The spectral envelopes are represented by their cepstral coefficients. The harmonic is also additionally characterized by the fundamental frequency of the frame f_0 .

4. To reconstruct the harmonic portion, the sinusoid amplitudes are sampled from the harmonic locations of the TAE, and a sinusoidal reconstruction is performed. For the residual, we simply perform the inverse FFT of the residual spectrum with random phases. The net reconstruction is the sum of the two.

4.2 Generative Model

Having discussed the parametric representation of audio we employ, we now move onto the generative model - the Variational Autoencoder. We begin our discussion by introducing and providing a theoretical framework for autoencoders and their variational counterpart. We then discuss our network architectures, and how we obtain the optimal values of hyperparameters for the networks.

4.2.1 Autoencoders

Having briefly introduced autoencoders (AE), we now explore them in more detail. A rather extensive comparison of several AE network structures can be found in [23]. The general structure of such a network is represented Figure 4.6.

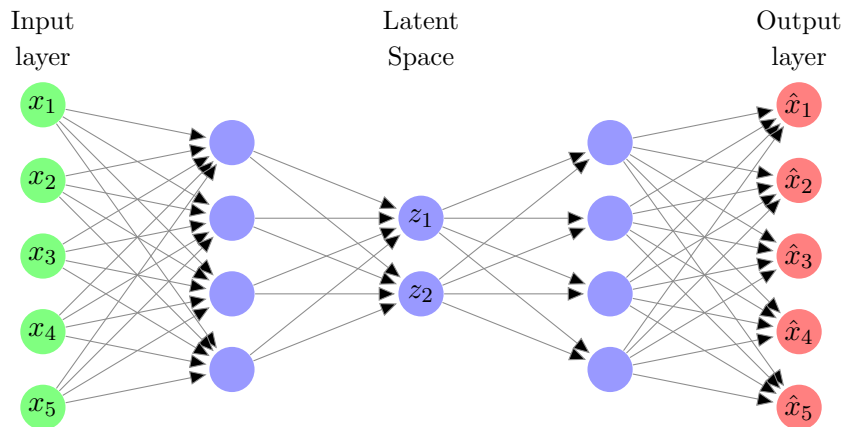


Figure 4.6: General architecture of an autoencoder

The network’s output and input should be the same. The network thus minimizes the Mean Squared Error (MSE) between the input and the network’s reconstruction (or its output) given by,

$$\mathcal{L} = |X - \hat{X}|^2 \quad (4.5)$$

with X as the input and \hat{X} is the network’s reconstruction of the input. Because of its bottlenecked shape, the AE is forced to learn a compact (lower dimensional) representation of the data. This lower dimensional representation is also called the ‘latent space’. What we would ideally like is for the AE to learn a latent space over which we can exercise some kind of control (useful for controlled synthesis of audio). However, the function being optimized in Equation 4.5 does not enforce any additional constraints. Very often, the latent space is sparse and only consists of certain regions where the data likely resides.

4.2.2 Variational Autoencoders

The Variational Autoencoder (VAE) is loosely based on an AE. However, the working principle and optimization criteria are different. VAEs are inspired from Variational Inference [40], which has its roots in Probabilistic Graphical Models. Figure 4.7 shows a graphical model highlighting the dependence of the data X on an underlying latent variable z .



Figure 4.7: Graphical Model showing dependence of X on z

From Figure 4.7, we can write the following,

$$P(X, z) = P(X|z)P(z),$$

However, we are interested in how the latent variable z depends on X . For this, we have to compute the posterior distribution,

$$P(z|X) = \frac{P(X|z)P(z)}{P(X)},$$

and to obtain $P(X)$, the following integral has to be computed:

$$P(X) = \int_z P(X|z)P(z)dz,$$

which becomes intractable for high dimensional z . Variational inference approximates $P(z|X)$ with another distribution $Q(z|X)$. To ensure that the approximation is good enough, you minimize the KL-Divergence between the two distributions,

$$\text{KL}[Q(z|X, \theta)||P(z|X, \theta)] = \mathbb{E}_{z \sim Q}\{\log Q(z|X, \theta) - \log P(z|X, \theta)\}, \quad (4.6)$$

where θ indicates that the distribution is parameterized by θ and \mathbb{E} is the expectation operator under Q . We drop the θ afterwards to lighten the expressions. Because a sufficiently general function can map a Gaussian distribution to any desired distribution [7], the prior on the latent variable z is chosen to be Normally distributed. Equation 4.6 can be rewritten using Bayes rule, and the terms can be rearranged to obtain:

$$\log P(X) - \text{KL}[Q(z|X)||P(z|X)] = \mathbb{E}_{z \sim Q}\{\log P(X|z)\} - \text{KL}[Q(z|X)||P(z)]. \quad (4.7)$$

We want to maximize the left side of that equation as we want to maximize $\log P(X)$. If $Q(z|X)$ is a good approximation of $P(z|X)$, the KL-divergence will be 0 and we will then actually maximize the log-likelihood. To make all terms of the equation above tractable, $Q(z|X)$ is usually chosen to be a Gaussian distribution whose parameters are learned from X by the encoder. The right-hand side can then be maximized using stochastic gradient descent. Figure 4.8 summarizes the VAE architecture. As described above, the encoder represents $P(z|X)$, and learns the Gaussian distribution parameters $\mu(X)$ and $\Sigma(X)$ from the data itself. A point ϵ is sampled from $\mathcal{N}(0, \mathbb{I})$, and transformed by the parameters in the following way,

$$z \leftarrow \Sigma(X)\epsilon + \mu(X). \quad (4.8)$$

This is then passed through the decoder, which represents $Q(X|z)$ to give the reconstruction X' . Sometimes, instead of using the optimization objective as mentioned in Equation 4.7, it is useful to weigh the terms relative to each other by multiplying the KL Divergence term with a weighting factor,

$$\log P(X) - KL[Q(z|X)||P(z|X)] = \mathbb{E}_{z \sim Q} \{ \log P(X|z) \} - \beta KL[Q(z|X)||P(z)]. \quad (4.9)$$

This leads to what is called a β VAE [41]. If $\beta = 1$, it reduces to a normal VAE. Extremely small β gives more importance to the MSE term, thus prioritizing perfect reconstruction (making the VAE behave more like an AE). Similarly, high β strongly enforces the prior to be Gaussian. A more complete and mathematically rigorous analysis of VAEs can be found in [7, 40].

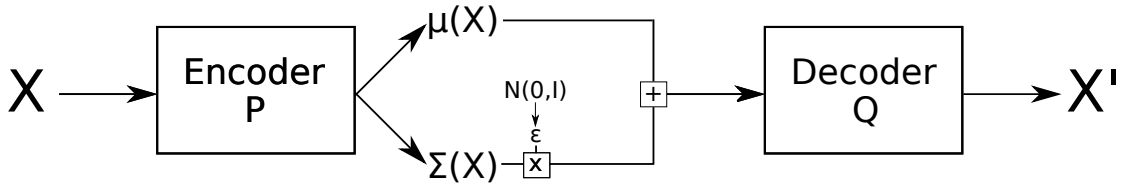


Figure 4.8: Architecture of a VAE

4.2.3 Conditional VAE

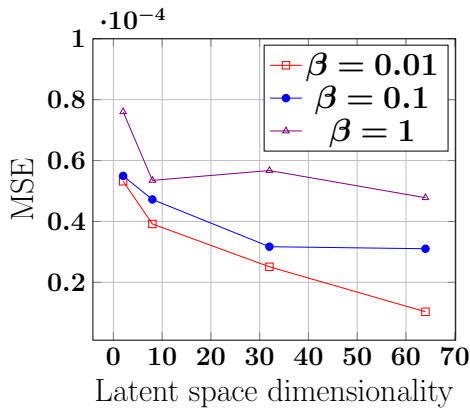
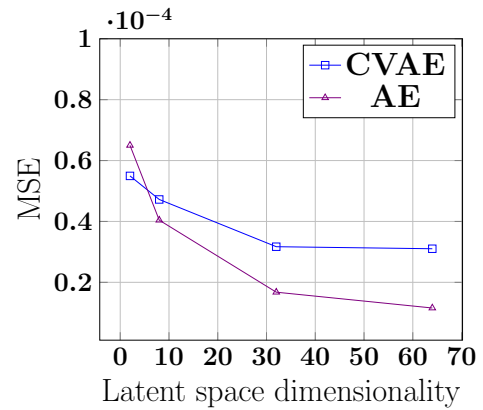
Getting new signals similar to the training data requires sampling from the trained VAE. The procedure is actually rather straightforward because of the Gaussianity imposed on the latent space. Indeed, creating new outputs simply requires sampling from $\mathcal{N}(0, \mathbb{I})$ and passing those samples through the decoder.

However, VAEs do not allow much control over the sampling procedure. Unless we have access to a representation of the latent space, we are always going to use a random z that will produce an output ‘similar’ to the training data. Say for example, our input data is multimodal, and we want to generate samples corresponding to a particular mode. This will not be possible until we know what part of the latent space generates which mode (ideally though, the VAE as presented should not be used for multimodal data, as the Gaussian prior is unimodal in itself!). To address that issue, we will use a slightly modified version of the VAE called a Conditional VAE (CVAE). It works by conditioning the generative process on an external variable [42, 7]. Thus, the only difference in the equations is that the distributions involved in the optimization are replaced by the conditional distributions.

4.2.4 Hyper-parameter Tuning

The main hyperparameters in our networks are the dimensionality of the latent space and the value of β . To decide these, we train the network on the Good-Sounds dataset. We split the dataset into an 80-20 split and label the 80 as train and 20 as test (the two are exclusive). We train the networks on train data instances with different hyperparameters, and evaluate the networks MSE with the test instances. The MSE reported here is the average reconstruction error across all the test instances.

We first find the optimal value for β . We vary β to be 0.01, 0.1 and 1. Figure 4.9 shows the MSE for the various β values for a CVAE. We observe that $\beta = 1$, the MSE is high (relative to the others). From Equation 4.9, we see that high β forces gaussianity

Figure 4.9: CVAE, varying β Figure 4.10: CVAE ($\beta = 0.1$) vs AE

at the expense of MSE, and low β forces the network to behave like an autoencoder. To choose between the other two values, we observed that for $\beta = 0.01$, the network starts behaving like an autoencoder (by giving lesser weight to the latent space regularization). Thus, from these plots, we choose to work with $\beta = 0.1$ for all our experiments. After fixing β to 0.1, the next decision is the latent space dimensionality. To decide this, we experiment 4 different values - 2,8,32,64 for the AE, and CVAE. Figure 4.10 shows the MSE plots for this. As one would expect, increasing the dimension of the latent space shows a decreasing trend in the MSE. Also, the MSE for the AE is lower than CVAE. This is also expected as the AE is trained to minimize the MSE unlike the CVAE, which also has a regularization term which enforces gaussianity on the latent space. The MSE decreases steeply till the dimension is 32, and then the decrease is less gradual from 32 to 64. Thus, we choose to work with a latent space of dimension 32.

All networks are implemented in PyTorch [43]. The encoders and decoders are neural networks with linear fully connected layers and use leaky ReLU activations to allow for stable training. The optimization was performed using ADAM [44] with an initial learning rate of 10^{-3} , and training was run for 2000 epochs with a batch size of 512 on an NVIDIA GeForce GTX 1070 Mobile GPU.

Having provided the details on the parametric model we choose and the generative model along with it, we move on to discuss the various experiments we perform.

Chapter 5

Experiments and Results

Having discussed our choice of parametric representation and the Generative Models we employ, we now ask the question - How applicable are our choices for the synthesis of Violin Audio? Through the experiments we will be performing, we aim to tackle the following questions,

1. How applicable is our parametric representation to model violin audio
2. Why do we use a Conditional VAE over other similar architectures like the VAE or AE
3. Incorporating the Residual modeling coherently with the Harmonic Modeling

5.1 Source-Filter Model for Violin Audio

Beauchamp [45] discusses the applicability of the Source-Filter (SF) model to violin audio. Unlike speech, the SF model has not been used widely to model musical instruments because of the possible coupling between the source and filter in instruments [35]. For the violin however, string vibrations are (largely) independent of the body resonances, thus the independence assumption in the SF model is considered to hold [45, 46].

