DrumGAN VST: A Plugin for Drum Sound Analysis/Synthesis with Autoencoding Generative Adversarial Networks

Anonymous Authors[1]

Abstract

In contemporary popular music production, drum sound design is commonly performed by cumbersome browsing and processing of pre-recorded samples in sound libraries. One can also use specialized synthesis hardware, typically controlled through low-level, musically meaningless parameters. Today, the field of Deep Learning offers methods to control the synthesis process via learned high-level features and allows generating a wide variety of sounds. In this paper, we present *DrumGAN VST*, a plugin for synthesizing drum sounds using a Generative Adversarial Network. DrumGAN VST operates on 44.1 kHz sample-rate audio, offers independent and continuous instrument class controls, and features an encoding neural network that maps sounds into the GAN's latent space, enabling resynthesis and manipulation of pre-existing drum sounds. We provide numerous sound examples and a demo of the proposed VST plugin.[1]

1. Introduction

Drum sound design plays a prominent role in arranging and producing a song in most contemporary popular music. Thanks to synthesizers and the availability of professionally-recorded sound libraries, producers can directly process drum samples in a computer or specialized hardware (e.g., in drum machines). However, the timbre diversity offered by sample packs or synthesizers is limited, and producers are forced to perform complex sound superpositions (i.e., layering) and envelope processing techniques to create variations. Interaction and exploration are other critical issues with such techniques, as browsing in sample packs or fiddling with expert controls in a synthesizer can be a barrier to creativity for some people.

Fueled by recent advances in Deep Learning (DL), the field of neural audio synthesis has shown the potential to overcome some of the inconveniences mentioned above. Many deep generative models can learn high-level latent variables that provide more expressive and intuitive means for sound exploration and synthesis (Nistal et al., 2020; Drysdale et al., 2021; Donahue et al., 2019). In addition, as DL models can be trained on arbitrary data, the sound diversity is not strictly limited to that of a particular synthesis process. Despite the unprecedented success of many of these methods in experimental settings, just a handful of works have culminated into production-ready tools for music creation (e.g., Mawf,[2] Neurorack (Devis & Esling, 2021)). The challenge of modeling high-quality raw audio at scale (Dieleman et al., 2018), coupled with the requirement for such music creation tools to operate in real-time in resource-limited environments, has remained the thorn in the side for their use in commercial applications.

In this work, we present *DrumGAN VST*, a drum sound synthesis plugin based on a prior work employing Generative Adversarial Networks (GANs) (Nistal et al., 2020). Driven by feedback from professional artists and music production standards, we perform a series of improvements on the original model: i) 44.1 kHz sample-rate audio operability; ii) continuous instrument control; iii) encoding-decoding of sounds to generate variations. Additionally, in Appendix A, we describe the development of a VST prototype and its integration into a commercial software from a top audio technology company.[3]

The rest of the paper is organized as follows: In Sec. 2 we review prior work on neural drum sound synthesis, paying special attention to DrumGAN (Nistal et al., 2020); in Sec. 3 we detail the proposed changes over the original model; Sec. 4 describes the experiments carried out. Results are presented in Sec.5, and we conclude in Sec. 6.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

[1]bit.ly/drumganvst-mlas

[2]https://mawf.io/

[3]to be disclosed upon acceptance of the paper

2. Previous Work

Many deep learning methods have been applied to address general audio synthesis. *Autoregressive models* are some of the most influential, achieving state-of-the-art results in many audio synthesis tasks, particularly for speech generation (van den Oord et al., 2016). However, autoregressive methods are generally slow at generation and afford little control, essential in a music creation tool. Latent variable models such as Generative Adversarial Networks (GANs) (Goodfellow, 2017), or Variational Auto-Encoders (VAEs) (Kingma & Welling, 2014), have been more widely used in this sense as they are faster and allow manipulating learned controls affecting high-level factors of variation in the generated data (Caillon & Esling, 2021; Aouameur et al., 2019; Engel et al., 2019). Specifically, GANs have shown promising results in drum sound synthesis (Nistal et al., 2020; Drysdale et al., 2021; Tomczak et al., 2020) and are generally superior to other generative methods in terms of speed and quality. Recently, Denoising Diffusion models have shown results on par with GANs (Ho et al., 2020) and were applied to drum sound synthesis obtaining unprecedented audio quality and diversity (Rouard & Hadjeres, 2021). However, diffusion models require an iterative denoising process that is often time-consuming.

