Master thesis on Sound and Music Computing

Universitat Pompeu Fabra

Learning to mix with neural audio effects in the waveform domain

Christian J. Steinmetz

Supervisor: Joan Serrà

Co-Supervisor: Frederic Font

September 2020

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Joan Serrà, for providing continuous encouragement and a belief in work. Throughout the process he has shared his extensive knowledge and provided invaluable guidance, all of which has had a significant impact in my development as a researcher. I am also grateful to my fellow collaborators, Jordi Pons and Santiago Pascual, whose expertise and insight greatly facilitated my investigations through our many discussions. This project was carried out over a six month period during my internship in the Barcelona office of Dolby Laboratories. This fruitful collaboration provided the resources, space, and collaborations necessary to successfully carry out my work. This project would not have been possible without the resources and support Dolby Laboratories provided.

I am also greatly indebted to the Audio Engineering Society Education Foundation, who provided financial support, first in my undergraduate education, and now in collaboration with Genelec, through the Ilpo Martikainen Audio Visionary Scholarship. This generous funding has played a major role in supporting my studies while completing my master at Universitat Pompeu Fabra. I want to also thank my fellow classmates in the Sound and Music Computing master at Universitat Pompeu Fabra, for their collaboration and fruitful discussions in this shared endeavor throughout this process. A special thanks goes to my parents who have continually supported all of my pursuits without question, and also to my family and friends who have provided moral support throughout this process.

Abstract

The process of transforming a set of recordings into a musical mixture encompasses a number of artistic and technical considerations. Due to this inherent complexity, a great deal of training and expertise on the part of the audio engineer is required. This complexity has also posed a challenge in modeling this task with an assistive or automated system. While many approaches have been investigated, they fail to generalize to the diversity and scale of real-world projects, with the inability to adapt to a varying number of sources, capture stylistic elements across genre, or apply the kinds of sophisticated processing used by mix engineers, such as compression.

Recent successes in deep learning motivate the application of these methods to advance intelligent music production systems, although due to the complexity of this task, as well as a lack of data, there are a number of challenges in directly applying these methods. In this thesis, we address these shortcomings with the design of a domain inspired model architecture. This architecture aims to facilitate learning to carry out the mixing process by leveraging strong inductive biases through self-supervised pre-training, weight-sharing, as well as a specialized stereo loss function.

We first investigate waveform based neural networks for modeling audio effects, and advance the state-of-the-art by demonstrating the ability to model a series connection of audio effects jointly over a dense sampling of their parameters. In this process, we also demonstrate that our model generalizes to the case of modeling an analog dynamic range compressor, surpassing the current state-of-the-art approach. We employ our pre-trained model within our framework for learning to mix from unstructured multitrack mix data. We show that our domain-inspired architecture and loss function enable the system to operate on real-world mixing projects, placing no restrictions on the identity or number of input sources. Additionally, our method enables users to adjust the predicted mix configuration, a critical feature that enables user interaction not provided by basic end-to-end approaches. A perceptual evaluation demonstrates that our model, trained directly on waveforms, can produce mixes that exceed the quality of baseline approaches. While effectively controlling all the complex processors in the console remains challenging, we ultimately overcome many of the challenges faced by canonical end-to-end deep learning approaches.

While the results presented in this work are preliminary, they indicate the potential for the proposed architecture to act as a powerful deep learning based mixing system that learns directly from multitrack mix data. Further investigations with this proposed architecture through the use of larger datasets, in addition to transforming the controller to act a generative model, prove to be promising directions in advancing intelligent music production, surpassing current knowledge based expert systems and classical machine learning approaches for this task.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Challenges 3

1.3 Research questions 5

1.4 Contributions 6

2 Background **8**

2.1 Music production 8

2.1.1 Recording 8

2.1.2 Mixing 9

2.1.3 Mastering 10

2.2 Audio effect modeling 11

2.2.1 White-box models 11

2.2.2 Black-box models 12

2.3 Intelligent music production 13

2.3.1 Early work 14

2.3.2 Knowledge-based systems 15

2.3.3 Machine learning 17

2.3.4 Deep learning 17

3 Multitrack mixing system design **21**

3.1 Problem formulation 21

3.1.1 Goal definition 22

3.2 Learned differentiable mixing console 23

3.2.1 Transformation network 25

3.2.2 Controller network 26

4 Audio effect modeling **28**

4.1 Problem formulation 28

4.2 Audio effect implementation 29

4.2.1 Gain and panning 29

4.2.2 Equalization 30

4.2.3 Dynamic range compression 32

4.2.4 Reverberation 34

4.2.5 Channel signal chain 35

4.2.6 Analog dynamic range compression 35

4.3 Architectures 36

4.3.1 Temporal convolutional network 37

4.3.2 Wave-U-Net 40

4.3.3 Conditioning 43

4.3.4 Receptive field 44

4.4 Dataset 45

4.5 Training 46

4.6 Evaluation 46

4.6.1 Metrics 47

4.6.2 Channel signal chain 50

4.6.3 Analog dynamic range compressor 52

4.7 Summary 54

5 Mixing system implementation **56**

5.1 Baselines 56

5.1.1 Deep learning baseline 56

5.1.2 Naive baseline 57

5.2 Differentiable mixing console 57

5.2.1 Architecture 57

5.3 Datasets 62

5.3.1 ENST-drums 62

5.3.2 MedleyDB 63
5.4 Stereo loss function 63
5.5 Training 64
5.6 Evaluation 65
5.6.1 Objective metrics 65
5.6.2 Subjective evaluation 69

6 Discussion **77**
6.1 Conclusions 77
6.2 Future work 78

List of Figures **79**

List of Tables **81**

Bibliography **83**

Chapter 1

Introduction

1.1 Motivation

The journey from the original seed of a musical idea to the final recorded production involves a number of different steps that are often not evident from the perspective of the music listener. This process generally involves the collaboration of a number of different individuals who perform unique roles, each with their own skills and specialization, such as songwriters, musicians, producers, as well as the recording, mixing, and mastering engineers. One critical step in this process is the task of transforming the individual recorded elements into a final mixture, which is undertaken by the mixing engineer, and is often an integral part of the creative process in modern recordings. Without this step, it would be impossible to generate the musical experience listeners are familiar with today.

This task of transforming a collection of audio signals into a cohesive mixture requires a deep understanding of disparate technical and creative processes. For this reason, audio engineers are required to study and practice mixing techniques for some years before gaining the experience required to address the complexities inherent in the interaction among signals. To effectively carry out this task, an audio engineer's specialized training involves developing the ability to recognize how to utilize an array of signal processing tools to achieve a set of desired technical and creative goals. Due to this reality, there are a number of driving factors in the development of intelligent music production (IMP) tools, tools that aim to offer assistance in parts of this complex process (De Man et al., 2019).

The past decade has seen an increase in the affordability and overall accessability of music production tools. This has been characterized by the growing availability of digital audio workstation (DAW) and plugin offerings, along with affordable studio quality microphones, headphones, and loudspeakers. Equipment once present only

in world-class studios can now be found in the bedrooms of musicians and amateur audio engineers, introducing music production to a new, diverse demographic. When coupled with the expansion of accessible online platforms for music distribution, such as Spotify, Apple Music, Pandora, YouTube, SoundCloud and others, these circumstances have lead to an explosion of interest in music production among amateur audio engineers and musicians (Walzer, 2017).

As a result, there are a number of factors motivating the development of tools that simplify the music production process. The first factor involves addressing the significant prerequisite experience in performing these music production tasks, which amateur audio engineers and musicians often lack. Such tools aim to extend the ability of inexperienced individuals by suggesting possible configurations, or providing feedback that incorporates expert knowledge. These tools provide the potential to greatly improve the quality of productions conceived by these creators, and ultimately enhance the experience for music listeners. As the current trend of independent music production continues to increase, demand for these kinds of tools will only continue to increase, and software offerings from iZotope[1], LANDR[2], and Accusonus[3] are all evidence of this trend.

The second motivating factor involves the automation of complex and time consuming tasks in the music production process with the aim of reducing the time investment required on the part of professional engineers. Clearly, intelligent music production tools can produce benefit not only for amateur audio engineers, but also for those with extensive experience, provided these tools are able to adequately carry out repetitive tasks, or offer new methods for more efficiently traversing the space of possible mix configurations. Such a system may have the effect of helping artists work more rapidly when they desire, or enable them to explore more creative ideas in a short period of time.

Finally, there is also the potential for intelligent music production tools to uncover elements that bring greater understanding to the established practices and techniques carried out by professional audio engineers. Moreover there is also the potential to discover new practices and techniques that also produce quality mixes in novel or interesting ways. Ultimately, these tools have the potential to enable musicians and amateur audio engineers to produce higher quality work, function as an educational tool, as well as expedite the production process for experienced audio engineers allowing for greater focus on artistic considerations.

[1] https://www.izotope.com/en/products/neutron.html
[2] https://www.landr.com/en/online-audio-mastering/
[3] https://accusonus.com/

1.2 Challenges

Over the past decade, researchers in the field of IMP have worked to develop new technologies that aim to address the interest in tools that provide assistance to audio engineers (De Man et al., 2017). A variety of approaches have been investigated, with most falling into one of two categories (Moffat and Sandler, 2019b): knowledge-based expert systems, or data-driven machine learning models, which will be addressed in greater detail in the following chapter. While previous approaches in both of these categories have demonstrated the ability to model the behavior of audio engineers in a set of limited mixing tasks, these systems still fail to adequately adapt to the diversity and scale present in many real-world projects (De Man, 2017).

Recently, deep learning has demonstrated impressive results on many audio tasks that were previously thought to be extremely challenging. Some examples include speech synthesis (van den Oord et al., 2016a, Wang et al., 2017, Engel et al., 2019), instrument synthesis (Engel et al., 2017, Défossez et al., 2018), source separation (Hershey et al., 2016, Stoller et al., 2018, Stöter et al., 2019), and speech enhancement (Pascual et al., 2017, Rethage et al., 2018). For this reason, there is interest in the application of these models in IMP, and more specifically within the context of methods for automated multitrack mixing (Martínez Ramírez and Reiss, 2017).

While these recent successes appear promising in our goal of advancing IMP systems, there are a number of challenges that impede our ability to design deep models for the multitrack mixing task. Foremost, is the limited multitrack mix data available. In a supervised training regime we are interested in parallel multitrack data, collections containing the original unprocessed multitrack recordings, along with mixes of those tracks made by trained audio engineers. Due to the realities of copyright and the challenge in cataloging this kind of data, there has been a significant challenge in developing large scale, open datasets of this kind, but efforts have been made to make data of this kind available. One example is MedleyDB (Bittner et al., 2014), which, after the latest update (Bittner et al., 2016), features 196 tracks, complete with unprocessed multitrack recordings and a corresponding mix. In addition, the Open Multitrack Testbed attempts to catalog multitrack content for educational and research contexts (De Man et al., 2014), but contains many of the same source as MedleyDB, and has significantly less organization with regards to metadata. Nevertheless, these sources provide many orders of magnitude less samples compared to common datasets used for deep learning in related domains. For example, the VoxCeleb dataset with over one million speech utterances (Nagrani et al., 2019), the Freesound dataset with quarter of a million sound recordings (Fonseca et al., 2017), or the Million Song Dataset (Bertin-Mahieux et al., 2011) with acoustic features extracted from over one million songs. Training end-to-end models operating in

the waveform domain may require upwards of one million samples to perform as effectively on classification tasks (Pons et al., 2017), and while spectrogram based approaches have been shown to perform more competitively with less data, this approach is more challenging for tasks that involve synthesis (Engel et al., 2019).

In addition to the lack of parallel data, there is also the challenge in building a model that is able to adapt to the diversity present in real-world multitrack projects. While many music productions are composed of similar sources, for example drums, guitars, piano, vocals, etc., there is no consistent format or structure for the inputs that compose a mix (Owsinski, 2006). Such characteristics are highly dependant on the genre and the composition itself. This presents a challenge since we cannot assume a priori the number of sources or their identity, and therefore the model must be able to handle such variable sized inputs, as well as adapt its behavior accordingly. Current state of the art models (e.g. convolutional neural networks for music source separation (Stöter et al., 2019)) exploit both the explicit number and ordering of inputs during the training of the network. In the case of multitrack mixing, since we have no unified format or taxonomy, we require the model to operate on a variable number of input sources, and for any arbitrary ordering of the same input tracks to produce the same mix. To achieve this kind of behavior, a convolutional neural network would require orders of magnitude more model capacity, as the learned transformations for each input channel would need to effectively process every type of input signal. This suggests that a more sophisticated architecture is required, which we will address in detail in Chapter 3.

As outlined by Moffat and Sandler (2019b), there are a number of different roles that an intelligent music production system can play in the process of creating a multitrack mix. These range from assistive algorithms that provide feedback, but do not directly generate a mix, all the way to fully automated systems that take inputs and produce a complete mix. There is an inherent challenge in applying end-to-end deep learning to the mixing task in this latter case, as the user will receive a complete mixture at the output of the system, but no method for further adjusting this result. A model that takes a set of inputs and maps them directly to a mix, without the ability for human interaction, provides limited utility since the mixing process is largely a creative task with no objectively "correct" output. Therefore, there is an interest in designing a mixing system such that the end-user has the ability to further tweak the results based on their internal goals. This interface could manifest itself either with the same controls audio engineers are accustomed to, or in the case of a tool designed for amateur users, these controls could be abstracted to higher level controls that provide an easy method for traversing a number of potential options.

The final and often overlooked challenge comes from the aforementioned ill-defined nature of the mixing task. While conventions are evident among professionally

produced mixes, it is clear that different, yet equally acceptable mixes, may be located in disparate areas of the so-called mix space or parameter space of the mixing console (Wilson and Fazenda, 2017). Even when ignoring the task of attempting to model this complex one-to-many mapping, there still remains an inherent challenge when training a model in a supervised fashion to regress a ground truth mix.

As a concrete example of how this issue may manifest itself during training, consider the case of panning a guitar source in a mix. During training, the model may have seen many examples that contain a guitar, and often this source may be panned either to the left or right side. For any song containing a guitar given as input, since the model has no way to predict whether this guitar will appear on the left or right in the ground truth mix, placing the source always in the center will minimize the error in the loss function (assuming a loss based on L1 or L2 distance in the time or frequency domains), and clearly this behavior is undesirable. For this reason, loss functions that address this issue and take into account domain knowledge of mixing may need to be designed, a claim we will address further in Chapter 5.

Due to these realities, it is unlikely that a canonical deep neural network architecture, where the input is a collection of multitrack recordings and the output is a mix of those tracks, will be able to be trained directly in a supervised fashion using the limited multitrack data available. This contention between a lack of available data and the need for data-demanding end-to-end models is one of the fundamental challenges in the application of deep learning for intelligent music production. In addition, these realizations suggest that more specialized architectures and loss functions that incorporate domain knowledge from music production may be required in order to train models that are able to learn mixing conventions from waveforms.

This thesis aims to provide a path towards rectifying this situation with an investigation of an approach that incorporates domain knowledge in the design of a new architecture and loss functions, as well as through self-supervised pre-training to produce component models with stronger inductive biases to more efficiently leverage the limited available training data.

1.3 Research questions

We aim to design a system that takes as input a number of sources that compose a musical work, and produce a stereo mixture of these sources that achieves a comparable level of quality to mixes produced by trained audio engineers. Our work addresses two main tasks, first that of modeling audio effects in an end-to-end fashion, and then the use of these models in the construction of a learned, end-to-end multitrack mixing system.

To begin, we first investigate the following questions on audio effect modeling.

(*i*) Can linear and nonlinear audio processors (equalizer, compressor, and reverb) be effectively modeled with a dense sampling over their parameter space?

(*ii*) Can a series connection of these processors, as well as their ordering within the chain, be effectively modeled jointly by a neural network?

(*iii*) Can such a model generalize to the case of modeling a nonlinear analog device with limited training data?

Afterwards, we address the following questions in order to construct a multitrack mixing system.

(*i*) Can a composition of these models be utilized to construct a differentiable mixing console with sufficient transformational expressivity?

(*ii*) Can a controller network be designed that generates parameters for this mixing console and adapt to handle a variable number of inputs?

(*iii*) Can an appropriate loss function be designed that encourages quality mixes by incorporating domain knowledge of multitrack mixing?

1.4 Contributions

Our contributions can be divided into two major areas. The first involves our work on extending current approaches in neural audio effect modeling. In our investigations we positively answer the research questions we propose and demonstrate the following:

(*i*) A dense sampling over the parameter space of a number of linear and nonlinear audio effects can be effectively modeled.

(*ii*) A series connection of audio effects, along with their parameters can be modeled with a single network.

(*iii*) This model can also capture the behavior that arises from varying the ordering of these processors within the signal chain.

(*iv*) Our proposed audio effect modeling architecture achieves state of the art performance on the black-box modeling of an analog compressor.

In the second part of our work, we propose a domain inspired architecture for learning to generate mutlitrack mixes using only multitrack recordings and mixes made by audio engineers. Our approach addresses the aforementioned challenges and we show the following about our proposed architecture:

(*i*) Can be trained using a limited number of multitrack training examples.

(*ii*) Makes no assumptions about the format or identity of input sources.

(*iii*) Is permutation invariant with respect to the order of inputs.

(*iv*) Places no upper or lower limit on the number of input sources in a mix.

(*v*) Enables interaction by users to adjust the output, providing interpretability.

(*vi*) Produces mixes that exceed baseline approaches in a perceptual evaluation.

Chapter 2

Background

2.1 Music production

The process of transforming a musical composition into a produced work, ready for distribution, incorporates a large number of diverse tasks. These tasks are generally carried out by a team of individuals working in tandem, and includes musicians, producers, record labels, and recording, mixing, and mastering engineers. While this process may involve composition and arrangement of musical pieces, where the act of composing the song takes place concurrently, here we consider only the general steps taken after all compositional decisions have been made.

2.1.1 Recording

Once a composition has been written, musicians often work in a recording studio, along with a producer and audio engineer, to carry out the process of recording each of the elements, either individually, jointly, or more often a combination of both (Huber and Runstein, 2010). In the case of acoustic instruments, the recording of each performance is often achieved with the use of a range of different types of microphones placed in an acoustically treated space, designed to reduce unwanted reflections. While some studios may feature a larger recording space, which imparts natural acoustic qualities to recordings, such facilities are often expensive and inflexible. For this reason, the predominant approach is to record sources in an acoustically "dry" or "dead" space, where as little of the room's qualities impact the final recording. Then, in the next stages artificial reverberation effects may be used to make the recordings sound more natural (Case, 2011).

To capture the performance, a series of analog-to-digital converters (ADCs) are used to convert the electrical signals of the microphones to a digital representation.

This is achieved by sampling the voltage of each microphone with a fixed number of possible voltage levels defined by the bit-depth (e.g. 16 or 24 bits) at a fixed rate (e.g. 48 kHz, 96 kHz, or 192 kHz)[1]. The resultant digital representations are then often stored on the hard disk of a computer running digital audio workstation (DAW) software, which provides a means to later manipulate these recordings to create a final product.

While many genres incorporate acoustic instruments, some genres like electronic music may include only electronic instruments. In these cases the workflow often remains quite similar to the traditional studio recording process. Individual elements are created either by recording the output of various synthesizers and samplers, or by writing MIDI data that will be sent to a hardware or software synthesizer during playback. These tracks can then be treated the same as those recorded with acoustic instruments when it comes time to create the final mix.

The entire process of tracking all the elements can take anywhere from days to years, depending on the complexity of the composition and the vision of the artist. Often the recording process is closely intertwined with the composition process. Artists may experiment with new ideas, building the composition piece-by-piece, stacking many small elements on top of each other to create the composition. Once the artist is satisfied with the recorded elements, this stage is often concluded by selecting the final takes that will be included and removing additional elements that are deemed unnecessary. The recording engineer will then deliver these final recordings to the mixing engineer for the next step in the process.

2.1.2 Mixing

After all the elements have been successfully captured, they are transferred to the mixing engineer. In this stage of the process, the mixing engineer applies their technical and artistic judgement to combine these elements into a cohesive stereophonic, or multi-channel mixture, which aims to achieve the artist's and producer's intention. This process generally involves the use of a mixing console as well as a combination of different signal processors that allow the audio engineer to shape the temporal and spectral characteristics of the recorded signals. The most common processes carried out by the mixing engineer include balancing, panning, equalization, compression, and additional effects such as reverb and delay (Owsinski, 2006).

Mixing is often challenging and requires great experience on the part of the audio engineer, as the interaction between the recorded elements can be very complex. For example, using an equalizer to adjust the quality of the low frequency content of the

[1]Readers are directed to Chapter 6 of Huber and Runstein (2010) for a complete treatment of digital audio fundamentals

kick drum may have a significant impact on the audibility of the bass guitar. Since they occupy a similar part of the frequency spectrum, they are prone to masking interactions (Wakefield and Dewey, 2015). This principle of interaction can then be applied between each pair of sources in the entire mix. Clearly then, as the number of individual tracks in the mix increases, the complexity of the mixing task increases.

This process also requires that the parameters of various signal processors be adjusted over time as elements within the mix evolve. This further complicates the process. Often there is no singular configuration of the mixing console that achieves a quality mix over the course of the whole work. To address this, mixing engineers often use what is known as automation, a process wherein the settings of different parameters are defined at each timestep in the project. Then during playback, the processors follow these pre-programmed settings.

While the process and art of mixing has been studied (De Man, 2017) and many resources are available to educate mixing engineers (Owsinski, 2006, Huber and Runstein, 2010, Case, 2011, Izhaki, 2013), formalizing these tasks into a set of predefined steps has not be achieved. Mixing engineers spend a great deal of time practicing and studying the work of accomplished mixing engineers in order to hone their craft. General practices and guidelines do exist in the aforementioned literature, but the diversity and complexity of the interactions among the elements of most mixes means that expertise and intuition specific to the mixing engineer is required to achieve the desired result.

2.1.3 Mastering

After the final mix has been created, it is often passed to the mastering engineer, who provides a final assessment of the work and additionally carries out any further processing to the complete mix. Their role is often to identify any problematic characteristics of the mix that will potentially result in poor translation when the content is reproduced across a range of different consumer playback systems (Katz and Katz, 2007). In order to achieve this goal, the mastering engineer often uses a very high fidelity monitoring system as well as signal processors similar to the mixing engineer, like equalizers and compressors to make corrective adjustments to the mix as a whole. In some cases, the mastering engineer may even request the mixing engineer return to the mixing stage to make specific adjustments that are better suited to be performed on an individual element directly, rather than across the stereo mix by the mastering engineer. Once the mastering engineer has signed off on the final production, the process of producing the work has been completed and it is now ready for distribution.

Reference	Effects	Param.	Architecture	Loss	F_s (kHz)	Dataset
Mendoza (2005)	dist. & EQ	no	MLP	L2 (t/f)	48.0	P
Holzmann and Hauser (2010)	distortion	no	RNN	L2 (t)	44.1	P
Covert and Livingston (2013)	distortion	no	RNN	L2 (t)	96.0	P
del Tejo Catalá and Masiá Fuster (2017)	equalizer	no	CNN+LSTM	L2 (t)	44.1	P
Martínez Ramírez and Reiss (2018)	equalizer	no	CNN	L1 (t) + L2 (f)	16.0	E
Schmitz and Embrechts (2018)	distortion	yes	LSTM	L2 (t)	22.05	P
Zhang et al. (2018b)	distortion	-	LSTM	L2 (t)	96.0	P
Damskägg et al. (2018)	distortion	yes	CNN	ESR (t)	44.1	P
Martínez Ramírez and Reiss (2019)	dist. & EQ	no	CNN	L1 (t)	16.0	A
Martínez Ramírez et al. (2019a)	time-varying	no	CNN+LSTM	L1 (t)	16.0	I
Martínez Ramírez et al. (2019b)	reverb	no	CNN+LSTM	L1 (t) + L2 (f)	16.0	I
Wright et al. (2019)	distortion	yes	LSTM	ESR (t) + DC(t)	44.1	B, C
Hawley et al. (2019)	compression	yes	MLP	Logcosh (t) + L2 (f)	44.1	G
Damskägg et al. (2019b)	distortion	yes	CNN	ESR (t)	44.1	B, C

Table 1: Summary of previous approaches in deep learning for audio effect modeling. The Param. column indicates whether or not the control parameters of the effect were modelled, or if only a single configuration of the model was used. For the loss functions, (t) denotes time domain and (f) denotes frequency domain. ESR denotes the error-to-signal ratio, and DC denotes a loss between the DC offsets. Complete details about the dataset used are shown in Table 2.2.2

2.2 Audio effect modeling

The field of virtual analog audio effect modeling has been an active area of research since the introduction of digital audio processors (Valimaki et al., 2010), wherein the goal is to model the often highly nonlinear response of various analog audio processors, using some kind of digital model to be implemented in software (Pakarinen and Yeh, 2009). Processors of interest often include compressors (Giannoulis et al., 2012), reverberators (Bilbao, 2013), and vacuum tube amplifiers (Karjalainen and Pakarinen, 2006). Virtual analog models provide the flexibility of working in the digital domain, most notably the reduction in physical space due to the elimination of hardware units, as well as effortless recall-ability, and extended use across a number of instances, generally limited only by the computing power available. This technology has become ubiquitous among both professional and hobbyist audio engineers with the deployment in many plugins that provide high quality emulations of analog hardware at a relatively low cost, such as offerings from plugin manufacturers Universal Audio[2] and Waves[3].

