Conditioning Autoencoder Latent Spaces for Real-Time Timbre Interpolation and Synthesis

Joseph T Colonel
Queen Mary University of London
London, UK
j.t.colonel@qmul.ac.uk

Sam Keene
Cooper Union
New York, USA
keene@cooper.edu

***Abstract*—We compare standard autoencoder topologies' performances for timbre generation. We demonstrate how different activation functions used in the autoencoder's bottleneck distributes a training corpus's embedding. We show that the choice of sigmoid activation in the bottleneck produces a more bounded and uniformly distributed embedding than a leaky rectified linear unit activation. We propose a one-hot encoded chroma feature vector for use in both input augmentation and latent space conditioning. We measure the performance of these networks, and characterize the latent embeddings that arise from the use of this chroma conditioning vector. An open source, real-time timbre synthesis algorithm in Python is outlined and shared.**

***Index Terms*—neural network, autoencoder, timbre synthesis, real-time audio**

I. Introduction

Timbre refers to the perceptual qualities of a musical sound distinct from its amplitude and pitch. It is timbre that allows a listener to distinguish between a guitar and a violin both producing a concert C note. Moreover, a musician's ability to create, control, and exploit new timbres has led, in part, to the surge in popularity of pop, electronic, and hip hop music.

New algorithms for timbre generation and sound synthesis have accompanied the rise to prominence of artificial neural networks. GANSynth [10], trained on the NSynth dataset [12], uses generative adversarial networks to output high-fidelity, locally coherent outputs. Furthermore, GANSynth's global latent conditioning allows for interpolations between two timbres. Other works have found success in using Variational Autoencoders (VAEs) [13] [5] [4], which combine autoencoders and probabilistic inference to generate new audio. Most recently, differential digital signal processing has shown promise by casting common modules used in signal processing into a differentiable form [11], where they can be trained with neural networks using stochastic optimization.

The complexity of these models require powerful computing hardware to train, hardware often out of reach for musicians and creatives. 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. One successful example we point to in the field of music composition is MidiMe [9], which allows a composer to train a VAE with their own scores on a subspace of a larger, more powerful model. Moreover, these training computations take place on the end user's browser.

Our previous work has tried to strike this three-way balance as well [7] [8], by utilizing feed-forward neural network autoencoder architectures trained on Short-Time Fourier Transform (STFT) magnitude frames. This work demonstrated how choice of activation functions, corpora, and augmentations to the autoencoder's input could improve performance for timbre generation. However, we found upon testing that the autoencoder's latent space proved difficult to control and characterize. Also, we found that our use of a five-octave MicroKORG corpus encouraged the autoencoder to produce high-pitched, often uncomfortable tones.

This paper introduces a chroma-based input augmentation and skip connection to help improve our autoencoder's reconstruction performance with little additional training time. A one-octave MicroKORG corpus as well as a violin-based corpus are used to train and compare various architectural tweaks. Moreover, we show how this skip connection conditions the autoencoder's latent space so that a musician can shape a timbre around a desired note class. A full characterization of the autoencoder's latent space is provided by sampling from meshes that span the latent space. Finally, a real-time, responsive implementation of this architecture is outlined and made available in Python.

II. Autoencoding Neural Networks

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 [25]. 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 $x \rightarrow y$ via

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

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

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 $x \rightarrow x_1 \rightarrow \ldots \rightarrow x_n$. Treating x_n as y in Equation 2, the decoder maps $x_n \rightarrow x_{n+1} \rightarrow \ldots \rightarrow x_{n+m} = \hat{x}$.

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.

III. Network Design and Topology

We build off of previous work to present our current network architecture.

A. Activations

The sigmoid function

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

and rectified linear unit (ReLU)

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

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 [7]. However, this hybrid model often would not converge for a deeper autoencoder [8].

More recently, the leaky rectified linear unit (LReLU) [21]

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

has been shown to avoid both the vanishing gradient problem introduced by using the sigmoid activation [16] and the dying neuron problem introduced by using the ReLU activation [19]. The hyperparameter α is typically small, and in this work fixed at 0.1.