To check the applicability of the SF model for the harmonic component of violin audio, we plot the filter (the spectral envelopes) for different fundamental frequencies (sources). If the Filter is indeed independent of the Source, then we should not be seeing any considerable change in the spectral envelope shape for different pitches. Figure 5.1 shows spectral envelopes for different MIDI notes from the Good-Sounds dataset. The plot indicates clear differences in the spectral envelope shape across the range of pitches shown. A cursory glance at the plots suggest that the SF model should not hold for violin audio. However, Beauchamp [45] has made an interesting observation about the violin filter. What makes the filter challenging to model is the observation that violin resonances are found to be much sharper (narrower) than those of voice.

In our process of estimation of the spectral envelope, we sub-sample the spectrum at the fundamental frequency and its multiples. For lower fundamental frequencies, we acquire more samples than for higher fundamental frequencies. This might lead to the indirect dependence of the filter on the source fundamental frequency f_0 because of the f_0 dependent sampling of the filter spectral envelope.

The consequence of narrow filter resonances in the violin is that even for a slight change in f_0 , the relative amplitudes of the adjacent harmonics can change quite drastically, which might affect the sound timbre drastically. This has been noted by Beauchamp

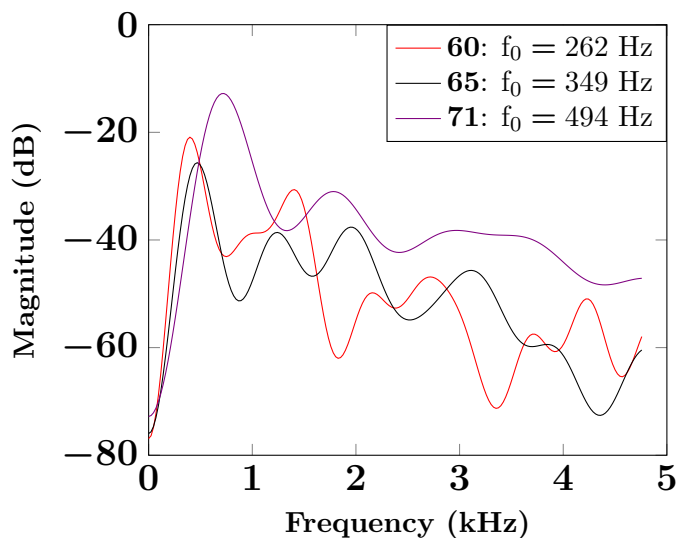


Figure 5.1: Harmonic Spectral Envelopes

in [45] and Fletcher in [47]. The envelopes we plot in Figure 5.1 shows exactly this variation across MIDI pitches.

The above results were on violin sounds from the Good-Sounds dataset. We also extract and plot the spectral envelope from our Carnatic Violin Dataset, as shown in Figure 5.2.

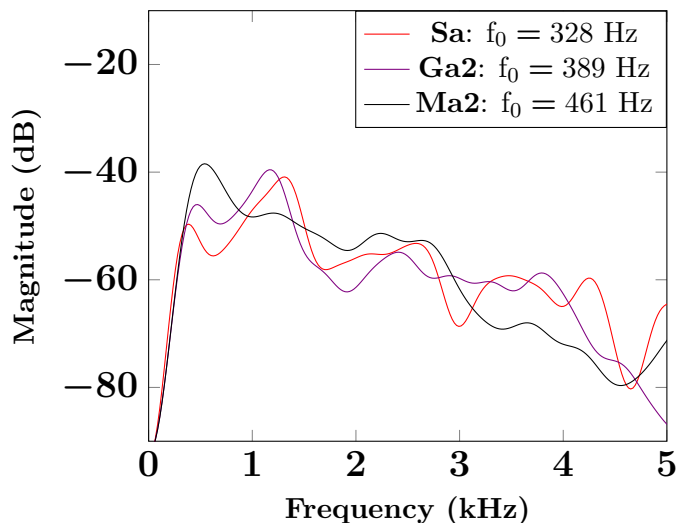


Figure 5.2: Harmonic Spectral Envelope from Carnatic Violin Dataset

From the plots, we come to the same conclusion that there is considerable variation in the envelopes across pitch. However, all the plots we show above only depict the harmonic spectral envelope, and not the residual spectral envelope. For a more complete picture we should also consider the parametric model for the residual component. Fletcher et al. [48, 47] performed a very interesting series of experiments on the perceptually important aspects of violin synthesis. The first study [48] discusses the salient aspects that could differentiate a ‘real’ violin tone from a ‘synthesized’ one. One of those that is of interest to us, and which we will explore further is the residual noise inherent

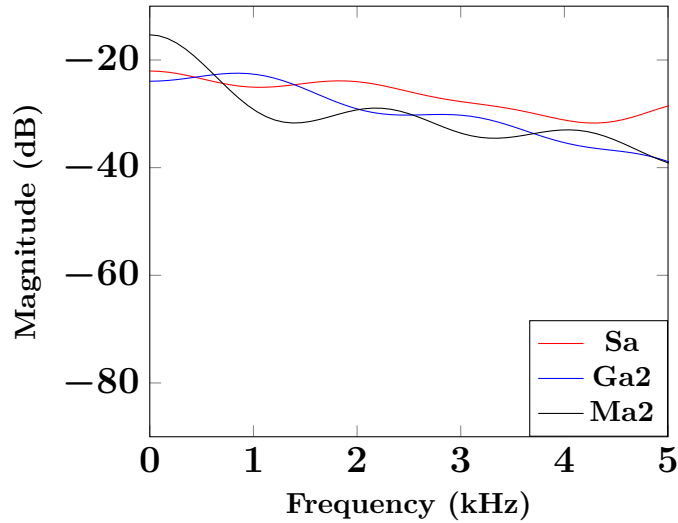


Figure 5.3: Residual Spectral Envelope from Carnatic Violin Dataset

in tone production (the noise produced when drawing the bow across the string). For the lower frequency notes, the fundamental and harmonics mask the noise. However, for the higher frequency notes, they are not able to mask the noise, hence the noise becomes audible. While this ‘noise’ helps in discriminating synthetic notes from real ones, the studies claim that it is usually inaudible for notes of lower frequencies, but becomes audible at notes of higher frequencies.

Similar to the harmonic component, the Residual Component could also depend on the pitch. Fletcher et al. in their work synthesize this noise by drawing the bow across the bridge without exciting the strings, thus effectively making it independent of the harmonic component. To test the dependence of the residual spectral envelope on pitch, we do the same thing as we did for the harmonic, we plot the residual spectral envelope for different pitches. For the audio, we use our Carnatic Violin Dataset.

Figure 5.3 shows that the Residual Spectral envelope does not significantly change for different pitches, thus hinting that the residual spectral envelope is indeed not dependent on the pitch (as the parametric model suggests). This can be explained by the fact that we have sufficiently sub-sampled the actual residual spectrum to capture variations in the residual envelope. We have also sub-sampled at the same frequency for all pitches (making the sampling independent of f_0 in effect).

If the harmonic Filter was independent of the source, then we should at least in principle be able to model the violin by only modeling the spectral envelope for a single f_0 . However, the discussion above tells us otherwise - There is variation in the harmonic spectral envelopes across pitch. This is where we invoke the next card in our hand - Generative Models!

5.2 Pitch Conditioned Generative Models

In Chapter 4, we discuss the Generative Model we employ, namely the Variational Autoencoder. The main reason in using a VAE over an AE is because it allows us to obtain a continuous latent space from which we can sample points (and synthesize the corresponding audio). The previous section tells us that the TAE estimate of the harmonic spectral envelope depends on the pitch. By conditioning on the pitch, we expect the net-

work to capture the subtle dependencies between the envelope and pitch, thus allowing us to generate the envelope more accurately, and at the same time giving us the ability to control the pitch. Figure 5.4 shows the networks we employ for this experiment. The H and R subscripts denote the harmonic and residual modeling networks respectively. Since the harmonic envelopes depend on pitch, we condition the harmonic cepstral coefficients on the pitch. For the residual, we do not do this conditioning because the residual envelopes are largely independent of the pitch, as seen from the previous section. Thus, we independently model the harmonic and residual components with separate networks. One question (a very relevant one) that might pop up in your mind is why do we model them independently? This is something we tackle later. Currently, we just model both of them independently.

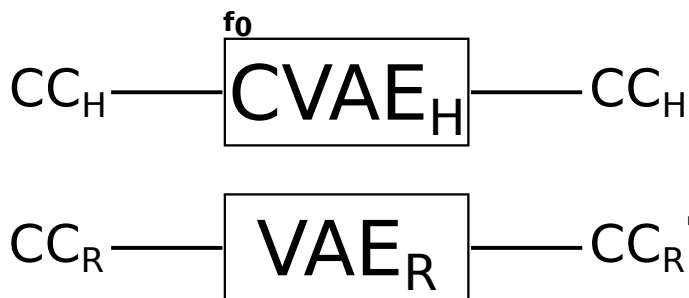


Figure 5.4: Independent Modeling (INet)

To additionally convince ourselves of the need for conditioning for the harmonic spectral envelope, we visualize the latent spaces of our harmonic VAE with and without pitch conditioning. A quick reminder about the latent space, it is the lower dimensional representation of the input data. However, the latent spaces we use are very high dimensional. We use violin clips from our Carnatic Violin Dataset here. From Chapter 4, we found the optimal latent space dimension to be 32, which is indeed quite high dimensional to visualize. To visualize the latent space, we have to use some kind of dimensionality reduction technique. The simplest is obviously Principal Component Analysis (PCA). However, we have no guarantee that our space is linear, and thus PCA would not be very helpful in visualizing structure. This non-linearity of our latent space motivates the use of a dimensionality reduction technique that preserves local structure in high dimensional space while projecting down to a lower dimensional space. t-SNE [49] is the technique we employ - it effectively projects high dimensional data onto lower dimensions (2 in our case) using matching between distributions, and helps in visualizing structure in the data through clustering.

Figure 5.5 shows the latent space visualization without any pitch conditioning. If the harmonic spectral envelope was independent of pitch, then we should ideally not be seeing any clustering in the latent space. However, we can see considerable clustering when we do not condition on the pitch. Another interesting thing to observe in the clustering is its structure. For notes close by in pitch, the clusters are close, and the clusters move away (from right to left) as you progress from the Sa to Ni3. The black arrow overlaid on top shows the progression of note clusters from Sa to Ni3. In essence, this plot tells us that the latent space still contains information on the pitch, thus providing additional motivation to condition the envelope on the pitch.

We repeat the above procedure, except this time we condition the harmonic spectral envelope on the pitch. Figure 5.6 shows the latent space visualization with pitch conditioning. We can see in the latent space that all the notes are clustered around

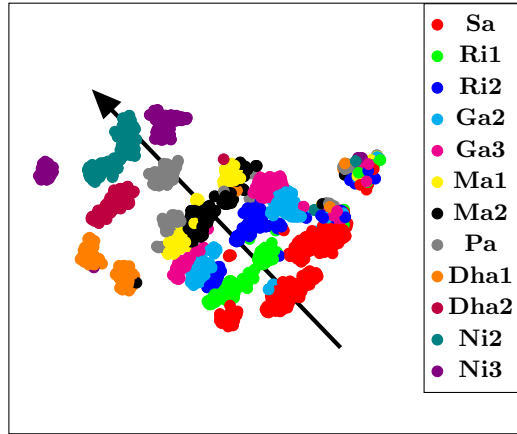


Figure 5.5: Harmonic VAE Latent Space without Pitch conditioning

together. What the pitch conditioning does is take care of the dependencies by learning a conditional distribution for the harmonic spectral envelope. It factors out the pitch dependence onto the external variable. Thus, with the pitch as a conditional, the decoder can correctly sample the latent space to obtain the optimal harmonic envelope for that pitch.

