STEALING GUITAR EFFECTS

J. Max Morris
University of Rochester
jmorr32@ur.rochester.edu

Alex Kim
University of Rochester
akim65@ur.rochester.edu

ABSTRACT

Guitarists often look to other guitarists for new tones and sounds to use. We propose a method to identify various guitar effects and replicate them via inputting a recording with effects and applying that to a dry recording to produce that effect without knowing the parameters. This is done using features extracted from a signal with effect with non-negative matrix factorization to compare to a known dataset.

1. INTRODUCTION

Guitarists often hope to replicate other guitarist's tones. Whether this is through using the same pedals, same amps, same guitar, or same plugins, these are usually quite expensive and do not necessarily include what settings they use. We propose a method to identify and reverse engineer guitar effects from a single source to apply to other guitar recordings. This includes time-based effects as well as timbre-based effects.

There have been several different methods that attempt to do this individually, like dynamic convolution using impulse responses [1], neural networks [2], or virtual analog representations [3], but this paper proposes various unique and novel methods to accomplish this task. These are all based on the features retrieved with non-negative matrix factorization (NMF) [4]. This paper proposes ways to identify and replicate delay, tremolo, and reverb effects given a wet signal with one effect applied. We also reviewed methods for timbre-based effects for further work.

2. BACKGROUND

NMF is a method used to decompose a V matrix into two matrices, W and H [4]. In terms of audio, after applying a short time Fourier transform (STFT), we can extract various characteristics of the sound. The H matrix for instance is the activation matrix which shows note onsets. The W matrix has the spectral characteristics of said activations. To retrieve these W and H matrices, they can either be constantly changed or saved. In this case, for time-based effects, we save the H matrix of an initial training set. This set was of various clean (dry) guitar recordings. Useful information can also be extracted with the NMF of signals with effects on them (wet) for identification. These were recorded with direct input using a Focusrite Scarlet audio interface. For time-based effects, this was based on taking the Fast Fourier Transform (FFT) of the activation matrix for both identification and in some case, finding the properties of the effects we hope to retrieve. We also hoped to accomplish a method to retrieve a distortion and/or overall timbre from a different recording and apply it to a dry signal. The program asks for a wet signal that one would like to identify, a clean signal to apply it to, datasets for dry and wet effects. There are choices for pretrained data, or new data if one would like to use their own.

3. DATASET

The dataset we used consisted of two main parts: the baseline dataset, and the effects dataset. The effects dataset consists of guitar audio with various effects applied, in various amounts including, reverb, delay, tremolo, and some with no effects, all of which were monophonic music. This dataset was used for training the identification weights as well as for initial testing. For any training for the NMF, we used 100 iterations and r value of 25.

On the other side, the baseline dataset can be broken down into two components, the frequency and time focused audio. For the frequency focused audio, we used recordings of a clean guitar playing every possible note on the guitar. This consisted of 49 notes ranging from an E2 to an E6, corresponding to a 24-fret guitar's range in standard tuning. This audio was run through an NMF algorithm and used to create a Frequency Dictionary matrix (W) which represents the frequency content of a typical clean guitar. To aid with accuracy, each column of the initial W matrix consists of 0's below the 49 fundamental frequencies in a guitar's range, which can be seen in Figure 1.

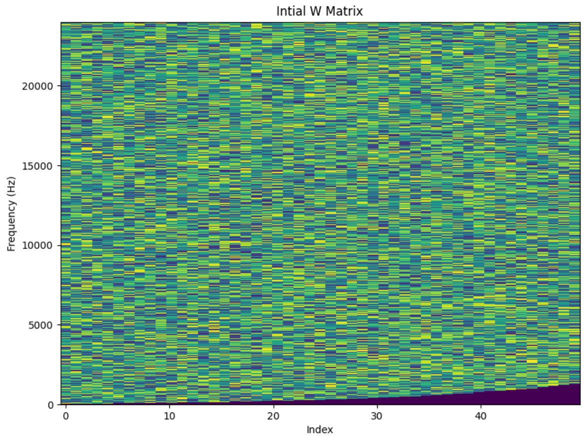

Figure 1. Initial W matrix shaped to follow pre-existing knowledge.

The W matrix output is then fed back into the next NMF algorithm, in the hopes of fine tuning the W matrix with each audio file. The resulting W matrix can be seen in figure 2.

The time focused audio consisted of recordings of a guitarist playing a song on the guitar. This audio was also run through an NMF algorithm to create an Activation Dictionary Matrix (H) which could be used to compare to temporal based effects. An FFT is then applied to the combination of all the rows, providing a baseline curve that can be used as a comparison for identification. An example of this can be seen in Figure 3.