Our work builds upon DrumGAN (Nistal et al., 2020) by carrying out a series of modifications aimed at building a functional drum synthesis tool complying with artists' workflows and industry standards. The following section briefly introduces the original architecture.

2.1. DrumGAN

DrumGAN (Nistal et al., 2020) is a Generative Adversarial Network (GAN) trained to generate drum sounds conditioned on high-level perceptual features (e.g., *boominess*, *hardness*, *roughness*). It operates on 16 kHz sample-rate audio in the form of a complex Short-Time Fourier Transform (STFT) spectrogram (i.e., using the *real* and *imaginary* components of the STFT as separate channels of a tensor).

The input to DrumGAN's generator G is a concatenation of $n_C = 7$ perceptual features $\boldsymbol{c}$ mentioned above, and, as it is typical in the GAN setting, a random vector $\boldsymbol{z}$ sampled from an independent Gaussian distribution $\boldsymbol{z} \sim \mathcal{N}_{n_z=128}(\mu = 0, \sigma^2 = \mathbf{I})$ with $n_z = 128$ latent dimensions. The resulting vector, with size $n_z + n_C = 135$, is fed to G to generate the output signal $G(\boldsymbol{z}, \boldsymbol{c})$. The discriminator D estimates the Wasserstein distance between the real and generated distributions (Gulrajani et al., 2017). Also, in order to encourage G to use the conditional features, D has to predict these and an auxiliary mean-squared error loss is added to the objective. The authors employ the progressive growing framework (Karras et al., 2018) where the architecture is built dynamically during training. As for the architecture, G and D follow a mirrored configuration composed of a stack of 6 convolutional blocks. Each block is followed by up/down-sampling steps (respectively for G and D) of the temporal and frequency dimension. For more architecture and training details, we refer the reader to the original paper (Nistal et al., 2020).

3. Contributions

In this section, we describe in more detail our contributions to the original DrumGAN implementation. We depart from a preliminary prototype built upon the original model (details in Appx. A). The plugin was tested by 3 artists over a month, followed-up by informal discussion sessions to drive modifications (see Sec. 3.1, 3.2, and 3.3).

3.1. Increasing the sampling rate

One main concern of artists is DrumGAN's audio quality, especially for instruments with significant energy density at the high-end of the spectrum (e.g., snares, cymbals). Since DrumGAN generates 16 kHz sample-rate audio, not only the highest represented frequency is 8 kHz (i.e., the Nyquist frequency), but also, as a result, we argue that artifacts derived from the model are more likely to appear in perceptually-relevant frequency regions. *DrumGAN VST* is trained on drum sounds with a sample rate of 44.1 kHz (see Sec. 4.1). In order to exploit the same architecture without losing representational capacity due to the increased audio resolution, we halve the duration of the generated audio to 0.5 seconds instead of 1 (see Sec. 4.1).

3.2. From perceptual features to soft class labels

Another reiterated concern from artists was the inconsistent behaviour of DrumGAN's perceptual controls, which we opt to remove altogether, as well as the impossibility of directly choosing the specific instrument class to be generated (kick, snare, or cymbal). Since DrumGAN is not conditioned on class labels, the only way to achieve this is by jointly manipulating the conditional features and the latent noise. On the other hand, artists appreciate the possibility to perform interpolations across classes (e.g., continuously transforming a snare into a kick). To allow for such type of control in *DrumGAN VST* we condition it on soft instrument labels, i.e., continuous instrument class probabilities instead of one-hot class vectors. This allows users to continuously and independently control instrument-specific features of the sound to be synthesized. Soft labels are obtained by separately training a classifier of kick, snare, and cymbal sounds using an ad-hoc implementation of the Inception network architecture (Szegedy et al., 2016) trained on 128-bin Mel spectrograms. Similar to prior work (Nistal et al., 2021), we distill knowledge from this classifier into our generative model by predicting class probabilities on the training data and using these to condition the GAN as explained in Sec. 2.1 (with $n_C = 3$ now).

