

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

LOGITECH EUROPE S.A.

MASTER THESIS

SCHOOL OF COMPUTER AND COMMUNICATION SCIENCES

Personalised Audio Enhancement

By Rayan DAOD NATHOO

EPFL Supervisor (LCAV)
Dr. Paolo PRANDONI

Logitech Supervisor
Dr. Milos CERNAK

August 2021 - February 2022

Acknowledgements

I would like to thank Milos Cernak and Paolo Prandoni for their trust and supervision. The recurring meetings, the interesting thoughts and the discussions saved me more than once and helped set the course for this project.

Damien Ronssin played a key role as well since this study is mostly a follow-up of the work he did for his Master Thesis. He also made himself available when I had questions to ask, and helped in the implementation of a real-time prototype at the end of this 6-month project. There is no doubt that he is a real asset to Logitech and I thank him a lot for his help.

I would also like to show my gratitude to Mikołaj Kegler, Neil Scheidwasser-Clow, Pierre Beckmann, and Gasser Elbanna for all the insightful discussions and all the interest you have shown on these Monday meetings. The topics discussed were not always easy to understand but still very enriching.

More generally, thanks to the other Fall 2021 interns who were there to help or to have good times, to Mélanie Poget for the internship organisation, to Jean-Michel Chardon for the internship program, and to Logitech as a company for that wonderful fulfilling experience.

As this internship and Master Thesis marks the end of my journey at EPFL, let me thank my parents from the bottom of my heart for the studies they offered me, for the love they gave me, and for the support they showed and still show every day. You are my role models and there is no wording to express it enough. I will do my best to keep making you proud.

Lausanne, February 18th, 2022

Rayan Daod Nathoo

Preface

As human beings, we physically experience life through our senses: we see, hear, smell, touch, taste, and more thanks to our body and the sensory organs that compose it. This gift allows us to feel every form of lives surrounding us, as well as ours. Tree leaves colliding, children laughing, birds singing, our breathing, ... together form complex audio scenes that our brain processes continuously every day.

On the other hand, we are on the edge of the digital age. Movies, video-games, virtual-worlds are thought and designed with the goal of immersing users into new experiences, and this is achieved by tricking our brain with signals close to the ones we usually receive from the physical world.

We inherently have, to some extent, the extraordinary faculty to focus on particular signal components. Additionally, signal processing techniques and machine learning are now well advanced to the point where they can help us to further filter these signals in order to only absorb what we want from the world. These techniques however still deserve to be explored and improved, and this is what I will try to do throughout my Master Thesis.

I hope you enjoy your reading!

Abstract

We live in a world in which customisation is increasingly central and achieving a higher focus state becomes more and more important. In any audio environment, many kinds of sounds occur, some more desirable than others depending on people's preferences at specific times. This study attempts to provide a solution to low-latency audio event manipulation, with an emphasis on the gaming use case. We propose a system able to manipulate footstep sounds in real-time in any audio environment. This system is a machine-learning-based time-frequency masking algorithm. It computes a time-frequency mask and multiplies it with the time-frequency representation of the input audio to yield the output audio signal. This machine learning algorithm was first trained and tested on six-second artificial mixtures, and obtained a 9.82 dB SI-SDR improvement. It was then tested on actual gameplay recordings on which it obtained less good but promising results. Potential reasons for that lower performance and future work are then discussed. Finally, an attempt on forty-one audio event classes is conducted, yielding a 9.81 dB SI-SDR improvement on four-second long mixtures of two audio events. Possible improvements are discussed as well.

Key words: audio source separation, audio scene, customisation, real-time, causal, footsteps



The École polytechnique fédérale de Lausanne (EPFL) is a public research university located in Lausanne, Switzerland. It specializes in natural sciences and engineering. It is one of the two Swiss Federal Institutes of Technology, with three main missions: education, research and innovation. The QS World University Rankings ranked EPFL as the 14th best university in the world across all fields in 2021, whereas The World University Rankings ranked EPFL as the world's 19th best school for engineering and technology in 2020.

Wikipedia

Logitech

logitech

Logitech is a Swiss company focused on innovation and quality, designing products and experiences that have an everyday place in people's lives. Founded in 1981 in Lausanne, Switzerland, and quickly expanding to the Silicon Valley, Logitech started connecting people through innovative computer peripherals and many industry firsts, including the infrared cordless mouse, the thumb-operated trackball, the laser mouse, and more. Since those early days, they have expanded both their expertise in product design and their global reach. For each of their products, they focus on how our customers connect and interact with the digital world. Logitech keeps design at the center of everything it creates, in every team and every discipline, to create truly unique and meaningful experiences.

Logitech website - About section

Contents

Table of Contents	vii
List of figures	viii
List of tables	ix
Glossary	xi
1 Introduction	1
2 Background	3
2.1 Vocabulary	3
2.2 Metrics	5
3 Feasibility Study with a Non-Causal Setup	8
3.1 Setup	9
3.1.1 Model	9
3.1.2 Mixture generation	10
3.1.3 Datasets	10
3.2 Experiments & Results	11
4 Causal & Real Time Setup	13
4.1 Model	14
4.1.1 Quick look at VoiceFilter Causal	14
4.1.2 Differences with our system	15
4.2 Data generation	18
4.2.1 Dynamic Mixing	18
4.2.2 The quest for more robustness	21
4.3 Training, validation, and testing setup	22
5 Gaming Application - Footsteps	25
5.1 Data	25
5.1.1 Foreground Datasets	25
5.1.2 Background Datasets	27
5.2 Experiments & Results	30

5.2.1		Footstep Focus Mode	30
5.2.2		Footstep Discovery Mode	36
6	Towards Real-Time Audio Scene Customisation		40
6.1	Data	40	
6.1.1	FSD Kaggle 2018	40	
6.1.2	2-foreground-ish mixture generation	40	
6.2	Experiment & Results	41	
7	General Discussion & Further Work		43
7.1	Data	43	
7.1.1	Mixture generation	43	
7.1.2	Taking advantage of known audio environments	44	
7.2	Model	45	
7.2.1	Integrating the shot learning methodology	45	
7.2.2	The slider-based model idea	46	
7.2.3	Other ideas worth exploring	47	
7.3	Summary	47	
8	Conclusion		49
A	Appendices		51
A.1	The removal task	51	
A.2	Hyperparameter tuning	52	
Bibliography			57

List of Figures

2.1	Illustration of a mixture as used in this thesis	4
2.2	Example data augmentations on an image	4
2.3	Example of a time-frequency masking process where $(c) = (a) * (b)$	5
3.1	Reproduced model architecture [17]	9
3.2	FSD Kaggle 2018 dataset - counts and manual verification ratios of the 41 audio event classes	11
3.3	FSD Kaggle 2018 dataset - duration boxplot of the 41 audio event classes	11
3.4	REVERB challenge corpus noises - time-frequency representations of the 9 classes of noise	12
4.1	VoiceFilter model architecture [33]	14
4.2	Illustration of the concatenation integration method used	16
4.3	Our VoiceFilter Causal model architecture	17
4.4	Two examples of mixture-target pairs	20
4.5	Two examples of only-background mixture-target pairs	21
4.6	Two adversarial examples of input-target pairs	22
4.7	Dynamic Mixing pipeline	23
5.1	<i>Ultimate Foley Sound Effects Collection</i>	26
5.2	<i>Foley Footsteps Sound Effects Library</i>	27
5.3	War Gaming Background Sounds - Original YouTube videos	29
5.4	Footstep Focus Mode - mixture-target pairs	31
5.5	Example result obtained on a background-foreground mixture	34
5.6	Footstep Focus Mode - Example result with a few seconds of <i>Call of Duty Warzone</i> gameplay (test vector 3)	35
5.7	Footstep Discovery Mode - mixture-target pairs	37
5.8	Footstep Discovery Mode - example result on a background-foreground mixture	39
7.1	Example of a user interface for audio event manipulation with sliders	46
A.1	Hyperparameter tuning - Comparison of different sets of parameters on Voice-Filter Causal	53

List of Tables

4.1	VoiceFilter Causal parameters used for the reported experiments	24
5.1	Footstep series dataset - training, validation, testing sets distribution	26
5.2	REVERB challenge corpus noises - training, validation, testing sets distribution	28
5.3	YouTube videos of war gaming background sounds - training, validation, testing sets distribution	28
5.4	Footstep Focus Mode - mixture generation parameters used	32
5.5	Footstep Focus Mode - background-foreground test mixtures results	32
5.6	Footstep Focus Mode - only-background test mixtures results	32
5.7	Footstep Focus Mode - chunk level results in background-foreground test mixtures	33
5.8	Footstep Discovery Mode - background-foreground test mixtures results	37
5.9	Footstep Discovery Mode - only-background test mixtures results	37
5.10	Footstep Discovery Mode - chunk level results in background-foreground test mixtures	38
6.1	Towards Real-Time Audio Scene Customisation - mixture generation parameters used	41
6.2	Towards Real-Time Audio Scene Customisation - 2-foreground-ish test mixtures results	42
6.3	Towards Real-Time Audio Scene Customisation - chunk level results on 2-foreground-ish test mixtures	42
A.1	Hyperparameter tuning - Parameter ranges given to Keras Tuner Hyperband . .	53

Glossary

AED Audio Event Detection. 3, 8

AES Audio Event Separation. 8

BSS Blind Source Separation. 1

CNN Convolutional Neural Network. 14–16, 43

CPU Central Processing Unit. 19

DSP Digital Signal Processing. 1, 6, 38, 46

FC Fully Connected. 14

GPU Graphics Processing Unit. 19, 23, 24

LR Learning Rate. 23

LSTM Long Short-Term Memory. 14, 15, 41

MAE Mean Absolute Error. 5, 7

ReLU Rectified Linear Unit. 16

SDR Signal-to-Distortion Ratio. 6, 7

SI-SDR Scale-Invariant Signal-to-Distortion Ratio. 7, 11, 14, 19, 23, 31, 33, 37, 38, 42, 43, 45, 49

SNR Signal-to-Noise Ratio. 5, 14, 19, 20, 24, 30, 31, 33, 34, 39, 41

STFT Short-Time Fourier Transform. 11, 14

TF Time-Frequency. 4, 5, 9, 14, 34, 35, 38, 39

1 Introduction

Blind Source Separation (BSS) is the process of isolating one or multiple sources from an input mixture without any or with few information about the sources or the mixing process. It has raised increasing interest over the last decades and is an active field of research in various tasks today. Speech Separation [13][14][33] is the task consisting in separating mixtures of overlapping speech signals. One typical example of an application is the cocktail party problem, where multiple people are talking at the same time and a listener is trying to follow one of them. The human brain has this natural faculty, to some extent, but this is a complex problem in Digital Signal Processing (DSP). Another BSS task is the Musical Source Separation task, consisting in separating or extracting particular instruments or groups of instruments from a musical input. One might want to remove the vocal signal from a music to generate a karaoke version for example, or to extract only the drums for practicing purposes, or to focus on specific parts for re-mixing or up-mixing goals. This study aims for another task, Audio Source Separation, more specifically Single-Channel Audio Source Separation, which consists in separating various types of sounds from a mono input mixture. This task has raised increasing interest this last few years [10][29].

In most digital activities such as gaming or video-conferencing, users do not have much control over the audio environment they experience and these do not always fit their preferences. In fact, one might prefer listening to particular types of sounds more than others at specific times. In a world where customisation becomes increasingly important, we are interested in manipulating one or several classes of audio events at the same time with low latency. The final goal is to improve the audio experience of the users by giving them more control over the sounds that surround them.

Logitech is a technology-oriented company highly interested in contributing to help its users. It is active in a broad range of applications such as gaming, video-conferencing, and live-streaming. The title of this thesis, *Personalised Audio Enhancement*, illustrates and sets the goal of this project which is meant to be a first step and a proof-of-concept in helping users to personalise their audio environments. During these 6 months, we first reproduced the work done by a recent paper [17] in order to assess the feasibility of the overall project

in an offline manner. Once we obtained conclusive results, we started experimenting with a (close-to-)real-time & causal setup. Driven by the needs of the company, we focused on one audio event class - *footsteps*. We also report preliminary results on 41 audio event classes.

We will start by a ***Background*** chapter to introduce the vocabulary and important concepts. Then, we will go through the two main phases of the project, namely the ***Feasibility Study with a Non-Causal Setup*** and the ***Causal & Real-Time setup*** that was used. After the causal and real-time setup definition, we will present the main experiments that were conducted, the first one related to gaming and the *footstep* audio event class - ***Gaming Application - Footstep Extraction***, and the second one being a first attempt at manipulating 41 different audio event classes with a single trained model - ***Towards Real-Time Audio Scene Customisation***. Last but not least, the ***General Discussion & Further Work*** chapter will recapitulate what we have achieved and what remains to be explored. Ultimately, the ***Conclusion*** chapter closes the thesis. An ***Appendices*** chapter is also made available to outline some of the additional experiments that have been conducted.

2 Background

This section introduces the vocabulary that will be used and the important concepts as well as the quantitative metrics that we will need. The reader is assumed to have basic knowledge in machine learning techniques to understand the few technical parts of this thesis. A non-technical reader can however read this thesis as well by skipping the technical parts.

2.1 Vocabulary

Audio event

An audio event is an event happening at the audio-level. Audio events are typically separated into classes, e.g *car*, *scissors*, *trumpet*, ... The most famous audio event dataset is AudioSet [7], an expanding ontology of 632 audio event classes and a collection of 2,084,320 human-labeled 10-second sound clips issued from YouTube videos. This dataset is however weakly labeled, which means that its audio clips do not come with precise timestamps, but with tags associated to the entire clips. For this reason it is mostly used in Audio Event Detection (AED) tasks rather than separation. A partially strongly labeled version [9] (with timestamps) was nevertheless released during the 2021 International Conference on Acoustics, Speech and Signal Processing (ICASSP 2021).

As explained in the introduction, our goal is to allow users to control their audio environment. To this end, we will often talk about audio event manipulation, which relates to audio event removal, extraction, or to in-between tasks (e.g partial removal).

Mixture

A mixture is, at its name suggests, a mixture of audio events. In this thesis and as in [2][17] a mixture is composed of a background audio signal (environmental sound for example) and one or more audio events on top.

An artificial mixture is a mixture created manually, by adding background and foreground audio events together, possibly with other parameters coming into play (gain, time-location, effects, ...).



Figure 2.1: Illustration of a mixture as used in this thesis

The time axis is represented horizontally, from left to right, and the amplitude vertically, bottom to top. The background audio is represented in light blue while the audio events (in the number of three here) are represented in green, yellow, and red.

On the same note, what we will call the target is the audio signal we desire at the output of some algorithm and the estimate is the actual resulting output of that same algorithm.

Data augmentation

Data augmentation consists in increasing the amount of elements in a dataset by adding slightly modified versions of them. This method acts as a regulariser, prevents over-specialisation and consequently helps a model to generalise to unseen data. The modifications applied to existing data can come in various forms, such as translations, gain-reductions, filters, ... Figure 2.2 shows different data augmentation techniques applied to an image.

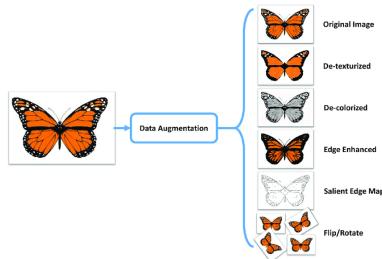


Figure 2.2: Example data augmentations on an image

Source: medium.com

Time-Frequency masking

Time-Frequency (TF) masking consists in multiplying the time-frequency representation of an audio input (a mixture for instance) by a matrix (the mask) having the same dimensions. The mask usually has amplitudes varying from 0 to 1, where a zero-element consequently nulls the corresponding element in the time-frequency representation of the input audio. This method has been widely used during the last decades in machine-learning-based source-separation tasks and showed good results, especially in speech separation [13][14][33].

