Audio Engineering Society

Convention Paper

Presented at the 149th Convention
2020 October 21 – 23, New York

Low Latency Timbre Interpolation and Morphing using Autoencoding Neural Networks

Joseph T Colonel[1] and Sam Keene[2]

[1]*Queen Mary University of London*
[2]*The Cooper Union*

Correspondence should be addressed to Joseph T Colonel (`j.t.colonel@qmul.ac.uk`)

ABSTRACT

A lightweight algorithm for low latency timbre interpolation of two input audio streams using an autoencoding neural network is presented. Short-time Fourier transform magnitude frames of each audio stream are encoded, and a new interpolated representation is created within the autoencoder's latent space. This new representation is passed to the decoder, which outputs a spectrogram. An initial phase estimation for the new spectrogram is calculated using the original phase of the two audio streams. Inversion to the time domain is done using a Griffin-Lim iteration. A method for avoiding pops between processed batches is discussed. An open source implementation in Python is made available.

1 Introduction

1.1 Neural Audio Effects and Synthesis

Deep neural networks have been increasingly used in audio effects modelling and sound synthesis [1] [2] [3]. As the field has improved and computation has become cheaper, networks have been developed to model increasingly complex audio effects and synthesizers [4] [5], and even synthesize entire songs [6].

A diversity of neural network architectures have been used in the field of sound synthesis. These models differ from models of audio effects as they do not perform a transformation of an input audio signal. Instead, audio is generated via other means. For example, generative adversarial networks have been used for spectrogram based and raw audio based synthesis [7] [8]. The SampleRNN architecture uses an autoregressive model to generate new audio sample-by-sample [3]. In another model, differentiable digital signal processing (DDSP) can directly synthesize new audio using modules such as harmonic oscillators, wavetable synthesizers, and convolutional reverbs [9]. The network presented in [2] uses WaveNet autoencoders to create a latent space which contains various timbres. Timbre can be described as the perceptual quality of a musical sound that is distinct from its loudness and pitch [10]. The WaveNet synthesizes timbres by outputting raw audio sample-by-sample based on an input latent sampling, and can output interpolations of two timbres by sampling from a line connecting the latent embeddings of the two timbres. Another implementation outlined in [11] creates a timbre embedding space using a Mel2Mel neural network architecture and synthesizes audio using a WaveNet vocoder.

Other implementations have used variational autoencoders, or VAEs, on such tasks as audio synthesis via a parameter space [12], or synthesis via a timbre space

[13]. The latter modifies the Kullback–Leibler divergence term used to train a VAE so that the latent space mirrors a distribution of a perceptually constructed timbre space. The best performing model encodes timbre using a non-stationary Gabor transform [14], with the Equivalent Rectangular Bandwidth scale (ERB scale [15]) of 400 bins outperforming all other models. Interpolation between timbres can be performed by sampling along a path in the latent space. Since this embedding space is 64-dimensional, a user can only traverse it via dimension reduction, whereby the 64 dimensions are projected to three.

Similarly, many different neural network architectures have been used as audio effects. These models are all trained to apply effects to arbitrary musical signals and produce modified audio, but will not synthesize new audio without an input. Some neural network models of audio effects use convolutional neural networks to digitally model analog effects, such as compressors [16], distortion [17], spring reverb [18], and equalization [19].

1.2 Timbre Transfer and Morphing

One effect which has seen widespread implementation using neural networks is timbre transfer. A timbre transfer audio effect takes an input signal with one timbre, such as a recording of a violin playing a C note, and outputs a signal with a different timbre, such as an electric guitar playing a C note. DDSP modules have been shown capable of performing timbre transfer via explicit modelling of pitch tracking, loudness curves, and a latent representation of timbre [9]. Similarly, [20] uses one WaveNet encoder to generate a latent representation of an input sound that can be decoded via one of many trained WaveNet decoders to transfer timbre.

Non-deep learning approaches to timbre transfer and sound morphing can also be found in the literature. The authors of [21] specify two analysis and synthesis algorithms to interpolate between two signals' timbres. The first algorithm only uses spectral interpolation of the magnitude spectra, and the second also uses attack information to modify a sustain synthesized by the spectral interpolation. In [22], automatic audio morphing is performed by mapping two timbres into a multi-dimensional space encoding spectral shape and pitch on orthogonal axes. Mapping is performed using mel-frequency cepstral coefficients (MFCCs), and synthesis is performed using spectral inversion. Using a framework similar to [22], the authors of [23] specify an algorithm for morphing transient sounds using discrete wavelet transforms and singular value decomposition. The authors of [24] and [25] based their interpolation algorithm on acoustic correlates of salient timbre dimensions derived from perceptual studies, including log attack time, temporal centroid, and measures of spectral shape.