B. Chroma Based Input Augmentation

The work presented in [8] showed how appending descriptive features to the input of an autoencoder can improve reconstruction performance, at the cost of increased training time. More specifically, appending the first-order difference of the training example to the input was shown to give the best reconstruction performance, at the cost of doubling the training time. Here, we suggest a basic chroma-based feature augmentation to help the autoencoder.

Chroma-based features capture the harmonic information of an input sound by projecting the input's frequency content onto a set of chroma values [22]. Assuming a twelve-interval equal temperment Western music scale, these chroma values form the set {C, C#, D, D#, E, F, F#, G, G#, A, A#, B}. A chromagram can be calculated by decomposing an input sound into 88 frequency bands corresponding to the musical notes A0 to C8. Summing the short-time mean-square power across N frames for each sub-band across each note (i.e. A0-A7) yields a $12 \times N$ chromagram.

In this work, a one-hot encoded chroma representation is calculated for each training example by taking its chromagram, setting the maximum chroma value to 1, and setting every other chroma value to 0. While this reduces to note conditioning in the case of single-note audio, this generalizes to the dominant frequency of a chord or polyphonic mixture. Furthermore this feature can be calculated on an arbitrary corpus, which eliminates the tedious process of annotating by hand.

C. Hidden Layers and Bottleneck Skip Connection

This work uses a slight modification of the geometrically decreasing/increasing autoencoder topology proposed in [8]. All layers aside from the bottleneck and output layers use the LReLU activation function. The output layer uses the ReLU, as all training examples in the corpus take strictly non-negative values. For the bottleneck layer, models are trained separately with all-LReLU and all-sigmoid activations to compare how each activation constructs a latent space.

The 2048-point first-order difference input augmentation is replaced with the 12-point one-hot encoded chroma feature explained above. Furthermore, in this work three separate topologies are explored by varying the bottleneck layer's width – one with two neurons, one with three neurons, and one with eight neurons.

Residual and skip connections are used in autoencoder design to help improve performance as network depth increases [14]. In this work, the 12-point one-hot encoded chroma feature input augmentation is passed directly to the autoencoder's bottleneck layer. Models with and without this skip connection are trained to compare how the skip connection affects the autoencoder's latent space. Figure 1 depicts our architecture with the chroma skip connection and eight neuron latent space. Note that for our two neuron model, the $8 \times N$ latent embedding would become a $2 \times N$ latent embedding, and similarly would become a $3 \times N$ for our three neuron model.

IV. Corpora

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

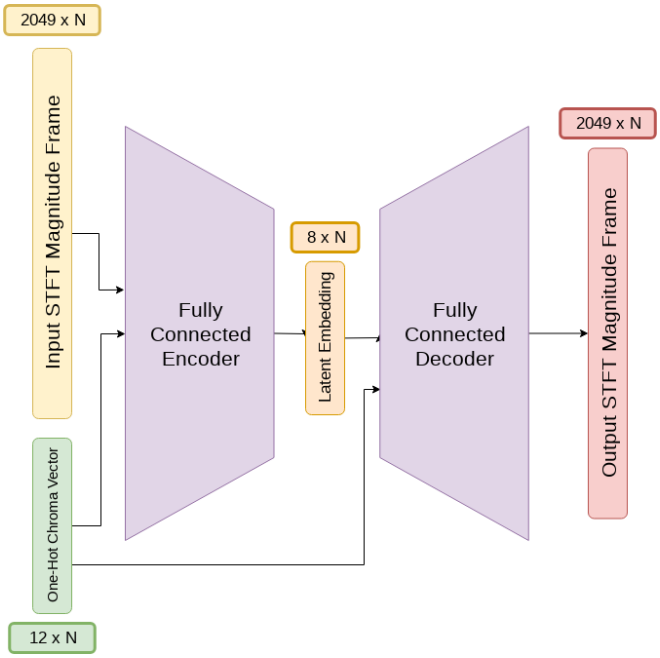

Fig. 1: Network diagram with eight neuron bottleneck and chroma skip connection

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 points 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 Hann window is used in all cases. Each frame is normalized to $[0, 1]$. This normalization tasks the autoencoder to encode solely the timbre of an input observation and ignore its loudness relative to other observations within the corpus. These corpora were not mixed for training; models were only trained on each corpus separately.