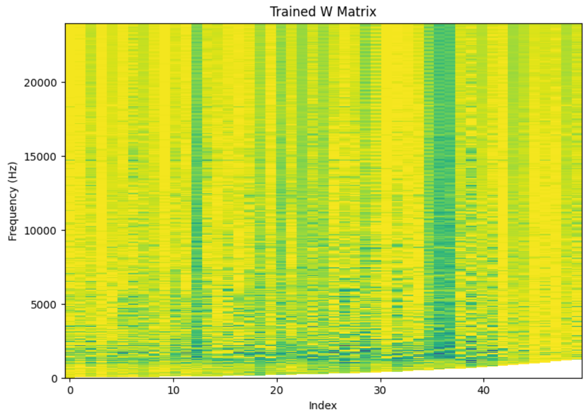

Figure 2. Trained Baseline W matrix.

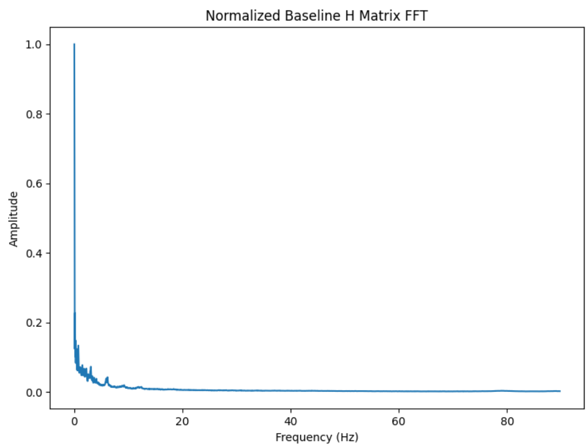

Figure 3. FFT of Baseline H matrices.

4. METHOD

4.1 Time Based Effects

The three time-based effects we looked at were tremolo, delay, and reverb. For each of these effects, the first step of each algorithm is to run the input signal through an NMF algorithm to extract the activation matrix (H) with size [r,n]. In some cases, we combined all the rows of the H matrix to produce a master H matrix, with size [n], for the audio. This ends up approximating the amplitude per frame for the audio. Either the amplitude per frame or the full H matrix is useful when attempting to derive its characteristics for replication.

4.1.1 Tremolo

The two controls that the tremolo algorithm aims to find are the frequency and depth of the amplitude modulation. To find the depth, we first take the time derivative of the master H matrix and then find all the zero crossing indexes. This provided us with a map to show when the amplitude changed directions. An example of this can be seen in Figure 4.

Using this we then looked to find the average percent change in amplitude between two consecutive peaks. On top of this, we only considered peaks where the following peak had a lower amplitude. The ratios between all these peaks are then averaged to find the predicted depth value.

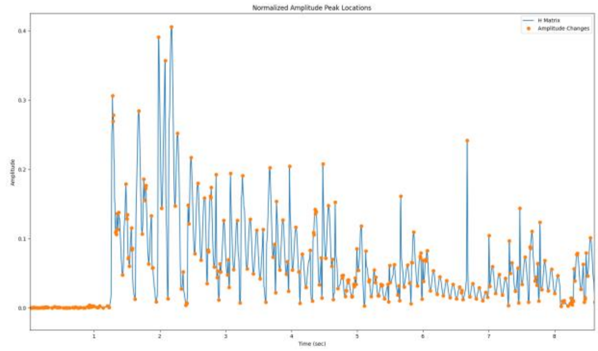

Figure 4. Zoomed in graph of peak amplitudes on top of the Master H matrix for a tremolo signal.

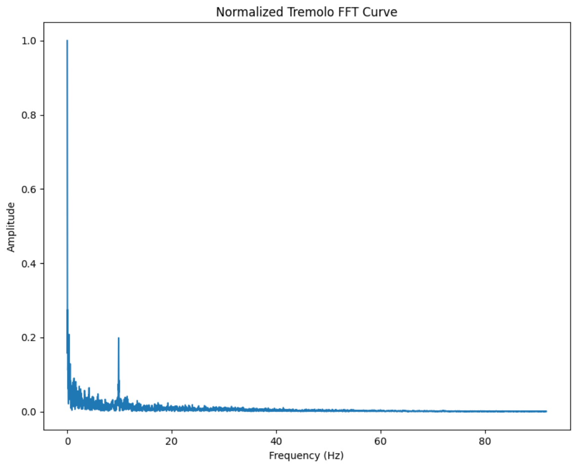

Figure 5. FFT of the master H matrix for a signal with a 10Hz tremolo effect applied.

For the frequency control, we applied an FFT to the master H matrix, which can be seen in Figure 5. For comparison, we also used our baseline dataset to find an expected FFT curve. We then subtract the baseline FFT from the tremolo FFT. This results in a frequency response with a strong peak at the frequency of the tremolo effect. An example of this can be seen in Figure 6. We then filter out amplitudes less than or equal to 0.1 to isolate this peak. An example of this can be seen in Figure 7.