2.2.1 White-box models

There are two approaches traditionally employed in the task of analog effect modeling. The first is known as white-box modeling, where the behavior of an analog device is modeled by examining the circuitry and design of the device, and then a simulation of this same circuitry is implemented (Yeh et al., 2007). This can be

[2]https://www.uaudio.com/uad-plugins.html
[3]https://www.waves.com/bundles/abbey-road-collection

achieved by using existing physical models of these circuit components with popular simulation tools like SPICE (Vladimirescu, 1994). While these approaches have been shown to be powerful, their major drawback comes from the computational expense of high accuracy circuit simulations. This often forces the designers to simplify parts of the circuitry within the simulation, hence reducing its level of realism. Nevertheless, this approach remains popular and can produce high quality simulations of analog hardware when care is taken in the design of the circuit simulations.

2.2.2 Black-box models

The second approach is known as black-box modeling, where instead of examining the inner workings of the device and carefully simulating each component, a large number of measurements are made by passing different signals to the device and testing how it responds (Eichas and Zölzer, 2016). These measurements can then be used to construct a nonlinear model that attempts to emulate the behavior of the original device. With these measurements, designers generally attempt to fit a parameterized model to re-create this system, such as the Volterra series (Hélie, 2009), Wiener models (Schattschneider and Olzer, 2000), or Wiener-Hammerstein models (Eichas and Zölzer, 2018).

Over the past decade, researchers began to investigate the application of neural networks in an attempt to model the nonlinear behavior of these analog signal processors. The proliferation of hardware specialized in the matrix operations employed by neural networks makes these approaches potentially more efficient that white-box modeling approaches that employ demanding physical models, and the ability to learn more expressive functions than those employed in traditional black-box modeling approaches, ultimately makes the application of neural networks attractive for this task. Table 1 presents a summary of the neural network-based approaches, including details about their implementation. Some of the earliest work in this area involved the emulation of vacuum tube amplifiers for electric guitar distortion, first in Mendoza (2005), which employed a simple multilayer perceptron with both frequency domain and time domain approaches, as well as Covert and Livingston (2013), which investigated the application of recurrent networks in the time domain.

More recently, the latest advancements in deep learning have been applied to this area and have brought about improved results, as well as additional insight into the most effective methods to train and implement these kinds of models. Continued work on modeling vacuum tube amplifiers with neural networks has resulted in further improvements. This was achieved in Damskägg et al. (2019a) by employing a WaveNet-like architecture, in Zhang et al. (2018b) with the application of LSTMs, and in Martínez Ramírez and Reiss (2019) with a hybrid CNN and bidirectional

ID	Name	Year	Sources	Effects	Samples	Hours	GB	Sample Rate	Reference.
A	IDMT-SMT-Audio-Effects	2010	Bass/Guitar	Many	55044	30	-	44.1 kHz	Stein et al. (2010)
B	IDMT-SMT-Bass-Single-Tracks	2013	Bass	None	17	-	-	44.1 kHz	Abeßer et al. (2013)
C	IDMT-SMT-Guitar	2014	Guitar	None	6000	-	-	4.1 kHz	Kehling (2014)
D	MedleyDB V1	2014	Many	None	849	120	43.1	Multiple	Bittner et al. (2014)
E	Salamander Grand Piano	2015	Piano	None	1440	1.6	1.3	48.0 kHz	-
F	MedleyDB V2	2016	Many	None	428	70	44.3	Multiple	Bittner et al. (2016)
G	SignalTrain LA2A	2019	Many/Noise	Comp.	67	20	21.0	44.1 kHz	Hawley et al. (2019)
H	FSDnoisy18k	2019	Noise	None	18179	43	9.5	44.1 kHz	Fonseca et al. (2019)
I	DL-AFx	2019	Bass/Guitar	Many	2372	1.5	0.9	44.1 kHz	Martínez Ramírez et al. (2020a)

Table 2: Summary of datasets commonly used in the audio effect modeling task, as well as mutlitack datasets. Information about the number of samples, their length, and total size of the datasets are provided when available.

LSTM architecture. In addition, other more complex signal processors have begun to be modeled with deep learning approaches, such as dynamic range compressors in Hawley et al. (2019), which models both a digital compressor and analog LA-2A compressor. In Martínez Ramírez et al. (2020b), the authors present a further extension of Martínez Ramírez and Reiss (2019), which enables the modeling of spring and plate reverberators through the addition of SFIR filtering within the latent space of the model.

Limitations

While these findings indicate that DNNs are capable of learning both the linear and nonlinear transformations characteristic of traditional signal processors utilized by audio engineers, the investigations conducted thus far have some limitations. Some of these works train models that operate on audio at sampling rates lower than 44.1 kHz, and therefore additional work must be carried out to demonstrate that a range of effects can be modeled effectively at higher sampling rates. Also, many works, with with of exception of SignalTrain (Hawley et al., 2019), make strong assumptions about the type of inputs to the model, restricting this to only a certain sets of instruments, such as the electric guitar, electric bass, and piano. Finally, most works only consider a single configuration of the audio effect parameters, or a sparse sampling over this parameter space. In order to fully capture the behavior of these audio effects we must be able to model the audio effects over a dense sampling of their parameter space, which may present additional challenges.

2.3 Intelligent music production

Intelligent music production (IMP) is a research field that aims to develop algorithms that provide assistance in the process of recording, mixing, or mastering a musical production (Moffat and Sandler, 2019b). Implementations can range from systems that simply provide feedback or recommendations on how to improve a production, to the complete autonomous control of an entire mixing console in real-time for live

sound reinforcement. The field of IMP extends beyond autonomous mixing systems and also encompasses tools that provide semantic control of signal processors.

Moffat and Sandler (2019b) outline two different approaches in designing intelligent music production systems. They first introduce the concept of parametric systems, where a model is designed such as to produce an appropriate parameterization of a pre-existing signal processing device, such as the mixing console. The majority of previous systems have followed this approach as they produce interpretable results and enable interaction from users to easily tweak the models predictions according to their goals. They go on to introduce the concept of direct transformation systems, which instead of attempting to produce parameters for traditional DSP modules, operate directly on the inputs to produce the desired output. These systems open the door for new kinds of transformations that are potentially more expressive than existing hand crafted DSP algorithms. Although, this greater level of expressivity generally comes at the cost of less interpretablity, and the inability for users to easily tweak the generated mixes. This class of models have been relatively unexplored within the context of intelligent music production, and therefore future work is needed to address these challenges.

2.3.1 Early work

Somewhat surprisingly, the earliest attempts at designing a system to automate the task of the mixing engineer predate the era of digital audio. Dugan famously introduced his design of an automatic microphone mixing system in 1975 at the Convention of the Audio Engineering Society in Los Angeles (Dugan, 1975). His proposed system aimed to control the gain of multiple microphones for speech in a sound reinforcement scenario adaptively. By monitoring the level of other microphones, an algorithm was developed to achieve the maximum output without feedback. These mixers evolved over the past decades and remain in use within the industry, but aim only to provide simple control over the level of multiple channels in a scenario where an expert audio engineer is not required. The limited scope of these devices left a desire to develop more advanced control algorithms that went beyond just controlling the level, and ultimately would be better able to emulate the actions of a professional audio engineer. Nearly 25 years later, Pachet and Delerue (2000) developed a system that allowed listeners to adjust the spatialization of a multitrack mix while meeting a set of constraints set by the audio engineer. This work suggested that multitrack mixing could be framed as an optimization problem, and would go on to influence the future directions of the field.

2.3.2 Knowledge-based systems

It was work by Perez Gonzalez and Reiss, beginning in 2007, that initially began the reexamination of this research question of designing more powerful intelligent tools for mixing. These early systems were characterized by framing various mixing tasks as optimization problems, such as the minimization of perceptual masking between elements in the mix (De Man and Reiss, 2013a). They often also featured simple logic or rules to ensure the device under control did not become uncontrolled. These early systems established a general architecture for the design of what are considered intelligent audio processors. Most systems contain two signal paths: the main processing signal path, and the side-chain pathway in which some analysis of the input signals is undertaken (De Man et al., 2019). This side-chain pathway often involves some kind of feature extraction, where additional processing is carried out on input signals to gather information about the inputs. This information is then fed to an optimization algorithm, a knowledge-base containing rules, or a combination of the two. Figure 1 illustrates the basic architecture of a mutli-channel knowledge-based system for audio processing.

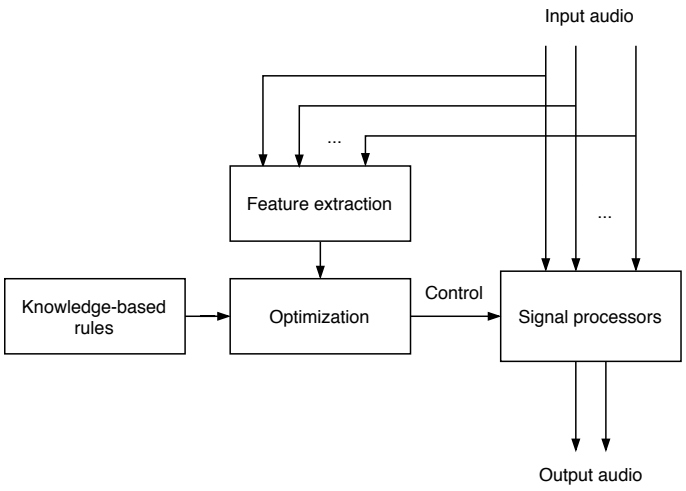

Figure 1: Block diagram of the basic design of a multi-channel knowledge-based intelligent audio effect, which is adapted from De Man et al. (2019).

For example, in Perez Gonzalez and Reiss (2007), the authors present an autonomous system for panning multiple tracks within a multitrack mixture. Their algorithm attempts to reduce spectral masking among K input tracks through a simple priority system and a filter bank of K filters that measure the energy within each band of each input. These values are then used to determine how to pan each source across different panning steps in the stereo field, with the goal of reducing spectral masking by panning sources with similar spectral content apart.

This initial work was extended to other tasks within the mixing process such as

equalization (Perez Gonzalez and Reiss, 2009a), delay correction (Perez Gonzalez and Reiss, 2008), and leveling (Perez Gonzalez and Reiss, 2009b). Similar targets such as masking reduction were used to direct the optimization process of controlling the parameters of these processors. Of course these methods assume both that the perceptual models used for masking are representative of human perception and furthermore that the set of underlying rules and constraints truly align with the internal goal of mixing engineers. Some of these early systems demonstrated promising results, specifically the results from the PhD thesis of Perez Gonzalez, which incorporated many of these subsystems together into a complete system for mixing a fixed set of multitrack sources. While this system achieved performance beyond any other existing system, the results left much to be desired. The algorithms were sensitive to parameter tuning and lacked the robustness required to adjust to the scale and diversity of real-world projects.

Following the groundwork laid by Perez Gonzalez and Reiss (2009a), more researchers began expanding upon these early systems. In addition to the goal of reducing spectral masking within a mix, approaches that attempted to achieve an equal loudness criterion across input channels were also investigated. This was first implemented in Perez Gonzalez and Reiss (2009b), which uses a simple model of time varying loudness based on the ISO 226 loudness curves (ISO 226:2003). Further advancements were made in Mansbridge et al. (2012) with the implementation of the EBU R-128 recommendation (R128) for extracting channel loudness features, and Ward et al. (2012) investigated the use of more perceptually accurate models of loudness. Work in this area has continued, most notably with Fenton (2018), which presented modifications to the ITU BS.1770 loudness algorithm (BS1770) for increased accuracy in the context of multitrack mixing.

While progress was made on expanding these largely constraint based approaches, a new direction emerged that placed a greater focus on expanding the quality of domain knowledge present in these systems. These systems aim to improve on earlier ones by building rich rule bases through the in-depth study of mixing practices. This work was led largely by De Man and Reiss who presented a framework for incorporating domain knowledge collected from a study of mixing engineers into a complete multitrack mixing system (De Man and Reiss, 2013b). While these systems offered additional benefit from their interpretablity, and often produced mixes with a level of quality above more basic baselines, they still lacked the ability to easily and seamlessly adapt to the diversity in real-world projects, and ultimately were unable to capture the same level of detail and adaptation afforded by human engineers De Man (2017).

2.3.3 Machine learning

Machine learning approaches differ from knowledge-based systems in that they involve the large scale processing of data in order to construct a system or algorithm that analyzes an input and predicts an output. In addition to the knowledge-based approaches, some of the early approaches in the field began by investigating machine learning methods. Notable works include a system in Kolasinski (2008) that utilized a genetic algorithm and audio features for leveling tracks, and Scott et al. (2011) that utilized linear dynamical systems to predict time varying gains based on extracted audio features and a dataset of mutlitack mixes. The application of machine learning approaches have also notably enabled the investigation of intelligent systems for more complex signal processors like dynamic range compression (Mimilakis et al., 2016) and artificial reverberation (Chourdakis and Reiss, 2016, 2017, Benito and Reiss, 2017), which have proven more difficult to design formalized rules for in knowledge-based approaches.

Ultimately, machine learning approaches have been relatively less explored in comparison to knowledge-based systems, and this is due to a number of factors. Probably the most significant is that lack of large scale datasets for music production tasks. Due to copyright and the lack of standardization across different music production software tools, it has remained challenging to collect significant amounts of data related to the processes normally carried out by audio engineers. In addition most machine learning formulations for this task require the mixing parameters themselves for the optimization process, which are often unavailable. While projects such as MedleyDB (Bittner et al., 2016) and the Open Multitrack Testbed (De Man et al., 2014) have aimed at filling this gap, the scale of data they provide is still very limited and often only contains the waveforms on the multitrack recordings and mixes. Nevertheless, interest in machine learning methods has been increasing in recent years (De Man et al., 2017), and as the field of deep learning develops, the application of these methods have become relevant as well (Martínez Ramírez and Reiss, 2017).

2.3.4 Deep learning

Deep learning is a subfield of machine learning that provides a framework for learning complex, nonlinear relationships from data, by composing multiple layers of parameterized computations. These techniques are generally characterized by the use of neural network architectures composed of multiple layers, whose parameters are optimized with the use of an appropriate loss function and the application of stochastic gradient descent and backpropagation. This approach differs from previous machine learning approaches in that instead of using features that are hand-crafted for each

type of input data and task, we train models that learn more powerful representations that are specialized for the task at hand.

The result of compositionality means that by stacking many layers of learnable computations we are able to build functions that are far more expressive than previous approaches, and this has led to major breakthroughs in many tasks in computer vision, audio, and natural language processing. As these successes in the field of deep learning have continued to increase across many domains, so have the results in application of these approaches to tasks in audio. Applications now have demonstrated state of the art performance across tasks in the audio domain such as automatic tagging (Pons and Serra, 2019), speech synthesis (Wang et al., 2017), speech recognition (Park et al., 2019), and music source separation (Stöter et al., 2019).

Many deep learning systems operating on audio follow a similar approach. Generally, they first perform a short-time Fourier transform (STFT) on the audio and extract the magnitude spectrum, discarding the phase component as it tends prove more complicated to utilize effectively. Optionally, additional transformations are performed on these representations, such as splitting the signal into mel-frequency bins or applying logarithmic scaling, in effort to instill perceptual characteristics. These time-frequency representations are then fed as input. For models that perform a classification task such as tagging or speech recognition, this kind of approach has been shown to work very well (Pons and Serra, 2019).

Challenges with this approach arise though when the goal of the model is to produce audio at its output. Since these methods operate only on the magnitude spectrum and produce a magnitude spectrum as output, they must reconstruct the time domain signal in an additional stage. Applications in source separation or speech enhancement often simply use the phase from the original signal to perform the reconstruction with the iSTFT (Stöter et al., 2019). In cases where this is not possible such as synthesis, the reconstruction process many utilize the Griffin-Lim algorithm (Griffin and Lim, 1984) to approximate the phase given a magnitude spectrum, but these results still leave much to be desired (Le Roux et al., 2008). More recent approaches train a network to reconstruct an audio waveform given a melspectrogram, and demonstrate promising results within the context of speech and music synthesis (Kumar et al., 2019). Although, such methods require the use of adversarial losses which may complicate training, and the use of intermediate representations like the melspectrogram adds additional complexity.

To address some of these shortcomings, a new class of models have been investigated, known as end-to-end approaches. They are unique in that they operate on audio in the time domain directly and at their output produce a time domain signal. Unlike the previous spectrogram-based approaches, there are no pre or post processing

stages to generate intermediate, hand-designed representations, instead we let the model find the optimal representations of the audio waveforms directly. This has the benefit of enabling the explicit modeling of the phase component of the signal jointly with the magnitude. Applications of this approach in speech enhancement (Pascual et al., 2017), instrument synthesis (Engel et al., 2017), and music source separation (Stoller et al., 2018), indicate similar performance to spectrogram based methods with potential benefits (Lluís et al., 2019). As these architectures have developed, the latest results indicate that they can surpass, or remain very competitive with the performance of spectrogram-based methods (Défossez et al., 2019).

Deep learning for multitrack mixing

A canonical end-to-end deep learning approach for the task of multitrack mixing may employ a convolutional neural network (CNN), where each input recording is passed to the model as a waveform, and the output is a two channel signal, representing the stereo mixture. Using a dataset of paired multitrack recordings and mixes of these inputs, the model could be trained using gradient descent where the L1 or L2 distance is minimized between the predicted and ground truth mixes. This problem formulation falls within the domain of direct transformation approaches introduced previously, and is quite reminiscent of the music source separation task, where instead of processing a mixture to produce multiple isolated sources, multiple isolated sources are combined to produce a processed mixture. Just as in the source separation task, this requires a model with sufficient depth to achieve a large enough receptive field, as well as enough model capacity to capture the variation among the mixes in the dataset, along with sufficient training examples.

As a direct transformation approach, the canonical end-to-end deep learning approach provides the potential benefit of not imposing any limiting inductive bias on the mixing task, leaving the network free to learn a set of expressive transformations, without the need for manually designing more complex subsystems. While this high level of model expressivity may be desireable, it comes at a cost. Likely more training data will be required as well as additional iterations over the training data, in order to achieve performance on par with a system with stronger inductive bias for the mixing task. In addition, since the entire mixing task is abstracted within the weights of the convolutional layers, and the output of the model is a complete mixture, there is no clear method to interpret the mix that is produced within the context of traditional mixing parameters. This proves quite problematic, since it leaves the user without any means to further adjust the output mix. This is true of course only in the simplest case of a discriminative model. The introduction of a generative component in the model could help to alleviate this issue, although this will now require users to navigate a latent space that could have arbitrary organiza-

tion, and therefore be quite unintuitive to navigate in comparison to the interfaces that audio engineers are already accustomed to.

In addition to these more apparent potential disadvantages, there are also a number of less obvious challenges in applying such an approach. The first comes from the requirement of defining the total number of input channels during training. The general, CNN architecture requires that the input layer contain one set of N kernels for each of the K input signals. In computer vision, this often manifests itself as the RGB channels of an image, or in the case music source separation, this may be the left and right channels of the input mixture. Since there is no single format for multitrack projects, the number of input channels may range from 1 to upwards of 200 depending on the complexity of the project. The simplest approach to this problem may involve setting the input of input channels equal to the largest expected input size, and then simply filling unused input channels with zeros. Unfortunately, due to the reality that the number of input channels across projects often appears as a long-tailed distribution, this method tends to be quite inefficient. Many of the input channels will be filed with zeros for a large number of projects, not to mention this greatly complicates the modeling problem.

Beyond this problem of defining the fixed number of input channels during training, there is also the problem of the ordering of these inputs. Again, since there is no established format with regards to the identity and number of inputs present in a project, different projects will contain different elements, which will be potentially passed as input to any of the network's input channels. In this case, we desire that, regardless of the ordering of these input channels, we produce the same mix. It is clear that the ordering of the these inputs (e.g. kick, snare, guitar, etc.), should not impact the mix produced. This is a challenge for the traditional CNN architecture, since it means that for a project with K input channels, there will be $K!$ different permutations of the input channels, each of which will require the model to produce the same mixture. All of these challenges remain unaddressed in the literature to our knowledge, and we aim to first bring these challenges to light and then address them through the design of a domain inspired deep learning approach, which we will introduce in the following chapter.

Chapter 3

Multitrack mixing system design

3.1 Problem formulation

At the beginning of the mixing process we have a collection of N monophonic audio signals, which we assume all are T samples in length. We can represent this set of inputs as $\mathbf{X} \in \mathbb{R}^{N \times T}$. Our task is then to process each of these signals with its own unique transformation and then sum these signals in order to create the final mixture. To address this task, audio engineers generally utilize an array of signal processing tools such as gain, panning, equalization, dynamic range compression, and reverberation. Each of these signal processing devices can be represented as an individual function $\mathbf{y} = f(\boldsymbol{\phi}_p, \mathbf{x})$ that takes as input a monophonic audio signal $\mathbf{x}$, a vector of parameters $\boldsymbol{\phi}_p$ that defines the configuration of the processor, and produces a processed audio signal $\mathbf{y}$, which may have one, two, or more channels.

A single channel strip within a mixing console is often constructed with a series connection of D processors, where the output of each processor is fed as input to the following one. The result is a composition of functions, where each function receives the output of the previous function, as well as its own unique parameterization $\boldsymbol{\phi}_d$, such that the output of the i^{th} channel is given by

$$\mathbf{y}_i = f_d(\boldsymbol{\phi}_d) \circ f_{d-1}(\boldsymbol{\phi}_{d-1}) \circ \cdots \circ f_1(\boldsymbol{\phi}_1, \mathbf{x}_i). \tag{3.1}$$

To generate a mix, a unique configuration for the processors in each channel is utilized to process each of the N input signals. Therefore, we can define the final stereo mix at the output of the mixing console, $\mathbf{m}$, as a sum of the outputs of each channel n, parameterized with its own set of states $\boldsymbol{\phi}_{n,d}$, for each processor d, given by

$$\mathbf{m} = \sum_{n=1}^{N} f_{n,d}(\boldsymbol{\phi}_{n,d}) \circ f_{n,d-1}(\boldsymbol{\phi}_{n,d-1}) \circ \cdots \circ f_{n,1}(\boldsymbol{\phi}_{n,1}, \mathbf{x}_n). \tag{3.2}$$

In practice, this composition of functions can be arbitrarily complex and may consist of many linear and nonlinear signal processors. There is no established method or technique for how to compose these processors, and hence the task of developing an appropriate signal chain, as well as generating the optimal parameterization is a very challenging task. Additionally, this representation of the mix is a simplification of the routing possibilities afforded by traditional mixing console, which include the ability to create auxillary busses. This complexity in the signal processing configurations is one of the factors in why extensive training is required on the part of the audio engineer for this mixing task.

For this reason, in our work we define a fixed chain of processors that provides adequate transformational power for a general multitrack mixing task. We will detail the design of this signal chain further in Chapter 4. It should be noted that this assumption simply places an upper limit of the number of the processors in the chain, as each processor can be parameterized in such a way that it does not affect the signal, and is effectively removed from the signal chain. Then, in the case of an automated multitrack mixing system, where we are given a collection of input recordings $\mathbf{X} \in \mathbb{R}^{N \times T}$, we aim to build a model that predicts a set of P parameters for each of the N inputs, $\boldsymbol{\phi} \in \mathbb{R}^{N \times P}$, such that a *desireable* mixture is achieved.

3.1.1 Goal definition

It is important to carefully define what we mean by a *desireable* mixture. The mixing task, seeing as it combines both technical and creative processes, will produce an output assessed within an artistic context. Unlike other common tasks in audio signal processing, such as source separation, denoising, or dereverberation, the mixing task is largely ill-defined, meaning there lacks a single, agreed upon target, given a set of input recordings. This is evidenced by the fact that when multiple audio engineers are provided with the same set of input recordings, they tend to produce mixes that are different from each other, but are still all rated favorably by listeners (De Man and Reiss, 2017). This presents a challenge for our learning problem.

Therefore, our aim is not to "solve" this task, since no objectively correct mix exists for a given input. Instead, our focus is on modeling the music production conventions present within our dataset. We can then classify a *desireable* mixture as one that follows the conventions implicit within the training data. Unfortunately evluating the degree to which a system follows these conventions remains quite challenging, and while multiple attempts have been made to formalize objective

metrics for characterizing this relationship (Wilson and Fazenda, 2015, 2016, Colonel and Reiss, 2019), this remains an open problem in the field. As a result, our aim is to build a model that is able to process recordings in a manner similar to trained audio engineers, based upon these conventions, with the goal of providing utility to users by suggesting appropriate mixing parameters. To address this limitation of objective evaluation metrics, we must rely most heavily on the feedback provided by trained listeners in the assessment of our approaches.

3.2 Learned differentiable mixing console

To facilitate learning from limited data, avoid undesirable artifacts, as well as produce interpretable mixing decisions, we would like to utilize an existing mixing console to carry out processing of the input recordings. A controller network would then be trained to predict the optimal parameterization of each channel, as shown in Figure 2, where the controller network receives the input signals, extracts information, and produces as output a set of parameters for each channel. In order to train this controller network with modern deep learning techniques, we need to employ an appropriate loss function and compute gradients with respect to the weights in the controller network, so we can optimize the network for the mixing task.

