NATIONAL RESEARCH UNIVERSITY
HIGHER SCHOOL OF ECONOMICS

Faculty of Computer Science
Bachelor's Programme "Applied Mathematics and Informatics"

Research Project Report on the Topic:

Creation of Audio Effects Using Deep Learning

(interim, the first stage)

Submitted by the Student:

group #БПМИ231, 2nd year of study — Kazakov Ivan Andreevich

Approved by the Project Supervisor:

Grinberg Petr Markovich
Visiting Teacher
Faculty of Computer Science, HSE University

Moscow 2025

Contents

Annotation

Classical approaches to creating audio effects are to use digital signal processing or analog converters. With the help of neural networks, we can expand the creative potential and achieve new sounds that are inaccessible to traditional solutions. In addition, using such algorithms, we can emulate analog sound without the need to buy equipment. In this work, we implement an existing solution based on the autoencoder architecture from scratch and propose a new Transformer-based approach built on top of that solution. Our results show a remarkable improvement for chorus and tremolo effects using a novel approach.

Аннотация

Классическими методами создания аудио эффектов являются использование цифровой обработки сигналов или аналоговых преобразователей. С помощью нейронных сетей можно расширить творческий потенциал и достичь нового звучания, недоступного традиционным решениям. К тому же, с помощью таких алгоритмов можно скопировать аналоговое звучание без необходимости в покупке техники. В результате нашей работы мы реализовали классическое решение на основе архитектуры автокодировщика с нуля и переработали более новую модель на основе архитектуры Трансформера для создания аудио эффектов. Кроме того, новый подход позволил нам улучшить качество для таких звуковых эффектов, как хорус и тремоло.

Keywords

Deep Learning, Digital Signal Processing, Audio Processing, Audio Effects, Digital Filters

1 Introduction

Audio effects can be defined as "the controlled transformation of a sound based on some control parameters" [21]. They are used to shape acoustics, tone, timbre, and other characteristics, which, in turn, significantly change the perception of transformed sound.

Audio effects are the essential part of music production and sound design. In recent years, with the growth of the music, video game, and film industries, there has been an increased interest in creating novel audio effects and more accurately recreating the sound of older analog devices.

Modern music production tools are mostly digital audio workstations (DAWs). Audio effects are either built into the workstation itself [5] or implemented via virtual studio technology (VST) plugins [12].

The creation of audio effects using deep learning is a promising direction, as many music producers want to recreate the characteristic "analog sound" that is lost due to imprecise digital modeling of analog circuits or use completely new, unexplored effects.

Modeling audio effects often requires knowledge of the underlying analog circuitry; however, detailed analysis of the target device is not always possible [1]. Instead, we can make assumptions about how the analog device works, build a model, and use deep learning techniques to tune that model. Furthermore, the deep learning approach allows us to create a model that is able to generalize from one effect to another [11]. This property allows streamlining the simulation development.

A plethora of audio effects exists. However, for the purposes of this study, the focus is on chorus, tremolo, and distortion.

In this coursework, a deep learning approach is used to emulate audio effects. Firstly, we reproduce an existing solution based on autoencoders [11]. Then we develop the ideas from the first approach and consider the Transformer-based [20] autoencoder [8], which, to the best of our knowledge, has not been used exclusively for the purpose of creating audio effects. Subsequently, a comparative study of these approaches is conducted. The results of the study show an improvement in the quality for chorus and tremolo effects in comparison to the classical approach.

The rest of the paper is organized as follows. Section 2 reviews existing solutions. In Section 3, our approaches are introduced and explained. Section 4 describes the setup of our experiments. In section 5, the results of our experiments are presented and discussed. Finally, our findings are summarized in section 6.

2 Literature review

2.1 Chorus

The purpose of chorus audio effect is to emulate multiple voices playing in unison [2]. The most common way to model chorus is with a delay unit or delay modulator [2, 21]

This effect is used extensively in many genres of music to create a distinctive melancholic sound, both on instruments and with vocals.

In order to simulate this effect, the deep learning model must be capable of modulating the delay line.

2.2 Distortion

Distortion can be defined as the controlled non-linear deformation of an audio signal, resulting in added harmonics and a compressed sound [21]. This effect initially emerged inadvertently when musicians discovered that they could amplify their guitars beyond maximum capacity to produce a characteristic heavy sound. Subsequently, the first devices were developed that employed the same approach. Distorted guitar sound has become a hallmark of rock and metal music.