Two algorithms which most closely resemble the work presented in this paper are [26] and [27]. The neural network architecture presented in [26] utilizes a standard autoencoding neural network with a long short-term memory (LSTM) bottleneck to perform timbre modification and transfer. This LSTM layer is implemented in order to model the temporal dynamics of a given input. The network is trained to encode and decode both the magnitude and phase of three adjacent frames of an input sound's short-time Fourier transform (STFT). Resynthesis is performed using the inverse STFT (ISTFT) of the network's output magnitude and phase predictions. In order to perform timbre transfer, the authors begin by training an autoencoder on a corpus containing only one timbre (i.e. trumpet sounds). Then, an audio sample with a different timbre is passed through the trained network (i.e. a woman's singing voice). Thus the authors transfer the timbral characteristics of a trumpet onto a woman's singing voice.

The work in [27] presents an algorithm for portamento between arbitrary audio sources which eschews neural networks entirely. Instead, an optimal transport is calculated to transform one spectrum into another. As the authors describe it, optimal transport is an optimization problem in which one distribution is transformed into another distribution using the least "amount of work (mass times distance) on each infinitesimal piece of mass." The authors introduce an interpolation parameter as well, which can sample spectra along the optimal transport route between two input sounds. Thus the authors can produce a "portamento" between a piano being struck and a person singing.

1.3 Contributions

Our previous work has focused on the standard multi-layer neural network autoencoder [28] [29] [30], as its lightweight implementation can put as much of the design process into the hands of musicians. By "lightweight," we mean a model that can be trained in a reasonable amount of time and which does not require powerful GPUs to train or infer. For example, an author of [9] has mentioned that the paper's timbre transfer topology takes "a few hours to train on [an Nvidia Tesla V100]" for a corpus of 14 minutes [31]. In comparison, our network presented in [29] takes eight minutes to train on a corpus eight minutes in length on a laptop's CPU.

The contributions of this paper are as follows. In this paper we outline an algorithm that can perform interpolation and morphing in a timbre space of two streams of input audio with low latency using an autoencoder. This autoencoder constructs a latent space containing representations of timbre by learning to encode and decode STFT magnitude frames. Timbre interpolation and morphing can be performed by sampling points in the latent space between those of the two input streams on a frame-by-frame basis. Moreover, this latent representation can be morphed using multiplicative gains on the latent representation. Our algorithm differs from [27] as we use a neural network's latent space containing representations of timbre rather than an optimal transport to interpolate between sounds. Our algorithm differs from [26] as it can interpolate timbre between two input sounds rather than only transferring the timbre of a training corpus to a new input. Furthermore we provide open-source code for our algorithm that runs online with low latency, whereas [27] and [26] do not. We have open-sourced this algorithm in Python with accompanying demonstrations, and made it available on github at `github.com/JTColonel/timbre-interp`.

When designing neural networks for creative purposes one must strike a three-way balance between the expressivity of the system, the freedom given to a user to train and interface with the network, and the computational overhead needed for sound synthesis. Given the improvements of neural network APIs over the past few years, it is now possible for musicians to train and run a standard autoencoder in a reasonable amount of time on a laptop, without needing a GPU. It is our hope that engaging musicians in the design process of these networks will lead to innovation in the field of neural audio effects and synthesis.

2 Background

2.1 Autoencoders

An autoencoding neural network (i.e. autoencoder) is a machine learning algorithm that is typically used for unsupervised learning of an encoding scheme for a given input domain, and is comprised of an encoder and a decoder [32]. For the purposes of this work, the encoder is forced to shrink the dimension of an input into a latent space using a discrete number of values, or "neurons." The decoder then expands the dimension of the latent space to that of the input, in a manner that reconstructs the original input.

In a single layer model, the encoder maps an input vector $x \in \mathbb{R}^d$ to the hidden layer $y \in \mathbb{R}^e$, where $d > e$. Then, the decoder maps y to $\hat{x} \in \mathbb{R}^d$. In this formulation, the encoder maps $x \to y$ via

$$y = f(Wx + b) \tag{1}$$

where $W \in \mathbb{R}^{(e \times d)}$, $b \in \mathbb{R}^e$, and $f(\cdot)$ is an activation function that imposes a non-linearity in the neural network. The decoder has a similar formulation:

$$\hat{x} = f(W_{\text{out}}y + b_{\text{out}}) \tag{2}$$

with $W_{\text{out}} \in \mathbb{R}^{(d \times e)}$, $b_{out} \in \mathbb{R}^d$.

A multi-layer autoencoder acts in much the same way as a single-layer autoencoder. The encoder contains $n > 1$ layers and the decoder contains $m > 1$ layers. Using Equation 1 for each mapping, the encoder maps $x \to x_1 \to \ldots \to x_n$. Treating x_n as y in Equation 2, the decoder maps $x_n \to x_{n+1} \to \ldots \to x_{n+m} = \hat{x}$.

The autoencoder trains the weights of the W's and b's to minimize some cost function. This cost function should minimize the distance between input and output values. The choice of activation functions $f(\cdot)$ and cost functions depends on the domain of a given task.