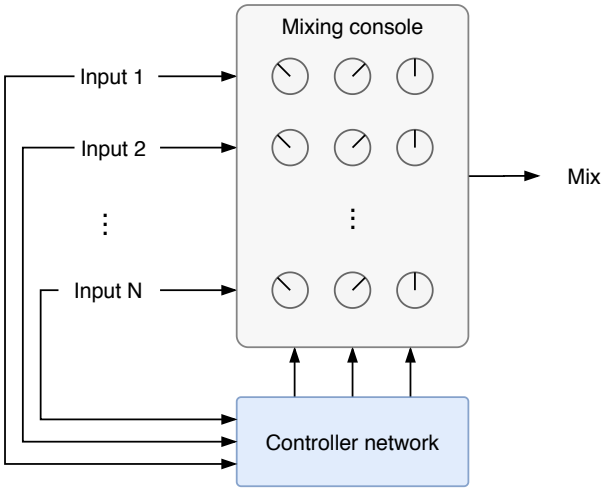

Figure 2: Desired formulation where a controller network learns to predict the optimal parameterization of a traditional mixing console given some input recordings.

Unfortunately, some of the processing operations employed in the mixing console likely do not have tractable derivatives, such as the dynamic range compressor. Furthermore, even if their derivatives prove to be tractable, they present a challenge in easily implementing them with existing automatic differentiation frameworks. Additionally, these functions may have a loss landscape that proves to be significantly

more difficult to optimize compared to traditional deep learning building blocks, for which we have robust techniques that are shown to facilitate convergence, such as weight initialization, batch normalization, and appropriate activation functions.

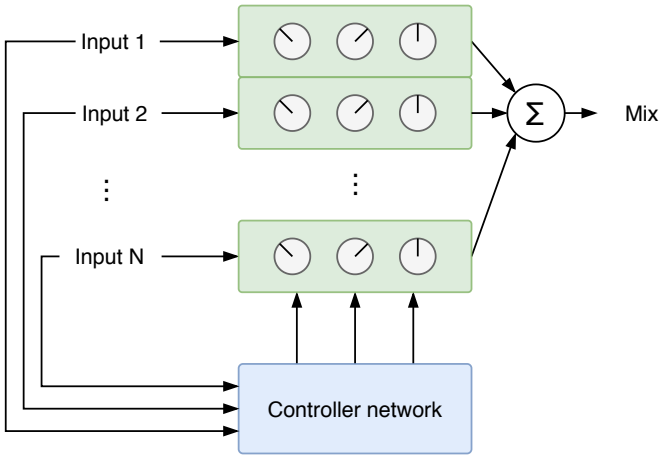

Figure 3: Proposed formulation where a controller network learns to predict the optimal parameterization of a mixing console proxy, which is a neural network pre-trained to emulate the processing of a mixing console.

Such an approach, where we explicitly define these processing functions and use their formulation to perform backpropagation, requires that we construct a unique differentiable implementation for each manifestation of the processor class (e.g. equalizer, compressor, etc.). This presents a challenge in scaling this method to the diversity of different existing processors. While this may not appear to be an issue, there is in fact large diversity among different formulations of the same class of processors, such as the dynamic range compressor, which can be implemented in a number of unique manifestations (Giannoulis et al., 2012).

To address these challenges, we propose an architecture in which the mixing console itself is replaced with a set of neural networks, allowing us to build a fully differentiable mixing console with deep learning building blocks. To do so, we note that the mixing console itself is composed of a set of repeated channels, all of which enable the same set of transformations using a composition of processors. Therefore, in order to emulate the mixing console, all we must do is to emulate a single channel within the console, and then repeat this channel across each input recording. We achieve this by designing an appropriate network and training it to emulate the signal processing chain of a single channel in the mixing console, with the ability to extend this network across multiple input recordings. Ideally, when this network is given an audio signal and parameters for processors in the channel, it will produce an audio signal that is indistinguishable from the output of the true mixing console channel.

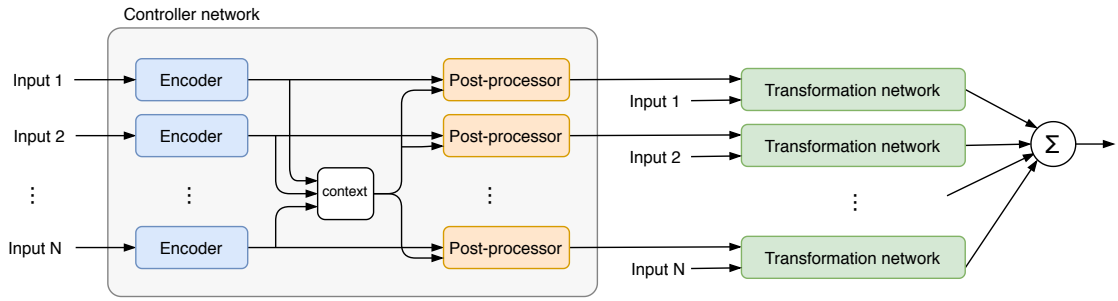

Figure 4: High level overview of the proposed multitrack mixing system.

This network, which we call the transformation network, is pre-trained in a separate task before being employed in the mixing console. We train this network to emulate the signal chain of a single channel in a mixing console, where the network takes as input a waveform and a set of parameters, and then produces a processed waveform, as would the original channel. As shown in Figure 3, this transformation network is then applied across each input channel, and the controller network now produces one set of parameters for each input channel to produce a mix, which is the sum of the outputs of each transformation network. This method of weight sharing across the input recordings addressed some of the previously identified challenges. With this proposed design we are able to train the controller network, as the complete system is fully differentiable, and moreover is composed of standard deep learning building blocks (e.g. convolutional layers and nonlinear activations), so we can take advantage of the existing techniques in training such networks. Additionally, this method provides the benefit of easily scaling to many different effects, and potentially growing the size of the signal chain without the need for creating unique differentiable implementations of each new digital audio effect. This even enables the ability to directly model well-known digital plugin implementations already in use by audio engineers by leveraging a VST host that is controllable programmatically[1].

3.2.1 Transformation network

The transformation network replaces the traditional channel in a mixing console, and attempts to operate in a identical manner when provided with an input signal and a set of parameters. By composing multiple instances of the same pre-trained transformation network, we can construct a complete differentiable mixing console that ultimately enables training of the controller network, while providing an inductive bias during the learning process to facilitate learning from limited data. This inductive bias is key to building a model that can generalize to the mixing of unseen examples since our training data is so limited, on the order of a few hundred.

[1]https://github.com/DBraun/DawDreamer

Before training the controller network, we must first train the transformation network in a separate task. In this task we aim to emulate the behavior of the mixing console channel. We achieve this by employing a self-supervised training regime. A DSP implementation of a channel is used, where its parameterization can be randomly configured during training, and a large dataset of audio examples can be passed as input to produce a practically infinite number of ground truth targets. We implement a number of different digital audio effects in an open source Python package called **pymixconsole**[2], for easy integration with the training process, as explained in further detail in Chapter 4. Once this network is trained, its weights can be frozen, and it can be placed within the complete system in order to train the controller network.

Previous work has demonstrated that a single audio effect, like those employed in a mixing console channel, can be modeled by a neural network in this manner (Martínez Ramírez et al., 2020a, Wright et al., 2020). Although, it has yet to be demonstrated if these methods are able to model a series of these signal processors, jointly, with a dense sampling over their continuous parameter spaces. To our knowledge, all pre-existing approaches have trained a single model for each effect to be modeled, and often only model a small number of configurations of the audio effect in question. In order to utilize such models for multitrack mixing, we aim to model a series of linear and nonlinear effects using a single network. We are also interested in the ability of this model to capture the behavior that arises for varying the ordering of these processors, if possible. In addition, we desire that this network operates on audio at a sampling rate of at least 44.1 kHz, makes very few assumptions about the type of input sources, fully captures the range of parametric control offered by the original effects, and introduces limited artifacts in the processing of inputs, all of which we will address in the following chapter.

3.2.2 Controller network

The controller network aims to generate a set of parameters, one for each of the input channels within the mix. To do so, there are two main steps that are employed as two separate subnetworks: the encoder, and the post-processor. Additionally, in order to address the previously introduced challenges, we employ weight sharing throughout the system, where one instance of the encoder and post-processor is applied individually to each of the input channels, producing a set of parameters for each channel to be passed to the set of transformation networks, which also have shared weights. The complete system is shown in Figure 4, where weight sharing across the N input channels is shown for each of the three sub-networks that comprise the complete system. Again, the controller network's role is to process the

[2]https://github.com/csteinmetz1/pymixconsole

input channels, extract useful information about these inputs, and finally produce a set of parameters that will be passed to the associated transformation network in order to process the input channel in question. Ideally, after training, the summation of all the processed channels will result in a *desireable* mix.

Encoder

The encoder assumes the role of extracting relevant information from the input channels. The kind of information that may be relevant to the mixing task may be characteristics like the identify of the input (e.g. guitar, percussion, voice, etc.), as well as more detailed information such as the energy envelop over time, or the allocation of energy across the frequency spectrum, which could be critical in understanding masking interactions among sources. These are the same kinds of considerations that audio engineers may make when attempting to create a mix. This encoder will then produce a representation for each input signal, which will be passed on to the set of post-processors.

Post-processor

The role of the post-processor is then to aggregate information from the encoder in order to make a decision about how to parameterize the transformation network operating on the associated input recording. Critically, we note that this decision cannot be made in isolation from the other input channels, as each mixing decision is highly dependent on all other inputs. These are the same considerations that a human audio engineer would make when accessing differing options in a mix. Therefore, we provide the post-processor with a learned representation not only of the respective input recording, but also a learned representation that summarizes all of the other inputs, in our case a simple average across all input representations, which we denote as the context representation, shown in Figure 4. With this information the post-processor can then make a decision about how to set the parameters for the respective transformation network.

Chapter 4

Audio effect modeling

Audio effects are tools used by audio engineers to apply processing to recorded signals in order to adjust perceptual attributes of signals such as loudness, timbre, pitch, spatialization, or rhythm (Wilmering et al., 2020). These devices can operate on signals either in the analog or digital domains. While analog signal processors retain some relevance in modern audio production, today the majority of audio effects are implemented using digital signal processing (Valimaki et al., 2010). These audio effects may manifest as physical hardware units employing digital circuitry, or more commonly as software that can run on a number of different consumer platforms such as mobile devices and laptops, generally within a DAW.

4.1 Problem formulation

We will define a generalized audio effect as a system that takes as input $\mathbf{x}(n)$, a discrete time signal of $n-1$ samples, and produces some discrete time signal $\mathbf{y}(n)$, as its output. Additionally, an audio effect generally has a set of controls or parameters $\boldsymbol{\phi}_p$ that define the manner in which the audio effect will operate on the input. Most often in digital audio effects, these parameters are exposed via user interface elements displayed on screen. By adjusting these parameters, users are able to modify the underlying behavior of the signal processing algorithm, and therefore achieve a range of perceptual effects.

We can view this generalized audio effect as a function $f(\boldsymbol{\phi}_p, \mathbf{x})$ that takes as input the signal $\mathbf{x}$, a vector of values sampled at a constant rate (we drop the sample index n for readability), and the parameters $\boldsymbol{\phi}_p$, a vector containing the parameterization of the audio effect. In the task of modeling the audio effect, we want to learn a function $g(\boldsymbol{\phi}_p, \mathbf{x})$ such that

$$g(\boldsymbol{\phi}_p, \mathbf{x}) = f(\boldsymbol{\phi}_p, \mathbf{x}) \; \forall \; \boldsymbol{\phi}_p, \mathbf{x}.$$

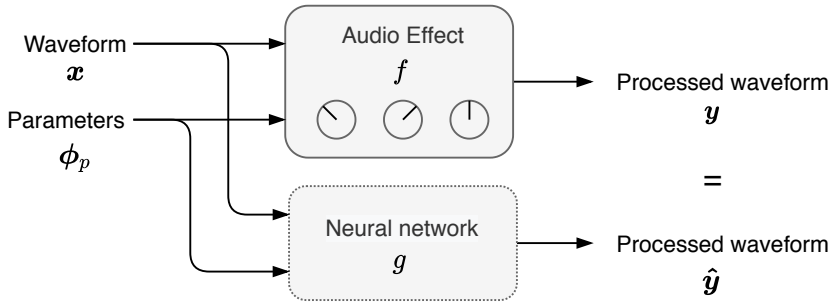

Figure 5: Block diagram of the audio effect modeling task, where we train a neural network to closely imitate an existing DSP implementation of the audio effect.

This audio effect takes as input two sets of values, the audio waveform itself, and the parameters of the effect, as illustrated in Figure 5, Our desire is to construct a model that when given these same inputs produces the same output. In our case, we have direct knowledge of the underlying function we aim to model, which we could use to help build an informed model, an approach known as white-box modeling. Although, for our use case we do not directly leverage this information, since unlike traditional modeling approaches that aim to simply emulate the underlying function, we aim to create a proxy for this function that is composed solely of deep learning building blocks. In a following section we will address the process of determining a set of deep learning architectures that may be appropriate for this modeling task.

4.2 Audio effect implementation

In this section we will briefly introduce the audio effect implementations that we attempt to model. While many of these audio effects have been modeled in the literature thus far, our ultimately goal is combine these effects into a single chain of effects, which will be modelled by a single network, jointly.

4.2.1 Gain and panning

The gain effect is the foundational transformation in audio processing, and involves applying a simple scalar, α, to the input vector of samples, $\mathbf{x}$,

$$\mathbf{y} = \alpha\mathbf{x}. \tag{4.1}$$

In the context of multitrack mixing, a gain is often implemented in the fader of a mixing console to enable the audio engineer to control the relative levels of the different elements within the mix. Due to the logarithmic nature of the human auditory system, it is often useful to adjust this gain value using a decibel (dB) scale, where the gain value α would be a function of the gain in decibels g_{dB},

$$\alpha = 10^{\frac{g_{\text{dB}}}{20}}. \tag{4.2}$$

Stereo or multichannel panning is also achieved with the use of a gain control. The panning process involves taking a single channel input $\mathbf{x}$, and producing a stereo output, by applying a different gain value to each output channel. This method of level-based panning takes advantage of the effect of the interaural level difference (ILD), one method by which the human auditory system is able to localize sounds in space simply by differences in the level of the sound reaching the ears (Stevens and Newman, 1936).

The simplest of these panning methods, know as linear panning, is parameterized by a single value $\theta \in [0, 1]$, where the two channels of the stereo signal are given by,

$$\mathbf{y}_L = \theta\mathbf{x} \tag{4.3}$$

$$\mathbf{y}_R = (1 - \theta)\mathbf{x} \tag{4.4}$$

such that as θ is increased, the signal is progressively panned more to the right, and the level of the signal in the left channel is decreased proportionally. Since both the fader and pan controls are implemented with simple gains, and hence are easily differentiable, we do not attempt to directly model these effects in our signal chain. Instead we directly apply these gain and pan values in the differentiable mixing console, as we can backpropagate through these operations.

4.2.2 Equalization

Equalization or EQ is concerned with providing a means to adjust the spectral content of a signal. This is generally achieved with the use of finite impulse response (FIR) or infinite impulse response (IIR) filters that can be applied to the input. FIR filters, those which the output is conditioned only of the current and previous inputs, and not on previous outputs, see some use in multitrack mixing applications. Although, to produce useful transfer functions they often must be of very high order in order, and therefore tend to induce significant delay, and require additional computations. In contrast, the IIR filter, which incorporates the output signal, by feeding it back to the input, is much more popular, as it can achieve complex transfer functions with filters of significantly lower order. For this reason we aim to model a cascade of second order IIR filters, known as biquad filters.

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \tag{4.5}$$

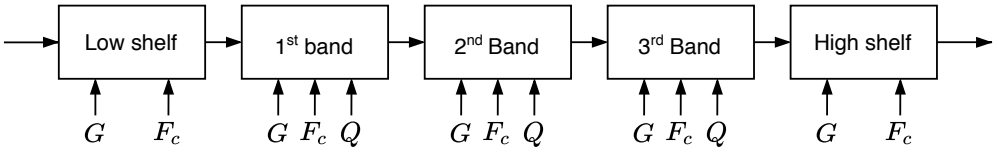

Figure 6: Block diagram of the parametric equalizer with a low shelf filter, three peaking filters, and a high shelf filter, where G is the gain, F_c is the cutoff frequency, and Q is the Q-factor, or bandwidth.

The biquad filter is defined as a function of six filter coefficients: a_0, a_1, a_2, b_0, b_1, and b_2, with a transfer function as shown in Equation 4.5. "The Cookbook formulae for audio equalizer biquad filter coefficients"[1] derives functions to compute these coefficients for a number of different filter types of interest.

In our formulation, we adopt the popular cascade of filters that composes what is known as a parametric equalizer, which features a low-shelf filter, three peaking filters, and finally a high-shelf filter. Parametric equalizers may come in a range of different varieties with more or less *bands*, as they are referred to, but in our formulation we choose a simple, yet fairly standard design that enables a significant amount of flexibility and control. Each of the peaking filters is intended to be used to adjust the spectral content of a different portion of the spectrum, and the shelving filters provide more broad control at the ends of the spectrum. Each of these filters is *parameterized* by the gain, cutoff frequency, and Q-factor, with the exception of the shelving filters, which feature only gain, and cutoff frequency controls. A block diagram of the complete parametric equalizer is shown in Figure 6. With such an equalizer, the audio engineer is provided quite extensive control over the spectral shape of a recording.

In order to compute the coefficients for these filters, we refer to the aforementioned derivations, which involve first computing intermediate values that are a function of the filter gain, G, in dB, cutoff frequency, F_c, in Hz, Q-factor, as well as the sampling rate, F_s, shown in Equations 4.6, 4.7, and 4.8. With these intermediate values, we can then compute the coefficients of the peaking filters as shown in Equation 4.9.

$$A = 10^{G/40} \tag{4.6}$$

$$\omega_0 = 2\pi \frac{F_c}{F_s} \tag{4.7}$$

$$\alpha = \frac{\sin \omega_0}{2Q} \tag{4.8}$$

[1] https://webaudio.github.io/Audio-EQ-Cookbook/audio-eq-cookbook.html

$$
\begin{aligned}
b_0 &= 1 + \alpha A \\
b_1 &= -2\cos\omega_0 \\
b_2 &= 1 - \alpha A \\
a_0 &= 1 + \frac{\alpha}{A} \\
a_1 &= -2\cos\omega_0 \\
a_2 &= 1 - \frac{\alpha}{A},
\end{aligned}
\tag{4.9}
$$

The computation for the low shelf filter coefficients differs, as shown below,

$$
\begin{aligned}
b_0 &= A\left((A+1) - (A-1)\cos\omega_0 + 2\sqrt{A}\,\alpha\right) \\
b_1 &= 2A\Big((A-1) - (A+1)\cos\omega_0\Big) \\
b_2 &= A\left((A+1) - (A-1)\cos\omega_0 - 2\sqrt{A}\,\alpha\right) \\
a_0 &= (A+1) + (A-1)\cos\omega_0 + 2\sqrt{A}\,\alpha \\
a_1 &= -2\Big((A-1) + (A+1)\cos\omega_0\Big) \\
a_2 &= (A+1) + (A-1)\cos\omega_0 - 2\sqrt{A}\,\alpha,
\end{aligned}
$$

and with some slight adjusts, finally the high shelf filter coefficients,

$$
\begin{aligned}
b_0 &= A\left((A+1) + (A-1)\cos\omega_0 + 2\sqrt{A}\alpha\right) \\
b_1 &= -2A\Big((A-1) + (A+1)\cos\omega_0\Big) \\
b_2 &= A\left((A+1) + (A-1)\cos\omega_0 - 2\sqrt{A}\alpha\right) \\
a_0 &= (A+1) - (A-1)\cos\omega_0 + 2\sqrt{A}\alpha \\
a_1 &= 2\Big((A-1) - (A+1)\cos\omega_0\Big) \\
a_2 &= (A+1) - (A-1)\cos\omega_0 - 2\sqrt{A}\alpha.
\end{aligned}
$$

4.2.3 Dynamic range compression

The dynamic range compressor is also a critical tool employed by the audio engineer in the mixing process. Unlike the equalizer, which applies a linear time-invariant

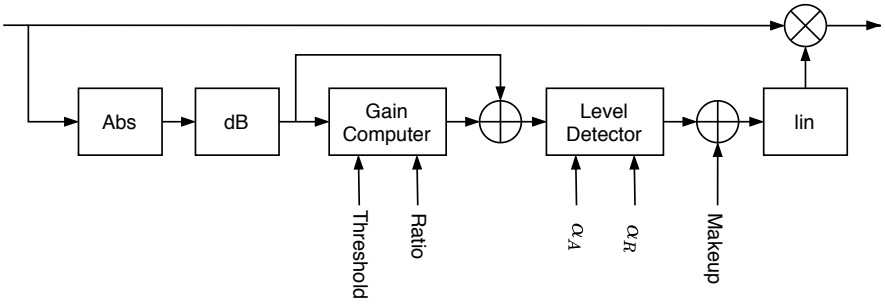

Figure 7: Block diagram of the simple hard-knee dynamic range compressor adapted from Reiss and McPherson (2014).

transformation to the underlying signal, the dynamic range compressor enables non-linear time-dependent processing, generally by adapting its behavior based upon the amplitude of the input signal. There are a number of uses for the dynamic range compressor, or simply compressor, including controlling transients, increasing the sustain of notes, and for creative effects (Reiss and McPherson, 2014).

There are many approaches that may be employed in the dynamic range compressor, and there tends to be far less consistency among implementations in comparison to the equalizer (Giannoulis et al., 2012). In our implementation we choose to employ a very basic feedforward compressor design with a hard knee, with a level detector with the difference equation with attack time, α_A and release time, α_R,

$$y_L[n] = \begin{cases} \alpha_A y_L[n-1] + (1-\alpha_A)x_L[n], & x_L[n] > y_L[n-1] \\ \alpha_R y_L[n-1] + (1-\alpha_R)x_L[n], & x_L[n] \leq y_L[n-1] \end{cases}$$

with the compression characteristic given by

$$y_G[n] = \begin{cases} x_G[n], & x_G[n] \leq T \\ T + \frac{(x_G[n]-T)}{R}, & x_G[n] > T \end{cases},$$

where we compute the input signal level in the log domain

$$\begin{aligned} x_G[n] &= 20 \times log_{10}|x[n]| \\ x_L[n] &= x_G[n] - y_G[n] \\ c_{dB}[n] &= -y_L[n]. \end{aligned}$$

4.2.4 Reverberation

Reverberation effects involve manipulating a recording to make it sound as if it was recorded in a different acoustic space. The room a sound is recorded in can often have a significant impact on the perception of sound, and audio engineers often use this reality to create productions that sound more natural, or potentially other worldly, through the use of artificial reverberation (Case, 2011).

Early artificial reverberation systems involved simply playing back recorded sounds through a speaker placed in a reverberant room, along with a microphone in the same room. Obviously this method was quite costly as it required a dedicated space, often known as the reverb chamber. More compact approaches generally employed vibrating materials, along with transducers to put the materials in motion, which would then be recorded by the other transducers, such as spring or plate reverberators (Doyle, 2005).

With the introduction of digital signal processing, digital implementations of feedback delay networks enabled the introduction of algorithmic reverb systems, which enabled parametric control of virtual spaces that aimed to approximate room reverberation (Schroeder and Logan, 1961). Additionally, rooms may also be modeled as linear time invariant systems, and therefore characterized by their impulse response, which can be measured using appropriate recording equipment. With this impulse response, a convolutional reverb can be used to closely recreate the acoustic characteristics of the captured room on dry signals during post-production. While convolutional reverbs can offer a high level of realism, they are often more computationally expensive than algorithmic reverb, and provide less flexibility.

In our implementation we employ a convolutional reverb, which given an input signal x of N samples, and the impulse response of an acoustic space, h, we can compute the reverberant output signal, y, as follows.

$$y[n] = (x \circledast h)[n] = \sum_{k=0}^{N-1} x[n-k] \cdot h[k]$$

Although, implementing the reverb as a convolution in the time domain can be very costly computationally, and since we aim to generate training examples on-the-fly while training, we would prefer an optimized implementation. Therefore, instead of performing the operation in the time domain, we take advantage of the convolution theorem, which allows us to perform a convolution in the time domain with a multiplication in the Fourier domain. Combined with block-based FFT algorithms we achieve a significantly faster convolutional reverb implementation.

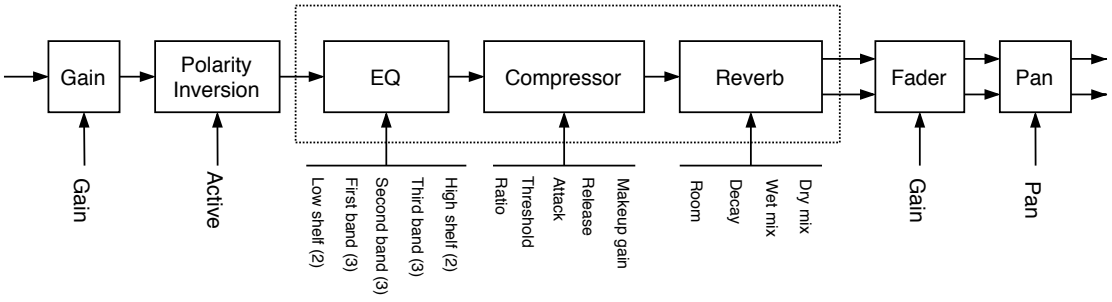

Figure 8: Block diagram of the complete channel strip formulation that aims to mimic the canonical series connection of processors found in a single channel of a multitrack mixing console. The input to the chain is always a mono signal, and the output will be a stereo signal.