Digital analogues frequently emulate this effect by imposing a degree of nonlinear clipping on the input signal [3]. The specific character of the output sound is determined by the particular nonlinearity and the selected mixing parameters.

Consequently, this effect requires the model to effectively manage various nonlinearities and short-term memory dependencies.

2.3 Tremolo

The audio tremolo effect is defined as the modulation of the amplitude of the input signal by a specific function. Typically, a low-frequency sine wave is employed as the modulator [21].

This effect will be used to explore the waveshaping capabilities and memory for long temporal dependencies of the implemented models.

2.4 CNN architecture for distortion effect

It is important that the audio effect can be applicable and audible in real time, thereby ensuring the convenience of musicians and sound designers. The authors of [1] build a model based on the WaveNet [19] architecture that reliably simulates some distortion pedals in real time.

A critical attribute of the CNN is its receptive field, which determines the number of preceding samples that impact the output sample. This parameter directly affects the quality of the model. The authors of the article investigated the impact of hyperparameters on the performance of their model. They experiment with various hyperparameters, including the dilation pattern and the number of convolutional layers, which directly influence the receptive field.

Overall, this approach develops the idea of incorporating CNN layers and various nonlinearities to model the distortion audio effect.

2.5 Discovering new audio effects

Creating novel audio effects is not an easy task, because it is necessary to maintain the line between interesting and unusual effects and absurd ones. Steinmetz and Reiss [18] proposed an approach that involves training a temporal convolutional network (TCN) [17] on a single clean input and output with an applied effect pair, additionally including variable parameters that remain constant during training. Post-training parameter adjustment produces new effects, which are similar to the one the model was trained on, but with intriguing details and inaccuracies, which opens up space for creativity. Furthermore, even an untrained neural network with a similar architecture is capable of producing a reverb-like effect [16].

Consequently, this approach investigates the limits of the applicability of the time domain in deep learning for the creation of audio effects.

2.6 Streamlining the creation of audio effects

Ramirez *et al.* [11] presented a model that is capable of emulating a broad range of time-varying audio effects. This approach has the potential to significantly increase the efficiency of the audio effects design process.

The model is composed of three distinct components: the front-end, the Bi-LSTM, and the back-end. The front-end transforms the input signal into the latent space by performing time-domain convolutions, applying nonlinearity and max pooling. The Bi-LSTM learns long temporal dependencies. The back-end synthesizes the output signal by unpooling the processed latent space and utilizing fully-connected neural network with adaptive activation functions.

The training phase has some interesting insights. The authors initially implemented unsupervised learning exclusively on the convolution layers. Subsequently, they incorporated the remaining layers and employed a supervised method to retrain the pre-trained model. This method helps to facilitate enhanced model fitting during the training process for time-varying tasks.

This approach was chosen by us for implementation because it encapsulates the ideas of previously described methods, transferring them to the autoencoder architecture and incorporating long-term memory, which allows us to model such time-varying audio effects as chorus and tremolo and nonlinear audio effects as distortion.

2.7 Audio Masked Autoencoder

Today, the Transformer architecture [20] is applied to a wide variety of deep learning tasks, including audio, and often outperforms other approaches [9]. One such model that uses the Transformer architecture for audio applications is the Audio Masked Autoencoder (Audio-MAE) [8]. Just like the previous approach, this architecture is an autoencoder, but it uses more novel techniques and therefore has great potential for creating audio effects.

The Audio-MAE essentially has two parts: Transformer encoder and Transformer decoder. The pipeline of this architecture is straightforward and consists of the following steps: the raw waveform is initially converted into a Mel-spectrogram and embedded into spectrogram patches, then the most of these embeddings are masked and only non-masked embeddings are fed into the Transformer encoder, subsequently, these patches are padded with trainable embeddings to recreate masked patches, these patches are then reordered to the correct spectrogram order, and after these steps they are passed to the Transformer decoder, which utilizes local window attention to reconstruct the predicted audio spectrogram.

3 Methodology

3.1 Lightweight approach

Our first approach to creating sound effects using deep learning is based on [11]. The model that was implemented consists of three components: the convolutional encoder, the latent space Bi-LSTM, and the decoder.