3.3. Adding an encoder

As mentioned in Sec. 1, various techniques are exploited by producers to create variations of existing drum sounds (e.g., layering, ADSR manipulation). To allow creating variations in our tool without giving up high-level control, we separately train an encoder network that maps incoming sounds into the GAN's latent space, similarly to previous works (Drysdale et al., 2021) (i.e., estimates the noise vector $\boldsymbol{z}$ and the instrument class probability $\boldsymbol{c}$). This way, one can generate variations of an existing sound by simply encoding it and moving away from the initially predicted noise vector $\boldsymbol{z}$.

The encoder E is composed of a stack of 6 convolutional layers with channels in {32, 64, 128, 128, 64, 32}, kernels of size 3x3, alternating stride of 2x2 and 2x1, and padding of size 1x1. These layers are followed by 4 fully connected layers (see Appx. B for details). E outputs an estimation of the latent parameters $\boldsymbol{z}$ and $\boldsymbol{c}$. The training objective is to minimize a reconstruction loss on the generation parameters $\boldsymbol{z}$ and $\boldsymbol{c}$, as well as the spectral distance between the original and reconstructed log-magnitude spectrograms generated from the predicted parameters. The resulting loss function is

$$L = \alpha \cdot \text{MSE}((\hat{\boldsymbol{z}}, \hat{\boldsymbol{c}}), (\boldsymbol{z}, \boldsymbol{c})) + \beta \cdot \text{MSE}(G(\hat{\boldsymbol{z}}, \hat{\boldsymbol{c}}), G(\boldsymbol{z}, \boldsymbol{c})),$$

where $(\hat{\boldsymbol{z}}, \hat{\boldsymbol{c}}) = E(G(\boldsymbol{z}, \boldsymbol{c}))$, MSE denotes the Mean Squared Error, and α and β are weighting coefficients for the latent vector and magnitude reconstruction errors respectively.

Despite *DrumGAN VST*'s encouraging audio quality results (see Sec. 5), we observe some systematic bias in the form of inaudible, high-frequency artifacts. In our first attempt to train E, we notice that it takes advantage of this bias to encode samples into the latent space, over-fitting on the training data and failing to encode real drum sounds that don't exhibit these artifacts. Therefore, we threshold the spectrograms below -1.5 dB to remove silent parts, forcing the model to learn from more salient information.

4. Experiment Setup

In this Section details are given about the experimental setup, including the dataset used and the evaluation method.

4.1. Dataset

DrumGAN VST is trained on a proprietary collection comprising over 300k one-shot audio samples equally distributed across kick, snare, and cymbal classes. Sounds have a sample rate of 44.1 kHz and variable lengths. Each sample is trimmed or zero-padded to a duration of 0.55 seconds as 80% of the data is below this duration. We perform a 90% / 10% split of the data for validation purposes. As in DrumGAN (Nistal et al., 2020), the model is trained on the real and imaginary components of the Short-Time Fourier Transform (STFT). The STFT is computed using a window size of 2048 samples and 75% overlapping. The generated spectrograms are inverted to the signal domain using the inverse STFT.

The encoder E is trained on a fixed set of latent random vectors $\mathbf{z}$ and the corresponding generations $G(\mathbf{z}, \mathbf{c})$ (i.e., no real sounds are used to train E), where the soft class labels $\mathbf{c}$ are obtained by running our classifier on randomly sampled instances of the dataset described above.

4.2. Evaluation

We evaluate our tool by computing various objective metrics on the generated content. In the accompanying website[4] we show extensive examples and musical material created by music producers using the tool. As suggested by prior work (Deruty et al., 2022), we believe that having music released by artists is an indirect but critical way of validation for any creative music tool.