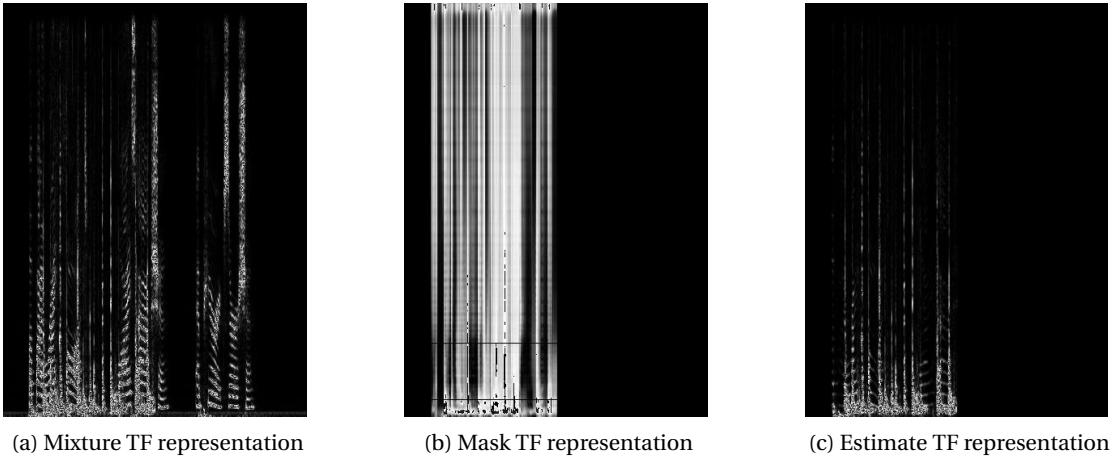


Figure 2.3: Example of a time-frequency masking process where (c) = (a) * (b).

The time-axis is represented horizontally, from left to right (0 - 6 seconds), and the frequency-axis is represented vertically, bottom to top (0 - 8kHz).

Embedding

An embedding is a relatively low-dimension vector used to represent a higher-dimension vector (image, audio signal, ...). It is widely used in the machine learning field as it allows to process rather complex inputs. Another key advantage of embeddings is that they can be learned and re-used across models. Examples of pre-trained embeddings for audio-related tasks are *d-vectors*, or *x-vectors* [26].

2.2 Metrics

In this thesis, when reporting our results, we will use different metrics to measure how well our trained models performed.

When mentioning the improvement of a metric (abbreviated with an *i* at the end of a metric abbreviation), we refer to the comparison between evaluation metric and baseline metric. The evaluation metric is computed between the target audio and the resulting estimate, and the baseline metric is computed between the target audio and the input mixture. Metrics measured in decibels are the higher the better.

We also report spectrogram metrics in this writing, namely the spectrogram Signal-to-Noise Ratio (spec_SNR) and the spectrogram Mean-Absolute-Error (spec_MAE). These are respectively the Signal-to-Noise Ratio (SNR) and the Mean Absolute Error (MAE) applied to the flatten TF representations of the signals, i.e unwrapped versions of the TF representations to form 1D-vectors.

Note that the reported metrics will be rounded to the second decimal. Also, for practical purposes, the metrics measured in decibels were clipped between -30 and 30 dB, levels at which humans start not to perceive any difference. Finally, each time there was a risk of encountering a division by zero, or a logarithm of zero, we added an epsilon value $\epsilon = 0.0005$ in the implementation of the corresponding metric for numerical stability. For instance, $\log(0)$ becomes $\log(0+\epsilon) = \log(0.0005)$.

In the following definitions, we define x to be the target signal, and \hat{x} to be the estimated signal. For more detailed explanations about these metrics, we recommend reading the later provided references.

Signal-to-Noise Ratio (SNR)

The Signal-to-Noise Ratio, or SNR, is a widely used metric in the DSP world and is computed as follows:

$$\text{SNR}(x, \hat{x}) = 10 \log_{10} \left(\frac{\|x\|^2}{\|x - \hat{x}\|^2} \right) \quad (2.1)$$

Basically, it is the ratio between the target power (or pressure) level, divided by the error power (or pressure) level. The error is defined to be $x - \hat{x}$ here, i.e the difference between the target signal and the estimated signal. A logarithm scale is often applied on top to mimic how humans perceive the sound, making it a perceptually-motivated metric.

Signal-to-Distortion Ratio (SDR)

The Signal-to-Distortion Ratio (SDR) was introduced in 2006 by Vincent et al. [31]. In their publication, they decomposed an estimated signal into four components:

$$\hat{x} = x + e_{interf} + e_{noise} + e_{artif} \quad (2.2)$$

In the above formulation, e_{interf} is the interference error term, e_{noise} is the noise error term, and e_{artif} is the artifact error term. We encourage interested readers to refer to the original publication for more computational details about these components, as this is not of much importance for our study. The SDR was hence defined as follows:

$$\text{SDR}(x, \hat{x}) = 10 \log_{10} \left(\frac{\|x_{target}\|^2}{\|e_{interf} + e_{noise} + e_{artif}\|^2} \right) = 10 \log_{10} \left(\frac{\|x_{target}\|^2}{\|\hat{x} - x_{target}\|^2} \right) \quad (2.3)$$

SDR is usually considered to be an overall measure of how good a source sounds. It however came with some issues, the main one being that the SDR could be cheated on and values could be artificially inflated by playing with the scale of the signals.

In 2018, Jonathan Le Roux thus introduced the Scale-Invariant Signal-to-Distortion Ratio (SI-SDR) [23], circumventing this issue by removing SDR's dependency on the amplitude scaling of the signal. The SI-SDR is defined as follows:

$$\text{SI-SDR}(x, \hat{x}) = 10 \log_{10} \left(\frac{\|\alpha x\|^2}{\|\alpha x - \hat{x}\|^2} \right) \quad (2.4)$$

with α a gain factor applied to the target signal x such that $\alpha = \underset{\alpha}{\operatorname{argmin}} \|\alpha x - \hat{x}\|^2$. It can be easily computed with

$$\alpha = \frac{\hat{x}^T x}{\|x\|^2} \quad (2.5)$$

Since then, the SI-SDR has been used in hundreds of publications and is the main metric we will use to report the results of our experiments.

Mean Absolute Error (MAE)

The Mean Absolute Error (MAE) is simply the averaged sum of absolute differences between some estimate signal and the corresponding target signal. It is defined as follows:

$$\text{MAE}(x, \hat{x}) = \frac{1}{n} \sum_{i=1}^n |\hat{x}_i - x_i| \quad (2.6)$$

Since we want the error between the target signal and the estimate signal to get closer to zero, the lower the MAE the better.

3 Feasibility Study with a Non-Causal Setup

The first step we took in this project was a feasibility study. To this end, we first started by reproducing the work of a recent paper close to our goal. We selected *Listen to What You Want: Neural Network-based Universal Sound Selector* [17] by Ochiai et al. (2020) to be the chosen study to reproduce for several reasons:

- It was a recent study that got submitted at InterSpeech 2020, which certified its quality,
- It based itself on a reknown model architecture - Conv-TasNet - which had proved its worth especially in speech separation tasks [13],
- It used an open-source 41-classes audio event dataset [6], in line with our global *Person-alised Audio Enhancement* idea,
- It focused on what they call the *selection* task, including the extraction and the removal tasks,
- It obtained promising results for mixtures of 2-to-5 audio events, and with the extraction/removal of up to 3 audio event classes at once.

Until now, research on audio event processing has mainly focused on Audio Event Separation (AES) and AED. AES's goal is to separate a mixture of audio events into signals containing each audio event classes that were present in the input mixture. Recent research has focused on the AES task [10][29], but these methods come with constraints that are the dependance on the number of outputs of the trained neural networks and the global permutation ambiguity at the separation's output, i.e. it is ambiguous which output corresponds to which audio event class, and this also affects the computation of the loss. AED's goal is to identify the class of each audio event in an input mixture with time-location information. One idea is therefore to combine AES and AED for the latter to help bypassing the constraints of the former. However, the limitation on the number of network output still remains. Additionally, possible artifacts at the output of the separation network could disturb the detection network, and combining two such networks is computationally expensive. The study we refer to attempts to circumvent all

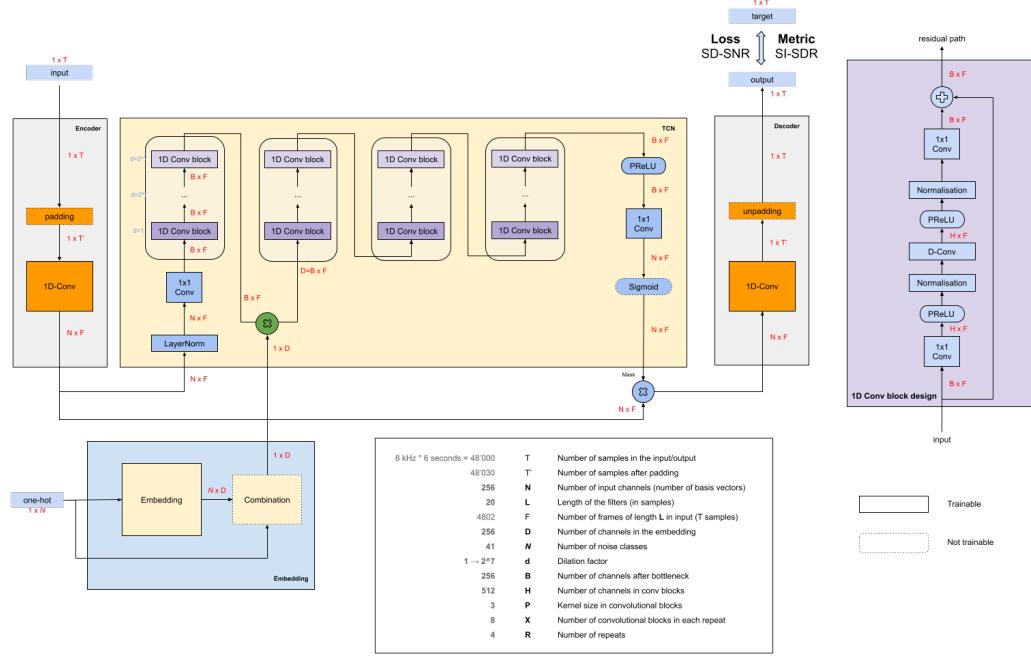


Figure 3.1: Reproduced model architecture [17]
It is advised to open the numerical version to be able to zoom in.

these constraints by focusing on audio event *selection*, thus by directly training a model to do all the work at once, i.e giving the expected output directly without passing by a separation nor a detection phase.

We will first introduce the setup by presenting the model and the data that was used, and then give the results that were obtained.

3.1 Setup

3.1.1 Model

The machine learning model that was used in this paper is based on Conv-TasNet, a TF masking based method using stacked dilated convolutional blocks and originally designed for the speech separation task [13]. As the aim is to choose which audio event classes to select, the authors combined the Conv-TasNet architecture with an embedding layer that receives a n-hot vector as a second input and yields an embedding vector. This vector is then element-wise-multiplied with the output of the first convolutional block of the Conv-TasNet architecture. Please refer to section 4.1.2.1 for a detailed explanation on hot vectors. Figure 3.1 illustrates the network and lists the parameters that were used.

3.1.2 Mixture generation

In order to generate mixtures of audio events, we used Scaper [24], a Python [30] library working with the PyTorch framework [19] that simply needs a dataset of foreground audio events and a dataset of background sounds, as well as mixing parameters to generate random mixtures. We used the same data generation parameters as mentioned in the reference paper and provided the foreground and background datasets presented below. We ended up with 50'000 training mixtures, 10'000 testing mixtures, and 10'000 validation mixtures. Note that the mixtures were mono signals sampled at 8kHz.

3.1.3 Datasets

3.1.3.1 FSD Kaggle 2018

The FreeSound Kaggle 2018 dataset contains 11,073 audio files put together by the AudioSet team [7] and the FreeSound team [5] in the context of a machine learning challenge on the Kaggle platform in 2018 [6]. The goal of this competition was to build an audio tagging system that can categorise an audio recording as belonging to one of a set of 41 classes drawn from the AudioSet Ontology.

These 41 audio event classes are the following: Acoustic guitar, Applause, Bark, Bass drum, Burping or eructation, Bus, Cello, Chime, Clarinet, Computer keyboard, Cough, Cowbell, Double bass, Drawer open or close, Electric piano, Fart, Finger snapping, Fireworks, Flute, Glockenspiel, Gong, Gunshot or gunfire, Harmonica, Hi-hat, Keys jangling, Knock, Laughter, Meow, Microwave oven, Oboe, Saxophone, Scissors, Shatter, Snare drum, Squeak, Tambourine, Tearing, Telephone, Trumpet, Violin or fiddle, Writing.

Figures 3.2 and 3.3 summarise the count and duration distributions per class as well as the manual verification ratios, corresponding to the ratios of recordings that were verified a second time by humans in each class [3].

One main feature of this dataset is that since it is collaboratively contributed, recording quality and techniques can vary widely. This can be seen as a drawback but also as a regularisation process, forcing the trained models to generalise to all types of inputs. Additionally, one can notice that there is a clear count and manual verification imbalance between classes.

Finally, this dataset is divided into a train set (approximately 18 hours) and a test set (approximately 2 hours).

3.1.3.2 REVERB challenge corpus noises

As described in chapter 2, a mixture is composed of a background sound and one or more audio events added on top. As the background dataset and similarly to the reference paper, we used the REVERB challenge [12] corpus noises. These are environmental noise, actually

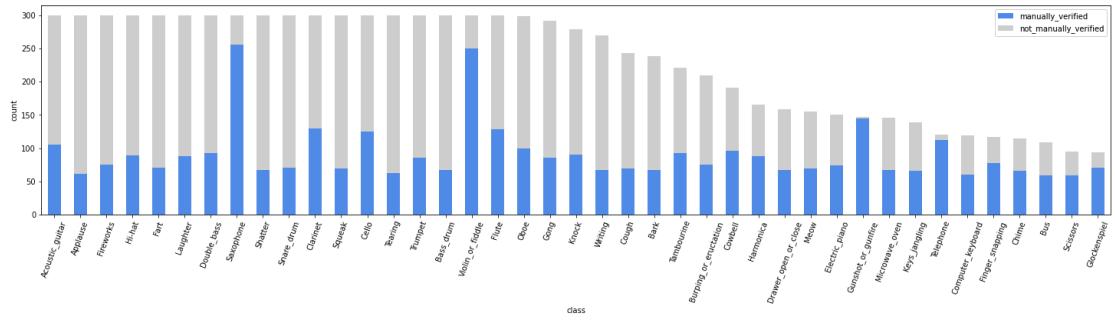


Figure 3.2: FSD Kaggle 2018 dataset - counts and manual verification ratios of the 41 audio event classes

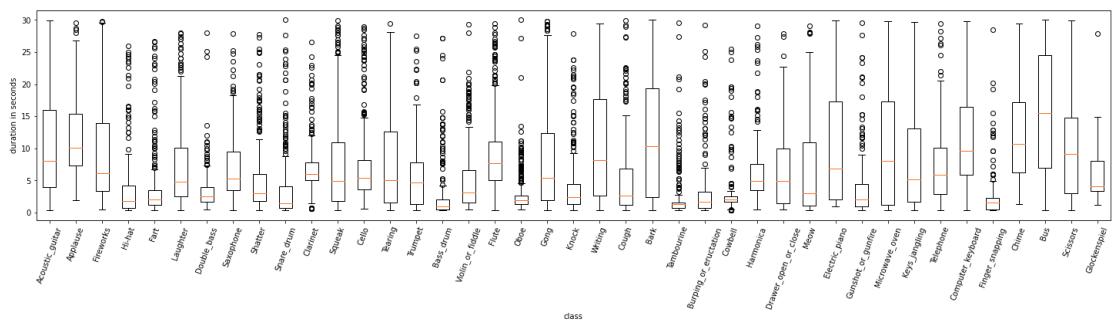


Figure 3.3: FSD Kaggle 2018 dataset - duration boxplot of the 41 audio event classes

air-conditioning systems recorded in several rooms, each having a different size. Figure 3.4 shows the Short-Time Fourier Transform (STFT) representations of example recordings of each of the 9 classes of noises that were recorded, corresponding to the 9 rooms that were used. For each room, 10 recordings of 30 seconds each are provided.

