Parameter Estimation of Digital Audio Effects from Spectrograms

Richard Lee
Stanford University
CCRMA
rilee@stanford.edu

Ethan Buck
Stanford University
Computer Science
eljbuck@stanford.edu

Wesley Larlarb
Stanford University
Computer Science
wlarlarb@stanford.edu

Abstract

In this report, we explore the use of vision deep neural networks for parameter estimation of three common audio effects: limiter, bitcrusher, and delay via pairwise comparisons of their spectrograms. To do so, we first compiled a novel dataset of Mel spectrograms of guitars with varying parameters set for unique, common audio effects. From here, we then trained three vision models (vanilla 2DCNN, Encoder-Decoder transfer learning, UNet transfer learning) and evaluated their efficacy.

1. Introduction

Across media landscapes, audio effects - such as reverb, distortion, and dynamic range compression, to name a few - are essential components of sounds that shape our perception. Originating in the production process, these effects provide means of manipulating spatial, tonal, and temporal characteristics of audio signals, promoting uniqueness of acoustic scenes distinct from their samples, instruments, or original audio scenes. The same can be said for voice: digital audio effects govern the space of possible sounds that can be sculpted from original inputs. While many of these effects have originated from traditional digital signal processing techniques, modern deep learning has increasingly democratized and permitted understanding of audio, often through the lens of rapidly evolving models in computer vision. Through this, the ability to automatically detect and classify audio effects has become a key component of recommendation systems, music information retrieval, and machine listening.

Previous work in the field of digital audio effects modeling has focused on feature extraction from raw audio or spectrogram, with less emphasis given to the deep learning contexts of learning the transformative signatures of the audio effects applied. When translating to the frequency domain via Fourier transforms, spectrograms have provided a strong entry way into learnable patterns, such as transients, saturation, and envelope shifts induced by these audio effects. Spectrograms, in creating compact, 2D representations of signal energy across time and frequency, offer a potent input modality for deep learning to translate machine listening contexts to state-of-the-art computer vision techniques. Additionally, while raw audio is represented as an amplitude over time, many important audio features such as pitch and timbre are most easily detected in the frequency domain. While sampling-level modeling is more approachable with 1D raw audio, in most use cases, signatures of audio signals are most aligned with human perception in frequency analysis via log-magnitude or Mel spectrograms, which are coherent inputs to 2D CNNs. This leveraging of deep learning for computer vision provides entry to a range of applications, including reverse engineering of production workflows, restoration of degraded media, and style transfer across learned parameters in AI-music generation.

1.1. Problem Statement

Given the importance of digital audio effects for music production and the aforementioned promises of CV-inspired, deep learning approaches to audio processing, we have our sights on the following problem. Given a ”wet” audio signal, or in other words, an audio signal that has been processed with some digital effect such as compression, reverb, delay, etc., our model should be able to estimate the parameters of the effect which produced that wet signal. As music producers, the authors of this paper believe this task to have wide creative applications. In particular, we are aware that music professionals commonly produce, mix and master songs using a ”reference track,” i.e. an example recording with the desired sonic properties, in order to try to replicate the effects chain present on that track. If our model could successfully automate the process of parameter-matching an effect to achieve the same result as a reference track, this could save countless hours for those working in the studio.

For this study, we aimed to reverse engineer parameters of both linear and non-linear transforms based on audio effects plugins. Using wet and dry spectrograms output from sampled audio, we focused our study on the following ef-

fects with respective rationale for their selection:

1. **Limiter**: a gain controlling module, limiters provide a shift of amplitude or overall intensity of the audio, particularly when a certain threshold is hit on the decibel scale. This was selected for it's linear scaling on amplitude despite it's non-linear effect on envelope over time.

2. **Bitcrush**: Bitcrusher is an effect which resamples audio at a lower bit rate. We sought to classify exactly what bit depths were resampled as a way of being able to (1) classify, perhaps, which console game audio was originally recorded for or (2) permit reconstruction of high quality audio from recordings designed for lower-bit resolution devices. In bitcrushing, audio is quantized to values of $2^{\text{bit\_depth}}$, which is akin to the effect of audio codecs, and an important consideration for audio machine learning on embedded systems at large.

Supplement: *Sample Rate*: in supplementation to the bit depth effect, sample rate considerations were experimented with to see if reconstruction could be done with the same model architecture. This, placed in context with bit depth, indicates if our model is scalable to classification of audio codecs at large or of analysis of how we may translate audio samples across devices and sampling rates.

