BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

MACHINE LEARNING IN AUDIO EFFECTS

STROJOVÉ UČENÍ V AUDIO EFEKTECH

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR / **AUTOR PRÁCE**	**JAKUB SYCHRA**
SUPERVISOR / **VEDOUCÍ PRÁCE**	**prof. Dr. Ing. JAN ČERNOCKÝ**

BRNO 2024

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

155963

Institut:	Department of Computer Graphics and Multimedia (DCGM)
Student:	**Sychra Jakub**
Programme:	Information Technology
Title:	**Machine learning in audio effects**
Category:	Signal Processing
Academic year:	2023/24

Assignment:

1. Get acquainted with principles of digital music effects and software tools for their creation and integration.
2. Survey current techniques for machine learning applied in the area of aduio effects (for example from ICASSP or DAFx conferences).
3. Select a suitable music processing technique including its software implementation and analyze it.
4. Based on your findings, suggest and implement audio effect classification and parameter estimation technique.
5. Design, perform and analyze objective assessment and subjective listening tests.
6. Create a poster or short video about your work

Literature:
according to supervisor's recommendation

Requirements for the semestral defence:
points 1-4

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Černocký Jan, prof. Dr. Ing.**
Head of Department:	Černocký Jan, prof. Dr. Ing.
Beginning of work:	1.11.2023
Submission deadline:	9.5.2024
Approval date:	11.4.2024

Abstract

Reverse engineering audio effects from mixed tracks is a complex topic requiring signal processing and music engineering experience. This work aims at creation of a system capable of identifying the sequence and parameters of guitar effects from a mixed audio track. Training data was created using clean guitar sounds from IDMT-SMT-Audio-Effects, augmented by known effects (BitCrush, Chorus, Clipping, Compressor, Delay, Distortion, High-pass filter, Ladder filter, Low-pass filter, Limiter, Phaser and Reverb), all implemented with a Python wrapper around standard VST effects. The system is based on VGGish neural network architecture with several classification (presence of effects) and regression (parameters of effects) heads. The performance of the algorithm is evaluated on classification and regression accuracy, as well as in informal listening tests.

Abstrakt

Získávání hudebních efektů z mixovaných skladeb je složité téma, které vyžaduje znalosti jak v oblasti zpracování signálů, tak zkušenosti s audio inženýrstvím. Tato práce cílí na tvorbu systém, který by byl schopen identifikovat sekvence a parametry kytarových efektů z mixovaných skladeb. Trénovací data byla vytvořena za využití čistých kytarových zvuků z datasetu IDMT-SMT-Audio-Effects. Tyto data byla následně augmentována populárními kytarovými efekty (BitCrush, Chorus, Clipping, Compressor, Delay, Distortion, High-pass filter, Ladder filter, Low-pass filter, Limiter, Phaser a Reverb), které byly implementovány pomocí knihovny Pedalboard tvořící mezivrstvu mezi jazykem Python a standartními VST efekty. Samotný rozpoznávací systém je založený na architektuře VGGish, k níž jsou přidány klasifikační (přítomnost efektu) a regresní (parametry efektů) hlavy. Výkon modelu je hodnocen na základě přesnosti klasifikace a regrese, a také v neformálních poslechových testech.

Keywords

sound, signal, audio effect, guitar, music, source separation, machine learning, neural network

Klíčová slova

zvuk, signál, zvukový efekt, kytara, hudba, separace zdrojů, strojové učení, neuronové sítě

Reference

SYCHRA, Jakub. *Machine learning in audio effects.* Brno, 2024. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor prof. Dr. Ing. Jan Černocký

Rozšířený abstrakt

Tato práce vychází z průzkumu hudebních nástrojů a schopoností digitálních hudebních efektů. Hudební efekty jsou podstatnou částí tvorby hudby a v některých žánrech se jedná o element, který je vždy přítomen. Tyto efekty mohou být buď analogové ve formě krabiček, které ovlivňují elektrický signál kytary a tvarují ho dle specifických potřeb umělce, nebo mohou být implementovány digitálně, obvykle v rámci softwarových nebo efektových balíčků. Samotný proces výběru je velice závislý na každém umělci a lze konstatovat, že takovýto proces není řízen nějakým souborem pravidel.

Už fakt, že hudební efekty které ovlivňují nějaký nástroj, lze považovat za chaotický element, značně komplikuje proces replikace těchto efektů. Zkoumaný nástroj je navíc mixovaný v rámci skladby s dalšími nástroji, které také mohou být ovlivněny dalšími druhy efektů. Ve finále je tedy námi pozorovaný nástroj s efekty součástí celku, ve kterém lze velice těžko odhadovat, které efekty byly použity, jelikož dané frekvence a charakteristiky, které můžeme se specifickým efektem spojovat mohou splývat ve skladbě.

Celý tento proces stojí na odhadu, který je pro amatérské hudebníky nedosažitelný a i s danou zkušeností může být tento odhad daleko od reality. Na základě této problematiky se tato práce soustředí na analýzu tohoto problému a implementaci řešení, které by bylo schopné efekty odhadnout včetně jejich parametrizace, lépe než lidský odhad.
Práce je rozdělena do následujících bodů:

- Analýza a průzkum řešení věnující se separaci mixovaných skladeb
- Implementace algoritmu, schopného odhadnout efekty a následně jejich parametrizace z kytarové skladby
- Poskytnutí odhadu a následná replikace efektového řetězce

Výběr separačního systému byl založen na existující soutěži, která mapovala jejich výsledky. Následná analýza výběru systémů proběhla v profesionálním studiovém prostředí a za pomocí těchto výsledků byl vybrán specifický systém, který danou analýzu nejlépe zvládal. Tento systém i přes dobré výsledky není perfektní a obsahuje nedostatky v podobě ořezávání frekvencí nebo prolínaní s jinýmy nástroji. Implementovaný detekční systém byl trénovaný na datasetu augmentovaném v rámci této práce a dosahoval dobrých výsledků v rámci klasifikace přítomnosti efektu s průměrem 75 % přesnost mezi 12 efekty. Následný odhad parametrů poskytuje dobré výsledky pouze v podmnožině případů a dosahuje průměrné chyby střední hodnoty 0.23, kde parametry jsou normalizovány do rozmezí 0.00-1.00.

Systém, který provede samotný odhad a následně rekonstrukci je implementován jako aplikace konzolové řádky, kde uživatel poskytne část skladby s efektem, který chce odhadovat a je mu v textové podobě ukázán odhad těchto efektů a parametrů. Následně je uživateli poskytnuta rekonstrukce efektového řetězce, na uživatelem specifikovaném vstupním a výstupním zařízení.

Machine learning in audio effects

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of prof. Jan Černocký. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

....
Jakub Sychra
May 8, 2024

Acknowledgements

I would like to thank my supervisor Jan Černocký for his guidance, advices and support throughout this thesis. I would also like to thank Tomáš Trkal for the help and provided resources during the Source Separation testing. Finally i would like to thank all the people that participated in the Listening Experiments.

Contents

1 Introduction **2**

2 Basics of Audio Effects and Machine Learning **4**

2.1 Digital audio and signals 4
2.2 Audio effects 5
2.3 Neural Networks 9

3 Evaluation of Music Source Separation Evaluation Techniques **13**

3.1 State of the Art 14
3.2 Demucs 15
3.3 Testing in Professional Environment 15
3.4 Conclusion 16

4 Data **18**

4.1 Reference Data 18
4.2 Data Augmentation 18
4.3 Data types 20
4.4 Data manager and loader 21

5 Neural Network design and training **23**

5.1 Past Architecture Experiments 23
5.2 Proposed Model 24
5.3 Training 25
5.4 Results 27

6 Design and Implementation of the software **28**

6.1 Pre-processing 29
6.2 Inference and results 29

7 Listening Experiments **31**

7.1 Conclusion from experiments 33

8 Conclusion **34**

Bibliography **36**

A Contents of the included storage media **38**

B Poster **39**

Chapter 1

Introduction

Audio effects are important building blocks of modern music, no matter the genre. The process of choosing effects does not follow rules, and as such, this process can be highly unpredictable. Due to this complexity, reverse engineering audio effects affecting an instrument in a mixed track can be quite difficult. Recreating a specific effect chain based on a sample requires experience in audio engineering, musical expertise, or both. With this knowledge, the process still highly relies on guessing based on specific features of the audio effects, and to make the process worse, the sample usually contains more than one stem, which can be defined as an instrument or a sound present in a mix. And thus some features may blend in together with other instruments. Even with the correct guess for an effect, the result may be sensitive to the effect's parameters or position in the effect chain, adding further unpredictability. With the ever-expanding field of machine learning, these methods can and are being used to improve tasks, which can be either very time-consuming or almost impossible to solve for people. This opens up an opportunity to explore the field of audio effects and experiment with machine learning methods with the goal of retrieving audio effect information. If this field were to improve, it would not only help musicians in terms of inspiration, practice, or just recreational playing, but it would also yield an opportunity for exploitation as the world of digital audio effects and various systems that work with audio can be quite an expensive venture. The coverage of this topic in the literature is quite scarce, and while some papers exist, their scope is rather limited as they focus on either single effects without parameters or heavily controlled combinations of effects.

The goal of this work is to analyse current music separation techniques and create a system that can be used to replace subjective evaluation and help musicians and audio engineers with the process of approximating reference audio effects from complete audio mixtures. More specifically, this work focuses on the **guitar** branch of the mix and works with its effects.

This work focuses on identifying the used audio effects and their parameters in tracks. The main challenge of this process is the presence of other instruments in that track, as these instruments cause overlapping sounds, and the feature detection and its subsequent association with parameters becomes harder. This problem and its proposed solution are described in Figure 1.1.