3.2 Experiments & Results

Due to the fact that our main goal was to assess the idea and to quickly start to experiment in a causal and real-time setup, we focused on the extraction and removal tasks of one audio event class. Moreover, we only trained our reproduced model with mixtures containing two or three audio events. We will indeed see hereafter that the model trained with mixtures containing two audio events yielded bad results, and that we consequently decided to stop at this point and to switch to a causal and real-time model.

In the end, we obtained encouraging results for mixtures containing two audio events, but worse results than expected on mixtures of three audio events. More precisely, we obtained an **average SI-SDR improvement of approximately 10 dB on 6-second mixtures of 2 audio events**. On 6-second mixtures of three audio events, we however only obtained an **average of 1 dB SI-SDR improvement** with model parameters presented in the reference paper, and an **average of 4 dB SI-SDR improvement** with our own tweaked parameters.

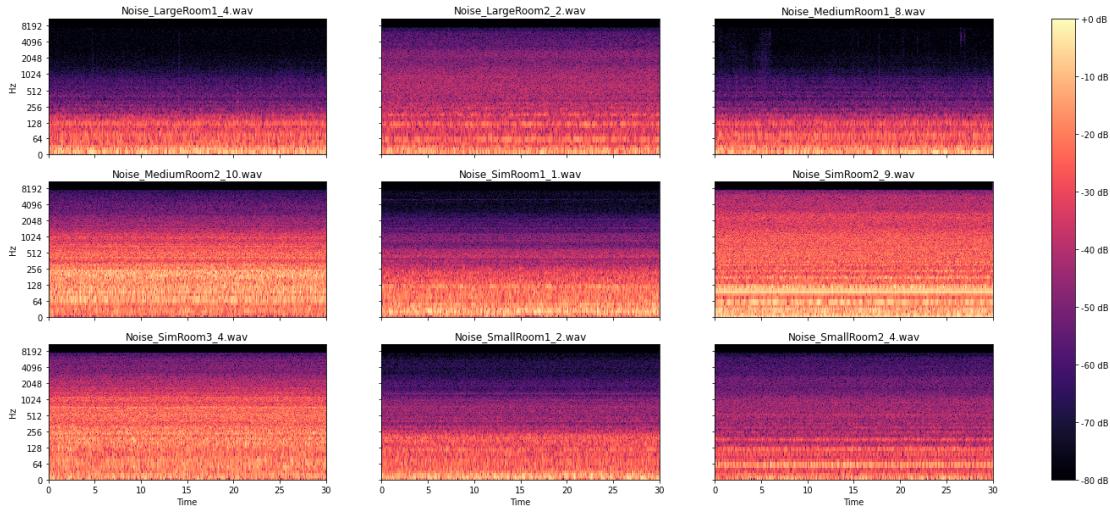


Figure 3.4: REVERB challenge corpus noises - time-frequency representations of the 9 classes of noise

We investigated this issue during a short period of time and concluded after discussing with the main author of the reference paper that the reason might be that we did not have had all the mixture generation parameters at our disposal. Another reason might be that we did not have the same computational power as the authors and therefore could not train our model for the same number of epochs (200 epochs announced). Also, some parameters mentioned in the paper were not the same as the one communicated by the author, so it might be the case that their reported setup was not the one ensuring reproducibility.

Still, the results we obtained with mixtures of two audio events were acceptable so we decided to go forward and to start experimenting with a causal and real-time machine learning model.

4 Causal & Real Time Setup

Once the feasibility of the task was assessed in a non-causal setup, we decided to go forward and to start experimenting with a new deep learning model, closer to satisfying end-user requirements. We therefore had to choose the model by considering two main constraints:

- Causality: The model should only process past frames, or past and current frames of the input audio mixture. Such a model has less contextual information at its disposal than a non-causal one, because when processing frame n , it has no information about frames $(n + 1), (n + 2), \dots$. One can also make the model *almost-causal*, in which case the system has access to frames $(n + 1), \dots, (n + k)$, k the number of *look-ahead* frames, allowing deeper context understanding. However, since the latter introduces additional delay at inference stage, one has to find an acceptable middle ground between model performance and look-ahead.
- Real-time inference: at inference stage, the model should be able to run in a few tens of milliseconds on a standard single core Computer Processing Unit (CPU). To this end, one has to carefully choose the model blocks to enable fast computation and find an acceptable middle ground between performance and model size in terms of number of parameters.

Motivated by the results of a previous intern at Logitech - Damien Ronssin - we used his model that we will call VoiceFilter Causal throughout this thesis, as well as his programming setup as starting points.

In this section, we will first present the machine learning model that was used, then we will describe in details how the data was generated, and finally we will discuss the experiments and their results.

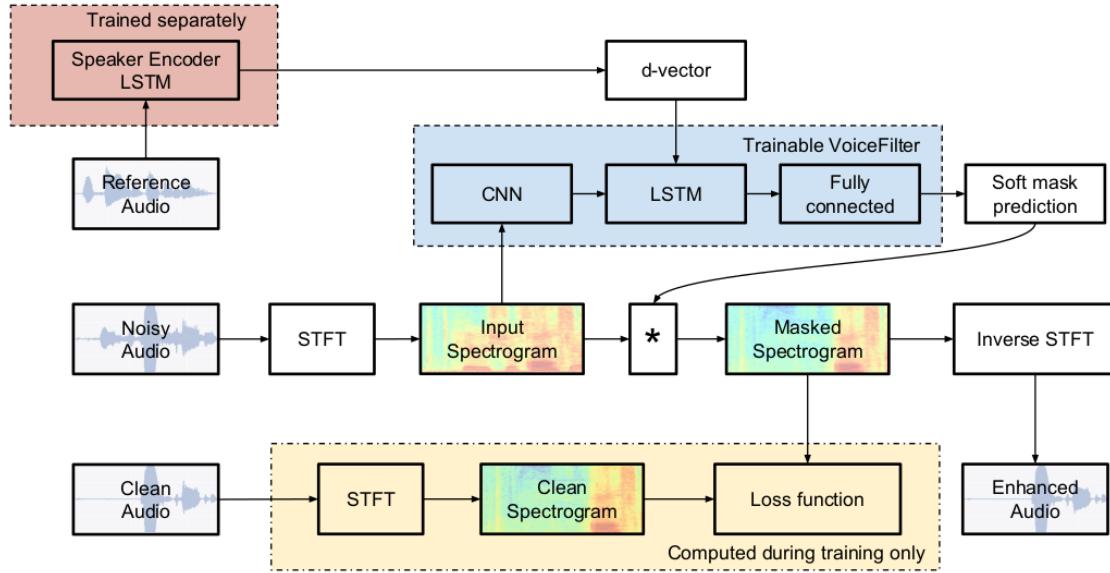


Figure 1: System architecture.

Figure 4.1: VoiceFilter model architecture [33]

4.1 Model

4.1.1 Quick look at VoiceFilter Causal

VoiceFilter Causal [22] is a causal deep learning model initially designed for the separation of mixtures of two different speakers. It is a modified version of *VoiceFilter* [33] - a non-causal model for speech separation, published by a Google Research team, and inspired from *VoiceFilter-lite* [32] - a follow-up work initially intended for Automatic Speech Recognition and published by the same Google team.

The model resulted in an overall **8.7 dB SI-SDR improvement** on mixtures of two overlapping speakers from Librispeech [18] mixed with random SNRs between -10 and 0 dB. It showed a total inference latency of 40 ms when run on a single core CPU, making it a model of choice for our task. More details can be found in Damien Ronssin's Master Thesis [22].

As described in figure 4.1, VoiceFilter Causal has an Audio-to-Audio model architecture but is trained on magnitude spectrograms. It is therefore composed of an encoder/decoder - chosen to be the STFT and its inverse here, and a separator module whose goal is to produce a mask. The separator module is a cascade of a Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM) layers and Fully Connected (FC) layers. This cascade produces a TF mask that is in turn multiplied by the encoded input to finally output the spectrogram of the resulting estimated audio.

We kept the same loss function as used in VoiceFilter Causal [22]: $L_{vf} = \left\| |X^{0.3}| - |\hat{X}^{0.3}| \right\|^2$, where

X is the magnitude spectrogram of the target signal, and \hat{X} is the magnitude spectrogram of the estimate signal.

4.1.2 Differences with our system

Even after conducting a short ablation study, we kept most of the blocks of VoiceFilter Causal, but a few components such as the second input and the mask activation function were adapted to our use-case. Figure 4.3 showcases the final model architecture that was used alongside the final parameters we chose for our experiments. Those will be discussed later in section 4.3.

4.1.2.1 User control input

The goal of the originally designed VoiceFilter [33] was to separate single-channel mixtures of overlapping speakers. To specify to the model which of the two speakers to focus on and to extract, a speaker embedding corresponding to that particular speaker was given to the model as a second input, and concatenated to the output of the CNN, before propagating through the LSTM layer and the rest of the network.

As opposed to the original task, we focused here on audio event manipulation. Using a pre-trained speech encoder thus did not make any sense since this was specific to voice signal discrimination. In order to adapt the architecture to our use-case and give to the network a vectorised representation of the target class we wanted to manipulate in a provided input mixture, we experimented two main techniques that we will describe hereafter: one-hot vectors and jointly-trained embeddings. Additionally, we will discuss how to integrate the second input with the other blocks of the model.

One-hot vector A one-hot vector is a vector of n 0s (zeros) - with n the number of categories to deal with - except at one single index where there is a 1 (one). In the case of multiple ones, we use the term *n-hot vector*.

Before going any further, one might ask oneself why did we not consider ordinal encoding, i.e only giving raw class indices as a second input to the network. The reason is that it has been shown in [1][20] that using one-hot encoding over ordinal encoding allows machine learning models to better deal with categorical data, especially when there is no order/hierarchy between categories, which was the case in our study.

In the context of VoiceFilter Causal, the idea is the following: similarly to the previously used speaker embedding and in the case of a model trained to handle multiple classes, the one-hot vector was generated according to the target and the corresponding input mixture, thus including a 1 at the index of the class we wished to manipulate. It was then concatenated to the output of the CNN layer and propagated to the LSTM layers. Each of these concatenated

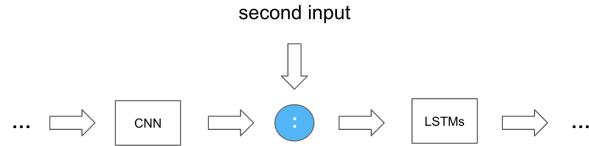


Figure 4.2: Illustration of the concatenation integration method used

units was therefore connected to all the subsequent units of the network. We consequently expected the training steps to link the concatenated one-hot vector and the target audio signal together via the model weights.

Jointly trained embedding For each class we desired to train the model on, we jointly trained a vector of weights that was in turn concatenated to the CNN output. By "jointly trained", we mean that this part of the network was trained during the same training steps as the other model blocks, leading this embedding to be adjusted to the needs of the network.

In addition to being the method of choice in [2][17], we believed that giving to the second input its own "learning space" - i.e not exclusively sharing the input mixture learning space - can only be beneficial in the long run. This is why we chose jointly trained embeddings to report our final results.

Integration method When adding a second input, the integration with existing blocks of the original model needs to be thought through. The most popular ones are concatenation and element-wise multiplication, the latter however requiring both elements to have the same shape (e.g in [17]). For practical reasons and encouraging results reported in the original VoiceFilter implementation and in [2], we used concatenation in our experiments. Similar to VoiceFilter, we chose the integration to happen right after the CNN layer, for time and frequency homogeneity reasons [33]. Figure 4.2 illustrates this paragraph.

4.1.2.2 Mask activation function

For uncertain reasons and for some time, we faced a numerical issue when training VoiceFilter Causal for particular tasks and with particular datasets. Switching from a Sigmoid to a Rectified Linear Unit (ReLU) activation function before the time-frequency masking step solved the issue. We feared that the training would take longer to converge or that the results would be different, but when comparing a removal model trained using Sigmoid (which never failed us in the removal task) and a removal model trained using ReLU, we did not notice significant differences. We therefore kept ReLU as the new mask activation function for all our next experiments.

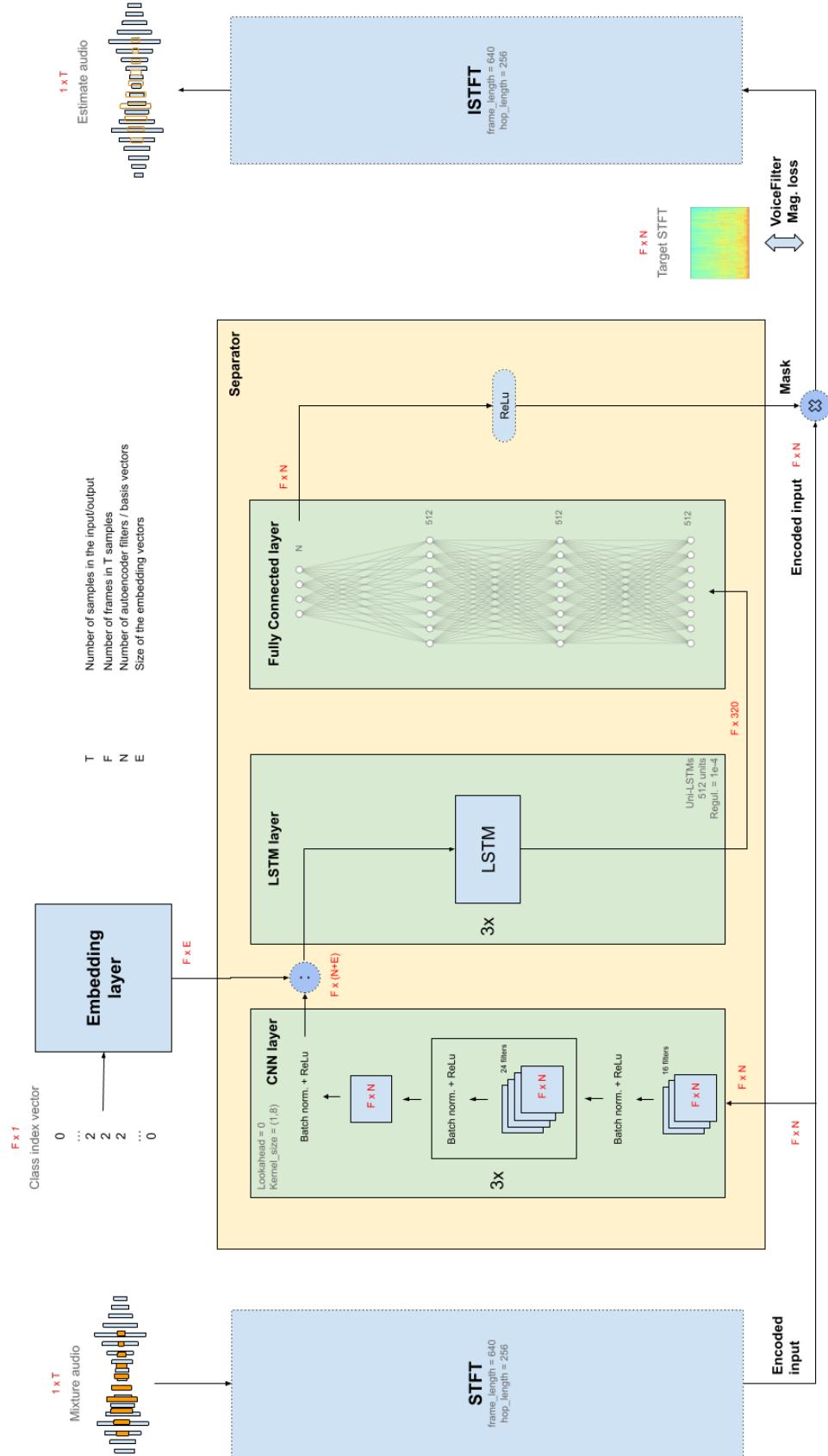


Figure 4.3: Our VoiceFilter Causal model architecture

4.2 Data generation

The method we selected for this project is a supervised-data-driven machine learning algorithm. This means that as opposed to a model-oriented one, we rely on data rather than the design of the network itself to achieve the desired task. The main inconvenient of this choice is that it requires the model to be trained on a large amount of data for it to understand the task and to generalise well, i.e to perform decently on unseen audio inputs.