3. **Delay**: Delay replays a signal after a set amount of time, repeatedly doing this at an exponentially decaying level. In creative applications, this can be used for dramatic effect or to "sweeten up" a vocal or instrumental track. Delay is typically modeled via digital signal processing of finite impulse responses (FIRs), delay indicates the replay of audio after a set amount of time.

1.2. Related Works

In the context of audio effect classification and extraction, Rice et al. (2023) denotes and explores general purpose solutions to the inverse problem of removing effects from sampled audio [3]. Designing RemFX, it was proposed that a monolithic network, or a singular neural network model, would be insufficient for adequate extraction of various different effects. Thus, a compositional audio effect removal architecture was proposed, FXAug, which utilized the RemFX paradigm to add distractor effects to fit randomly sampled parameters before removing the classified effects from a newly formed intermediate signal. In terms of experimental setup, VocalSet was used for singing voice audio, GuitarSet provided acoustic guitar, DSD100 bass guitar, and IDMT-SMT-Drums for drumkit samples. From the data, removal models included Hybrid Demucs, DCUNet, DPTNet, TCN, and UMX, while detection models utilized convolutional architectures trained on Mel spectrograms fed into linear layers on top of audio representative networks such as PANNs, wav2vec2.0, and WAV2Clip. [3]

Applied to instrument specific conditions, Jürgens et al. (2020) fits the audio effects-parameter extraction problem distinctly to guitars [1]. The authors noted that their ten studied guitar effects could be split into three categories, Nonlinear, Ambience, and Modulation contexts. Using both monophonic and polyphonic samples, Hann windows were applied to audio samples fed thorugh a FFT (Fast Fourier Transform) to produce 649-dimensional input vectors for an SVM classifier. With a delta of 1 except for MFCC (Mel Frequency Cepstral Coefficients), where $\Delta = 20$, estimation effects were classified in those three denoted settings at 98% accuracy, particularly effective when using a 32 neuron hidden layer. As the average errors for the parameters of distortion, delay, and tremolo effect were between $\pm 5\%$ and $\pm 16\%$, the study noted that estimation was most accurate for parameters of highest impact on the sound and when commonly used, providing a margin potentially close to estimation by human experts. [1]

Interpreting a specific, singular plugin class, Steinmetz and Reiss (2022) evaluated real-time modeling of analog dynamic range compression via very sparse convolutional kernel architectures for efficient neural networks [6]. As such, temporal convolutional networks (TCNs) were used in this study to facilitate real-time operations on CPU, providing a shallower network to model analog mechanisms of a Universal Audio LA-2A compressor. Similar to WaveNet, the researchers introduced a modified TCN with a series of convolutional blocks along with MLP including batch normalization and feature-wise linear modulation (FiLM), alongside a PReLU nonlinearity. Using the SignalTrain dataset, 20 hours of LA-2A input-output recordings ($f_s = 44.1kHz$) were processed through 10 layers of the TCN with a dilation pattern of $d_n = 2^n$ with 32 channels at each layer. An additional LSTM architecture was tested with a recurrent layer of 32 hidden units; optimization via Adam ($lr = 3 * 10^{-4}$ and 60 epochs with batch size 32 were tested on 65536 samples. Via analysis of mean absolute error in the time domain (L_{time}) and multi-resolution short-time Fourier Transform error for the frequency domain (L_{freq}), metrics indicated high performance and efficiency of the TCN model, which had a MAE of 1.38e-2 and STFT of 0.587, indicating similar performance to the LSTM while speeding up the training by a factor of 8. A listening study was additionally conducted to evaluate model performance, where a MUSHRA found that 19 audio engineering-trained participants agreed that the TCN model output perceptual similarity to the original LA-2A, while only using 1% of the full training dataset inthe process. [6]

In a recent paper by Steinmetz et al. (2024), ST-ITO

(Style Transfer with Inference-Time Optimization) was introduced as a novel method of audio production style transfer, capable of searching the parameter space of an audio effect chain at inference [7]. This meant that across several effects chained together, regardless of arbitrary parameters affecting the input audio signal, the paper presented contributions of (1) scalable pretraining for audio production style similarity metrics (AFx-Rep), (2) ST-ITO, a style transfer architecture that optimizes the control parameters per similarity, (3) a multi-task benchmark for valuation of these systems. Using 60 hours of content generated via Pedalboard API with gain adjustments [-32dB, 0 dB], 1000 parameter configurations were randomly sampled, wherein MFCCs and K-means clustering were conducted to create a parameter space. Then, zero-shot style classification was conducted revealing an average of 0.86 accuracy in classification of the randomly chained paramaters. [7]