A. MicroKORG Dataset

Two corpora were created by recording C Major scales from a MicroKORG synthesizer/vocoder. In both cases, 70 patches make up the training set, 5 patches make up the validation set, and 5 patches make up the test set. These patches ensured that different timbres are present in the corpus. To ensure the integrity of the testing and validation sets, the dataset is split on the "clip" level. This means that the frames in each of the three sets are generated from distinct passages in the recording, which prevents duplicate or nearly duplicate frames from appearing across the three sets.

The first corpus is comprised of approximately $79,000$ magnitude STFT frames, with an additional $6,000$ frames held out for validation and another $6,000$ for testing. This makes the corpus $91,000$ frames in total, or roughly 35 minutes of audio. The audio used to generate these frames is composed of five octave C Major scales recorded from a MicroKORG synthesizer/vocoder across 80 patches.

Input Augmentation	Test Set MSE	Training Time
No Append	3.185×10^{-4}	35 minutes
1^{st} Order Diff	3.001×10^{-4}	44 minutes
One-Hot Chroma	2.920×10^{-4}	37 minutes

TABLE I: Five Octave Dataset Autoencoder holdout set MSE loss and training time

Bottleneck Activation	Skip?	Test Set MSE
LReLU	No	3.516×10^{-4}
LReLU	Yes	3.598×10^{-4}
Sigmoid	No	3.358×10^{-4}
Sigmoid	Yes	3.472×10^{-4}

TABLE II: One Octave Dataset Autoencoder holdout set MSE loss, 2 neuron bottleneck

The second corpus is a subset of the first. It is comprised of one octave C Major scales starting from concert C. Approximately $17,000$ frames make up the training set, with an additional $1,000$ frames held out for validation and another $1,000$ for testing. This makes the subset $19,000$ frames in total, or roughly 7 minutes of audio.

By restricting the corpus to single notes played on a MicroKORG, the autoencoder needs only to learn higher level features of harmonic synthesizer content. These tones often have time variant timbres and effects, such as echo and overdrive. Thus the autoencoder is also tasked with learning high level representations of these effects.

B. TU-Note Violin Sample Library

A third corpus was created using a portion of the TU-Note Violin Sample Library [26]. The dataset consists of recordings of a violin in an anechoic chamber playing single sounds, two-note sequences, and solo performances such as scales and compositions. The single notes were used to construct a training corpus, and the solo performances were cut into two parts to form the validation and test sets. These two parts were split on the "clip" level to ensure that no frames from the same passages were found across the validation and test sets. Approximately $91,000$ frames make up the training set, with an additional $10,000$ frames held out for validation and another $10,000$ for testing. This makes the subset $111,000$ frames in total, or roughly 43 minutes of audio. Here, the autoencoder is tasked with learning the difference in timbre one can here when a violin is played at different dynamic levels, semitones, and with different stroke techniques.

C. Training Setup

All models were trained for 300 epochs using the ADAM method for stochastic optimization [17], initialized with a learning rate of 5×10^{-4}. Mean squared error was used as the cost function, with an L2 penalty of 10^{-7} [18]. All training utilized one NVIDIA Quadro P2000 GPU, and all networks were implemented using Keras 2.2.4 [6] with Tensorflow-GPU 1.9.0 [1] as a backend.