4.2.5 Channel signal chain

We combine these audio effects, in order to construct a signal chain that imitates a chain of effects often found within a single channel of a multitrack mixing console. This signal chain is shown in Figure 8, which introduces a few additional elements to complete the channel strip. The parameters that each processor receives are shown adjacent to each processor. Only the three processing elements within the dashed box (EQ, compressor, and reverb) will be modeled by the neural network, since the rest of the elements are simply gain-like processors, for which gradients can easily be computed in the final stage when we will train the controller network. Therefore, at this stage, we are interested only in modeling the behavior of the three core processors that present some challenges in computing their gradients.

A Python implementation of these digital audio effects is available in our open source library **pymixconsole**[2], which includes optimized code using the **numba**[3] library for runtime compilation. This library enables the ability to easily pass numpy arrays containing audio signals, so that they can be processed by a series connection of audio processors. Additionally, this library provides specific support for handling the randomization of the parameters of all of the implemented processors, and enables the ability to carefully define the distributions from which their values are sampled.

4.2.6 Analog dynamic range compression

In addition to the digital audio effects presented so far, for which we generate effectively limitless training examples, we are also interested in determining how the proposed models perform in the task of modeling analog signal processors. In this

[2]https://github.com/csteinmetz1/pymixconsole
[3]http://numba.pydata.org/

task, the behavior of the system is often nonlinear, and since the process is implemented with physical hardware components, we are able to generate data only by passing different input signals, varying the physical parameters, and recording the signals at the output. Ultimately, this is a quite labor intensive and slow process, so only a limited amount of training data is available.

The task of modeling the LA-2A leveling amplifier, an analog dynamic range compressor originally introduced in the early 1960s, was proposed in Hawley et al. (2019). They propose a time domain based architecture for this task, along with a dataset of input and output pairs over a discrete range of settings of the physical compressor. In our work we aim to explore also how our proposed models perform on this task, wherein the device to be modeled has no tractable mathematical representation, and we are limited to using only the relatively small number of training examples provided in their dataset, which totals around 20 hours of recordings.

4.3 Architectures

From the machine learning viewpoint, the task of modeling the audio effect can be seen as sequence modeling task, where the input is a sequence that we aim to modify in order to create a new sequence. The traditional framework for modeling sequential data within the deep learning paradigm has been the recurrent neural network (RNN), which extends feed-forward neural networks for sequential data with the ability to utilize past states to influence current predictions (Goodfellow et al., 2016). While early RNNs presented a number of challenges during the training process, such as exploding and vanishing gradients (Hochreiter, 1998), improved formulations such as long short-term memory (LSTM) networks (Hochreiter and Schmidhuber, 1997) and the gated-recurrent unit (GRU) (Cho et al., 2014) have addressed many of these challenges. With these improvements, more modern RNNs have demonstrated impressive performance on many sequence modeling tasks like machine translation (Wu et al., 2016) and speech recognition (Graves et al., 2013).

While modern RNNs exhibit a number of desirable characteristics, such as the ability to capture long-term temporal structure, they cannot be parallelized due to their autoregressive nature, meaning they tend to be significantly slower than feed-forward approaches. More recently, convolutional neural networks (CNNs), which traditionally have been employed in computer vision tasks, performing convolution in the spatial domain, have been demonstrated to outperform comparable RNNs on a range of sequence modeling tasks, with greater efficiency (Bai et al., 2018). For this reason, in our investigations we opt to employ CNNs for the modeling task instead of RNNs.

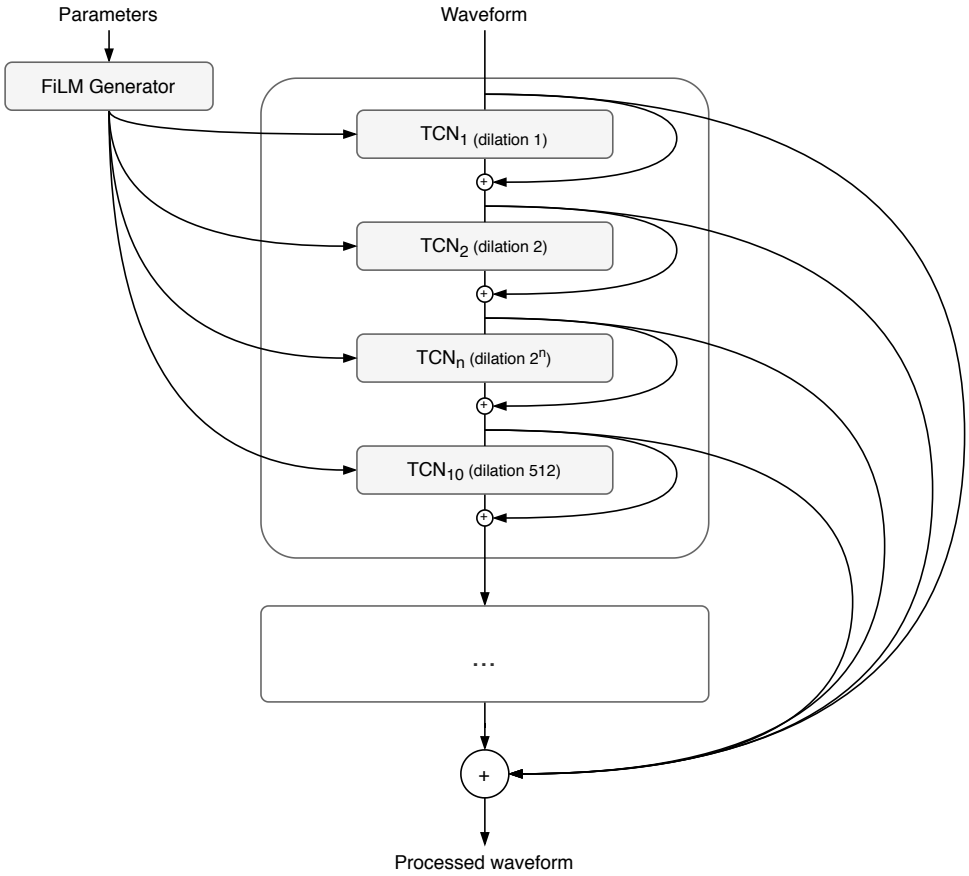

Figure 9: High-level view of the TCN architecture for the effect modeling task.

4.3.1 Temporal convolutional network

While CNNs have traditionally been employed for processing visual data, originally proposed for handwritten digit recognition (LeCun et al., 1998) and further popularized by their early success in image recognition (Krizhevsky et al., 2012), interest has grown in the application of these models for sequential data. In this formulation, instead of learning stacks of two dimensional kernels in the spatial domain, one dimensional kernels are learned that search for patterns in the input across time.

The temporal convolutional network (TCN) formalizes the application of CNNs to time series data, and includes a number of key components such as one dimensional kernels, convolutions with exponentially increasing dilation factors, as well as residual connections (Bai et al., 2018). Generally, the TCN formulation includes causal convolutions in which only past information is used to preform prediction, since such assumptions are useful for time series forecasting. For cases where the entire signal is available, as is the case in many audio signal processing tasks, this condition can be relaxed and non-causal convolutions can be employed taking advantage of "future" information, which often enables better performance.

While the TCN formalized the application of CNNs for time series data, there were applications that employed components of this model architecture previously in the literature. The WaveNet architecture, a generative model for speech synthesis operating in the time domain, was one of these early examples (van den Oord et al., 2016a). While it employed an autoregressive formulation that proved slow and difficult to train, one of the core components of this architecture was the use of dilated convolutions. Dilated convolutions enable a larger receptive field in a parameter efficient manner by inserting zeros between values in the kernels, effectively increasing their size. In the case of WaveNet, this was critical in order to produce a model with a receptive field large enough to capture temporal dependencies in speech signals, where a single second of audio was composed of 16,000 individual time steps.

Following the initial success of WaveNet, interest in the application of CNNs with dilated convolutions for processing audio signals in the time domain increased. In tasks where the goal is to process an audio signal (e.g. speech de-noising) instead of synthesizing a new signal (e.g text-to-speech), researchers realized that the autoregressive formulation of WaveNet could be eschewed in favor of a feed-forward formulation. This resulted in a TCN-like architecture with dilated convolutions, residual and skip connections, and a non-causal structure, which was applied first for speech de-noising (Rethage et al., 2018), and later for music source separation (Lluís et al., 2019), with some success compared to previous approaches.

In our formulation of the TCN for the task of audio effect modeling we adapt the architecture introduced in Rethage et al. (2018), but make a number of modifications. In a similar manner, we construct a network by creating stacks of convolutional blocks, with exponentially increasing dilation factors, such that the n^{th} convolutional block within a stack has a dilation factor of 2^n. Each block within a stack has a residual connection as well as an additive skip connection to the output. We create a single stack with the connection of 10 convolutional blocks, producing a series of layers with dilations $d = 1, 2, 4, ..., 512$, as shown in Figure 9. We further extend the receptive field of the model by repeating this stack structure multiple times, each time using this same dilation pattern. We then construct three different TCN models, each with an additional stack of ten convolutional blocks, which we denote TCN-10, TCN-20, and TCN-30.

While the convolutional blocks employed in their implementation more closely follow the original WaveNet, we make a number of modifications as shown in Figure 10. Each convolutional block is composed of a standard formulation that features a one-dimensional convolution, batch normalization (Ioffe and Szegedy, 2015), an affine transformation to inject conditioning via the FiLM mechanism (to be addressed in greater detail in Section 4.3.3), and finally a parameterized rectified linear unit (PReLU) activation, which generalizes the LeakyReLU activation by making the

slope a trainable parameter (He et al., 2015). Following the original FiLM implementation, we use batch normalization as shown in Equation 4.10, without its own affine transformation, such that $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are applied as a result of FiLM (Perez et al., 2018).

$$\mathbf{y} = \frac{\mathbf{x} - \mathrm{E}[\mathbf{x}]}{\sqrt{\mathrm{Var}[\mathbf{x}] + \epsilon}} * \boldsymbol{\gamma} + \boldsymbol{\beta} \tag{4.10}$$

Additionally, we employ convolutions with larger kernel sizes (15) compared to those used in previous works (3), which further helps to increase the receptive field.

To compute the final output of the block, a residual connection is included, with a learned scaling coefficient, (denoted g_n in Figure 10). We opt to apply no zero padding throughout the network, which causes the output of each convolution to be smaller than the input. We make this choice since as the depth of the network increases a significant portion of the computations will be computed on the padded part of the activations, which ultimately do not provide additional information, and simply increases the number of computations. To rectify the shape discrepancy between the output and the residual connection, we simply center crop the input activations to the same size as the output of the convolutional layer. In addition to the residual connections at each layer in the network, we also include additive skip connections which aggregate information from each layer at the output of the model.

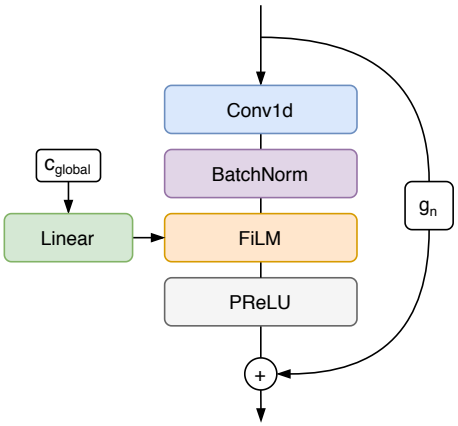

Figure 10: Single convolutional block in the TCN architecture.

An important characteristic of the TCN as implemented in our work, is that the activations throughout the network are never downsampled or pooled along the temporal dimension. As a result, information about the high frequency components of the signal are not lost, as is the case in architectures that employ an autoencoder

Padding	none (valid)
Dilation pattern	1, 2, 4, 8, 16, 32, 64, 128, 256, 512
Kernel width	15
Stride	1
Residual gain init	1.0
Residual gain clamp	[0, 1]

Table 3: TCN model hyperparameters

with downsampling and upsampling pathways (Goodfellow et al., 2016). This choice to avoid temporal downsampling comes as a cost though, both in the memory required to store all of the intermediate activations for backpropagation, as well as in the models inability to operate on the latent representations in a lower dimension space, as is often the main motivation for employing an autoencoder style architecture, as we will see in the following section. Complete details of the TCN based model hyperparameters are shown in Table 3.

4.3.2 Wave-U-Net

In addition to the TCN architecture, we also investigate the Wave-U-Net style architecture, another approach that also operates in time domain, but follows different design principles. Originally introduced in Stoller et al. (2018), it adapted the original U-Net architecture (Ronneberger et al., 2015), for the task of music source separation in the time domain. The concept of the U-Net architecture is to implement an autoencoder, which includes an encoder that downsamples the input progressively through each layer, increasing the depth, ultimately building a lower dimensional representation of the input signal. This lower dimensional representation, or latent representation, can be operated on to produce a modified latent representation. Conceptually, it is often considered "easier" to perform complex operations in this lower dimensional space, compared to the original space, whether that be the pixel space for images, or the sample space for audio signals. With this modified latent representation, it can then be passed to the decoder, which through upsampling, restores the signal to its original dimensionality. The U-Net architecture improves upon this original autoencoder formulation with the introduction of skip connections, like those shown in Figure 11, which shuttle the activations from the encoder at each decimated resolution, to the upsampling layer in the decoder with the corresponding resolution. These connections were shown to greatly speed up convergence, and enable better reconstruction of the desired outputs.

The major contribution of the Wave-U-Net work was adapting this original U-Net architecture to operate on waveforms for the source separation task, including some new approaches to incorporate additional input context, and address the well-known

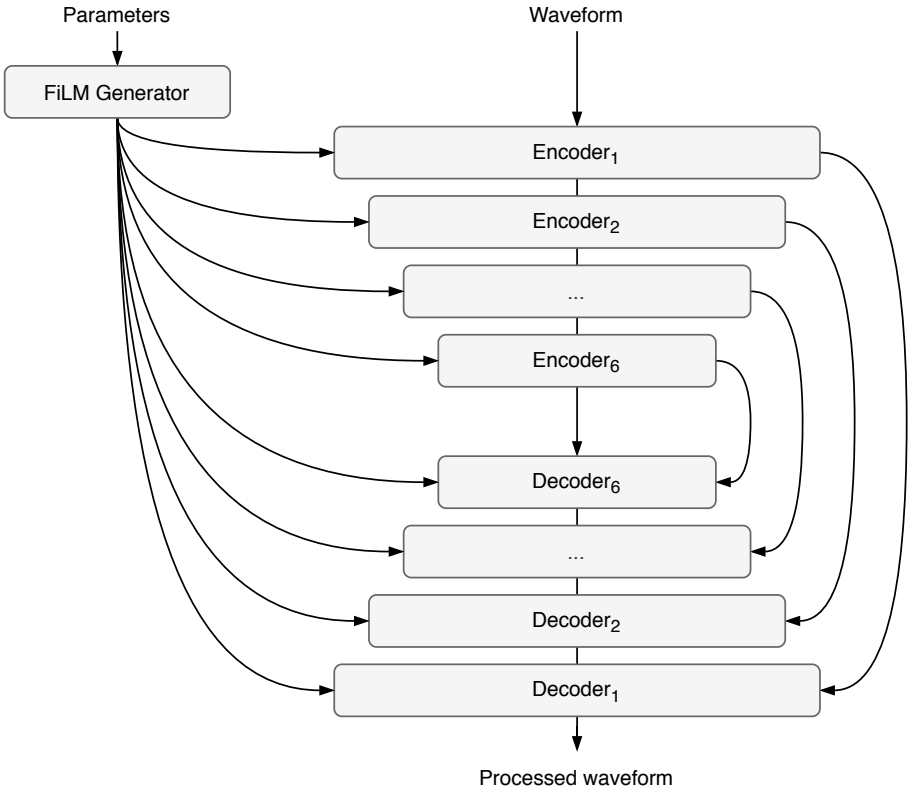

Figure 11: High level overview of the modified Wave-U-Net architecture.

problems with using strided transposed convolutions for upsampling (Odena et al., 2016). Recently, this original formulation was extended with the Demucs architecture by Défossez et al. (2019), which retained the time-domain convolutions and skip connections, but adapted the internal structure of the encoder and decoder for improved performance in the music source separation task.

As in the original Wave-U-Net, the core operation of each block in the encoder is composed of a 1-dimensional convolution. In the encoder, as shown in Figure 12, the activations at the output of this operation are then modulated by the conditional FiLM operation, to be outlined in the following section, and then followed by the PReLU activation. Another convolutional layer is used to double the number of channels, which are then passed to the Gated Linear Unit (GLU) activation (Dauphin et al., 2017), which is a modified version of the gated activations employed in the original WaveNet, and have been shown to bring about better performance in effectively handling contextual information throughout the network. The output of this activation is then saved and passed to the corresponding layer in the decoder.

In the decoder, as shown in Figure 13, we add the corresponding activations from the encoder, and then pass this signal through the first convolution, which again, as in the encoder, doubles the number of channels, before passing the activations to the

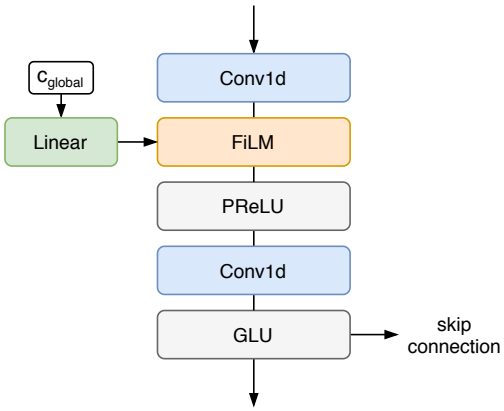

Figure 12: Convolutional encoder block in the Wave-U-Net architecture with Demucs modifications.

GLU operation. We then employ a strided transposed convolution, which upsamples the signal by a factor of 4 at each layer. Full details of the model hyperparameters as shown in Table 4. After this upsampling, but before applying the final PReLU activation, we again apply the feature modulation operation of FiLM. While previous works with autoencoder style architectures found success in using FiLM conditioning only in the encoder (Meseguer-Brocal and Peeters, 2019), we choose to apply FiLM conditioning in both the encoder and the decoder, and find that this provides good results, with limited additional computation cost, since in our application there is no clear intuition about the ideal placement for this conditioning signal. In our informal experiments we tried also placing the FiLM conditioning in only the encoder and only the decoder, but did not observe a clear difference in performance.

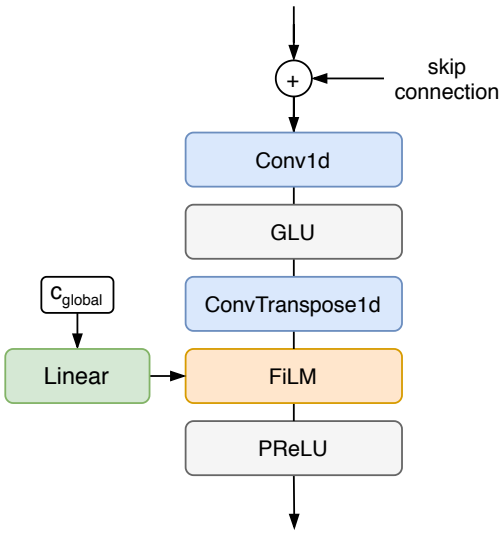

Figure 13: Convolutional decoder block in the Wave-U-Net architecture with Demucs modifications.

Encoder	Kernel width	8
	Stride	4
	No. channels	16
Decoder	Kernel width	8
	Stride	4
	No. channels	16
	Context kernel	3
Weight scaling		0.1
Bottleneck		No LSTM
Upsampling		Transposed Conv.

Table 4: Wave-U-Net-8 model hyperparameters

4.3.3 Conditioning

Since we aim to model the behavior of the audio effect over a continuous range of user-controlled parameters, we can condition the outputs of the model on the values of these parameters. This then easily facilitates dynamic behavior of the feed-forward, deterministic models, with the use of a single model. Traditionally, in time domain audio models like WaveNet (van den Oord et al., 2016a), conditioning is applied by the means of the modified gated activation, originally introduced with PixelCNN (van den Oord et al., 2016b). This activation is given by

$$z = \tanh(W_{f,k} \circledast \mathbf{x} + V_{f,k}^T \mathbf{h}) \odot \sigma(W_{g,k} \circledast \mathbf{x} + V_{g,k}^T \mathbf{h}), \tag{4.11}$$

where $\mathbf{h}$ is a latent vector representing global conditioning (speaker identity for example), $\circledast$ represents the convolution operator, $\odot$ represents the element-wise multiplication operator, $\sigma(\cdot)$ is the sigmoid function, and $W_{f,k}$ and $W_{g,k}$ are learnable filters for the filter, f, and gate, g. Additional linear projections, $V_{f,k}$ and $V_{g,k}$, are applied to the latent vector, $\mathbf{h}$, with the resultant vectors, $V_{f,k}^T\mathbf{h}$ and $V_{g,k}^T\mathbf{h}$, broadcast across the timesteps. This approach was then used by a number of previous TCN-like models (Rethage et al., 2018, Lluís et al., 2019, Damskägg et al., 2019b).

In our work, we employ FiLM, or feature-wise linear modulation, a method of conditioning introduced in (Perez et al., 2018). Inspired by earlier instance normalization techniques (Huang and Belongie, 2017), FiLM formalizes the these techniques as a learned, affine transformation that is performed on intermediate features of a CNN.

We define two functions, which produce outputs, $\gamma_{i,c}$ and $\beta_{i,c}$, respectively, as a function of the input $\mathbf{x}_i$. The FiLM operation, F is then performed on the c^{th} feature at the i^{th} layer of the network,

$$F(\mathbf{z}_{i,c}, \gamma_{i,c}, \beta_{i,c}) = \gamma_{i,c}\mathbf{z}_{i,c} + \beta_{i,c}. \tag{4.12}$$

In practice, the functions are implemented as a single neural network, called the FiLM generator, which generates jointly the vectors $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$. We adjust this formulation with the addition of adaptor layers, where an additional learned, linear transformation is applied to the global conditioning vectors, $(\boldsymbol{\gamma}, \boldsymbol{\beta})$, to adapt the dimensionality and produce a unique affine transformation at each layer in the network given by,

$$F(\mathbf{z}_i, \boldsymbol{\gamma}, \boldsymbol{\beta}) = (\boldsymbol{\gamma} W_{i,\gamma}^T + b_{i,\gamma})\mathbf{z}_i + (\boldsymbol{\beta} W_{i,\beta}^T + b_{i,\beta}). \tag{4.13}$$

In the context of waveform based models, the FiLM operation performs a conditional scaling and shifting of activations without regards for the temporal dimension, hence making this method scalable and memory efficient in fully convolutional networks operating on long sequences, like those in our work. In the case of the audio effect modeling task, the conditioning information passed to the FiLM generator is simply the current parameter values of the effect, which are randomized during training. Since these parameters have varying ranges, each parameter is normalized between -1 and 1 based upon the minimum and maximum possible values. For the case where there are multiple effects present in the signal chain, we concatenate the vector from each effect to create a single vector containing all of the normalized parameters.

4.3.4 Receptive field

It is often useful to quantify the receptive field for convolutional models, which defines the number of input samples used in computing a signal sample at the output of the model. In the case of audio signals, in order to model behavior that requires some long term consistency, for example modeling the reverberation characteristics of the room, the receptive field must be large enough so as to fully capture these time dependencies. In a convoltuional model the receptive field is a function of the kernel size, stride, dilation factor, and the total number of layers in the architecture. It can be solved as a recursion, as the receptive field at each layer in the network is a function of the previous layer, with the receptive field growing as the depth of the network increases. This recurrence equation for the i^{th} layer of the network is

$$r_i = r_{i-1} + d_i \times (s_i \times (k_i - 1)), \tag{4.14}$$

where r_{i-1} is the receptive field of the previous layer, k_i is the kernel size, s_i is the stride, and d_i is the dilation factor. The receptive field of the first layer of the model is the same as the kernel size of that layer.

Both the TCN and Wave-U-Net architectures attempt to construct a model with sufficient receptive field, but they do so using different approaches. In the case of the TCN, we take advantage of exponentially increasing dilation factors to achieve a receptive field that also increases exponentially as the depth is increased. In contrast, the Wave-U-Net architecture uses downsampling or decimation, which is akin to using a strided convolution. This is another efficient method for achieving a large receptive field, but it comes at the cost of needing a powerful enough decoder to reconstruct the original resolution output, whereas the use of dilated convolutions in the TCN achieves a comparable receptive field without the need for upsampling. Both models use relatively large kernel sizes (greater than 3, as is often used in computer vision tasks), in order to increase the effective receptive field.

4.4 Dataset

To train all of our channel-bsaed audio effect models we utilize the unprocessed monophonic audio samples from the SignalTrain LA2A Dataset[4]. We then design an online data generation pipeline that enables us to generate an effectively limitless number of training examples. To achieve this, we construct examples by first randomly sampling small patches (0.37, 0.74, 1.48 seconds depending on the receptive field of the model) of audio from the training set. With this sample, we then apply at random a number of simple augmentations such as scaling the amplitude, inverting the polarity, mixing two samples together, adding white noise, randomly zeroing samples, and randomly zeroing a contiguous block of samples. With this augmented sample, we then set up the signal chain, consisting of the equalizer, compressor, and reverb, by sampling values from a uniform distribution over the range of each of the parameters of each processor. Then we pass this augmented sample through the signal chain to produce the target signal, and create also a conditioning vector of the normalized parameters from the signal chain.

Things are simpler in the case of the of the analog dynamic range compressor, since we are unable to generate our own ground truth targets while training. We utilize the target outputs provided by the SignalTrain LA2A dataset for training, instead of generating our own on-the-fly. This means we are limited to roughly 20 hours of training data, 4 hours of validation data, and 0.7 hours of testing data, with different processor parameterizations split across the three sets.