In this study, we built on the foundation of previous research with an extended premise of testing specific plugins at set values to reduce the arbitrary selection and pretraining conditions. Instead, we hope that by constraining our problem to focus on sounds within a particular musical context, our model will implicitly learn about the likely prior distribution of audio signals, and thus will be capable of doing some meaningful estimation of effects parameters given the processed audio signals. We sought to construct a flexible, deep architecture that could sufficiently serve as a classifier or transfer model such that specific parameters could be extracted using computer vision-oriented neural networks via interpretation of spectrograms. Further, given a novel architecture of a detachable decoder, our model is able to use the features extracted in contexts specific to the plugins they were trained on. As a form of data ablation, we provide several model architectures using a baseline convultional network, and pretraining of an encoder-decoder model and Unet. Furthermore, our use of informed, scalable parameter values in data augmentation allows appropriate emulation of common plugin effects without restriction to a specific model of plugin device (such as the LA-2A used in [6]), also reducing the introduction of arbitrary values that are either redundant or unrealistic to be used in typical audio production.

2. Data Compilation

2.1. Collection

While there are several existing datasets for the invese formulation of our problem (i.e. given dry signal and parameter values, output wet signal), there are, to our knowledge, no datasets that map the wet signals to their associated parameter values. As such, we decided to create our own wet signals from an existing dataset of dry sound files. We downloaded GuitarSet, a collection of 360 recordings

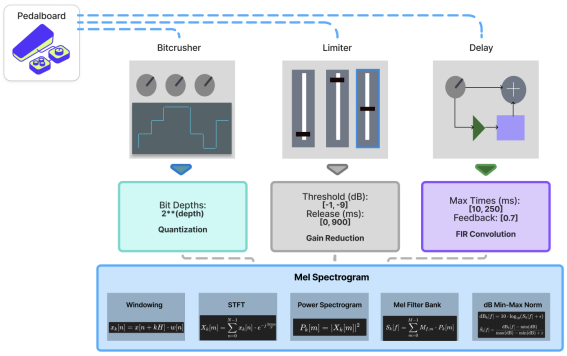

Figure 1. Digital Audio Effects Generation pipeline

of guitar, from 6 players, 5 styles, 3 chord progressions, and with slow versions, fast versions, comping and soloing [2]. For the proof of concept, we selected ten 2-second chunks of guitar sound, and then processed them using a given plugin. We run the dry audio samples through a Python script that applies an audio effect using Spotify's open-source Pedalboard API [5]. As detailed in the following sections, the plugin effects were chosen to apply certain parameters. Furthermore, for each GuitarSet audio sample, the plugins were then applied to generate pairwise data (dry and wet spectograms) that was stored on Google Cloud for retrieval and training on Colab CPU and/or T4 GPU.

2.2. Augmentation (Plugin Effects)

To enhance the variety and generalizability of training data, we applied a suite of audio effects using plugin-style augmentations. Each plugin is parameter-swept to generate distinct wet versions of the original dry signal. All wet/dry pairs are aligned in duration (2 seconds) and resampled to a consistent bit depth of 16 for comparison and supervised learning.

2.2.1 Bitcrusher

We generated 16 wet samples with different bit depth values ranging from $[0, 1, \ldots, 14, 15]$, as well as the ground truth dry sample with a bit depth of 16. Additional sample rate downsampling was conducted using TorchAudio to values of [8, 11.025, 12, 16, 22.05, 24, 32, 44.1] kHz, which correspond to telephony, PCM, VoIP, Zoom, half-CD, MPEG, broadcasting, and MP3 standards, respectively.

2.2.2 Delay

For each recording, we created 8 training examples ranging from 0 second delay to 0.25 second delay, with a constant feedback parameter of 0.7.

2.2.3 Limiter

We generated 9 wet samples using all combinations of limiter threshold values $[0.0, -5.0, -9.0]$ dB and release times $[0.0, 500.0, 900.0]$ ms, as well as the ground truth dry sample with a bit depth of 16.

2.3. Representation

Following many previous approaches, we will use spectrogram representations of the audio[1] [3] [6]. With time-frequency representations, we can leverage image-based architectures, like CNNs, that have historically outperformed a purely time-domain input. There have been recent improvements in end-to-end models for audio, but these models typically require incredibly large datasets and compute [4], neither of which are readily available for this problem. Within the possible time-frequency representations, the spectrogram and the Mel-spectrogram are popular approaches. We will proceed first with the Mel-spectrogram as it is a bit more popular for audio tasks, but we will try both representations and see how they differ.