Bottleneck Activation	Skip?	Test Set MSE	Training Time
LReLU	No	7.731×10^{-4}	41 minutes
LReLU	Yes	6.478×10^{-4}	58 minutes
Sigmoid	No	8.900×10^{-4}	43 minutes
Sigmoid	Yes	6.388×10^{-4}	43 minutes

TABLE III: TU-Note Violin Sample Library Dataset Autoencoder holdout set MSE loss and training time, two neuron bottleneck

Corpus	Skip?	Test Set MSE	Training Time
One Octave	No	3.228×10^{-4}	8 minutes
One Octave	Yes	3.055×10^{-4}	8 minutes
Violin	No	8.545×10^{-4}	44 minutes
Violin	Yes	5.763×10^{-4}	45 minutes

TABLE IV: Sigmoid Bottleneck Autoencoder holdout set MSE loss and training time, three neuron bottleneck

V. Results

Table I shows the performance of three autoencoders with an eight neuron bottleneck layer using LReLU activations trained on the five octave MicroKORG corpus. The model with the chroma augmentation outperforms both the first-order difference augmentation and no augmentation models. Moreover, the chroma augmentation only increases training time by two minutes. Therefore, the rest of the models in this work utilize the chroma input augmentation.

Table II show the performance of four autoencoders with a two neuron bottleneck layer trained on the one octave MicroKORG corpus. Models used either the LReLU or sigmoid activation for the bottleneck, and either did or did not utilize a chroma skip connection. All models took eight minutes to train. Both sigmoid models outperformed each LReLU model, and the sigmoid model with no skip connection performed the best.

Table III show the performance of four autoencoders with a two neuron bottleneck layer trained on the TU-Note Violin Sample Library corpus. Models used either the LReLU or sigmoid activation for the bottleneck, and either did or did not utilize a chroma skip connection. With this corpus, the chroma skip connection significantly improved the reconstruction error for both sigmoid and LReLU activations. Furthermore, the sigmoid activation with the chroma skip connection outperformed all models.

With these results in mind, two models were trained on the one octave MicroKORG corpus using a three neuron bottleneck with sigmoid activations: one with the chroma skip connection, and one without. Two more models with corresponding topologies were trained on the TU-Note Violin Sample Library corpus. Table IV shows the reconstruction performance of each model. In this case, the models with the chroma skip connection outperformed the models without.

A. Latent Embeddings

When designing an autoencoder for musicians to use in timbre synthesis, it is important not only to measure the network's reconstruction error, but also to characterize the latent space's embedding. The software synthesizer implemented in [8] allows a musician to chose a point in the autoencoder's latent space and generate its corresponding timbre. By exploring the latent space, the musician can explore an embedding of timbre.

A clear understanding of the boundedness of an embedding ensures that a musician can fully explore the latent space of an arbitrary training corpus, and a clear understanding of the density of the latent embedding can help a musician avoid portions of the latent space that will generate unrealistic examples while interpolating between two encoded timbres [24] [27].

Recent work has attempted to encourage an autoencoder to interpolate in a "semantically continuous" manner [3]. The authors sample from their autoencoder's latent space along a line that connects two points to demonstrate this meaningful interpolation. The authors also characterize their latent space using a method proposed by [28], where an unsupervised clustering accuracy is measured to see how well ground truth labels are separated in the latent space. In the case of our work, however, we are less concerned with how clusters separate in the latent space and more concerned with how uniform samplings of the latent space produce note classes and timbres.

We begin with a visual inspection of the training set embeddings produced by the eight distinct autoencoders referred to in Tables II and III. Figure 2 shows the embeddings for the one octave MicroKORG corpus, and Figure 3 shows the embeddings for the TU-Note Violin Sample Library corpus. Models trained with the LReLU bottleneck activation are plotted in the top row, and models trained with the sigmoid bottleneck activation are plotted in the bottom row. Models trained without the chroma skip connection are plotted in the left column, and models trained with the chroma skip connection are plotted in the right column. Each note class is plotted as one color (i.e. C is dark blue, F is teal, B is yellow) using a perceptually uniform colormap.

In all cases, the chroma skip connection appears to encourage the embedding to be denser and contain fewer striations. Note that by definition, all models with sigmoid activations are bounded by $(0, 1)$. On the other hand, the models with LReLU activation vary their bounds greatly. Moreover, the first and second dimensions of the LReLU embeddings appear to have linear correlations, rather than populating the latent space in a more uniform manner. As such we move forward using only sigmoid activations at the bottleneck.