The underlying theory for the entire work is described in Chapter 2. This work analyses the current state of Music Source Separation models in Chapter 3 which lays the foundation for the proposed system as it allows for extracting isolated instrument tracks. For precise effect determination, a robust data set is needed as is described in Chapter 4, where a generative system is proposed to cover all needed aspects for model training based on accessible

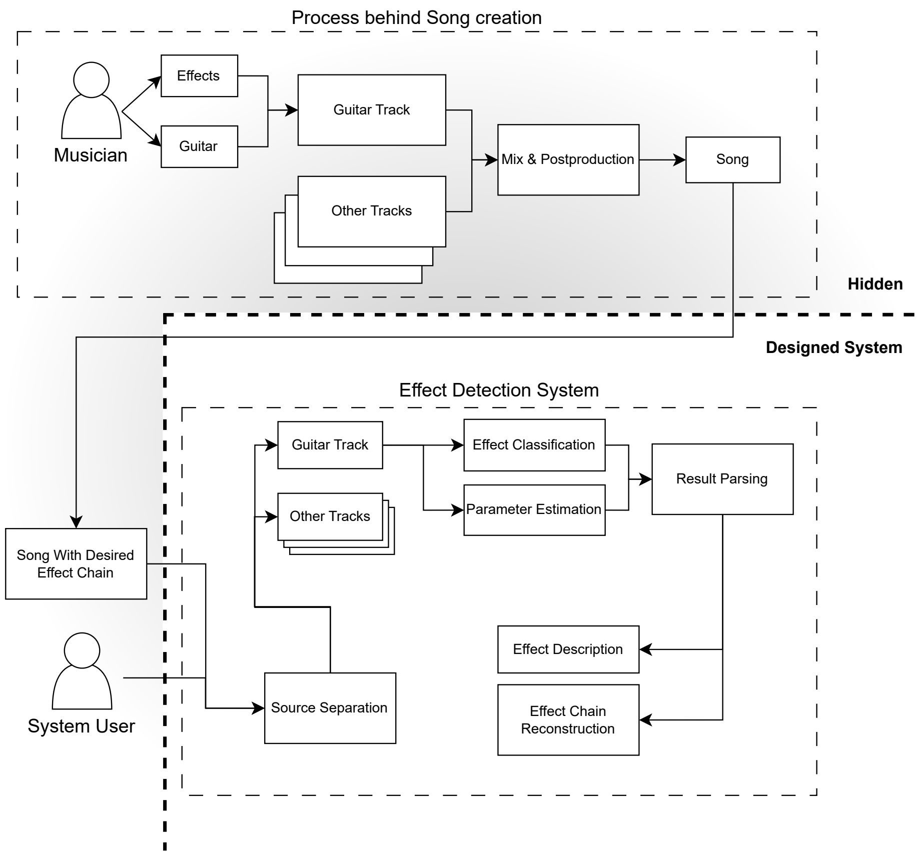


Figure 1.1: Problem and its solution.

audio effects that can be exactly replicated with the correct parameters. This chapter also describes the process of optimizing the upcoming model using datasets of various data types. In Chapter 5, the system for effect evaluation and parameter estimation is proposed, briefly mentioning experiments with several model architectures and commenting on the proposed model architecture utilising multiple model heads, establishing an effective way to load and train models within a single architecture. Chapter 6 implements the system utilising the proposed model architecture and describes the process of inference and reconstruction of detected effects as well as the issues with implementing this final layer. Chapter 7 presents the evaluation of the implemented system using listening tests. Chapter 8 summarises the results, achievements, and shortcomings of this work and describes the ways, in which this work could be improved based upon ideas that were partially implemented. Finally, this chapter goes over the potential use of this work in its current and potential future improved state.

Chapter 2

Basics of Audio Effects and Machine Learning

For the needs of this work, the reader will be presented with theory, underlining important topics that are either used in this work or that form together used systems or methods.

This begins with the essential signal theory, used in the pre-processing part and most importantly in the audio effects themselves. Afterwards, the reader is introduced to audio effects and most importantly their characteristics and implementations that are important for the Neural Network and Data augmentation. Then we deal with the general process and goals of Music Source Separations, which is an essential part of retrieving the needed data for effect determination in the final system architecture. Finally, Neural Networks are introduced with some basic theory and functions used later in this work.

2.1 Digital audio and signals

To capture, process, and reproduce sound, it is essential to understand the methods for transforming analogue audio into digital formats. These transformations involve converting continuous audio signals into a digital format that can be manipulated using digital processing techniques. This section provides a theoretical framework for understanding concepts used in this work either directly (Mel-spectrogram processing) or indirectly (Audio effects).

Fourier Transformations

The Fourier Transform is a fundamental tool used in signal processing to convert time-domain audio signal to frequency-domain representation.

The Fast Fourier Transform is a computationally efficient version of the standard Fourier Transform. It is instrumental in decomposing complex audio signals into their frequencies. This transform is essential for spectral analysis, filtering, and tasks that require identifying frequency components.

The basic equation of Fourier Transform that transforms a time-domain signal into its frequency-domain, is as follows:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt, \tag{2.1}$$

where $f(t)$ is a function in time domain, ω represents the frequency and t represents the time.

Logarithmic Mel-spectrogram

The Mel-spectrogram is a representation of audio signals that modifies the standard frequency spectrogram by aligning its frequency to the Mel scale.

The Mel scale is constructed to approach spectral analysis to human hearing. Humans perceive pitch differently than computer-based description methods, which becomes more apparent with higher frequencies. This scale tries to model the human perception of sound pitch and is based on empirical data. The most common formula is shown in equation 2.2. Using a specific frequency f, this formula finds its corresponding Mel value m on the Mel scale.

$$m = 2595 \log_{10}(1 + \frac{f}{700}) \tag{2.2}$$

Additionally, the Mel scale can be employed in audio processing using Mel filter banks consisting of overlapping triangular filters. These filters are spaced uniformly on the Mel scale, highlighting the lower frequencies. Afterwards, using logarithmic scaling compresses the dynamic range, which helps highlight acoustic features that might be lost in linear scale.

2.2 Audio effects

The main goal of audio effects is the modification of sound characteristics of the input signal. This modification is usually achieved by either manipulating the electrical signal directly using analog devices such as guitar pedals, or by utilising software tools, that are often aimed at digitally replicating the effects of analog devices.

To fully understand the features of sound that are results of modifying signal with various effects, we have to analyse the used effects first. The following section dissects audio effects used in this work and describes their features and characteristics using resources that can be found in [8, 10, 14, 15, 17].

Upcoming effects use the following general definitions as well as the dry/wet setting, controlling the ratio of the original (dry) and the processed (wet) signal:

- $y[n]$ represents output signal
- $x[n]$ represents the input signal

Delay based audio effects

These effects utilise the Delay Lines and their notable characteristic, is as the name suggests, delaying the input signal. This however is a variable process and does not only mean delaying the input by a set amount. It can for instance simulate quasi-periodic variations in pitch of a tone.

- **Delay**

  This effect plays back an audio signal after a specified delay. It is often mixed with the original audio, resulting in an echo effect. With feedback modification, the signal reoccurs continuously while losing strength with each echo until it fades completely.

The basic implementation of feedback delay:

$$y[n] = x[n-N] + g_{FB} \times y[n-N] \tag{2.3}$$

Where

- N represents the delay in samples
- G_{FB} represents the feedback gain

Such effect has two parameters, the delay time and the feedback. With feedback set to 0, the delayed sample repeats only once. This effect can be also considered the digital delay line, and as such, is the base of other delay based effects.

- **Chorus**

  Simulates timing and pitch variations, using copies of the original signal. Through recombining the original signal and its altered copies, the effect creates an illusion of several instruments playing together.

  Basic chorus implementation:

$$y[n] = x[n] + x[n - M[n]], \tag{2.4}$$

  where $M[n]$ is a modulated delay time, generally controlled by a Low-Frequency Oscillator, which can utilise various waveforms.

Filter effects

Any audio effect could be considered a filter effect based on the way they affect the signal. This section focuses on a specific types of filters, being the low-pass and high-pass, special type of filter used in synthesizers and an effect based on the all-pass filter.

The upcoming three filters are not a typical type of audio effect and their presence in this work is to enhance the tonal shaping and to allow for integration with other effects, to capture effects outside the training scope.

- **High-pass and Low-pass filters**

  These types of filters aim to eliminate a specific range of frequency from their input. The low-pass filter passes low frequencies below some cutoff frequency f_c while attenuating frequencies above this threshold. In a similar fashion, the High-pass performs the inverse process of attenuating frequencies below the cutoff frequency f_c and passing the frequencies above. Attenuation is the ability of the filter to gradually reduce the amplitude.

  A basic first-order low-pass filter can be modelled after the following equation:

$$y[n] = \alpha x[n] + (1-\alpha)y[n-1] \tag{2.5}$$

  where α is the filter coefficient that determines the amount of influence of the current input $x[n]$. Its value is determined by following the formula, using the cutoff frequency and the sampling rate represented by f_s:

$$\alpha = \frac{f_c}{f_s + f_c} \tag{2.6}$$

- **Ladder filter**

  This filter is based on the Moog synthesizer design. It can operate in various filter modes consisting of low-pass, high-pass and band-pass modes with various levels of attenuation per octave below or above the cutoff for that specific filter, meaning that the strength of frequency reduction increases by the specified level for every doubling of the cutoff frequency below or above the cutoff point.

  As the band-pass filter was not defined up until this point, the filter operates by cutting frequencies within a range around the cutoff frequency.

- **Phaser**

  This effect attenuates or eliminates frequencies, based on a set of notches in the audio spectrum. These notches are implemented using all-pass filters, that do not change the magnitude but rather introduce phase lag. The output of all-pass filters is added to the original signal, where this process creates destructive inference.

  Destructive inference occurs when two sound waves of the same frequency and amplitude converge but are out of phase by 180 degrees. This results in the waves cancelling each other out, leading to amplitude of zero.

  An example of the Phaser effect can be seen in Figure 2.1, where the left sine wave was altered using the Phaser effect into the signal on the right.

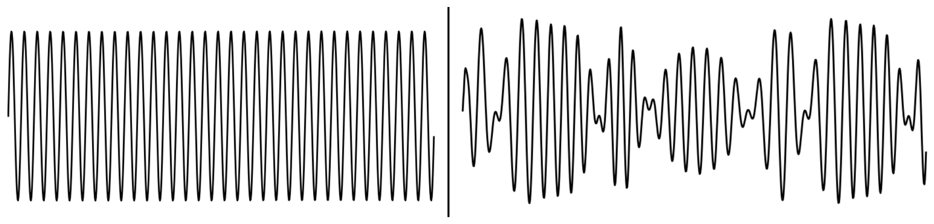


Figure 2.1: Example of the Phaser on a sine wave.

Distortion effects

These types of effects transform the signal using nonlinear transformations, creating concepts such as clipping and saturation.

- **Compressor**

  This effect is used to reduce the difference between the loudest and softest parts of audio. The following equation utilises *threshhold*, to control above which gain level is the compression applied and *ratio* determines the amount of gain reduction:

$$y[n] = x[n] \times (1 - \frac{threshold - |x[n]|}{threshold} \times ratio) \tag{2.7}$$

  Figure 2.2 shows the effect of compression on a sine wave. The signal starts the same, but its amplitude is gradually compressed towards the threshold.

- **BitCrush**

  Reduces the bit depth of the digital audio, introducing a quantization noise. This process results in a distorted and digitized-sounding tone. The process of this effect

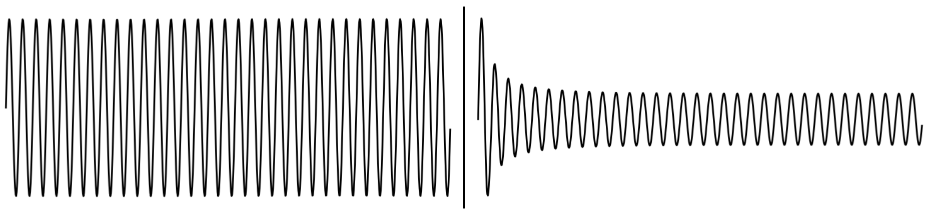


Figure 2.2: Effect of compression on a sine wave.

can be described by utilising Q as a quantization step that is determined by the bit depth:

$$y[n] = round(x[n] \times Q)/Q \tag{2.8}$$

- **Clipping**

  This effect implements a type of distortion, by forcing the signal to exceed the dynamic range, creating a clipped sound by shearing the peaks.

  Clipping can be described by Equation 2.9, which limits the output to the range of the negative threshold and the positive threshold.

  $$y[n] = \min(\max(x[n], -\text{threshold}), \text{threshold}) \tag{2.9}$$

  Figure 2.3 demonstrates the abrupt cut-off caused by the clipping, where the threshold is represented by a dotted line.