2.4. Augmentation

In order to facilitate faster training, we segmented each audio file into 2-second chunks, as done in [1]. From here, the only further augmentation is normalizing the Mel-spectrograms. Mel spectrograms provide an effective means of maintaining perceptual scale, dimensionality reduction, and compatibility with CNN architectures. [6].

2.5. Filtering

Mel spectrograms were constructed from the initial audio dataset, with parameters of sample rate $f_s = 44.1kHz$, $FFT_n = 1024$, hop length of 512, and 128 Mel filter banks. The full Mel spectrogram formulation is detailed in the appendix.

3. Methodology

3.1. Model Architecture

Our model aims to drastically reduce the dimensionality of audio input signals to a feature vector with class scores of N many scalable parameters. Furthermore, it seeks to reconstruct a "dry" or pre-plugin sampled audio, which must have flexibility depending on the plugin parameters used to generate the "wet" signal. As such, our model architectures were the following:

(1) 2DConv downsampling layers feeding to a fully connected network for classification / extraction of plugin parameter feature vector

(2) U-Net down-to-upsampling for predicting dry signal from wet, then feature extraction of parameters from low-dimensional internal representations.

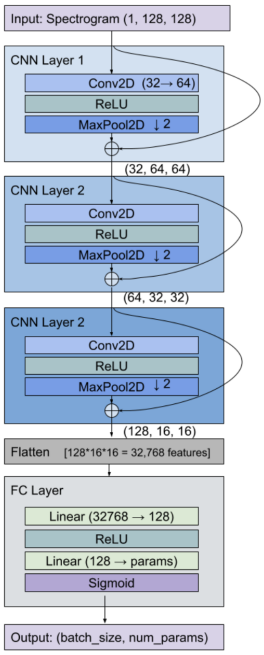

Figure 2. 2DConv downsampling feature extractor architecture

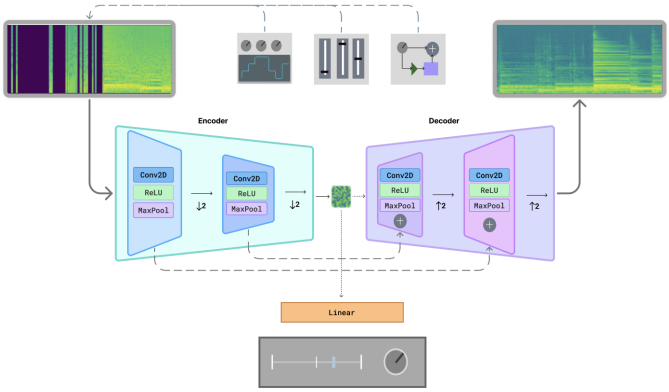

Figure 3. Full U-Net model architecture (path from feature vector (middle) to either linear classification (bottom) or reconstruction decoder (right))

Ultimately, our system will feature a Encoder-Decoder premise to (2), where-in we will train the full image transfer network, then freeze the weights of the encoder and add a finally linear layer to aid parameter estimation.

3.1.1 Hyperparameters

The following hyperparameters were chosen off of initial training and remained the same throughout testing of each plugin across all epochs for consistency of training and evaluation: BatchSize : 16, LR : $1e-4$, Epochs : 30. For all regression tests, an L1 Loss was calculated using an Adam optimizer. For image transfer an L2 loss was employed with

Adam as well. All cases were trained with a schedule with $\gamma = 0.5$ and StepSize $= 5$.

4. Experiments & Results

4.1. Limiter Testing with Conv2D Downsampler

To verify the behavior of the limiter, we compared the sample-wise differences between dry and wet signals under different threshold values. As expected, more extreme thresholds led to greater deviations from the original signal:

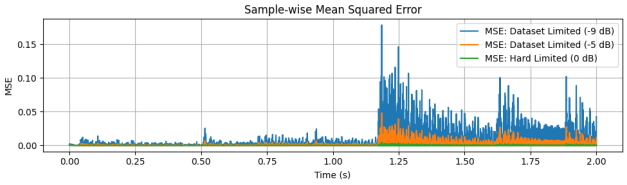

Figure 4. Sample-wise difference between dry signal and limiter-processed signals at different thresholds.

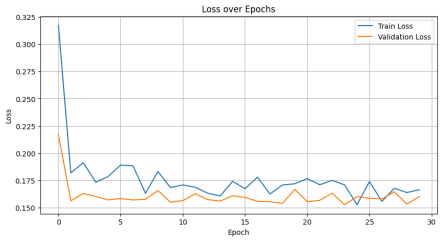

Figure 5. Training and validation losses across 30 epochs

Figure 5 shows our training and validation losses while training on a small subset of our data ($N = 72$). This indicates that our downsampling encoder model architecture is indeed able to learn on our data and will be a starting point for training larger amounts of data.