2.2 Activations

The sigmoid function

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

and rectified linear unit (ReLU)

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{4}$$

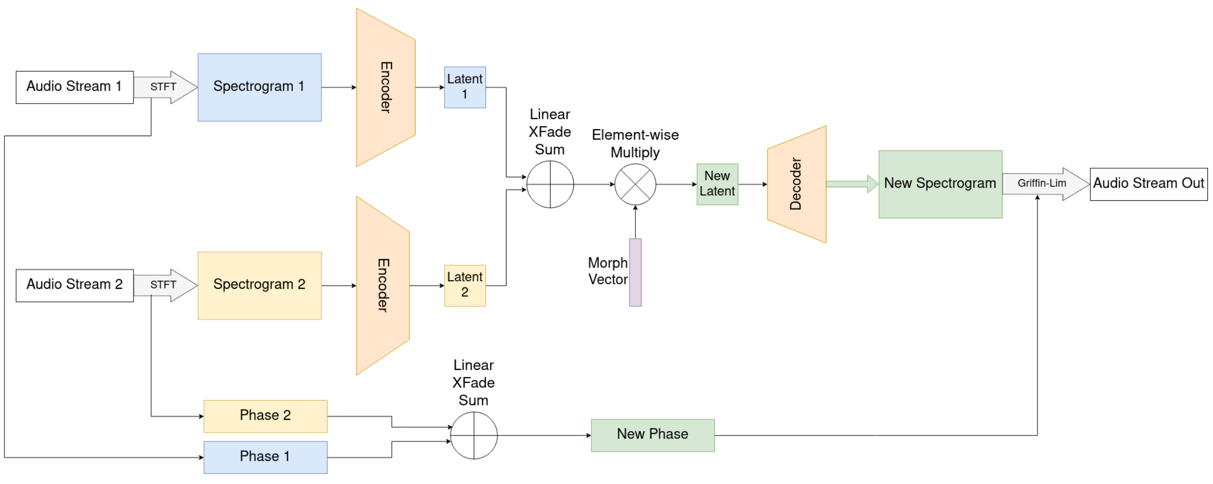


Fig. 1: *Signal flow diagram of the timbre interpolation and morphing algorithm*

are often used to impose the nonlinearities $f(\cdot)$ in an autoencoding neural network. A hybrid autoencoder topology combining both sigmoid and ReLU activations was shown to outperform all-sigmoid and all-ReLU models in a timbre encoding task [28].

More recently, the leaky rectified linear unit (LReLU) [33]

$$f(x) = \begin{cases} \beta x, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{5}$$

has been shown to avoid both the vanishing gradient problem introduced by using the sigmoid activation [34] and the dying neuron problem introduced by using the ReLU activation [35]. The hyperparameter β is typically small, and in this work fixed at 0.1. In [30], we demonstrated that a hybrid model imposing ReLU activations on the output, sigmoid activations in the latent space, and LReLU activations elsewhere encouraged an autoencoder to produce a well-distributed latent embedding of an input corpus. This is beneficial because it creates a well populated latent space for an arbitrary corpus without a potentially inexperienced user having to tune hyperparameters.

2.3 Training Regime

In this work a multi-layer neural network autoencoder is trained to learn representations of musical timbre. The aim is to train the autoencoder to contain high level descriptive features in a low dimensional latent space that can be easily manipulated by a musician. As in the formulation above, dimension reduction is imposed at each layer of the encoder until the desired dimensionality is reached. All audio used to generate the corpora for this work is stored as a 16-bit PCM wav file with 44.1kHz sampling rate.

The various corpora used to train the autoencoding neural network are formed by taking 2049 values from a 4096-point magnitude STFT $s_n(m)$ as its target, where n denotes the frame index of the STFT and m denotes the frequency index, with 75% frame overlap. The 2049 values span the frequency bins from 0Hz to half the Nyquist frequency, in this case 22050Hz. The Hann window is used in all cases. Each frame is normalized to $[0,1]$. This normalization allows the autoencoder to ignore a frame's peak level relative to other observations within the corpus. In this work mean squared error is used as the cost function, although both spectral convergence and mean absolute error have also been investigated in previous work [29].

3 Timbre Interpolation Algorithm

In short, the timbre interpolation algorithm works by encoding normalized spectrograms of two input audio streams into a timbre space, decoding a new spectrogram from interpolated and morphed points in the latent space, and inverting the new spectrogram into a time domain signal. Refer to Figure 1 for a signal flow diagram.