As for the objective evaluation, a common practice in the generative modeling literature is to measure the Inception Score (IS), Kernel Inception Distance (KID), and Fréchet Audio Distance (FAD).[5] These metrics assess, to some degree, the quality and diversity of the GAN generations. In order to evaluate E, we also compute a set of audio reconstruction metrics: the Mean-Squared Error (MSE), Log-Spectral Distance (LSD), Signal-to-Noise Ratio (SNR), Distortion Index (DI), and the Objective Difference Grade (ODG). Note that DI and ODG are a computational approximation to users' subjective evaluations when comparing two signals.

5. Results

Table 1 shows G's evaluation results for the IS, KID, and FAD. We compare results against real data and two prior works: *Style-DrumSynth* (Drysdale et al., 2021), based on StyleGAN, and CRASH (Rouard & Hadjeres, 2021), based on denoising diffusion models.[6] Overall, DrumGAN scores the best results for most metrics, closely followed by CRASH. *DrumGAN VST* and *CRASH* obtain an IS that is on a par with real data, which suggests that both models can generate diversity across the instrument classes and that the generated samples are somewhat classifiable into one of all possible classes. *Style-DrumSynth* obtains slightly worse results, although this could be due to the mismatch between

[4] bit.ly/drumganvst-mlas

[5] https://github.com/google-research/google-research/tree/master/frechet_audio_distance

[6] We employ 100 denoising steps in the denoising process

	↑ IS	↓ KID	↓ FAD
real data	1.86	0.004	0.09
DrumGAN VST	**1.83**	0.009	**1.49**
Style-DrumSynth	1.64	0.085	1.72
CRASH	1.81	**0.004**	1.91

Table 1. Results of IS, KID, and FAD (see Sec. 4.2), scored by *DrumGAN VST*. We compare against real data and two baselines: *Style-DrumSynth* (Drysdale et al., 2021) and *CRASH* (Rouard & Hadjeres, 2021).

		↑ DI	↑ ODG	↓ MSE	↓ LSD	↑SNR
DrumGAN VST	*gen*	**0.14**	**-1.73**	0.03	**2.94**	**-1.67**
	our	-0.06	-1.92	0.06	7.36	-3.17
	sds	0.04	-1.83	0.05	7.28	-2.85
	test	-0.20	-2.06	**0.01**	11.10	-2.87
Style-DrumSynth	*gen*	-1.12	-2.78	0.03	10.05	-2.88
	our	-1.76	-3.06	0.09	15.31	-4.82
	sds	-1.56	-2.97	0.08	12.15	-3.60
	test	-2.27	-3.21	0.02	26.18	-6.98

Table 2. Results of DI, ODG, MSE, LSD, and SNR (see Sec. 4.2) computed on encoded and reconstructed pairs from i) generated data (*gen*), ii) *our* training data, iii) the baselines's training data (*sds*), and iv) a *test* set including unseen sounds (e.g., toms).

Style-DrumSynth's training dataset and the one used to train the Inception model. The KID reflects whether the generated data overall follows the distribution of real data in terms of timbre features (the Inception model is trained to predict instrument classes and features from the audio-commons timbre models[7]). *CRASH* obtains results that are on a par with real data, followed closely by *DrumGAN VST*, suggesting that both models can generate sounds sharing timbral characteristics with real data. *Style-DrumSynth* obtains relatively worse KID, suggesting that the real and generated data diverge in terms of timbral features. Finally, FAD is a reference-free measure that correlates with the perceived audio quality of the individual sounds. We observe that *DrumGAN VST* obtains lower FAD than the baselines, suggesting that the generated audio contains fewer artifacts and resembles real data in terms of perceived quality.