Figure 6 indicates a series of spectrograms that describe the feature maps output from each convolutional layer. These images exhibit intensity ratings within similar frequency bands, indicating that the model is interpolating on similar frequencies where the guitar is playing, as well as where transients lie.

As a preliminary benchmark, we compare our model to random chance. Let $x_1, x_2 \in [0, 1]$ be the ground truth plugin values and $\hat{x}_1, \hat{x}_2 \in [0, 1]$ be our model's estimated parameters. Assuming random chance, we have $\hat{x} \sim \text{Uniform}(0, 1), x \sim \text{Uniform}(0, 1)$. We can calculate

$$\mathbb{E}(\hat{x} - x) = \int_0^1 \int_0^1 |x - y| dx dy = \frac{1}{3} \approx 0.333$$

We can see from our MAE values, our initial Conv2D model does not performing any better than random chance. However, we proceed in the following sections on training

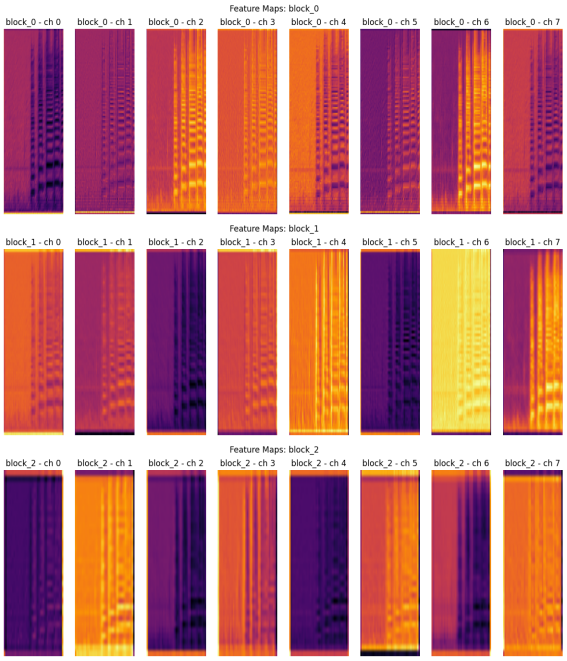

Figure 6. Feature maps of first 8 output channels of three convolution layers

Dataset	MSE	MAE
Train ($N = 72$)	0.1615	0.3377
Valid ($N = 8$)	0.1603	0.3228
Test ($N = 8$)	0.1620	0.3454

Table 1. Limiter regression evaluation results on train, validation, and test datasets.

the model in an Encoder-Decoder manner to see if that is due to limiter parameter features, or if additional training on other parameters promotes scalable training accuracy.

4.2. Bit Depth Extraction

We compared the three models discussed on a 1000 sample subset of our training dataset.

4.2.1 CNN

The results of our CNN training are shown below.

Dataset	MSE	MAE
Train ($N = 800$)	0.0054	0.0499
Valid ($N = 100$)	0.0085	0.0716
Test ($N = 100$)	0.0062	0.062

Table 2. Bitcrusher regression evaluation results on train, validation, and test datasets trained with CNN.

As is evident, this performed much better than our previous test with our limiter. We believe this is because we only inference on one parameter (as opposed to two) and the difference that a bitcrusher effect makes might be more

obvious to learn than that of a limiter. A confusion matrix of our results are shown in figure 7.

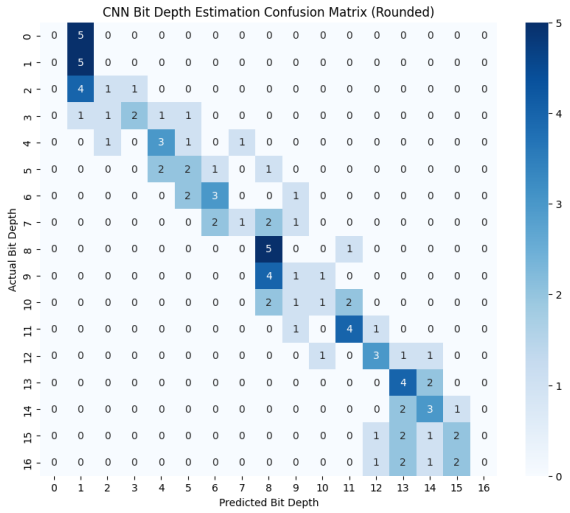

Figure 7. Confusion matrix of CNN bitcrusher test set.