Algorithm 1: Timbre Interpolation Algorithm for n^{th} chunk of audio

```
input  : Audio batches $a_1$,$a_2$, previous output
         audio chunk $a_{prev}[n-1:n]$, interpolation
         value $\alpha$, morphing vector $w$, encoding
         map $E$, decoding map $D$
output : Interpolated audio $a_{new}[n:n+1]$
$a_{prev}[n] \leftarrow xfade_{out}(a_{prev}[n])$ ;
while Make Next Frame do
    $batch_1 \leftarrow a_1[n-3:n+2]$ ;
    $batch_2 \leftarrow a_2[n-3:n+2]$ ;
    $S_1 \leftarrow STFT(batch_1)$ ;
    $max_1 \leftarrow max(S_1)$ ;
    $S_1 \leftarrow S_1/max_1$ ;
    $S_2 \leftarrow STFT(batch_2)$ ;
    $max_2 \leftarrow max(S_2)$ ;
    $S_2 \leftarrow S_2/max_2$ ;
    $s_1 \leftarrow \alpha w \odot E(S_1)$;
    $s_2 \leftarrow (1-\alpha) w \odot E(S_2)$;
    $s_{new} \leftarrow s_1 + s_2$ ;
    $S_{new} \leftarrow (\alpha max_1 + (1-\alpha) max_2) D(s_{new})$ ;
    $\angle S_{new} \leftarrow wrap(\alpha \angle S_1 + (1-\alpha) \angle S_2)$ ;
    repeat
        $a_{new} \leftarrow Griffin\_Lim(S_{new}, \angle S_{new})$ ;
    until convergence criterion;
end
$a_{new}[n] \leftarrow xfade_{in}(a_{new}[n]) + a_{prev}[n]$ ;
$a_{prev} \leftarrow a_{new}$ ;
return
```

To describe this algorithm, assume an autoencoder with a d dimensional latent space, two audio signals a_1 and a_2 with STFTs S_1 and S_2, morphing vector $w \in \mathbb{R}^d$, and interpolation value α.

Assume we want to output a *batch* of audio that is composed of N *chunks* of 1024 samples. Due to the 75% overlap used when performing the STFT, the algorithm requires two batches of audio size $(N+5) \times 1024$ samples from the two input audio streams. The algorithm begins by performing a STFT on each batch of the two audio streams. 3 look-back and 1 look-forward chunks are necessary to perform 4098 point STFTs with hop size 1024 samples and Hann window on the $N \times 1024$ samples of audio, and the additional 1 look-forward frame is used to eliminate pops between batches. The magnitude and phase response are separated for each STFT. Then, the magnitude response is normalized to $[0,1]$ for each frame in preparation for the autoencoder. These normalizing gains and phase responses are stored for later use.

S_1 and S_2 are encoded into two representations s_1 and s_2 of size $d \times N$. A new representation is calculated using α and w such that

$$s_{new} = w \odot [\alpha s_1 + (1-\alpha) s_2] \tag{6}$$

where $\odot$ is an element-wise multiplication across each latent dimension. The α parameter determines which point along the line that connects s_1 and s_2 within the latent space gets sampled and passed to the decoder, with $\alpha = 0$ just passing s_1 to the decoder and $\alpha = 1$ just passing s_2 to the decoder. The morphing vector w is an additional effect that can be applied to the latent representation of the input. When each element of w is set to 1, the representation is unmodified. Otherwise, the element-wise multiplication of w stretches $\alpha s_1 + (1-\alpha)s_2$ towards an arbitrary region in the latent space, thus morphing the timbre.

s_{new} is then decoded to synthesize S_{new}. Note that each frame of S_{new} only takes values from $[0,1]$, thus to transfer the dynamic qualities of a_1 and a_2,

$$S_{new} \leftarrow (\alpha ||S_1||_\infty + (1-\alpha)||S_2||_\infty) \odot S_{new} \tag{7}$$

where $||\cdot||_\infty$ refers to the L-infinity norm and $\odot$ refers to an element-wise multiplication across time slices of S.

Finally, S_{new} is inverted into the time domain using the Griffin-Lim algorithm initialized with a phase of

$$\angle S_{new} = wrap(\alpha \angle S_1 + (1-\alpha)\angle S_2) \tag{8}$$

This phase initialization is informed by complex addition, in which the phase of the sum of two complex numbers falls on the closed interval bound by the phase of each number. Furthermore, this linear sum of coherent phase responses from two real audio signals produces a coherent phase initialization across all frames.

The output of the Griffin-Lim iterations is an $(N+4) \times 1024$ batch of synthesized audio a_{new}. The first $3*1024$ samples are dropped due to the 75% overlap used in this work, leaving $(N+1) \times 1024$ samples.

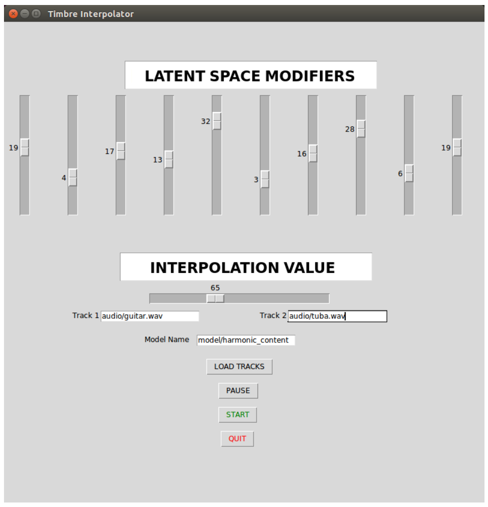
To avoid pops in between two adjacent batches of $N \times 1024$ samples, a linear crossfade is applied to the $N+1^{st}$ frame of the previous batch and the 1^{st} frame of the current batch.