Consequently, we decided to generate artificial mixtures and their corresponding targets by mixing together appropriate background recordings and foreground audio events, thanks to various randomly-chosen mixing parameters. This way, by leveraging the number of different combinations of background-foreground pairs and the wide range of mixing parameter values, we would be able to feed the network with large amounts of different data and to fully take advantage of the datasets that we had in our possession.

It is important to note that when defining the target of a particular mixture, we assumed that the background sound used to create the mixture did not contain the audio event of interest. That way, we could consider the background and the foreground separately, and build an appropriate target audio signal.

In this section, we will discuss how the mixture-target pairs we fed to the network were generated and the data augmentation techniques that we applied.

4.2.1 Dynamic Mixing

As mentioned before, the chosen approach heavily relies on data. Hence, we needed to generate a large number of different input mixtures in order for the model to understand and learn the desired task. To this end, we decided to use Dynamic Mixing.

Dynamic Mixing consists in generating mixtures *on-the-fly*, i.e continuously creating random new mixtures during the training phase, by selecting a random background audio segment, a random foreground audio segment out of the provided datasets, and random mixing-parameter values in provided ranges, at each step. This technique has many advantages:

- First, it does not require an offline computation of the mixtures prior to the training and only requires us to store the background and foreground audio files once, saving us computer space.
- Secondly, the number of possible combinations of background and foreground audio events together with the limitless number of possible mixing-parameter values allows us to generate an infinite number of different mixtures, which is in line with our data-driven approach.
- Also, it is reported in several publications [28] [34] that Dynamic Mixing brings signifi-

cant performance improvements (approximately 2 dB SI-SDR improvement) over offline mixing, thus increasing our chances to train an efficient model.

- Finally, since we based ourselves on the codebase put together by Damien Ronssin, a previous intern at Logitech, we pursued his work [22] and built the entire Dynamic Mixing pipeline using Tensorflow functions [15], allowing it to run directly in our Graphics Processing Unit (GPU), thus running very fast compared to a Central Processing Unit (CPU) data generation implementation.

Dynamic Mixing brings a lot of diversity to the mixture dataset we feed to the model. However, simply adding background and foreground audio signals together does not generate representative real-world mixtures. In order to improve the diversity of our mixture-set even more and to get closer to covering the large amount of mixtures that the trained model would actually encounter in the deployment phase, we added three main data augmentation techniques: random chunks, random SNR levels, and random overall gains. These methods were inspired by Scaper. Further data augmentation techniques will be discussed in chapter 7 about further works.

Random chunks

For each newly-generated artificial mixture and once the background and foreground audio files were chosen by the Dynamic Mixing process, three random parameters were drawn:

1. A random chunk length, selected from a range of possible values (e.g [2.0, 6.0] seconds).
2. A random part of the previously chosen foreground audio file, chosen according to the selected chunk length.
3. A random timestamp, to choose where to add the chosen chunk in the mixture, selected so that the entire chunk can fit.

Once these parameters were drawn, the background and foreground signals were merged in accordance. This data augmentation technique allowed us to generate more complex mixtures with both only-background parts and background-foreground parts, giving a sense of on/off switching to the model.

Random SNR levels

In order to teach the model to adapt to any kind of data we needed the background and the foreground to have different relative levels. Indeed, if we take the gaming example with footsteps as the foreground audio event, sometimes the background is very loud and footsteps are difficult to hear. Some other times, the footsteps are very loud compared to the



(a) First example of mixture-target pair, in which we desire the model to output only the foreground audio event of interest

(b) Second example of mixture-target pair, in which we desire the model to output the foreground audio event of interest when it is detected, and to output the background otherwise.

Figure 4.4: Two examples of mixture-target pairs

The background audio is represented in blue, the foreground audio event in orange. The dotted lines represent what is not desired in the resulting outputs.

environmental sounds. We therefore decided to add a random SNR level parameter, chosen within a predefined range. Once that SNR level was chosen, we applied the required gain to the foreground audio event, making it quieter/louder than the audio background.

The gain was computed using the following formula:

$$\text{gain} = \sqrt{10^{(SNR_{goal} - SNR_{raw})/10}} \quad (4.1)$$

With SNR_{goal} being the aimed SNR between the foreground event and the background signal, and SNR_{raw} being the SNR computed on raw foreground and background signals, i.e when leaving them untouched.

Note that the applied gain was computed at the chunk-level. Indeed, what interested us here was the SNR level between the active part of the foreground audio event and the background audio, and not the entire signals.

A main drawback from using an SNR-based gain is that the SNR formula does not compute the actual human-perception of loudness since it is basically a ratio of powers. It is therefore not very consistent depending on the audio event class that we focus on. To tackle this issue, Scaper [24] uses PyLoudNorm [27], a Python library computing the loudness using a filter-based method. Due to time limitations, for efficiency reasons in the Dynamic Mixing process, and because the results were most of the time reasonable during listening sessions, we kept the SNR as the main formula. It is nevertheless encouraged to keep trying to improve this process. The final goal being to improve user experience, it is important to reason in a perceptual-motivated way.



(a) Example of only-background mixture-target pair, in which we desire the model to output a zero-signal since the foreground audio event of interest is not included.

(b) Second example of only-background mixture-target pair, in which we desire the model to output the input signal.

Figure 4.5: Two examples of only-background mixture-target pairs
The dotted line represents what is not desired in the resulting output.

Random overall gain

Since in practice the input audio is not always normalised to have maximum absolute amplitudes close to 1, we decided to apply a random overall gain at the end of the mixture generation process to simulate quiet and loud mixtures.

4.2.2 The quest for more robustness

Even with the *random chunk* feature, generating mixtures only using the aforementioned data augmentation techniques and feeding those to a machine learning model would mostly train that model to deal with favorable conditions, i.e dealing only with input mixtures containing the audio event of interest. However, since the goal was to develop a user-experience-level model, we needed to train our model to adapt to unfavourable conditions too. To this end, we also generated different adversarial mixtures and their corresponding targets. More precisely, we added mixtures not containing the audio event class of interest, and mixtures associated with “adversarial” second inputs when those were used.

Adding only-background mixtures

The first idea was to generate mixtures composed of background sound only, and to associate them with targets chosen in accordance (zero-signals or the inputs themselves for example). Figure 4.5 illustrates that data generation technique.

Adding more adversarial user-control inputs

When a second input is accompanying the mixture, e.g a one-hot vector or a jointly trained embedding, we can picture it as an end-user pressing a button on their keyboard to *activate* the model for a particular audio event class. However, such a user will most probably press this key at times where no audio event of the class of interest will occur. Such adversarial inputs therefore need to happen during the training phase to teach the model to deal with these scenarios. Figure 4.6 illustrates this idea.

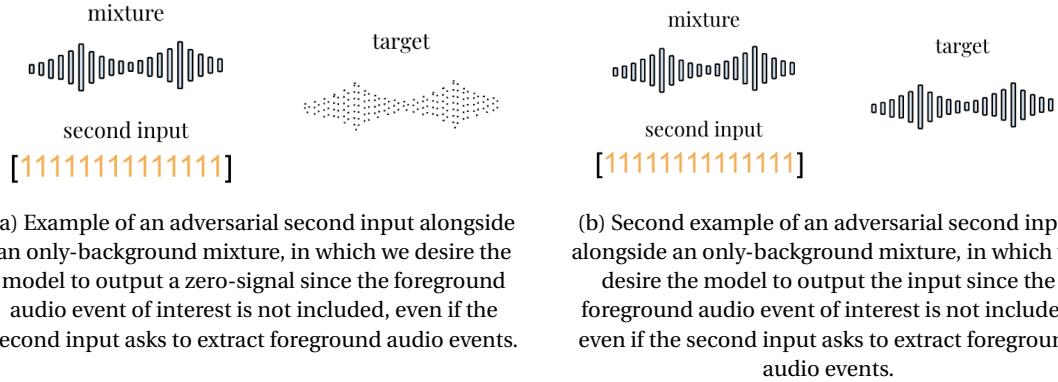


Figure 4.6: Two adversarial examples of input-target pairs

Note that it will be especially useful when training a model for multiple classes of audio events, in which case an example of an adversarial input-target pair can be a mixture containing a foreground event from some class and the second input could require another class, not in that mixture, hence expecting the model not to extract anything.

Due to time limitation and because our setup was not yet designed for multiple-audio-event mixtures, we could not explore that idea further so it will *not* be discussed in this thesis. We leave it as a future work for upcoming multi-class models.

4.3 Training, validation, and testing setup

As mentioned in the Data section, we used Dynamic Mixing to continuously generate different mixtures during the training phase. Our training mixture-set was consequently of theoretical infinite size, which is an important asset of our training pipeline. The batch size was chosen to be 8, as this is the parameter that worked the best in early experiments.

Our validation set was composed of 400 batches in total. This corresponds to 1/5 of the number of batches trained on during each epoch. Note that since our training set was of infinite size, an epoch as defined in that thesis is not the same as a typical epoch one might encounter in typical machine learning literature, i.e cycling through the training set once. Indeed, here we define an epoch as 2000 batches (or steps) on which the model is trained on. Consequently, our model is trained on different generated mixtures every epoch, which is reported in several publications to highly increase the performances [28] [34].

Our testing set was composed of 3200 mixtures in total, which is actually the same size as the validation set (400 batches * batch size 8), except that here we evaluated our models on individual mixtures and computed the overall mean of each metric before reporting the results.

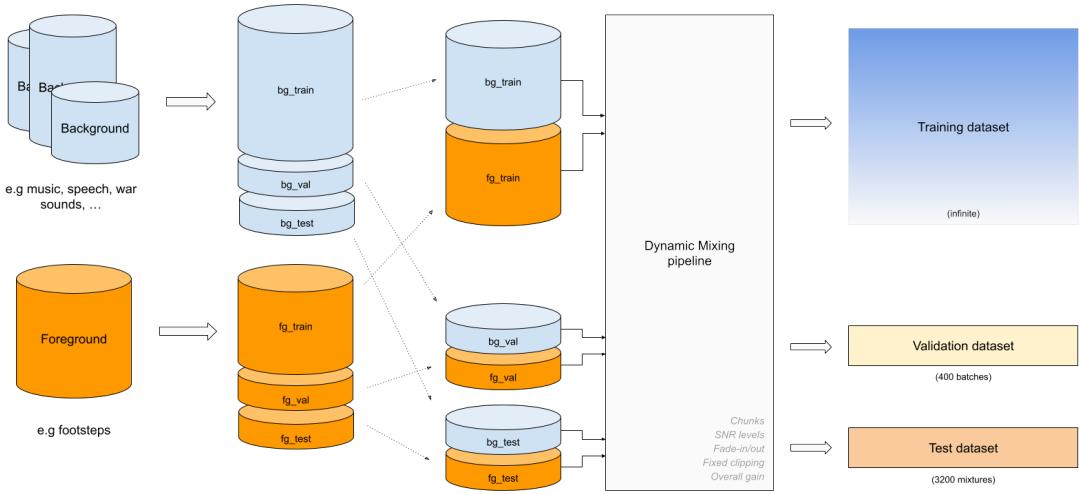


Figure 4.7: Dynamic Mixing pipeline

Each time we evaluated a model with additional types of input data (e.g. only-background mixtures, adversarial second input, ...) the testing and validation sets were built in accordance, by respecting the ratios of data types originally fed to the training model. Figure 4.7 illustrates how the training, validation, and test sets were generated using Dynamic Mixing.

On the model side, table 4.1 lists the parameters that we chose for both experiments. Notice that they are the same as reported in VoiceFilter Causal [22] which focused on the speech separation task. Section A.2 of the Appendices chapter however highlights the subtle but highly efficient hyperparameter search that we tried and that could be again conducted in the future. Since we often changed tasks during this 6-month project and since we did not start any in-depth-optimisation phase, we decided to keep the original parameters for now.

We chose Python [30] as the programming language, Tensorflow [15] as the framework for all our experiments, and we ran each training on a Nvidia GeForce GTX 1080 Ti GPU. Similar to the original VoiceFilter implementation, we used Gradient Clipping with a maximum l_2 norm of 0.25, and we selected the Adam optimizer [11] with a Learning Rate (LR) of 0.0005 and Tensorflow's default betas and epsilon parameters ($\beta_1=0.9$, $\beta_2=0.999$, $\epsilon=1e-07$). The learning rate was halved if the validation objective metric (SI-SDR, as commonly used in recent literature [2][8][17], on background-foreground mixtures) did not improve during 15 epochs. Finally, the trainings were stopped if there was no improvement during 20 epochs.

4.3.0.1 Training Reproducibility

During the different phases of that project, we wanted to be able to compare the results more accurately across trainings. We realised that even with a fixed global seed, classical randomisation functions in Tensorflow do not output the same results during the data generation.

STFT	Frame length	640
	Hop length	256
	Number of frequency bins	1024
CNN	Number of convolutions	4
	Number of filters in each conv.	32
	Kernel size	(1, 8)
	Normalisation type	Batch
LSTM	Number of LSTM layers	3
	Number of units in each	512
	l2 regulariser	0.0001
	Normalisation type	None
FCNN	Number of layers	3
	Number of units in each	512
Mask	Final mask activation	ReLu
Total	7.5M trainable parameters	

Table 4.1: VoiceFilter Causal parameters used for the reported experiments

To tackle this issue, each time a random number was required in the data pipeline, we used Tensorflow's *stateless* random functions, whose goal are to produce the exact same result every time it is called with the same seed, allowing to produce same chunks, SNRs, gains, etc. across trainings. We fed it with a predefined tensor of random seeds, generated outside the Tensorflow data generation pipeline, therefore reproducible.