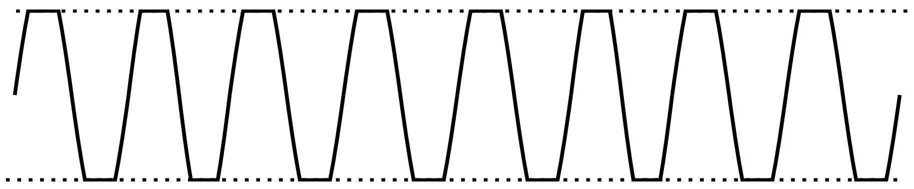


Figure 2.3: The effect of clipping on a signal.

- **Distortion**

  The distortion effect can be as simple as applying one function to the input. The following equation demonstrates this with the hyperbolic tangent type distortion:

  $$y[n] = \tanh\left(gain \times x[n]\right) \tag{2.10}$$

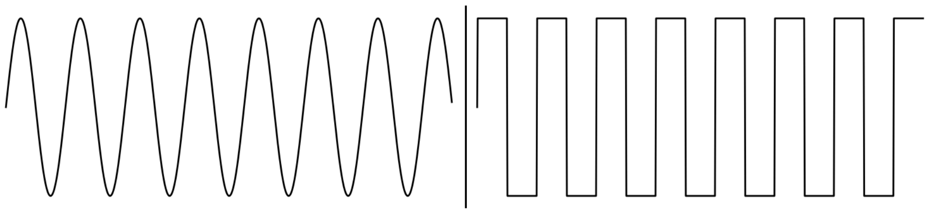


Figure 2.4: Example of Distortion on a sine wave.

- **Limiter**

  The main goal of the limiter is to control the dynamic range of the signal, so it does not exceed a specified level. The notable characteristic of the limiter is its ability to prevent clipping while maintaining a consistent loudness. Limiter can be implemented using a set of compressors, that reduce the gain of the signal followed by hard clipping.

  The effect of the limiter can be observed in Figure 2.5, the main difference from the Clipping effect, is that the signal is not abruptly cut off.

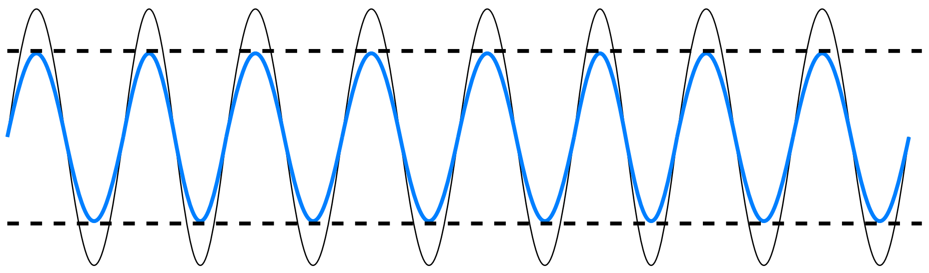


Figure 2.5: The effect of limiter on a signal.

Reverb

Reverb is designed to simulate the acoustics of physical space, adding the effect of sound reflection of surfaces, ambience and spatial depth. This effect can be implemented using a combination of feedback delay networks and filtering techniques that mimic the sound absorptions and reflections of a real environment.

2.3 Neural Networks

Based on a biological concept of neurons in a brain, an Artificial Neural Network consists of layers made of neurons modelling the synapses of a biological nature to process information. The goal of this process is to identify phenomena, weigh options and arrive at conclusions based on complex patterns in data.

In an Artificial Neural Network, neurons are assembled in layers. These layers include an input layer, output layer and one or more hidden layers as can be seen in the example in Figure 2.6.

The purpose of the input layer is to pass values into the Neural Network without modifying these values. These values are passed into the first connected hidden layer. The

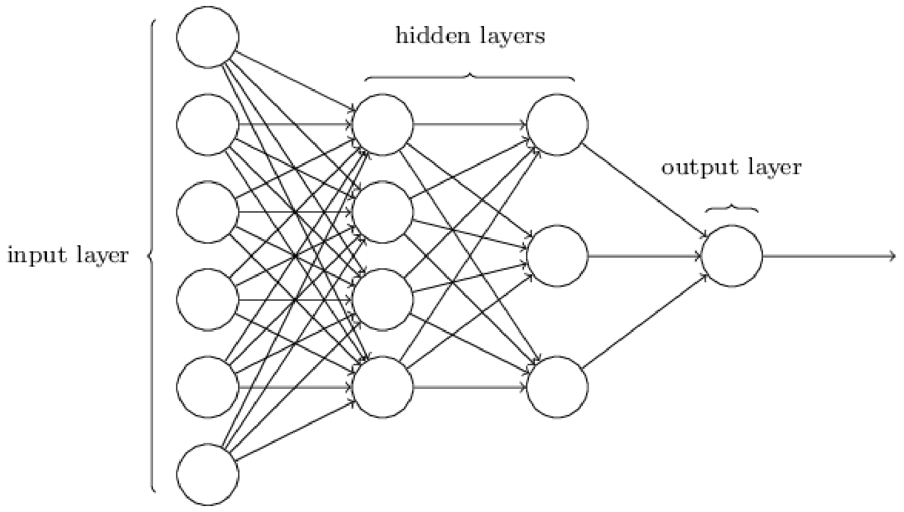


Figure 2.6: Four-layer neural network with two hidden layers. Source: [6].

depth of the neural network is represented by the amount of hidden layers, where a neural network consisting of two or more hidden layers is considered a Deep Neural Network. Each hidden layer applies some transformation to the input and passes the result into the next connected layer. The final hidden layer is connected to the output layer, which produces the final output.

Each neuron in a hidden layer consists of any number of inputs $\vec{\boldsymbol{x}}$, weights $\vec{\boldsymbol{w}}$, bias $\vec{\boldsymbol{b}}$ and forms connections to other neurons as output $\vec{\boldsymbol{y}}$. To calculate the output of the neuron, the input data are multiplied by the weight of that specific input. The weighted inputs are afterwards summed and bias is added. Bias shifts the results towards a certain point, affecting the activation function. The activation function affects the output of the neuron by enabling non-linearity.

The output of a neuron can be defined as:

$$y = f(b + \sum_{i=1}^{n} x_i w_i), \tag{2.11}$$

where n is the number of inputs, x_i represents the inputs, w_i represents their corresponding weight, b represents the bias and f is the chosen activation function.

The Matrix 2.12 represents a neural network layer with 3 neurons using the notation specified earlier. The input layer consists of 4 neurons, which are multiplied by the weight matrix. Afterwards, a bias corresponding to the given layer consisting of 3 components is added and finally, the output is passed through an activation function denoted by f.

$$\vec{\mathbf{y}} = f\left(\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right) \tag{2.12}$$

The end goal for neural networks is to adjust their weights and biases (the parameters), so that when applied to a yet-unseen example in the input, they produce the desired output [4].

2.3.1 Activation Function

The activation function determines the output of a neuron and introduces non-linearity to the neural network. The choice of an activation function shapes the output of the neural network and affects learning dynamics and model accuracy.

- Sigmoid Activation Function

  The sigmoid scales the input in the range of 0.0 to 1.0:

  $$\mathbf{y} = \frac{1}{1+e^{-x}}, \tag{2.13}$$

  where y represents the output of the function and x represents the input of the function.

- Rectified Linear Activation Function

  One of the most widely used non-linear functions, its value is mainly in its speed and efficiency:

  $$\mathbf{y} = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{2.14}$$

  This function introduces non-linearity to data without affecting neurons with positive outputs.

- Softmax Activation Function

  This activation function transforms the output into a distribution of probabilities for each defined class. The target class has the highest value. The output is shaped into the range of 0 and 1.

  This function can be represented by:

  $$y_i = \frac{e^{x_i}}{\sum_j e^{x_j}}, \tag{2.15}$$

  where y_i is the probability score for class i and the denominator $\sum_j e^{x_j}$ represents the sum of all input scores, which normalizes the sum of the output probabilities to 1.

Generally, neural networks use two activation functions, the first function is used in hidden layers and usually stays the same for each layer and the second function is used in the output layer for the final result.

2.3.2 Loss Function

To properly train a neural network model, a calculation must occur that measures its error. This process improves the model's accuracy and confidence.

Loss is the error of the model and the goal is to get it as close to 0, as possible. There are various types of loss functions, that are each optimal for different tasks.

- Binary Cross-Entropy Loss

  Common function for binary classification used between binary predictions and actual binary targets.

The calculation of the loss is based on the distance of the predicted value from the target value, where the actual target can be either 0 or 1:

$$BCE = -\frac{1}{N}\sum_{i=1}^{N}[y_i \times log(\hat{y}_i) + (1 - y_i) \times log(1 - \hat{y}_i)] \tag{2.16}$$

Where

- N is the number of samples
- y_i is the actual binary label of the sample
- $\hat{y}_i$ is the predicted probability of the sample

- Mean Squared Error Loss

  Common for regression models. This function calculates the squared and averaged difference between the predictions and the ground truth

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{2.17}$$

Where

- y is the ground truth
- $\hat{y}$ is the prediction

2.3.3 Training

All previously mentioned parts play a role in the training process. This process starts with random weight values assigned to the network and using loss calculations, backpropagation and an optimization algorithm, the network adjusts the weights in the correct direction over some number of iterations, that are specified based on the model goals and complexity [1].

The training process is repeated over numerous iterations with the goal of minimizing the loss function. A significant impact on this process is also the quality of the data, on which the model is trained. This data is split into batches and fed gradually into the network while calculating their loss.

Backpropagation

This process begins with the loss calculation. The backpropagation then calculates the gradient of the loss function by moving backwards through the network layers from the last to the first. The derived gradients indicate the direction and magnitude by which the weights need to be adjusted to reduce the loss.

Chapter 3

Evaluation of Music Source Separation Evaluation Techniques

As seen in Figure 1.1, popular songs come only in a mixed form and obtaining the original files that these songs are composed of is a nearly impossible task, especially for more popular samples, the need for source separation arises. After demixing existing songs, we can further analyse them and utilise this knowledge for our work.

This work utilises the Deep Neural Network type of Source Separation to analyse mixed tracks to extract the guitar stem and analyse its features for effect determination and subsequent reconstruction. As highlighted in Chapter 2, the field of Music Source Separation (MSS) is not flawless and as such, an optimal method has to be chosen, to best suit the needs of this work. As this work focuses on extracting information from the separated stem, its quality is a deciding factor in the quality of the final outcome, as bad separation will introduce artefacts, distortion, noise and frequency cuts to the outgoing stem which would negatively affect the effect determination. A major need is also the correct labelling of the extracted data.

This chapter aims at analysing the current market and choosing the system best suited for the needs of this work. Afterwards, tests and various experiments are performed with the chosen system to verify its suitability.

The upcoming section analyses the current state of Music Source Separators. Using this information the most suitable System is chosen and is afterwards tested using listening tests in a studio environment for its suitability for this work.

The Music Source Separation revolves around the decomposition of a mixed signal into its components. The main challenge of this process is the extraction of these components without the knowledge of the original mixture and its sources.

To tackle this topic, various methodologies have been developed over the years focusing on the separation using various features.

Spectral Clustering

This method uses the similarity measures between different components of a signal, typically frequencies across time frames to group similar sounds into clusters. Clusters then theoretically represent audio sources such as individual instruments. This method requires the selection of a number of classes, which may vary across audio samples and thus this method is not optimal for blind separation.