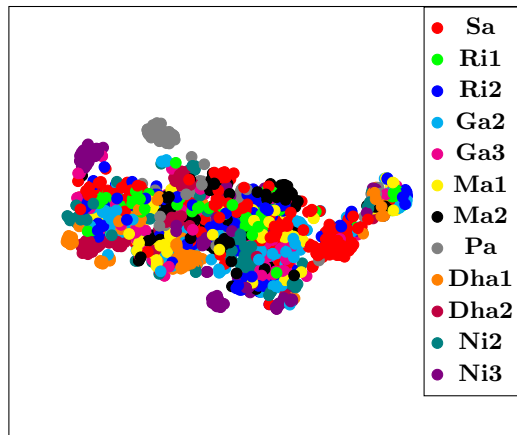


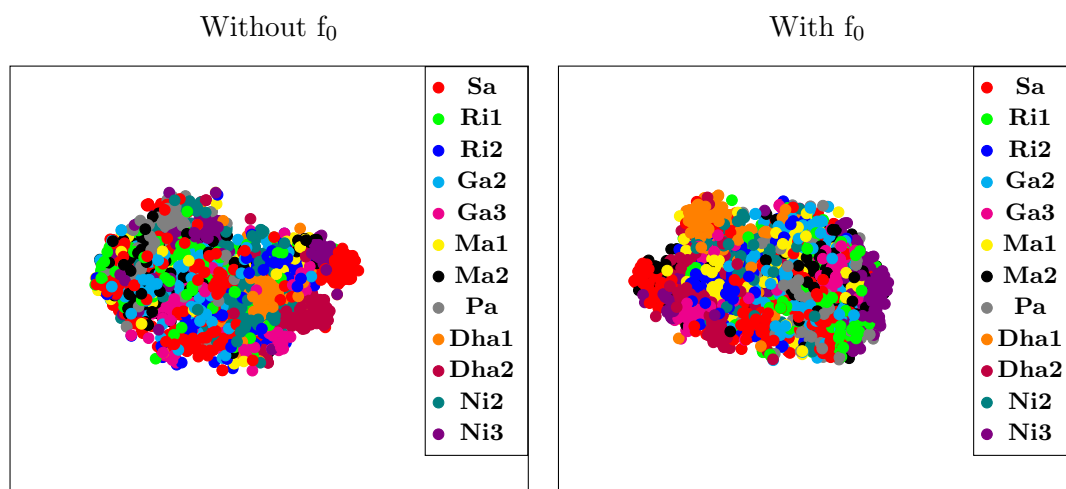
Figure 5.6: Harmonic VAE Latent Space with Pitch conditioning

Having analyzed the need for conditioning for the harmonic spectral envelope, we proceed to do the same for the residual spectral envelope. From the previous section, we saw that the residual spectral envelopes do not vary considerably for different pitch. We visualize their latent space, without and with conditioning in [Figure 5.7](#). We can see that there is no clustering observed, either without or with pitch conditioning, thus suggesting that the residual spectral envelopes are indeed independent of the pitch.

Summarizing what we have demonstrated so far,

1. The harmonic spectral envelope depends on the pitch. Using a Conditional VAE can potentially help in modeling these inter-dependencies
2. The residual spectral envelope does not depend on the pitch. Thus, no conditioning is needed for them

We have not established a procedure that can confirm conditioning does indeed improve

Figure 5.7: Residual VAE Latent Spaces without and with f_0 conditioning

the harmonic spectral envelope estimation. We perform ‘Reconstruction’ experiments, detailed ahead.

5.2.1 Reconstruction

As the name ‘Reconstruction’ suggests, we omit all instances of certain selected pitches during training, and see how well our model can reconstruct a note of the unseen target pitch. The spectral envelope of a note instance of the target pitch is input to the network. The output of the network is the reconstructed envelope, to be evaluated with respect to the input. These reconstruction experiments give us an idea of how well the networks can generalize to unseen cases.

We first present experiments on the Good-Sounds dataset. However, these only consider the harmonic component, and not the residual.

Good-Sounds Analysis

We consider two distinct training contexts for the reconstruction of a note with unseen pitch. (a) all instances of the neighboring MIDI notes upto 3 neighbors are included in the training set, as shown in Table 5.1; this is performed for $T = [63, 64, 65, 66, 67, 68]$. (b) the training set contains instances of only the octave endpoint MIDI notes, 60 and 71; we reconstruct instances of all the intermediate notes.

MIDI Kept	T - 3	T - 2	T - 1	T	T + 1	T + 2	T + 3
	✓	✓	✓	×	✓	✓	✓

Table 5.1: ✓ indicate MIDI note instances included in the training set for the synthesis of a given target note of MIDI label T.

In each of the above cases, we compute the MSE as the frame-wise spectral envelope match across all frames of all the target instances. The results are presented in Figure 5.8.

The first experiment asks the question if our network can synthesize the missing MIDI pitch when trained on the neighboring pitches on either side which provide some

context. The left plot in Figure 5.8 shows the MSE when each note is left out for $T = [63, 64, 65, 66, 67, 68]$. This plot tells us that there is no clear winner between the CVAE and AE. The next experiment skips all the pitches in an octave, sans the octave endpoints. We can see that the CVAE produces better reconstruction, especially when the target pitch is far from the pitches available in the training data. In the latter case, the MSE is seen to decrease as the target pitch moves closer to its nearer octave end pitch in both networks, as one might expect. Overall, the conditioning provided by the CVAE helps to capture the pitch dependency of the harmonic spectral envelope more accurately.

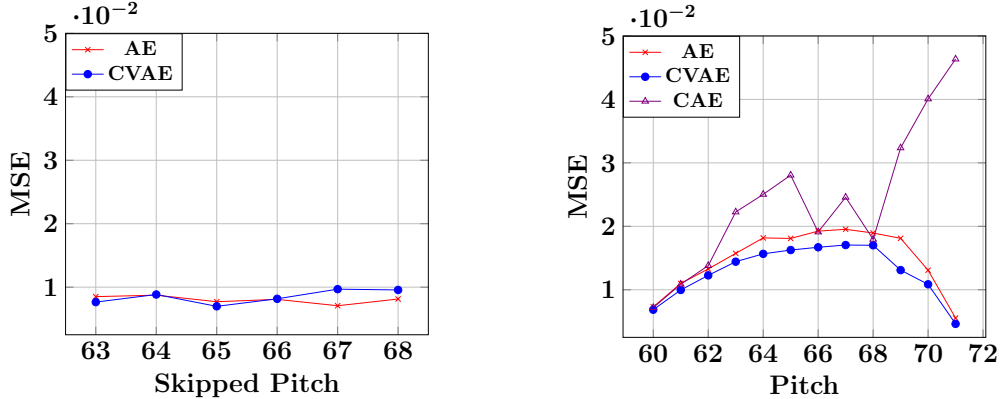


Figure 5.8: Harmonic spectral envelope MSE across unseen pitch note instances with close MIDI neighbors in training data (left), and only octave end notes in training data (right).

The above comparison between the AE and CVAE is not technically fair, as the CVAE has additional pitch information to work with. To emulate the effect of pitch conditioning with an Autoencoder (AE), we train the AE by appending the pitch to the input CCs and reconstructing this appended input as shown in Figure 5.9. We were motivated to do this from Wyse [14] who followed a similar approach of appending the conditional variables to the input of his model. We expect that the network can utilize the pitch information and learn the dependencies between the harmonic CCs and the pitch. This way, the AE is comparable to our proposed CVAE in that the network might potentially learn something from the f_0 during reconstruction. For reconstruction, we do not work with the reconstructed f_0' , rather we use the original f_0 given as an input.

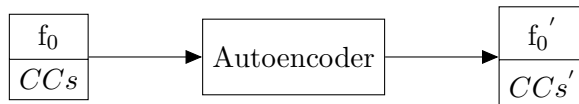


Figure 5.9: ‘Conditional’ AE (CAE)

We then perform the experiment of training this CAE only on the endpoints. The right plot in Figure 5.8 shows the MSE plot obtained for the CAE. As can be seen, appending f_0 does not seem to be improving the model, on the contrary, it seems to worsen the AEs performance in reconstructing the skipped notes. This is expected because the AE loss function does not force the network to learn the joint dependencies, it simply forces it to reconstruct the input as well as it can. Thus, using a CVAE is more beneficial than simply appending the pitch to the input and expecting the network to do the rest.

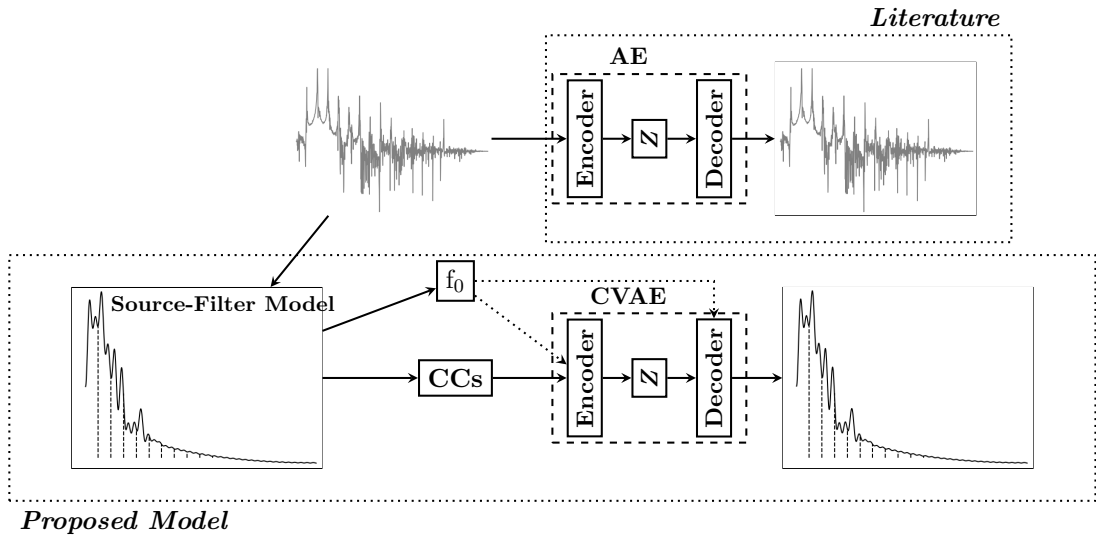


Figure 5.10: Flowchart of the state of the art frame-wise audio synthesis pipeline (upper branch) and our proposed model (lower branch). Z represents the latent space learned by the (CV)AE.

The lower branch in Figure 5.10 summarizes our proposed model. To highlight the contrast between our proposed Parametric Model and the Spectral models currently in use in literature, we also carry out reconstruction experiments with the frame-wise magnitude spectra as detailed by Roche et al. [23], which directly autoencode the magnitude spectra. The network has been trained on MIDI 61,62,64,65. Then, we both skip and include MIDI 63. The network will be able to reconstruct MIDI 63 when trained on it. Will it be able to do so if we skip MIDI 63? That is what we demonstrate. Figure 5.11 shows the input MIDI 63 spectrogram to the model.

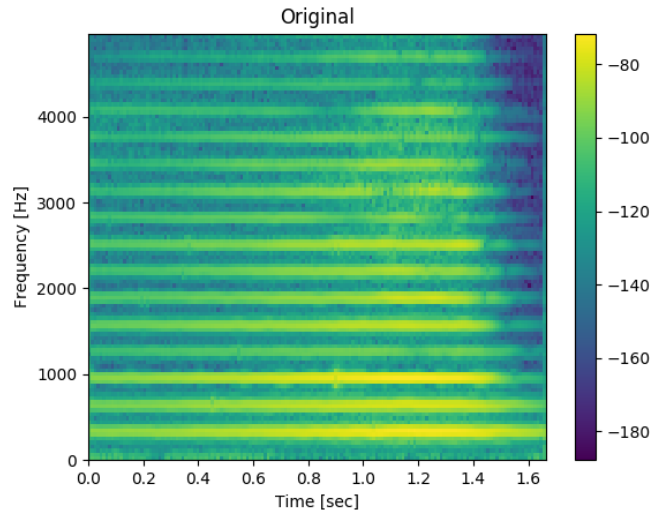


Figure 5.11: Input Magnitude (dB) Spectrogram

As expected from an AE, it can reconstruct MIDI 63 when it has been trained on it. Figure 5.13 shows the reconstructed spectrogram.

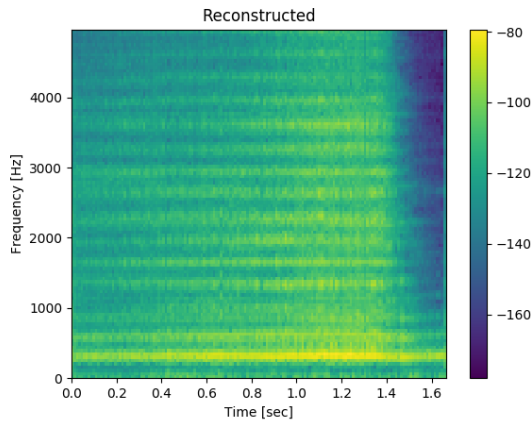


Figure 5.12: Reconstructed magnitude spectrogram with MIDI 63 skipped

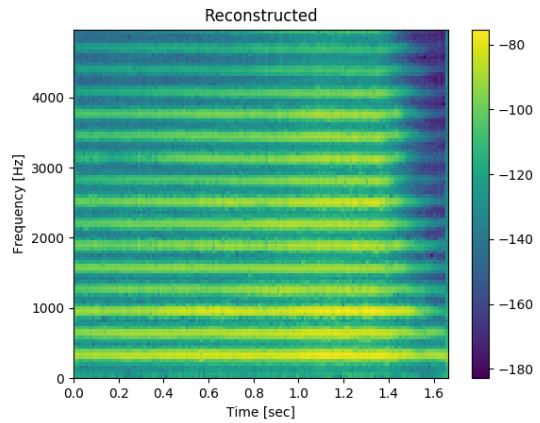


Figure 5.13: Reconstructed magnitude spectrogram with MIDI 63 included

However, the AE fails to reconstruct the note it has not been trained on, in spite of having been trained on nearby notes. Figure 5.12 shows the reconstructed spectrogram. It is quite distorted and lacks even a clear harmonic structure. Because of the audio representation used (a magnitude spectrogram), the generalization capability of an AE is limited. If the magnitude spectrogram of nearby notes were similar, then the AE should in principle be able to reconstruct the skipped notes. However, the spectrogram contains joint information from both the source (pitch) and filter (spectral envelope). If the AE is instead trained on the filter, we know that the filter does not drastically change across notes. Further, if we use a CVAE instead of an AE (as demonstrated above), the network also captures the source-filter interdependencies.