In Table 2 we present the evaluation results for the encoder E. Again, results are compared against *Style-DrumSynth*, which also incorporates a separate encoder that maps sounds into the GAN's latent space, following a method analogous to ours. The metrics are computed on encoded/reconstructed pairs from different sets of data: *gen* refers to generated data (i.e., each model encodes/reconstructs its own generated data), *our* training data (see Sec. 4.1), *Style-DrumSynth*'s training data (*sds*), and, finally, a *test* set containing percussive sounds including examples out of the training distribution (e.g., toms, shakers). Overall, *DrumGAN VST* outperforms the baseline in most metrics. It is interesting to see that this is the case even for the baseline's training data (*sds*), never seen by *DrumGAN VST* at training. It is also surprising that *DrumGAN VST* generally obtains better MSE and SNR performance than the baseline, considering that these metrics are highly sensitive to phase information and that *DrumGAN VST*'s E only receives as input the magnitude of the STFT. Nonetheless, SNR is negative for all models, indicating that the time-domain residual signal from the difference between encoded and decoded sounds has greater power than the actual encoded sound. Therefore, despite the superiority of *DrumGAN VST* on this metric, we can argue that neither of the models does a good job in terms of phase preservation. However, in terms of magnitude spectrogram reconstruction, *DrumGAN VST* seems to obtain a much better performance than the baseline as suggested by the lower LSD (which only compares log-magnitude spectrograms). Finally, *DrumGAN VST* outperforms the baseline on the DI and ODG metrics, which are precise metrics used for the perceptual evaluation of the audio quality. ODG ranges from 0 to -4, where lower values denote greater quality degradation between signals. *DrumGAN VST* obtains ODG values between -1 and -2, indicating that there exist slight impairments between encoded and decoded sounds, although, in the case of the baseline, these impairments are generally annoying (ODG < -2). Differences between *DrumGAN VST* and the baseline are accentuated in the case of DI, which correlates to the ODG but has higher sensitivity towards poor signal qualities.

We conclude from these results that the proposed encoder E can competently encode and decode sounds into *DrumGAN VST*'s latent space, even for timbres never seen during training, and with better performance than *Style-DrumSynth*. As we show in the website accompanying this paper, this is the case even for, e.g., vocal percussion (i.e., beat-box sounds).

6. Conclusion

In this work, we presented *DrumGAN VST*, a plugin for analysis/synthesis of drum sounds employing an autoencoding Generative Adversarial Network (GAN). The model operates on 44.1 kHz sample-rate audio, and it enables continuous control over kick, snare, and cymbal classes. When compared to prior work on neural synthesis of drums (Rouard & Hadjeres, 2021; Drysdale et al., 2021), our model obtains superior results according to objective metrics assessing the quality, diversity, and reconstruction of sounds. The proposed plugin is developed in collaboration with professional music artists from whom we show released musical material on an accompanying website. The tool is integrated into commercial software from a top audio-tech company.

[7] https://github.com/AudioCommons/ac-audio-extractor

References

Aouameur, C., Esling, P., and Hadjeres, G. Neural drum machine: An interactive system for real-time synthesis of drum sounds. In *Proc. of the 10th International Conference on Computational Creativity, ICCC*, June 2019.

Caillon, A. and Esling, P. RAVE: A variational autoencoder for fast and high-quality neural audio synthesis. *CoRR*, abs/2111.05011, 2021. URL https://arxiv.org/abs/2111.05011.

Deruty, E., Grachten, M., Lattner, S., Nistal, J., and Aouameur, C. On the development and practice of ai technology for contemporary popular music production. *Transactions of the International Society for Music Information Retrieval*, 5(1), 2022.

Devis, N. and Esling, P. Neurorack: deep audio learning in hardware synthesizers. In *EPFL Workshop on Human factors in Digital Humanities*, Dec. 2021.

Dieleman, S., van den Oord, A., and Simonyan, K. The challenge of realistic music generation: modelling raw audio at scale. In *NeurIPS*, pp. 8000–8010, Montréal, Canada, Dec. 2018.

Donahue, C., McAuley, J., and Puckette, M. Adversarial audio synthesis. In *Proc. of the 7th International Conference on Learning Representations, ICLR*, May 2019.

Drysdale, J., Tomczak, M., and Hockman, J. Style-based drum synthesis with gan inversion. In *Extended Abstracts for the Late-Breaking Demo Sessions of the 22nd International Society for Music Information Retrieval (ISMIR) Conference.*, 2021.

Engel, J., Agrawal, K. K., Chen, S., Gulrajani, I., Donahue, C., and Roberts, A. Gansynth: Adversarial neural audio synthesis. In *Proc. of the 7th International Conference on Learning Representations, ICLR*, May 2019.