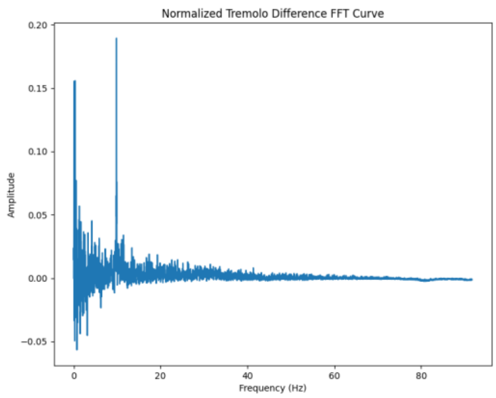

Figure 6. Difference of FFTs of the master H matrix for a 10Hz tremolo signal and baseline signal.

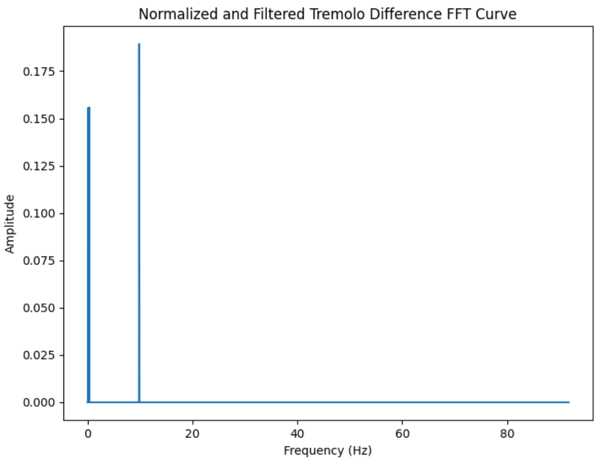

Figure 7. Filtered difference of FFTs of the master H matrix for a 10Hz tremolo signal and baseline signal.

We identify this peak and then use parabolic interpolation to determine the exact frequency identified. Finally, using this information we can generate a low-frequency oscillator (LFO) to control the amplitude of the provided clean audio. Once this is applied, the modified signal is then output to a rendered file.

4.1.2 Reverb

For this effect, the algorithm takes a slightly different approach compared to the previous one. This algorithm first finds the decay time, and then uses it to create an impulse response (IR) that can be convolved with the dry signal. For the decay time, it looks to find the average number of frames it takes the envelope of each row in the H matrix to go from a peak amplitude to below a specified threshold. For our tests, we used a threshold of 0.05. An example of one instance of an H matrix can be seen in Figure 8. Once this is known, it looks to find the average amplitude curve over that number of frames. Again, this is done for every row of the H matrix, and the results are averaged. An example of this can be seen in Figure 9.

Once the decay time and decay curve are found, we then look at the STFT of the wet audio. It then finds the decay curve, using the same method mentioned for the amplitude curve, for each frequency bin. Once each frequency bin's decay is found, it is multiplied by the amplitude curve, to ensure the IR follows the amplitude decay. An example STFT of a calculated IR can be seen in Figure 10. An inverse STFT function is then applied to the calculated IR to convert it into the time domain. Once this is done, it is convolved with the dry audio, and the algorithm outputs the modified audio and impulse response.

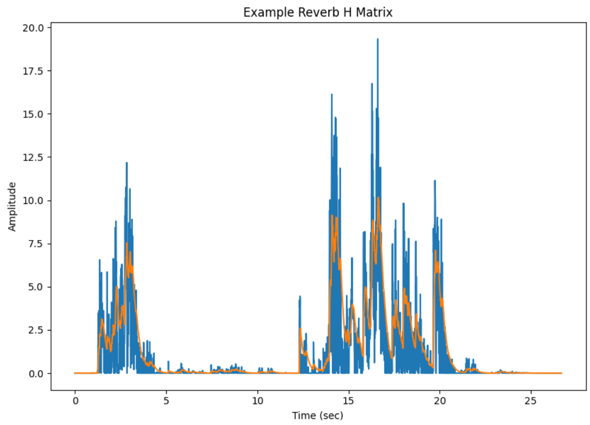

Figure 8. Example of H matrix row, with envelope and peak identification.

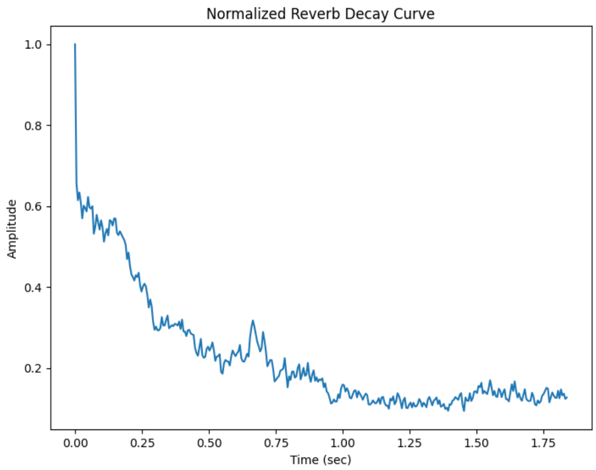

Figure 9. Example of calculated decay curve.