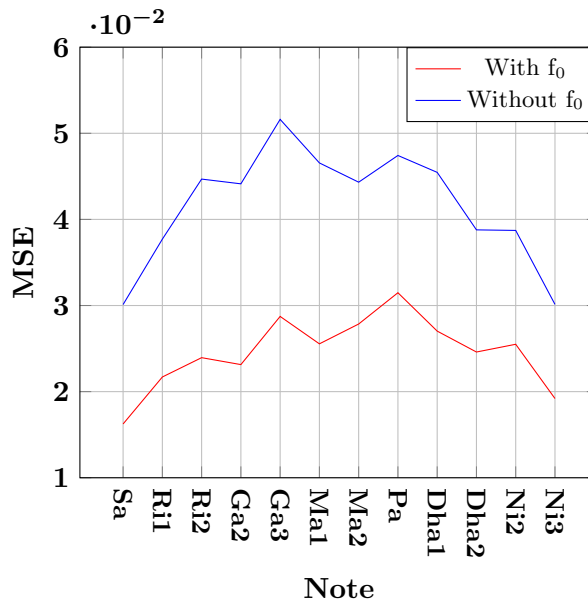


Figure 5.14: CVAE MSE with and without Pitch Conditioning

Carnatic Violin Analysis

We also perform a few experiments on the Carnatic Violin dataset we have recorded. This is to ensure consistency of results across datasets. It also functions as a sanity check for the assumptions we make.

Similar to one of the experiments demonstrated, we train networks on the pitch endpoints in the middle octave (as shown in Table 3.3). The endpoints in this case correspond to the notes Sa and Ni3. Similar to the previous experiment, we compute the MSE in the frame-wise harmonic spectral envelope match across all frames of all the target instances. The only difference is that here, both the networks are Variational Autoencoders, and we compute the MSE with and without pitch conditioning. We see a similar trend in Figure 5.14 and Figure 5.8 in that pitch conditioned networks have a lower MSE overall as compared to without pitch conditioning. This reaffirms our belief in the necessity to pitch condition the networks modeling the harmonic spectral envelope.

5.3 Interdependence of Harmonic, Residual

In the previous section, we only discuss the importance of conditioning for the harmonic CVAE. We did begin the discussion noting that the residual spectral envelope does not vary considerably with pitch, hence there is no need to condition. However, we did not consider the possibility of interdependencies between the harmonic and residual component, and we modelled both of them independently of each other. In this section, we now provide motivation as to why the harmonic and residual might depend on each other, and we present motivation for the same. After this, we also propose network architectures that could potentially take care of these interdependencies.

Before the discussion on interdependence, why is the residual so important? Fletcher et al. [48, 47] formally define the residual as the noise produced when drawing the bow across the string. They perform a series of experiments to compare the noise levels with harmonic partial levels for a violin. Figure 5.15 shows the plot they obtain.

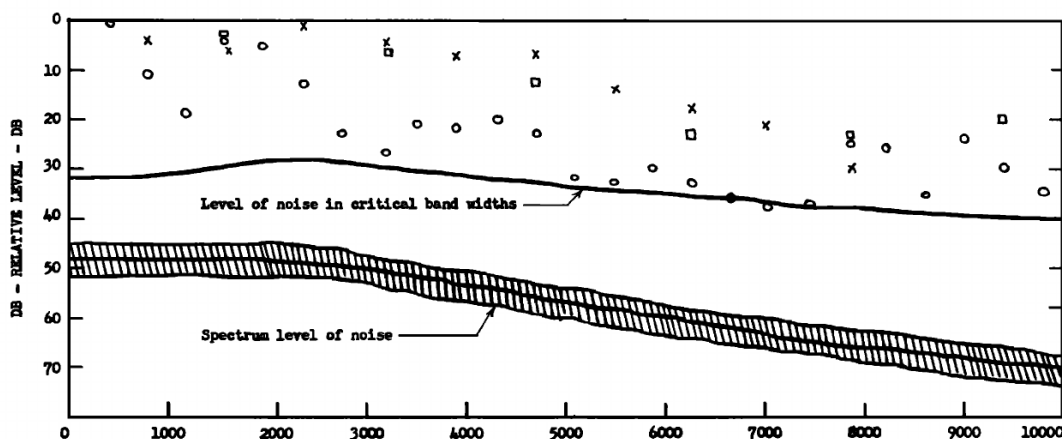


Figure 5.15: Noise Levels vs Frequency (figure taken from [48])

In Figure 5.15, \circ represents the partials for G (392 Hz), \times for G1 partials (784 Hz) and \square for G2 partials (1568 Hz). The G' (196 Hz) partials have not been shown in the plot. For the lower frequency notes, the fundamental and harmonics mask the noise.

However, for the higher frequency notes, it is not able to mask the noise, hence the noise becomes audible. As said in the paper [48] (verbatim in the authors own words),

“An examination of [Figure 5.15](#) shows why for the tones G’ and G the noise is inaudible. The fundamental and the harmonics mask the noise. For both G’ and G, the noise that is lower in frequency than that for the fundamental is masked by the fundamental. However, this is not true for G1 and G2. For example, for G2, the fundamental frequency is 1568 cps. When the sound-pressure level of this fundamental at the ear is about 80 dB, then it will not mask pure tones that have frequencies below 1000 cps and levels higher than 50 dB. The noise in a critical band acts like a pure tone. So it is seen why the noise for G2. is audible. The same is true for G1. The tone G is on the border line so the noise is neither definitely audible or definitely inaudible.”

Thus, the residual noise is more prominent for the higher frequency notes than it is for the lower frequency ones.

Fletcher et al. in [47] extends the above analysis, from sustained single notes to vibrato notes. A few observations brought out are:

1. During vibrato, the relative amplitudes of the partials are not fixed over time i.e. there is a phase offset amongst the amplitudes of various partials. Thus, the harmonic structure varies across time.
2. The superimposed noise is usually inaudible for notes of lower frequencies, but becomes audible at notes of higher frequencies.

As we mentioned previously, to synthesize the residual, the authors draw the bow across the bridge without setting the strings into motion (i.e. no harmonics!). Thus, in effect, they synthesize the same residual for all of the pitches, further justifying that we do not have to condition the residual generation network on the pitch. What then justifies the interdependence of the harmonic and residual component? Mathews et al. [46] in their studies propose a theory of ‘Resonant Enhancement’ of tones which states that the rich timbre of the violin is essentially due to the string vibrations being filtered at the resonant locations of the violin body. This effectively tells us that if the string vibrations are filtered, then the noise produced by the bowing should also be filtered by the same resonances. Thus, both the harmonic and residual components are processed by the same violin body resonances and cannot be assumed to be independent.

When we bow the string harder to produce a louder tone, the residual component will also be loud, and they both will be filtered by the violin body simultaneously, thus indicating that the harmonic and residual fundamentally depend on the playing style of the note. To check our hypothesis, we show the harmonic and residual spectral envelope variations in [Figure 5.16](#) for the same note by varying the loudness from soft to loud.

The vertical blue lines in the harmonic spectral envelope in [Figure 5.16](#) represent the absolute magnitude differences for the harmonics. If loudness variation were a simple amplitude scaling, then both the harmonic and spectral envelopes should be shifted up (log-plots) and the blue lines should all be the same length. However, as we see, a loudness increase is not just a scaling. It causes certain frequencies to be boosted, others to be suppressed, and also changes the tilt in the spectral envelope.

To contrast the harmonic with the residual, we also plot the residual spectral envelopes in [Figure 5.17](#). Even the residual envelope is not a simple shifting as one would

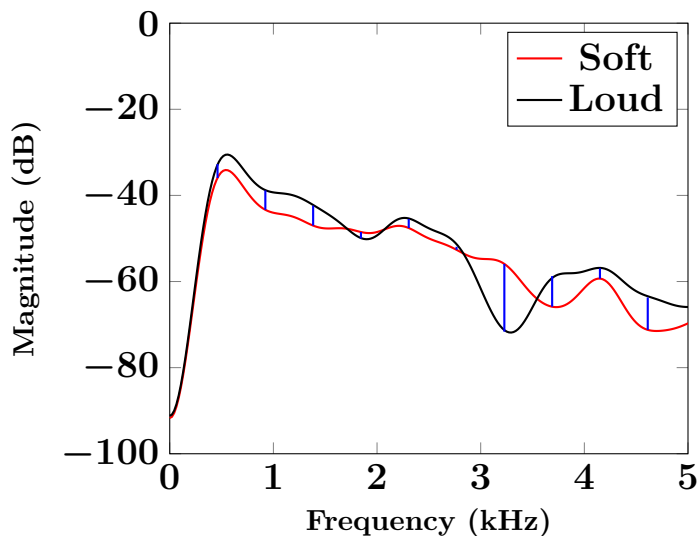


Figure 5.16: Harmonic Spectral Envelope for Different Loudness

expect if it were simply scaled. This further strengthens our hypothesis that the harmonic and residual envelopes must be dependent as they have a common underlying origin in the played loudness style of the note.

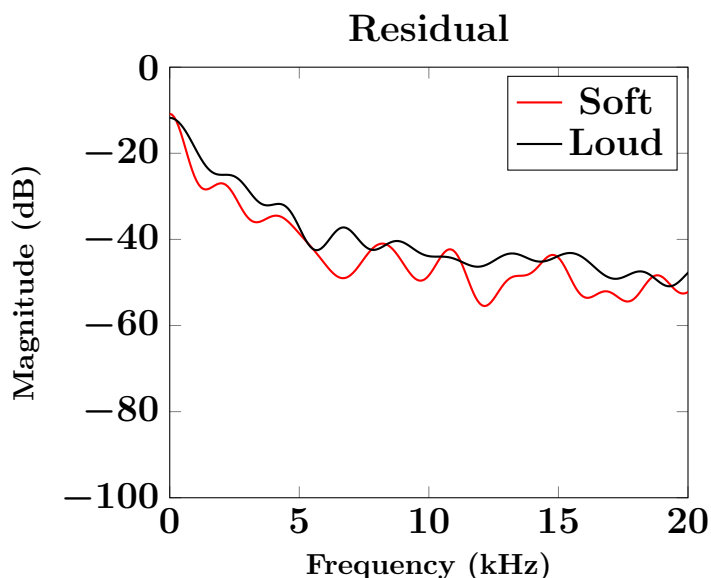


Figure 5.17: Residual Spectral Envelope for Different Loudness

There could be many different ways to try joint modeling in a neural network. The simplest procedure however is to simply concatenate the inputs and feed them to a CVAE to model them together, as shown in Figure 5.18. Since the encoder and decoder are given as input both the harmonic and residual CCs, the reconstruction inherently takes into account both the harmonic and residual components. The second approach of modeling the sum and difference of CCs is more non-trivial. The intuition behind it comes from current methods that generatively model the magnitude spectrum of the sound [22, 23]. The magnitude spectra is the sum of the harmonic and residual spectra. Thus, by directly modeling the spectrum, the autoencoder takes care of both of them

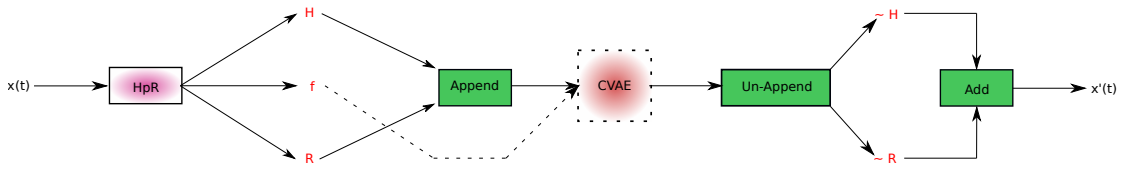


Figure 5.18: Concatenative Modeling (ConcatNet)