Long story short, we used a global seed to produce a reproducible dataset of seeds used in Tensorflow stateless randomisation functions.

Note: Even with that method, the same training ran twice yielded slightly different training curves. Our guess is that since we are also asking Tensorflow to generate the data in an *autotune* way, it might cause a slight mixture disorder depending on what GPU process is faster at some time t . Nevertheless, the order is globally kept and the mixtures are in the end the same across different runs.

5 Gaming Application - Footsteps

After the non-causal feasibility study and after discussing with interested parties at Logitech, we decided to focus on the *footstep* audio event class in the gaming context.

In this chapter, we will first present the data that we used for this new task, then introduce the conducted experiments and their results.

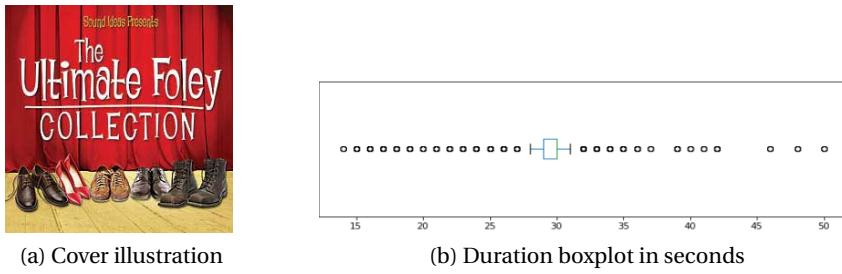
5.1 Data

To achieve the aforementioned goal and in order for the model to generalize well to any kind of footsteps, we needed foreground audio data (footstep audio recordings here) and background audio data to artificially create our mixtures. Again, we chose the background datasets with the requirement of not containing any footstep sounds, so that we could consider the background and the foreground audio of one particular mixture separately and build the audio targets in accordance.

In this part, we will present the foreground datasets and the background datasets that we selected for this application. A random 60% section of each dataset was selected for the training phase, a 20% section was chosen for the validation steps, and the remaining 20% were reserved for the testing phase.

5.1.1 Foreground Datasets

A review of the different available footstep datasets was thus conducted, based on criterions such as the dataset size, the variety of recording conditions (e.g the materials stepped onto, subject genres, movement speeds). We finally bought the following two: the *Ultimate Foley Sound Effects Collection* and the *Foley Footsteps Sound Effects Library* from sound-ideas.com.

Figure 5.1: *Ultimate Foley Sound Effects Collection*

	Split	Duration in hours	# files
Raw footstep series	Train	7	834
	Val	2.3	278
	Test	2.3	279
Reverbed footstep series	Train	1	115
	Val	0.33	38
	Test	0.33	39
Stairs footstep series	Train	0.75	112
	Val	0.3	38
	Test	0.25	38
Total	Train	8.75	1071
	Val	2.95	354
	Test	2.9	356

Table 5.1: Footstep series dataset - training, validation, testing sets distribution

5.1.1.1 Footstep series

This dataset we purchased on sound-ideas.com was composed of more than 2000 footstep series of 30-second average duration each (figure 5.1), i.e 15 hours of recording in different conditions: walking/running/shuffling, single people / 2 people / groups of people, bare feet / boots / dress shoes / ... These series were moreover recorded on a large number of materials: marble, wood (hollow and hard), linoleum, cement, grass (wet and dry), dirt, carpet, sand, forest, gravel, mud, snow, and bleachers. Footstep sounds while walking on stairs were included as well, alongside reverbed footsteps sounds. It was initially sold as a sound effect library for movie soundtracks, but we concluded that it filled the criterions the best.

During the preprocessing part, we removed the footsteps series recorded on *Sand* and *Snow*, considered as outliers. Indeed we realised that these footstep series were very different than the ones we would encounter in video-games of interest. We also removed the stereo files since we are focusing on single-channel audio signals and because these recordings were already in the dataset in a mono format. This way, we avoided having duplicates that may happen to be in the training and the validation or testing split, thus falsing the evaluation. The training, validation, and testing splits of this dataset were distributed as depicted in table 5.1.

5.1.1.2 Single footsteps

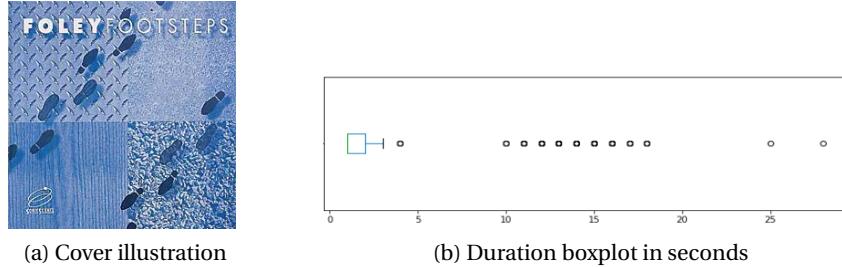


Figure 5.2: *Foley Footsteps Sound Effects Library*

The second footsteps dataset we purchased on the same website is the *Foley Footsteps Sound Effects Library*, and was composed of 700 recordings, mostly single footsteps. Naturally, they are in average much shorter than footstep series (figure 5.2). Those were recorded on many different materials as well (cement, wood, grass, gravel, metal, snow, ...) and in different conditions (walking/runnung/jumping/stairs-climbing, men/women, different types of shoes, ...). However, due to their nature, those recordings required us to generate footstep series before being able to integrate the dataset in the data pipeline. Note that this is similar to how game-developpers do, it would therefore be interesting to feed the network with similar footstep series as it would hopefully generalise to this kind of audio event. We did not have the time to investigate this idea further and will leave it as a possible further work. More details can be found in chapter 7.

5.1.2 Background Datasets

Our first thought when looking for background audio was to search for video-game background sounds that did not contain the audio event of interest. For example, when we focused on the *helicopters* audio event class at the beginning of the project, this was rather simple since we only had to get the soundtrack of a game that did not have any helicopters in it, e.g hours of gameplay of any *Assassin's Creed* game, which can be easily found on YouTube for instance. However, when the foreground audio event class is *footsteps* and that we focus on war video-games, we realised that it was not anymore that easy and decided to build a background dataset composed of multiple "sub-datasets" of different types: music, speech, random noise, ... We hypothesise that despite not being exactly similar to the background sounds of interest, this diversity would help the model to generalise to any type of background sounds. This section describes the background "sub-datasets" that we used to create the final bigger one.

5.1.2.1 REVERB challenge corpus noises

As in the non-causal feasibility study, we kept the REVERB challenge [12] corpus noises as one of the background datasets. Please refer to section 3.1.3.2 for a more complete description.

	Duration in hours	Number of files
Training	0.5	54
Validation	0.15	18
Testing	0.15	18

Table 5.2: REVERB challenge corpus noises - training, validation, testing sets distribution

	Duration in hours	Number of files
Training	1.5	838
Validation	0.5	280
Testing	0.5	280

Table 5.3: YouTube videos of war gaming background sounds - training, validation, testing sets distribution

Table 5.2 shows how the splits were distributed.

5.1.2.2 MUSAN

The MUSAN dataset [25] is a corpus composed of music (60 hours), speech (42 hours), and various noises (6 hours) for a total of approximately 109 hours of data. The music part is composed of Western art music (e.g., Baroque, Romantic, and Classical) and popular genres (e.g., jazz, bluegrass, hiphop, etc). The speech part is composed of free public domain audio books from Librivox and US government (federal and various states) hearings, committees and debates that are believed to be in the US Public Domain. Since the noises part contains footsteps, we did not use that category of the dataset. As it was already given as single-channel files sampled at 16kHz, we only had to split the *music* and *speech* folders into training, validation, and testing sets according to the ratios previously mentioned.

5.1.2.3 War Gaming Background Sounds

Since we were mostly interested in the gaming use-case, more particularly in war video-games such as *Call of Duty Warzone*, the *Battlefield* saga, etc, we were also looking for background audio recordings similar to war environments, *without* footsteps of course. After several random searches online, we stumbled across YouTube videos of war-video-game game-plays recorded in spectator mode, far from the battlefield itself, thus (mostly) without footsteps. These 2 hours and 30 minutes of videos were converted into audio files, downsampled to 16kHz, and chopped into six-second recordings before being split into training, validation, and testing sets. Table 5.1 showcases how the splits were distributed.

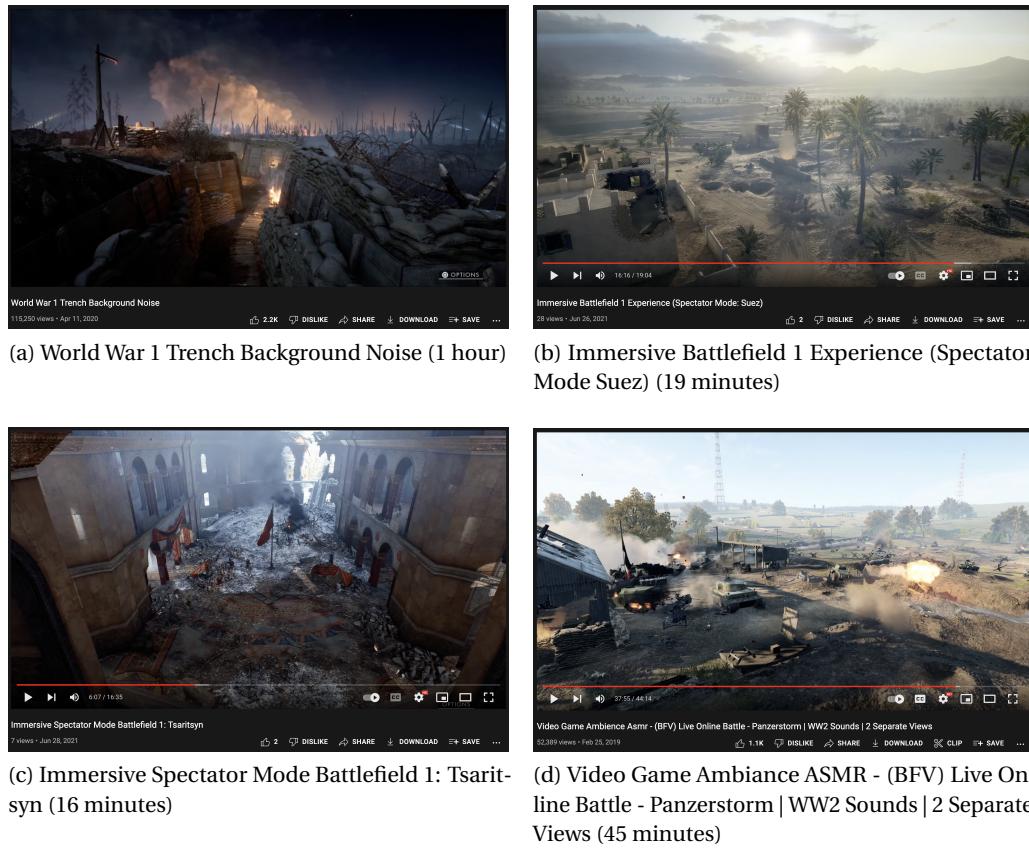


Figure 5.3: War Gaming Background Sounds - Original YouTube videos

5.2 Experiments & Results

Lots of experiments were conducted during this phase of the project, involving different mixture generation and different model parameters. We will cover the experiments with the latest and best results and will also divide this section into two distinct tasks: *Footstep Focus Mode* and *Footstep Discoverer Mode*.

It is important to note that all the experiments were conducted with single-channel mixtures sampled at 16kHz. The reason is that the project was mainly meant to assess the feasibility of the overall *Personalised Audio Enhancement* idea. We therefore decided to keep the input as simple as possible in order to keep the model relatively small and adapted to the hardware at our disposal. Finally, the mixtures were 6-second long for more context understanding.

5.2.1 Footstep Focus Mode

The first task and the one we spent the most time experimenting on during this project is the *focus mode* for footstep extraction. The idea was to allow a user, once the model activated, to be able to only focus on the footstep sounds around him. A more concrete example is the following:

You are playing Call of Duty - Warzone, your character has been hit and you are hiding in a house, not moving and waiting to heal. You activate the footstep focus mode allowing you to silence every surrounding sounds that are not footsteps, hear the enemy coming your way, even in a noisy environment, and use this information at your advantage. Thank you focus mode!

More formally, we asked the model's output to fulfill the following requirement:

$$\text{output} = \begin{cases} \text{footsteps}, & \text{if footsteps detected} \\ \text{zero-signal}, & \text{otherwise} \end{cases}$$

The mixture-target pairs were designed as shown in figure 5.4.

Table 5.4 summarises the mixture generation parameters we used, but here are a bit more details:

- We chose the SNR range to be [-10, 10] dB to cover a maximum number of scenarios,



Figure 5.4: Footstep Focus Mode - mixture-target pairs
The dotted lines represent what is not desired in the resulting output.

from low-level-footsteps with high-level-background to high-level-footsteps with low-level-background environment.

- The overall gain range was in turn chosen to teach the model to adapt to any global gain of any mixture. After listening and analysing some recorded gameplays of video-games of interest, we concluded that the [0.1, 1.0] was the one to go with.
- We included only 30% of only-background mixtures, because we were more interested in having a good footstep extractor than a good background remover. Still, this was important for the final goal so 30% seems like a good choice that showed motivating results.
- Finally, we did not train this model with a second input as it did not yield better results than without it during our experimentation phase. Nevertheless and as brought up in the section 4.1.2.1, second inputs will mostly be used in models trained to handle multiple classes.

The best model we obtained was trained during 150 epochs (20 hours). Tables 5.5 and 5.6 summarise respectively the results obtained on background-foreground mixtures and the results obtained on only-background mixtures generated with the test splits of our datasets. In each of the aforementioned tables, the metrics are the ones defined in chapter 2 (*Background*), namely the Signal-to-Noise Ratio (SNR), the Scale-Invariant Signal-to-Distortion Ratio (SI-SDR), the Spectrogram SNR, and the Spectrogram Mean-Absolute-Error. The reported results were averaged over the 3200 generated test mixtures. If we reason at the mixture level, the *Baseline* column values were computed between the target and the input mixture itself, the *Evaluation* column values were computed between the target and the output of the model, and the *Improvement* column values represent what we gained from the *Baseline* to *Evaluation*.

Recall that the training parameters we chose were defined in the previous subsection.

Let's first start by commenting on the reported SI-SDR_i for the only-background mixture validation set. The reason why the baseline and the evaluation SI-SDR_i values are stuck at 30.0 dB is because of three factors combined: the scaling invariant nature of the SI-SDR [23], the fact that we compare audio mixtures with targets being zero-signals, and the fact that decibel metrics were clipped at -30 and 30 dB as mentioned in section 2.2. Hence, we will look at the

Foreground audio events dataset	Footstep series dataset
Background sounds datasets	REVERB corpus challenge noises MUSAN speech MUSAN music YouTube War Gaming Sounds
Chunk length range	[2.0, 6.0]s
SNR level range	[-10, 10] dB
Overall gain range	[0.1, 1.0]
Only-background mixtures	30%
Second input	No

Table 5.4: Footstep Focus Mode - mixture generation parameters used

Metric	Baseline	Evaluation	Improvement
SNR (dB)	-2.38	8.4	10.78
SI-SDR (dB)	-2.38	7.44	9.82
spec_SNR (dB)	-2.42	8.4	10.82
spec_MAE	0.14	0.04	0.1

Table 5.5: Footstep Focus Mode - background-foreground test mixtures results

Metric	Baseline	Evaluation	Improvement
SNR (dB)	-30.0	-6.79	23.21
SI-SDR (dB)	30.0	30.0	0.0
spec_SNR (dB)	-30.0	-10.14	19.86
spec_MAE	0.40	0.00	0.40

Table 5.6: Footstep Focus Mode - only-background test mixtures results

Metric	Baseline	Evaluation	Improvement
SNR (dB)	-0.04	8.42	8.46
SI-SDR (dB)	-0.04	7.47	7.51
spec_SNR (dB)	-0.09	8.42	8.51
spec_MAE	0.13	0.058	0.075

Table 5.7: Footstep Focus Mode - chunk level results in background-foreground test mixtures

SNRi row for this type of mixtures, which is more meaningful here because of its scale-variant nature.