Non-negative Matrix Factorization

This statistical technique decomposes the magnitude spectrogram of a sample into a set of additive components, typically associated with the spectral profiles of individual sources. This method effectively isolates different instruments or sounds within a mixed audio track by approximating the original signal as a linear combination of non-negative basis functions. Each component in the resulting matrices $\boldsymbol{W}$ (basis matrix) and $\boldsymbol{H}$ (activation matrix) represents distinct sound sources and their temporal activation, respectively [13].

This method is also negatively affected in blind separation as it also requires an estimation of the number of sources.

Deep Learning Models

This approach to Music Source Separation is a trending topic due to the ever-increasing popularity of AI and machine learning in general. These models utilise Convolutional Neural Networks and Recurrent Neural Networks trained on extensive datasets to extract features and perform highly accurate separation, that surpasses the traditional methods. Utilising architectures like U-Nets allows the models to learn from local contextual information across the audio spectrum.

3.1 State of the Art

Music Source Separation is a growing space because of its various potential uses. Due to the demand for such software, there are various options to choose from when it comes to picking the right separator.

To efficiently analyse the current state of these systems, using the Music Demixing Challenge [5] organised by Sony is an efficient approach. This challenge performs a crowd-based competition, with Music Source Separation on a hidden dataset using Signal-to-Distortion Ratio as an evaluation metric. It only measures four instrument stems (Vocals, Drums, Bass, Other), meaning that the target of this work is not within the scope of the rating (focusing on a single stem accuracy is not possible) and as such, the system performance as a whole is the key to choosing the right system for this work. Additionally, some of the systems rated within this work do not include the ability to separate the guitar stems and as such, they are discarded from the inclusion in this system.

As this challenge featured two leaderboard, where the first focused on system trained exclusively on MUSDB18-HQ [9] and the second posed no limitations, both of the leaderboard forerunners were analysed for this work.

- **AudioShake**: This system was placed in first place on the leaderboard with no limitations. It performs slightly better than the second-best-performing system, which is also the upcoming Hybrid Demucs. The major downside of this system is that it is a closed system running on a paid model. As such this system is unusable for this work.

- **Hybrid Demucs**: As a system that placed first in the controlled dataset leaderboard and second in the leaderboard with no limitations with just a slight difference from the winning system. This system poses a good choice for further analysis due to its performance and also the open-sourced nature [11].

3.2 Demucs

This system was created by Meta Research and features two versions: the standard Demucs and the Hybrid Demucs, the successor to the original design.

Standard Demucs

The standard Demucs features a Deep Learning Model that operates on raw input waveform data and generates a waveform for each output source. It utilises U-net architecture with a convolutional encoder and a decoder based on wide transposed convolutions with large strides. Additionally, this model uses bidirectional Long Short Term Memory (LSTM) between the encoder and decoder [2].

- **Encoder-Decoder Structure**: allows the model to capture both low-level and high-level semantic information. The encoder reduces the dimensionality of the input, capturing features while the decoder reconstructs the output from the compressed feature representation
- **LSTM Layers**: are formed between the encoder and decoder. They are a type of recurrent neural network layers that help the model capture temporal dependencies in audio data, which helps with understanding the musical structure over time.

Hybrid Demucs

Building upon the original Demucs architecture, the Hybrid Demucs utilises both convolutional and transformer layers to process audio data. Additions from the original architecture include:

- Cross-Domain Feature Processing

  By using transformers alongside convolutional layers, the transformer blocks are capable of modeling relationships across long sequences, making them useful in capturing temporal dynamics in music.

- Self-Attention Mechanism

  Self-Attention allows the model to weigh the importance of different parts of the signal differently. This helps the model to focus on relevant features for each source.

The architecture of the Hybrid Demucs can be seen in Figure 3.1.

For the purpose of this work, an experimental model under the Hybrid Demucs architecture is used, which will be tested alongside other models in the upcoming section. This model `htdemucs_6s` separates additional 2 stems to the original vocal, bass, drums and other stems: guitar and piano, form which this work utilises the guitar stem.

As the architecture itself is not flawless and the extraction of the guitar is an experimental feature, it may come with various deficiencies that will be further discussed in Section 6.1.

3.3 Testing in Professional Environment

To evaluate the selected separator, an opportunity was presented, to consult Ing. Tomáš Trkal, due to his experience with professional sound mixing[1].

[1]http://www.tntrecords.cz/

After choosing the models to be analysed a set of tests were conducted in a studio setting to test the accuracy of these separators and verify their suitability for this work. The most important model to analyse was the experimental model allowing for the separation of the guitar stem, the other models were analysed to observe the system's general performance. The analysed models were:

- `htdemucs` - The default version of the model.
- `htdemucs_ft` - Fine-tuned version of the default model promising better results.
- `htdemucs_6s` - Default model expanded with piano and guitar separation.

For the purpose of this test, studio tracks were provided with their original sources, thus allowing the comparison between the separated stem and the ground truth. The comparison was performed by listening tests and in some cases, spectrum analysis.

First, to test out the general separation, hip-hop tracks were processed. These tracks were correctly separated without any stems leaking to others, but with light frequency cutoffs. These were minimal and the separation was very successful. Afterwards, tracks containing the guitar stem were processed. The results were identical to the previous tests without a noticeable change in quality.

After these, tests focused specifically on the guitar stem quality were run on tracks predominated by a guitar.

Tested genres were Pop, Rock and Metal. These tests were again evaluated by listening tests and results were more diverse then the previous tests. These tests shown that the model is able to separate the guitar from all track successfully, but the quality vastly differs case by case. The most problematic are tracks where the guitar blends or creates a „wall of noise". In these cases, the guitar stem becomes scattered over other stems and the resulting quality of the isolated guitar stem is inadequate. All results did not contain any major change in the quality of the stems, confirming, that the use of the expanded model, will not drastically affect the quality of the separation.

Another observation from the tests was that the guitar and bass stems do very occasionally cause issues between the two. If the track contains a higher-pitched bass or a lower-pitched guitar, one of these stems may be falsely classified as a part of the other. This issue will be expanded upon in Chapter 6.

Samples tested in the professional environment cannot be shared due to privacy reasons and as such, a reference to the quality of separation can be taken from the example page for the Hybrid Demucs[2], containing the separation of two samples and comparison of the model variations.

3.4 Conclusion

For the purpose of this work, the selected separator Hybrid Demucs with model version **htdemucs_6s** might perform well in certain cases, but there is still a level of uncertainty for edge cases as was described in testing. Fortunately for this system, the separator only forms one part of the final pipeline and thus, with the gradual improvement of Music Source Separator systems, this part can be eventually replaced if it ends up underperforming.

[2]https://ai.honu.io/papers/htdemucs/index.html

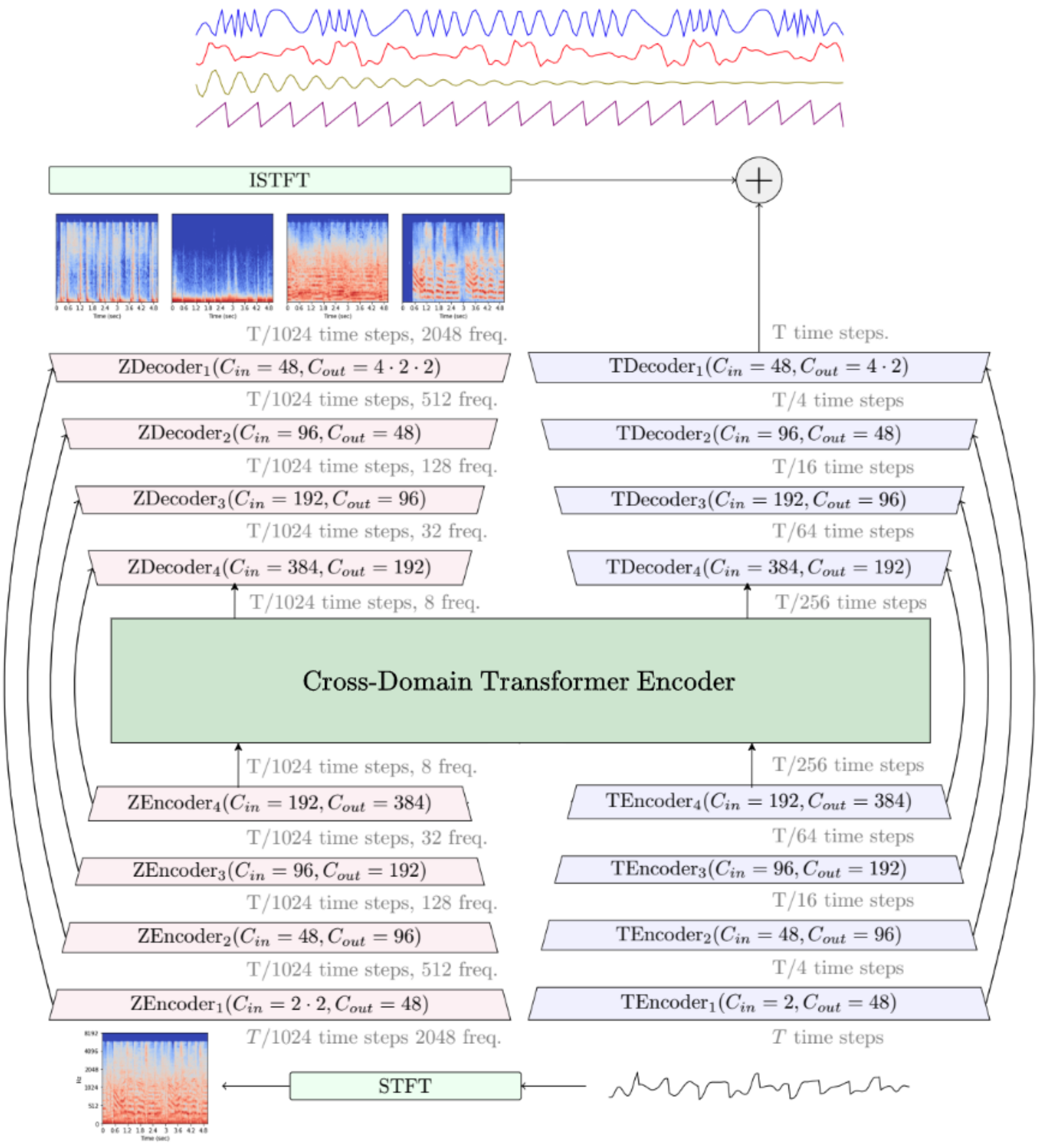

ISTFT
T/1024 time steps, 2048 freq.
T time steps.
ZDecoder$_1$($C_{in}=48, C_{out}=4\cdot2\cdot2$)
TDecoder$_1$($C_{in}=48, C_{out}=4\cdot2$)
T/1024 time steps, 512 freq.
T/4 time steps
ZDecoder$_2$($C_{in}=96, C_{out}=48$)
TDecoder$_2$($C_{in}=96, C_{out}=48$)
T/1024 time steps, 128 freq.
T/16 time steps
ZDecoder$_3$($C_{in}=192, C_{out}=96$)
TDecoder$_3$($C_{in}=192, C_{out}=96$)
T/1024 time steps, 32 freq.
T/64 time steps
ZDecoder$_4$($C_{in}=384, C_{out}=192$)
TDecoder$_4$($C_{in}=384, C_{out}=192$)
T/1024 time steps, 8 freq.
T/256 time steps
Cross-Domain Transformer Encoder
T/1024 time steps, 8 freq.
T/256 time steps
ZEncoder$_4$($C_{in}=192, C_{out}=384$)
TEncoder$_4$($C_{in}=192, C_{out}=384$)
T/1024 time steps, 32 freq.
T/64 time steps
ZEncoder$_3$($C_{in}=96, C_{out}=192$)
TEncoder$_3$($C_{in}=96, C_{out}=192$)
T/1024 time steps, 128 freq.
T/16 time steps
ZEncoder$_2$($C_{in}=48, C_{out}=96$)
TEncoder$_2$($C_{in}=48, C_{out}=96$)
T/1024 time steps, 512 freq.
T/4 time steps
ZEncoder$_1$($C_{in}=2\cdot2, C_{out}=48$)
TEncoder$_1$($C_{in}=2, C_{out}=48$)
T/1024 time steps 2048 freq.
T time steps
STFT

Figure 3.1: Hybrid Demucs Architecture, adopted from [2].