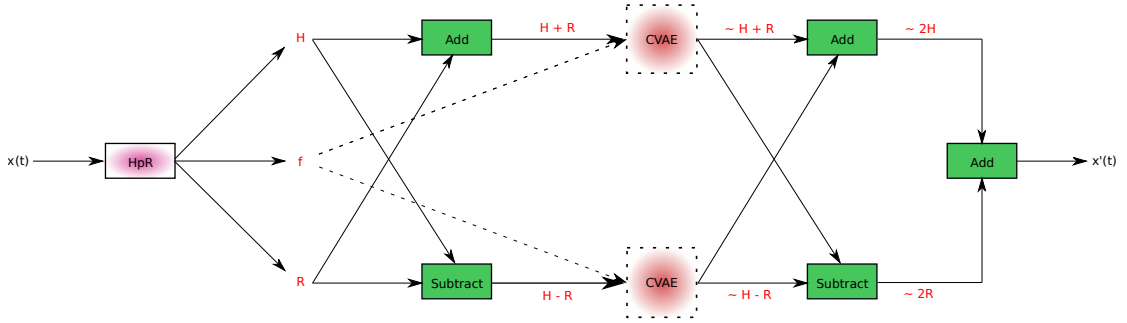


Figure 5.19: Modeling sum and differences (JNet)

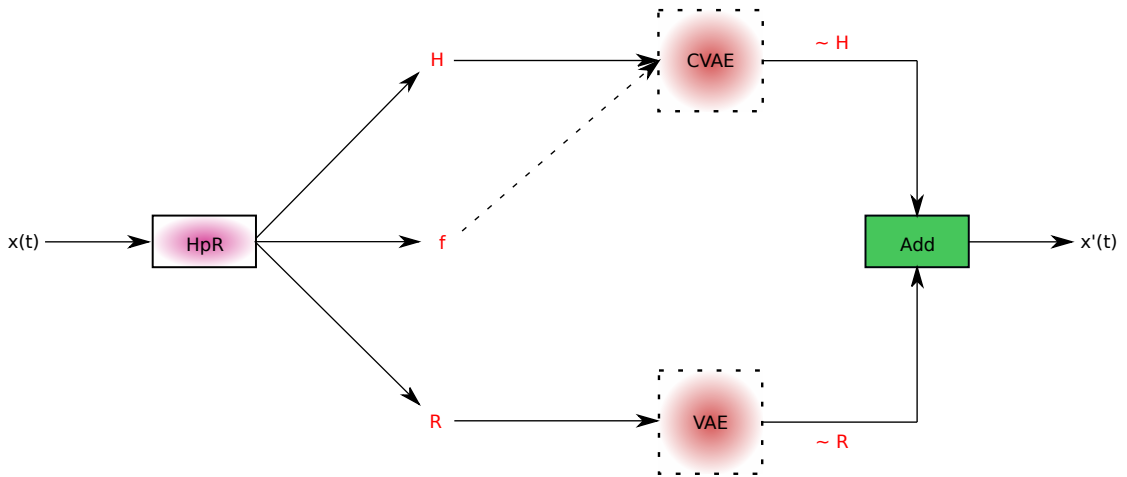


Figure 5.20: Independent Modeling (INet)

together. If we could somehow model the difference of the harmonic and residual spectra as well, we could individually obtain the harmonic and residual components. That is exactly what we try to do via our network, as shown [Figure 5.19](#). We have 2 networks, the sum and difference networks. The sum network, in the process of autoencoding the sum of the harmonics and residual inherently learns their joint dependencies. The difference network is a ‘trick’ to extract the individual harmonic and residual components from the sum network. We can obtain the harmonic and residual vectors by simply adding and subtracting the outputs of the sum and difference networks. One might ask why do we need the individual components? Keeping in mind the end-goal of being able to synthesize audio, it would be good to have the harmonic and residual components if one is additionally interested in ‘modifying’ the audio (time stretching, frequency scaling, morphing etc.). As a baseline for comparison, we also model the harmonic and residual independently as shown in [Figure 5.20](#). The dotted line in the networks represents the pitch being used as a conditional variable to the networks. H,R represent the framewise

harmonic and residual cepstral coefficients to the network, and $\sim H$ and $\sim R$ represent the network reconstructions of the same. We use our Carnatic Violin Dataset for this experiment.

How to decide which network works better? We plot the reconstruction MSE, which is computed as the average over all test instance frames given as input to the network (test here refers to the fact that the network has not seen these during training). We work with the sustain portion of the notes in our dataset, and split it to train and test data evenly. To allow the network to learn the potential dependencies of the harmonic and residual components, we train with frames of both loudness's - soft and loud. Also, we choose notes in the higher octave because Fletcher et al. [48, 47] mentions explicitly that the residual plays a more important role perceptually in the higher octaves. Thus, with this joint modeling, we hope to see the residual being reconstructed at a lower MSE.

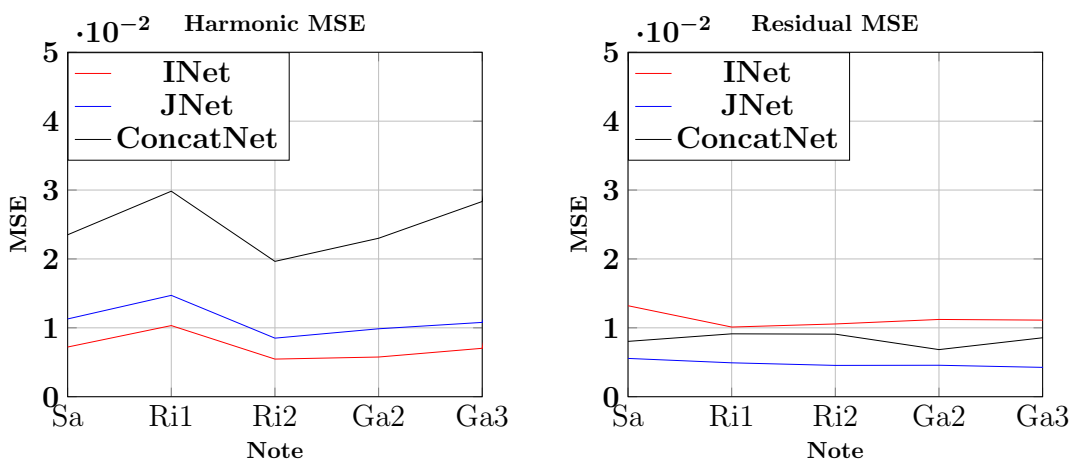


Figure 5.21: Reconstruction MSE

Figure 5.21 shows the note-wise reconstruction MSE. For the Harmonic MSE, the Independent Modeling proves to be most superior. Interestingly, for the Residual MSE, the joint modeling methods result in a lower MSE though, thus strengthening our belief in joint modeling of the harmonic and residual components.

5.4 Generation/Synthesis

The previous experiments evaluated the networks reconstruction capabilities. However, we are also interested in using it as a synthesizer which is capable of generating instrument audio. Thus, in this experiment, we see how well the network can generate the spectral envelope of an instance of a desired pitch (not available in the training data of the network). We use the Good-Sounds dataset for the current experiment, with a similar experiment being performed for our Carnatic Violin dataset as well.

We train the network on instances across the entire octave sans MIDI 65, and then generate MIDI 65. Generation comes naturally to the CVAE, as we just have to sample latent points from the prior distribution, and pass them through the decoder along with the conditional parameter f_0 to generate the spectral envelope, as shown in Figure 5.22. An immediate question that might arise is how to sample multiple frames, since a single latent variable only represents a single frame. Thus, we have to coherently sample multiple latent variables and decode them to obtain multiple contiguous frames.

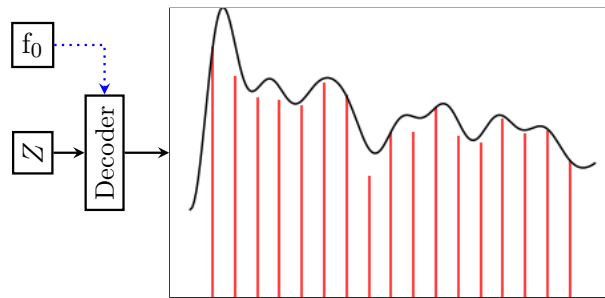


Figure 5.22: Sampling from a CVAE

One naive strategy is to sample a single point again and again. The problem with this approach is that the reconstructed note sounds ‘synthetic’ because of lack of any variation. A simplistic approach to introduce these variations could be to introduce a small jitter in the pitch track. This jittered pitch track would further cause variability in the generated spectral envelopes. This approach does not work however, and the synthesized note still sounds fairly synthetic. Another approach is to sample points from the latent space in a small neighborhood. Motivated from [26], we perform a random walk with a small step size near the origin in the latent space to sample points coherently. We synthesize the audio by sampling the envelope at the harmonics of the specified f_0 , and perform a sinusoidal reconstruction. Further, we try to generate a practically useful output, viz. a vibrato violin note with typical vibrato parameters. This exercise involves reconstructing spectral envelopes corresponding to the continuum in the neighborhood of the note MIDI pitch.

To synthesize the residual bow sound, we train a network on the residuals, and repeat the above sampling procedure. An interesting thing to note is that the ‘synthesized’ residual does not sound anything like a violin bow. It sounds closer to noise in-fact. How are the networks we proposed in the previous section able to reconstruct the residual then? The reason for this lies in the sampling procedure. Because we have trained our networks on single frames, the latent space does not represent any temporal information. We introduce temporality in our synthesized audio via sampling the latent space in a particular way (the random walk being one such way). However, this might not correspond to the ‘actual’ temporal order of the audio. Because of this, the generated residual might be sounding simply like noise because if you want to make it sound like an actual violin bow, there exists a certain trajectory along which you should sample points from the latent space. We will discuss this point again in more detail in the next chapter when we discuss listening tests conducted on our network audio.

Chapter 6

Continuous Pitch Contour Analysis

In our work so far, we have only discussed the reconstruction and synthesis of audio notes with fixed pitches. However, Carnatic Music is much more than a sequence of these fixed pitch notes being played one after another. As seen in [Figure 3.5](#) and [Figure 3.6](#), Carnatic Music contains highly variable pitch contours. The ornamentations in these pitch contours are also known as ‘Gamakas’. An immediate thing to note is that the task of reconstructing these pitch contours is not as trivial as the previous fixed pitch notes. This is because the network is only trained on certain fixed notes. From these fixed notes, the network has to interpolate in between notes to generate the audio at the pitch values not corresponding to the fixed notes.

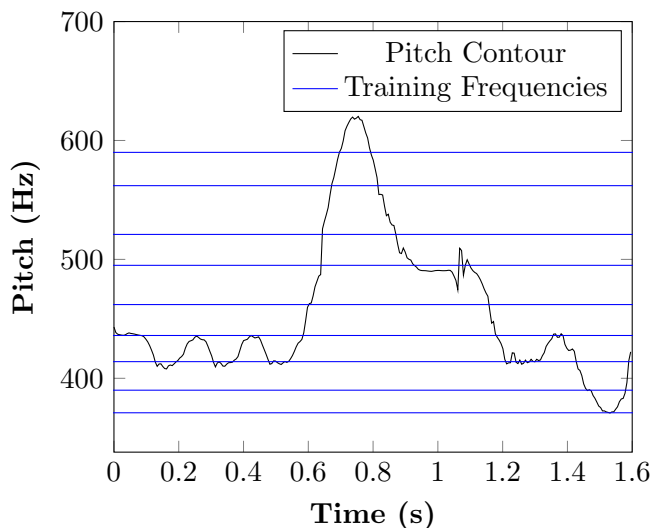


Figure 6.1: Pitch Contour and Discrete Notes

[Figure 6.1](#) shows an example pitch contour with the fixed note training data overlaid on top of it. As you can clearly see from the plot, most of the pitch contour does not lie on the training data, and it is up-to the network to correctly determine what the optimal interpolation is. From the experiments in [Chapter 5](#), we see that conditioning the harmonic network on the pitch enables the network to obtain a more accurate reconstruction. We now proceed to train the network on the same discrete notes, and see how well it can reconstruct the continuously varying pitch contour. Before we do so

however, we point out some of the challenges involved in obtaining the pitch contour for Carnatic Music.