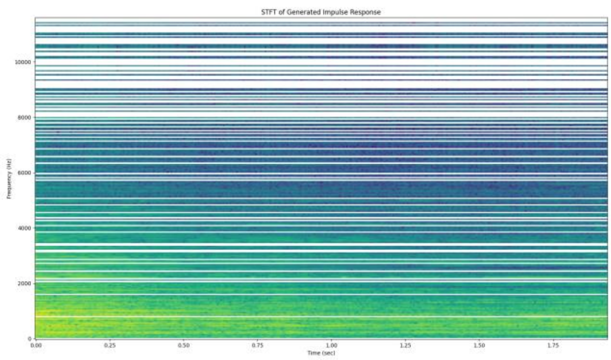

Figure 10. Example of calculated impulse response.

4.1.3 Delay

For a delay effect, there are two controls that our proposed algorithm aims to find, the delay time and the echo amplitude curve. The first control that the algorithm looks for is the delay time. To find this we used the same method used to find the frequency for the tremolo effect. Unlike the FFT results gotten from the tremolo effect, this FFT results in a frequency spectrum with small evenly spaced spikes which correlate to the delay “frequency.” An example of this can be seen in Figure 11. To determine this “frequency” we first subtract the baseline FFT from the delay FFT, giving us the graph in Figure 12. We then use peak detection to find the location of each onset.

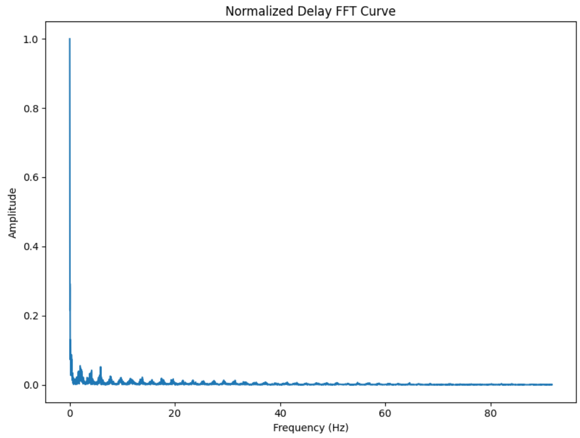

Figure 11. FFT of the H matrix of a signal with a 500ms delay applied.

We then use an algorithm to find the most common spacing between peaks that is also equal to a peak value. This corresponds to the “fundamental frequency.” Inverting this frequency yields the delay time, which we can easily convert from frames to seconds to get a usable value.

The next control parameter the algorithm looks for is the echo amplitude curve. This curve represents how loud each delayed signal is compared to the clean signal. To find this value, we use a similar method to find the decay curve for the reverb effect. However, in this case we use the already identified peaks and compare those amplitude peaks to the amplitude peaks that are integer multiples of the delay time away.

Essentially, we are using the fact that we know when to expect another peak, based on the delay time, to help determine the amplitude change. This algorithm is run on each row in the activation matrix to get an average curve for each row. When we combine these curves, we also ignore any large rising amplitudes (>= 0.2) as we assume that is a new note, rather than an echo. An example of this curve can be seen in Figure 16.

Once this is performed the results are averaged to find the predicted delay mix. Using the delay time and echo curve, we add the clean audio to itself, delayed by the amount we found, and multiplied by the averaged curve of the amplitude decays to output the signal. We then output the clean file with delays.

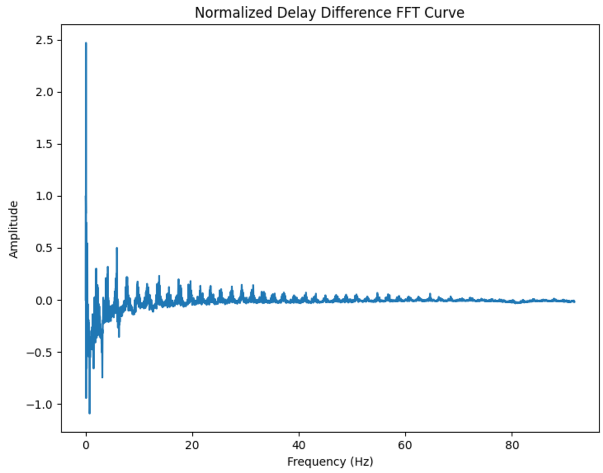

Figure 12. Difference of FFTs of the H matrix for a delayed signal and baseline signal.

4.2 Timbral Based Effects

The goal for the timbral based effects is to be able to transfer the timbral information from a wet signal to a dry signal. The focus was on distortion effects, but the algorithm could be expanded to any timbral based effect, like equalization and amplifier/cabinet modeling.