4.2.2 Encoder-Decoder

The results of our Encoder-Decoder with transfer learning are shown below.

Dataset	MSE	MAE
Train ($N = 800$)	0.0249	0.1112
Valid ($N = 100$)	0.0259	0.1214
Test ($N = 100$)	0.0257	0.1195

Table 3. Bitcrusher Encoder-Decoder regression evaluation (with frozen weights) results on train, validation, and test datasets.

We can see that this performs about half as well as just using a CNN. A confusion matrix of our results are shown in Figure 8. Compared to the CNN downsampler, there appears to be fewer features represented in the 13-16 bit depth classification range.

4.2.3 UNet with Transfer Learning

The results of our UNet with transfer learning are shown below.

Dataset	MSE	MAE
Train ($N = 800$)	0.3139	0.4728
Valid ($N = 100$)	0.3202	0.4802
Test ($N = 100$)	0.3201	0.4816

Table 4. Bitcrusher UNet regression evaluation (with frozen weights) results on train, validation, and test datasets.

Due to the low performance of the transfer learning, we supplemented this Bitcrusher reconstruction with additional consideration via testing of sampling rate.

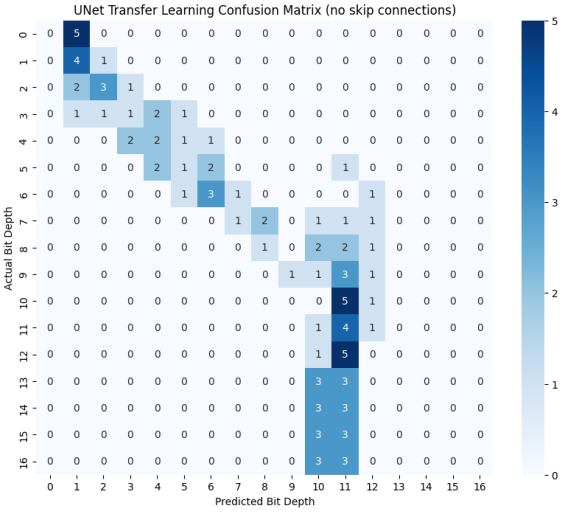

Figure 8. Confusion matrix of Encoder-Decoder bitcrusher test set.

4.2.4 UNet Reconstruction of Downsampled Audio

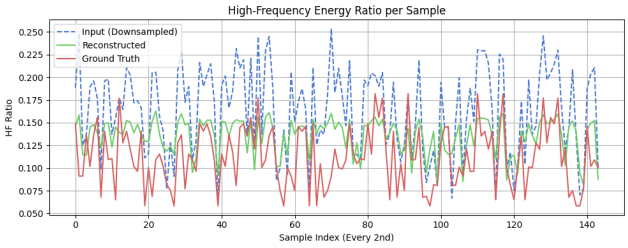

Figure 9. High Frequency Ratio Indicating U-Net Aliasing Reduction

On testing the alternatively "bitcrushed" audio, where the bitcrushing effect was emulated via downsampling of audio sampling rates as opposed to bit depth quantization, per Figure 9 our U-Net Reconstructed audio appeared to have a lower overall high-frequency energy ratio per sample than the downsampled input. This indicates that our U-Net overall has less aliasing, as high frequency content that does not match the ground truth is an indicator of poor reconstruction.

As one additional experiment in this supplement, we fed 44.1 kHz dry signals into the U-Net to see how reconstruction may be affected by the downsampling and upsampling nature of the full U-Net.

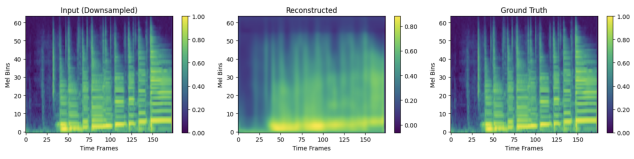

Figure 10. Spectrograms of 44.1 kHz Dry Input (left), Reconstructed U-Net Output (middle), and Ground Truth (right)

This indicates that our U-Net is able to maintain highlighted features of the original spectrogram input, even when no effect, or downsampling, is applied. However, there is this "watercoloring" effect, wherein there is interpolation occurring between feature values in the decoder when upsampling back to a reconstructed output.

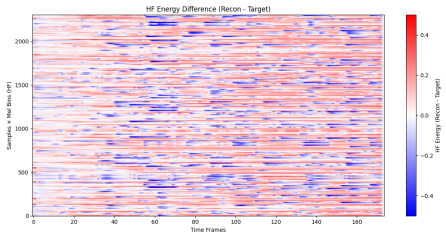

Figure 11. Reconstruction minus Target High Frequency Difference Loss Graph

The reconstruction effectiveness is found in Figure 11, where the heat map indicates that there are indications of low reconstruction loss across all frequencies (highlighted by the blue portions in the frequency domain).