A full accounting of the two neuron sigmoid bottleneck autoencoder's latent space is shown in Figures 4 and 5. These graphs were created by setting the chroma conditioning vector to a given note class, and then sampling the autoencoder's latent space using a 350×350 point mesh grid. Each note class is plotted as one color (i.e. C is dark blue, F is teal, B is yellow) using a perceptually uniform colormap. We observe that the autoencoder is able to use the majority of the alloted two dimensions to produce timbres that match the conditioned chroma vector. We note that most mismatches occur near the boundaries of the latent space. We suspect this may be caused by the asymptotic behavior of the sigmoid function coupled

with the L2 penalty encouraging the network to choose smaller weights, though a full characterization is outside the scope of this paper.

This mesh sampling procedure was repeated for the three neuron and eight neuron sigmoid bottleneck models. Due to compuational constraints, the three neuron model used a $50 \times 50 \times 50$ mesh and the eight neuron model used a $5 \times 5 \times ... \times 5$ mesh. The accuracies of the model samplings are shown in Table V. We suspect that some of the decreases in prediction accuracy as the number of neurons in the bottleneck increases may be due in part to the coarser meshes over-weighing samplings near the boundaries of the latent space, though a full characterization is outside the scope of this paper.

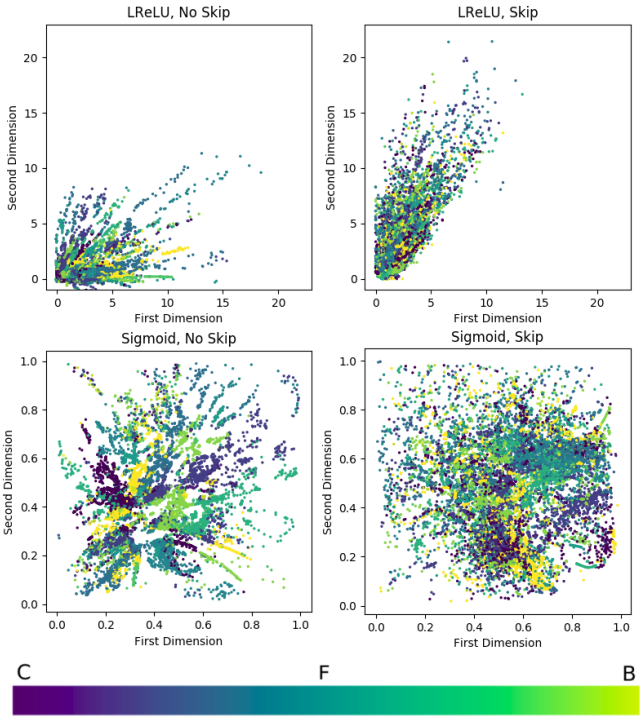

Fig. 2: 2D embeddings of the One Octave MicroKORG Corpus

VI. Python Implementation

As outlined in [8], a spectrogram with no phase information can be generated via bypassing the network's encoder and passing latent activations to the decoder. To generate the true phase of this spectrogram, the real-time phase gradient heap integration algorithm can be used [23]. However, to decrease the computational overhead involved in our algorithm, we store the stripped the phase of a white noise audio signal and use it to invert the generated spectrogram.

Our implementation is purely Python, using Tkinter as our GUI backend. Once a user selects a trained decoder to sample from, Keras loads the model into memory. The user is presented with sliders that correspond to each neuron in the model's bottleneck, and a twelve-value radio button is used