Three main methods were tested for these effects. These were all generally based on the principal of cross synthesis [5]. The first was using the average envelope shape to adjust the frequency information of a clean signal to match the envelope of the wet signal. For this, both audio files were run through an NMF algorithm to extract the W matrix. From this, the envelope of each instance (column in the W matrix) is calculated before they are averaged together. For the clean signal, the envelope is also averaged with the baseline dataset’s averaged envelope. The wet and dry averaged envelopes are then compared to find a transform function that is then applied to each instance in the clean signal’s W matrix. From this, the modified W matrix and clean signal’s H matrix are recombined with the original phase, and an inverse-STFT is performed to generate the modified audio.

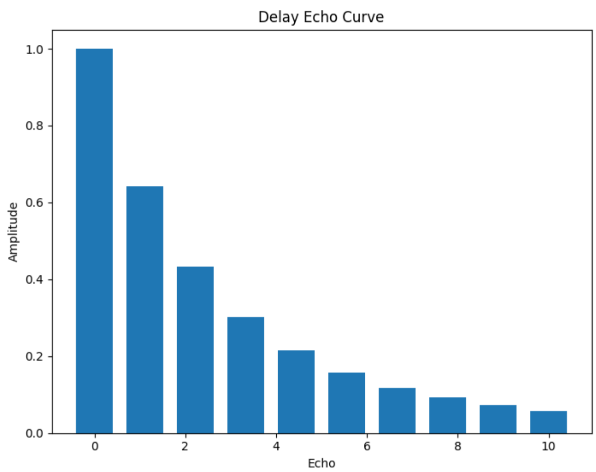

Figure 16. Example of echo amplitude curve.

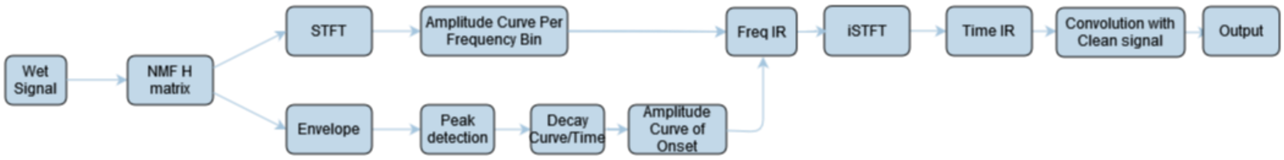

Figure 13. Reverb algorithm block diagram.

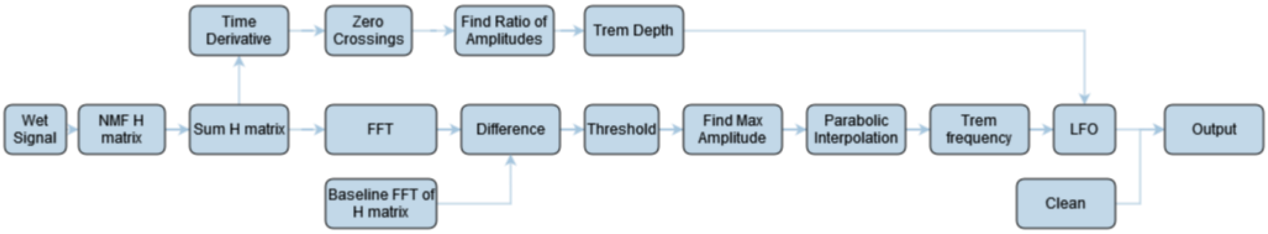

Figure 14. Tremolo algorithm block diagram.

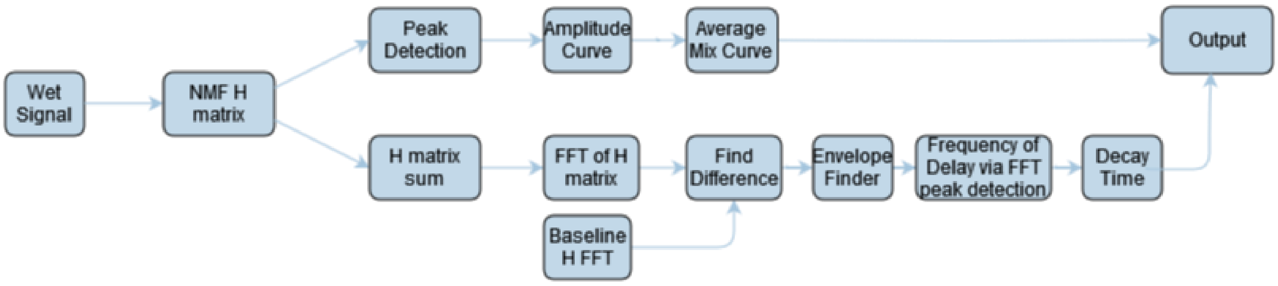

Figure 15. Delay algorithm block diagram.