Looking at the background-foreground mixture results in table 5.5, we observe a Scale-Invariant Signal-to-Distortion Ratio improvement of **9.82 dB**. This is a very good start as we can assume that the model successfully distinguished the footsteps from the different background sounds. By listening to generated test mixtures and their estimates, we can confirm this assumption.

If we now compute the same metrics but this time by only looking at the chunks, i.e the 2-to-6-second part of the 6-second mixture where the footsteps are actually added, we obtain the results depicted in table 5.7. This is again very encouraging as it means that the improvement is not only due to the model setting the only-background part of the background-foreground mixture to a zero-signal, i.e the samples not in the chunk range, and only giving us the chunk part of the mixture; this is actually responsible for $9.82 - 7.51 = 2.31$ dB. Instead, the model actually extracts the foreground from the mixture with a **chunk improvement SI-SDRi of 7.5 dB**, and this was what we targeted.

Regarding the only-background mixtures results in table 5.6, we observe a Signal-to-Noise Ratio improvement of **23.21 dB** on 6-second only-background mixtures. This means that only-background mixtures, whose targets are zero-signals here, were not completely set to zero by the model. Since we do not have an improvement of 30 dB or more, this further means that the model interpreted parts of the background audio signals as footsteps, and therefore kept those in the output. Still, this is very satisfying as there is a difference of almost 16 dB between the SNR improvement for background-foreground mixture chunks and the SNR improvement for only-background mixtures. Besides, when listening to the output audio signals of the only-background test mixtures, the output remaining noise is barely audible. We can therefore consider this only-background scenario as successful in our setup.

One could ask oneself why the validation objective metric was chosen to be only the background-foreground SI-SDRi, and did not take into account the only-background mixtures as well. The reason is the following: choosing the objective metric as the mean between the background-foreground and only-background metrics did not yield good results because of a high variance in only-background mixtures. It actually prematurely triggered an early-stopping. We consequently decided to keep the objective metric to only consider the background-foreground mixtures. Nevertheless, it is important to remind that those mixtures are generated

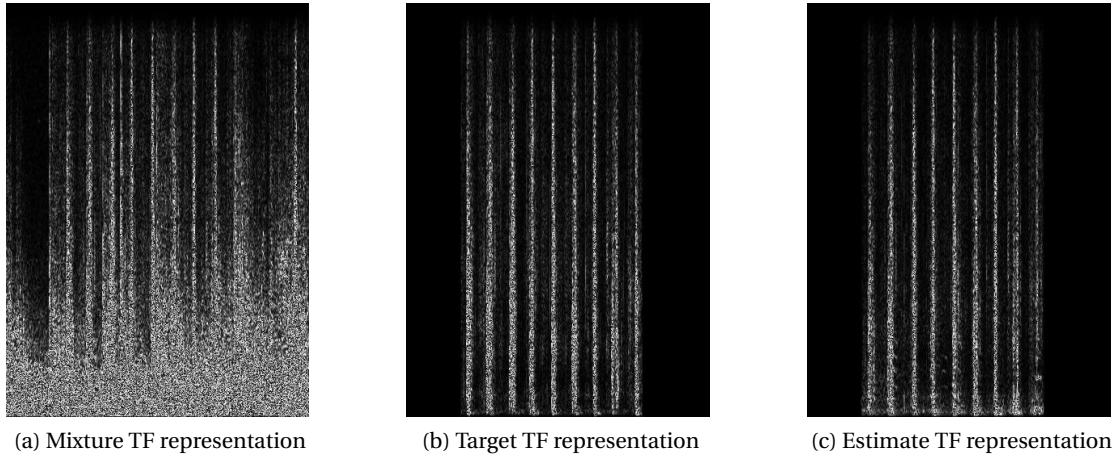


Figure 5.5: Example result obtained on a background-foreground mixture

It was mixed at an SNR level of -8.64 dB (meaning that the footsteps energy is lower than the background noise energy) with a 3.5s chunk on top of war background noise (seed 3 - example 24). The first picture is the spectrogram of the input mixture, the second is the desired output (target), and the last one is the resulting output. The time-axis is represented horizontally, from left to right, and the frequency-axis is represented vertically, bottom to top (0 - 8kHz).

in a chunk mode. Therefore, they also contained between 0 and 4s of only-background audio signal which means that improving the performance of the model on such mixtures also improves the performance on only-background mixtures. Still, it could be helpful to find a way to deal with such varying validation metric when we wish to keep the best model weights for two purposes: having a good footstep extractor, and having a good background remover.

You can find example mixture-target-estimate audio tuples alongside their spectrograms and individual metrics at [this link](#). Figure 5.5 shows an example result with TF representations of a generated background-foreground mixture, the desired target, and the corresponding estimate. We let the reader explore the folder previously linked to listen to examples on only-background mixtures.

It is now time to report the results of our *Footstep Focus mode* model on actual gameplay recordings. To do so, we retrieved parts of *Call of Duty Warzone* gameplays in which the player's footsteps were more or less audible, with more or less environmental background sounds around, converted the videos to audio files, and downsampled them to 16kHz before giving them as inputs to our trained model. Note that we did not have the time to record perfect scenarios in which another player would for example run or walk at different distances from the main player, the latter not moving. Instead, the test vectors we provide here were meant to evaluate the model on the main player's footstep sounds, as a first stage of the project. It would however surely help the evaluation phase to test the model on practical scenarios, which we encourage in the future.

The results on our test vectors are also available at the [provided link](#). Figure 5.6 shows an

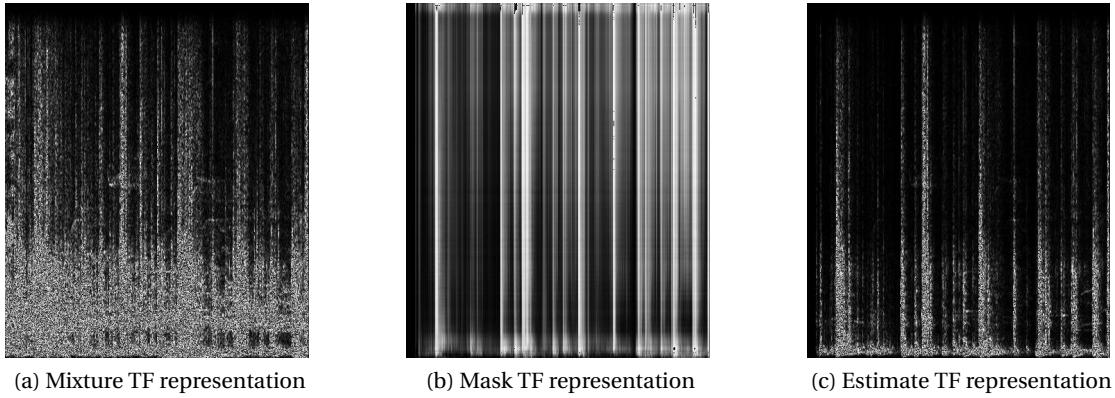


Figure 5.6: Footstep Focus Mode - Example result with a few seconds of *Call of Duty Warzone* gameplay (test vector 3)

The mixture was downsampled at 16kHz. The time-axis is represented horizontally, from left to right, and the frequency-axis is represented vertically (0 - 6 seconds), bottom to top (0 - 8kHz).

example result on a few seconds of gameplay recorded in noisy conditions (present noises are mostly speech, gunshots, game sound effects) (test vector 3).

When listening to the test vectors and their estimates, we can observe that the model works well when the footsteps are clearly audible in the mixture (e.g test vector 4). We can however still hear part of the background noise when a footstep is detected and passed through, meaning that we are doing a good job at separating in the time-domain, but that there is still room for improvement in the frequency domain. Especially since the background noise we hear at footsteps times seems to be more tonal than the target footstep themselves.

If we keep listening to more mixture-estimate pairs from actual gameplays, we notice that often, audio effects and sounds with big transients are interpreted as footsteps by the model, letting them passing through as well. This implies that the sound signature of the footstep class, even though footstep sounds can be very diverse, was not yet mastered by the model. Last but not least, we also noticed that when applying the model on gameplay recordings in which the main character was changing floor material or changing speed, the model had difficulties to adapt. We will try to provide potential solutions in the *General Discussion & Further Work* chapter.

Before closing the discussion on that part, let us remind that these test vectors are recordings of the main player's footsteps, and that in practice, those are typically very loud compared to the enemy footsteps one would like to focus on. In fact, when the main player's footsteps are very low in the mixture (for example around 65 seconds in test vector 6, when the player was walking instead of running), the model did not detect anything. This is a major downside and it is very important to keep looking in that direction to eventually achieve a real *focus mode*. We will elaborate about that in the *General Discussion & Further Work* chapter.

To conclude on that experiment, the results we obtained are very promising. As we will describe in the *General Discussion & Further Work* chapter of this thesis, we expect the results to be even better when training a more optimised model with even more complex and various mixtures with additional data. Moreover, if we take advantage of recurring sounds in video-games, we could try to teach the model to remove exactly those sounds, consequently improving our footstep-focus approach in such known audio environments.

5.2.2 Footstep Discovery Mode

For this second task, we asked VoiceFilter Causal to switch between extracting footstep-only signals, and passing the background sounds through when no footsteps were detected. Here is a more concrete example:

You are playing Call of Duty Warzone and you are waiting in a house for your friends to come and join you from the other side of the map. You are not sure to be alone in that area so you activate the footstep discovery mode. It allows you to keep playing as usual unless (!) it detects footsteps around you, in which case it automatically silences every non-footstep surrounding sounds. You can now focus on footsteps and use this information at your advantage. Thank you discovery mode!

More formally, we asked the model to do as follows:

$$\text{output} = \begin{cases} \text{footsteps}, & \text{if footsteps detected} \\ \text{input}, & \text{otherwise} \end{cases}$$

If we analyse this requirement in the machine learning sense, this time we require the mask to be filled with ones when no footsteps are detected, so that the input is kept unchanged when multiplying the mask and the input spectrograms together, compared to the *Footstep Focus Mode* in which we desired the mask to be filled with zeros in this case.

The mixture-target pairs were designed as shown in figure 5.7:

Since this task was added at the end of the project, we did not have the time to search for the best working parameters. We will therefore report the results obtained using the same parameters as the *Footstep Focus Mode* previously reported, except with a few changes:

- The mixture-target pairs are no longer built to teach the model to remove everything that is not in the footstep class, but rather to keep that information in the estimated

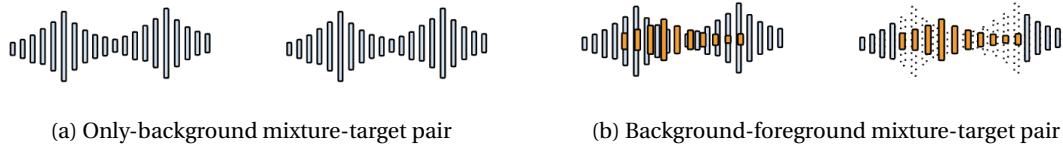


Figure 5.7: Footstep Discovery Mode - mixture-target pairs
The dotted lines represent what is not desired in the resulting output.

Metric	Baseline	Evaluation	Improvement
SNR (dB)	2.78	10.74	7.96
SI-SDR (dB)	2.78	10.28	7.50
spec_SNR (dB)	2.74	10.85	8.11
spec_MAE	0.086	0.040	0.046

Table 5.8: Footstep Discovery Mode - background-foreground test mixtures results

output when the input does not contain any footstep sounds for a certain time period.

- We set the only-background mixture percentage to be 50% of the mixtures fed to the network, instead of 30% in the previous focus task. This is because in this section we are evenly interested in keeping the background, and in retrieving the footsteps when there are some. This way, the model encounters as much background-foreground mixtures as only-background mixtures and is evenly-trained for both cases so that it hopefully finds a good balance. This is however questionable and deserves more investigation.

The results obtained are reported in tables 5.8 and 5.9, respectively for background-foreground test mixtures and only-background test mixtures. For a detailed explanation about the result tables, please refer to the previous section, *Footstep Focus Mode*.

First, we can notice in the only-background case that this time, since the target is not a zero-signal, the SI-SDR values are more meaningful. The spec_MAE values are however fixed at 0.00 because here, the only-background mixtures' targets are... themselves. Thus, having a 0.00 difference between the baseline and the evaluation is actually great news.

Now that we have commented on the questionable parts of the tables, let us analyse the

Metric	Baseline	Evaluation	Improvement
SNR (dB)	30.0	26.33	-3.67
SI-SDR (dB)	30.0	26.34	-3.66
spec_SNR (dB)	30.0	28.19	-1.81
spec_MAE	0.00	0.00	0.00

Table 5.9: Footstep Discovery Mode - only-background test mixtures results

Metric	Baseline	Evaluation	Improvement
SNR (dB)	-0.10	8.27	8.37
SI-SDR (dB)	-0.11	7.39	7.50
spec_SNR (dB)	-0.15	8.30	8.45
spec_MAE	0.13	0.06	0.07

Table 5.10: Footstep Discovery Mode - chunk level results in background-foreground test mixtures

results. On background-foreground mixtures, we can observe a **SI-SDR improvement of 7.5 dB**. If we take a quick look at the same metrics computed this time at the chunk level in table 5.10, we see that the SI-SDR values on both the complete mixtures and at the chunk level are the same. If we further compare those with the results obtained at the chunk level in the *Footstep Focus Mode*, we also notice similar results. This is very encouraging as this implies that both tasks perform the same, even with a different requirement about the background signal.

On the only-background test mixture side, table 5.9 show a SI-SDR degradation of just 3.66 dB, which is very acceptable since that difference was computed between a baseline SI-SDR of 30 dB and an evaluation SI-SDR of 26.34 dB, levels at which humans do not hear the difference.

Figure 5.8 shows an example result with TF representations of a generated mixture, the desired target, and corresponding estimate. Again, audio examples including this one (seed 3 - example 24) are available at this [link](#).

Regarding the results on actual gameplay recordings, this time we ask the reader to directly listen to the provided test vectors with the above link. Since this model is asked to switch between background noise and footsteps-only, the result will be better assessable this way.

To conclude on that second gaming-related experiment, we obtained similar results to the Footstep Focus Mode task. Hence, it might be better to focus on the latter and to use real-time DSP techniques to re-create the discovery mode. Indeed, both tasks are very related and one could think of a real-time system that would switch between the game's raw input and the focus mode output when the latter exceeds a predefined threshold for example.