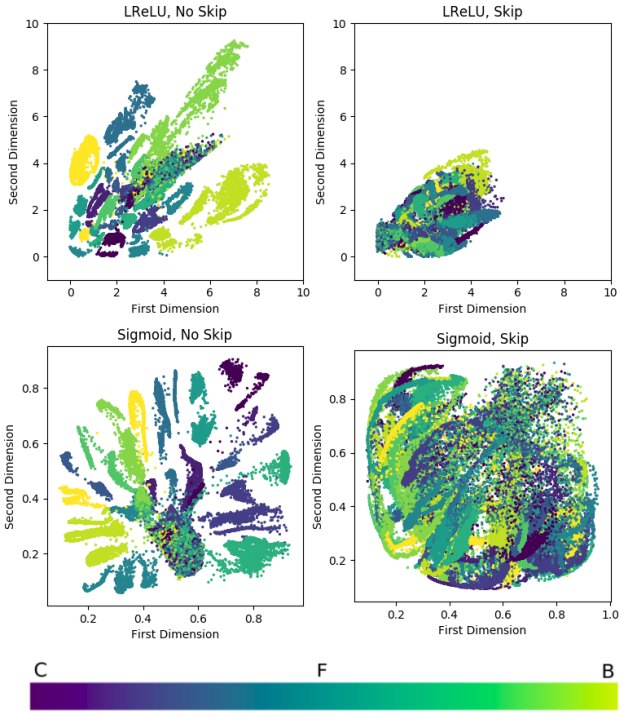

Fig. 3: 2D embeddings of the TU-Note Violin Sample Library Corpus

to set the chroma conditioning vector. The Pyaudio library provides Python bindings to PortAudio [2], and handles the audio stream output.

Our implementation has been made available at *github.com/JTColonel/manne*, along with code to create a corpus from an audio file for training, code to train a model, and code to plot the samplings of a model's latent space. We have tested our implementation on a laptop with an Intel Core i7-8750H CPU @ 2.20GHz × 12 with 16GB of RAM.

We also provide code to train and sample from Variational Autoencoder implementation (specifically a β-VAE [15]), with a word of caution. We found that all of our trained models exhibited posterior collapse [20], wherein the variational distribution would closely match the uninformative prior for a subset of latent variables, and the rest of the latent variables would output high mean, low variance samplings. Moreover, we did not find that the non-conditioned β-VAE disentangled the note class from timbre. We found that the note class would change when varying any one latent dimension while fixing the rest. Unfortunately a full treatment of this behavior is outside the scope of this paper.

VII. Conclusion

We present an improved neural network autoencoder topology and training regime for use in timbre synthesis and interpolation. By using a one-hot encoded chroma vector as both an augmentation to the autoencoder's input and a conditioning vector at the autoencoder's bottleneck, we improve the autoencoder's reconstruction error at little additional computational

Model	Mesh Length	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
2D One Octave	350	.896	–	.941	–	.981	.884	–	.908	–	.863	–	.937
3D One Octave	50	.827	–	.857	–	.847	.953	–	.851	–	.904	–	.933
8D One Octave	5	.741	–	.650	–	.949	.526	–	.605	–	.834	–	.660
2D Violin	350	.980	.840	.941	.902	.999	.846	.905	.864	.892	.992	.884	.984
3D Violin	50	.942	.629	.999	.692	.999	.997	.659	.980	.999	.999	.885	.977
8D Violin	5	.837	.833	.950	.844	.814	.805	.956	.782	.820	.844	.805	.920

TABLE V: Percent of sampled outputs matching conditioned chroma skip vector

cost. Moreover, we characterize how this conditioning vector shapes the autoencoder's usage of its latent space. We provide an open source implementation of our architecture in Python, which can sample from its latent space in real-time without the need for powerful computing hardware.

REFERENCES

[1] M. Abadi, "Tensorflow: Learning functions at scale," *ICFP*, 2016. [Online]. Available: http://dl.acm.org/citation.cfm?id=2976746

[2] R. Bencina and P. Burk, "Portaudio - an open source cross platform audio api," in *ICMC*, 2001.

[3] D. Berthelot, C. Raffel, A. Roy, and I. Goodfellow, "Understanding and improving interpolation in autoencoders via an adversarial regularizer," *arXiv preprint arXiv:1807.07543*, 2018.

[4] A. Bitton, P. Esling, A. Caillon, and M. Fouilleul, "Assisted sound sample generation with musical conditioning in adversarial auto-encoders," 2019.

[5] A. Chemla-Romeu-Santos, S. Ntalampiras, P. Esling, G. Haus, and G. Assayag, "Cross-modal variational inference for bijective signal-symbol translation," 2019.