To address the limited amount of training data for this case, as well as the challenging split of the train and test splits, in which there is no overlap in the parameterization of the compressor, we employ the *mixup* technique (Zhang et al., 2018a) during training. This is a simple technique, to generate new input/output pairs $(\tilde{x}, \tilde{y})$,

[4]https://zenodo.org/record/3824876

$$\tilde{x} = \lambda x_i + (1 - \lambda) x_j, \tag{4.15}$$

$$\tilde{y} = \lambda y_i + (1 - \lambda) y_j, \tag{4.16}$$

by taking a linear combination of two random input/output pairs from the training set, (x_i, y_i) and (x_j, y_j). This linear combination involves scaling the input and output examples by a factor, $\lambda \in [0, 1]$, which we sample from a beta distribution (generally with $\beta = \alpha \leq 0.4$). This process is intended to encourage the model to better interpolate between training examples, and hence improve generalization.

4.5 Training

In the training process, optimization of all models is performed using the Adam optimizer (Kingma and Ba, 2015) with a learning rate of 3e−4. Simple plateau learning rate scheduling is employed, which halves the learning rate when the validation loss has not improved for 15 epochs. Additionally, we employ early stopping, which completes the training process after the validation loss has not improved for 40 epochs. Full details on the training runs for each model are shown in Table 8.

With regards to the loss function, we utilize the mean absolute error (MAE) or L1 loss in training all of the audio effect models. This loss is defined as the absolute difference in the sample space between the predicted and ground truth signals, as shown in Equation 4.17. Since this loss attends closely to the exact sample-by-sample structure of the ground truth, the calculated error may not align with perception. For this reason, the MAE or L1 distance is often considered a "phase" loss. Since we are largely interested in modeling these effects as accurately as possible, such a loss function, even if it does not completely agree with perceptual tolerances, seems to be a good choice.

$$\ell_{\text{MAE}}(\hat{y}, y) = \frac{1}{n-1} \sum_{i=0}^{n-1} |\hat{y}_i - y_i| \tag{4.17}$$

4.6 Evaluation

For our evaluation we use audio samples from the test set of the SignalTrain LA2A dataset, which is comprised of audio samples that have not been seen during training. Similarly to the generation process employed during training, we randomly sample parameterizations of the signal chain from a uniform distribution, to most effectively evaluate the performance across the range of possible configurations. We

then compute the following metrics, comparing the outputs of the models to the generated ground truth signals.

4.6.1 Metrics

In order to evaluate the performance of models in their task of emulating audio effects we desire to utilize objective metrics that enable a method for determining how closely a model is able to capture the behavior of the original processor or chain of processors. An important consideration for all objective metrics employed in the evaluation of signals that are perceived by humans, such as audio, is to understand the perceptual relevance of these metrics.

Due to the realities of human perception, subtle changes in the underlying signal may be completely undetectable by listeners. This causes a problem, as it is often the case that objective metrics, ones that compare two signals by performing an operation to measure their distance, will penalize subtle differences in the signals that would be otherwise imperceptible to listeners. Since our goal is to produce audio signals that will be auralized and consumed by listeners, we are ultimately not concerned with the absolute performance, but only to what degree is perceptible. The most effective manner in performing this kind of evaluation is to ask listeners to manually listen and evaluate content to compare the results, but this process is time consuming and expensive. Therefore, we utilize a collection of objective metrics to provide a general impression about the performance of our systems.

Mean absolute error

The mean absolute error (MAE) is the absolute value of the distance in the sample space between the predicted and ground-truth signal, as introduced previously in Equation 4.17. Since this error is computed in the time-domain it enforces strict adherence to the not only the magnitude response, but also the phase of the ground-truth signal, which may attend to differences that are not perceptual. While this metric provides some information about the performance with regards to directly matching the ground truth signal, it is known to not agree with perception, and in our informal listening, we often found outputs with lower MAE that sounded worse than other outputs with higher MAE. For that reason, we can use this measure only as a single metric among many in our evaluation.

Error-to-signal ratio

The error-to-signal ratio (ESR) is a straightforward extension of the mean squared error (MSE), where the squared error is computed between each sample value in the sequence, as shown in Equation 4.18. The ESR adjusts this error by normalizing

by the sum of the square of the ground truth signal, which represents its energy, as shown in Equation 4.19. This metric therefore attempts to penalize more equally errors when the signal is lower in absolute amplitude. Previous works focused on modeling audio effects have also employed the ESR as both a loss function and evaluation metric (Damskägg et al., 2019b). Nevertheless, since this metric attends also to a distance in the sample space, it is likely to have distances that do not agree with perception.

$$\ell_{\text{MSE}}(\hat{y}, y) = \frac{1}{n-1} \sum_{i=0}^{n-1} |\hat{y}_i - y_i|^2 \tag{4.18}$$

$$\ell_{\text{ESR}}(\hat{y}, y) = \frac{1}{n-1} \frac{\sum_{i=0}^{n-1} |\hat{y}_i - y_i|^2}{\sum_{i=0}^{n-1} |y_i|^2} \tag{4.19}$$

Mel cepstral distortion

The mel cepstral distortion (MCD) was first introduced for the evaluation of speech signals produced by text-to-speech systems (Kubichek, 1993). This metric differs from the previous sample-based metrics, since this operation is performed on features extracted from the signals in the frequency domain, instead of the time domain. Frequency domain metrics are often considered to be more perceptually relevant, as they often attend to differences in the allocation of energy across the frequency spectrum, and hence are less likely to penalize small differences in the time domain signal. This is largely a result of using the magnitude of the frequency domain representation, and ignoring the differences in the phase components of the signals.

To compute the MCD, we first must compute the mel-frequency cepstrum coeffients (MFCCs). This requires that we start by computing the STFT across both signals, and then map the power across the frequency spectrum onto the mel scale, and further log scale these values. Then we take the discrete cosine transform (DCT) of these mel scaled log powers, as if it was a signal, which will produce the (MFCCs). Once the MFCCs have been computed for the ground truth and predicted signal, we select N of these coefficients, and then measure the distance between the two signals in this MFCC space. The specific implementation we employ follows that of the method from Sahidullah and Saha (2012), which follows an earlier implementation from Kominek et al. (2008). This metric is computed between the target signal, C, and the predicted signal, $\hat{C}$, with $k-1$ coefficients and $n-1$ MFCC frames, as shown in Equation 4.20. Additionally, this metric includes a scaling factor α, shown in Equation 4.21 and based upon Mashimo et al. (2001), which has be retained largely for historical reasons. In our evaluation, all values are reported with $k = 25$.

$$\ell_{\text{MCD}}(\hat{C}, C) = \frac{\alpha}{n-1} \sum_{i=0}^{n-1} \sqrt{\sum_{j=0}^{k-1} (\hat{C}_{i,k} - C_{i,k})^2} \tag{4.20}$$

$$\alpha = \frac{10\sqrt{2}}{\ln 10} \approx 6.14185 \tag{4.21}$$

Multi-resolution STFT

The multi-resolution STFT is a simple extension of the STFT, where we compute STFT frames with multiple different frame sizes across the entire signal (Yamamoto et al., 2020). The use of multiple frame sizes is motivated by the inherent trade-off between greater resolution in the time domain and greater resolution in the frequency domain, as a function of the frame size. By utilizing multiple frame sizes and aggregating the information across all resolutions, we can capture a more realistic representation of the signal. This measure is conceptually similar to the MCD, but provides greater detail by directly utilizing the STFT frames, instead of computing the MFCCs as intermediate features.

$$\ell_{\text{sc}}(\hat{y}, y) = \frac{\| |\text{STFT}(y)| - |\text{STFT}(\hat{y})| \|_F}{\| |\text{STFT}(y)| \|_F} \tag{4.22}$$

$$\ell_{\text{mag}}(\hat{y}, y) = \frac{1}{N} \| log(|\text{STFT}(y)|) - log(|\text{STFT}(\hat{y})|) \|_1 \tag{4.23}$$

$$\ell_{\text{MR-STFT}}(\hat{y}, y) = \frac{1}{M} \sum_{m=1}^{M} \ell_{\text{sc}}(\hat{y}, y) + \ell_{\text{mag}}(\hat{y}, y) \tag{4.24}$$

The multi-resolution STFT is computed by first computing multiple STFT frames for the ground truth signal y, and the predicted signal $\hat{y}$ with varying window and FFT sizes. We then compute two intermediate measures, first the spectral convergence, shown in Equation 4.22, where $||\cdot||_F$ is the Frobenius norm, and second the log magnitude error, shown in Equation 4.23, where N represents the number of STFT frames, M represents the number of different FFT sizes, and $||\cdot||_1$ is the L_1 norm. Finally, we compute the complete multi-resolution STFT error, shown in Equation 4.24, by averaging the spectral convergence and log magnitude errors across all of the different M frame sizes. The utilized frame sizes, along with the related hop size and window lengths, are shown in Table 5. These values are adopted from Yamamoto et al. (2020) and Yang et al. (2020). The optimal number of bands, and their parameters, for a particular task remains an open research question.

Band	FFT size	Hop size	Window length
1	512	50	240
2	1024	120	600
3	2048	240	120

Table 5: Multi-resolution STFT bands

4.6.2 Channel signal chain

To begin our evaluation, we investigate the performance of our TCN and Wave-U-Net based models on the channel modeling task. In this task, the model is passed an audio signal, as well as randomly generated parameters for the channel processors. All of the audio used for this evaluation has not previously been seen during training, and is sourced from the test set of the SignalTrain LA-2A dataset. The parameter values for the processors are sampled randomly from the same distribution and ranges as those used during training. We use the same set of audio examples and randomly sampled parameters to evaluate each model, and we report the four metrics introduced in the section above, as shown in Table 6.

Model	Params	R.f (ms)	Task	MAE	ESR	MCD	MR-STFT
TCN-10	290k	324.8	equalizer	0.015	0.156	577.8	2.449
			Compressor	0.005	0.100	545.5	1.895
			Reverb	0.021	0.129	443.9	1.931
			Combined	0.035	0.430	592.2	2.701
TCN-20	528k	649.5	equalizer	0.017	0.189	597.3	2.608
			Compressor	0.006	0.140	618.1	2.123
			Reverb	0.011	0.038	309.2	1.340
			Combined	0.027	0.292	**495.6**	2.315
TCN-30	764k	974.3	equalizer	0.017	0.203	578.8	2.668
			Compressor	0.006	0.132	544.1	2.075
			Reverb	0.011	0.041	253.1	1.212
			Combined	**0.024**	**0.252**	496.6	**2.210**
Wave-U-Net-6-8	23M	216.6	equalizer	0.009	0.140	592.7	2.417
			Compressor	0.004	0.102	762.6	2.319
			Reverb	0.020	0.144	1037.2	3.808
			Combined	0.026	0.305	933.7	3.995
Wave-U-Net-6-16	9M	464.3	equalizer	0.009	0.206	633.0	3.025
			Compressor	0.004	0.108	712.2	2.637
			Reverb	0.034	0.378	1117.1	4.746
			Combined	0.038	0.481	1026.2	4.890

Table 6: Overview of results for modeling the channel effects (static order) with the TCN and Wave-U-Net architecture across varying receptive fields.

In order to better understand the performance of the models across the entire channel modeling task, which involves modeling three different processors in a series connection, we have four test cases for each model. The first three test cases involve sampling parameters such that only one of the three processors, equalizer, compressor, and reverb, are active at any given time, such that the other two processors are effectively bypassed. This helps to demonstrate the performance of each model for the different processors, each which present their own challenge. The fourth test case involves sampling parameter values such that all three processors are active at the same time. This presents the most challenging case, and is the case that we are most concerned with, as it gives an indication of how the model might perform when employed later in our mixing task. For this reason, we mark in bold the model with the lowest error for the Combined task across each metric.

In addition to the evaluation across all the models, we are also interested in the potential for this model to capture the behavior of the signal chain, when the ordering of the processors in the chain is another parameter. We are interested in this task, as it would potentially provide the ability for the controller network to modify the order of the processors in the signal chain when determining the optimal mix. While the order of elements in the signal chain may not always be critical, due to nonlinear processors like the compressor, there are cases wherein the desired behavior can only be achieved by ordering the processors in a specific configuration.

Model	Proc. ordering	Params	Cond. size	Task	MAE	ESR	MCD	STFT
TCN-10	Static	290k	25	Equalizer	0.015	0.156	577.8	2.449
				Compressor	0.005	0.100	545.5	1.895
				Reverb	0.021	0.129	443.9	1.931
				Combined	0.035	0.430	592.2	2.701
	Variable	291k	31	Equalizer	0.022	0.316	725.5	3.142
				Compressor	0.010	0.435	716.8	2.684
				Reverb	0.024	0.171	494.3	2.241
				Combined	0.060	3.846	785.5	4.917

Table 7: Overview of results with the TCN-10 model for the static or dynamic ordering of the processors in the channel.

For this evaluation we follow the same methodology as above. This time we compare the performance of the TCN-10 model with another TCN-10 model, with the difference that it has been trained with examples that involve not just a static order of the processors, but a dynamic ordering, where the ordering of the processors is also passed as conditioning information to the model. Given that there are 3 processors in the signal chain, there are then $N = 3! = 6$ permutations of these processors, so we simply concatenate a one-hot encoded vector with these 6 values at the end of the original parameter values.

We report the results of this evaluation in Table 7 following the same structure as before. Here we see that across all the metrics, the model with dynamic ordering performs worse than the model with a static ordering, as we would expect, given that both models have nearly the same capacity. It appears that in the single processor cases the performance of the dynamic ordering model is more comparable, albeit still less performant, yet in the combined case there is an even larger disparity.

These results agree with our informal listening, which showed an inability for the dynamic ordering model to accurately capture the underlying processing of the signal chain for some more challenging parameter configurations. For simpler cases, the model was able to generate plausible results. It is possible that increasing the model capacity would improve performance, as these results indicate that the model is capable of capturing some aspects of the dynamic ordering case. Nevertheless, these results indicate that it is in fact possible for the model to capture details about the ordering of the processors in the signal, although it presents a more significant challenge than the fixed ordering case.

Model	Steps	Time (hr.)	Time (days)
TCN-10	235	87	3.6
TCN-10 variable order	116	32	1.3
TCN-20	310	186	7.8
TCN-30	290	262	10.9
Wave-U-Net-8	457	46	1.9

Table 8: Channel based effects training runs details.
(All runs use one NVIDIA RTX 2080 Ti)

4.6.3 Analog dynamic range compressor

In addition to the the complete channel modeling task, we also examine the performance of a number of models on the analog dynamic range compressor modeling task, with the results shown in Table 9. We perform the evaluation using examples from the test set of the original SignalTrain LA-2A dataset, and compare the performance across the original implementation presented in Hawley et al. (2019), along with 8 our own models. This set contains settings of the compressor that were not seen during training of any of the models, and therefore requires the models to have sufficient ability to interpolate between the training examples seen during training.

In the first section we report the same metrics using the implementation[5] provided by the authors of Hawley et al. (2019). We do not retrain their model, but simply use a checkpoint provided by the authors of their model for this task. The second

[5]https://github.com/drscotthawley/signaltrain

Model	Params	R.f (ms)	MAE	ESR	MCD	MR-STFT
SignalTrain	4M	5015.5	7.97e-3 (2.66e-3)	0.023 (0.009)	809.6 (219.7)	1.657 (0.216)
TCN-8	167k	80.0	8.11e-3 (2.49e-3)	0.024 (0.010)	392.9 (87.6)	0.808 (0.104)
TCN-10	202k	324.8	4.95e-3 (1.05e-3)	0.012 (0.007)	339.4 (74.8)	0.754 (0.073)
TCN-10-mixup	202k	324.8	4.60e-3 (1.23e-3)	0.009 (0.006)	350.2 (99.1)	0.748 (0.100)
TCN-20-mixup	377k	649.5	**4.24e-3** (1.36e-3)	**0.008** (0.004)	263.0 (64.6)	0.606 (0.066)
TCN-30-mixup	552k	974.3	5.53e-3 (1.80e-3)	0.013 (0.006)	297.1 (78.2)	0.679 (0.083)
Wave-U-Net-6-8	5M	216.6	1.30e-2 (6.39e-3)	0.157 (0.100)	260.0 (158.1)	**0.502** (0.079)
Wave-U-Net-6-16	8M	464.3	1.13e-2 (6.43e-3)	0.046 (0.029)	**259.5** (165.5)	0.599 (0.416)
Wave-U-Net-6-16-mixup	8M	464.3	1.14e-2 (5.32e-3)	0.084 (0.057)	455.3 (232.2)	0.987 (0.206)

Table 9: Results for modeling the analog LA-2A compressor on the SignalTrain test set. Standard deviation across all of the test set examples is shown in parentheses.

section shows the results from the TCN based models, with varying stacks of dilated convolutions, as well as models trained with and without the *mixup* technique. The final section shows a few different Wave-U-Net based models. Early on in our investigations, we found that the results from the TCN based models sounded significantly better than the Wave-U-Net models, so for much of our work we focused more of our effort on evaluating the TCN based models, and hence the evaluation of different Wave-U-Net models is less exhaustive.

As expected, there is a bit of disagreement between the different metrics. We find that the time domain based metrics (MAE and ESR) tend to agree with each other, as do the frequency domain metrics (MCD and MR-STFT), but these groups often do not agree with one another. It is clear that the TCN based models perform the best with respect to the time domain metrics, but the Wave-U-Net models appear to the surpass the the best TCN model in the MCD and MR-STFT. Although, it should be noted that these differences in the frequency domain errors may not be significant, since the standard deviation values show that the TCN-20-mixup and Wave-U-Net-6-16 model are within one standard deviation. For this reason, we conclude that the TCN-20-mixup model is the best performing model overall, which agrees with our informal listening. We found that while the Wave-U-Net models match the underlying signal, they tend to impose high frequency ringing artifacts, which we posit may not be adequately penalized in the frequency domain based error metrics. In the case of the SignalTrain model, it falls behind in both the time domain and frequency domain metrics, and this result is confirmed by our information listening, with the outputs from this model containing a significant amount of consistent broadband noise in nearly all the test examples.

We first investigated the performance of the TCN-10 with and without *mixup*, since this model was faster to train than the larger models, and since we found that *mixup* improved slightly the performance without additional cost, we opted to use *mixup* in training the remaining TCN models. While we did see some performance gains

Model	Steps	Time (hr.)	Time (days)
TCN-8	106	27	1.1
TCN-10	102	29	1.1
TCN-10-mixup	218	81	3.4
TCN-20-mixup	216	112	4.7
TCN-30-mixup	241	145	6.0
Wave-U-Net-8	343	8	0.3
Wave-U-Net-16	557	15	0.6
Wave-U-Net-16-mixup	151	5	0.2

Table 10: Analog compressor training runs details. (All runs use one NVIDIA RTX 2080 Ti)

when using *mixup*, its not clear that there was a significant improvement based upon the standard deviation values. Further work on tuning the α hyperparameter of this technique may bring about more insight into the efficacy of this method.

We found that the TCN based models with larger receptive field performed the best, with performance degrading only once the receptive field approached 1 second. We noted that the training loss was noisier when training the TCN-30-mixup model and this could have contributed the early stopping ending the training before the model had fulled converged, even though this model trained for longest as shown in Table 10. There is some indication that at some point, further increases to the receptive field beyond that required to model most of the compressor's behavior makes training more unstable, since the TCN-30 model appears to perform somewhat worse than the TCN-20, although the standard deviation values indicate this difference may not be significant. Overall, it is clear that both the Wave-U-Net and the TCN models are capable of modeling the nonlinear behavior of the LA-2A compressor using the limited training data available. Not only are these models proficient at this task, they surpass the performance of the previously proposed SignalTrain model.

4.7 Summary

Through our evaluations we demonstrated that both the Wave-U-Net and TCN based models were able to capture the behavior of the signal chain containing a series connection of signal processors. In addition, we also found that the these models were capable of modeling some aspects of the dynamic ordering of the processors in the signal chain, albeit it is clear this is a more difficult task than with a static ordering. We also extended our results beyond the channel case, which modeled a connection of digital processors, with a nearly limitless number of training examples, to the task of modeling an analog dynamic range compressor, with a limited number of training examples. In these experiments we demonstrated that our proposed models not only performed well on this task, but also exceeded the current

state-of-the-art approach in modeling this compressor. Ultimately, we extended the performance of previous neural audio effects implementations, and now intend to apply these findings in the design of our differentiable mixing console, in an effort to build a model that is able to learn to perform multitrack mixing directly at the waveform level.

Chapter 5

Mixing system implementation

In this chapter we will apply the findings from our work on neural audio effects in order to construct a system that is capable of learning directly from the waveforms of multitrack mixes, without the need for the original mix parameters. In addition, we aim to address many of the challenges in this mixing task, such as the large variance in the number of input recordings, the inability to define a holistic taxonomy of input classes, permutation invariance with respect the input recordings, as well as the ability to learn to utilize the stereo field with a specialized loss function.

5.1 Baselines

5.1.1 Deep learning baseline

The deep learning baseline we introduce here represents a canonical approach for processing audio in the time domain, based upon the Wave-U-Net architecture (Stoller et al., 2018). In this formulation, the input to the model is a two dimensional tensor, with each input source stacked along the channel dimension. The output of the model is a two dimensional tensor with two channels, representing the left and right signals. We adapt our earlier implementation of the Wave-U-Net architecture with Demucs modifications for the mixing task by extending the number of input channels from one to eight. In our experiments we also greatly increase the capacity of the model by adding additional convolutional filters at each layer.

This formulation presents a number of challenges. Since the model directly predicts the stereo mix, it is not possible to directly inspect the processing applied to each input source. Therefore, it does not provide a method for adjusting the resultant mixture, such a adjusting the relative levels, or processing. In addition this model cannot adapt to more channels than the number set at training time. While fewer

channels can be used during inference by filling unused input channels with zeros, this will still impose the same number of operations regardless of the number of input recordings. While this may be relatively insignificant when the maximum number of tracks is small and there is little variance in the number of tracks across the dataset, most real-world datasets contain songs with a wide range of input recordings, from 1 to over 100, making this formulation highly inefficient. Furthermore, it is likely that this model will struggle to handle learning from examples that have varied instruments that do not follow a consistent input ordering or structure.

5.1.2 Naive baseline

To further aid in our evaluation, we also would like to compare the performance of these deep learning methods to more conservative and simplistic baselines. For this purpose we employ two different baseline methods for producing a mix given a set of input recordings. The first is a monophonic sum of the input recordings, which is potentially the most conservative method for generating a mix. This method is likely to produce a mix in which many of the elements are audible and it is unlikely to produce artifacts, but the resultant mix will contain no stereo content, and the mix may appear very cluttered for cases in which there are a large number input recordings. The second method is a randomized gain and panning mix, where each input recording is first scaled by a random gain (bounded between -12 dB and +12 dB), and then panned randomly across the stereo field.

In informal listening tests, we found that the random baseline often produced more desireable mixes than those from the mono baseline when the number of input recordings was small ($\leq$ 8 inputs). This is likely due to the reality that panning sources across the stereo field reduces potential masking interactions, and helps to increase intelligibility of elements within the mix. We also observed that this method occasionally produced lower quality mixes in cases where critical elements, such as the vocals, were panned heavily to one side. By performing formal evaluations of the deep learning methods compared against both of these baseline methods, we hope to gain a better picture of the relative performance of these methods.

5.2 Differentiable mixing console

5.2.1 Architecture

As addressed in Chapter 3, we introduce the differentiable mixing console (DMC), which aims to address many of the shortcomings of the deep learning baseline. The underlying challenge in the multitrack mixing task comes from the requirement that the model must operate on a group of signals with complex interactions.

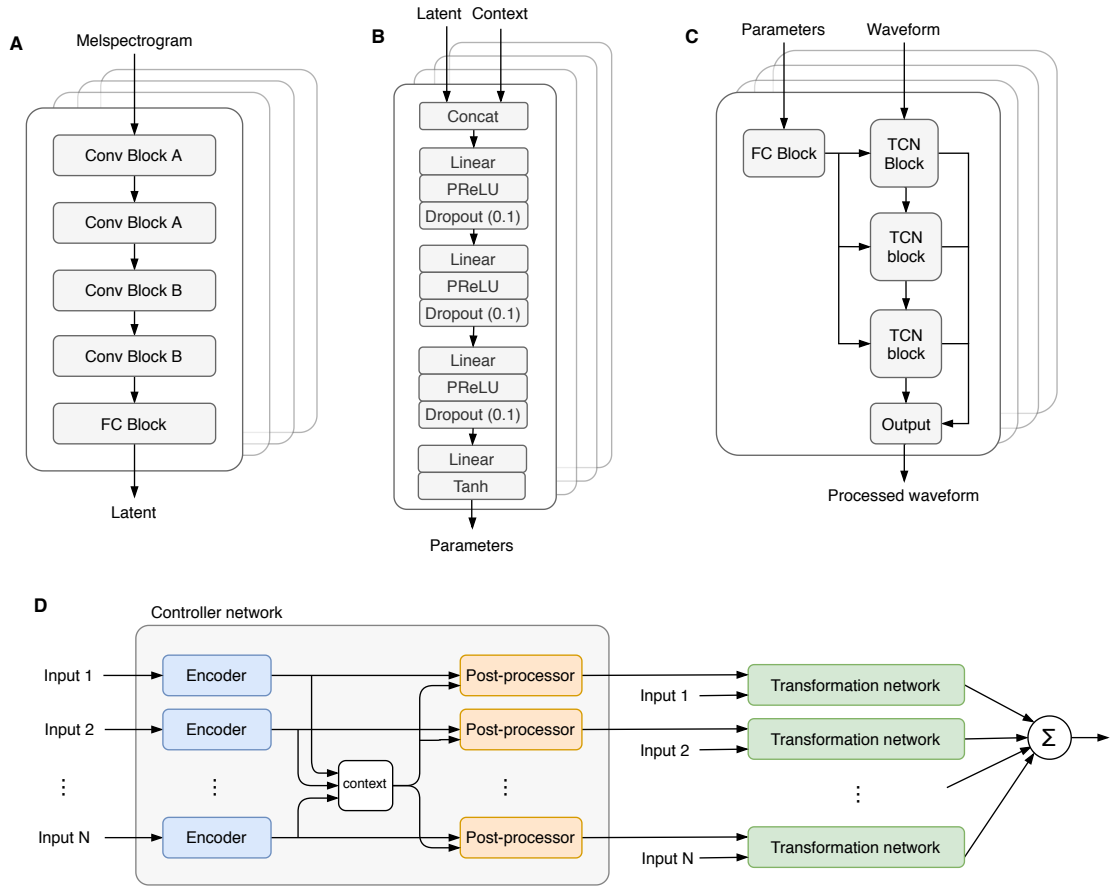

Figure 14: (**A**) The pre-trained CNN encoder architecture operating on melspectrograms following the VGGish implementation. (**B**) The post-processor network, which is a simple MLP that takes the latent vector from the current channel along with the context vector. (**C**) The pre-trained TCN based transformation network that processors waveforms given a set of appropriate parameters. (**D**) The complete architecture showing the interconnection of all the subsystems.