Chapter 4

Data

This chapter introduces the datasets and the methods used for augmenting the data with various Audio Effects for model training and testing of this system. This chapter also describes the methods of creating the additional data used in the system, such as Mel-spectrograms and pre-processed Tensors that optimise the Neural Network training process. Finally, the modules that handle the data splitting and loading are introduced.

4.1 Reference Data

To be able to recognise effects in mixed tracks, the training process must have a broad dataset containing not only various guitar tones and chords, but also taking into account the varying state of the instrument, different timbre, types of guitar pickups and other variables, that may alter the sound.

For this purpose, it would be appropriate to use a dataset of some music samples containing the target instrument and run it through the Music Source Separator to obtain isolated guitar parts. The downside of this method is the unknown in the form of used effects, which is the essential part of this system. Labelling such data poses another challenge as such process would rely on previously mentioned know-how and with the way that audio effects work, the final labels may not represent the true setup used by the musician.

For this reason, data has to be obtained with known effects and known parameters and also optimally with a way to recreate these effects and parameters for reconstruction.

With these points in question, a method of creating a custom dataset was chosen. To ease the creation process a reference dataset containing clear guitar samples of tones and chords, played with various guitars was selected.

IDMT-SMT-Audio-Effects Dataset [16] contains 2 electric guitars with various plucking styles and pick-up settings. This results in around 500 clean guitar samples with total length of 17 minutes, that can be further augmented for the purpose of this work.

4.2 Data Augmentation

Audio effect can be applied using either mathematical functions to some signal, or by using an existing tool, usually in the form of a VST plugin. VST plugins are loaded into Digital Audio Workstations and operated in a way, that usually does not allow for a procedural generation, that is needed for the augmentation. For this purpose, systems like Pedalboard [15] exist. Serving as a Python library that allows the use of VST within code, and thus allowing for easy data augmentation. This library also implements basic effects, that are used to generate data for this system. The types of effects used and their general features are described in Chapter 2.2.

The augmentation is implemented with a simple methodology. The augmentation system randomly picks a sample from all available guitar samples, randomly determines how many effects to use, randomly chooses their parameters and creates a configuration string to be saved in a CSV file. The system then hashes the configuration string to be used as a file name for two purposes, the first being that the whole configuration would exceed the character limit of files, and the second being that it allows for easy checking, if a duplicate exists, in which case the configuration is discarded. Unique configurations are then written into a CSV file containing a list of files and their configurations and the augmented sample is created with that configuration.

Using this method, a total of 101k audio samples are created with a total duration of 61 hours, affected by 0-12 audio effects with varying number of parameters. These parameters are afterwards scaled into a range of 0.0 to 1.0 for description purposes.

Table 4.1 highlights the distribution of effects over the dataset and Table 4.2 highlights the used parameters and their ranges.

BitCrush	54720
Chorus	54951
Clipping	54986
Compressors	54923
Delay	55005
Distortion	54875
High-pass filter	55041
Low-pass filter	55220
Ladder filter	55179
Limiter	55146
Phaser	55114
Reverb	55110
Total Samples	101694

Table 4.1: Data augmentation statistics.

As the parameters are also randomized, Figure 4.1 shows the distributions of three selected effects.

- **Limiter**: with the `threshold` (blue) and `release` (orange) parameters.
- **Chorus**: with the `rate` (blue), `depth` (orange), `centre delay` (green), `feedback` (red) and `mix` (purple) parameters
- **Reverb**: with the `room size` (blue), `damping` (orange), `wet level` (green), `dry level` (red), `width` (purple) and `freeze` (brown) parameters.

Each graph represents one effect and each different colour in histogram represents different parameter. It is important to note that some parameters can't be scaled over the whole range, as demonstrated in the Reverb graph, where the parameter represented by the brown colour has only a state of 0 or 1.

BitCrush	Depth 1.0 to 16.0					
Chorus	Rate (Hz) 0.1 to 10.0	Depth 0.0 to 1.0	Centre Delay (ms) 0.1 to 20	Feedback -1.0 to 1.0	Mix 0.0 to 1.0	
Clipping	Threshold -12.0 to 0.0					
Compressor	Threshold -12.0 to 12.0	Ratio 1.0 to 10.0	Attack 0.1 to 100.0	Release 10.0 to 1000.0		
Delay	Delay (s) 0.1 to 2.0	Feedback 0.0 to 1.0	Mix 0.0 to 1.0			
Distortion	Drive -12 to 12					
High-pass Filter	Cutoff (Fq) 20.0 to 2000.0					
Ladder Filter	Cutoff (Fq) 20.0 to 2000.0	Resonance 0.0 to 1.0	Drive 1.0 to 10.0			
Limiter	Threshold -20.0 to 0.0	Release 10.0 to 1000.0				
Low-pass Filter	Cutoff (Fq) 20.0 to 2000.0					
Phaser	Rate (Hz) 0.1 to 10.0	Depth 0.0 to 1.0	Mix 0.0 to 1.0	Feedback -1.0 to 1.0	Centre Frequency 20.0 to $\frac{SampleRate}{2}$	
Reverb	Room Size 0.1 to 0.9	Damping 0.1 to 0.9	Wet level 0.1 to 0.9	Dry level 0.1 to 0.9	Width 0.1 to 1.0	Freeze 0 or 1

Table 4.2: Used parameter limits.

4.3 Data types

The dataset is created and stored in three data types, due to the need for optimizing the model and ever-changing architecture needs. The main types included are:

- **Waveform files**: These files contain the basic augmented guitar samples in a wave format matching the format of the original dataset. The format of these files matches the original polyphonic samples with a 44.1kHz sample rate and a bit depth of 16 bits.

- **Mel-spectrograms**: These files are the direct next step in dataset creation and are directly tied to the waveform files. As the model requires input in Mel-spectrograms rather than waveform, these data can be used as input without the need to process every single waveform loaded. These Mel-spectrograms are created with a window length of 25 milliseconds, an overlap of 10 milliseconds, a 64 Mel band and an NFFT value of 512.

- **Tensors**: Due to the time complexity of the model with processing this amount of data every epoch, a way to optimise the model efficiency further presented itself in a way of storing the features output by the model itself. As will be explained further in the next chapter, the model is separated into two sections, where the first section performs the extraction of features from the input. This process is again quite time-consuming when repeated over a large dataset in every epoch, so by storing the features themselves as tensors, a significant amount of time is saved. The feature extraction model is frozen, so its output stays the same.

The dataset augmentation occurs in the `dataGenerator` script and the Mel-spectrogram and Tensor creation is handled in `dataPreprocessing` script. All these scripts are implemented in Python.

4.4 Data manager and loader

The final module of the data handling part of the system is the data manager and data loader. This part is essential for the training of neural networks as it provides the data itself and manages its labels and batching. This module also manages data splitting for the training, validating and testing of a model.

The module defines arrays and dictionaries of labels (the presence of an effect) and settings (effect parameters). With a defined root of a dataset, it loads the CSV file containing links between files and their configuration. Afterwards, a dictionary of labels and settings is filled with keys corresponding to filenames and values to the labels or settings of the key file. These dictionaries are processed into arrays containing the filename and either label or settings. This is separated due to training never requiring both the file label and the settings at the same time. These arrays serve as the main data point.

For regression, an additional method exists that discards every file from the array, that has the currently trained effect disabled. This avoids issues, that may be caused by training the model on data, that do not even contain that specific effect.

Finally, a getter is implemented that takes the filename and label/settings from the currently used array and returns these data with the addition of the loaded file based on the filename. Each training session uses a data manager, that splits the dataset into training, validation and testing segments. These data are randomized and are different for each subsequent training session. The data manager is implemented using the data split module, which takes the dataset as an argument and splits it into portions based on the given tasks. This module was implemented using the code available at GitHub[1].

In this phase, dictionaries are loaded that tie specific files to labels (effect presence) and settings (effect parameters). In the batch phase, where there are batches of 100 samples created, these labels and settings can be used to reference their configuration and compare them with the model predictions.

[1]https://palikar.github.io/posts/pytorch_datasplit

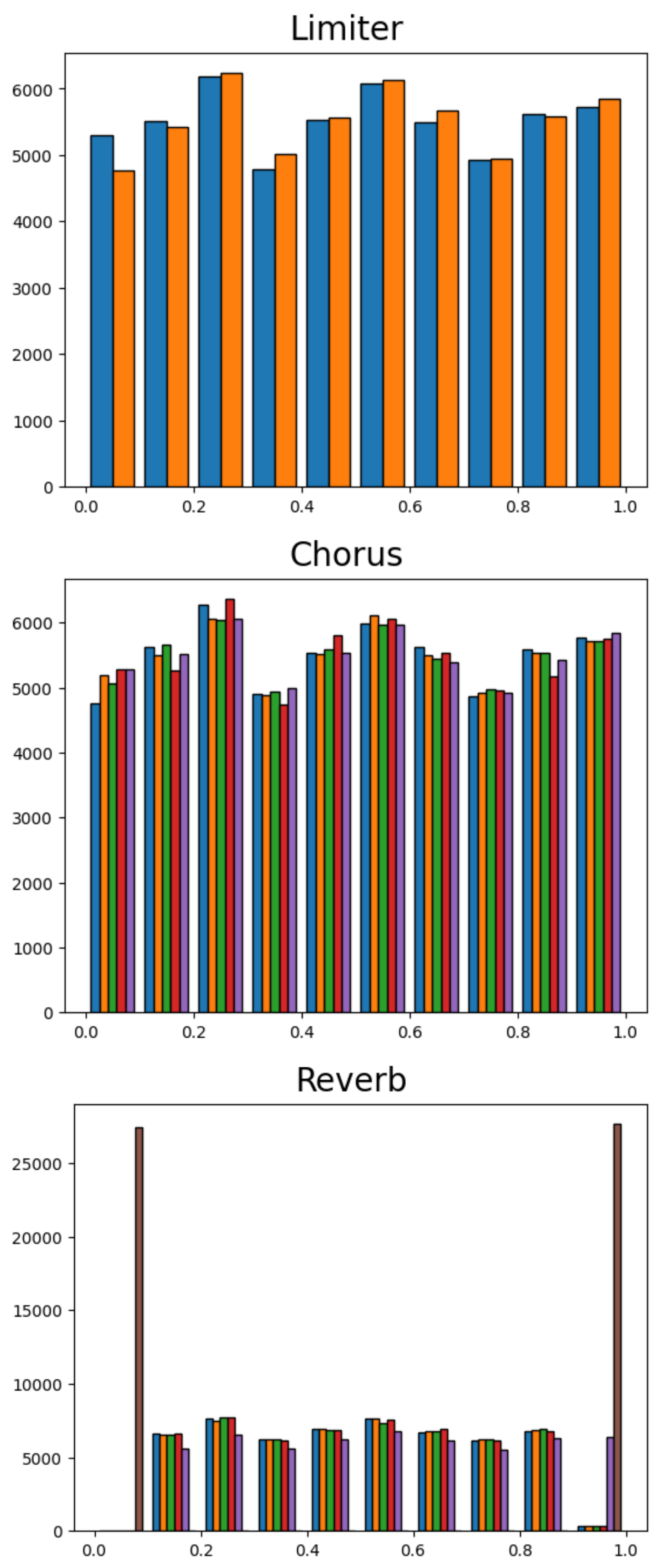


Figure 4.1: Parameter Distributions for Limiter, Chorus and Reverb.