The second method improves upon the first by looking at the average harmonic series amplitudes, rather than the average envelope. Similarly, to the first method, the clean signal's W matrix is joined with the baseline W matrix. From this, the peaks in each instance of both the wet and clean W matrices are identified. Then using the same algorithm used for the delay effect, find the most common spacing between all the peaks. This allows us to identify the fundamental frequency and the peaks that correspond to the harmonic series.

Examples of this can be seen in Figure 17 and Figure 18. We can average the amplitude of the first 16 overtones, which can be seen in Figure 19. We then calculate a transfer function based on the ratio between the wet and clean signal's harmonic series amplitudes. This transform function is then applied to the provided clean signal, which goes through the same algorithm to identify the harmonic series, before the transform function is applied.

The final method we looked at used the tanh function to approximate distortion specifically [6]. As a result, we tried to implement a machine learning based method that applies a tanh function to the inputted clean signal, and then compares it to the provided wet signal, to update the approximation. The planned method to find the difference between the two signals was to compare Mel Frequency Cepstrum Coefficients (MFCC) of both signals. However, we were unable to fund an effective equation to handle this.

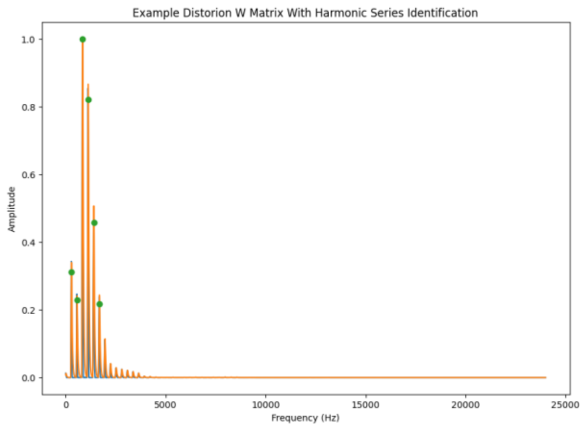

Figure 17. Predicted overtone series of one column of W matrix of clean signal.

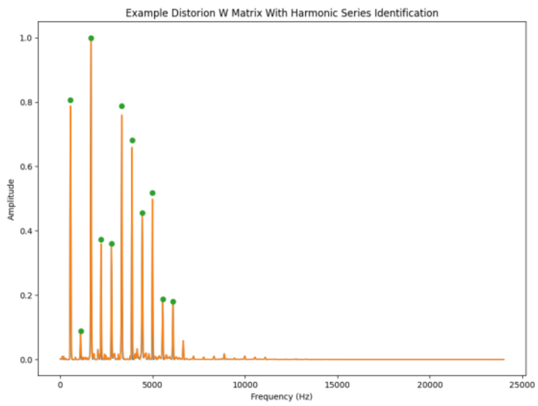

Figure 18. Predicted overtone series of one column of W matrix of distorted signal.

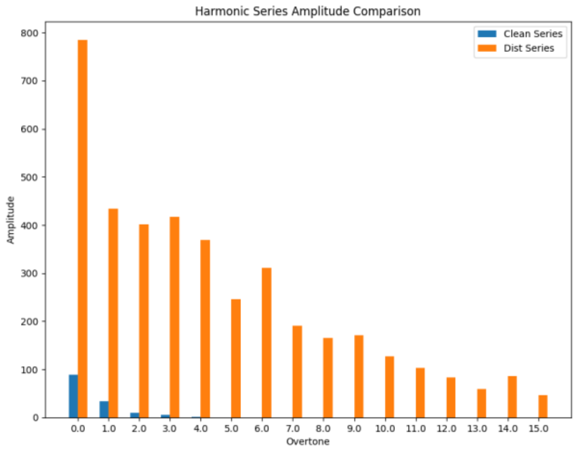

Figure 19. Comparison of overtone series for clean and distorted signal

4.3 Identification

The final aspect of this algorithm to consider is its ability to identify what effect is applied to the provided signal. For timbral based effects, this method has not been implemented. As a result, this will be a discussion of a proposed method. The proposed algorithm will calculate the MFCC of the provided audio. An example of these can be seen in Figure 20.

It will then compare this to a pretrained library of MFCCs. This pretrained library will be used to find some equation that is able to represent where most values of the MFCCs are expected to be. Then these equations will be applied to the MFCC of the provided audio, and the percentage of values that fall within the ranges of this equation will be recorded. The equation that produces the best match will then be selected as the predicted effect.