To overcome these challenges, we propose the following key aspects in the design of our the architecture:

(*i*) Weight sharing across all input channels.

(*ii*) Specialized pre-training of the encoder and transformation networks.

(*iii*) The use of context embeddings capturing information about all input channels.

These design characteristics are shown in Figure 14, which outlines the inner structure of the three core subsystems (**A** - encoder, **B** - post-processor, and **C** - transformation network), as well as their connection within the complete system (**D**).

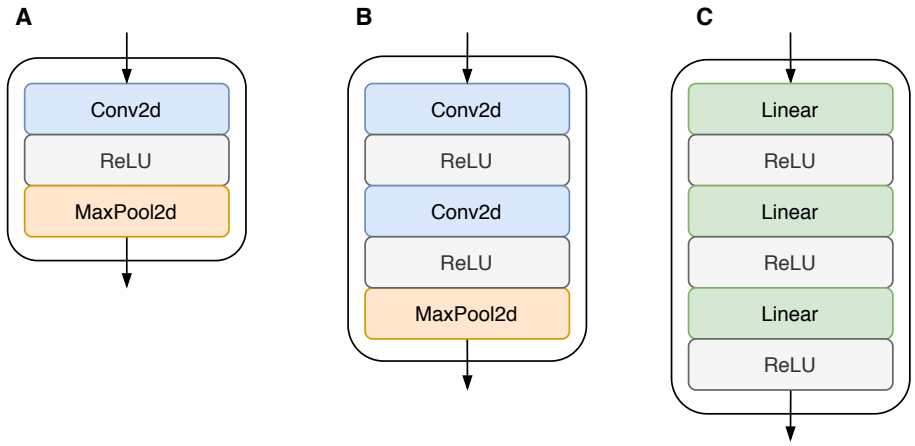

Figure 15: Composition of the blocks within the VGGish (Hershey et al., 2017) architecture. (**A**) Conv block A, with a single 2D convolution followed by max pooling. (**B**) Conv block B, with two convolutions followed by max pooling. (**C**) FC block, with a series of linear layers and ReLU activations.

Encoder

As previously introduced, the role of the encoder is to extract useful information from the input recordings. This information will then be passed on to the post-processor in order to make a judgement about how to set the parameters of the transformation network for the input in question. In our implementation, we apply the same encoder to each input recording to produce a separate latent representation, which will then be passed on to the post-processor.

To design this network, we adopt the popular VGGish CNN architecture from Hershey et al. (2017), that has been demonstrated to be successful on audio classification tasks. As shown in part **A** of Figure 14, the VGGish architecture takes as input a log power melspectrogram representation of the audio signal, which is first downsampled to 16 kHz. The network is then composed of a series of convolutional blocks which employ max pooling in order to downsample further along the time frequency axes, while the depth is increased with the growing number of convolutional kernels at each layer. There are two different arrangement of the convolutional blocks, which are shown in Figure 15. After the final convolutional block, the resultant activations are passed to a fully connected block, which features a series of linear layers and ReLU activation functions, ultimately producing a 128 dimensional vector for each one second of input audio. Since the model can receive input frames of arbitrary length, we apply mean pooling to produce a single 128 dimensional embeddings for each input recording.

In order to strengthen the inductive bias of the model, and facilitate the training of the post-processor network, we utilize the pre-trained weights provided from

their original implementation[1], which has been trained on AudioSet, an expansive dataset that sources 10 second audio clips from YouTube videos to create and ontology of 632 audio event classes with over 2 million human annotations (Gemmeke et al., 2017). We found also that fine-tuning these weights during the training of the post-processor helped to improve performance, enabling the encoder to learn representations optimized for the different instruments seen during training. During training, we use input patches of 5 seconds of audio to provide sufficient context about the mix, in order to make better mixing predictions. Informally, we found in our pre-analysis that patches of less than 5 seconds provided a challenge in generalizing to the examples in the validation set.

Post-processor network

The post-processor forms the core decision making subsystem of the mixing system. Its role is to analyze the information provided by the encoder in order to make choices about how the respective transformation network should operate on the input. It is important to note that the decision making process in setting the parameters for a single channel is dependant upon the content of the input as well as the content of all the other inputs. This presents a challenge in our formulation, in which we share weights across the inputs, so that we decouple any processing across the input channels. So to address this we introduce the context embedding, which is created by computing the mean embedding across all of the inputs. Then when the post-processor is tasked with predicting a set of parameters for a given input, it will look both at the individual embedding for the respective input, along with the context embedding that contains information about all of the input recordings.

While it may appear as if the mean embedding does not capture sufficient information about the other recordings, recall that we also fine-tune the weights of the encoder during the training of the post-processor, hence the encoder additionally has the ability to adjust its behavior such as to maximize the information presented by this context embedding. During training, the input to the controller network is the simple concatenation of the input latent representation with the content embedding, which results in a 256 dimensional vector.

The network structure of the post-processor is a straightforward fully connected network, as shown in part **B** of Figure 14. It is composed of three hidden layers with 2048 units, PReLU activations, and Dropout with $p = 0.1$ (Srivastava et al., 2014). This is followed by a output layer that maps the final activations to the 25 parameters of the full channel. In the case of the DMC-basic model, where only the gain and panning is predicted, this model will produce just 2 outputs. A tanh

[1] https://github.com/tensorflow/models/tree/master/research/audioset/vggish

activation is used at the output to bound the parameters between -1 and 1, which correspond to the scaling that was used to normalize the parameters when training the transformation network.

Transformation network

Based on our findings from Chapter 4, we choose to use the TCN-20 architecture as the transformation network. The pre-trained weights from the channel modeling task are then used in this complete mixing system, and due to the high memory overhead required for using this model during backpropagation, we freeze the weights, and instead only update the weights of the controller network.

In order to condition each TCN model, the output from the post-procesor, which generates the normalized parameters, is connected to the FiLM generator. Then the waveform from the respective input is passed to the TCN as well. In our implementation, we perform the processing of an N channel mix by placing each input recording along the batch dimension. This enables us to produce all of the processed inputs in parallel, which are simply summed to create the final mix. In order to simulate a larger batch size to improve the estimate of the gradient during training, we employ gradient accumulation, where we simply perform multiple forward passes through the network with different training examples, before performing backpropagation.

We also found it useful to design two different versions of the DMC model. The first we call DMC-basic, where the controller network produces only gain and panning values, which are directly applied to the input recordings in order to create a mix. This enables us to bypass the TCN, and hence save a significant amount of memory and processing time during training. The DMC-basic model then enables the training of models with a larger number of inputs, and we train a model using all of the samples in MedleyDB that have 16 or less input recordings.

The second model we call DMC-full, and it features the complete transformation network, with gain, polarity inversion, panning, and the three processor of the channel (EQ, compression, and reverb). Due to memory constrains imposed by the TCN-20 model, we were restricted in the number of input recordings we could use during training, so we train using samples in MedleyDB that have 5 or less input recordings. The memory impact of the TCN model is clearly a limiting factor in the scalability of this complete architecture design, but we leave the optimization of this implementation to future work. In both DMC-basic and DMC-full the controller network is identical, only the presence of the TCN model is changed.

5.3 Datasets

To train the controller network we require a dataset comprised of both the original, individual input recordings, along with a mix of these tracks produced by a skilled audio engineer. As mentioned previously, there are no standardized datasets specifically for the automated mixing task, but other datasets have been developed in the study of other MIR tasks, such as MedleyDB, or for general study of mixing practice, such as the Open Multitrack Testbed. In our work we employ two different datasets in order to investigate the ability of our proposed model to handle both a simpler mixing task, involving just elements from the drum kit, as well as a more realistic task involving mixing a larger number of input recordings with no pre-defined taxonomy across a range of different genres and styles.

5.3.1 ENST-drums

The ENST-drums dataset is composed of a number of professionally recorded drum passages, in a number of different styles, as performed by three different drummers (Gillet and Richard, 2006). Each example contains the input sources from eight different microphones placed around the drum kit, as well as two mixes made by an engineer, one that includes only setting the gain and panning of these microphones, as well as mix that includes gain setting, panning, equalization, compression, and reverberation. While this dataset was originally proposed for applications in automatic drum transcription and processing, since it includes both the original, unprocessed input recordings, as well as mixes, we aim to use it for training the controller network in the task of automated multitrack mixing. The complete dataset contains around 3 hours of recordings, with each of the three drummers comprising about 1 hour in length. The configuration of the microphones used in each recording is consistent, including the snare, kick drum, left overhead, right overhead, hi-hat, high tom, mid tom, and optionally a third tom.

This dataset has been used previously for the automatic mixing task (Moffat and Sandler, 2019a), but they considered only gain setting of the input recordings to create mono mixes. We create a training (80%), validation (10%), and test split (10%), with each set including passages from all of the drummers, which we believe follows the approach taken by previous works. Additionally, we question the validity of the drum mixing task, as it may not be an ecologically valid task. It is not common for an audio engineer to be tasked with mixing the element of the drum kit on their own, and these drum recordings will nearly always be accompanied by other elements of the complete song, which will have an impact on the decision that the mixing engineer makes in the process of the create the final mix. Nevertheless, we investigate this basic mixing task as a manner to test the operation of our system.

5.3.2 MedleyDB

On the other hand, MedleyDB (Bittner et al., 2014, 2016) features a significantly more diverse and more realistic multitrack mixing task. In total, the dataset features 196 songs with around 7 hours of recordings, with a wide range of instrument types, styles, and number of input recordings. While this dataset was originally designed for use in classic MIR tasks, such as melodic analysis, instrument identification, and genre classification, we propose to use it for the multitrack mixing task, as it includes complete unprocessed input recordings, as well as stereo mixes of all tracks in the dataset. To our knowledge, MedleyDB has not been applied to multitrack mixing task, and we present this dataset as a challenging task representative of many real-world use cases.

5.4 Stereo loss function

With the use of the L1 loss in time or the multi-resolution STFT loss, we found that on the more challenging mixing task of MedleyDB dataset, the models tended to always pan all elements of the mix to the center. Curiously, this was not a problem in the case of the ENST-drums dataset. The major difference between these datasets is that the ENST-drums dataset contains mixes all produced by the same engineer, and all of the passages contain an identical input layout (kick, snare, hi-hat, etc.). What we realized was that in the case of the ENST-drums dataset, the model was able to take advantage of the identity of the source (e.g. hi-hat), in order to determine the proper panning, since across the dataset, nearly every passage had each element of the drum kit panned to the same position.

In the case of the MedleyDB dataset, this breaks down completely. Since each song is mixed by different engineers and is composed of different elements, there is no consistent structure to the panning of the elements. For example, in one mix the acoustic guitar may be panned strongly to the left, but in another mix the acoustic guitar is panned strongly to the right. Therefore, when the model is tasked with predicting the panning location of an acoustic guitar, it cannot know a priori, the panning location of this element in the ground truth mix. This means that using a traditional distance based metric on the stereo signal, like the L1 in time of multi-resolution STFT loss, the model will be encouraged to place the element in the center, since this minimizes the error when there is uncertainty. Unfortunately, this behavior is highly undesirable, While the model has minimized its risk, we know that panning elements to the center is quite perceptually distant from the alternative, which would be panning the element to the opposite side than the side in the ground truth mix. This finding motivated the design of a specialized loss function that provides invariance to the absolute orientation of the stereo field, but

maintains the overall stereo balance of the elements in the mix.

To address this we introduce the stereo loss function, based upon the previously introduced multi-resolution STFT loss. This loss is based upon the concept of the sum and difference signals, which are often used in the analysis and processing of stereo signals (Streicher and Burden, 1985). To generate the sum signal, we take the sum of the left and right channels, and to generate the difference signal we take the sum of the left channel and the inverse of the right channel, as shown in Equation 5.1 and 5.2. These two signals fully retain the stereo information of the original left and right signals, but represent the spatial information in a different manner.

$$y_{\text{sum}} = y_{\text{left}} + y_{\text{right}} \tag{5.1}$$

$$y_{\text{diff}} = y_{\text{left}} - y_{\text{right}} \tag{5.2}$$

But this transformation is not enough to ensure invariance between the stereo orientation that we desire. That is because the phase of these signals encodes information about the absolute stereo orientation between the left and right channels, and therefore using a time domain loss like the L1, will not address the behavior we have observed. The final step involves applying the multi-resolution STFT loss to these sum and difference signals, to compute a distance between the ground truth and predicted sum and difference signals in the *magnitude* STFT space, as shown in Equation 5.3. By attending to differences in the magnitude across the frequency range, and ignoring the phase information, we achieve a distance function where swapping the left and right channels will result in the same distance. We found that by applying this loss we our model was able to achieve realistic stereo panning on a diverse real-world dataset like MedleyDB.

$$\ell_{\text{Stereo}}(\hat{y}, y) = \ell_{\text{MR-STFT}}(\hat{y}_{\text{sum}}, y_{\text{sum}}) + \ell_{\text{MR-STFT}}(\hat{y}_{\text{diff}}, y_{\text{diff}}) \tag{5.3}$$

5.5 Training

To train all of the multitrack mixing models we follow a fairly standard method where we employ the Adam optimizer (Kingma and Ba, 2015) with a learning rate of 3e−4. We use a simple plateau learning rate scheduler that halves the learning rate when the validation loss has not decreased for 200 epochs. We then additionally train until the validation loss has not decreased for 500 epochs. Full details on all the models we train are shown in Table 11.

We train two different models on the ENST-drums mixing task, which requires at most 8 input recordings. The ENST-drums dataset provides two mixes for each passage, the dry mix with only gain and panning, and the wet mix that features gain, panning, equalization, compression, and reverb. In our experiments with this dataset, we train using the dry mix using the DMC-basic and Wave-U-Net models. Some samples in this dataset have only 7 input recordings, so to address this for the Wave-U-Net model, which expects 8 inputs, we simply pass an input with zeros. This is not a consideration for the DMC-basic model, since it can adapt to process any number of input recordings, only ever processing the inputs it is passed.

As previously indicated, the drum mixing task is likely not highly realistic, and serves mostly as a strong test case for our models. After observing acceptable performance on this task from both the Wave-U-Net and DMC-basic models, we decided to focus our efforts on the significantly more challenging task of learning from the MedleyDB dataset. We trained models with all samples in MedleyDB with 8 input recordings or less (DMC-basic-8), as well as all samples with 16 input recordings or less (DMC-basic-16). And finally we train a complete modeling using the complete signal processing chain, with all samples in MedleyDB with 5 input recordings or less (DMC-full-5). We also attempted to train a Wave-U-Net model with both 8 and 16 inputs using samples from MedleyDB, but we found that neither of these models were able to converge, and the output was heavily distorted and noise-like. We claim that this is due to the challenges previously discussed.

Model	Parameters	Dataset	Steps	Time (hr.)	Time (days)	GPU
Wave-U-Net	83M	ENST-drums	6.1k	30	1.3	RTX 2080 Ti
DMC-basic-8	89M	ENST-drums	1.0k	24	1.0	RTX 2080 Ti
DMC-basic-8	89M	MedleyDB	1.2k	24	1.0	RTX 2080 Ti
DMC-basic-16	89M	MedleyDB	1.1k	42	1.8	RTX 2080 Ti
DMC-full-5	89M	MedleyDB	1.8k	125	5.2	V100

Table 11: Multitrack mixing system training runs.

5.6 Evaluation

5.6.1 Objective metrics

While the objective evaluation of multitrack mix quality has been studied (Wilson and Fazenda, 2015, 2016, De Man and Reiss, 2017, Colonel and Reiss, 2019), there is still a lack of agreed upon models or metrics for accessing the perceived quality of mixes. For this reason, the gold-standard for evaluating mix quality remains formal listening tests, conducted with trained audio engineers (Jillings et al., 2016).

The desire for perceptually grounded, objective metrics for accessing the perceived

quality of audio content extends beyond the realm of multitrack mixing. There is great interest in such metrics across a number of different audio signals, with speech signals likely of greatest interest. Objective metrics prove invaluable in the assessment of text-to-speech, noise reduction, and voice conversion systems, for example, since the process of collecting evaluations from human listeners is often time consuming and costly. A number of metrics have been proposed for speech, such as PESQ (Rix et al., 2001), STOI (Taal et al., 2010) and SI-SDR (Le Roux et al., 2019), which have shown to have some correlation to the evaluations from human evaluators, but they still fall short in many cases with respect to human perception. For this reason, there has been a growing interest in the development of more sophisticated methods for perceptual quality metrics.

Furthermore, interest in perceptual metrics extend beyond audio signals, they are also of great interest in the visual domain as well. With the rapidly expanding interest in image generation, there has also been great focus on objective metrics for image quality. One of the most notable ideas in this domain, the Fréchet Inception Distance (FID), was introduced in Heusel et al. (2017), and proposed the idea of comparing the distance between multivariate Gaussian distributions of the embeddings generated from a pre-trained Inception network (Szegedy et al., 2015) on real and synthesized images.

To calculate this metric a large number of features are extracted with the pre-trained network from real images, as well as images generated by the model under evaluation. This Fréchet distance can then be measured between the two multivariate Gaussian distributions, $\mathcal{N}_b$ and $\mathcal{N}_e$, following the derivations from Dowson and Landau (1982),

$$\mathbf{F}(\mathcal{N}_b, \mathcal{N}_e) = ||\mu_b - \mu_e||^2 + tr(\Sigma_b + \Sigma_e - 2\sqrt{\Sigma_b \Sigma_e}),$$

where tr is the trace if a matrix. While this metric was proposed originally for images, it has also be adapted for audio signals in Kilgour et al. (2019), which proposed the Fréchet Audio Distance (FAD). This metric uses features extracted from audio signals with the pre-trained VGGish network (Hershey et al., 2017), which, similarity to the Inception network, was tasked with classifying sounds into a number of different categories. More recently, this work has been extended in Bińkowski et al. (2019), which introduces the Fréchet DeepSpeech Distance (FDSD) that takes advantage of features extracted from the pre-trained DeepSpeech2 network (Amodei et al., 2016), a model trained on the speech recognition task.

We adopt this method of computing a distance in the feature space between two distributions of audio examples in an attempt to evaluate the perceived quality of multitrack mixes. To do so we utilize two pre-trained VGGish models from Pons

and Serra (2019) that were trained on music auto-tagging tasks with the MagnaTagATune Dataset (`MTT-vgg`) (Law et al., 2009) and Million Song Dataset (MSD-vgg) (Bertin-Mahieux et al., 2011). To compute the scores for each method we first generate mixes of all the songs in the test set using each of the proposed methods, and then we measure the Fréchet distance as defined above between the distribution of features for each method and the target mixes.

ENST-drums

First, we report the results for the ENST-drums dataset, as shown in Table 12. Here we show the mean multi-resolution STFT distance (our specialized stereo version used during training), as well as the Fréchet distances with features from both VGGish models. In this case, the Wave-U-Net and DMC-basic models were tasked with recreating the dry mix, consisting of only gain and panning. In this case, we find that that DMC-basic model outperforms the other methods across all three metrics. We note additionally that the Wave-U-Net model outperforms the baseline approaches as well.

Task	Model	MR-STFT	Fréchet $_{\text{MTT-vgg}}$	Fréchet $_{\text{MSD-vgg}}$
Dry mix	Mono mix	6.038	4.57	3.01
	Random mix	3.403	4.63	2.89
	Wave-U-Net	2.763	3.13	1.94
	DMC-basic	**1.825**	**2.24**	**1.16**

Table 12: Objective metrics on the ENST-drums dataset with 8 input channels. Here the DMC-basic model predicts only gain and panning values to generate the mix.

MedleyDB

Things become more complicated when we move to the MedleyDB dataset. There is significantly less homogeneity with regards to the content and style of the mixes in this dataset in comparison to tne ENST-drums dataset, which includes drum passages all mixed by the same engineer. In this more complicated case, there is evidence that these objective metrics begin to break down. In Table 13, objective metrics are shown for three different tasks using the MedleyDB dataset. In the first part of the table, we compare models that were trained using songs in MedleyDB that had 8 or less input recordings. These metrics indicate the the random mixes on average produced mixes with the lowest error in STFT space and that the mono mixes had the best Fréchet scores, although our informal listening indicates that the mixes produced by the DMC-basic model far exceeded the quality of these mono

mixes. The results of the Wave-U-Net model do make sense though, as we were unable to successfully train this model to converge, and even after extended training the output of the model was heavily distorted and noise like. For this reason, we refrained from training further Wave-U-Net models on the MedleyDB mixing tasks.

Moving on the second part of the table, where a new DMC-basic model was trained using songs with 16 or less input recordings, the results appear quite similar to those in the previous case. There are no clear trends or agreement among the metrics. In the final section of the table, a third model, DMC-full, which includes not only gain and panning, but also equalization, compression, and reverb, was trained using songs in the MedleyDB that were comprised of 5 or less input recordings. Again, the results are quite similar, with the mono mixes scoring most highly from the Fréchet metrics, and the DMC-full model appears to perform worse with the addition of multiple effects in the chain.

Channels	Model	MR-STFT	Fréchet $_{\text{MTT-vgg}}$	Fréchet $_{\text{MSD-vgg}}$
8	Mono mix	5.363	**2.21**	**1.41**
	Random mix	**3.018**	2.66	1.48
	Wave-U-Net	7.386	40.45	29.62
	DMC-basic	3.024	2.91	1.64
16	Mono mix	5.840	**2.34**	2.92
	Random mix	**3.186**	2.40	2.63
	DMC-basic	3.341	3.00	**1.65**
5	Mono mix	5.300	**2.89**	**1.67**
	Random mix	**3.040**	3.42	1.87
	DMC-full	3.689	6.72	4.04

Table 13: Objective metrics on the MedleyDB dataset with 8 and 16 input channels.

From informal listening we determined that there was evidence to suggest that both the STFT and Fréchet metrics are likely not to agree with perception of mix quality when then underlying distributions of mixes in the reference set are diverse. In the case of the ENST-drums dataset we find that the Fréchet distance metrics agree with the STFT distance metric, but we know that the underlying distribution of mixes in the training and test sets are quite similar, and therefore it is likely that generated mixes based upon conventions in the training set will align closely with those in the test. The case is quite different for MeldeyDB based models, since in the case of complete songs there are a greater number of potential mix configurations that would be considered of high quality, but are likely located very far apart in STFT space. Ideally, the Fréchet metrics would address this disparity, but we posit that the features learned in the auto-tagging task are more closely related to the STFT features, as these models are concerned with identifying different musi-

cal elements, like style, and are less so concerned with distinguishing small details between different mixes of the same content with regards to quality. This finding motivates the design of a task formulation for training a model that learns features to differentiate mix quality among different mixes of the same underlying content, but we leave this investigation to future work.

5.6.2 Subjective evaluation

Due to the apparent inadequacy of the proposed objective metrics in evaluating the perceived quality of multitrack mixes, we must rely on the feedback from experienced audio engineers. To do so, we design and conduct a perceptual evaluation using the Web Audio Evaluation Tool (Jillings et al., 2016), using the APE test design (De Man and Reiss, 2014). This test extends the traditional Multiple Stimuli with Hidden Reference and Anchor (MUSHRA) test (mus, 2003), to instead use a series of sliders placed on the same axis, one for each stimulus. This encourages comparative ratings and relaxes the requirement for both anchor and reference stimuli. This test was designed as a more flexible alternative to the MUSHRA test, with a focus on providing a robust method for evaluating multitrack mixes, since unlike the MUSHRA test which was designed for evaluating the performance of evaluating codecs, there is often no known reference, and it may be possible for potential anchor stimuli, such a simple monophonic sum of the inputs, to be rated higher than competing mixes.