Chapter 5

Neural Network design and training

This chapter describes the process of implementing the model and its training. Both are implemented using PyTorch. Previously implemented and tested methods will also be discussed as they played a major role in our research.

As seen in Figure 5.1, the final model consists of two major parts, the first part being the Backbone and the second part being split into Model heads, that form abstract Audio Effect Heads, where each abstract head targets one specific audio effect and its parameters.

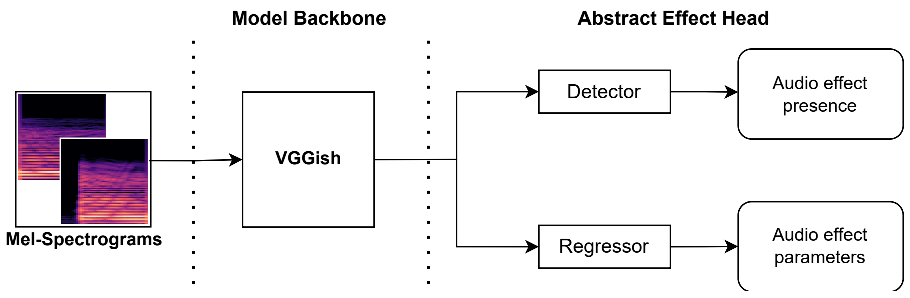


Figure 5.1: General Model Architecture.

5.1 Past Architecture Experiments

The proposed system went through many iterations of implementation. Some of the past implementations are noteworthy, to map the process towards the current implementation.

The first iteration of the system utilized Mel-Spectrograms as images, and as such the overlining architecture was an image-based model. This model consisted of two models, one for effect detection and one for parameter estimation. Both models followed the same architecture of two convolutional layers and two fully connected linear layers, where each layer was followed by a batch normalization, their difference was in the final activation function, where parameter estimation used Sigmoid to squash final results into the correct range. These models focused on data affected by a single effect and as such, the result was the class of detected effect. Results of this experiment yielded a detection accuracy of 61 % and parameter estimation accuracy with a mean absolute error of 0.34.

These results were not optimal and when retrained with multiple effects, the accuracy significantly dropped. The accuracy of the multiple effects was measured as a success if all the effects present were correctly detected. The multi-effect model has achieved an accuracy of 1.5 %, and as such brought up the need to restructure the process.

5.2 Proposed Model

The main issue in the first iteration was the detection of multiple effects in a sample. From this inadequacy, a new proposed system was conceptualized, that utilized an architecture of multiple heads which can be defined as modules, that work as separate models. This allows to separate effects into their detection and regression within the scope of one single model implementation, while allowing the separate training and loading of these parts.

This proposed architecture needs a Backbone that extracts features for these model heads to process. After testing this architecture with Image based models, that were unable to extract proper features for improvement of the model heads, Audio based models were the next step in this architecture evolution.

Audio-based models that focused on feature extraction from waveform data were usually designed for speech recognition tasks, and due to this were unable to properly extract features relevant for effect estimation, these waveform effects formed another issue, as their technical requirements were quite larger than the previous methods and work could not continue due to the technical limitations of the hardware used for creation of this work.

Finally, the last model tested that was ultimately chosen for the needs of a model backbone was VGGish, which will be further explained in the upcoming section.

Backbone VGGish

VGGish is a variant of the VGG model [12] trained with audio features. In particular, this modification is based on the VGG Configuration A. This configuration contains 8 convolutional layers and 3 fully connected layers. The convolutional layers use 64, 128, 256 and 512 filters in sequence each followed by max–pooling layer. The final 3 fully connected layers contain 4096, 4096 and 1000 units in sequence followed by softmax in the output layer. The input of this model is a 224×224×3 RGB image and the output is probability distribution across 1000 classes.

VGGish changes the input size to 96×64 for the Log Mel-spectrogram inputs framed into non-overlapping frames of 0.96 seconds in length. The last group of convolutional and maxpool layers is discarded changing the structure to four groups of convolution/maxpool layers and the fully connected layer at the end is changed from width of 1000 to 128, acting as a compact embedding layer [3].

Due to this model being implemented for TensorFlow, an experimental PyTorch port has been used, that replicates the procedures of the original variation and promises negligible differences.

The output of this model as mentioned in the previous section, uses raw embedding in a format of $batchsize \times numberOfFrames \times 128$.

Effect Heads

Figure 5.2 shows the architecture of an Abstract Effect Head. This term envelops the detection and regression heads under the specific effect, on which they are trained. Each audio effect has one detection head and one regression head.

Detection heads are composed of three linear layers followed by the ReLU function. Regression heads contain four linear layers. The first three linear layers are followed by a Batch Normalization and the ReLU function. Both model head architectures have the same output layer composed of one linear layer followed by the sigmoid function.

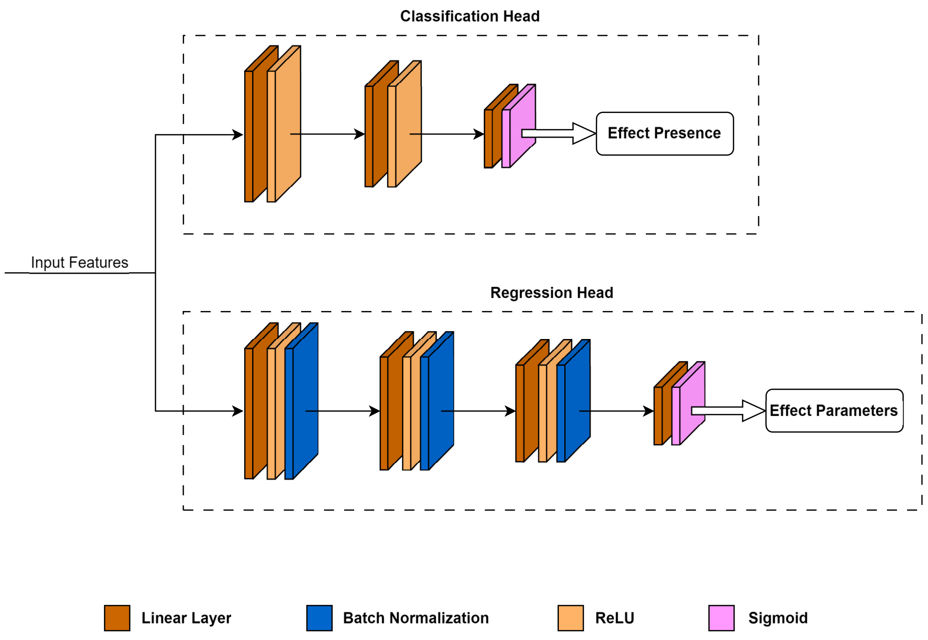


Figure 5.2: Architecture of an Abstract Effect Head.

Input of this model is the direct output of the Backbone model which has the input of $batchsize \times numberOfFrames \times 128$ features which are subsequently flattened into a single layer. The $numberOfFrames$ represents the number of Mel-spectrograms, that entered the Backbone model. This number is based on $sampleLengthSeconds/0.960$, creating the non-overlapping 960 ms frames. In general, the model works with two of these windows due to the length of samples in the dataset and the implemented limitation of input length further defined in 6.1.

5.3 Training

As previously mentioned, multi-head architecture allows, for a individual training of each effect detector and regressor. Due to this, the training is split into 24 sessions, 12 for effect detectors and 12 for effect regressors.

This allows for every individual part of the model, to work independently with the only constraint being the presence of the effect. This means that regressor results are ultimately discarded if the detection yields that the effect is not present.

In the initialization phase of training, all the necessary data are loaded and split using the data manager described further in detail in Chapter 4 and the device used for training is declared based upon the availability of cuda devices.

All training experiments used PyTorch [7] and were trained using the Adam optimizer. The learning rate was chosen based on a number of experiments with various values, and the chosen learning rate was 0.001.

In detection training, the loss calculation is performed using the Binary Cross-entropy. Regression utilises the Mean Square Error loss function.

Pseudoalgorithm 1 describes the general training process:

Algorithm 1 Training process.

```
Choose and initialise a training device
Define dataset source
Define currently trained model head
Create dataset splits for training, validation and testing
Define the Optimizer and Loss function
Define result arrays
for Number of epochs do
    for Batch in total batches of data do
        Convert batch data to the currently used device
        Pass batch data to model
        Calculate loss
        Measure current accuracy
        Log the results
    end for
    if Validation results exceed the best validation results then
        Save model head state
    end if
end for
Save the result log files
```

5.4 Results

For measuring the detection accuracy, a separate test set was defined. The obtained average effect detection accuracy was 75 %, while parameter estimation had an average mean error of 0.23 from target parameters. Parameters are scaled to the 0.00-1.00 range for consistency.

The detailed results can be seen in Table 5.1 and 5.2. These results are individual for each head, from samples containing multiple effects.

BitCrush 77.5 %	Chorus 83.8 %	Clipping 67.4 %	Compressor 65.8 %	Delay 74.9 %	Distortion 67.7 %
High-pass Filter 74 %	Low-pass Filter 88.5 %	Ladder Filter 73.5 %	Limiter 76.4 %	Phaser 78.7 %	Reverb 69.2 %

Table 5.1: Results of effect detection.

BitCrush 0.24	Chorus 0.24	Clipping 0.23	Compressor 0.24	Delay 0.24	Distortion 0.23
High-pass Filter 0.30	Low-pass Filter 0.24	Ladder Filter 0.24	Limiter 0.25	Phaser 0.23	Reverb 0.37

Table 5.2: Results of parameter estimation.

Chapter 6

Design and Implementation of the software

This chapter focuses on the design and implementation of the final system. This system is implemented as an interface between the user and the effect determination. The basic scheme of the system's inner workings, its inputs and outputs can be seen in 6.1. The user inputs either a mixed track containing a guitar with effects to be determined or an isolated guitar track. This distinction is passed as a parameter. This input is further limited to 2 second long samples.

One of the options for implementation of such system was implementing it as a VST plugin, but due to the time constraint of the work, a simpler solution in the form of an inference script was chosen.