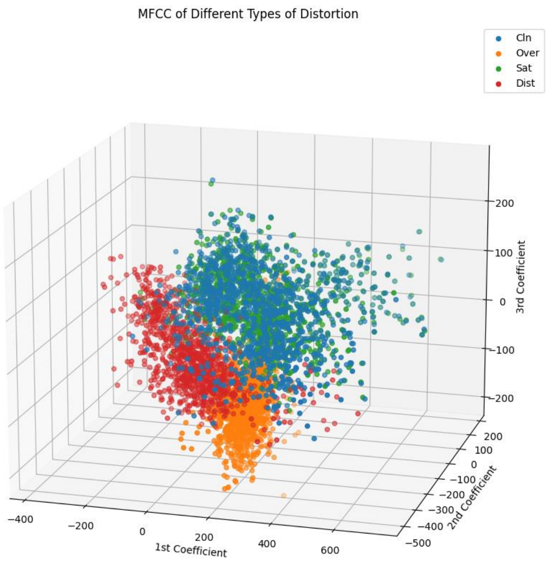

Figure 20. MFCC Coefficients 1, 2, and 3 for different types of distortion

For time-based effects, it will use the FFT of the H matrices for comparison. From our testing, we have identified that the FFTs of the H matrices vary significantly enough between a tremolo, delay, and reverb effect to allow for identification. To differentiate between them, we look at the number of peaks (Peak) above 0.01 and the number of large peaks (LPeak), which are the peaks above 0.1. Since the FFT of a tremolo effect results in one large spike and significant low amplitude noise, it has many total peaks and the largest peaks. The many small amplitude peaks in the FFT of a delay effect, results in a moderate number of total peaks, and typically has no large peaks. The FFT of a reverb effect is typically very similar to the clean baseline, and as a result it has the least number of total peaks, and few large peaks. To help with identification e also looked at the difference between a clean guitar FFT curve and the baseline. We found that the FFT of a clean signal is very similar to the baseline, but has a lot of low amplitude noise, like the tremolo effect. As a result, it has a similar number of total peaks as the tremolo effect, but with less large peaks, allowing the two to be differentiated. To determine these values, we trained the algorithm on our effects dataset, and the resulting values can be seen in Table 1.

	Clean	Tremolo	Delay	Reverb
Peak	82.77	86.33	29.75	13.5
LPeak	1.69	3.166	0	1.5

Table 1. Trained identification weights

To calculate the match percentage, we use the following equations. In all cases, if the value calculated for A from Eqn (1) is less than 0, we use 0 in place of a negative value. If the expected LPeak (Exp LPeak) is above zero, we use Eqn (2), and in this case, if B is below 0, we use 0 in place of a negative value. If B is 0 and Exp LPeak is 0, then we assume B/(Exp LPeak) is 1. In all other cases we assume the second term is 1/B, in this case we take the absolute value of B.

$$A = Exp\ Peak - |Exp\ Peak - Peak|$$
$$B = Exp\ LPeak - |Exp\ LPeak - LPeak|$$

Equation 1. Expected minus calculated distance.

$$\left(\frac{A}{Exp\ Peak} + \frac{B}{Exp\ LPeak}\right) * 0.5 = Closeness$$

Equation 2. Closeness percentage for when Exp LPeak does not equal 0.

5. RESULTS

5.1 Time Based Effects Results

Our results for replicating the delay are as follows. For identifying the delay time, the algorithm can correctly identify the delay time within 14% error. However, due to the nature of using NMF algorithms, which are random, the algorithm can occasionally result in guesses within 22% error. This is likely due to some of the peaks in the FFT being less prominent, making it harder for the algorithm to separate the peaks from noise. For the mix percent value, the algorithm can correctly identify the value within 44% error. This is most likely due to the calculated echo curve, still being susceptible to other note onsets adjusting the amplitude values. The specific results for the delay effect can be seen in Table 2.

	Run 1	Run 2	Run 3	Expected	Lowest Error	Avg Error
Delay Time	585ms	519ms	611ms	500ms	4%	14%
Mix	71%	73%	72%	50%	42%	44%

Table 2. Delay effect results for delayed signal with 500ms delay time.

Next, our results for replicating the tremolo are as follows. For the frequency, the algorithm can correctly identify the value within 2% error. On the other hand, for the depth, the algorithm can correctly identify the value within 31% error. This is likely due to noise in the activation matrix, and the peak identification, that causes inaccuracies in the amplitudes. The specific results for the tremolo effect can be seen in Table 3.

	Run 1	Run 2	Run 3	Expected	Lowest Error	Avg Error
Freq	9.84Hz	9.84Hz	9.84Hz	10.055Hz	2%	2%
Depth	41%	42%	42%	60%	30%	31%

Table 3. Tremolo effect results for tremolo signal with 10Hz modulation frequency.

Finally, our results for replicating the reverb are as follows. For the decay time, the algorithm can correctly identify the value within 8% error. The specific results for the reverb effect can be seen in Table 4.