4.3. Delay

We also utilized our models to estimate the delay value of a delay effect. To bolster training accuracy, 2 second recordings were extracted from all 360 samples in the original GuitarSet dataset, resulting in a total of 2800 training examples.

4.3.1 CNN

Our results with the CNN architecture on the delay parameter estimation task are show below:

Dataset	MSE	MAE
Train ($N = 2240$)	0.00025	0.01167
Valid ($N = 280$)	0.00032	0.01342
Test ($N = 280$)	0.00026	0.01212

4.3.2 UNet with Transfer Learning

Our results with pretraining the UNet, then freezing the encoder half and training only the fully connected output layer on the delay parameter estimation task are show below:

Dataset	MSE	MAE
Train ($N = 2240$)	0.00042	0.01491
Valid ($N = 280$)	0.00059	0.01751
Test ($N = 280$)	0.00044	0.01584

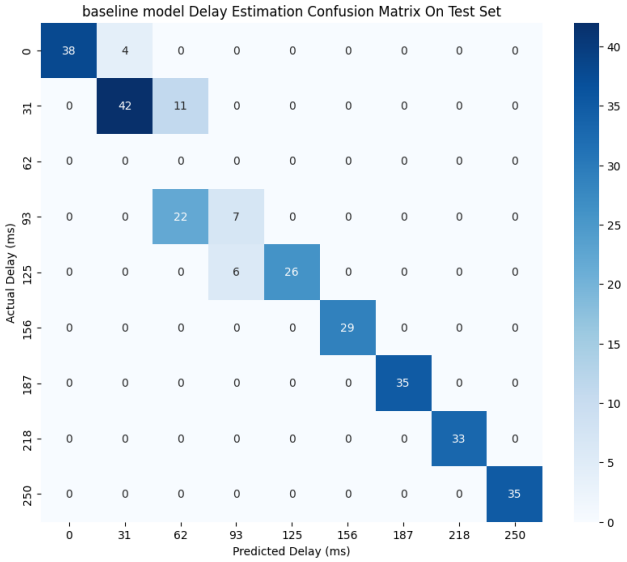

Figure 12. Confusion matrix of CNN delay test set.

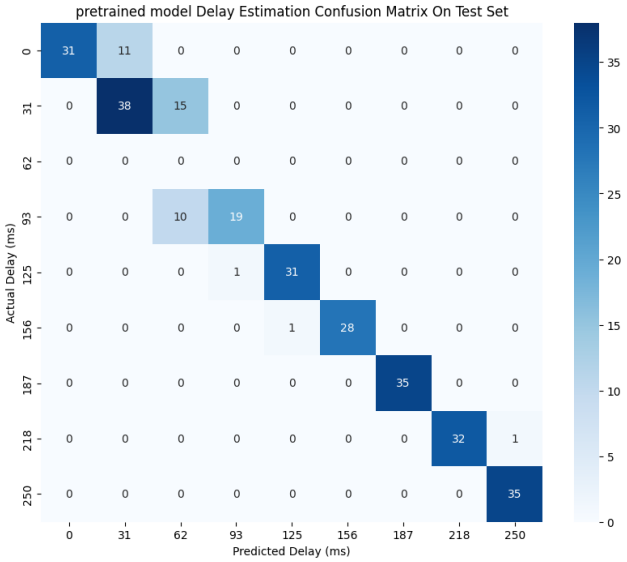

Figure 13. Confusion matrix of Unet with pretraining delay on test set.

5. Conclusion

In this work, we presented a series of designed deep learning architectures of estimating digital effect parameters trained on Mel spectrogram representations, serving as a cross-modal means of interpreting audio via advanced computer vision techniques. By constructing our own dataset from dry guitar recordings, we processed bitcrushing and sample rate changes to mimic audio codec constraints, limiter parameters for gain attenuation and envelope modulation, and delay effects for impulse response convolution effects. This incurred a structured and scalable

dataset to test our feature-vector classifier as well as a novel detachable decoder for reconstruction approach, with additional consideration of down-to-upsampling transfer learning effects of a full U-Net.

Our experiments indicated that downsampling CNNs can very effectively monitor bit depth with respect to bitcrushing plugin parameters, with promising accuracy well beyond random chance. We then amplified our model into Encoder-Decoder and U-Net structures, exploring signal reconstruction in the frequency domain as well as transfer learning on discrete parameters. While the U-Net improved alias suppression and representation quality, it exhibited reduced parameter regression accuracy when compared to the CNN, highlighting trade-offs in generalizability.