This model is designed to accept n consecutive frames of the same size of raw audio signal, where n is an odd positive integer. The output of the model is a prediction of the intermediate frame of the audio signal that has been processed with the selected effect. These frames are obtained using the framing procedure: the waveform is segmented into frames of size 4096 with a hop size of 2048. Each sequence of n successive frames is then used as input to the model. Additionally, the audio is padded with zeros so that every part is utilized.

The encoder is composed of two convolutional layers, a pooling layer, and a residual connection to the decoder. The first layer of this block convolves the input data with 32 kernels, with each kernel having a size of 63. The temporal dimension of the input is padded so that after this operation the output with the same temporal dimension is obtained. The purpose of this layer is to divide the input signal into 32 channels, each of which will contain its own unique set of features. The residual connection to the decoder is the result of the convolution operation for the intermediate frame. The absolute value function is used as a nonlinearity for this layer. The second layer has 32 kernels of size 127, and computes the convolution of each kernel with the corresponding channel of the previous layer. Softplus [4] is then used as an activation function. Afterwards, the max-pooling operation with kernel size of 64 and ceiling mode is applied to the temporal dimension of all channels of the previous layer. Each of the operations in this block is applied to all n input frames. That is, if an input of shape $batch\ size \times n \times input\ channels \times frame\ size$ is fed into the model, it is reshaped to have size $(batch\ size \cdot n) \times input\ channels \times frame\ size$, all operations are applied, and then the result is reshaped back. In essence, the purpose of this part is to learn the representation of the input signal in a compressed latent space.

The Bi-LSTM component is made up of three Bi-LSTM layers with hidden units of sizes 64, 32, and 16, respectively, a fully connected layer, and a smooth adaptive activation function (SAAF) [6]. The result of the preceding encoder is reshaped to have $32n$ channels and 4096 temporal size. Accordingly, 4096 vectors of size $32n$ are obtained, which are then fed into Bi-LSTM. After the first two Bi-LSTM layers, the dropout is applied to the results with a probability equal to 0.1. Additionally, a fully-connected layer with 32 hidden units is incorporated following [13]. This layer is supposed to enhance channel interdependencies and learn a new latent representation. Subsequently, a trainable activation function SAAF is employed. This function is characterized by 26 breakpoints, which are uniformly distributed between -1 and 1. Overall, this block must further reduce the dimension and learn long temporal dependencies of an audio signal.

The decoder has three parts: an upsampling part, a deep neural network (DNN) block, and a deconvolution layer. The first part involves upsampling the resultant output from the Bi-LSTM block and multiplying it with the residual connection from the encoder block. The element-wise multiplication with the residual connection should facilitate the synthesis of the output signal. The result of this multiplication is another residual connection from this part to the deconvolution part. The DNN block consists of 4 fully-connected layers with hidden units of sizes 32, 16, 16, 32, respectively. For the first three of these layers rectified linear unit (ReLU) is used as an activation function. For the last layer the trainable SAAF nonlinearity is employed. Then the squeeze-and-excitation (SE) [7] block is incorporated. SE block applies absolute value function to the result of

the previous part, performs global average pooling for temporal dimension, applies a linear layer of size 512 with the ReLU activation function, and, subsequently, another fully-connected layer with 32 hidden units is employed with the sigmoid activation function, and the result is multiplied by the initial input of this block. Afterwards, the addition of the result of the SE block is performed with the second residual connection. This addition is supposed to model the delay line. The last deconvolution part is the transposed one-dimensional convolution of the result of the last addition with the weights of the first convolution in the encoder block. In summary, this block is intended to synthesize the output signal with an applied audio effect.

To construct the audio waveform with the frames obtained from the model prediction, the overlap-add method is used: each resulting frame is multiplied by a Hann window function and summed over the output array with a hop size of 2048.