[6] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[7] J. Colonel, C. Curro, and S. Keene, "Improving neural net auto encoders for music synthesis," in *Audio Engineering Society Convention 143*, Oct 2017. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=19243

[8] ——, "Neural network autoencoders as musical audio synthesizers," *Proceedings of the 21st International Conference on Digital Audio Effects (DAFx-18). Aveiro, Portugal*, 2018.

[9] M. Dinculescu, J. Engel, and A. Roberts, Eds., *MidiMe: Personalizing a MusicVAE model with user data*, 2019.

[10] J. Engel, K. K. Agrawal, S. Chen, I. Gulrajani, C. Donahue, and A. Roberts, "Gansynth: Adversarial neural audio synthesis," 2019. [Online]. Available: https://openreview.net/pdf?id=H1xQVn09FX

[11] J. Engel, L. H. Hantrakul, C. Gu, and A. Roberts, "Ddsp: Differentiable digital signal processing," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=B1x1ma4tDr

[12] J. Engel, C. Resnick, A. Roberts, S. Dieleman, D. Eck, K. Simonyan, and M. Norouzi, "Neural audio synthesis of musical notes with wavenet autoencoders," 2017.

[13] P. Esling, N. Masuda, A. Bardet, R. Despres, and A. Chemla-Romeu-Santos, "Universal audio synthesizer control with normalizing flows," 2019.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[15] 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," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017. [Online]. Available: https://openreview.net/forum?id=Sy2fzU9gl

[16] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[18] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *NIPS*, vol. 4, 1991, pp. 950–957.

[19] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis, "Dying relu and initialization: Theory and numerical examples," *arXiv preprint arXiv:1903.06733*, 2019.

[20] J. Lucas, G. Tucker, R. B. Grosse, and M. Norouzi, "Understanding posterior collapse in generative latent variable models," in *DGS@ICLR*, 2019.

[21] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, no. 1, 2013, p. 3.

[22] M. Müller, F. Kurth, and M. Clausen, "Audio matching via chroma-based statistical features." in *ISMIR*, vol. 2005, 2005, p. 6th.

[23] Z. Pruša and P. L. Søndergaard, "Real-time spectrogram inversion using phase gradient heap integration," in *Proc. Int. Conf. Digital Audio Effects (DAFx-16)*, 2016, pp. 17–21.

[24] V. Verma, A. Lamb, C. Beckham, A. Najafi, I. Mitliagkas, A. Courville, D. Lopez-Paz, and Y. Bengio, "Manifold mixup: Better representations by interpolating hidden states," *arXiv preprint arXiv:1806.05236*, 2018.

[25] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, no. Dec, pp. 3371–3408, 2010.

[26] H. Von Coler, "Tu-note violin sample library—a database of violin sounds with segmentation ground truth," 2018.

[27] T. White, "Sampling generative networks," *arXiv preprint arXiv:1609.04468*, 2016.

[28] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *International conference on machine learning*, 2016, pp. 478–487.