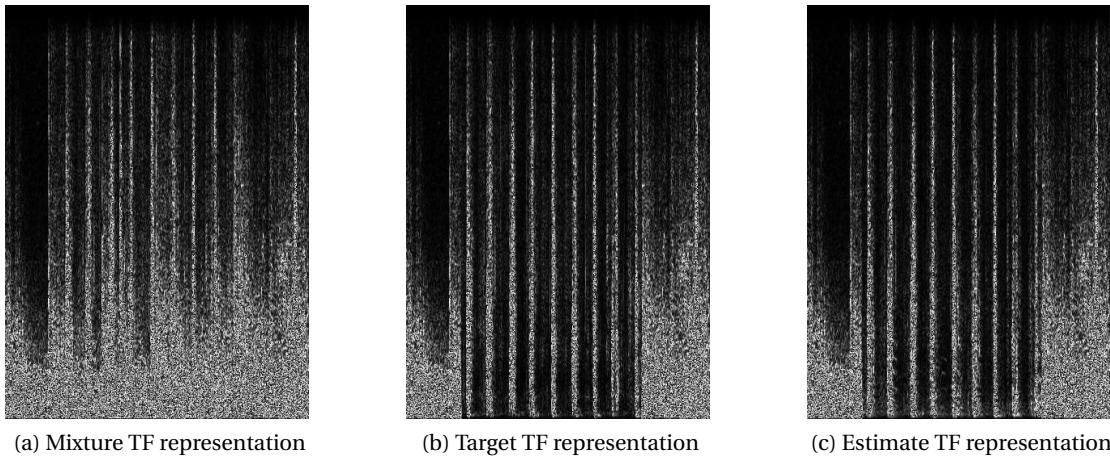


Figure 5.8: Footstep Discovery Mode - example result on a background-foreground mixture. It was mixed at an SNR level of -8.64 dB (meaning that the footsteps energy is lower than the background noise energy) with a 3.5s chunk on top of war background noise (seed 3 - example 24). The first picture is the spectrogram of the input mixture, the second is the desired output (target), and the last one is the resulting output. The time-axis is represented horizontally, from left to right (0 - 6 seconds), and the frequency-axis is represented vertically, bottom to top (0 - 8kHz).

6 Towards Real-Time Audio Scene Customisation

Once we finished our causal & real-time feasibility study experiments on the footstep extraction application for gaming purposes, we started testing the same model on a larger range of classes, using the same dataset as in the non-causal phase.

In this section, we will first present the data that we used, and finally the experiment and its results on 41 classes of audio events.

6.1 Data

6.1.1 FSD Kaggle 2018

The audio event dataset that we used here is the dataset put together for the 2018 Kaggle Freesound General-Purpose Audio Tagging Challenge [6] by the Freesound [5] and AudioSet [7] teams. Please refer to section 3.1.3.1 of this thesis for more information.

6.1.2 2-foreground-ish mixture generation

As mentioned above, we focused on the Gaming Application (section 5) during most of the project and realised this experiment at the very end. Hence, due to the fact that our mixture generation setup was not yet ready for multiple-foreground mixtures but only for background-foreground mixtures or only-background mixtures (either 0 or 1 audio event on top of background sound), we decided to use the FSD Kaggle 2018 dataset as our background and as our foreground dataset at the same time to create 2-foreground-ish-mixtures.

This trick has however some disadvantages:

- If during the random mixture generation process it occurred that two audio events from the same class appeared in the same mixture, the target audio signal would only contain the "foreground" one, so a wrong target. Nevertheless, since we deal with 41 classes

Background audio events dataset	FSD Kaggle 2018
Foreground audio events dataset	FSD Kaggle 2018
Chunk length range	[2.0, 4.0]s
SNR level range	[-5, 5] dB
Overall gain range	[0.5, 1.0]
Only-background mixtures	0%
Second input	Class index to Embedding layer

Table 6.1: Towards Real-Time Audio Scene Customisation - mixture generation parameters used

here, the probability that this happens was considered negligible.

- As opposed to other works that used the FSD Kaggle 2018 dataset, this trick deprived us of background sounds in the mixtures and prevented us from comparing the results with existing studies [2][17]. To alleviate that issue, we generated the mixtures by 1/ Trimming in/out the quiet parts of the audio events in the dataset 2/ Repeating the background audio event to fill the 4-second mixtures during the Dynamic Mixing process.

For these reasons, we fed the network with 2-foreground-ish-mixtures only.

6.2 Experiment & Results

We report here the results of the extraction task obtained with the same model as described in chapter 4, i.e VoiceFilter Causal, with the same model and training parameters as in the gaming related experiments (chapter 5). Since this is an attempt at dealing with multiple classes at the same time, we chose the second input of the model to be the class index, in turn given to a jointly trained embedding matrix, as depicted in section 4.1.2.1, each embedding of that matrix being a 256-long weight vector as in [17]. Note that due to the added weights from the embedding matrix and the new LSTM units this requires at the first LSTM layer, this model was 7.9M-parameter-big.

On the data side, this experiment was also conducted with single-channel mixtures sampled at 16kHz and the mixtures were chosen to be 4-second long this time, to force more overlaps between 2-to-4-second audio event chunks mixed with an SNR between -5 and 5 dB. The targets contained the foreground audio event, and the class index given as a second input was chosen in accordance and fed to the network for all the frames of the corresponding mixture. In order to cope with mixture-level gain variations, we also added an overall gain range of [0.5, 1.0] to choose randomly from for each mixture. Table 6.1 summarises the previous information.

Table 6.3 depicts the results obtained after training this multi-class model, and after evaluating it on mixtures generated from audio event test sets. We can observe a improvement

Metric	Baseline	Evaluation	Improvement
SNR (dB)	-4.96	6.96	11.92
SI-SDR (dB)	-4.96	4.86	9.81
spec_SNR (dB)	-4.91	6.96	11.87
spec_MAE	0.26	0.071	-0.19

Table 6.2: Towards Real-Time Audio Scene Customisation - 2-foreground-ish test mixtures results

Metric	Baseline	Evaluation	Improvement
SNR (dB)	0.16	7.13	6.97
SI-SDR (dB)	0.16	5.44	5.28
spec_SNR (dB)	0.033	7.09	7.06
spec_MAE	0.25	0.13	-0.12

Table 6.3: Towards Real-Time Audio Scene Customisation - chunk level results on 2-foreground-ish test mixtures

of **9.81 dB** on the aforementioned 4-second mixtures, but we notice in table 6.3 that the improvement is almost halved when focusing on the 2-to-4-second chunks, for which we obtain a SI-SDRi of **5.28 dB**. After further investigation we noticed that the chunks did not have an average duration of 3 seconds as expected but of 2 seconds. We ran some more tests and attempt the hypothesis that this SI-SDRi drop is due to the short duration of the chunks composing the mixtures and because the model is better at performing the separation in the time-domain than in the frequency-domain, which was confirmed by listening to example test mixtures. The reason for the short chunks still needs to be investigated as figure 3.3 showed no such short durations. One explanation could be the trimming process mentioned above, which might have shorten the audio events below our desired chunk length, in average.

When listening to more example test mixtures (available at this link), we can further notice that transient-based classes such as *snap*, *hat*, etc. are better extracted compared to more tonal audio event classes, for instance *flute*, that can be well separated from other tonal classes in the time-domain, but not as well in the frequency-domain, meaning that we can still hear the other audio event slightly. Based on that, we recommend training the model for this task in a better suited setup, further investigating on the performance per pairs of classes, and keep testing the limits of the model by training it again on a smaller number of classes as well as on a bigger dataset, such as FSD50k [5].

As we said, this setup and results must be interpreted as a proof of concept of the feasibility study yet to be conducted. Still, the results are encouraging.

7 General Discussion & Further Work

VoiceFilter Causal has shown very motivating results over the last experiments. First, in section 5.2.1 we obtained a **SI-SDR improvement of 9.82 dB on 6-second footstep mixtures composed of 2-to-4-second chunks of footstep sounds**. This value was confirmed by a chunk level evaluation and by the section 5.2.2. Then, we applied the model to a multi-class extraction task thanks to a jointly trained embedding layer, whose output was concatenated to the CNN layer output. In this experiment, we obtained a **preliminary average SI-SDR improvement of 9.81 dB on 4-second mixtures composed of 2-to-4-second audio event chunks from 41 classes**.

These are encouraging results but not yet good enough for a user-experience-level product for instance. Indeed, when applying the Footstep Focus Mode on actual gameplay recordings, all the audible footsteps are not always extracted, especially if the environment is very noisy or if the footsteps are particularly quiet. The model also has difficulties to deal with changes of the material stepped onto. Finally, many short-transient sounds from the game are also extracted, confirming that the model did not yet master the general sound signature of footsteps, if there is any.

About the experiment conducted on 41 classes of audio events, apart from the sub-optimal training setup that will have to be improved, we noticed a performance discrepancy between different types of classes.

In this chapter, we will discuss and suggest potential solutions and further works to the aforementioned issues.

7.1 Data

7.1.1 Mixture generation

On the data side, we *need* to generate more complex mixtures. First we need to train our model on multi-audio-event-mixtures, i.e mixtures containing multiple audio events. This

is motivated by the results obtained on actual recordings of gameplays while applying the *Footstep Focus Mode*. Indeed in the results, we could still hear distant gunshots, video-game sound effects, and other noises. Additionally, we need to add further data augmentation in the mixture generation process: gain modulation, pitch shifting, equalisation, reverberation... This will help the current model or any future model to be more robust and to better generalise.

More specifically to the footstep application, we would above all encourage generating multi-audio-event-mixtures with appropriate classes such as *gunshots*, *speech*, *breath*, and generate mixtures with footsteps stepping on different materials in the same input. Moreover, we believe that gain-modulation overtime will help the model cope with changes in footstep loudness. Note that adding more foreground events does not necessarily imply to train the model to handle these new classes. Even though this is also a desirable goal, one could generate more complex mixtures with audio events from lots of classes, but still train their model for one audio event such as footsteps.

When aiming to manipulate broader ranges of classes as in chapter 6, I would advise to use cleaner datasets and to group the types of audio events together in the future (e.g short-transient sounds together, tonal sounds together, ...) to better evaluate the performance and to go forward safely, with a user-experience level model in mind.

In parallel to fully taking advantage of the data we have, gathering more foreground and background data is of course necessary before having a product-level system.

7.1.2 Taking advantage of known audio environments

What we call known audio environments here are environments for which we can make stronger assumptions compared to unknown ones, the latter for which we are forced to design much more robust algorithms.

7.1.2.1 Overfitting to recurring sounds

Video-games are usually developed with a finite bank of sounds. The audio engines consequently re-use these sounds and add environment-related effects on top (spatialisation, gain, reverb, echo, ...) depending on the players' actions to improve their immersion experience.

My main suggestion for video-game footstep extraction or any audio event classes manipulation in known audio environments is therefore to take advantage of such a bank of sounds. In other words, if we were able to record those recurring audio events in non-noisy conditions or close to non-noisy conditions, we could train a machine learning model to specialise to these particular sounds (e.g annoying game notifications, recurring speech signals, bips, or even footsteps), in order to manipulate them with more ease.

This idea has shown high efficacy in early-experiments with small-size machine learning

models. Indeed, we tried overfitting a smaller version of VoiceFilter Causal to a recurring sound from *Call of Duty Warzone* (a buzz sound) that we could record in close-to-non-noisy conditions, and as expected, the results were pretty impressive. Audio samples are available at this link.

Together with a method similar to [2], we could aim for a product allowing users to train their personal-and-sound-specific machine learning models. We could also raise our ambition and train our model to handle many classes and specific sounds at the same time, and thus allowing users to even further personalise their in-game audio scene.

7.1.2.2 Generating footstep series ourselves

One other idea we had was to try to feed the network with footstep series randomly generated out of single footsteps (see section 5.1.1.2). Indeed and as mentioned earlier, this is typically how game-developers code video-games' sound, i.e by (almost-)periodically juxtaposing single footsteps audio events according to the material, the movement speed, the character, ... This would also allow more flexibility on the footstep series speed, and to change materials more easily. Combined with the datasets in our posession and data augmentation techniques applied on top would surely help the model perform even better on such games.

7.2 Model

As we mentioned, VoiceFilter Causal is very versatile as it can switch from speech separation to audio event manipulation. This is great news, but we will go through a few ideas worth exploring to make the most of it.

7.2.1 Integrating the shot learning methodology

The overall idea of the shot learning methodology is to give as a second input an audio example of the same class as the target audio event to extract, instead of giving a one-hot vector for instance.

In [8], the authors trained a U-Net-based model [21] for single-class audio event extraction and obtained an average SI-SDR improvement of 9.6 dB on the FSD50k dataset [5] and 14 dB improvement on Librispeech [18]. More importantly, they obtained an average SI-SDRi of 9.4 dB when training on one part of FSD50k and evaluating on the other part (unseen data), proving that there model generalised to unseen audio event classes.

A follow-up idea was described in [2]. In this study, the authors trained a Conv-TasNet-based model [13] to perform audio event extraction by alternating between a jointly-trained embedding and a jointly-trained shot embedding as a second input. This switch during the training phase in the end allowed them to add a new audio event class to the list of supported

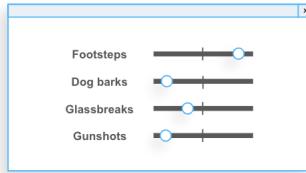


Figure 7.1: Example of a user interface for audio event manipulation with sliders

classes by only providing a few example audios and retraining their model for a few epochs.

An additional future work would therefore be to integrate shot-learning with the previously described technique in the current pipeline to allow even more control on users' audio environment. Note however that the method described in [2] might only work when the corresponding model is already trained with a certain number of different classes (in the number of 41 in the cited paper), and might not work when training a model for a few classes only. Indeed, such a model needs to generalise to enough types of sounds in order for a new class to only need a few epochs of retraining to be added to the list.

7.2.2 The slider-based model idea

If you remember well, the original goal was to allow users to control multiple classes of audio events at the same time, in real-time. This can be illustrated with the mock user-interface showed in figure 7.1. During this project and throughout this thesis, we mainly focused on the manipulation of single audio event classes at the same time. However, in order to fulfill the original goal, we still need to find a way to manipulate multiple classes at the same time and with different levels each.

The challenge is the following: when we are focusing on only one particular class, e.g footsteps, and assuming the extraction or the removal model for that class works sufficiently well, it is rather simple to mix the original input with the model output to obtain a DSP based slider thanks to the ambivalence described in A.1.

However, when we desire to deal with multiple audio events at the same time, it is not anymore that simple since at best, with the current setup, we would reach what has been done in [17] and consequently only be able to extract/remove multiple classes together, as a group. We therefore considered three options:

- We could train a model for the separation of multiple audio events [10], but that raises the constraints evoked in chapter 3 and it might become computationally expensive and affect the real-time capacity of our current model.
- We could also chain several models, each one being specialised in one type of audio events. However, not only this might also raise computational capacity and latency

issues, but the possibly corrupted output of one model might affect the output of the next one.

- The best option we thought of is therefore to use the second input of our model as the slider input, i.e training a slider-based model. The idea is then to be able to directly give the slider value to the model for each of the classes we would like to manipulate, which would hopefully produce the final estimate. There is no known reference to base ourselves on, but we leave this idea here for future work.

7.2.3 Other ideas worth exploring

I suggest, as in Damien Ronssin's thesis [22], to investigate the learned encoder-decoder with offline training. Indeed, when we jointly trained an encoder-decoder, it ended up with output mixtures containing artifacts, but training such an encoder-decoder pair in an offline manner could yield better results since the training would not be "disturbed" by the audio event manipulation task.

Due to time limitation, we couldn't explore the complex domain, but learning time-frequency *phase*-masking might allow more context understanding and better separation overall [35]. I would therefore suggest exploring this as well.