6.1 Pitch Extraction for Carnatic Music

The HpR Model by Serra [4, 5] uses the Two way Mismatch (TwM) [50] algorithm for estimating the fundamental frequency of a frame. However, this procedure does not give us a very good estimate of the pitch contour from violin recordings.

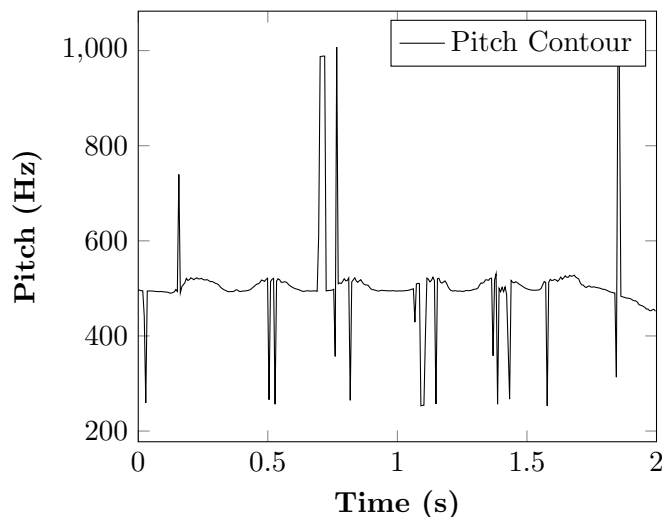


Figure 6.2: Pitch Contour extracted with the TwM algorithm

Figure 6.2 shows a pitch contour extracted using the TwM algorithm. As we clearly see, there are estimation errors whenever the pitch is transitioning. Thus, if we use TwM to estimate the pitch, we might end up with a poor representation of the pitch, which in turn will not allow us to represent the audio in its parametric representation. There is the need for a better pitch estimation algorithm. pYIN is one solution we have found to this issue. pYIN or Probabilistic YIN [51] builds on top of the original YIN [52] pitch estimation algorithm. The YIN algorithm works by trying to estimate the pitch from the time domain waveform, along with a few post processing steps to optimize the estimation. pYIN build on top of this by using probabilistic thresholds and a Hidden Markov Model to allow for smooth pitch transitions across frames. Refer to this online demo¹ by the author as part of a course project that explains and compares YIN and pYIN in more detail specifically for Indian Classical Music.

Figure 6.3 compares the pYIN extracted pitch contour with the TwM extracted one. The pYIN is clearly a much better extractor, as it is not as noisy as the TwM estimate. The pitch contours in Figure 3.5 and Figure 3.6 have also been extracted using the pYIN algorithm.

We thus modify the HpR algorithm to work with the pYIN pitch instead of the TwM pitch. Here is where we face another technical snag. The pYIN pitch contour obtained is still not very good. This was observed when listening to the parametric reconstruction of the audio, which was quite bad. When we plot the pYIN pitch against the first partial (twice the fundamental, thus by dividing it by 2, we obtain an estimate for the pitch) obtained using the HpR algorithm, we observe something interesting.

¹https://www.ee.iitb.ac.in/student/~krishnasubramani/data/pyin_demo

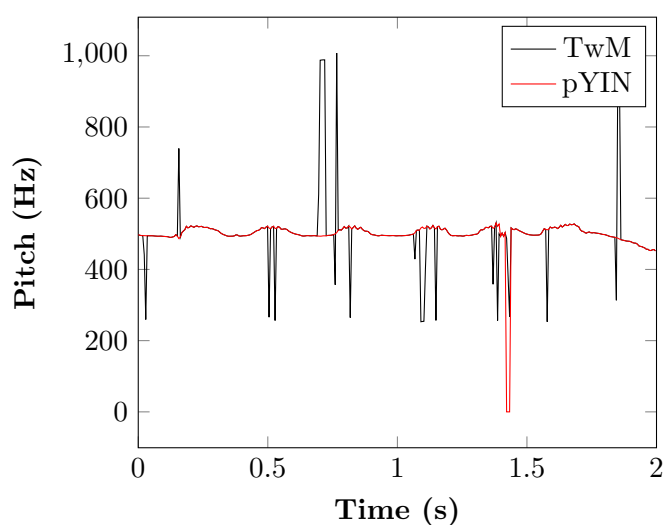
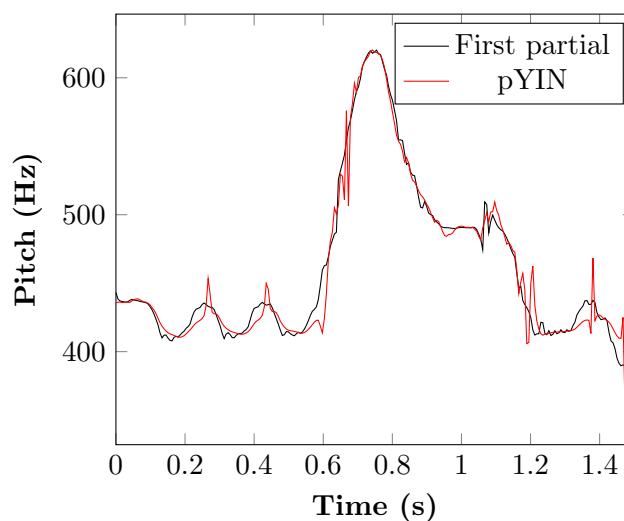


Figure 6.3: Comparison of pYIN and TwM

Figure 6.4 shows the comparison of the two pitch contours. The pYIN estimate is quite bad compared to the estimate from the first partial, which is a lot smoother. This is because, the higher harmonics are estimated more accurately using the approximate information obtained from the pYIN estimate.

Figure 6.4: Comparison of pYIN and P_1

Mathematically, if P_1 represents the first partial, the fundamental frequency estimate is simply,

$$f_0 = \frac{P_1}{2}. \quad (6.1)$$

One important thing to mention here is the difficulty in the extraction of the residual, especially with the varying pitch contour. Although the pitch estimate is improved, it is not a perfect estimate. This causes the residual (obtained by subtracting the harmonic from the actual signal) to contain harmonic content as well, and this affects the residual. Thus, in our experiments ahead, we only work on reconstructing the harmonic component of audio with variable pitch contours.

6.2 Continuous f_0 Analysis of Harmonic Component

6.2.1 Reconstruction

We train our harmonic network on all the 3 octaves of the Carnatic Violin Dataset [Table 3.3](#) to give it information about all the notes. Having trained it, we then perform the framewise reconstruction of the variable pitch contour, with the conditioning variable being the pitch value for that frame. It is similar to the procedure shown in the lower branch of [Figure 5.10](#), except that the f_0 corresponds to the variable pitch contour. [Figure 6.5](#) shows an input and reconstructed spectrogram with the red curve in the reconstruction indicating the extracted pitch.

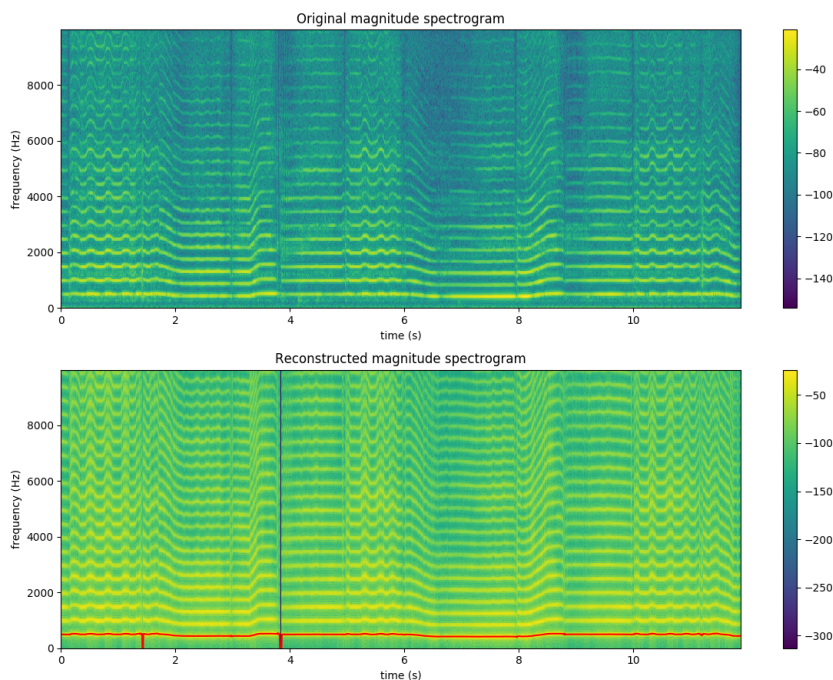


Figure 6.5: Spectrograms of Input and Reconstructed Audio

The results are as we expect; our network is able to successfully reconstruct the variable pitch contour in spite of having been trained only on the fixed notes (refer [Figure 6.1](#)). However, what is observable from [Figure 6.5](#) is the clear lack of the residual (in terms of the spectrogram appearing less noisy) for the reconstructed audio. However, the successful reconstruction of the harmonic strengthens our choice of representation for the network - the parametric Source-Filter inspired model we use for audio is able to ‘decouple’ the timbre and the pitch, and by conditioning the network on pitch, the network can model the inter-dependencies.

6.2.2 Generation/Synthesis

Having successfully reconstructed variable f_0 pitch contours, we now see if we can also generate/synthesize variable pitch contours with the procedure discussed in [section 5.4](#). The only difference is that the pitch contour is variable instead of fixed. [Figure 6.6](#) shows the spectrogram of a synthesized vibrato with the red line depicting the pitch contour.

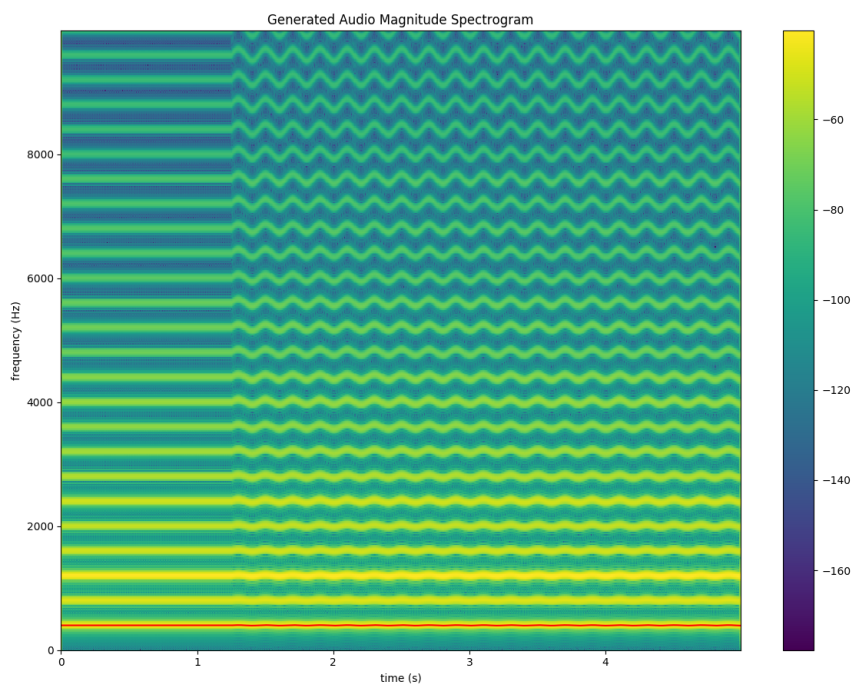


Figure 6.6: Spectrogram of a Generated Vibrato

6.3 Listening Tests

Both in [Chapter 5](#) and [Chapter 6](#), we perform a series of experiments broadly labeled as ‘Reconstruction’ and ‘Generation’. In the reconstruction experiments, we reconstructed test audio i.e. the audio the network has not been trained on. In the generation, we provide the pitch as a conditional, sample points coherently from the latent space, and then ‘synthesize’ the audio corresponding to the pitch input. We have reported the MSE for the reconstruction experiments. However, the MSE need not directly correspond to audio perception. Thus, to get more effective feedback on the quality of the audio reconstructed/generated by the network, we conduct a simple informal listening test with 2 experienced violinists.