In our evaluation, we aim to determine the performance of the proposed deep learning methods, in comparison to the simplistic baselines we proposed. To achieve this, we select a number of short passages (4 to 5) from the test set of each dataset, and generate a mix with each method. During the test, participants are shown an interface like the one shown in Figure 16, with each stimulus represented by a green sliding bar. Clicking on each bar will play the mix for the associated method, allowing listeners to quickly switch among the different methods, for more detailed comparison. The listeners were then instructed to compare all the mixes, dragging them along the scale based upon their perceived level of quality. Participants were not informed of the underlying methods to generate the mixes, only that they should attend to differences in the mixtures, and make judgments based upon their internal preferences with regards to general qualities of music productions, such as balance between the sources, spectral characteristics, and dynamics. Our evaluation consisted of 16 participants, none of which indicated they had any hearing impairments. Out of the 16 participants, all 16 reported having previous experience with multitrack mixing. In the following sections we will perform an evaluation of these results for all of the mix generation methods, across to the two different datasets.

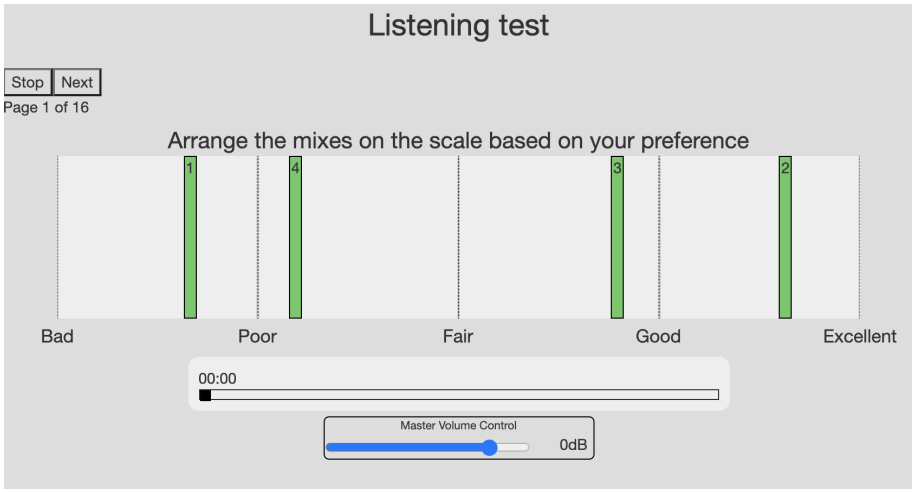

Figure 16: Screenshot of one of the pages from the test interface, where four different mixes are presented to the listener, and they are tasked with rating each mix relative to each other on the scale.

ENST-drums

In the evaluation, participants were presented with five different drum passages from the ENST-drums set, each with a mix generated from one of the five different methods (DMC-basic (ours), Wave-U-Net, Mono, Random, and Target). Individual box plots for the evaluations of each of the five different passages are shown in Figure 17. Additionally, Figure 18 shows the distribution of all scores for each method across all the drums passages.

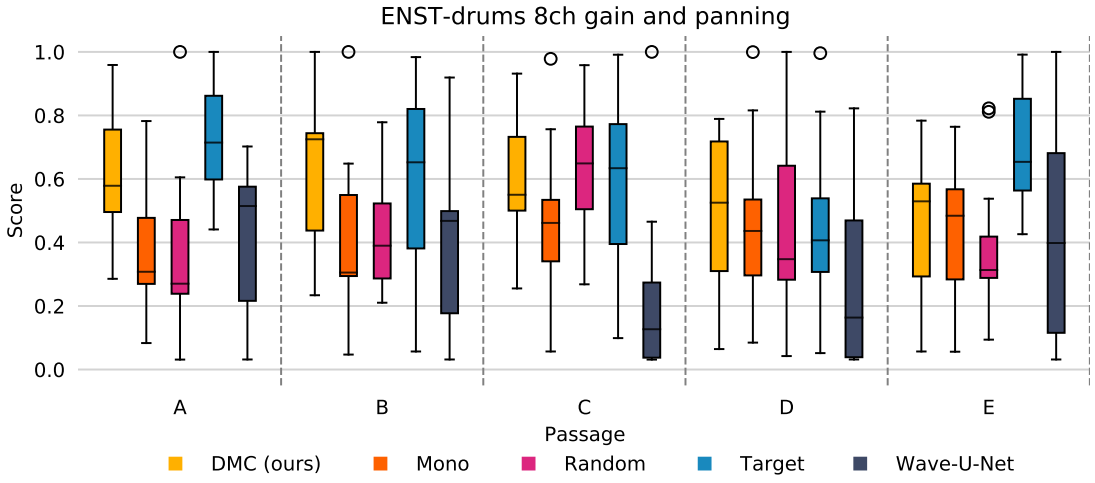

Figure 17: Individual ratings of the five drum passages from the ENST-drums dataset used in the perceptual evaluation.

Across the five passages, it seems that the target mixes tend to be rated the highest, with the mixes generated by the DMC-basic model following closely behind. In passages B and D, the DMC-basic model even has a mean score greater than the target mixes. The Wave-U-Net model appears to perform the worst across most of the songs, and the mono and random mixes perform similarly across all the songs, somewhere between the DMC-basic and Wave-U-Net mixes. These findings are confirmed by the boxplot shown in Figure 18, which shows the aggregation of ratings for all the passages together for each method. Feedback from participants indicated that the low ratings for the Wave-U-Net model were likely due to the artifacts that were present on some mixes caused by the transposed convolutions used in the decoder portion of the network (Odena et al., 2016).

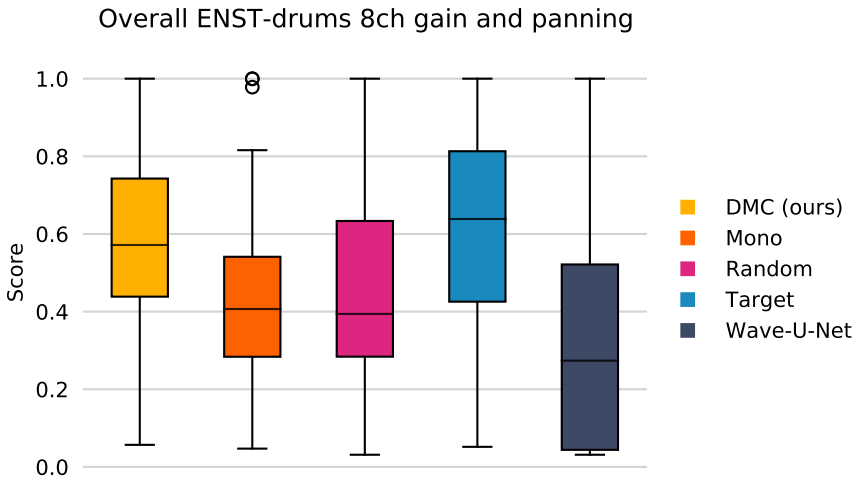

Figure 18: Aggregation of all ratings for each of the five mix generation methods for the ENST-drums dataset.

To formalize these results, we perform the Kruskal-Wallis H-test to test the null hypothesis that the mean of the scores for mixes generated by each method are the same. We opt to not perform the one-way analysis of variance (ANOVA), since there is no evidence that our samples are normally distributed. We reject the null hypothesis and find that there is a statistically significant difference in the means of the five methods ($F = 64.01, p = 8.0\text{e}{-14}$). To continue our investigation, we employ Conover's test to compute a multiple comparison of means across all mix generation methods. This test reveals that our proposed DMC-basic method and the target mixes do not have a significant difference in their means ($P_{adj} = 8.03\text{e}{-1}$), indicating this method produces mixes on par with the ground truth mixes. There is a significant difference between the means of the DMC-basic and mono mixes ($P_{adj} = 2.34\text{e}{-3}$), as well as random mixes ($P_{adj} = 1.71\text{e}{-2}$). Unsurprisingly, we

also find there is a significant difference between the means of the target mixes, and those mixes generated with the mono ($P_{adj} = 1.55\text{e}{-3}$), random ($P_{adj} = 1.25\text{e}{-3}$), and Wave-U-Net ($P_{adj} = 4.69\text{e}{-9}$) methods, indicating that these methods produce mixes of lower quality on average compared to the ground truth. The results largely affirm the trends we observed in the aggregate box plots in Figure 18. We report the complete test results in Table 14.

Group 1	Group 2	Mean Diff.	P_{adj}	Reject
DMC-basic	Mono	0.142	3.24e-3	True
DMC-basic	Rand	0.113	1.71e-2	True
DMC-basic	Target	-0.037	8.03e-1	False
DMC-basic	Wave-U-Net	0.232	4.50e-7	True
Mono	Rand	-0.029	8.02e-1	False
Mono	Target	-0.179	1.55e-3	True
Mono	Wave-U-Net	0.091	1.13e-1	False
Rand	Target	-0.150	1.25e-3	True
Rand	Wave-U-Net	0.119	3.57e-2	True
Target	Wave-U-Net	0.269	4.69e-9	True

Table 14: Results from Conover's test of multiple comparison of means for different mix generation methods on the ENST-drums dataset.

MedleyDB

In the case of MedleyDB, we train two different models, DMC-basic, which performs only gain and panning, and DMC-full, which performs gain, panning, equalization, compression, and reverb. In the evaluation, participants were presented with five different passages from the gain and panning task, and six different passages from the full mixing task, each with a mix generated from one of the four different methods (DMC-basic (ours), Mono, Random, and Target). We chose to omit the Wave-U-Net models from the perceptual evaluation since we found that this model was unable to be trained for the MedleyDB task, as the output after training was noise.

We start our analysis with the gain and panning task, with individual box plots for the evaluations of each of the five different passages are shown in Figure 19. Additionally, Figure 20 shows the distribution of scores for each method across all the passages. Similar trends to those present in the ENST-drums models are present here, with the target mixes generally rated more highly than the other methods. The means of the DMC-basic-16 mixes tend to be above the baselines for most mixes, but they appear closer than in the drum mixing task.

We again perform the Kruskal-Wallis H-test to test the null hypothesis that the mean

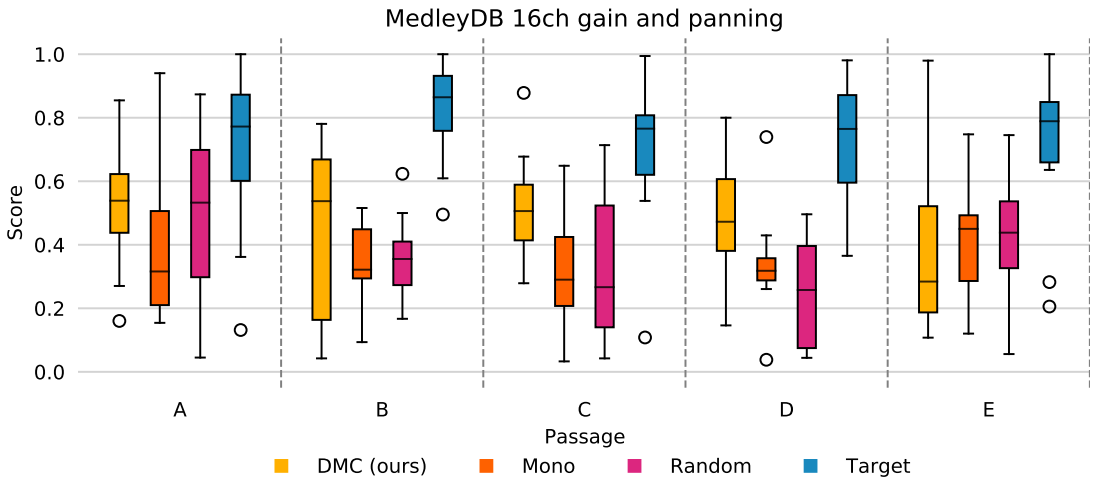

Figure 19: Individual ratings of the five passages from the MedleyDB dataset used in the perceptual evaluation. In this evaluation the DMC-basic model predicts only gain and panning values.

Group 1	Group 2	Mean Diff.	P_{adj}	Reject
DMC-basic-16	Mono	0.111	2.15e-3	True
DMC-basic-16	Random	0.097	1.04e-2	True
DMC-basic-16	Target	-0.265	3.92e-10	True
Mono	Random	-0.014	5.45e-1	False
Mono	Target	-0.376	6.77e-20	True
Random	Target	-0.362	4.53e-18	True

Table 15: Results from Conover's test of multiple comparison of means for different mix generation methods on the gain and panning task of the MedleyDB dataset.

of the scores for mixes generated by each method are the same. We find that there is a statistically significant difference in the means of the four methods ($F = 87.7, p =$ 6.6e−19). To continue our investigation, we employ Conover's post hoc test, which reveals that three methods present statistically significant difference in their means in comparison to the target mixes, for DMC-basic-16 ($P_{adj} = 3.92\text{e}{-10}$), mono mixes ($P_{adj} = 6.77\text{e}{-20}$), and random mixes ($P_{adj} = 4.53\text{e}{-18}$). Nevertheless, we find that the DMC-basic-16 has a mean that is statistically different in comparison to both the mono ($P_{adj} = 2.15\text{e}{-3}$) and random ($P_{adj} = 1.04\text{e}{-2}$) baseline approaches, indicating that the DMC-basic-16 model produces mixes of higher quality than the baselines, but not quite on par with the target mixes on average. We report the complete test results in Table 15.

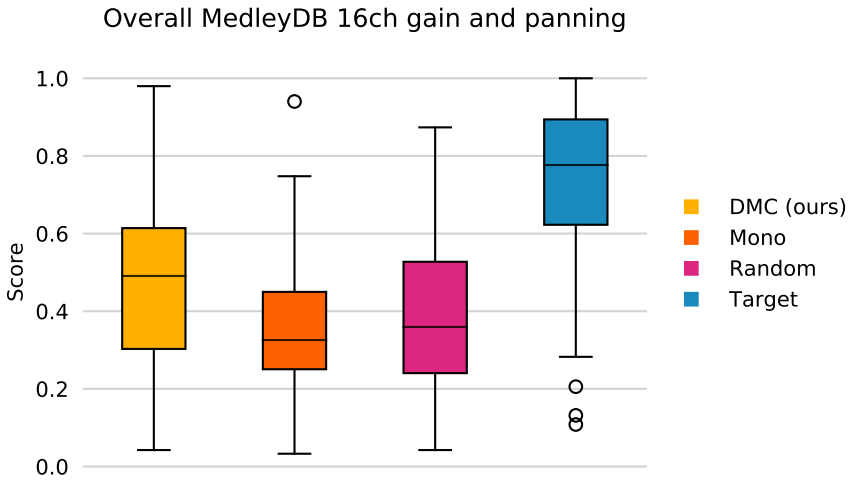

Figure 20: Aggregation of all ratings for each of the four mix generation methods for the MedleyDB dataset on the gain and panning task.

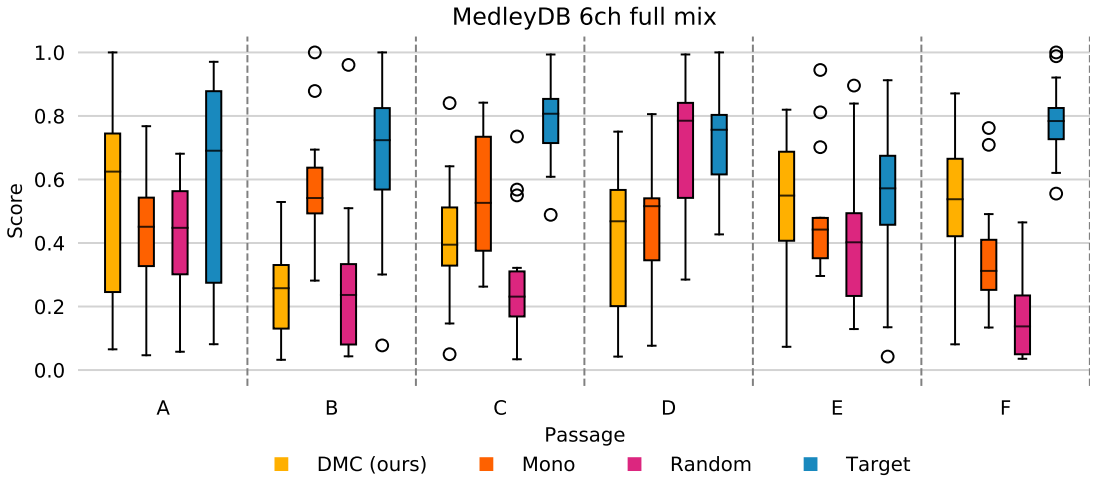

Figure 21: Individual ratings of the six passages from the MedleyDB dataset used in the perceptual evaluation. In this evaluation the DMC-full model predicts gain, panning, equalization, compression, and reverb values.

We carry out the same analysis for the final case, which involves the full mixing task. Individual box plots for the evaluations of each of the six different passages are shown in Figure 21. Additionally, Figure 22 shows the distribution of scores for each method across all the passages. We perform the Kruskal-Wallis H-test to test the

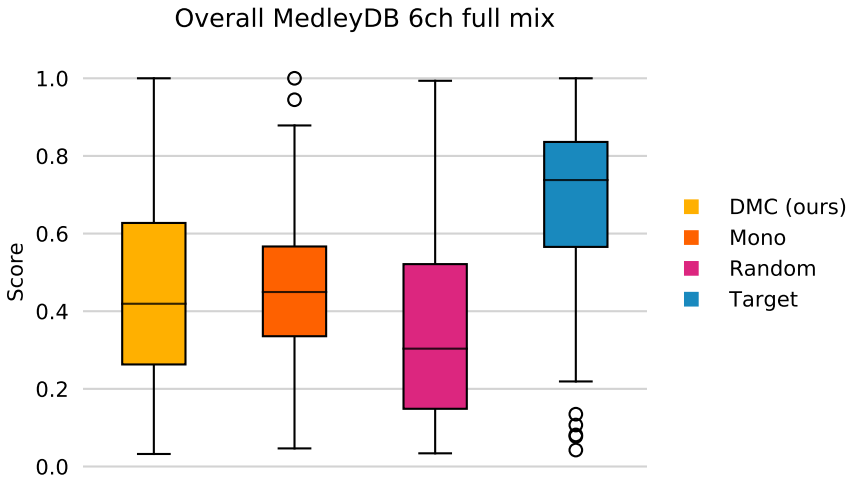

Figure 22: Aggregation of all ratings for each of the four mix generation methods for the MedleyDB dataset on the full mixing task.

Group 1	Group 2	Mean Diff.	P_{adj}	Reject
DMC-full-5	Mono	-0.037	3.02e-1	True
DMC-full-5	Random	0.068	1.22e-1	True
DMC-full-5	Target	-0.245	1.09e-9	True
Mono	Random	0.105	1.15e-2	True
Mono	Target	-0.208	2.74e-7	True
Random	Target	-0.314	6.64e-15	True

Table 16: Results from Conover's test of multiple comparison of means for different mix generation methods on the full mixing task of the MedleyDB dataset.

null hypothesis that the mean of the scores for mixes generated by each method are the same, and find once again that there is a significant difference among the means ($F = 48.1, p = 8.8\text{e}{-10}$). We continue with Conover's test and present the complete results in Table 16. In this case, as we can observe in Figure 22, performance for each of the methods, DMC-full, mono, and random mixes all preform worse than the target mixes, and have largely similar distributions.

This is confirmed by our post hoc test that shows a significant difference between the mean of the target mixes and the means of all the other methods. In our informal listening this result was confirmed as well. While the DMC-full model produced mixes that had clear equalization, compression, and reverb effects, with minimal artifacts, these effects were often overly present, causing them to be perceived as lower quality mixes. Furthermore, the individual boxplots in Figure 21 demonstrate that for some passages the DMC-full model is nearly on par with the target mixes

(A and E), but in other passages this model performs quite poorly (B and C). In our informal listening, there is evidence to suggest these failures come in cases where the model fails to set the level of the lead vocal loud enough, which listeners often use a main component in their evaluation.

Chapter 6

Discussion

6.1 Conclusions

In this thesis, we outline a number of challenges in applying deep learning methods in the task of building an automated multitrack mixing system, many of which have yet to be addressed in the literature. To address these new challenges, we proposed a domain-inspired architecture along with a specialized stereo loss function. By leveraging a stronger inductive bias for the mixing task with pre-trained subnetworks, along with extensive weight sharing throughout the system, we construct a model that can be trained using a limited number of multitrack training examples, adapts to variation among real-world multitrack projects, and ultimately was shown in a perceptual evaluation to produce mixes that exceed our baseline approaches.

In the process of building this multitrack mixing system, we extend the state of the art in neural audio effects. We show that a dense sampling of the control parameters across a series connection of linear and nonlinear audio effects can be jointly modelled by a neural network. Additionally, we demonstrated the ability of our methods to generalize to the task of modeling an analog dynamic range compressor in the limited data regime, exceeding the performance of the previously proposed approach.

Ultimately, we demonstrated we were able to leverage paired, yet unstructured multitrack mixing data in the task of learning how to perform multitrack mixing across both the drum mixing and complete mixing project datasets. And while there remains a challenge for the system to create quality mixes consistently when controlling all of the complex processors in the mixing console, positive results from the simpler gain and panning mixes indicate that with additional training data it is likely we can improve the generalization abilities of the model when controlling a set of complex signal processors.

6.2 Future work

As this thesis aimed to investigate the emerging research direction of applying deep learning methods in the design automated of multitrack mixing systems, this work is far from presenting a complete solution. Instead, our work was focused on the development of a deep learning framework for building a system with the ability to learn from the high dimensional, unstructured data common in multitrack mixing. Therefore, our work leads into a number of open research directions that could have significant impact in IMP by building upon the framework that we introduced.

One of the critical elements in designing effective mixing systems is the ability to quickly and accurately evaluate the quality of the mixes they produce. Currently, running time consuming evaluations with human listeners remains the only effective method. While we investigated potential objective metrics for this evaluation, we ultimately found that further work is needed to develop metrics that align with human perception. While we employed the Fréchet distance in the feature space of models trained on general audio signals, it is likely that building new models pre-trained on tasks related to multitrack mixing will produce more useful representations.

The next clear path towards improving these systems is in the use of larger and more diverse training datasets. While we restrict our experiments in this thesis to only the ENST-drums and MedleyDB datasets, there are additional sources of mixing data, such as the Mixing Secrets Multitrack Library[1], which contain many more samples, but are less consistent in their structure. Most likely, in order to achieve a significant step forward in the performance of these systems, we will need to leverage non-parallel mix data with the use of an adversarial loss function. This will relax the constraint of needing the original tracks as well as mix of those tracks in order to learn about the mixing process, hence enabling the ability to use large collections of produced music, such as the MTG-Jamendo (Bogdanov et al., 2019) or Million song (Bertin-Mahieux et al., 2011) datasets. Further extending this adversarial formulation to make the model generative would provide the ability to model the mixing process as a one-to-many mapping, and provide a method for users to sample from the latent mix space, providing further methods for user interaction.

Beyond these main directions, other directions include architectural extensions such as incorporating additional context information in the controller network, the development of more compute efficient transformation networks, extending the routing capabilities of the differential mixing console to include auxiliary busses, as well as the the application of attention mechanisms in order to more effectively take advantage of the information within the input representations across time.

[1] https://cambridge-mt.com/ms/mtk/

List of Figures

1 Block diagram of the basic design of a multi-channel knowledge-based intelligent audio effect, which is adapted from De Man et al. (2019). . 15

2 Desired formulation where a controller network learns to predict the optimal parameterization of a traditional mixing console given some input recordings. 23

3 Proposed formulation where a controller network learns to predict the optimal parameterization of a mixing console proxy, which is a neural network pre-trained to emulate the processing of a mixing console. . . 24

4 High level overview of the proposed multitrack mixing system. 25

5 Block diagram of the audio effect modeling task, where we train a neural network to closely imitate an existing DSP implementation of the audio effect. 29

6 Block diagram of the parametric equalizer with a low shelf filter, three peaking filters, and a high shelf filter, where G is the gain, F_c is the cutoff frequency, and Q is the Q-factor, or bandwidth. 31

7 Block diagram of the simple hard-knee dynamic range compressor adapted from Reiss and McPherson (2014). 33

8 Block diagram of the complete channel strip formulation that aims to mimic the canonical series connection of processors found in a single channel of a multitrack mixing console. The input to the chain is always a mono signal, and the output will be a stereo signal. 35

9 High-level view of the TCN architecture for the effect modeling task. 37

10 Single convolutional block in the TCN architecture. 39

11 High level overview of the modified Wave-U-Net architecture. 41

12 Convolutional encoder block in the Wave-U-Net architecture with Demucs modifications. 42

13 Convolutional decoder block in the Wave-U-Net architecture with Demucs modifications. 42

14 (**A**) The pre-trained CNN encoder architecture operating on mel-spectrograms following the VGGish implementation. (**B**) The post-processor network, which is a simple MLP that takes the latent vector from the current channel along with the context vector. (**C**) The pre-trained TCN based transformation network that processors waveforms given a set of appropriate parameters. (**D**) The complete architecture showing the interconnection of all the subsystems. 58
15 Composition of the blocks within the VGGish (Hershey et al., 2017) architecture. (**A**) Conv block A, with a single 2D convolution followed by max pooling. (**B**) Conv block B, with two convolutions followed by max pooling. (**C**) FC block, with a series of linear layers and ReLU activations. 59
16 Screenshot of one of the pages from the test interface, where four different mixes are presented to the listener, and they are tasked with rating each mix relative to each other on the scale. 70
17 Individual ratings of the five drum passages from the ENST-drums dataset used in the perceptual evaluation. 70
18 Aggregation of all ratings for each of the five mix generation methods for the ENST-drums dataset. 71
19 Individual ratings of the five passages from the MedleyDB dataset used in the perceptual evaluation. In this evaluation the DMC-basic model predicts only gain and panning values. 73
20 Aggregation of all ratings for each of the four mix generation methods for the MedleyDB dataset on the gain and panning task. 74
21 Individual ratings of the six passages from the MedleyDB dataset used in the perceptual evaluation. In this evaluation the DMC-full model predicts gain, panning, equalization, compression, and reverb values. 74
22 Aggregation of all ratings for each of the four mix generation methods for the MedleyDB dataset on the full mixing task. 75

List of Tables

1 Summary of previous approaches in deep learning for audio effect modeling. The Param. column indicates whether or not the control parameters of the effect were modelled, or if only a single configuration of the model was used. For the loss functions, (t) denotes time domain and (f) denotes frequency domain. ESR denotes the error-to-signal ratio, and DC denotes a loss between the DC offsets. Complete details about the dataset used are shown in Table 2.2.2 11

2 Summary of datasets commonly used in the audio effect modeling task, as well as mutlitack datasets. Information about the number of samples, their length, and total size of the datasets are provided when available. 13

3 TCN model hyperparameters 40

4 Wave-U-Net-8 model hyperparameters 43

5 Multi-resolution STFT bands 50

6 Overview of results for modeling the channel effects (static order) with the TCN and Wave-U-Net architecture across varying receptive fields. 50

7 Overview of results with the TCN-10 model for the static or dynamic ordering of the processors in the channel. 51

8 Channel based effects training runs details.
(All runs use one NVIDIA RTX 2080 Ti) 52

9 Results for modeling the analog LA-2A compressor on the SignalTrain test set. Standard deviation across all of the test set examples is shown in parentheses. 53

10 Analog compressor training runs details. (All runs use one NVIDIA RTX 2080 Ti) 54

11 Multitrack mixing system training runs. 65

12 Objective metrics on the ENST-drums dataset with 8 input channels. Here the DMC-basic model predicts only gain and panning values to generate the mix. 67

13 Objective metrics on the MedleyDB dataset with 8 and 16 input channels. 68
14 Results from Conover's test of multiple comparison of means for different mix generation methods on the ENST-drums dataset. 72
15 Results from Conover's test of multiple comparison of means for different mix generation methods on the gain and panning task of the MedleyDB dataset. 73
16 Results from Conover's test of multiple comparison of means for different mix generation methods on the full mixing task of the MedleyDB dataset. 75

Bibliography

Method for the subjective assessment of intermediate quality level of coding systems. Recommendation ITU-R BS.1534-1, 2003.