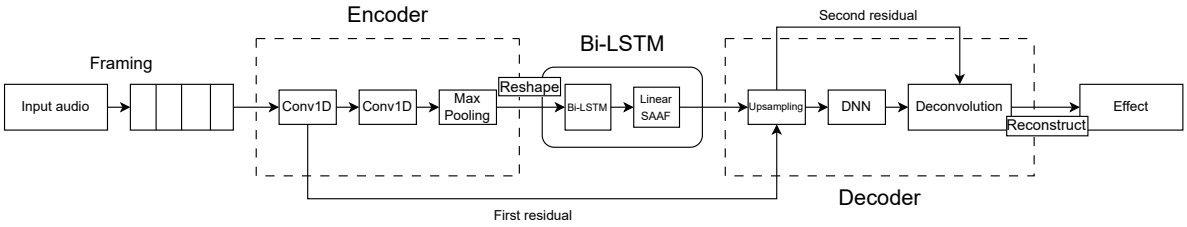

Figure 3.1: Diagram for the pipeline of the lightweight model. Inspired by [11].

Figure 3.1 displays the pipeline of the implemented model, but with less detail.

3.2 Audio-MAE

The creation of sound effects using deep learning is further investigated through the application of the Audio-MAE architecture. The model was taken from the original repository[1], and its pipeline was redesigned for the purpose of modeling audio effects.

Firstly, raw input audio is converted into a spectrogram by performing a short-time Fourier transform (STFT) with a frame length of 254, a hop length of 127, and a Hann windowing function. The squared magnitude of this transform is then taken as a spectrogram image, and the logarithm function is applied. The phase of the resulting conversion is also preserved to synthesize the audio from the spectrogram obtained from the model. Furthermore, the temporal dimension of the spectrogram is padded or cropped to match the 1024×128 shape, as this model is designed to accept images of this specific dimension.

The resultant spectrogram is then patchified with a zero mask ratio, because we want to preserve as much information about the signal as possible, and passed to the encoder. Then following the original pipeline, the result is passed to the decoder and unpatchified. The resultant image is a predicted spectrogram.

The synthesis of audio from a predicted spectrogram is achieved through the inverse short-time Fourier transform (ISTFT). The phase of the signal that is passed to the ISTFT is the phase of the input audio, and the magnitude is the predicted spectrogram with sequentially applied exponential and square root functions.

Self-supervised pretrained weights provided by the authors of the original papers are not utilized by us for two reasons: they employ a more compressed Mel-spectrogram representation, which is more difficult to convert back to audio, and the pretrained weights turned out to be suboptimal for zero mask ratio.

4 Experimental setup

IDMT-SMT-Audio-Effects dataset [14] was used to train both models. This dataset is the same as used in [11]. It was chosen to compare our implementation with the paper's results.

For each effect, 624 pairs of two second monophonic bass guitar recordings were selected, including both clean recordings and recordings with an applied corresponding effect. The effect setting for these recordings was set to 2. Additionally, all recordings were resampled to a sampling rate of 16000 and a normalization was applied. The size of the validation subset and the training

[1] https://github.com/facebookresearch/AudioMAE

subset are approximately equivalent to 5 percent of the entire dataset, respectively[2]. Following the original paper, no audio augmentations were used.

For the first model for all experiments the n value was set to 9, as it was set in [11]. This number of frames is supposed to capture all the context needed to learn long temporal dependencies. A framing procedure was employed to obtain a single batch of size 16 from each audio recording. The training of the model was divided into two stages: unsupervised pre-training and supervised training. In the pre-training stage, everything except the convolutional encoder and the last deconvolution layer is removed from the model. In addition, upsampling operation is substituted by an unpooling operation, i.e. after this operation all positions that did not have maximal value at max pooling have zeros. The concatenation of clean and processed recordings was used to train a model to recover input signal, that is, both input and target have the same waveform. The learning rate for this stage was 10^{-4}, and number of epochs was 30, as it was determined that the loss remained stable after a specified number of epochs. After this stage, the remaining layers were re-inserted into the model, the weights of the pre-trained layers were loaded, and the model was trained to predict an audio with an applied effect from clean input for 150 epochs at a constant learning rate of 10^{-4}. Multi-resolution STFT loss [22] between the reconstructed output and target waveforms was used in both training steps.

The training of the Audio-MAE model proceeded in a manner similar to the training of the first model, except that no layers were removed in the pre-training stage. For the pre-training the batch size was 16, the learning rate was $2 \cdot 10^{-4}$, and the number of epochs was 150. For the training stage (i. e. the fine-tuning of an autoencoder for the specific task) the batch size was 8, learning rate was $5 \cdot 10^{-5}$, and the number of epochs was 100. The mean squared error (MSE) between the target and predicted spectrograms was used as loss function in both training steps.