These findings reinforce notions that while deep learning can generalize and interpolate from visual features and perceptual cues in audio spectrograms, a "one-size-fits-all" model may not be optimal across diverse audio effects with varying parameter spaces when utilizing a limitedly scoped dataset. Future work could explore implementation of hybrid architectures with plugin-conditioned decoders with additional features such as self attention or cross-attention between plugin weights. We would like to continue the detachable decoder paradigm with models that are keen to navigate the parameter spaces of distinct plugins with pretraining via models such as wav2vec and CLAP [7]. Our study navigates foundations for machine interpretation of audio production landscapes, caving insight into the process for which artificial intelligence may enhance creativity and understanding of audio applications in contexts such as automated mixing, intelligent audio restoration, and neural audio codecs.

6. Acknowledgments & Contributions

We'd like to thank the CS231n instructors and course staff for a very informative quarter and their guidance leading up to this report. From conceptualization to data augmentation to model architecting and testing, the authors worked in collaboration. We are grateful to the GuitarSet and Pedalboard contributors.

References

[1] H. Jürgens, R. Hinrichs, and J. Ostermann. Recognizing guitar effects and their parameter settings. In *Proceedings of the 23rd International Conference on Digital Audio Effects (DAFx)*, Vienna, Austria, 2020. 2, 4

[2] J. P. X. Y. Q. Xi, R. Bittner and J. P. Bello. Guitarset: A dataset for guitar transcription. In *19th International Society for Music Information Retrieval Conference*, Paris, France, 2018. 3

[3] M. Rice, C. J. Steinmetz, G. Fazekas, and J. D. Reiss. General purpose audio effect removal, 2023. 2, 4

[4] T. N. Sainath, R. J. Weiss, A. Senior, K. W. Wilson, and O. Vinyals. Learning the speech front-end with raw waveform cldnns. In *Interspeech 2015*, pages 1–5, 2015. 4

[5] P. Sobot. Pedalboard, July 2021. 3

[6] C. J. Steinmetz and J. D. Reiss. Efficient neural networks for real-time modeling of analog dynamic range compression, 2022. 2, 3, 4

[7] C. J. Steinmetz, S. Singh, M. Comunità, I. Ibnyahya, S. Yuan, E. Benetos, and J. D. Reiss. St-ito: Controlling audio effects for style transfer with inference-time optimization, 2024. 3, 8

7. Appendix

For generation of our Mel Spectrograms, the following sequence of equations and transformations were used:

Framing and Windowing. Let $x[n]$ be the time-domain signal, divided into overlapping frames using a window function $w[n]$ of length N and hop size H. The k-th frame is:

$$x_k[n] = x[n + kH] \cdot w[n], \quad 0 \leq n < N \tag{1}$$

Short-Time Fourier Transform (STFT). Each frame is transformed into the frequency domain using the discrete Fourier transform:

$$X_k[m] = \sum_{n=0}^{N-1} x_k[n] \cdot e^{-j\frac{2\pi mn}{N}}, \quad 0 \leq m < N \tag{2}$$

Power Spectrogram. The power of each STFT coefficient is computed as:

$$P_k[m] = |X_k[m]|^2 \tag{3}$$

Mel Filter Bank Projection. Let $\mathbf{M} \in \mathbb{R}^{F \times M}$ be a mel filterbank matrix with F mel bins and $M = N/2 + 1$ FFT bins. The mel spectrogram is:

$$S_k[f] = \sum_{m=0}^{M-1} M_{f,m} \cdot P_k[m] \tag{4}$$

or in matrix form:

$$\mathbf{S} = \mathbf{M} \cdot \mathbf{P} \tag{5}$$

where $\mathbf{P} \in \mathbb{R}^{M \times T}$, $\mathbf{S} \in \mathbb{R}^{F \times T}$, and T is the number of frames.

Logarithmic Compression (dB scale). To mimic perceptual loudness, mel spectrogram values are converted to decibels:

$$\text{dB}_k[f] = 10 \cdot \log_{10}(S_k[f] + \epsilon) \tag{6}$$

where ϵ is a small constant to prevent $\log(0)$.

Min-Max Normalization. The dB-scaled spectrogram is normalized to $[0, 1]$:

$$\tilde{S}_k[f] = \frac{\text{dB}_k[f] - \min(\text{dB})}{\max(\text{dB}) - \min(\text{dB}) + \varepsilon} \tag{7}$$

where ε is a small constant added for numerical stability.