4 Implementation

We provide an open-source implementation of our algorithm in Python, as well as additional code to train an autoencoder on a user's dataset. Python bindings for Port Audio are used to handle the callback function for audio output. Keras with a TensorFlow backend is used to handle the neural network. Scipy is used for the Griffin-Lim iterations. The GUI is coded using TKinter. The code is available at `github.com/JTColonel/timbre-interp`. A screenshot of our GUI can be found in Figure 2.

To begin, a user trains an autoencoder on a target corpus. Once trained, a model is saved in Keras such that it takes two batches of normalized spectrogram frames, an interpolation scalar, and multiplicative gain constants. These tensors, as well as two copies of the encoder weights locally connected, are pre-allocated and pre-broadcast to improve execution time. After the user chooses two audio files to load into memory, the network begins to process the audio streams and output a new audio stream. The user can choose an interpolation that "crossfades" the two audio streams within the latent space, and modify this new latent representation with gain constants set by the vertical sliders. The actual gain constants applied to the latent representation are the number presented next to each slider divided by 10, such that they range from $[0,3]$.

Currently, our implementation takes two saved audio files as the audio streams. While we have not implemented it by the time of writing, we believe the algorithm will function with an input from a recorded audio stream, such as a microphone.



19 4 17 13 32 3 16 28 6 19

65

Track 1 audio/guitar.wav

Track 2 audio/tuba.wav

Model Name model/harmonic_content

Fig. 2: *Screenshot of the timbre interpolation GUI. The 10 sliders correspond to gain multipliers sent to each of the 10 dimensions of the latent embedding.*

5 Performance

Our current implementation processes a batch of 5 chunks at a time, with each chunk being 1024 samples. At 44.1kHz sampling rate, this results in a maximum of 10ms of latency between user input and processed audio being output. The network only polls the α and w parameters once to begin processing a batch; in other words, the α and w parameters do not vary within a batch.

The loading of a trained model and instantiation of its graph requires approximately 500MB of RAM with encoder widths $2049 \rightarrow 256 \rightarrow 128 \rightarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow 10$ and mirrored decoder widths. Additional layers between the 2049 wide layer and 256 wide layers would improve the network's reconstruction performance [28], at the cost of increased RAM usage. The authors note that the majority of the weight parameters in this topology are produced by the first layer of the encoder (and last layer of the decoder). For example, reducing the 2049 dimensional input to 256 dimensions requires about 1 million weight parameters between the encoder and decoder; whereas reducing from 2049 to 1024 requires over 4 million weight parameters.

Figure 4 illustrates the spectra produced when two streams of audio with the same fundamental frequency but distinct timbres are processed by the interpolation algorithm when varying α. From top to bottom, α takes values $\{0, 0.2, 0.4, 0.6, 0.8, 1\}$. Figure 5 illustrates the spectra of the same two streams of audio processed by the interpolation algorithm, but with a warping vector $\{1.29, 0.73, 1.33, 0.86, 1.62, 0.54, 1.32, 1.33, 1.82, 0.2\}$ applied. In both cases, the algorithm retains the fundamental frequency while altering harmonics. For this example, the autoencoder was trained on the one octave dataset mentioned in [30], and thus was tasked with encoding a latent space with single note timbres.

Figure 3 illustrates the waveforms produced when a stream of audio with stationary timbre and another stream of a drum loop are processed by the interpolation algorithm when varying α. From top to bottom, α takes values $\{0, 0.25, 0.5, 0.75, 1\}$. This demonstrates that the interpolation tool can interpolate dynamics as well. Because this autoencoder was trained on the one octave dataset, the $\alpha = 1$ output has more harmonic content than the unprocessed drum loop.

Recordings of all the examples shown here can be found at `github.com/JTColonel/timbre-interp`.

6 Conclusion

We outline an algorithm for low latency interpolation of timbre between two audio streams using an autoencoding neural network. The autoencoder is trained to encode and decode normalized short-time Fourier transform magnitude frames. The timbre interpolation algorithm encodes two spectrograms from two audio streams and calculates a new encoding by interpolation. This new encoding is passed to the decoder, which produces a new spectrogram. This spectrogram is inverted to the time domain using a Griffin-Lim algorithm. This Griffin-Lim iteration is initialized with a weighted sum of the two input audio streams' phase responses. A short linear crossfade is applied between two processed batches to avoid pops and clicks. Our implementation is purely Python and open-source. The architecture is lightweight, allowing musicians to train the autoencoder on their own corpus and run the algorithm in a live setting without the need for powerful GPUs.

References

[1] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K., "WaveNet: A Generative Model for Raw Audio," in *9th ISCA Speech Synthesis Workshop*, pp. 125–125, 2016.