Goodfellow, I. J. NIPS 2016 Tutorial: Generative Adversarial Networks. *CoRR*, abs/1701.00160, 2017. URL http://arxiv.org/abs/1701.00160.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. Improved training of wasserstein gans. In *Proc. of the International Conference on Neural Information Processing Systems, NIPS*, Long Beach, CA, USA, Dec. 2017.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/4c5bcfec8584af0d967f1ab10179ca4b-Abstract.html.

Karras, T., Aila, T., Laine, S., and Lehtinen, J. Progressive growing of gans for improved quality, stability, and variation. In *International Conference on Learning Representations, ICLR*, May 2018.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *Proc. of the 2nd International Conference on Learning Representations, ICLR*, Banff, AB, Canada, Apr. 2014.

Nistal, J., Lattner, S., and Richard, G. DrumGAN: Synthesis of Drum Sounds With Timbral Feature Conditioning Using Generative Adversarial Networks. In *Proc. of the 21st International Society for Music Information Retrieval, ISMIR*, Montréal, Canada, 2020.

Nistal, J., Lattner, S., and Richard, G. Darkgan: Exploiting knowledge distillation for comprehensible audio synthesis with gans. *arXiv preprint arXiv:2108.01216*, 2021.

Rouard, S. and Hadjeres, G. CRASH: raw audio score-based generative modeling for controllable high-resolution drum sound synthesis. In Lee, J. H., Lerch, A., Duan, Z., Nam, J., Rao, P., van Kranenburg, P., and Srinivasamurthy, A. (eds.), *Proceedings of the 22nd International Society for Music Information Retrieval Conference, ISMIR 2021, Online, November 7-12, 2021*, pp. 579–585, 2021. URL https://archives.ismir.net/ismir2021/paper/000072.pdf.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the Inception Architecture for Computer Vision. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pp. 2818–2826, Las Vegas, NV, USA, June 2016. IEEE Computer Society. doi: 10.1109/CVPR.2016.308.

Tomczak, M., Goto, M., and Hockman, J. Drum synthesis and rhythmic transformation with adversarial autoencoders. In Chen, C. W., Cucchiara, R., Hua, X., Qi, G., Ricci, E., Zhang, Z., and Zimmermann, R. (eds.), *MM '20: The 28th ACM International Conference on Multimedia, Virtual Event / Seattle, WA, USA, October 12-16, 2020*, pp. 2427–2435. ACM, 2020. doi: 10.1145/3394171.3413519. URL https://doi.org/10.1145/3394171.3413519.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W., and Kavukcuoglu, K. Wavenet: A generative model for raw audio. In *Proc. of the 9th ISCA Speech Synthesis Workshop*, Sunnyvale, CA, USA, Sept. 2016.

A. User Interfaces

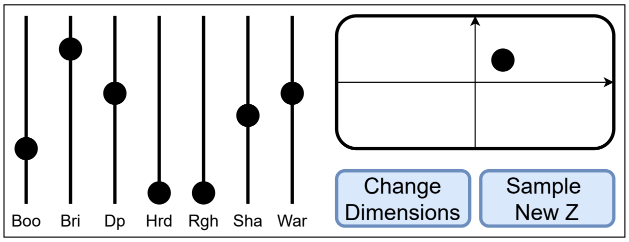

Figure 1. Schematic of the first interface developed to interact with DrumGAN. It offers sliders to control the conditional perceptual features (e.g., *boominess*, *brightness*, *depth*). A 2D plane, centered in $\boldsymbol{z}_{\text{center}}$ and directed by vectors $(\boldsymbol{e_1}, \boldsymbol{e_2})$ allows the user to explore different values for $\boldsymbol{z}$. $(\boldsymbol{e_1}, \boldsymbol{e_2})$ are orthonormal vectors sampled from a gaussian. A button (*Change Dimensions*) allows to change these vectors and another button (*Sample New Z*) allows to randomly sample a new center for the plane $\boldsymbol{z}_{\text{center}} \sim N(0, I)$. Ultimately, from the circled marker at coordinates (α, β) we decode $\boldsymbol{z} = \boldsymbol{z}_{\text{center}} + \alpha \boldsymbol{e_1} + \beta \boldsymbol{e_2}$