My last model-related advice is to further optimise VoiceFilter Causal with a hyperparameter search. We did not have the time to really enter that phase except for a subtle hyperparameter tuning process which has proven to be very effective (section A.2), so there is indeed room for improvement.

7.3 Summary

To simplify the reading, here is a summary of the suggested improvements to be made in the future:

- Data
 - Generating multi-audio-event-mixtures
 - Adding more data augmentation techniques
 - Getting cleaner datasets and using less classes first to better understand the performances of the model in multi-class conditions
- Model
 - Integrating the few-shot learning method to our multi-class model
 - Training a slider-based model for multi-class manipulation

- Investigating the performance of an encoder-decoder trained offline, the complex domain, and optimising VoiceFilter Causal

8 Conclusion

During this Master Thesis and this internship, we focused on audio event manipulation with the idea in mind that maybe we could manipulate several classes of audio events in real-time with current machine learning techniques.

To achieve that, we first started with an assessment of the idea by reproducing an offline model [17] and by using a dataset composed of 41 audio event classes [6]. We obtained promising results with mixtures of two audio events, but worse results than expected with mixtures of three audio events. Nevertheless, we concluded that it was worth continuing on that path, and switched to a causal model with a more concrete goal - footstep extraction in the gaming context, the goal being to conduct a feasibility study for footstep extraction in First Person Shooting war games such as *Call of Duty Warzone*.

We used VoiceFilter Causal [22], a causal and proven-to-be-(almost-)real-time model originally designed for speech separation [33] and Automatic Speech Recognition [32] to perform footstep extraction in noisy conditions. We obtained an overall SI-SDR improvement of 9.82 dB on 6-second artificial mixtures containing 4 seconds of footsteps in average, mixed at an SNR level of -10 to 10 dB with respect to various background noises. This is a first step, but results on actual recordings of gameplays are not perfect yet. Indeed, all the perceptible footstep sounds are not always extracted when they are too quiet, in too noisy conditions, or when there is a change in the material stepped onto. Moreover, other audio events still remain in the resulting output.

We also started testing that same model on the FreeSound Kaggle 2018 dataset [4][6], and obtained a SI-SDR improvement of 9.81 dB on 4-second mixtures of 2 overlapping audio events. This is also very promising, but we believe that with cleaner data, more complex mixtures, and maybe focusing on less classes first, we could achieve even better results with time.

As next steps, we therefore recommend to generate more complex mixtures with more data augmentation techniques, with additional environmental audio data, to tune the model

further, and to take advantage of the information at our disposal in the audio environment of interest, if any. Combined with the ideas depicted in the *General Discussion & Further Work* section, we believe that this project could someday result in a user-experience-level audio-event manipulator product at Logitech.

Thank you again to Milos Cernak and Paolo Prandoni for trusting and supervising me, and to Logitech for allowing such an incredible experience. The project was challenging but very interesting, and I am even more thirsty for knowledge now. I hope that the overall idea will live on at Logitech and that we will cross paths again someday!

To the readers who have made it this far, thank you for your interest and do not hesitate to contact me at rayan@daodnathoo.com if you have any question.

In the mean time, have a great day!

“After 6 months spent working on audio at Logitech, all I see is spectrograms”
- Me, during some dinner with other interns

A Appendices

A.1 The removal task

Throughout the entire thesis, we have focused on the extraction task, but during this project we also conducted experiments for the removal task, i.e removing the audio event class of interest from an input mixture instead of extracting it. Let us explain why we focused in the end on only one of the two tasks.

In the context of a single audio event class manipulation, there is an ambivalence between noise and target. In other terms, when extracting the class of interest from an audio mixture, one simply has to subtract the output from the mixture to obtain the desired output of the removal task. This works the other way around as well. More formally:

$$target_{rem} = mixture - target_{ext}$$

$$target_{ext} = mixture - target_{rem}$$

One might therefore ask oneself: "Is the removal trained model really better for the removal task? And with the above "trick", how does it compare to an extraction trained model for the extraction task?"

We compared both models with the same training and data generation parameters, with several foreground datasets, and several background datasets. We then computed the result of each of the removal and extraction models for its opposite task thanks to the equations above.

The overall outcomes were mostly similar for both the removal and extraction task in every scenarios we tested. This is the reason why we stuck with one task our of the two, which was chosen to be the extraction.

Again, this only works when only one audio event class is targeted. On the other hand, when

the goal is to manipulate several audio event classes at the same time with different parameters each, the aforementioned ambivalence is no longer relevant and one needs to find another way. One idea was described in the section 7.2.2.

A.2 Hyperparameter tuning

Since the original VoiceFilter Causal parameters were meant for speech separation and since we focused on audio events, more specifically footsteps, we decided to look for optimal parameters of the VoiceFilter Causal for our task.

Keras Tuner [[16]] is a Python library built to easily perform hyperparameter tuning with Tensorflow 2.0. Here we used the Hyperband class, an optimized version of random search using early-stopping to speed up tuning process. It proceeds by rounds, by training the same model with different sets of hyperparameters, each time for a small number of epochs, and by keeping only the ones obtaining the best validation results for the next round.

Table A.1 lists the parameter and the ranges we gave to *Keras Tuner Hyperband*, in which:

- `n_conv_filters_1` is the number of convolution filters in the first convolution.
- `n_conv_filters_2` is the number of convolution filters in the second set of convolutions.
- `n_conv` is the number of convolutions in the second set of convolutions.
- `lstm_reg` is the LSTM regularization factor.
- `n_lstm` is the number of LSTM layers.
- `lstm_size` is the size of each LSTM layer, i.e the number of LSTM units.
- `n_fc` is the number of fully connected layers.
- `fc_size` is the number of units in each fully connected layer.
- `lr` is the learning rate.

The objective metric was set to be the Mean Absolute Error, the maximum number of epochs for each training to be 15, and an early stopping with patience 5 was set with respect to the validation loss.

Parameter	Search range
n_conv_filters_1	[8, 16 , 24, 32]
n_conv_filters_2	[8, 16, 24 , 32]
n_conv	[1, 2, 3]
lstm_reg	[0.0001, 0.0005, 0.0001]
n_lstm	[2, 3 , 4]
lstm_size	[64, 128, 192, 256, 320 , 384, 448, 512]
n_fc	[1, 2, 3]
fc_size	[64, 128, 192, 256 , 320, 384, 448, 512]
lr	[0.0001, 0.0005 , 0.0001]

Table A.1: Hyperparameter tuning - Parameter ranges given to Keras Tuner Hyperband
Old parameters are written in red, and best parameters found are written in blue. When there
is only a blue parameter, this means that the best value is the same as the old one.

After 20 hours of parameter search, the resulting parameters were the ones written in **bold** in the table A.1. We then ran *VoiceFilter Causal* for the same task (footstep removal here) using both sets of hyperparameters to compare the results and validate the new set.

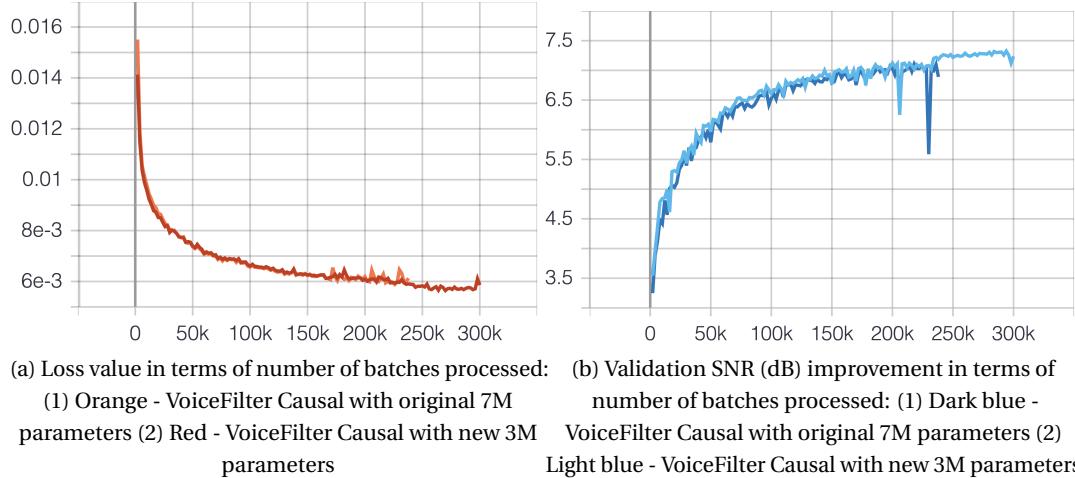


Figure A.1: Hyperparameter tuning - Comparison of different sets of parameters on VoiceFilter Causal

Figure A.1 showcases the loss value (left plot) and the validation SNR (right plot) for both hyperparameter sets on test mixtures composed of background and foreground audio. We can observe that both sets of parameters lead to the same learning curve shapes and yield similar results once reaching the predefined maximum number of batches in terms of Signal-to-Noise-Ratio (SNR). Further comparison were conducted by listening to the output audio and no major difference was detected, validating the hypothesis. Even if these new parameters

did not result in a significantly better accuracy, the number of trainable parameters dropped from 7.5M to 3M parameters, naturally increasing the training speed from 4.8 minutes to 3.5 minutes processing per batch, and certainly decreasing the inference latency. This set of parameters was however not used in the reported experiments, as the data and the task evolved since that hyperparameter search but these results should motivate whoever continues on this path to try tuning the model in the future as it seemed to be very effective.

Bibliography

- [1] MATTHEW DANIELSON. *Categorical Encoding Comparison*. 2019. URL: <https://www.kaggle.com/mdanielson/categorical-encoding-comparison>.
- [2] Marc Delcroix et al. *Few-shot learning of new sound classes for target sound extraction*. 2021. arXiv: 2106.07144 [eess.AS].
- [3] Eduardo Fonseca et al. “Freesound Datasets: A Platform for the Creation of Open Audio Datasets”. In: *ISMIR*. 2017.
- [4] Eduardo Fonseca et al. “Freesound Datasets: A Platform for the Creation of Open Audio Datasets.” In: *Proceedings of the 18th International Society for Music Information Retrieval Conference* (Suzhou, China). Suzhou, China: ISMIR, Oct. 2017, pp. 486–493. DOI: 10.5281/zenodo.1417159. URL: <https://doi.org/10.5281/zenodo.1417159>.
- [5] Eduardo Fonseca et al. *FSD50K: an Open Dataset of Human-Labeled Sound Events*. 2020. arXiv: 2010.00475 [cs.SD].
- [6] Eduardo Fonseca et al. *General-purpose Tagging of Freesound Audio with AudioSet Labels: Task Description, Dataset, and Baseline*. 2018. arXiv: 1807.09902 [cs.SD].
- [7] Jort F. Gemmeke et al. “Audio Set: An ontology and human-labeled dataset for audio events”. In: *Proc. IEEE ICASSP 2017*. New Orleans, LA, 2017.
- [8] Beat Gfeller, Dominik Roblek, and Marco Tagliasacchi. *One-shot conditional audio filtering of arbitrary sounds*. 2020. arXiv: 2011.02421 [eess.AS].
- [9] Shawn Hershey et al. *The Benefit Of Temporally-Strong Labels In Audio Event Classification*. 2021. arXiv: 2105.07031 [cs.SD].
- [10] Ilya Kavalerov et al. *Universal Sound Separation*. 2019. arXiv: 1905.03330 [cs.SD].
- [11] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].
- [12] Keisuke Kinoshita et al. “The reverb challenge: A common evaluation framework for dereverberation and recognition of reverberant speech”. In: *2013 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*. 2013, pp. 1–4. DOI: 10.1109/WASPAA.2013.6701894.

- [13] Yi Luo and Nima Mesgarani. “Conv-TasNet: Surpassing Ideal Time–Frequency Magnitude Masking for Speech Separation”. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 27.8 (Aug. 2019), pp. 1256–1266. ISSN: 2329-9304. DOI: 10.1109/taslp.2019.2915167. URL: <http://dx.doi.org/10.1109/TASLP.2019.2915167>.
- [14] Yi Luo and Nima Mesgarani. *TasNet: time-domain audio separation network for real-time, single-channel speech separation*. 2018. arXiv: 1711.00541 [cs . SD].
- [15] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: <https://www.tensorflow.org/>.
- [16] Tom O’Malley et al. *Keras Tuner*. <https://github.com/keras-team/keras-tuner>. 2019.
- [17] Tsubasa Ochiai et al. *Listen to What You Want: Neural Network-based Universal Sound Selector*. 2020. arXiv: 2006.05712 [eess . AS].
- [18] Vassil Panayotov et al. “Librispeech: An ASR corpus based on public domain audio books”. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2015, pp. 5206–5210. DOI: 10.1109/ICASSP.2015.7178964.
- [19] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems* 32. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
- [20] Kedar Potdar, Taher Pardawala, and Chinmay Pai. “A Comparative Study of Categorical Variable Encoding Techniques for Neural Network Classifiers”. In: *International Journal of Computer Applications* 175 (Oct. 2017), pp. 7–9. DOI: 10.5120/ijca2017915495.
- [21] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs . CV].
- [22] Damien Ronnissin. *Real-Time Speech Separation for Gaming Audio Chat*. 2021.
- [23] Jonathan Le Roux et al. *SDR - half-baked or well done?* 2018. arXiv: 1811.02508 [cs . SD].
- [24] Justin Salamon et al. “Scaper: A library for soundscape synthesis and augmentation”. In: *2017 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. 2017, pp. 344–348. DOI: 10.1109/WASPAA.2017.8170052.
- [25] David Snyder, Guoguo Chen, and Daniel Povey. *MUSAN: A Music, Speech, and Noise Corpus*. 2015. arXiv: 1510.08484 [cs . SD].
- [26] David Snyder et al. “X-Vectors: Robust DNN Embeddings for Speaker Recognition”. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, pp. 5329–5333. DOI: 10.1109/ICASSP.2018.8461375.
- [27] Christian J. Steinmetz and Joshua D. Reiss. “pyloudnorm: A simple yet flexible loudness meter in Python”. In: *150th AES Convention*. 2021.
- [28] Cem Subakan et al. “Attention is all you need in speech separation”. In: *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2021, pp. 21–25.

- [29] Efthymios Tzinis et al. “Improving Universal Sound Separation Using Sound Classification”. In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (May 2020). DOI: 10.1109/icassp40776.2020.9053921. URL: <http://dx.doi.org/10.1109/ICASSP40776.2020.9053921>.
- [30] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.
- [31] Emmanuel Vincent, Rémi Gribonval, and Cédric Févotte. “Performance measurement in blind audio source separation”. In: *IEEE Transactions on Audio, Speech and Language Processing* 14.4 (2006), pp. 1462–1469. URL: <https://hal.inria.fr/inria-00544230>.
- [32] Quan Wang et al. *VoiceFilter-Lite: Streaming Targeted Voice Separation for On-Device Speech Recognition*. 2020. arXiv: 2009.04323 [eess.AS].
- [33] Quan Wang et al. *VoiceFilter: Targeted Voice Separation by Speaker-Conditioned Spectrogram Masking*. 2019. arXiv: 1810.04826 [eess.AS].
- [34] Neil Zeghidour and David Grangier. “Wavesplit: End-to-end speech separation by speaker clustering”. In: *arXiv preprint arXiv:2002.08933* (2020).
- [35] Lu Zhang et al. *Multi-Task Audio Source Separation*. 2021. arXiv: 2107.06467 [eess.AS].