	Run 1	Run 2	Run 3	Expected	Lowest Error	Avg Error
Decay Time	1.838 sec	1.835 sec	1.932 sec	2 sec	3%	7%

Table 4. Reverb Effect results for reverb signal with 2 second decay time.

Listening to the outputted audio, the algorithm can apply a similar sounding reverb to the audio, however, there is still some artifacts and lack of clarity.

For the time-based identification, we found that our algorithm was able to consistent correctly identify the effect. A confusion matrix showing the percent match the algorithm found for each inputted signal can be seen in Table 5. For clarity, the highest match has been highlighted in green, and any other match percents 50% or above have been highlighted in orange.

Guess	Input Signal			
	Clean	Tremolo	Delay	Reverb
Clean	69.83%	25.58%	50%	37.8%
Tremolo	65.82%	65.6%	35.37%	23.66%
Delay	25%	16.67%	92.86%	72.97%
Reverb	33.33%	0%	33.33%	81.48%

Table 5. Confusion matrix between inputted signal and the algorithm's identification guess.

Looking at this table, while the algorithm correctly identified each effect, some had higher percentages with other effects. This is most likely due to similarities with the identification weights. For example, the clean and tremolo weights are almost identical, and the inputted clean signal had a large match percent for being a tremolo. As a result, it would be expected that these two have similar match percents, as the difference between the number of large peaks is the only value that is useful for differentiating the two effects. For the delay/clean and reverb/delay confusion, it is likely due to the closeness of the number of large peaks. Since they are so close, it is very likely that the match percent is inflated as a result.

5.2 Timbre Based Effects Results

Our results for replicating timbre-based effects are as follows. All three methods discussed are unable to accurately recreate the provided effect. The first two methods, involving the envelope and harmonic series amplitudes, result in audio which consists of the clean audio with significant artifacts and unintentional distortion. On the other hand, the third method, tanh approximation, can apply a more "typical" distortion sound but is unable to match the exact tone from the provided wet signal. There is also the issue of returning the original phase, which there are some solutions, like the Griffin-Lim algorithm. This is important for when we return the inverse-STFT to reproduce the sound with minimal artifacts.

6. DISCUSSION

6.1 Summary

This work can apply multiple effects to a provided clean guitar audio file, given a guitar recording with a single effect. This was tested using monophonic guitar recordings. The baseline dataset did include polyphonic guitar recordings (strummed with chords), but the effects data set did not include any. We found success in retrieving tremolo, delay, and reverb effects with usable results, while timbre was not useable. The timbral results we did obtain were interesting as a unique effect, but not the replication we intended.

6.2 Future Work

The large scope of this project requires various techniques to process each individual effect. While this may be more intuitive to figure out and process than a neural network, it requires a lot more study into methods per effect. We hope that improvements on timbre replication as well as a better approximation of delays can be done.

While these techniques are not exclusive to guitar, we focused on typical effects that guitarists use. We have not attempted to see how our program reacts with non-guitar effects, which may produce interesting results. We also have only tested with supplying our program with input signals of singular notes. Our methods should be robust to chords for time domain effects. We have not tested identifying signals that have more than one effect. This project set out to obtain guitar effects from a supplied recording. With some training already applied, we can take a single effect and apply it to a signal we want. This proves that at a minimum, this is a viable concept, but there are various aspects to improve. This include, on a broader range of topics, various effects and being robust to multiple effects. This would mean adding more common effects as well as being able to identify them. On a smaller scale, we believe that each method we use in this: tremolo, reverb, and delay, could be more robust to minute changes, as well as more consistent in identification and replication. We also acknowledge the small dataset we used, which were selfsupplied recordings. Another consideration, given more time, would be to use a much larger dataset to develop a more accurate reference and see if and or how the results change.

7. REFERENCES

[1] M. Kemp, "Analysis and Simulation of Non-Linear Audio Processes using Finite Impulse Responses Derived at Multiple Impulse Amplitudes," *AES 106th Convention,* Munich, Germany, 1999,

[2] J. Engel, L. Hantrakul, C. Gu, A. Roberts, "DDSP: Differentiable Digital Signal Processing," *International Conference on Learning Representations*, Online 2020.

[3] D. T. Yeh and J. S. Abel, Automated Physical Modeling of Nonlinear Audio Circuits for Real-Time Audio Effect – Part 1: Theoretical Development, "*IEEE Transactions on Speech and Audio Procession,"* vol. 18, no. 3, March 2010.

[4] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature Journal*, vol. 401, pp 788-791, October 1999.

[5] G. Roma, O. Green, P. Tremblay, "Audio Morphing Using Matrix Decomposition and Optimal Transport," in *International Conference on Digital Audio Effects,* Online, 2020.

[6] J. T. Colonel, M. Comunita, J. Reiss, "Reverse Engineering Memoryless Distortion Effects with Differentiable Waveshapers," *AES 153rd Convention,* New York, NY, 2020.