In both models Adam [10] was used as an optimizer.

The implementation of these models and experiments was written in the Python programming language using the PyTorch framework. The code is available in our Github repository[3].

5 Results and discussion

The experimental results were assessed using both quantitative and qualitative metrics.

Demo samples for each effect were prepared for the qualitative analysis of the implemented

[2]Since the authors of the original paper did not specify the sample partition index, there may be a slight mismatch between our data and those in the paper.

[3]https://github.com/ylxsbn/creation_of_audio_effects

models. These samples are available on our website[4]. Subjectively, the chorus effect appears to behave roughly the same in both the lightweight model and the Audio-MAE, but the characteristic melancholic sound is more audible in the Audio-MAE. The tremolo effect is significantly more pronounced in the Audio-MAE model. The distortion effect in the Audio-MAE has many artifacts; however, the lightweight model was able to capture the characteristic heavy sound. As demonstrated in Figure 5.1, both models show a certain degree of error in approximating the target waveforms. In the case of the lightweight model, inaccuracies in the reproduction of the signal amplitude are particularly noticeable for the tremolo effect. For Audio MAE, the presence of artifacts, which appear as spikes, is visible for each effect.

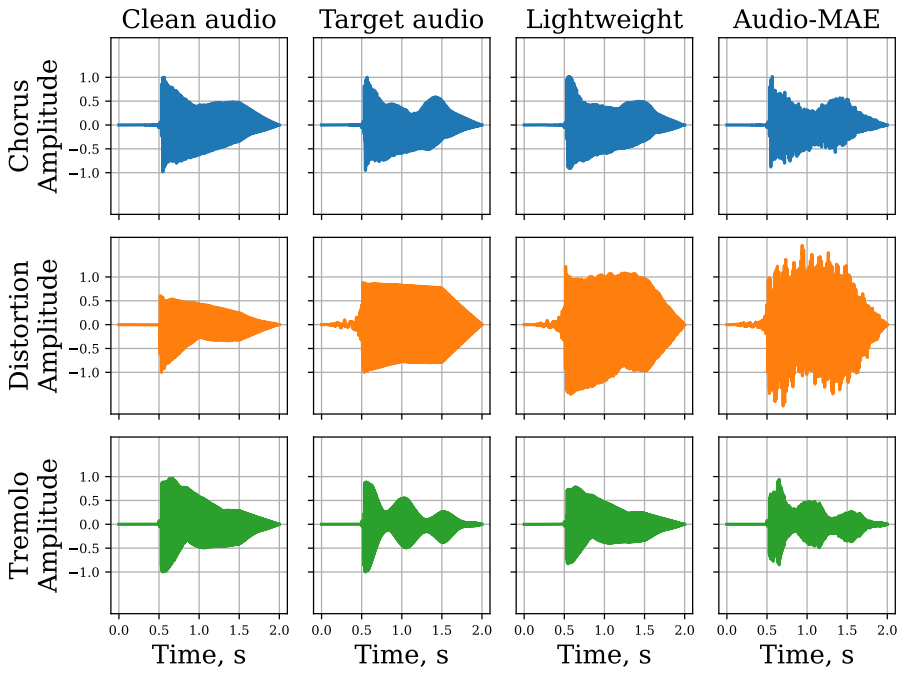

Figure 5.1: Comparison of waveforms for chorus, distortion, and tremolo for the lightweight model and the Audio-MAE.

For the comparison of our approaches with the reference, energy normalized mean absolute error (MAE) that was proposed in [11] is considered.

Table 5.1: Comparison of MAE metric on test subset for reference, the lightweight model and Audio-MAE. Lower values indicate better performance.

Audio Effect	Reference	Lightweight	Audio-MAE
	MAE	MAE	MAE
Chorus	0.0190	0.0615	0.0564
Distortion	Not provided	0.0906	0.2101
Tremolo	0.0100	0.0341	0.0139

[4] https://ylxsbn.github.io/demos.html

Table 5.1 shows the results of the comparison of the implemented model with reference values. Reference values are taken from the bar graph and are approximate. The performance of the implemented model is found to be comparable to that of the reference. In addition, the advantage of Audio-MAE over the first architecture was seen for the chorus and tremolo effects, and the superiority of the lightweight model over Audio-MAE was observed for the distortion effect according to the MAE metric.