[2] Engel, J., Resnick, C., Roberts, A., Dieleman, S., Norouzi, M., Eck, D., and Simonyan, K., "Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders," in D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1068–1077, PMLR, International Convention Centre, Sydney, Australia, 2017.

[3] Mehri, S., Kumar, K., Gulrajani, I., Kumar, R., Jain, S., Sotelo, J., Courville, A., and Bengio, Y., "SampleRNN: An unconditional end-to-end neural audio generation model," in *5th International Conference on Learning Representations, ICLR*, 2017.

[4] Huang, S., Li, Q., Anil, C., Bao, X., Oore, S., and Grosse, R. B., "TimbreTron: A WaveNet(CycleGAN(CQT(Audio))) Pipeline for Musical Timbre Transfer," in *7th International Conference on Learning Representations, ICLR*, 2019.

[5] Luo, Y.-J., Agres, K., and Herremans, D., "Learning Disentangled Representations of Timbre and Pitch for Musical Instrument Sounds Using Gaussian Mixture Variational Autoencoders," in *Proceedings of the 20th ISMIR Conference*, 2019.

[6] Dhariwal, P., Jun, H., Payne, C., Kim, J. W., Radford, A., and Sutskever, I., "Jukebox: A Generative Model for Music," 2020.

[7] Engel, J., Agrawal, K. K., Chen, S., Gulrajani, I., Donahue, C., and Roberts, A., "GANSynth: Adversarial Neural Audio Synthesis," in *International Conference on Learning Representations*, 2018.

[8] Donahue, C., McAuley, J., and Puckette, M., "Adversarial Audio Synthesis," in *International Conference on Learning Representations*, 2018.

[9] Engel, J., Gu, C., Roberts, A., et al., "DDSP: Differentiable Digital Signal Processing," in *International Conference on Learning Representations*, 2019.

[10] Grey, J. M. and Gordon, J. W., "Perceptual effects of spectral modifications on musical timbres," *The Journal of the Acoustical Society of America*, 63(5), pp. 1493–1500, 1978.

[11] Kim, J. W., Bittner, R., Kumar, A., and Bello, J. P., "Neural music synthesis for flexible timbre control," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 176–180, IEEE, 2019.

[12] Esling, P., Masuda, N., Bardet, A., Despres, R., Chemla, A., et al., "Universal audio synthesizer control with normalizing flows," in *International Conference on Digital Audio Effects (DaFX 2019)*, 2019.

[13] Esling, P., Chemla-Romeu-Santos, A., and Bitton, A., "Generative timbre spaces with variational audio synthesis," in *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, 2018.

[14] Balazs, P., Dörfler, M., Jaillet, F., Holighaus, N., and Velasco, G., "Theory, implementation and applications of nonstationary Gabor frames," *Journal of computational and applied mathematics*, 236(6), pp. 1481–1496, 2011.

[15] Moore, B. C. and Glasberg, B. R., "Suggested formulae for calculating auditory-filter bandwidths and excitation patterns," *The journal of the acoustical society of America*, 74(3), pp. 750–753, 1983.

[16] Hawley, S., Colburn, B., and Mimilakis, S. I., "Profiling Audio Compressors with Deep Neural Networks," in *Audio Engineering Society Convention 147*, Audio Engineering Society, 2019.

[17] Ramírez, M. A. M. and Reiss, J. D., "Modeling nonlinear audio effects with end-to-end deep neural networks," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 171–175, IEEE, 2019.

[18] Martinez Ramirez, M. A., Benetos, E., and Reiss, J. D., "Modeling Plate and Spring Reverberation Using A DSP-Informed Deep Neural Network," *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, doi:10.1109/icassp40776.2020.9053093.

[19] Ramírez, M. A. M. and Reiss, J. D., "End-to-end equalization with convolutional neural networks," in *21st International Conference on Digital Audio Effects (DAFx-18)*, 2018.

[20] Mor, N., Wolf, L., Polyak, A., and Taigman, Y., "A Universal Music Translation Network," 2018.

[21] Serra, M.-H., Rubine, D., and Dannenberg, R., "Analysis and synthesis of tones by spectral interpolation," *Journal of the Audio Engineering Society*, 38(3), pp. 111–128, 1990.

[22] Slaney, M., Covell, M., and Lassiter, B., "Automatic audio morphing," in *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, volume 2, pp. 1001–1004, IEEE, 1996.

[23] Ahmad, W., Hacihabiboglu, H., and Kondoz, A. M., "Morphing of transient sounds based on shift-invariant discrete wavelet transform and singular value decomposition," in *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 297–300, IEEE, 2009.

[24] Caetano, M. and Rodet, X., "Sound morphing by feature interpolation," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 161–164, IEEE, 2011.

[25] Caetano, M. and Rodet, X., "Musical instrument sound morphing guided by perceptually motivated features," *IEEE Transactions on Audio, Speech, and Language Processing*, 21(8), pp. 1666–1675, 2013.