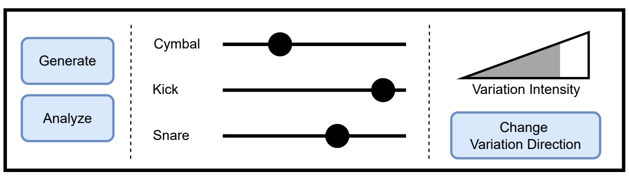

Figure 2. Schematic of the interface developed to integrate DrumGAN into a commercially available software. The interface now has sliders to control the soft-class conditional vectors $\boldsymbol{c}$. A button 'Analyze' triggers the encoding of a sample, setting the sliders to the appropriate values, as well as the internal variable $\boldsymbol{z}_{\text{center}}$. The button 'Generate' is used to sample a new $\boldsymbol{z}_{\text{center}}$. The 2D plane is replaced with a simpler 1D slider. From a variation intensity α, we decode $\boldsymbol{z} = \boldsymbol{z}_{\text{center}} + \alpha \boldsymbol{e_1}$

DrumGAN (Nistal et al., 2020) showed encouraging results for drum synthesis in terms of control, quality and diversity, yet, having it accessible through a command-line interface impedes artists from incorporating it into their music production workflow, which ultimately hinders the possibility of obtaining valuable feedback. Hence, our first focus prior to devising a more elaborate music production tool from DrumGAN is to develop a usable prototype featuring a graphical user interface that artists can test comfortably. The Virtual Studio Technology (VST) standard[8] for the integration of virtual effect processors and instruments into digital audio environments seems like the natural choice for this task. This standard is open-source, and many C++ frameworks exist, such as JUCE[9], which allows embedding deep learning models easily. In Fig 1, we show the schematic of a simple interface developed with JUCE that naively exposes DrumGAN's parameters: sliders are used for the perceptual features, and a 2D plane is used the traverse the latent space.

This prototype is assessed by three professional music producers that provide feedback and guide improvements on the interface and the model. As a result of artist feedback, and in addition to the model modifications described in this work, we also conduct some improvements over our initial interface. Artists generally find it difficult to interpret the 2D plane used to control the latent space navigation. Hence, in the last interface version, depicted in Fig. 2, we replace this plane with a single slider (*Variation Intensity*) that specifies the magnitude of the displacement from some initially sampled $\boldsymbol{z}$. The button *Change Variation Direction* sets a new random direction for the displacement. This interface is finally chosen to be implemented into commercial software.

B. Encoder Details

Layer	Type	Ch.	tensor size	ker.	str.	pad.	activation
Output	FC	-	131	-	-	-	SoftMax
Layer9	FC	-	512	-	-	-	LReLU
Layer8	FC	-	1024	-	-	-	LReLU
Layer7	FC	-	3072	-	-	-	LReLU
Layer6	CNN	32	16x6	3x3	2x1	1x1	LReLU
Layer5	CNN	64	32x6	3x3	2x2	1x1	LReLU
Layer4	CNN	128	64x12	3x3	2x1	1x1	LReLU
Layer3	CNN	128	128x12	3x3	2x2	1x1	LReLU
Layer2	CNN	64	256x24	3x3	2x1	1x1	LReLU
Layer1	CNN	32	512x24	3x3	2x2	1x1	LReLU
Input	-	1	1024x48	-	-	-	-

Table 3. Architecture details for the encoder. The tensor output by the last FC layer is split up in $\boldsymbol{z}$ and $\boldsymbol{c}$. The SoftMax is applied to $\boldsymbol{c}$ only, while $\boldsymbol{z}$ does not go through a non-linearity.

The encoder E architecture is detailed in Table 3. Batch normalization is followed after every layer and bias are removed. We employ a learning rate of 10^{-4} and a batch size of 28 training instances. We train the model using weights $\alpha = 1$, and $\beta = 3$ in loss L (see Sec. 3.3) for the latent vector and magnitude reconstruction errors respectively.

[8] https://developer.steinberg.help/display/VST/VST+Home
[9] https://juce.com/