Additional metrics are then used to further explore the quality of the implemented models. To estimate the system dynamics and dynamics range, the root mean square energy error (RMS) and crest factor (CF) were used [15, 23]. Additionally, spectral centroid error (SCE) was employed to examine how similar two audio signals are in terms of timbre and equalization properties [15].

Table 5.2: Comparison of CF, Multi-resolution STFT loss, SCE, RMS and spectrogram MSE on the test subset for the lightweight model and the Audio-MAE. Lower values are associated with higher quality for these metrics. Bold numbers indicate the lowest error for the corresponding effect.

Audio Effect	Lightweight					Audio-MAE				
	CF	STFT	SCE	RMS	MSE	CF	STFT	SCE	RMS	MSE
Chorus	**0.9158**	**0.0638**	**0.0007**	**0.8503**	0.2709	2.0421	0.0761	0.0027	1.6041	**0.1536**
Distortion	**6.8552**	**0.0623**	**0.0068**	**1.5204**	0.5557	16.0442	0.1232	0.0145	2.7963	**0.3778**
Tremolo	3.6102	0.0939	0.0024	2.3386	0.4134	**1.4498**	**0.0548**	**0.0009**	**1.1328**	**0.0998**

Table 5.2 displays the results of the comparison of objective metrics for the lightweight approach and the Audio-MAE model. For the chorus effect on the metrics CF, STFT, SCE, and RMS, the lightweight model is found to perform better than the Audio-MAE. However, the difference is not too large, and the chorus effect has a complicated temporal character, so it is difficult to assess the performance of the algorithm by these metrics alone. In our opinion, it is therefore better to consider a qualitative metric for evaluating this effect. According to the RMS and CF, which indicate dynamic range and system dynamics, and MSE, STFT and SCE, which indicate the general similarity of the sound and timbre, the lightweight model is found to perform significantly better for the distortion effect. This can be explained by the fact that the distortion effect has a complex fuzzy spectrogram, as can be seen in Figure 5.2. The Audio-MAE model struggles to extract features from such a spectrogram, so the time domain approach is advantageous in this case. For the tremolo effect, the Audio-MAE is seen to achieve better results according to all metrics, demonstrating a remarkable improvement.

Additionally, the lightweight model has only 277 thousand parameters compared to Audio-MAE's 111 million. It can be beneficial in case of limited computing power or training time.

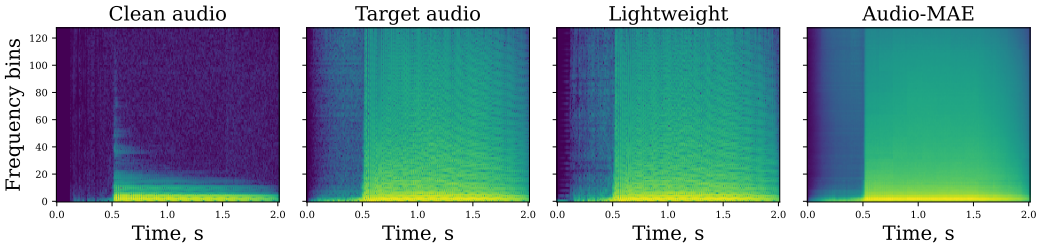

Figure 5.2: Comparison of spectrograms for the distortion effect.

6 Conclusion

In this coursework, the lightweight model from the paper was implemented and the Audio-MAE architecture was reworked to create audio effects following the ideas from the first approach. Chorus, tremolo, and distortion audio effects were emulated using both models. Our experiments showed that the proposed novel architecture outperforms an existing solution for chorus and tremolo effects: the chorus audio effect was seen to be superior using qualitative analysis, and the tremolo was found to be significantly better for both qualitative and quantitative metrics. However, the distortion audio effect was observed to perform better for the lightweight model. This could be explained by the complex nature of the spectrogram for this effect. This suggests that the time domain may be superior to the frequency domain for some tasks, but further research is needed.

Future work may include training the models on larger and more diverse datasets using different loss functions. In addition, other architectures, such as diffusion models, could be explored for the purpose of creating audio effects.

References

[1] Eero-Pekka Damskägg, Lauri Juvela, and Vesa Välimäki. "Real-time modeling of audio distortion circuits with deep learning". In: *Sound and music computing conference*. Sound and Music Computing Association. 2019, pp. 332–339.