[26] Gabrielli, L., Cella, C. E., Vesperini, F., Droghini, D., Principi, E., and Squartini, S., "Deep learning for timbre modification and transfer: An evaluation study," in *Audio Engineering Society Convention 144*, Audio Engineering Society, 2018.

[27] Henderson, T. and Solomon, J., "Audio Transport: A Generalized Portamento via Optimal Transport,"

in *International Conference on Digital Audio Effects*, 2019.

[28] Colonel, J., Curro, C., and Keene, S., "Improving Neural Net Auto Encoders for Music Synthesis," in *Audio Engineering Society Convention 143*, 2017.

[29] Colonel, J., Curro, C., and Keene, S., "Neural Network Autoencoders as Musical Audio Synthesizers," *Proceedings of the 21st International Conference on Digital Audio Effects (DAFx-18). Aveiro, Portugal*, 2018.

[30] Colonel, J. T. and Keene, S., "Conditioning Autoencoder Latent Spaces for Real-Time Timbre Interpolation and Synthesis," *arXiv preprint arXiv:2001.11296*, 2020.

[31] Engel, J., "10/ One of the coolest things about using these priors is it takes much less data and compute to do ML with audio. All examples here use less than 13 minutes of data and a few hours on a V100. Hanoi could even train a model on his own Salo instrument," 2020, *https://twitter.com/jesseengel/status/1217567007383511041*.

[32] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A., "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, 11(Dec), pp. 3371–3408, 2010.

[33] Maas, A. L., Hannun, A. Y., and Ng, A. Y., "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, volume 30, p. 3, 2013.

[34] Hochreiter, S., Bengio, Y., Frasconi, P., and Schmidhuber, J., "Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies," 2001.

[35] Lu, L., Shin, Y., Su, Y., and Karniadakis, G. E., "Dying ReLU and Initialization: Theory and Numerical Examples," *arXiv preprint arXiv:1903.06733*, 2019.

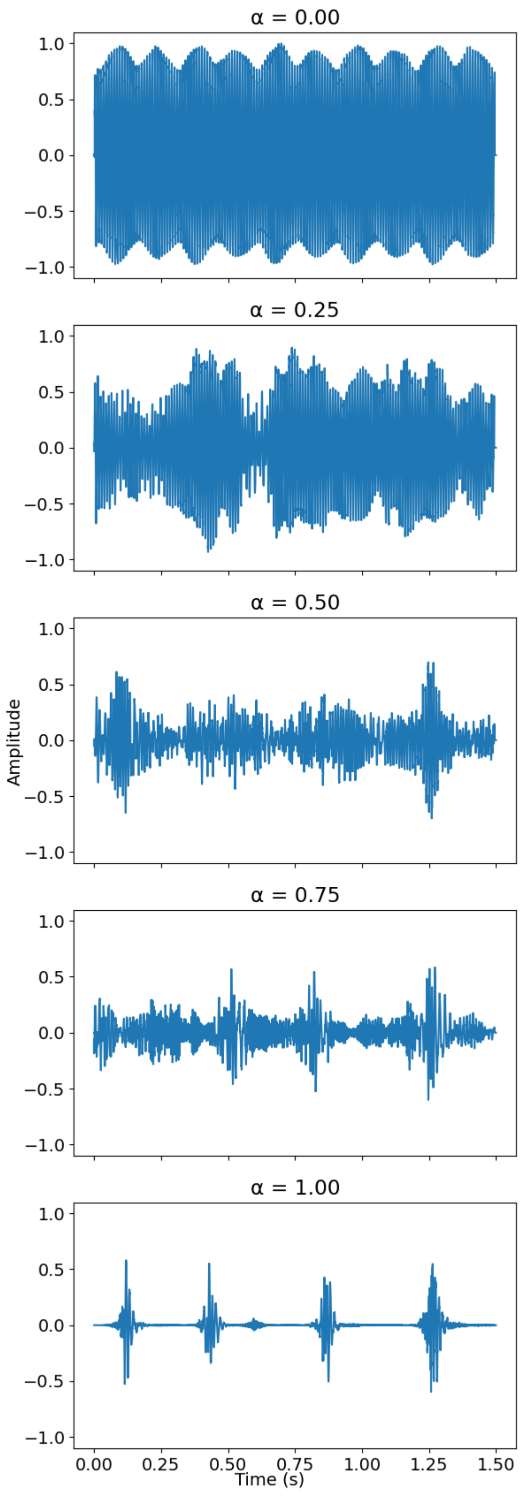


Fig. 3: *Snapshots of the waveforms produced by interpolating between an audio stream of stationary timbre and a drum loop.*