Jakob Abeßer, Patrick Kramer, Christian Dittmar, Gerald Schuller, and IDMT Fraunhofer. Parametric audio coding of bass guitar recordings using a tuned physical modeling algorithm. In *Abeßer, Jakob, et al. "Parametric audio coding of bass guitar recordings using a tuned physical modeling algorithm." Proc. of the 16th Int. Conference on Digital Audio Effects (DAFx-13). Maynooth, Ireland*, 2013.

Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*, pages 173–182, 2016.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

Adán L. Benito and Joshua D. Reiss. Intelligent multitrack reverberation based on hinge-loss markov random fields. In *Audio Engineering Society Conference: 2017 AES International Conference on Semantic Audio*, Jun 2017.

Thierry Bertin-Mahieux, Daniel Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, pages 591–596, 01 2011.

Stefan Bilbao. Numerical Simulation of Spring Reverberation Citation. In *Proceedings of the 16th International Digital Audio Effects Conference*, Maynooth, Ireland, 2013.

Mikołaj Bińkowski, Jeff Donahue, Sander Dieleman, Aidan Clark, Erich Elsen, Norman Casagrande, Luis Cobo, and Karen Simonyan. High fidelity speech synthesis with adversarial networks. In *ICLR 2019*, 09 2019.

Rachel Bittner, Justin Salamon, Mike Tierney, Matthias Mauch, Chris Cannam, and Juan Bello. Medleydb: A multitrack dataset for annotation-intensive mir research. In *15th International Society for Music Information Retrieval Conference, ISMIR 2014*, 2014.

Rachel Bittner, Julia Wilkins, Hanna Yip, and Juan Pablo Bello. Medleydb 2.0: New data and a system for sustainable data collection. 2016.

Dmitry Bogdanov, Minz Won, Philip Tovstogan, Alastair Porter, and Xavier Serra. The mtg-jamendo dataset for automatic music tagging. In *Machine Learning for Music Discovery Workshop, International Conference on Machine Learning (ICML 2019)*, Long Beach, CA, United States, 15/06/2019 2019.

BS1770. Algorithms to measure audio programme loudness and true-peak audio level. Recommendation, International Telecommunication Union, oct 2015.

Alex Case. *Mix smart: Pro audio tips for your multitrack mix*. Focal Press, 2011.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179.

Emmanouil T. Chourdakis and Joshua D. Reiss. Automatic control of a digital reverberation effect using hybrid models. In *Audio Engineering Society Conference: 60th International Conference: DREAMS (Dereverberation and Reverberation of Audio, Music, and Speech)*, Jan 2016.

Emmanouil T. Chourdakis and Joshua D. Reiss. A machine-learning approach to application of intelligent artificial reverberation. *J. Audio Eng. Soc*, 65(1/2):56–65, 2017.

Joseph Colonel and Joshua D. Reiss. Exploring preference for multitrack mixes using statistical analysis of mir and textual features. In *Audio Engineering Society Convention 147*, Oct 2019.

J. Covert and D. L. Livingston. A vacuum-tube guitar amplifier model using a recurrent neural network. In *2013 Proceedings of IEEE Southeastcon*, pages 1–5, April 2013. doi: 10.1109/SECON.2013.6567472.

Eero-Pekka Damskägg, Lauri Juvela, Etienne Thuillier, and Vesa Välimäki. Deep Learning for Tube Amplifier Emulation. *arXiv e-prints*, art. arXiv:1811.00334, Nov 2018.

Eero-Pekka Damskägg, Lauri Juvela, Etienne Thuillier, and Vesa Välimäki. Deep learning for tube amplifier emulation. *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 471–475, 2019a.

Eero-Pekka Damskägg, Lauri Juvela, Vesa Välimäki, et al. Real-Time Modeling of Audio Distortion Circuits with Deep Learning. In *16th Sound and Music Computing Conference (SMC2019)*, May 2019b. doi: 10.5281/zenodo.3249374.

Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941, 2017.

Brecht De Man. *Towards a better understanding of mix engineering.* PhD thesis, Queen Mary University of London, 2017.

Brecht De Man and Joshua D Reiss. A semantic approach to autonomous mixing. *Journal on the Art of Record Production*, 2013a.

Brecht De Man and Joshua D. Reiss. A knowledge-engineered autonomous mixing system. In *Audio Engineering Society Convention 135*, Oct 2013b.

Brecht De Man and Joshua D Reiss. Ape: Audio perceptual evaluation toolbox for matlab. In *Audio Engineering Society Convention 136.* Audio Engineering Society, 2014.

Brecht De Man and Joshua D. Reiss. The mix evaluation dataset. In *Proceedings of the 20th International Conference on Digital Audio Effects*, September 2017.

Brecht De Man, Mariano Mora-Mcginity, György Fazekas, and Joshua D. Reiss. The Open Multitrack Testbed. In *137th Convention of the Audio Engineering Society*, October 2014.

Brecht De Man, Joshua D Reiss, and Ryan Stables. Ten years of automatic mixing. In *Proceedings of the 3rd Workshop on Intelligent Music Production*, September 2017.

Brecht De Man, Ryan Stables, and Joshua D. Reiss. *Intelligent Music Production.* Audio Engineering Society Presents. Taylor & Francis, 2019. ISBN 9781351679022.

Alexandre Défossez, Neil Zeghidour, Nicolas Usunier, Leon Bottou, and Francis Bach. Sing: Symbol-to-instrument neural generator. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 9041–9051. Curran Associates, Inc., 2018.

Alexandre Défossez, Nicolas Usunier, Léon Bottou, and Francis Bach. Music source separation in the waveform domain. *arXiv preprint arXiv:1911.13254*, 2019.

Omar del Tejo Catalá and Luis Masiá Fuster. Audio effect emulation with neural networks. 2017.

DC Dowson and BV Landau. The fréchet distance between multivariate normal distributions. *Journal of multivariate analysis*, 12(3):450–455, 1982.

Peter Doyle. *Echo and reverb: fabricating space in popular music recording, 1900-1960*. Wesleyan University Press, 1st. edition, 2005. ISBN 0819567949.

Dan Dugan. Automatic microphone mixing. *Journal of the Audio Engineering Society*, pages 442–449, 1975.

Felix Eichas and Udo Zölzer. Black-box modeling of distortion circuits with block-oriented models. In *Proceedings of the International Conference on Digital Audio Effects (DAFx), Brno, Czech Republic*, pages 5–9, 2016.

Felix Eichas and Udo Zölzer. Gray-box modeling of guitar amplifiers. *Journal of the Audio Engineering Society*, 66(12):1006–1015, 2018.

Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Mohammad Norouzi, Douglas Eck, and Karen Simonyan. Neural audio synthesis of musical notes with wavenet autoencoders. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1068–1077. JMLR. org, 2017.

Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. Gansynth: Adversarial neural audio synthesis. 2019.

Steven Fenton. Automatic mixing of multitrack material using modified loudness models. In *Audio Engineering Society Convention 145*, Oct 2018.

Eduardo Fonseca, Jordi Pons, Xavier Favory, Frederic Font, Dmitry Bogdanov, Andrés Ferraro, Sergio Oramas, Alastair Porter, and Xavier Serra. Freesound datasets: a platform for the creation of open audio datasets. In *Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR 2017)*, pages 486–493, Suzhou, China, 2017.

Eduardo Fonseca, Manoj Plakal, Daniel PW Ellis, Frederic Font, Xavier Favory, and Xavier Serra. Learning sound event classifiers from web audio with noisy labels. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 21–25. IEEE, 2019.

Jort F. Gemmeke, Daniel P. W. Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R. Channing Moore, Manoj Plakal, and Marvin Ritter. Audio set: An ontology and human-labeled dataset for audio events. In *Proc. IEEE ICASSP 2017*, New Orleans, LA, 2017.

Dimitrios Giannoulis, Michael Massberg, and Joshua D Reiss. Digital Dynamic Range Compressor Design. *Journal of the Audio Engineering Society*, 60(6):399–408, 2012.

Olivier Gillet and Gaël Richard. Enst-drums: an extensive audio-visual database for drum signals processing. In *ISMIR*, 2006.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

Daniel Griffin and Jae Lim. Signal estimation from modified short-time fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32 (2):236–243, 1984.

Scott H. Hawley, Benjamin L. Colburn, and Stylianos Ioannis Mimilakis. Signaltrain: Profiling audio compressors with deep neural networks. *ArXiv*, abs/1905.11928, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

Thomas Hélie. Volterra series and state transformation for real-time simulations of audio circuits including saturations: Application to the moog ladder filter. *IEEE transactions on audio, speech, and language processing*, 18(4):747–759, 2009.

J. R. Hershey, Z. Chen, J. Le Roux, and S. Watanabe. Deep clustering: Discriminative embeddings for segmentation and separation. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 31–35, March 2016. doi: 10.1109/ICASSP.2016.7471631.

Shawn Hershey, Sourish Chaudhuri, Daniel PW Ellis, Jort F Gemmeke, Aren Jansen, R Channing Moore, Manoj Plakal, Devin Platt, Rif A Saurous, Bryan Seybold, et al. Cnn architectures for large-scale audio classification. In *2017 ieee*

international conference on acoustics, speech and signal processing (icassp), pages 131–135. IEEE, 2017.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems*, pages 6626–6637, 2017.

Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Georg Holzmann and Helmut Hauser. Echo state networks with filter neurons and a delay&sum readout. *Neural Networks*, 23(2):244–256, 2010.

Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1501–1510, 2017.

D.M. Huber and R.E. Runstein. *Modern Recording Techniques*. Audio Engineering Society Presents Series. Focal Press/Elsevier, 2010. ISBN 9780240810690.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

ISO 226:2003. Normal equal-loudness-level contours. Standard, International Organization for Standardization, aug 2003.

Roey Izhaki. *Mixing audio: concepts, practices and tools*. Taylor & Francis, 2013.

Nicholas Jillings, Brecht De Man, David Moffat, and Joshua D Reiss. Web audio evaluation tool: A framework for subjective assessment of audio. In *Proceedings of the 2nd Web Audio Conference*, 04 2016.

Matti Karjalainen and Jyri Tapani Pakarinen. Wave digital simulation of a vacuum-tube amplifier. *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, 5:V–V, 2006.

B. Katz and R.A. Katz. *Mastering Audio: The Art and the Science*. Elsevier/Focal Press, 2007. ISBN 9780240808376.

Christian Kehling. Automatic tablature transcription of electric guitar recordings by estimation of score-and instrument-related parameters. In *Proceedings of the 17th International Conference on Digital Audio Effects (DAFx, 2014), Erlangen, Germany*, 2014.

Kevin Kilgour, Mauricio Zuluaga, Dominik Roblek, and Matthew Sharifi. Fréchet audio distance: A reference-free metric for evaluating music enhancement algorithms. pages 2350–2354, 09 2019. doi: 10.21437/Interspeech.2019-2219.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

Bennett Kolasinski. A framework for automatic mixing using timbral similarity measures and genetic optimization. In *Audio Engineering Society Convention 124*, May 2008.

John Kominek, Tanja Schultz, and Alan W Black. Synthesizer voice quality of new languages calibrated with mean mel cepstral distortion. In *Spoken Languages Technologies for Under-Resourced Languages*, 2008.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

R. Kubichek. Mel-cepstral distance measure for objective speech quality assessment. In *Proceedings of IEEE Pacific Rim Conference on Communications Computers and Signal Processing*, volume 1, pages 125–128 vol.1, 1993.

Kundan Kumar, Rithesh Kumar, Thibault de Boissiere, Lucas Gestin, Wei Zhen Teoh, Jose Sotelo, Alexandre de Brébisson, Yoshua Bengio, and Aaron C Courville. Melgan: Generative adversarial networks for conditional waveform synthesis. In *Advances in Neural Information Processing Systems*, pages 14881–14892, 2019.

Edith Law, Kris West, Michael Mandel, Mert Bay, and J. Downie. Evaluation of algorithms using games: The case of music tagging. pages 387–392, 01 2009.

Jonathan Le Roux, Nobutaka Ono, and Shigeki Sagayama. Explicit consistency constraints for stft spectrograms and their application to phase reconstruction. In *SAPA@ INTERSPEECH*, 2008.

Jonathan Le Roux, Scott Wisdom, Hakan Erdogan, and John R Hershey. Sdr–half-baked or well done? In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 626–630. IEEE, 2019.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Francesc Lluís, Jordi Pons, and Xavier Serra. End-to-end music source separation: Is it possible in the waveform domain? *Proc. Interspeech 2019*, pages 4619–4623, 2019.

Stuart Mansbridge, Saoirse Finn, and Joshua D. Reiss. Implementation and evaluation of autonomous multi-track fader control. In *132nd Audio Engineering Society Convention*, 2012.

Marco A Martínez Ramírez and Joshua D Reiss. Deep learning and intelligent audio mixing. In *3rd Workshop on Intelligent Music Production (WIMP)*, 2017.

Marco A Martínez Ramírez and Joshua D Reiss. End-to-end equalization with convolutional neural networks. In *21st International Conference on Digital Audio Effects (DAFx-18)*, September 2018.

Marco A Martínez Ramírez and Joshua D Reiss. Modeling nonlinear audio effects with end-to-end deep neural networks. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 171–175, May 2019. doi: 10.1109/ICASSP.2019.8683529.

Marco A Martínez Ramírez, Emmanouil Benetos, and Joshua D Reiss. A general-purpose deep learning approach to model time-varying audio effects. In *22nd International Conference on Digital Audio Effects (DAFx-19)*, September 2019a.

Marco A Martínez Ramírez, Emmanouil Benetos, and Joshua D Reiss. Modeling plate and spring reverberation using a dsp-informed deep neural network. *arXiv preprint arXiv:1910.10105*, 2019b.

Marco A Martínez Ramírez, Emmanouil Benetos, and Joshua D Reiss. Deep learning for black-box modeling of audio effects. *Applied Sciences*, 10:638, 01 2020a. doi: 10.3390/app10020638.

Marco A Martínez Ramírez, Emmanouil Benetos, and Joshua D Reiss. Modeling plate and spring reverberation using a DSP-informed deep neural network. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2020b.

Mikiko Mashimo, Tomoki Toda, Kiyohiro Shikano, and Nick Campbell. Evaluation of cross-language voice conversion based on gmm and straight. In *Eurospeech 2001*, pages 361–364, 01 2001.

David Sanchez Mendoza. Emulating electric guitar effects with neural networks, 2005.

Gabriel Meseguer-Brocal and Geoffroy Peeters. Conditioned-u-net: Introducing a control mechanism in the u-net for multiple source separations. In *Proceedings of the 20th International Society for Music Information Retrieval Conference*, 2019.

Stylianos Ioannis Mimilakis, Konstantinos Drossos, Tuomas Virtanen, and Gerald Schuller. Deep neural networks for dynamic range compression in mastering applications. In *Audio Engineering Society Convention 140*, May 2016.

David Moffat and Mark Sandler. Machine learning multitrack gain mixing of drums. In *Audio Engineering Society Convention 147*, Oct 2019a.

David Moffat and Mark B. Sandler. Approaches in intelligent music production. *Arts*, 8(4), 2019b. ISSN 2076-0752. doi: 10.3390/arts8040125.

Arsha Nagrani, Joon Son Chung, Weidi Xie, and Andrew Zisserman. Voxceleb: Large-scale speaker verification in the wild. *Computer Science and Language*, 2019.

Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 1(10):e3, 2016.

B. Owsinski. *The Mixing Engineer's Handbook 2nd Edition*. Bobby Owsinski Media Group, 2006. ISBN 1-59863-251-5.

Francois Pachet and Olivier Delerue. On-the-fly multitrack mixing. In *Audio Engineering Society Convention 109*, Los Angeles, Califronia, USA, 2000.

Jyri Pakarinen and David T Yeh. A review of digital techniques for modeling vacuum-tube guitar amplifiers. *Computer Music Journal*, 33(2):85–100, 2009.

Daniel S Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D Cubuk, and Quoc V Le. Specaugment: A simple data augmentation method for automatic speech recognition. *arXiv preprint arXiv:1904.08779*, 2019.

Santiago Pascual, Antonio Bonafonte, and Joan Serrà. Segan: Speech enhancement generative adversarial network. *Proc. Interspeech 2017*, pages 3642–3646, 2017.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Enrique Perez Gonzalez. *Advanced automatic mixing tools for music*. PhD thesis, Queen Mary University of London, 2010.

Enrique Perez Gonzalez and Joshua D Reiss. Automatic mixing: Live downmixing stereo panner. In *Proc. of the 10th Int. Conference on Digital Audio Effects*, September 2007.

Enrique Perez Gonzalez and Joshua D Reiss. Determination and correction of individual channel time offsets for signals involved in an audio mixture. In *Audio Engineering Society Convention 125*, Oct 2008.

Enrique Perez Gonzalez and Joshua D Reiss. Automatic equalization of multichannel audio using cross-adaptive methods. In *Audio Engineering Society Convention 127*, Oct 2009a.

Enrique Perez Gonzalez and Joshua D Reiss. Automatic gain and fader control for live mixing. In *2009 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 1–4, Oct 2009b. doi: 10.1109/ASPAA.2009.5346498.

Jordi Pons and Xavier Serra. musicnn: pre-trained convolutional neural networks for music audio tagging. In *Late-breaking/demo session in 20th International Society for Music Information Retrieval Conference (LBD-ISMIR2019)*, 2019.

Jordi Pons, Oriol Nieto, Matthew Prockup, Erik M. Schmidt, Andreas F. Ehmann, and Xavier Serra. End-to-end learning for music audio tagging at scale. *ArXiv*, abs/1711.02520, 2017.

R128. Loudness normalisation and permitted maximum level of audio signals. Recommendation, European Broadcasting Union, aug 2011.

Joshua D Reiss and Andrew McPherson. *Audio effects: theory, implementation and application*. CRC Press, 2014.

Dario Rethage, Jordi Pons, and Xavier Serra. A wavenet for speech denoising. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5069–5073. IEEE, 2018.

Antony W Rix, John G Beerends, Michael P Hollier, and Andries P Hekstra. Perceptual evaluation of speech quality (pesq)-a new method for speech quality assessment of telephone networks and codecs. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 01CH37221)*, volume 2, pages 749–752. IEEE, 2001.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

Md Sahidullah and Goutam Saha. Design, analysis and experimental evaluation of block based transformation in mfcc computation for speaker recognition. *Speech communication*, 54(4):543–565, 2012.

J Schattschneider and Udo Olzer. Discrete-time models for nonlinear audio systems. 03 2000.

Thomas Schmitz and Jean-Jacques Embrechts. Nonlinear real-time emulation of a tube amplifier with a long short time memory neural-network. In *Audio Engineering Society Convention 144*. Audio Engineering Society, 2018.

Manfred R Schroeder and Benjamin F Logan. "colorless" artificial reverberation. *IRE Transactions on Audio*, (6):209–214, 1961.

Jeffrey Scott, Matthew Prockup, Erik M Schmidt, and Youngmoo E Kim. Automatic multi-track mixing using linear dynamical systems. In *Proceedings of the 8th Sound and Music Computing Conference*, 2011.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

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

Stanley Smith Stevens and Edwin B Newman. The localization of actual sources of sound. *The American journal of psychology*, 48(2):297–306, 1936.

Daniel Stoller, Sebastian Ewert, and Simon Dixon. Wave-u-net: A multi-scale neural network for end-to-end audio source separation. In *ISMIR*, 2018.

Fabian-Robert Stöter, Stefan Uhlich, Antoine Liutkus, and Yuki Mitsufuji. Open-Unmix - A Reference Implementation for Music Source Separation. *Journal of Open Source Software*, 4(41):1667, September 2019. doi: 10.21105/joss.01667.

Ron Streicher and Richard Burden. A practical analysis of sum-and-difference matrixing for stereo broadcast transmission systems. In *Audio Engineering Society Convention 78*, May 1985.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

Cees H Taal, Richard C Hendriks, Richard Heusdens, and Jesper Jensen. A short-time objective intelligibility measure for time-frequency weighted noisy speech. In *2010 IEEE international conference on acoustics, speech and signal processing*, pages 4214–4217. IEEE, 2010.

V. Valimaki, F. Fontana, J. O. Smith, and U. Zolzer. Introduction to the special issue on virtual analog audio effects and musical instruments. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(4):713–714, May 2010. doi: 10.1109/TASL.2010.2046449.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *9th ISCA Speech Synthesis Workshop*, pages 125–125, 2016a.

Aäron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in neural information processing systems*, pages 4790–4798, 2016b.

Andrei Vladimirescu. *The SPICE book*. Wiley New York, 1994.

Jonathan Wakefield and Christopher Dewey. An investigation into the efficacy of methods commonly employed by mix engineers to reduce frequency masking in the mixing of multitrack musical recordings. In *Audio Engineering Society Convention 138*, May 2015.

Daniel A. Walzer. Independent music production: how individuality, technology and creative entrepreneurship influence contemporary music industry practices. *Creative Industries Journal*, 10(1):21–39, 2017. doi: 10.1080/17510694.2016.1247626.

Yuxuan Wang, RJ Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, et al. Tacotron: Towards end-to-end speech synthesis. *arXiv preprint arXiv:1703.10135*, 2017.

Dominic Ward, Joshua D. Reiss, and Cham Athwal. Multitrack mixing using a model of loudness and partial loudness. In *Audio Engineering Society Convention 133*, Oct 2012.

Thomas Wilmering, David Moffat, Alessia Milo, and Mark B. Sandler. A history of audio effects. *Applied Sciences*, 10(3), 2020. ISSN 2076-3417. doi: 10.3390/app10030791.

Alex Wilson and Bruno Fazenda. 101 mixes: A statistical analysis of mix-variation in a dataset of multitrack music mixes. In *Audio Engineering Society Convention 139*, 10 2015.

Alex Wilson and Bruno Fazenda. Variation in multitrack mixes: analysis of low-level audio signal features. *Journal of the Audio Engineering Society*, 64(7/8):466–473, 2016.

Alex Wilson and Bruno Fazenda. Populating the mix space: Parametric methods for generating multitrack audio mixtures. *Applied Sciences*, 7(12):1329, Dec 2017. ISSN 2076-3417. doi: 10.3390/app7121329.

Alec Wright, Eero-Pekka Damskägg, and Vesa Välimäki. Real-time black-box modelling with recurrent neural networks. In *International Conference on Digital Audio Effects (DAFx-19)*, 09 2019.

Alec Wright, Eero-Pekka Damskägg, Lauri Juvela, and Vesa Välimäki. Real-time guitar amplifier emulation with deep learning. *Applied Sciences*, 10(3):766, 2020.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

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

Geng Yang, Shan Yang, Kai Liu, Peng Fang, Wei Chen, and Lei Xie. Multi-band melgan: Faster waveform generation for high-quality text-to-speech. *arXiv preprint arXiv:2005.05106*, 2020.

David T Yeh, Jonathan Abel, and Julius O Smith. Simulation of the diode limiter in guitar distortion circuits by numerical solution of ordinary differential equations. *Proceedings of the Digital Audio Effects (DAFx'07)*, pages 197–204, 2007.

Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *International Conference on Learning Representations*, 2018a.

Zhichen Zhang, Edward Olbrych, Joseph Bruchalski, Thomas J. McCormick, and David L. Livingston. A vacuum-tube guitar amplifier model using long/short-term memory networks. *SoutheastCon 2018*, pages 1–5, 2018b.