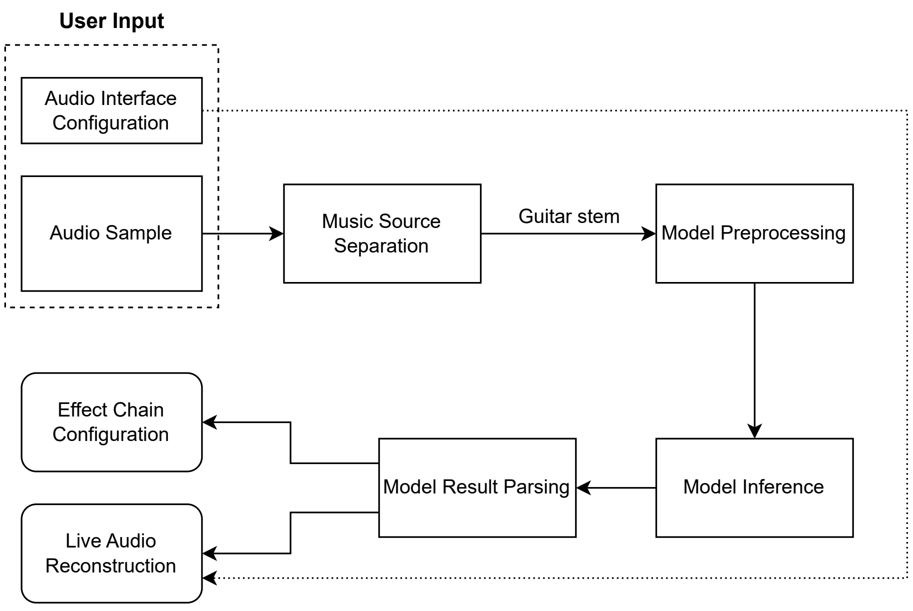


Figure 6.1: Inference System architecture.

6.1 Pre-processing

This section describes the choices behind input limits, the ways the pre-processing part of the system works, how the model inference is managed and the process of parsing the final results with their reconstruction.

The system accepts 2 second long waveform samples. This choice was made to avoid the input of whole songs, as such input would pose issues. Firstly the extraction of the guitar stem, from a minutes long input would vastly increase the processing time. This, combined with the fact that the system detects a single audio effect chain, that would ideally have to stay the same for the whole duration of the song, the choice of shortening the input forces the user to pick a certain section containing the effect chain up for determination. This makes the processing time shorter as well as avoids the issues of possible multiple effect chains that could cause undefined system behaviour.

A major part of the pre-processing process is the source separation. The user can specify whether the input track is already separated using an optional parameter, if the parameter is not specified, the track is assumed to be up for separation. The reason behind the need for manually specifying the nature of the input is due to the experimental nature of the used separation model `demucs_6s`. As the experimental part of this system is the extraction of the guitar, its handling of clean guitar tracks can be unpredictable. In some tests, this model tends to classify the isolated guitar as a bass track rather than a guitar, thus the distinction for such cases has to be made, in which case the separation is skipped.

The separation system Demucs offers an API, that simplifies working with the separated sample. This API uses a specific version of PyTorch, that conflicts with the version used for the experimental backbone model VGGish. Due to this, the API is not used and the separation is done in a way, where the input is saved into a special folder `/separated/htdemucs_6s/nameOfSample`, from which it is subsequently loaded. Although this process is more tedious, it offers the user the ability to check the quality of the separation.

In the final step of pre-processing, the guitar sample is processed using the internal VGGish processor, that converts the waveform input into a tensor consisting of a log Mel-spectrogram.

6.2 Inference and results

This section uses the methods described in Chapter 5. The input of the inference model is the tensor composed of log Mel-spectrograms created in the pre-processing part. The model inference part then loads 24 individual model head states, representing the detector and the regressor for each of the effects in the scope this work. After processing the input, inference returns a tensor containing the states of the 12 effects. This state is composed of the effect's presence and its individual parameters.

These states are parsed into comprehensive text that is displayed to the user and during the string construction, the effect information is loaded as an effect into a new effect chain. This process uses effects which have their value in tensor above a certain value. From system tests, the value was set to 0.7 even with the model training consisting of a threshold of 0.5. When it comes to samples outside of the dataset, the effect detection needs a slight tuning in the form of threshold adjustment to properly filter all the effects.

After displaying the text information to the user, the program takes the specified input and output device from the user input and loads the effect chain onto this configuration.

Afterwards, the system stays in the state of reconstruction while transforming the audio on the set device, until the user manually ends with a press of a key.

During the testing of the final system, a few shortcomings were detected, the upcoming list goes over the major issues.

- Problematic Effects

  The first notable issue was with the effects, the detection itself is closely tied to the quality of separation as will be discussed later. But another issue formed with two specific effects. First, the effect BitCrush did not perform as expected and rather introduced an unsatisfactory hiss to tracks. Another problem with this effect was its prominence in major number of tested samples. The second problematic effect was Reverb, that was also detected in almost all samples. The first logical reason might be, that the training of this certain head was not conducted properly, but after analyzing the effect further, it is entirely possible that due to the dataset being single tones or chords, the reverb effect would multiply these sounds. This would be the prominent feature of this effect and as such, analysed tracks that contain any repetitions would get falsely classified.

  Due to this reason, these two effects are disabled by default in detection and can be enabled by changing a global variable.

- Music Source Separation Issues

  As described in Chapter 3, the separation can have issues with some tracks, this became prominent when combining the inference with separation, as the separation very often found these edge cases and was unable to properly extract the guitar sample. Either the sample was completely displaced or affected by too much noise. This issue unfortunately does not have clear answer and as such, the system behavior for mixed tracks can be unexpected.

- Reconstruction Quality

  The reconstruction is done within the Python script on an audio stream, from a selected input device, which is not the most professional approach when compared to Digital Audio Workstations. This approach also suffers from high latency and reduced quality. As such, the quality of reconstruction can audibly suffer.

Chapter 7

Listening Experiments

This chapter will introduce experiments with various samples testing the capabilities of the implemented system. These tests focus on the subjective evaluation by the listeners. Listeners were selected to form a specific sample of potential users. These users include amateur musicians, casual listeners and people with music production experience. The number of surveyed users was 8.

The tests were split into two parts. The first part included the comparison of isolated guitar samples, affected by audio effects different from the ones, used in the dataset and reconstruction. In this part, the similarity of the reconstruction to the original and its overall sound quality was rated. The second part demonstrated the use of the full system including the Source Separation. In this test, users were able to listen to the original mixed track, the separated guitar for the reference and the reconstruction of that effect chain. For the reconstruction part, clean guitar samples were recorded to match the original samples. The rated part of this section was also the similarity of the reconstruction to the separated guitar and its overall sound quality, and also the quality of the separated guitar stem after the separation.

The goal of tests is to observe how well the system performs, its strengths and shortcomings.

The upcoming lists analyse the key points of the experiments and their results in the two mentioned parts. The similarity and quality were rated on ten ten-point scale, where 0 represented very poor results and 10 represented great results. The samples used in the tests can be found in a Google Drive[1].

Isolated Guitar Samples with known effects

- **Test 1**: Chorus & Drive & Boost

  This test includes known effect type chorus and two unknown effects: Drive and Boost. The purpose of including different types of effects is to observe the ability of the system to adapt to the unknown and observe the similarity of its subsequent reconstruction.

  `Similarity score`: 6.67

  `Sound quality of reconstructed sample`: 4.67

- **Test 2**: Octaver & Distortion

[1] https://drive.google.com/drive/folders/1_HxN2qGksFUsZpV0N4yWXwiQ0RLUudHm

Similarly to the first test, this one also includes an unimplemented type of effect. This effect was specially chosen as it manipulates the octave, which could possibly invalidate the result due to a tone deviation from the training dataset.

`Similarity score`: 4.67

`Sound quality of reconstructed sample`: 3.83

- **Test 3**: Reverb & Distortion

  This and the upcoming sample feature the known types of effects, to test the accuracy with known audio effects.

  `Similarity score`: 6

  `Sound quality of reconstructed sample`: 5.8

- **Test 4**: Phaser & Compressor

  `Similarity score`: 8.5

  `Sound quality of reconstructed sample`: 8.8

The system performed better with known types as is expected. The reconstruction for unknown types was not the worst and the similarity would be even better if the model did not include additional effects that invalided the result, as the determined effects reassembled the actual configuration aside from the unknowns.

Blind tests utilising Music Source Separation

- **Sample 1**: Jimi Hendrix - Purple Haze

  `Quality of separated stem`: 8.83

  `Similarity score`: 4.17

  `Sound quality of reconstructed sample`: 4.5

- **Sample 2**: Gojira - Silvera

  `Quality of separated stem`: 6

  `Similarity score`: 5.67

  `Sound quality of reconstructed sample`: 4

- **Sample 1**: Ghost - Square Hammer

  `Quality of separated stem`: 7.6

  `Similarity score`: 4.33

  `Sound quality of reconstructed sample`: 5.33

- **Sample 1**: Pink Floyd - Comfortably Numb

  `Quality of separated stem`: 8.5

  `Similarity score`: 6.17

  `Sound quality of reconstructed sample`: 6.33

- **Sample 1**: Chuck Berry - Johnny B. Goode

  `Quality of separated stem`: 9.5

  `Similarity score`: 6.67

  `Sound quality of reconstructed sample`: 6.5

The overall quality was significantly lower than with an isolated guitar. The similarity also significantly dropped which could be the consequence of the separation, but the model did not perform well even with samples that featured a good separation.

7.1 Conclusion from experiments

The subjective experiments conducted in this chapter demonstrate the ability of the system to reconstruct audio effects from reference samples, either isolated or mixed within a track. These results do not achieve perfect levels of similarity and in some cases, even produce samples of bad sound quality due to incompatible effect combinations. After analysing additional comments from the experiments, users commend the system for its ability to at the very least approximate the features of the reference samples. While not achieving perfect similarity, some of the notable features of the reference tracks can be heard, which demonstrates the system's potential. Overall, these results highlight the need for future enhancements to refine the system's accuracy and expand its capabilities.

Chapter 8

Conclusion

In this work, the reader is introduced to the subject of Neural Networks and Music Source Separation and most importantly the characteristics of Digital Audio Effects. Afterwards, methods for source separation are analysed and a system for audio effect detection and parameter estimation is proposed. Furthermore, a system is introduced, that connects these topics and offers the user audio effect description and reconstruction based on the provided input.

An augmented dataset is introduced alongside with methods used for said augmentation. This dataset can not be provided due to copyright constraints of the reference data, but an augmentation script is included alongside the system to allow for reproducibility of our work. From the results, the model is capable of learning the features of the audio effects included in the system for their determination. The model even shows the capability of determining the effects in tracks affected by audio effects of the same distinction, but differing implementation. The parameter estimation is not as capable as effect detection, as the used model and previously tested variations had issues with the learning process and only a handful of parameters of certain effects correctly learned to associate the inputs with target values.

The final system is conceived as a console application with rather simple implementation due to time constraints. An optimal implementation of such system would be a VST plugin, that can be used in Digital Audio Workstations, which would have more appeal to the average user as well as an optimal place for the effect chain reconstruction.

Future work

This system has a lot of potential to be improved. As the topic of Music Source Separation will improve in the future, the currently used separator could be replaced, which would certainly result in improvement of separated instrument effect estimation. Furthermore, improvements to the model would certainly improve the results.