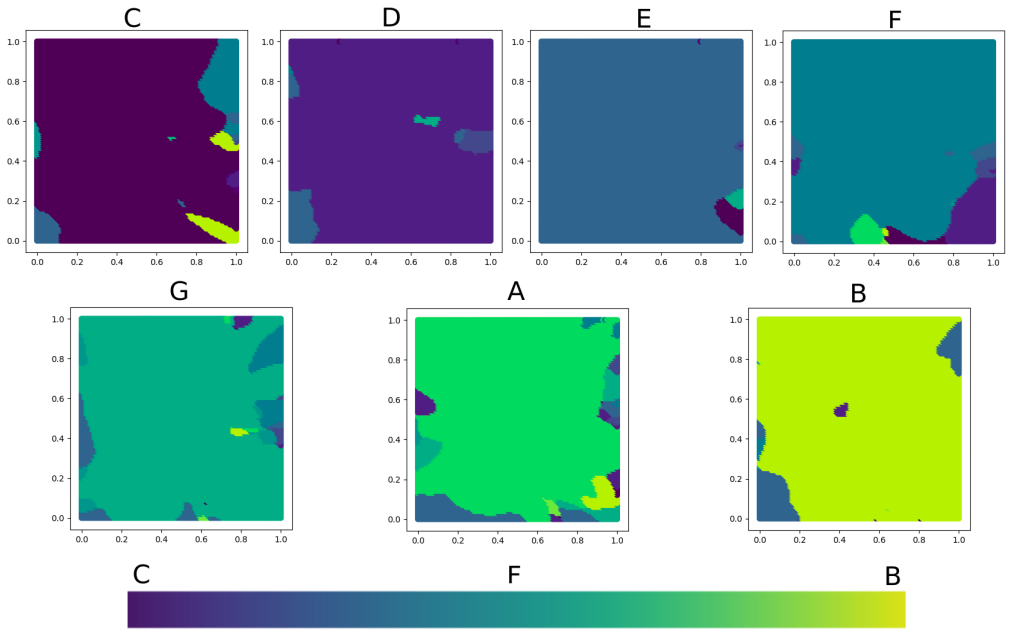

Fig. 4: 2D embeddings of the One Octave Corpus

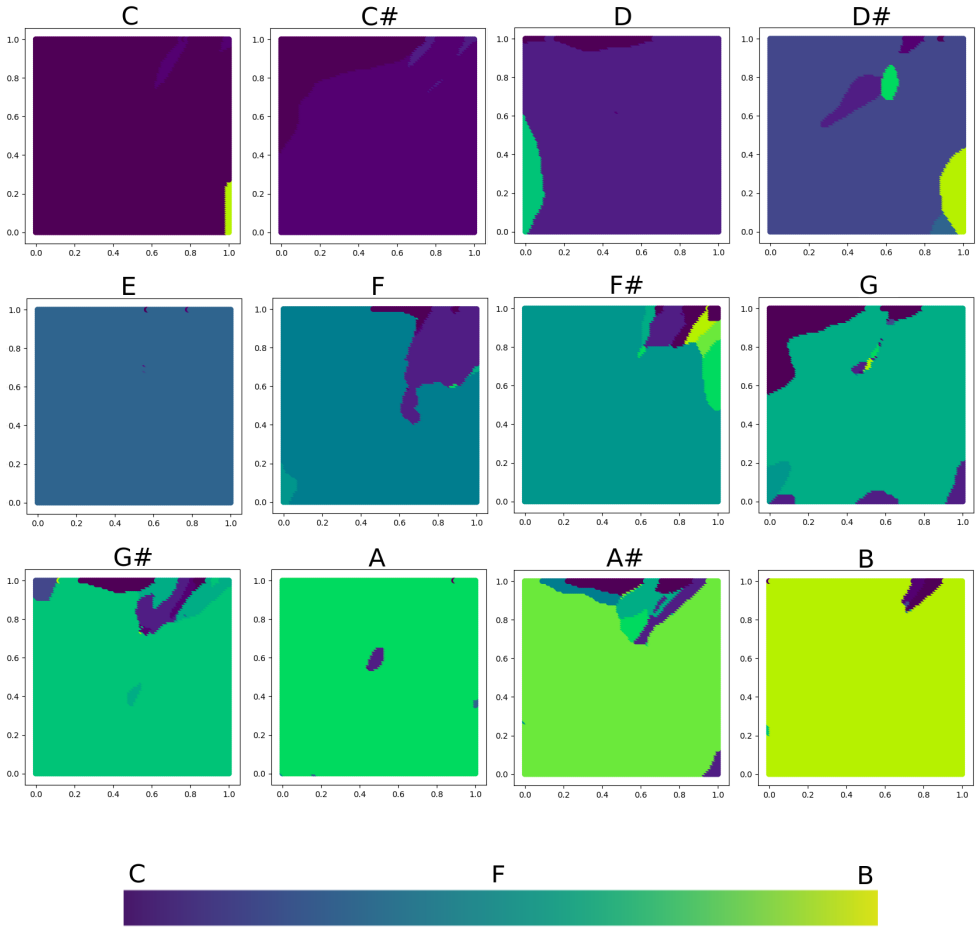

Fig. 5: 2D embeddings of the TU-Note Violin Sample Library Corpus