[2] Jon Dattorro. "Effect design, part 2: Delay line modulation and chorus". In: *Journal of the Audio engineering Society* 45.10 (1997), pp. 764–788.

[3] P. Dutilleux, K. Dempwolf, M. Holters, and U. Zölzer. "Nonlinear Processing". In: *DAFX: Digital Audio Effects*. John Wiley Sons, Ltd, 2011. Chap. 4, pp. 101–138. ISBN: 9781119991298. DOI: https://doi.org/10.1002/9781119991298.ch4. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119991298.ch4. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119991298.ch4.

[4] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 315–323.

[5] Ethan Hein. "Ableton live 11". In: *Journal of the American Musicological Society* 74.1 (2021), pp. 214–225.

[6] Le Hou, Dimitris Samaras, Tahsin Kurc, Yi Gao, and Joel Saltz. "Convnets with smooth adaptive activation functions for regression". In: *Artificial Intelligence and Statistics*. PMLR. 2017, pp. 430–439.

[7] Jie Hu, Li Shen, and Gang Sun. "Squeeze-and-excitation networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7132–7141.

[8] Po-Yao Huang, Hu Xu, Juncheng Li, Alexei Baevski, Michael Auli, Wojciech Galuba, Florian Metze, and Christoph Feichtenhofer. "Masked autoencoders that listen". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 28708–28720.

[9] Saidul Islam, Hanae Elmekki, Ahmed Elsebai, Jamal Bentahar, Nagat Drawel, Gaith Rjoub, and Witold Pedrycz. "A comprehensive survey on applications of transformers for deep learning tasks". In: *Expert Systems with Applications* 241 (2024), p. 122666.

[10] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[11] Marco A Mart'inez Ram'irez, Emmanouil Benetos, and Joshua D Reiss. "A general-purpose deep learning approach to model time-varying audio effects". In: *22nd International Conference on Digital Audio Effects (DAFx-19)*. Birmingham, UK, Sept. 2019.

[12] Will Pirkle. *Designing audio effect plugins in C++: for AAX, AU, and VST3 with DSP theory*. Routledge, 2019.

[13] Marco A Martínez Ramírez and Joshua D Reiss. "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)*. IEEE. 2019, pp. 171–175.

[14] Michael Stein, Jakob Abeßer, Christian Dittmar, and Gerald Schuller. "Automatic detection of audio effects in guitar and bass recordings". In: *Audio Engineering Society Convention 128*. Audio Engineering Society. 2010.

[15] Christian J Steinmetz, Nicholas J Bryan, and Joshua D Reiss. "Style transfer of audio effects with differentiable signal processing". In: *arXiv preprint arXiv:2207.08759* (2022).

[16] Christian J Steinmetz and Joshua D Reiss. "Randomized overdrive neural networks". In: *arXiv preprint arXiv:2010.04237* (2020).

[17] Christian J. Steinmetz and Joshua D. Reiss. "Efficient neural networks for real-time analog audio effect modeling". In: *152nd Audio Engineering Society Convention*. 2022.

[18] Christian J. Steinmetz and Joshua D. Reiss. "Steerable discovery of neural audio effects". In: *arXiv preprint, arXiv:1810.10927, version 2* (2021).

[19] Aaron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, Koray Kavukcuoglu, et al. "Wavenet: A generative model for raw audio". In: *arXiv preprint arXiv:1609.03499* 12 (2016).

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[21] Thomas Wilmering, David Moffat, Alessia Milo, and Mark B. Sandler. "A History of Audio Effects". In: *Applied Sciences* 10.3 (2020). ISSN: 2076-3417. DOI: 10.3390/app10030791. URL: https://www.mdpi.com/2076-3417/10/3/791.

[22] Ryuichi Yamamoto, Eunwoo Song, and Jae-Min Kim. "Parallel WaveGAN: A fast waveform generation model based on generative adversarial networks with multi-resolution spectrogram". In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 6199–6203.

[23] Yen-Tung Yeh, Wen-Yi Hsiao, and Yi-Hsuan Yang. “Hyper recurrent neural network: Condition mechanisms for black-box audio effect modeling”. In: *arXiv preprint arXiv:2408.04829* (2024).