These improvements could stem from more complex datasets, that would introduce more effects or variations of the same effect. Another improvement could focus on determining the family of the effect rather than the specific type, and afterwards constructing these effects based on this information. Another big improvement could be in the form of additional model heads, that would focus on the positional index of the effect in its effect chain. Our system so far does not have this capability and as such, the reconstruction can possibly contain all the correct effects, but their order in the chain can completely devaluate the result. As such, the positional information would greatly improve the reconstruction and would also give additional effect information to the system user.

Another improvement may come in a form of a better interface between the system and the user. The best approach would be to implement this system as a VST plugin for the immediate availability of the reconstructed effect chain right in the Digital Audio Workstation. This would require either reimplementing the Pedalboard effects in JUCE, which is used for the creation of VST plugins, or the complete overhaul of the available effects with the plugin implementation in mind.

Exploitation

This type of system could certainly be commercially exploited. The topic of digital audio effects is currently a profitable topic as companies focused on making software plugins offering these effects monetise these types of systems quite successfully. These plugins often try to implement effects simulating analogue brands and offer a quite broad spectrum of options and presets that emulate existing bands and artists. A system that could estimate effects used in popular tracks and offer a user these effects to the detail of parameter settings would certainly be a profitable venture, especially if combined with existing commercially successful effect implementations.

Bibliography

[1] Chollet, F. *Deep Learning with Python.* 2nd ed. Manning, 2021. ISBN 1617296864.

[2] Défossez, A., Usunier, N., Bottou, L. and Bach, F. Music Source Separation in the Waveform Domain. *ArXiv preprint arXiv:1911.13254.* 2019.

[3] Hershey, S., Chaudhuri, S., Ellis, D. P. W., Gemmeke, J. F., Jansen, A. et al. CNN Architectures for Large-Scale Audio Classification. In: *International Conference on Acoustics, Speech and Signal Processing (ICASSP).* 2017. Available at: https://arxiv.org/abs/1609.09430.

[4] Kinsley, H. and Kukieła, D. *Neural Networks from Scratch in Python.* Harrison Kinsley, 2020.

[5] Mitsufuji, Y., Fabbro, G., Uhlich, S., Stöter, F.-R., Défossez, A. et al. Music Demixing Challenge 2021. *Frontiers in Signal Processing.* Frontiers Media SA. january 2022, vol. 1. DOI: 10.3389/frsip.2021.808395. ISSN 2673-8198. Available at: http://dx.doi.org/10.3389/frsip.2021.808395.

[6] Nielsen, M. A. *Neural Networks and Deep Learning.* Determination Press, 2015. Available at: http://neuralnetworksanddeeplearning.com/.

[7] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: *Advances in Neural Information Processing Systems 32.* Curran Associates, Inc., 2019, p. 8024–8035. Available at: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[8] Pirkle, W. *Designing Audio Effect Plug-ins in C++ with Digital Audio Signal Processing Theory.* Focal Press, 2013. ISBN 9780240825151. Available at: https://books.google.cz/books?id=v0ulUYdhgXYC.

[9] Rafii, Z., Liutkus, A., Stöter, F.-R., Mimilakis, S. I. and Bittner, R. *MUSDB18-HQ - an uncompressed version of MUSDB18.* August 2019. DOI: 10.5281/zenodo.3338373. Available at: https://doi.org/10.5281/zenodo.3338373.

[10] Reiss, J. D. and McPherson, A. *Audio effects: theory, implementation and application / by Joshua D. Reiss and Andrew McPherson.* 1st editionth ed. Boca Raton, FL: CRC Press, an imprint of Taylor and Francis, 2014. ISBN 0-429-09723-9.

[11] Rouard, S., Massa, F. and Défossez, A. Hybrid Transformers for Music Source Separation. In: *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* 2023, p. 1–5. DOI: 10.1109/ICASSP49357.2023.10096956.

[12] Simonyan, K. and Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv 1409.1556.* September 2014.

[13] Smaragdis, P., Févotte, C., Mysore, G., Mohammadiha, N. and Hoffman, M. Static and Dynamic Source Separation Using Nonnegative Factorizations [A unifed view]. *Signal Processing Magazine, IEEE.* May 2014, vol. 31, p. 66–75. DOI: 10.1109/MSP.2013.2297715.

[14] Smith, J. *Delay Lines* [http://ccrma.stanford.edu/~jos/pasp/Delay_Lines.html]. Online Book, 2010. Accessed: 12.3.2024.

[15] Sobot, P. *Pedalboard.* Zenodo, July 2021. DOI: 10.5281/zenodo.7817838. Available at: https://doi.org/10.5281/zenodo.7817838.

[16] Stein, M. *IDMT-SMT-Audio-Effects Dataset.* Zenodo. DOI: 10.5281/zenodo.7544032. Available at: https://doi.org/10.5281/zenodo.7544032.

[17] Zölzer, U., Amatriain, X., Arfib, D., Bonada, J., De Poli, G. et al. *DAFX - Digital Audio Effects.* John Wiley & Sons, 2002. ISBN 9780471490784. Available at: https://books.google.cz/books?id=h9OHIV0uwVsC.

Appendix A

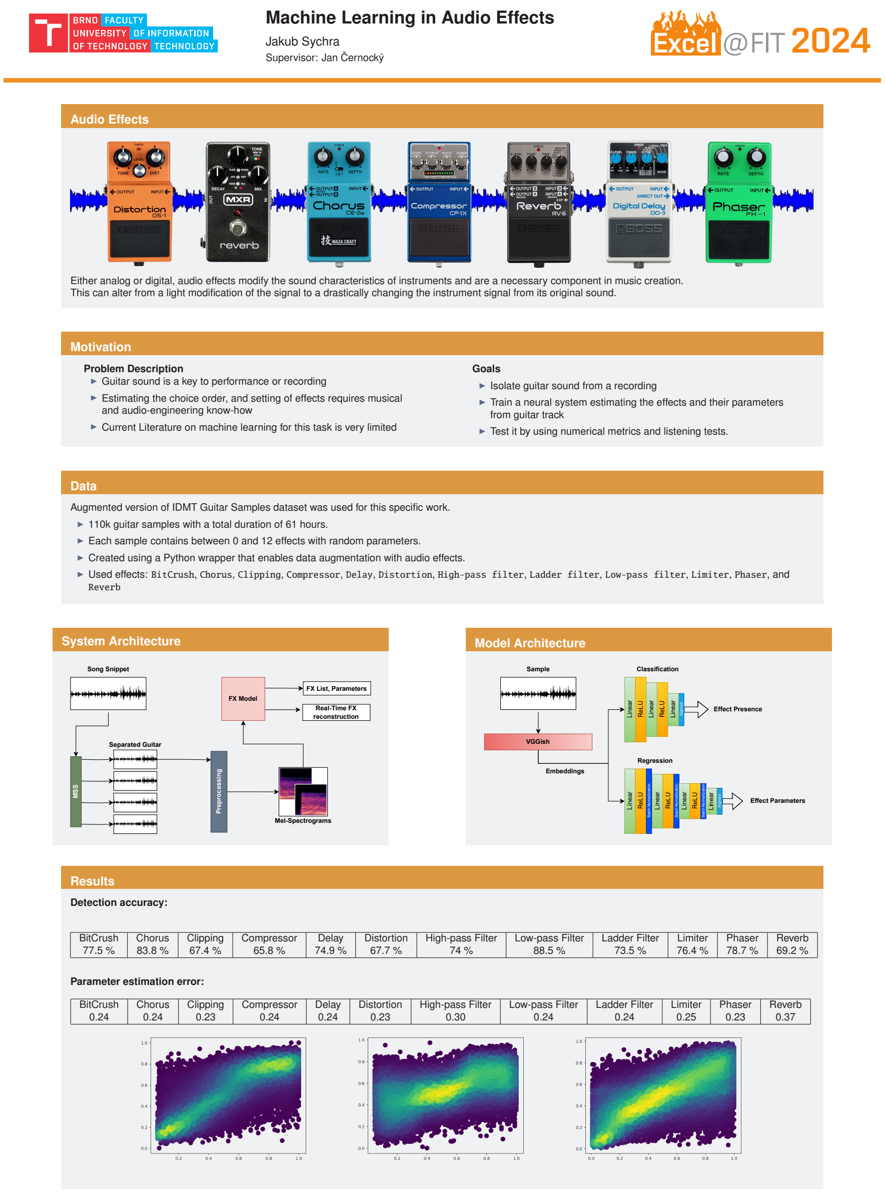
Contents of the included storage media

/

classify.py	The inference and reconstruction script
dataAnalysis.ipynb	Graph generation from Data section.
data_generator.ipynb	Dataset augmentation script
datamanager.py	Data manager for model training
datasplit.py	Module for splitting training data.
datatypes_processor.ipynb	Dataset data type processing
LaTeX/	LaTeX source files
model.py	Proposed Model Architecture
poster.pdf	Poster in format .pdf
samples/	Samples from Listening Tests
saved/	Current Model states
prod/	
README.md	System Instructions
requirements.txt	Required Python libraries
thesis.pdf	Text of the Bachelor's thesis
train.py	Module for training functions
typhon_individual.py	Trainer for detection
typhon_individual_r.py	Trainer for regression

Appendix B

Poster



BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Machine Learning in Audio Effects

Jakub Sychra

Supervisor: Jan Černocký

Excel@FIT 2024

Audio Effects

Either analog or digital, audio effects modify the sound characteristics of instruments and are a necessary component in music creation. This can alter from a light modification of the signal to a drastically changing the instrument signal from its original sound.

Motivation

Problem Description

- Guitar sound is a key to performance or recording
- Estimating the choice order, and setting of effects requires musical and audio-engineering know-how
- Current Literature on machine learning for this task is very limited

Goals

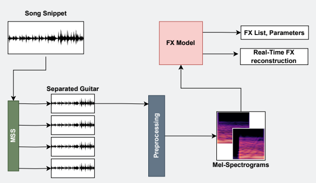
- Isolate guitar sound from a recording
- Train a neural system estimating the effects and their parameters from guitar track
- Test it by using numerical metrics and listening tests.

Data

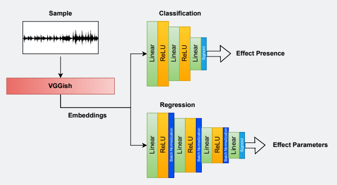
Augmented version of IDMT Guitar Samples dataset was used for this specific work.

- 110k guitar samples with a total duration of 61 hours.
- Each sample contains between 0 and 12 effects with random parameters.
- Created using a Python wrapper that enables data augmentation with audio effects.
- Used effects: `BitCrush`, `Chorus`, `Clipping`, `Compressor`, `Delay`, `Distortion`, `High-pass filter`, `Ladder filter`, `Low-pass filter`, `Limiter`, `Phaser`, and `Reverb`

System Architecture



Model Architecture



Results

Detection accuracy:

BitCrush	Chorus	Clipping	Compressor	Delay	Distortion	High-pass Filter	Low-pass Filter	Ladder Filter	Limiter	Phaser	Reverb
77.5 %	83.8 %	67.4 %	65.8 %	74.9 %	67.7 %	74 %	88.5 %	73.5 %	76.4 %	78.7 %	69.2 %

Parameter estimation error:

BitCrush	Chorus	Clipping	Compressor	Delay	Distortion	High-pass Filter	Low-pass Filter	Ladder Filter	Limiter	Phaser	Reverb
0.24	0.24	0.23	0.24	0.24	0.23	0.30	0.24	0.24	0.25	0.23	0.37