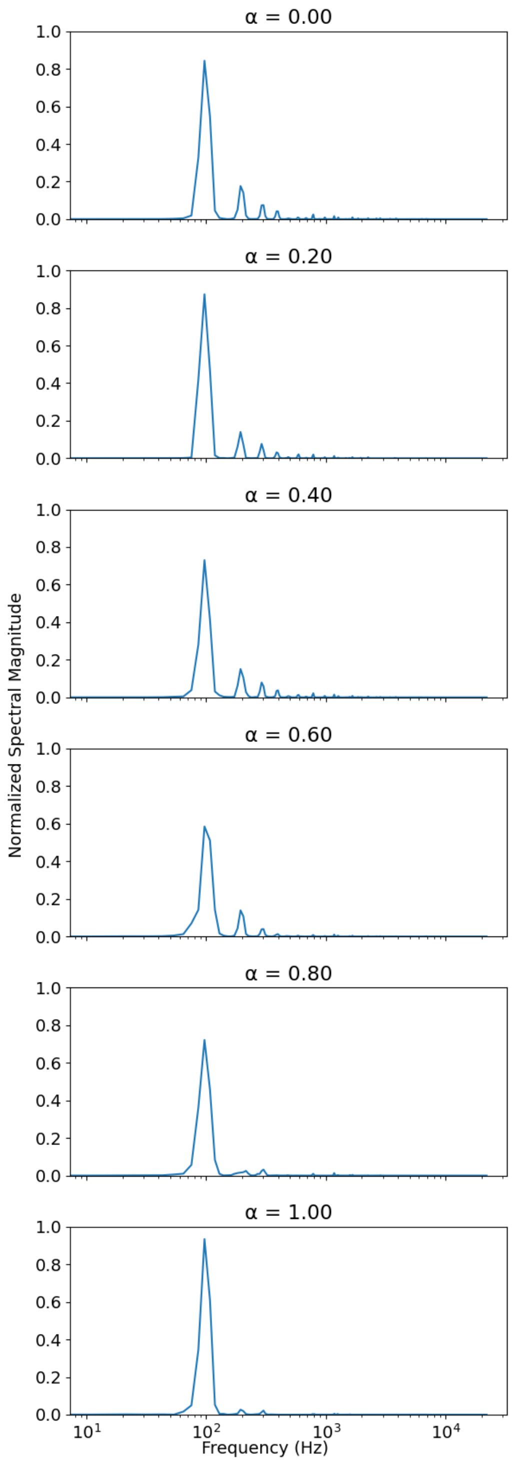


Fig. 4: Snapshots of the spectra that are produced when interpolating between two stationary timbres.

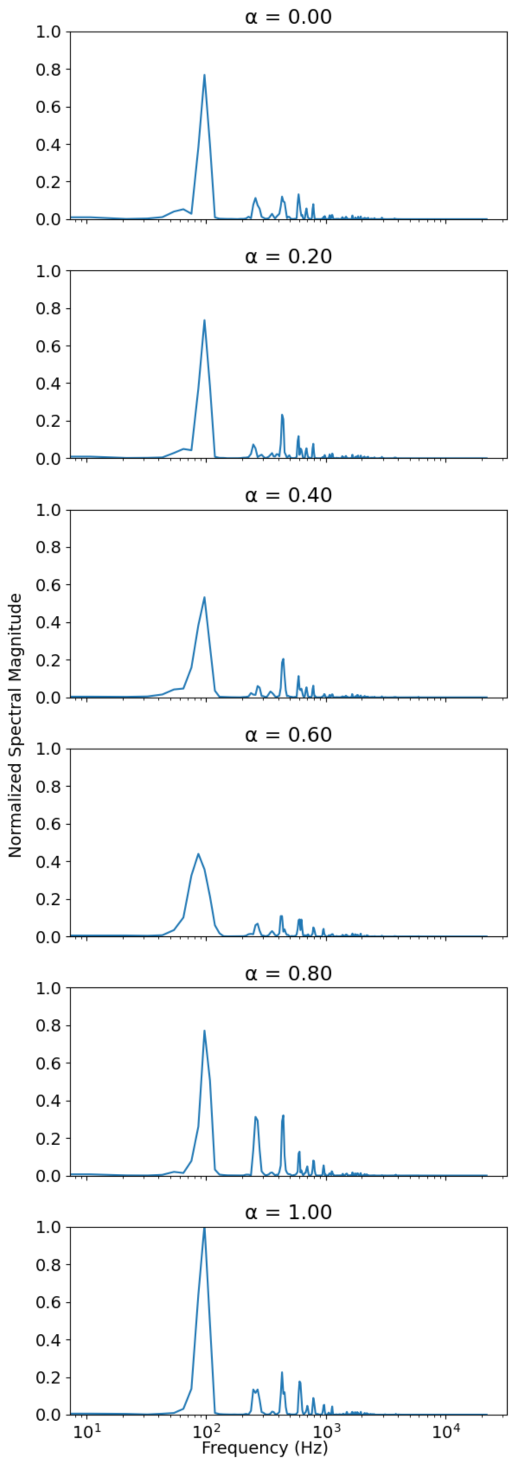


Fig. 5: Spectra produced by interpolating and warping between the two stationary timbres in Fig 4.