Why do we consider feedback from only 2 violinists instead of crowd-sourcing our survey? Firstly, we cannot guarantee the quality of feedback we receive from crowd-sourcing the survey on a platform like Amazon Mechanical Turk. Second, there is no guarantee that the people taking the survey would have listened to or played the violin to make insightful comments on it. Also, by asking specific questions to professionally trained violinists, we can zero in onto the perceptually salient aspects of violin audio. We will refer to the two violinists as V1 and V2. V1 has been trained in Carnatic Violin and Vocal for 16 years. V2 has been trained in Carnatic Violin and Vocal for 14 years. Both of them have been trained on the acoustic violin. We present the two violinists with audio files in an online notebook², and ask the violinists to listen to these. We then ask a series of questions which will allow us to obtain a better insight about the perceptual quality of the audio.

²<https://www.ee.iitb.ac.in/student/~krishnasubramani/LT.html>

6.3.1 Network Reconstruction

We ask the violinists to listen to the network reconstruction of both fixed pitch (as presented in [Chapter 5](#)) and variable pitch contours as presented in this chapter.

Fixed Pitch

We evaluate the outputs of the 3 networks which jointly model the harmonic and the residual (detailed in [section 5.3](#)). We consider the upper octave notes because the residual is more significant in the higher frequencies. We present the violinists with 4 audio clips each of two notes, Sa and Ri2. The 4 audio clips for each note are the original audio, and the three network reconstructions (INet, JNet and ConcatNet) (the harmonic and residuals are modeled by the networks, and added together to obtain the final output from the network).

For both the notes Sa and Ri2, both violinists V1 and V2 found it difficult to distinguish the original note from the INet reconstructed note. They observed the residual to be a bit louder than the original in the reconstruction, but still perceived it as natural. When we say natural here, we mean that it sounds similar to the input, and can be produced on an acoustic violin. The JNet reconstructed note is perceived as slightly less natural. The ConcatNet reconstructed note is however perceived to be noisy i.e. the residual is noisy. There is also a change in the timbral quality of the note. Both V1,V2 noted that it sounded like a poorly recorded violin note.

From this, we can make an observation that simple concatenation of the harmonic and residual (ConcatNet) is not an optimal method to model inter-dependencies. However, between independent modeling (INet) and joint modeling (JNet), INet seems to be more natural, though JNet is also perceived to be natural.

Variable Pitch Contours

We only consider the harmonic section of INet here, because, as mentioned above, the residual for the variable pitch contours is not extracted properly (contains significant harmonic content). We use short snippets from Raga Mayamalavagowla and Raga Shankarabharanam containing Gamaka segments. We then convert these to their parametric representation, and only reconstruct the harmonic portion. We then present the network reconstructed audio, and ask the violinists to compare it to both the actual audio and the harmonic parametric version of the audio.

Both V1 and V2 spotted the difference between the original audio and its harmonic representation. They commented that the harmonic representation lacked the bow movements at the note onsets. V1 made an interesting comment, he said that there were a few artifacts in the harmonic representation, which he compared to the artifacts introduced when one uses the autotune software. This was especially prominent during the note slides, where he felt some slight distortion was present. When presented with the INet reconstruction of the same, both V1 and V2 comment that there is a slight timbral change in the reconstruction. V1 additionally commented that if someone were not to pay too much attention to the audio, then both the harmonic version and the network reconstruction could be passed off as natural sounding violin audio.

Their observations on the distortion during glides is interesting. It tells us that the parametric representation we use for the audio is still not perfect. As we explain in the previous section, pYIN is not without estimation errors, thus these could be responsible for affecting the quality of the reconstructed audio.

6.3.2 Network Generation

We give fixed and variable pitch contours as conditional inputs to our network. The network has been trained on all available notes of all three octaves. We then present these generated audio to the violinists and ask them for their opinion on it.

Fixed Pitch

The network generated notes were immediately perceived to be synthetic akin to a complex tone not filtered by violin body and not at all like an acoustic violin. Both V1 and V2 comment that it sounded like an electronic violin, or like a violin note played on a MIDI synthesizer. The generated note sounded ‘too perfect’ to be played by humans i.e. there was minimal (or almost no) variability in the pitch, loudness and timbre. Even after a vibrato was added to the generated tones, they are still not perceived as natural. This is because the vibrato sounded too ‘perfect’. Real players would have variations within the vibrato itself (variations in vibrato depth and frequency) which contribute to its perceived naturalness.

Continuously Varying Pitch

The procedure is exactly the same as the previous fixed pitch note generation, except that we provide a continuously varying pitch contour as the conditional variable. We extract the pitch from short snippets of Raga Mayamalavagowla and Raga Shankarabharanam containing Gamaka segments, and provide them as the conditional input to our harmonic network, and synthesize the audio. In this case as well, both the violinists comment that the network generated audio does not sound like an acoustic violin, rather, it sounds like an electronic violin. Also, because of the lack of bow noise, the audio is perceived to sound artificial. The lack of dynamic variability (the synthesized audio has the same loudness throughout) also contributes to its artificiality. V1 however does comment that since the pitch movements are common in Carnatic Music, these synthesized notes sound a bit more natural than the fixed pitch ones discussed before this.

This brings us again to the importance of sampling the latent space for audio synthesis. We simply perform a random walk with small step-size. However, as we learned from the violinists, this random walk is not enough to introduce variability into the synthesized note- it still sounds artificial. How do we pinpoint and say that the sampling trajectory is the reason? This is because, our network is still able to reconstruct notes quite well (as commented by the violinists in the previous section, for INet and JNet, the reconstruction and actual note sound quite similar). The only difference in reconstruction and generation is that in reconstruction, we project the input onto the latent space, and then sample in the order of the frames projected. However, for generation, we do not sample in that order, we do so randomly. Doing this random walk to generate the harmonic component still generates audio that at least sounds harmonic. However, if we do this for the residual component, the generated residual simply sounds like noise, and not at all like a violin bow sound. This is a drawback in our model that has to be addressed if one wants to generate more realistic audio from our model. To address the issue of sampling effectively from the latent space, we could follow the procedure highlighted in [53] where the authors train neural networks to traverse the latent space. Similar to their approach, we could train another neural network to learn an effective sampling trajectory from the reconstruction of the note recordings. This would allow

us to generate more realistic sounding audio from the network.

6.4 Graphical User Interface for Experiments

To easily demonstrate our networks capabilities, and to allow people to interact with our network more easily, we present a Graphical User Interface (GUI) for researchers to interact and play around with our code more easily. A screenshot of our GUI is shown in [Figure 6.7](#).

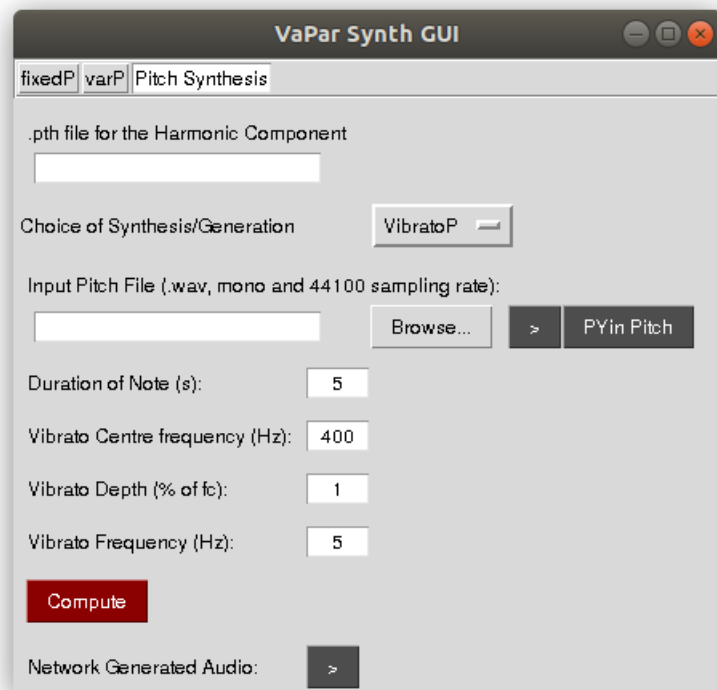


Figure 6.7: GUI Snapshot

A more simplistic GUI could be designed for musicians instead of researchers. It could take as input the midi notes directly from the instruments, and then render a synthesized accompaniment accordingly. Our GUI does three things,

1. *fixedP*: Reconstruction of fixed pitch notes
2. *varP*: Reconstruction of continuously varying pitch contour audio
3. *Generation*: Synthesis of notes in two ways; (1). Provide Vibrato parameters or (2). Provide an external monophonic audio file whose pitch contour will be extracted and a violin rendering corresponding to the extracted pitch will be synthesized.

In the first two, the GUI asks for the PyTorch trained network (the *.pth* file which contains the weights and network description), and the input audio which has to be reconstructed by the network. In the third case, you have to provide the external monophonic audio file whose pitch has to be extracted, or the vibrato parameters for the generated audio.

Chapter 7

Conclusion

The goal of this work was to explore autoencoder frameworks in generative models for audio synthesis of instrumental tones. We critically reviewed recent approaches and identified the problem of natural synthesis with flexible pitch control. We then presented VaPar Synth - our variational parametric model to generate audio. Through our parametric representation, we can decouple the ‘timbre’ and ‘pitch’, and can thus rely on the network to model the inter-dependencies. We use a variational model as it gives us the ability to directly sample points from the latent space. Moreover, by conditioning on the pitch, we can generate the learnt spectral envelope for that pitch (something which would not be possible in a vanilla VAE), thus giving us the power to vary the pitch contour continuously in principle. We then present a few experiments demonstrating the capabilities of our model. To the best of our knowledge, we have not come across any work using a parametric model for musical tones in the neural synthesis framework, especially exploiting the conditioning function of the CVAE. To aid in our analysis, we also introduce a new Carnatic Violin dataset. With the aid of this dataset, we highlight the necessity of pitch conditioning for the harmonic component. We also provide motivation to jointly model the harmonic and residual components instead of independently modeling them. We also present a GUI for researchers to interact with and understand our work more easily.

An immediate application of our work is in an automatic accompaniment system for singing voice, as depicted in [Figure 7.1](#). Consider you have a singer singing a song, and you wanted a musical instrument to accompany her/him. Common accompaniments in Indian Classical Music are the harmonium in Hindustani and the Violin in Carnatic Music. Thus, with the singer’s melodic pitch tracked, it would be possible to ‘generate’ an accompaniment for the song by giving this as input to a pre-trained model, as we showed in [Chapter 6](#).

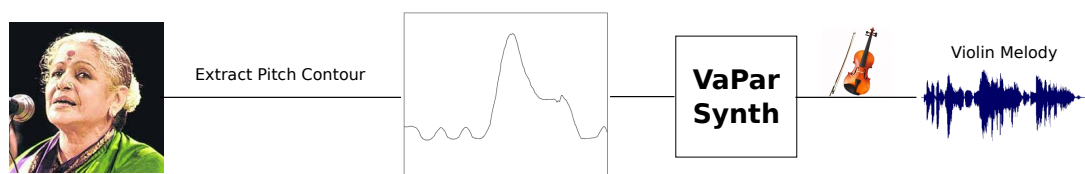


Figure 7.1: Accompaniment Generation

An accompaniment system solely based on the pitch contour will not capture the expressivity of an actual violin player. Hence, to capture this ‘expressivity’, there should be another block that can model this. That is simpler said than done, which highlights

one of the challenges involved in expressive audio synthesis.

Another application of our model could be in Source Separation. Instead of viewing Source Separation as a ‘Demixing’ problem where you separate out the individual sources, it could be alternately viewed as a source identification and reconstruction problem. The task thus boils down to identifying the sources and using generative models to reconstruct the sources. The potential advantage of this approach is the ability to separate the sources into high quality audio with the use of powerful generative models (high quality synthesis is very desirable for musical source separation!). Consider the task of separating the sources in a violin - mridangam mixture. Instead of learning to separate the two instruments, we could instead do the following; We could extract the pitch, residual and dynamic features corresponding to the violin, and then use our model to generate the violin audio. Thus, we effectively ‘separate’ out the violin from the violin-mridangam mixture.

A creative application of our work is in timbre conversion. We can train our model on different instruments, and then interpolate in the latent space to create ‘hybrid’ timbres. Or, we can simply use the pre-trained models to convert one instrument audio to another.

Our model does suffer from two main drawbacks, which are,

1. Our frame-wise modeling procedure does not take into account the temporal evolution of audio. By only modeling the sustain portion of the audio, we fail to take into account the attack and decay portions of audio, that might be perceptually relevant to certain instruments.
2. Generating high quality audio from the network is still a challenging task because of the sampling procedure we follow, which fails to synthesize a perceptually good harmonic and residual audio from the network.

Our main contributions are:

- Published and presented our work at ICASSP 2020 and ISMIR 2019 and submitted to ISMIR 2020. Refer to [List of Publications](#) for more details on the publications
- Documented code for our model, experiments and GUI open sourced on our GitHub repositories¹²
- Our Carnatic Violin Dataset, which we plan on making open source to the Music Information Retrieval research community

It was a really great experience to explore Generative Models for Audio Synthesis in this thesis. We hope that our dataset and code encourages further research in Carnatic music synthesis.

¹<https://github.com/SubramaniKrishna/VaPar-Synth>

²<https://github.com/SubramaniKrishna/HpRNet>

List of Publications

1. *VaPar Synth - A Variational Parametric Model for Audio Synthesis*

Published as a Conference Paper in the International Conference on Acoustics, Speech and Signal Processing (ICASSP) 2020 [54]

Abstract: With the advent of data-driven statistical modeling and abundant computing power, researchers are turning increasingly to deep learning for audio synthesis. These methods try to model audio signals directly in the time or frequency domain. In the interest of more flexible control over the generated sound, it could be more useful to work with a parametric representation of the signal which corresponds more directly to the musical attributes such as pitch, dynamics and timbre. We present VaPar Synth - a Variational Parametric Synthesizer which utilizes a conditional variational autoencoder (CVAE) trained on a suitable parametric representation. We demonstrate our proposed model’s capabilities via the reconstruction and generation of instrumental tones with flexible control over their pitch.

2. *HpRNet : Incorporating Residual Noise Modeling for Violin in a Variational Parametric Synthesizer*

Submitted as a Conference Paper to the International Society for Music Information Retrieval (ISMIR) 2020, Under Review [55]

Abstract: Generative Models for Audio Synthesis have been gaining momentum in the last few years. More recently, parametric representations of the audio signal have been incorporated to facilitate better musical control of the synthesized output. In this work, we investigate a parametric model for violin tones, in particular the generative modeling of the residual bow noise to make for more natural tone quality. To aid in our analysis, we introduce a dataset of Carnatic Violin Recordings where bow noise is an integral part of the playing style of higher pitched notes in specific gestural contexts. We obtain insights about each of the harmonic and residual components of the signal, as well as their interdependence, via observations on the latent space derived in the course of variational encoding of the spectral envelopes of the sustained sounds.

3. *Generative Audio Synthesis with a Parametric Model*

Presented extended abstract as a Poster at the International Society for Music Information Retrieval (ISMIR) 2019 [56]

Bibliography

- [1] M. V. Mathews, “The digital computer as a musical instrument,” *Science*, vol. 142, no. 3592, pp. 553–557, 1963.
- [2] J. O. Smith, “Physical modeling synthesis update,” *Computer Music Journal*, vol. 20, no. 2, pp. 44–56, 1996.
- [3] J. O. Smith, *Physical Audio Signal Processing*. <http://ccrma.stanford.edu/~jos/pasp/>, accessed 2020. online book, 2010 edition.
- [4] X. Serra, “A system for sound analysis/transformation/synthesis based on a deterministic plus stochastic decomposition,” *Ph.D. Thesis, Stanford University*, 1989.
- [5] X. Serra *et al.*, “Musical sound modeling with sinusoids plus noise,” *Musical signal processing*, pp. 91–122, 1997.
- [6] J. O. Smith, *Spectral Audio Signal Processing*. <http://ccrma.stanford.edu/~jos/sasp/>, accessed 2020. online book, 2011 edition.
- [7] C. Doersch, “Tutorial on variational autoencoders,” *arXiv preprint arXiv:1606.05908*, 2016.
- [8] C. Schörkhuber, A. Klapuri, N. Holighaus, and M. Dörfler, “A matlab toolbox for efficient perfect reconstruction time-frequency transforms with log-frequency resolution,” in *Audio Engineering Society Conference: 53rd International Conference: Semantic Audio*, Audio Engineering Society, 2014.
- [9] G. Tzanetakis, G. Essl, and P. Cook, “Audio analysis using the discrete wavelet transform,” in *Proc. Conf. in Acoustics and Music Theory Applications*, vol. 66, Citeseer, 2001.
- [10] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [11] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, “Wavenet: A generative model for raw audio,” *arXiv preprint arXiv:1609.03499*, 2016.
- [12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, pp. 2672–2680, 2014.
- [13] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *science*, vol. 313, no. 5786, pp. 504–507, 2006.

- [14] L. Wyse, “Real-valued parametric conditioning of an rnn for interactive sound synthesis,” *arXiv preprint arXiv:1805.10808*, 2018.
- [15] A. Défossez, N. Zeghidour, N. Usunier, L. Bottou, and F. Bach, “Sing: Symbol-to-instrument neural generator,” in *Advances in Neural Information Processing Systems*, pp. 9041–9051, 2018.
- [16] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, *et al.*, “Conditional image generation with pixelcnn decoders,” in *Advances in Neural Information Processing Systems*, pp. 4790–4798, 2016.
- [17] J. Engel, C. Resnick, A. Roberts, S. Dieleman, M. Norouzi, D. Eck, and K. Simonyan, “Neural audio synthesis of musical notes with wavenet autoencoders,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1068–1077, JMLR. org, 2017.
- [18] A. v. d. Oord, Y. Li, I. Babuschkin, K. Simonyan, O. Vinyals, K. Kavukcuoglu, G. v. d. Driessche, E. Lockhart, L. C. Cobo, F. Stimberg, *et al.*, “Parallel wavenet: Fast high-fidelity speech synthesis,” *arXiv preprint arXiv:1711.10433*, 2017.
- [19] C. Donahue, J. McAuley, and M. Puckette, “Adversarial audio synthesis,” *arXiv preprint arXiv:1802.04208*, 2018.
- [20] J. Engel, K. K. Agrawal, S. Chen, I. Gulrajani, C. Donahue, and A. Roberts, “Gansynth: Adversarial neural audio synthesis,” *arXiv preprint arXiv:1902.08710*, 2019.
- [21] D. Griffin and J. Lim, “Signal estimation from modified short-time fourier transform,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 2, pp. 236–243, 1984.
- [22] A. M. Sarroff and M. A. Casey, “Musical audio synthesis using autoencoding neural nets,” in *ICMC*, 2014.
- [23] F. Roche, T. Hueber, S. Limier, and L. Girin, “Autoencoders for music sound modeling: a comparison of linear, shallow, deep, recurrent and variational models,” *arXiv preprint arXiv:1806.04096*, 2018.
- [24] P. Esling, A. Bitton, *et al.*, “Generative timbre spaces: regularizing variational auto-encoders with perceptual metrics,” *arXiv preprint arXiv:1805.08501*, 2018.
- [25] A. Roebel and X. Rodet, “Efficient spectral envelope estimation and its application to pitch shifting and envelope preservation,” in *International Conference on Digital Audio Effects*, (Madrid, Spain), pp. 30–35, Sept. 2005. cote interne IRCAM: Roebel05b.
- [26] M. Blaauw and J. Bonada, “Modeling and transforming speech using variational autoencoders,” in *Interspeech*, pp. 1770–1774, 2016.
- [27] J. Engel, L. Hantrakul, C. Gu, and A. Roberts, “Ddsp: Differentiable digital signal processing,” *arXiv preprint arXiv:2001.04643*, 2020.

- [28] O. Romani Picas, H. Parra Rodriguez, D. Dabiri, H. Tokuda, W. Hariya, K. Oishi, and X. Serra, "A real-time system for measuring sound goodness in instrumental sounds," in *Audio Engineering Society Convention 138*, Audio Engineering Society, 2015.
- [29] C. Haigh, *Indian violin*, 2014 [Accessed: 21-Oct-2019]. <http://www.fiddlingaround.co.uk/india/>.
- [30] G. K. Koduri, J. Serrà Julià, and X. Serra, "Characterization of intonation in carnatic music by parametrizing pitch histograms," in *Gouyon F, Herrera P, Martins LG, Müller M. ISMIR 2012: Proceedings of the 13th International Society for Music Information Retrieval Conference; 2012 Oct 8-12; Porto, Portugal. Porto: FEUP Ediçoes, 2012.*, International Society for Music Information Retrieval (ISMIR), 2012.
- [31] G. N. Swift, "South indian "gamaka" and the violin," *Asian Music*, vol. 21, no. 2, pp. 71–89, 1990.
- [32] S. K. SUBRAMANIAN, "Modelling gamakas of carnatic music as a synthesizer for sparse prescriptive notation," 2013.
- [33] D. Bogdanov, N. Wack, E. Gómez Gutiérrez, S. Gulati, H. Boyer, O. Mayor, G. Roma Trepas, J. Salamon, J. R. Zapata González, X. Serra, *et al.*, "Essentia: An audio analysis library for music information retrieval," in *Britto A, Gouyon F, Dixon S, editors. 14th Conference of the International Society for Music Information Retrieval (ISMIR); 2013 Nov 4-8; Curitiba, Brazil.[place unknown]: ISMIR; 2013. p. 493-8.*, International Society for Music Information Retrieval (ISMIR), 2013.
- [34] M. Caetano and X. Rodet, "Musical instrument sound morphing guided by perceptually motivated features," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, no. 8, pp. 1666–1675, 2013.
- [35] W. Slawson, "The color of sound: a theoretical study in musical timbre," *Music Theory Spectrum*, vol. 3, pp. 132–141, 1981.
- [36] M. Caetano and X. Rodet, "A source-filter model for musical instrument sound transformation," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 137–140, IEEE, 2012.
- [37] S. IMAI, "Spectral envelope extraction by improved cepstrum," *IEICE*, vol. 62, pp. 217–228, 1979.
- [38] A. V. Oppenheim, "Speech analysis-synthesis system based on homomorphic filtering," *The Journal of the Acoustical Society of America*, vol. 45, no. 2, pp. 458–465, 1969.
- [39] R. McAulay and T. Quatieri, "Speech analysis/synthesis based on a sinusoidal representation," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 34, no. 4, pp. 744–754, 1986.
- [40] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

- [41] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, “beta-vae: Learning basic visual concepts with a constrained variational framework.,” *Iclr*, vol. 2, no. 5, p. 6, 2017.
- [42] K. Sohn, H. Lee, and X. Yan, “Learning structured output representation using deep conditional generative models,” in *Advances in neural information processing systems*, pp. 3483–3491, 2015.
- [43] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in PyTorch,” in *NIPS Autodiff Workshop*, 2017.
- [44] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [45] J. W. Beauchamp, “Comparison of vocal and violin vibrato with relationship to the source/filter model,” in *Studies in Musical Acoustics and Psychoacoustics*, pp. 201–221, Springer, 2017.
- [46] M. V. Mathews and J. Kohut, “Electronic simulation of violin resonances,” *The Journal of the Acoustical Society of America*, vol. 53, no. 6, pp. 1620–1626, 1973.
- [47] H. Fletcher and L. C. Sanders, “Quality of violin vibrato tones,” *The Journal of the Acoustical Society of America*, vol. 41, no. 6, pp. 1534–1544, 1967.
- [48] H. Fletcher, E. D. Blackham, and O. N. Geertsen, “Quality of violin, viola, cello, and bass-viol tones. i,” *The Journal of the Acoustical Society of America*, vol. 37, no. 5, pp. 851–863, 1965.
- [49] L. v. d. Maaten and G. Hinton, “Visualizing data using t-sne,” *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.
- [50] R. C. Maher and J. W. Beauchamp, “Fundamental frequency estimation of musical signals using a two-way mismatch procedure,” *The Journal of the Acoustical Society of America*, vol. 95, no. 4, pp. 2254–2263, 1994.
- [51] M. Mauch and S. Dixon, “pyin: A fundamental frequency estimator using probabilistic threshold distributions,” in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 659–663, IEEE, 2014.
- [52] A. De Cheveigné and H. Kawahara, “Yin, a fundamental frequency estimator for speech and music,” *The Journal of the Acoustical Society of America*, vol. 111, no. 4, pp. 1917–1930, 2002.
- [53] A. Pati, A. Lerch, and G. Hadjeres, “Learning to traverse latent spaces for musical score inpainting,” *arXiv preprint arXiv:1907.01164*, 2019.
- [54] K. Subramani, P. Rao, and A. D’Hooge, “Vapar synth - a variational parametric model for audio synthesis,” in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 796–800, 2020.
- [55] K. Subramani and P. Rao, “Hprnet : Incorporating residual noise modeling for violin in a variational parametric synthesizer,” *Submitted to ISMIR 2020, Under review*.

- [56] K. Subramani, A. D’Hooge, and P. Rao, “Generative audio synthesis with a parametric model,” *arXiv preprint arXiv:1911.08335*, 2019.